CONTROLLING WORLD MODEL GENERATION FOR IMAGINED CURRICULUM LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent work has begun to explore performing Unsupervised Environment Design (UED) within learned world models, aiming to enable adaptive training curricula without requiring continued access to the true environment. However, existing approaches provide limited control over the trajectories generated by the world model. In this work, we propose an improved method of controllable world-model-based UED by introducing **Dynamically-controlled Model Generation (DyMGen)**, a world-model architecture that learns latent generation actions through which an adversarial teacher can control world generation dynamically as a student explores the environment. This enables online adversarial environment generation without requiring a handcrafted simulator that explicitly supports this dynamic environment modification. We show that agents trained in adversarially controlled DyMGen worlds significantly outperform those trained with random generation, and approach the performance of UED agents trained with full access to the true environment. Our results demonstrate the potential for learning environment-design interfaces directly from offline experience, illustrating the potential for UED within learned, controllable world models.

1 INTRODUCTION

The effectiveness of Deep Reinforcement Learning (RL) has been shown in a number of diverse applications (Kober et al., 2013; Levine et al., 2016; Mirowski et al., 2017; Silver et al., 2016; 2017; Berner et al., 2019), yet trained agents often generalise poorly to environments that differ even slightly from their training distribution (Zhang et al., 2018; Cobbe et al., 2019; 2020). Efforts to alleviate this generalisation gap have examined Automated Curriculum Learning (ACL), where training curricula are generated, such that the agent is consistently trained in diverse and challenging environments, effective for learning (Florensa et al., 2018; Wang et al., 2019; Portelas et al., 2020a;b; Narvekar et al., 2020). Unsupervised Environment Design (UED) has specifically focused on generating these curricula without prior domain knowledge (Dennis et al., 2020). Typically, UED methods search over parameterised environments for levels that induce high regret, providing an adaptive curriculum that evolves alongside the student (Dennis et al., 2020; Jiang et al., 2021b; Parker-Holder et al., 2022; Beukman et al., 2024; Furelos-Blanco et al., 2026). Whilst the majority of this work has focused on curriculum design using handcrafted environments and simulators, a compelling direction is to perform UED within a learned world model (Berdica et al., 2024; Güzel et al., 2025). Rather than requiring continued access to an environment simulator, a world model (Ha & Schmidhuber, 2018) can generate trajectories on which the student can train, allowing curricula to be constructed with access only to fixed datasets. With the advent of recent hyperrealistic, general world models, (Hu et al., 2023; Yang et al., 2023; Bruce et al., 2024; Hafner et al., 2025; Seo et al., 2026), the goal of *world-model-based UED* presents potential direction for training generally-capable, open-ended agents.

The core challenge presented by world model UED, is providing the UED adversary with sufficient control over the trajectories generated by the world model. Existing approaches primarily influence generation through the initial observation Güzel et al. (2025), after which the trajectory is determined by the interaction between the student and world model. In partially observable environments, this initial observation may contain little information about the resulting trajectory, severely limiting the adversary's ability to generate useful training environments. Ideally, online environment generation would allow adversarial control as the student explores the level. Recent work (Mead et al., 2025)

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

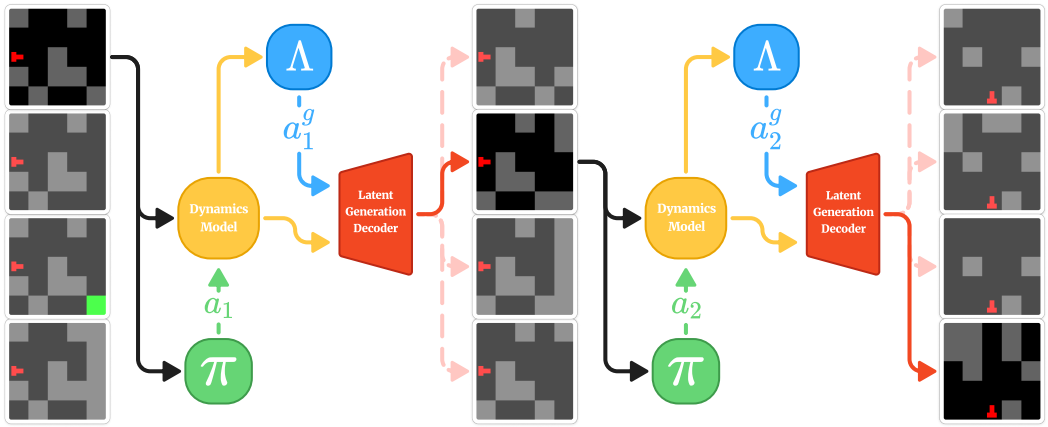


Figure 1: Controllable World Model Inference. The student policy π takes an action a, based on the previously generated observation. This action, combined with the observation, is used to update the dynamics model's history vector. This vector is passed to the adversary Λ, which selects a generation action a^g. Based on the generation action and the dynamics model, the latent generation decoder generates the next observation.

has highlighted the efficacy of online generation, but crucially requires specially designed simulators where environment parameters can be modified online while ensuring consistency with trajectory history. We bridge these approaches with *Dynamically-controlled Model Generation* (DyMGen), a world-model architecture that exploits learned latent generation actions to enable adversary control throughout a trajectory. DyMGen allows the adversary to influence the environment as the student interacts with it, while maintaining consistency with previously generated observations. Crucially, these generation actions are learned directly from offline student trajectories rather than requiring a handcrafted, dynamically controllable simulator.

In this work, we re-examine the domains explored in Mead et al. (2025), and show that the DyMGen learned generation actions provide sufficient control for an adversarial teacher to construct effective training trajectories within the world model. Students trained under adversarial DyMGen control consistently outperform the world model baselines, whilst also outperforming domain-randomisation (DR) agents trained directly in the true environment. Additionally, we provide direct comparison with agents trained using a UED adversary with online access to the true environment, illustrating the current gap between world model and true environment methods. However, these results provide an initial demonstration that adaptive UED curricula can be generated within learned world models, and motivate further work towards replacing handcrafted environment-generation interfaces with learned, controllable generative environments.

Our contributions are:

- We introduce **Dynamically-controlled Model Generation (DyMGen)**, a world-model architecture that learns latent generation actions from offline trajectories, enabling per-step control over generated environments while maintaining consistency with trajectory history.
- We show that a UED adversary can exploit these learned generation actions to generate effective world model training curricula, consistently improving student performance over world model baselines.

2 Background

2.1 Unsupervised Environment Design

In partially observable settings, a Partially Observable Markov Decision Process (POMDP) can be used to formalise environment dynamics. A POMDP is defined by the 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, defining the distribution of subsequent states s_{t+1}, given a state-action pair (s_t, a_t); $\mathcal{I} : S \to O$ is the observation function, defining a distribution of observations, given a specific state; $\mathcal{R} : S \times A \to \mathbb{R}$ is the reward function, ρ_0 is the distribution over initial states; and γ is the discount factor. Within a given domain, Unsupervised Environment Design is typically formalised as an Underspecified Partially Observable Markov Decision Process (UPOMDP). Introduced in Dennis et al. (2020), and defined by the tuple $\mathcal{M} = \langle S, A, O, \mathcal{T}^{\mathcal{M}}, \mathcal{I}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \rho_0^{\mathcal{M}}, \gamma, \Theta \rangle$, the UPOMDP extends the POMDP to represent a set of possible environment instantiations. In the UPOMDP specification, Θ represents the set of possible environment parameters θ. A single set element θ, also referred to as a level, represents a fully specified POMDP, defined by the transition, state and reward functions $\mathcal{T}^{\theta}, \mathcal{I}^{\theta}, \mathcal{R}^{\theta}$ and the initial state distribution ρ_0^{θ}.

The curriculum objective for UED is generally framed as a two-player minimax regret game between a student π and adversary Λ. For a given policy π and level θ, the expected discounted return is defined as $U(\pi, \theta) = \mathbb{E}_{\pi, \mathcal{M}_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$. Therefore, the regret can be defined as

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

where π^*_θ is the optimal policy given θ. As such, the adversary aims to find the level θ that maximises regret, given the current policy π

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

2.2 Related Work

Existing UED methods for selecting high-regret levels can generally be categorised as either relying on curation of randomly generated levels (Jiang et al., 2021b;a; Parker-Holder et al., 2022; Rutherford et al., 2024; Güzel et al., 2025), or explicitly training level generators (Dennis et al., 2020; Azad et al., 2023; Chung et al., 2024; Mead et al., 2025). Prioritised Level Replay (PLR) (Jiang et al., 2021b), a curation-based method, relies on randomly generating levels, and maintaining a replay buffer of the highest regret levels. Curriculum design involves alternating between level generation, and sampling levels for replay from the buffer. ACCEL (Parker-Holder et al., 2022) extends this by allowing for level mutation, where new levels can also be generated by perturbing the environment parameters of existing high regret levels. Most relevant to this work, Güzel et al. (2025) introduces PLR-based sampling to generate the initial state for world model trajectories.

Alternatively, generation-based methods rely on directly learning to generate high-regret levels, often relying on RL to train a teacher capable of high-regret generation. Methods such as PAIRED (Dennis et al., 2020) rely on sequentially building the level with a series of generation steps, prior to any student interacting with the environment. CLUTR (Azad et al., 2023) instead first trains a Variational Autoencoder (VAE) (Kingma & Welling, 2014) to allow level generation in a latent space. ADD (Chung et al., 2024) uses a pre-trained diffusion level generator, and a learned a differentiable regret-approximating function, to enable regret-guided level diffusion. Most similarly to our work, DEGen (Mead et al., 2025) implements online level generation, where the environment is only generated as the student agent observes it, enabling improved teacher control of level generation. However, DEGen requires handcrafted simulators, where the environment can be dynamically modified, whilst still maintaining consistency over previous observations. With DyMGen, we instead aim to learn the interface through which a UED adversary can control an environment

2.3 Regret Approximation

As outlined above, UED aims to produce a high-regret curriculum for training, but it generally will be infeasible to directly compute level regret. As such, regret approximations are relied on instead. The most commonly used approximations, often referred to as score functions, rely on analysis of the value function of the trained policies. Metrics previously shown to be effective include Temporal Difference (TD) error, Positive Value Loss (PVL) and Maximum Monte Carlo (MaxMC). Recently, Maximised Negative Advantage (MNA) (Mead et al., 2025) has shown strong empirical results compared to existing metrics.

MNA aims to approximate the expected value of the optimal policy at a given state by taking the maximum over a series of value approximations of a given state s_t, considering the value function

evaluated between the current state, and n-steps ahead. The full derivation is provided in Appendix A, but can be summarised as

$$\text{MNA} = \left(\frac{1}{T} \sum_{t=0}^{T} \hat{G}_t^{\lambda} \right) \cdot \hat{C}, \tag{3}$$

where $\hat{G}_t^{\lambda}$ is a per-time-step regret approximation, and $\hat{C}$ is 1 if the agent has solved the level at least once, and 0 otherwise. $\hat{G}_t^{\lambda}$ is a dense reward signal with respect to the agent trajectory, so when using an online-generation adversary Λ, the adversary reward at time step t can be assigned as

$$r_t^{\Lambda} = \hat{G}_t^{\lambda} \cdot \hat{C}, \tag{4}$$

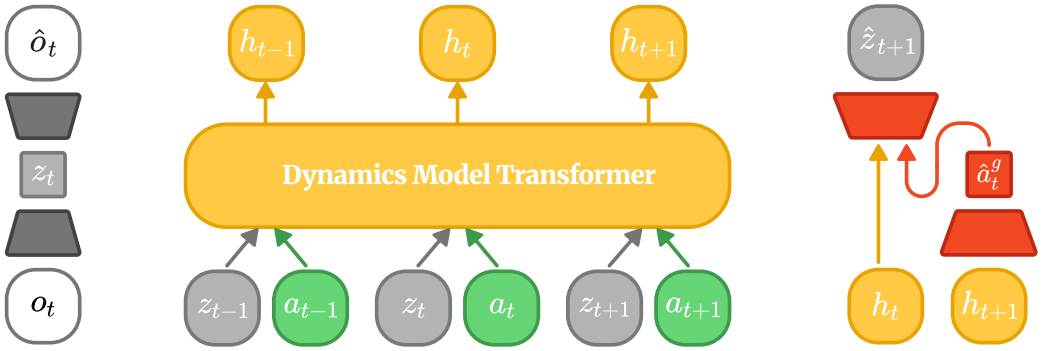
The use of this dense reward signal reduces the difficulty in adversary credit assignment.

3 METHOD

The two core components of our method are: a world model allowing for controllable level generation; and an adversary that is trained to maximise the MNA regret approximation during level generation. We first outline the architecture of the controllable world model, with further detail on the adversary outlined in Section 5.

3.1 CONTROLLABLE GENERATION

As with many recent works (Robine et al., 2023; Micheli et al., 2023; Hafner et al., 2025) focused on training effective world models, we adopt a transformer-based architecture. In order to enable controllable generation, our world model consists of a number of components, as outlined below. Additional details on the world model architecture and training can be found in Appendix B.



(a) Obs. (b) Dynamics (c) Latent Generation

Figure 2: World Model Architecture, showing the a) Observation Encoder and Decoder, b) the Dynamics Model, and the c) Latent Generation Encoder and Decoder.

Observation Model: Similar to previous work (Robine et al., 2023), we adopt a variational autoencoder (Kingma & Welling, 2014) observation model that maps environment observations to discrete latent states. The model consists of an encoder, that takes the observation as an input, and maps this to a distribution over a set of discrete latent states; and a decoder, that takes a sampled discrete latent state, and maps this back to observation space.

$$\begin{aligned} &\text{observation encoder:} && z_t \sim p_\phi(z_t|o_t) \\ &\text{observation decoder:} && \hat{o}_t \sim p_\phi(\hat{o}_t|z_t) \end{aligned} \tag{5}$$

As with DreamerV2 (Hafner et al., 2021), the latent state is a set of n discrete vectors, each with m possible categories. These discrete vectors are one-hot encoded, and then concatenated, resulting in a latent representation of length $n \times m$. In this work, we used a latent space with 32 discrete vectors, and 16 categories. The observation model was pre-trained on the training dataset, using a standard reconstruction loss (See Appendix B), before being frozen prior to training the rest of the world model.

Dynamics Model: As we are modelling a POMDP, predicting the observation o_t requires the world model to be conditioned on the full observation $o_{0:t}$ and action $a_{0:t}$ history. To aggregate this full history into a single vector, we use a transformer-based dynamics model.

$$\text{dynamics model:} \qquad h_t = f_\psi(z_{0:t}, a_{0:t}) \tag{6}$$

For this transformer, for each timestep, we sum the action a_t and latent observation z_t embeddings to produce a single observation-action token. We use a standard transformer, using RoPE (Su et al., 2024) for positional embedding, and QK normalisation (Dehghani et al., 2023) for stability.

3.2 Latent Generation Actions

In order to enable controllable generation in our world model, we need to be able to learn latent generation actions. As with Genie (Bruce et al., 2024), these actions are not present in the training dataset, but must instead be learned. However, unlike the original Genie work, which learned 8 distinct agent control actions, we have access to the agent actions. Instead want to learn to control the generation of newly observed parts of the world as it is explored by the agent. We aim to learn a latent representation that encapsulates the new information that is contained within a new observation.

Latent Generation Model: Akin to Genie, we train both a latent generation (LG) encoder and decoder.

$$\begin{aligned} \text{LG encoder :} \qquad & \hat{a}_t^g \sim p_\psi(\hat{a}_t^g | h_{t+1}) \\ \text{LG decoder :} \qquad & \hat{z}_{t+1} \sim p_\psi(\hat{z}_{t+1} | h_t, \hat{a}_t^g) \end{aligned} \tag{7}$$

The LG encoder is conditioned on the history vector h_{t+1}, meaning has access to both the history of latent states $z_{0:t}$, but also the subsequent latent state z_{t+1}, and encodes a distribution over latent generation actions. As with the observation model, the LG actions are a set of k discrete vectors, each taking a value from g distinct categories. The LG decoder is conditioned by the current history vector h_t, as well as the LG action $\hat{a}_t^g$, with the aim for the LG action to be an expressive but compressed representation of the next observation. As well as the state decoder, the latent generation model also consists of a reward and continue predictor, conditioned on $\bar{h}_t = [h_t, z_{t+1}]$.

$$\begin{aligned} \text{LG reward predictor :} \qquad & \hat{r}_t \sim p_\psi(\hat{r}_t | \bar{h}_t, \hat{a}_t^g) \\ \text{LG continue predictor :} \qquad & \hat{c}_t \sim p_\psi(\hat{c}_t | \bar{h}_t, \hat{a}_t^g) \end{aligned} \tag{8}$$

With a frozen observation model, the Dynamics Model and the Latent Generation Model are trained together with the loss:

$$\mathcal{L}_\psi^{\text{Dyn}} = \mathbb{E}\left[\sum_{t=0}^{T} \underbrace{KL\Big[\text{sg}(p_\phi(z_{t+1}|o_{t+1}))||p_\psi(\hat{z}_{t+1}|h_t, \hat{a}_t^g))\Big]}_{\text{latent generation decoder loss}} \underbrace{- \log p_\psi(r_t|\bar{h}_t, \hat{a}_t^g)}_{\text{reward predictor}} \underbrace{- \log p_\psi(c_t|\bar{h}_t, \hat{a}_t^g)}_{\text{continue predictor}}\right] \tag{9}$$

where $\hat{a}_g^t$ is sampled from the latent generation encoder.

4 Generation Consistency

At inference, this controllable world model somewhat resembles a Variational Recurrent Neural Network (V-RNN) (Chung et al., 2015), where the generation of future observations relies both on the history, and a sampled random latent variable. However, unlike a V-RNN, this random latent variable is not sampled from a prior that is conditioned on the history, but instead is controlled by an adversarial teacher agent. As such, it is necessary to ensure that the world model is robust to adversarial perturbation of these LG actions. We need to ensure that, for all LG actions, the subsequent world model output remains consistent with the full current observation history.

4.1 Generation Action Noise

If we consider the underlying POMDP we are trying to model, we aim to learn a model where conditioning the output on discrete latent generation actions shifts the distribution of the next

observation. Therefore, we can consider both the conditioned observation distribution, dependent on the latent generation actions, but also the underlying, unconditioned observation distribution, inherent to the environment.

$$\begin{aligned} \text{conditioned observation distribution :} \quad & o_{t+1} \sim p(o_{t+1}|(o_{0:t}, a_{0:t}), \hat{a}_t^g) \\ \text{unconditioned observation distribution :} \quad & o_{t+1} \sim p(o_{t+1}|(o_{0:t}, a_{0:t})) \end{aligned} \tag{10}$$

We need to ensure that the conditioned observation distribution remains consistent with the underlying unconditioned distribution. Specifically, we need to ensure that, any observation that has zero probability mass in the unconditioned distribution, and is therefore impossible given the current history, also has zero probability mass in the conditioned observation distribution, for all possible generation actions. By expanding the conditioned observation distribution using Bayes' Rule, we can see the relationship between the two distributions.

$$p(o_{t+1}|(o_{0:t}, a_{0:t}), \hat{a}_t^g) = \frac{\overbrace{p(\hat{a}_t^g|(o_{0:t}, a_{0:t}), o_{t+1})}^{\text{latent generation encoder}} \overbrace{p(o_{t+1}|(o_{0:t}, a_{0:t}))}^{\text{unconditioned world model}}}{\underbrace{p(\hat{a}_t^g|(o_{0:t}, a_{0:t}))}_{\text{latent action evidence}}} \tag{11}$$

We see in the numerator of Equation 11 that, if the unconditioned world model prior has a probability of zero for a given action, the numerator will be zero. This appears to ensure that the conditioned probability will also be zero - however this neglects to account for the denominator also being zero, which may result in unpredictable behaviour. However, if we can ensure that the denominator is always non-zero, then this consistency between the conditioned and unconditioned world model will hold. The denominator being zero indicates that the given LG action is impossible given the observation-action history. Therefore, we need to ensure that all LG actions are possible, given any history. We can expand the denominator as below:

$$p(\hat{a}_t^g|(o_{0:t}, a_{0:t})) = \int_{\mathcal{O}} \underbrace{p(\hat{a}_t^g|(o_{0:t}, a_{0:t}), o)}_{\text{latent generation encoder}} \overbrace{p(o|(o_{0:t}, a_{0:t}))}^{\text{unconditioned world model}} \quad \mathrm{d}o. \tag{12}$$

We cannot affect the unconditioned world model, as the distribution over future observations is inherent to the environment. However, this distribution must be non-zero for some observation o. Therefore, if the latent generation encoder is non-zero for all observations o, the integral must be non-zero. To ensure this, we set the distribution over which the latent generation actions are sampled to be a mixture of a learned distribution, conditioned on the history and uniform noise.

$$p_\psi(\hat{a}_t^g|h_{t+1}) = (1-\epsilon)q_\psi(\hat{a}_t^g|h_{t+1}) + \epsilon U(\hat{a}^g), \tag{13}$$

where $U(\hat{a}_t^g)$ is a uniform distribution across all latent generation actions.

4.2 World Model Output Selection

When training the world model, the latent generation model aims to learn expressive latent generation actions $\hat{a}^g$, such that these generation actions contain the relevant information required for the LG decoder to generate previously unseen parts of the next observation o_{t+1}. However, some observations may only contain parts of the environment that have already been observed, and so can be generated accurately, exclusively using history h_t. This presents two issues; the first is that, for a potentially large proportion of time steps, the adversary actions should have no effect on generation. However, the adversary will still take an action at these states, and due to the noisy reward signal, these actions may still induce a policy update when the adversary is trained. This results in additional noise to adversary training, resulting in a more difficult learning problem. The second issue is, although these adversary actions should have no effect, errors in the world model may enable certain actions to 'break' the consistency of the world model.

Therefore, we aim to learn a classifier that identifies these no-effect states, and at inference, rely on this classifier to select whether to rely on the conditioned world model, or to instead use a world model unconditioned on the generation actions.

As each latent state is a set of discrete latent vectors, we can assess a world model's $p(z_t|\cdot)$ accuracy by considering the error rate for a single predicted state

$$\text{Error Rate}\Big(p(z_t|\cdot)\Big) = \frac{1}{n}\sum_{i=0}^{n} \mathbf{1}\left\{\arg\max_{z^{(i)}} p(z^{(i)}|\cdot) \neq \arg\max_{z^{(i)}} p_\phi(z^{(i)}|o_t)\right\}. \tag{14}$$

Therefore, we can compare error rates between a conditioned and unconditioned world model. For each prediction, we can evaluate a selector boolean b_t, which indicates whether the unconditioned error rate matches or is lower than conditioned error rate.

$$b_t = \text{Error Rate}\Big(p_\psi(z_t|h_t)\Big) \leq \text{Error Rate}\Big(p_\psi(z_t|h_t, a_t^g)\Big) \tag{15}$$

We then train a selector predictor, with a standard cross-entropy loss, which predicts $\hat{b}_t$.

$$\text{selector predictor}: \qquad \hat{b}_t \sim p_\psi(\hat{b}_t|h_t) \tag{16}$$

As when the world model is being trained, the conditioned world model has access to the true, albeit noisy, latent generation actions, if the b_t predictor is true, meaning no additional accuracy was gained with the conditioned world model, it suggests that there is likely to be no additional information in these actions. Therefore at inference, if $\hat{b}_t$ is true, we use the output of the unconditioned world model to generate the next observation. We also use $\hat{b}_t$ to mask out adversary policy updates on time steps where the generation actions will have no effect.

Table 1: World Model Error Rates

	unconditioned	*conditioned*			*selector*
Noise mixing ϵ	n/a	0.0	0.2	0.5	0.2
True-Action Error Rate (%)	2.02	0.55	0.65	1.12	0.70
Post-Exploration Error Rate (%)	0.62	3.16	0.80	0.73	0.64

4.3 World Model Evaluation

To evaluate world model performance, we aim to assess both controllability, and consistency. In order to assess controllability, we evaluate the mean error rate when the world model is provided with LG actions sampled from the LG encoder, compared to an unconditioned world model. This aims to evaluate the level of information imparted by the LG actions. In order to assess the consistency, we evaluate the mean error rate when the world model is provided with random LG actions. However, in order to better assess the world model generation accuracy when all information is contained in the history h_t, we consider the *post-exploration* error rate, where we evaluate the mean error rate for $t > T_e$, where T_e is approximately the number of steps required for the full environment to be observed. We evaluate world model performance using the key-minigrid environment (Mead et al., 2025), where $T_e = 100$.

In Table 1, we clearly see the necessity of the generation action noise for world model consistency, with significantly higher post-exploration error rate when no noise is added. However, we also see the controllability reduction with higher noise levels, as the true-action error rate increases. The selector world model matches the consistency of the unconditioned world model, with minimal reduction in controllability. Appendix D contains additional details on world model evaluation.

5 World Model UED

With the implementation of this adversarially-controllable world model, we are able to train an RL adversary policy capable of online control of the student trajectory generation.

Dynamically-controlled Model Generation (DyMGen). Algorithm 1 outlines the student and adversary training. As the student explores, the adversary selects latent generation actions to control the generation of the next observation. After the trajectory has been completed, the dense MNA reward is computed, and assigned to the adversary actions, which is then used to update the adversary policy.

Experimental Setup. In order to assess the effectiveness of DyMGen, we examine performance against the most comparable true-environment UED method, DEGen Mead et al. (2025). We evaluate performance in the three domains explored in Mead et al. (2025), Key Minigrid, Minigrid and Sokoban, where prior work has shown online environment generation using MNA-scoring is effective.

As is standard, we assess student performance on a held-out set of hand-designed levels, using the same level set as in Mead et al. (2025). Further environment details can be found in Appendix C.

Baselines. For baselines, we compare performance between our method, the Dynamically-controlled Model Generation (WM - DyMGen), to a controllable World Model using random generation actions (WM - Random). We also compare to domain randomisation in the true environment (Env - DR), and DEGen (Mead et al., 2025) in the true environment (Env - DEGen). We also include world-model-based PLR (WM - PLR), the method proposed in Güzel et al. (2025), where high-scoring initial states are re-sampled using PLR.

World Model. For each domain, we train a single world model, and use this single model for all world model baselines. We provide additional details on world model training, including all hyperparameters, in Appendix B.

Algorithm 1 DyMGen

Initialise: student policy π_{θ_1}, generator policy Λ_{θ_2}
while not converged **do**
  // Sample N trajectories
  for $n \in 1 : N$ **do**
    Sample o_1
    Encode $z_1 = \arg\max p_\phi(\cdot|o_1)$
    // Take T student steps
    for $t \in 1 : T$ **do**
      // Get student action
      Decode $o_t = \arg\max p_\phi(\cdot|z_t)$
      Sample student $a_t \sim \pi_\theta(a_t|o_t)$ action
      // Get adversary action
      $h_t = f_\psi(z_{0:t}, a_{0:t})$
      Sample adversary Λ action $a_t^g \sim \Lambda_{\theta_2}(a_t^g|h_t)$
      // Get next latent state
      $z_{t+1} = \arg\max p_\psi(\cdot|h_t, \hat{a}_t^g)$
    end for
    compute adversary reward using student trajectory
    assign reward to adversary trajectory
  end for
  Update θ_1 according to sampled student trajectories
  Update θ_2 according to sampled adversary trajectories
end while

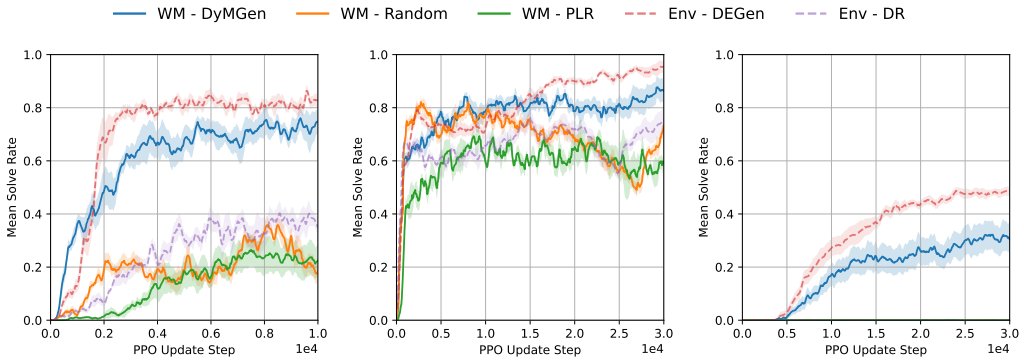
6 Results

From Figure 3, we see that DyMGen clearly outperforms the world model baselines, whilst also outperforming naive domain randomisation. We see substantial gains in Key Minigrid, and in Sokoban; DyMGen is the only world model method that results in non-zero performance on the evaluation level set. In Minigrid, the performance remains relatively comparable between all methods, which aligns with results shown in existing work (Rutherford et al., 2024; Mead et al., 2025). Even after training for three times more steps than key minigrid, there remains minimal performance difference. This can likely be explained by the lack of complexity possible in standard minigrid levels, as without additional obstacles, there is limited capacity for highly difficult levels. However, in both other domains, we see the ineffectiveness of naive methods in generating sufficiently challenging levels. We also see very comparable results between WM - Random and WM - PLR. This highlights the issues arising from only controlling the initial state. This lack of controllabilty means that, in these partially observable settings, WM - PLR effectively becomes equivalent to domain randomisation. Whilst DyMGen does not match the performance of DEGen, it consistently outperforms the remaining baselines, and does so without requiring a handcrafted environment generator.

7 Discussion & Limitations

From the results presented, we clearly demonstrate that a latent-generation-action controlled world model enables online, adversary-controlled trajectory generation that can be leveraged to improve student performance. Where relying on random world model generation, or PLR controlling only the initial state, agents show consistently worse zero-shot capacity compared to those trained using an adversary-controlled world model. This indicates that the controllable world model enables the adversary to generate the level structures required for student learning. Whilst the demonstrated results illustrate a clear and meaningful contribution, there are a number of limitations with this work.

Performance Gap: Currently, there consistently remains a performance gap between the performance of DyMGen and DEGen. There are a number of possible explainations for this gap, but one of the evident issues is adversary exploitation of the world model. We observed generated trajectories



(a) Key Minigrid (b) Minigrid (c) Sokoban

Figure 3: Comparison between DyMGen and baselines of solve rate on hand-designed evaluation levels. Lines show the mean across 8 seeds and shaded regions show standard error.

inconsistent with the true environment, specifically in minigrid, and key minigrid, with trajectories containing multiple goals, keys and doors. This widens the set of possible levels, and so may worsen performance on evaluation sets that can only contain one of these objects. This potentially could be reduced with increased compute or data; we have already shown the performance benefits of increased data in Sokoban in Appendix C.8. Alternatively, explicitly trying to maintain consistency through the UED score-function, or introducing adversarial pressure on the world model during training may be solutions to reducing this issue.

Computational Cost: The additional world model training requirements add a high computational overhead to training policies. Training the minigrid world models took 35 hours on an A40, and training the sokoban world model took 20 hours on 4xH100. If a suitable simulator is available, it is clearly optimal to train in the real environment.

Dataset Requirement: DyMGen does require existing datasets for world model training, and this may be challenging to obtain in certain domains. However, there are many compelling domains, such as autonomous driving Hu et al. (2023), where these datasets already exist, and are clear candidates for methods similar to DyMGen.

Domain Limitations: The set of domains used in this work remain very simple. They contain clear boundaries between what has and has not been observed, and so learning expressive latent generation is likely comparably easy compared to more complex domains. However, these domains do represent a notable first step, and demonstrate that latent generation actions can be used for world model UED.

Environment Stochasticity: Whilst not unique to this work, UED methods may perform poorly in highly stochastic environments (Jiang et al., 2022). UED necessarily shifts the training distribution, but this distribution shift may result in suboptimal agents, given the ground truth level distribution. Combining DyMGen with existing methodology (Jiang et al., 2022) to alleviate this issue may allow for improved curriculum design in stochastic environments.

8 Conclusion

In this work, we introduce a novel world model architecture that enables controllable generation of the world as it is explored by an agent. Using this, we have implemented an adversarial teacher, capable of generating a automated training curriculum that significantly improves student zero-shot generalisation. Following from existing work, we examine the performance of this method in three environments, and show that agents trained in using world model consistently exceed the performance of naive random training, even with access to the true environment. Whilst there are evident limitations with this work, we believe that this work highlights there is significant potential in the use of world model UED, and suggest this represents a promising line of future work, potentially enabling substantially more complex and general environment generation through the use of automatically generated curricula in trained, controllable world models.

AI USE STATEMENT

In this work, we used generative AI tools primarily only for coding, and enabling easier implementation of ideas we proposed. We did not use generative AI to generate synthetic data, or process existing datasets, nor did we use it to develop our methodology for this work, be it writing of mathematical proofs, design of research direction, or interpretation of results. Additionally, we used generative AI tools for editing of this manuscript, and in writing code to generate plots and images used. We also used generative AI as a tool to discover related and relevant research. All generated code was thoroughly hand-checked, and was also subject to explicit tests to ensure correct behaviour was being produced.

REPRODUCIBILITY STATEMENT

We outline the full methodology and architectures necessary for the controllable world model training Sections 3, 4. Additional world model training details are provided in Appendix B, including hyperparameters. Section 5 clearly outlines the world model UED training pipeline, with hyperparameters provided in Appendix E. We include full environment specifications in Appendix C for all domains used, including details on the hand-designed evaluation level set. We link to all relevant code from related work in Appendix B, and in due course, will release all code and datasets necessary to reproduce this work.

REFERENCES

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*, pp. 1361–1395. PMLR, 2023.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Uljad Berdica, Kelvin Li, Michael Beukman, Alexander David Goldie, Perla Maiolino, and Jakob Nicolaus Foerster. Robust offline learning via adversarial world models. In *The Third Workshop on New Frontiers in Adversarial Machine Learning*, 2024.

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.

Michael Beukman, Samuel Coward, Michael Matthews, Mattie Fellows, Minqi Jiang, Michael D. Dennis, and Jakob Nicolaus Foerster. Refining minimax regret for unsupervised environment design. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 3637–3657. PMLR, 2024.

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.

Hojun Chung, Junseo Lee, Minsoo Kim, Dohyeong Kim, and Songhwai Oh. Adversarial environment design via regret-guided diffusion models. *Advances in Neural Information Processing Systems*, 37:63715–63746, 2024.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28, 2015.

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

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

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

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International conference on machine learning*, pp. 7480–7512. PMLR, 2023.

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.

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

Daniel Furelos-Blanco, Charles Pert, Frederik Kelbel, Alex F Spies, Alessandra Russo, and Michael Dennis. Beyond fixed tasks: Unsupervised environment design for task-level pairs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 21145–21153, 2026.

Ahmet H Güzel, Matthew Thomas Jackson, Jarek Luca Liesen, Tim Rocktäschel, Jakob Nicolaus Foerster, Ilija Bogunovic, and Jack Parker-Holder. Imagined autocurricula. In *Advances in Neural Information Processing Systems*, 2025.

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

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

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

Anthony Hu, Lloyd Russell, Hudson Yeo, Zak Murez, George Fedoseev, Alex Kendall, Jamie Shotton, and Gianluca Corrado. Gaia-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*, 2023.

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, 2021a.

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

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Andrei Lupu, Heinrich Küttler, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. Grounding aleatoric uncertainty for unsupervised environment design. *Advances in Neural Information Processing Systems*, 35:32868–32881, 2022.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Harry Mead, Bruno Lacerda, Jakob Foerster, and Nick Hawes. Improving regret approximation for unsupervised dynamic environment generation. *Advances in Neural Information Processing Systems*, 2025.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. In *International Conference on Learning Representations*, 2023.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *International Conference on Learning Representations*, 2017.

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.

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*, pp. 17473–17498. PMLR, 2022.

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*, pp. 835–853. PMLR, 2020a.

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*, 2020b.

Jan Robine, Marc Höftmann, Tobias Uelwer, and Stefan Harmeling. Transformer-based world models are happy with 100k interactions. In *International Conference on Learning Representations*, 2023.

Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. *Advances in Neural Information Processing Systems*, 37:16071–16101, 2024.

Junyoung Seo, Hyunwook Choi, Minkyung Kwon, Jinhyeok Choi, Siyoon Jin, Gayoung Lee, Junho Kim, JoungBin Lee, Geonmo Gu, Dongyoon Han, et al. Grounding world simulation models in a real-world metropolis. *arXiv preprint arXiv:2603.15583*, 2026.

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.

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.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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.

Mengjiao Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *arXiv preprint arXiv:2310.06114*, 1(2):6, 2023.

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

A MAXIMISED NEGATIVE ADVANTAGE DERIVATION

Below is an outlined derivation of the Maximised Negative Advantage regret approximation (Mead et al., 2025). Approximating policy regret presents two challenges.

$$\text{Regret}(\pi, \theta) = -U(\pi, \theta) + U(\pi^*_\theta, \theta). \tag{17}$$

The first is approximating $U(\pi, \theta)$, the expected return of the current policy. This can be approximated relatively easily using a learned value function $\hat{V}(s)$, based on policy trajectories. The more challenging problem is approximating $U(\pi^*_\theta\theta)$, the expected return of the optimal policy, as the optimal policy is generally unknown.

If we assume a deterministic environment, and we have a true value function $V(s)$ for any policy, given a trajectory of length T, we can lower bound $U(\pi^*_\theta, \theta)$

$$U(\pi^*_\theta, \theta) \geq \max \begin{pmatrix} V(s_0), \\ \gamma V(s_1) + r(s_0, a_0), \\ \vdots \\ \gamma^T V(s_T) + \sum_{k=0}^{T-1} \gamma^k r(s_k, a_k) \end{pmatrix}. \tag{18}$$

We label this maximum over value functions

$$V_n^{\max}(s_t) = \max \begin{pmatrix} V(s_t), \\ \vdots \\ \gamma^n V(s_{t+n}) + \sum_{k=t}^{t+n-1} \gamma^{k-t} r(s_k, a_k) \end{pmatrix}. \tag{19}$$

Therefore if we have the true value function for a policy, we can lower bound the policy regret with the equation

$$\text{Regret} \geq -V(s_0) + V_T^{\max}(s_0). \tag{20}$$

In practise, we assume that we do not have access to the true value function, and instead require a learned value function $\hat{V}(s)$. However, as this learned value function may overestimate the state value, the inequality in Equation 18 does not necessarily hold. 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 $n < T$. However, this may then be an overly-conservative approximation for $U(\pi^*_\theta, \theta)$. Similarly, using $\hat{V}(s)$ rather than $V(s)$ to approximate $U(\pi, \theta)$ introduces bias into the value estimate. We can reduce the bias of our approximation 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. By combining these two approximations, we get an n-step regret approximation

$$\hat{G}_t^{(n)} = -\Big(\gamma^n V(s_{t+n}) + \sum_{k=t}^{t+n-1} \gamma^{k-t} r(s_k, a_k)\Big) + \hat{V}_n^{\max}(s_t). \tag{21}$$

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, an exponentially-weighted mean of the n-step regret approximations.

$$\hat{G}_t^{\lambda} = (1 - \lambda) \sum_{n=0}^{\infty} \lambda^n \hat{G}_t^{(n)}. \tag{22}$$

The full regret approximation can be computed across the full trajectory

$$\text{MNA} = \left(\frac{1}{T} \sum_{t=0}^{T} \hat{G}_t^{\lambda} \right) \cdot \hat{C} \tag{23}$$

where $\hat{C}$ is 1 if the agent has solved the level at least once, and 0 otherwise.

B Additional World Model Training Details

B.1 Training Dataset

For training the world models, we used a fixed dataset. For key minigrid, and minigrid, this consisted of $250,000$ environment trajectories, each containing 512 time steps. These datasets were generated from rolling out a mix of random and expert policies (20% random, 80% expert) on levels generated through domain randomisation. We use a dataset generated on random levels to demonstrate that the adversary is able to shift the distribution of trajectories the world model generates away from the dataset on which it was trained on. Our results clearly show the adversarially-controlled world model agents outperform agents trained using domain randomisation, even though the world model was trained from this latter distribution of levels. For Sokoban, we did need to include some non-randomly generated levels, as DR levels are so highly unlikely to be solvable. 25% of the dataset was generated using true-environment UED to generate levels, which were played using the same mix of random and expert policies. For Sokoban, we also had to scale up the size of the world model datasets, which is detailed in Appendix C.8.

B.2 Observation Model

The environment observations consist of both image data, and discrete variables. We used a CNN (LeCun et al., 1989) to learn an image embedding, which was concatenated with the discrete variables embedding, which was then passed through an MLP to learn the latent state distribution. Based on previous work (Hafner et al., 2021; Robine et al., 2023), we use the loss function:

$$\mathcal{L}_{\phi}^{\text{Obs}} = \mathbb{E}\left[\sum_{t=0}^{T} \underbrace{-\log p_{\phi}(o_t|z_t)}_{\text{reconstruction loss}} - \alpha \underbrace{H(p_{\phi}(z_t|o_t))}_{\text{encoder entropy}}\right], \tag{24}$$

$$z_t \sim p_{\phi}(z_t|o_t) \tag{25}$$

where α is a hyperparameter that can be tuned to select the relative weight of the losses. As the latent state encoder distribution is a set of n categorical distributions, each with m categories, the latent states sampled are discrete. To propagate gradients through discrete samples, we use straight-through gradients (Bengio et al., 2013), following previous work (Hafner et al., 2021; Robine et al., 2023).

When training the decoder, uniform noise was added to the decoder distribution before z_t was sampled to improve the robustness of the decoder. For each observation, the amount of noise was sampled randomly from between 0 and 0.05. The observation model hyperparameters are shown in Table 2

Table 2: Observation Model Hyperparameters.

Parameter	
Num updates	500000
lr	1e-4
Batch Size	32
Anneal lr	yes
Optimiser	AdamW
Weight Decay	1e-6
Anneal lr	yes
Entropy coefficient (α)	1e-4
Num conv layers	3
Num Conv Channels	3
Conv kernel size	1x1
Num fully-connected layers	3
Hidden layer Size	512
Hidden activation	gelu
Num latent state vectors (n)	32
Num latent state categories (m)	16

B.3 Dynamics Model

The dynamics model is a standard transformer, with approximately 45 million parameters. The dynamics model hyperparameters are shown in Table 3.

B.4 Latent Generation Model

The dynamics model, and the latent generation model is trained using the loss:

$$\mathcal{L}_{\psi}^{\text{Dyn}} = \mathbb{E}\left[\sum_{t=0}^{T} \underbrace{KL\Big[\text{sg}(p_{\phi}(z_{t+1}|o_{t+1}))||p_{\psi}(\hat{z}_{t+1}|h_t, \hat{a}_t^g))\Big]}_{\text{latent generation decoder loss}} \underbrace{-\log p_{\psi}(r_t|\bar{h}_t, \hat{a}_t^g)}_{\text{reward predictor}} \underbrace{-\log p_{\psi}(c_t|\bar{h}_t, \hat{a}_t^g)}_{\text{continue predictor}}\right] \tag{26}$$

$$\hat{a}_t^g \sim p_{\psi}(\hat{a}_t^g|h_{t+1}) \tag{27}$$

As with the observation model, each latent generation action a^g is a set of k discrete variables, sampled from g categories. Therefore, straight-through gradients are also required to propagate gradients through discrete samples from the latent generation decoder. At training, $\bar{h}_t = [h_t, z_{t+1}]$, where z_{t+1} is drawn from the observation encoder for the next observation o_{t+1}. At inference, $\bar{h}_t = [h_t, \hat{z}_{t+1}]$, where $\hat{z}_{t+1}$ is drawn from LG decoder.

For the reward predictor, the rewards are two-hot encoded, aligning with previous work (Hafner et al., 2023). Cross-entropy loss is used for both the two-hot encoded reward predictor, and the continue predictor.

B.5 Selector Model

The selector model is trained using the loss:

$$\mathcal{L}_{\psi}^{\text{Sel}} = \mathbb{E}\left[\sum_{t=0}^{T} -\log p_{\psi}(b_t|h_t)\right]. \tag{28}$$

However, the the selector model also requires the unconditioned world model to be trained, using the unconditioned dynamic loss:

$$\mathcal{L}_{\psi}^{\text{UDyn.}} = \mathbb{E}\left[KL\Big[\text{sg}(p_{\phi}(z_{t+1}|o_{t+1}))||p_{\psi}(\hat{z}_{t+1}|h_t))\Big] - \log p_{\psi}(r_t|h_t) - \log p_{\psi}(c_t|h_t)\right]. \tag{29}$$

As the unconditioned world model relies on the same history vector h_t from the dynamics model, the additional overhead to both training and inference is minimal.

Table 3: World Model Hyperparameters.

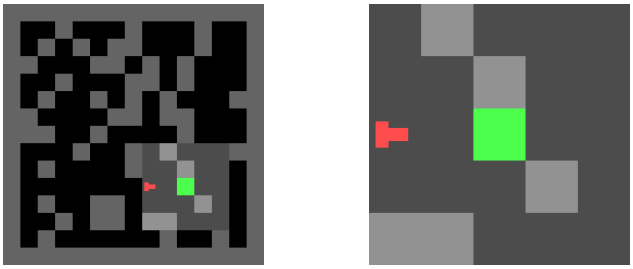
Parameter	**Key Minigrid and Minigrid**	**Sokoban**
Num updates	2000000	2500000
lr	1e-4	8e-4
Batch Size	16	64
Anneal lr	yes	
Optimiser	AdamW	
Weight Decay	1e-6	
Hidden activation	gelu	
Dynamics Model		
Embed dim	512	
Feedforward dim	2048	
Dropout	0.1	
Num layers	8	
Num heads	8	
Latent Generation Model		
Num fully connected layers	2	
Num latent generation actions (k)	6	8
Num latent generation categories (g)	6	8
Generation action noise (ϵ)	0.2	

C Environment Specification

C.1 Minigrid and Key Minigrid

We use the key minigrid implementation from existing UED work (Mead et al., 2025). Examples of levels are shown in Figure 11.

Observations: The agent, depicted in red, observes a 5x5 square in front of it. It also has access to its absolute direction, e.g. whether it is facing *North*, *South*, *East* or *West*, as well as whether it has picked up the key.



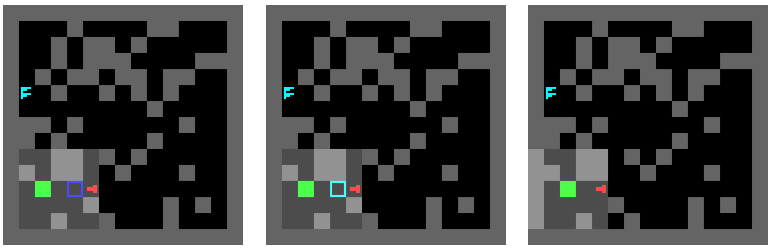
(a) Environment State (b) Agent Observation

Figure 4: Minigrid Agent Observations

Rewards: The agent receives reward r_g if it reaches the goal. The size of this reward is determined by the number of steps taken to reach the goal T, and the parameter $T_{\max}$, specifying the maximum number of steps before an episode terminates.

$$r_g = 1 - 0.9\left(\frac{T}{T_{\max}}\right) \tag{30}$$

Actions: At each step, the agent can either move *forward*, turn *left* or turn *right*, or take a *use* action. When a door is directly in front of the agent, if it picked up the key, the use action will unlock the door



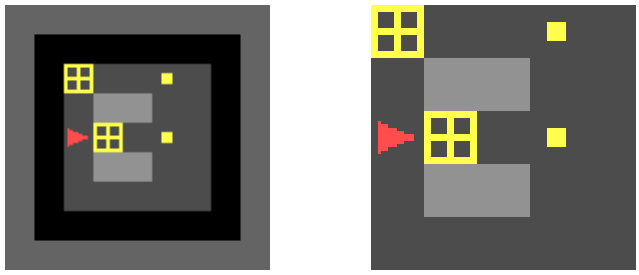
(a) Locked Door (b) *use* action (c) Unlocked Door

Figure 5: Key Minigrid Door Unlocking

C.2 SOKOBAN

We use the partially-observable sokoban version from Mead et al. (2025). The agent must push all boxes to box goal locations.

Observations: As with minigrid, the agent observes a 5x5 square, with absolute direction included in the observation as well.

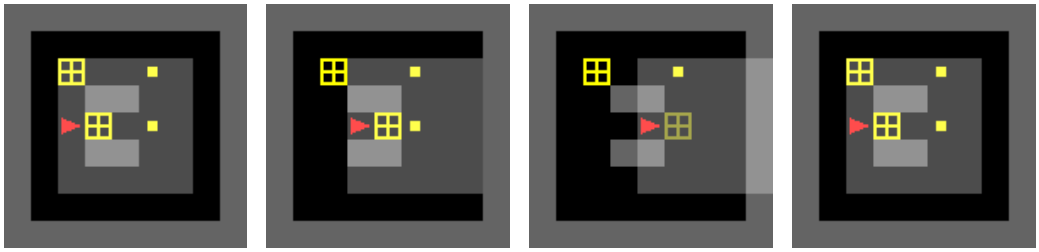


(a) Environment State (b) Agent Observation

Figure 6: Sokoban Agent Observations

Rewards: The agent receives reward r_g once all boxes have been pushed onto box goal locations. Note that if a level has an unequal number of boxes and goal locations, it is unsolvable. The size of this reward is determined using the same calculation as minigrid.

Actions: The sokoban action space is the same as minigrid, but is augmented with a *reset* action. The reset action resets the agent and all boxes to their original position, but does not reset the step count - the reward is calculated using the total number of steps, not the number of steps since the last reset.



(a) Initial State (b) *forward* action (c) Box on Goal (d) *reset* action

Figure 7: Sokoban Example Actions

C.3 ADVERSARY

The adversary Λ aims to maximise a chosen regret approximation, in this work we exclusively use MNA, by controlling the world model generation through the latent generation actions. In this work, we use an RNN-based adversary. During inference, we deterministically sample from the world model, with the exception of the initial state, which we sample from the softmax of the world model output distribution. However, to ensure that the adversary environment remains deterministic, rather than directly sampling from the softmax, we instead first sample Gumbel noise

$$\xi_{\sim} \operatorname{Gumbel}(0, 1) \tag{31}$$

and use this to condition the initial adversary observation. We can then sample the adversary latent generation actions, compute the world model output distribution, and then sample from this distribution using the Gumbel-Max trick, enabling the adversary action to be conditioned on how the output distribution will be sampled.

Observations: The adversary uses an augmented version of the history vector h_t from the dynamics model. Along with h_t, we also concatenate the student's current state value $\hat{V}(s_t)$, as well as the current time step t, and the action the student has taken a_t, giving the augmented vector

$$\tilde{h}_t = [h_t, a_t, t, \hat{V}(s_t)] \tag{32}$$

We include the student action a_t as, although the h_t does contain a_t in its context, the dynamics model parameters are frozen when the adversary is trained, so explicitly providing a_t ensures the adversary is capable of determining the student's most recent action.

Rewards: In this work, the adversary rewards are computed using MNA. However, the adversary could be trained with any regret approximation, although the full benefits of online generation would be lost for sparser metrics.

$$r_t^{\Lambda} = \hat{G}_t^{\lambda} \cdot \hat{C}. \tag{33}$$

Actions: The adversary action space is the world model latent generation action space. As the latent generation actions consist of k discrete variables, each taking one of a possible g categories, the adversary action space is $b \times g$, with each action consisting of k sub-actions.

$$a^{\Lambda} = \{a^1, a^2, a^3, \dots a^b\} \tag{34}$$

C.4 EXAMPLE ROLLOUTS

In order to illustrate the effectiveness of the learned latent generation actions, we show example rollouts in each of Key Minigrid, Minigrid and Sokoban, presented in Figures 8, 9 and 10 respectively. These plots show how the latent generation actions vary the generation of the environment. For each Figure, the same student actions are taken at each timestep. However, each row represents a different set of randomly generated latent generation actions. The greyed-out observations indicate that the same latent generation actions were used to generate this observation as the row above, allowing visualisation of how the same initial trajectory path can result in a variety of possible generations, when the latent generation actions differ. We can also clearly see it is only the unobserved parts of the trajectory are controlled, and when an observation contains exclusively already-observed cells, the latent generation actions have no effect.

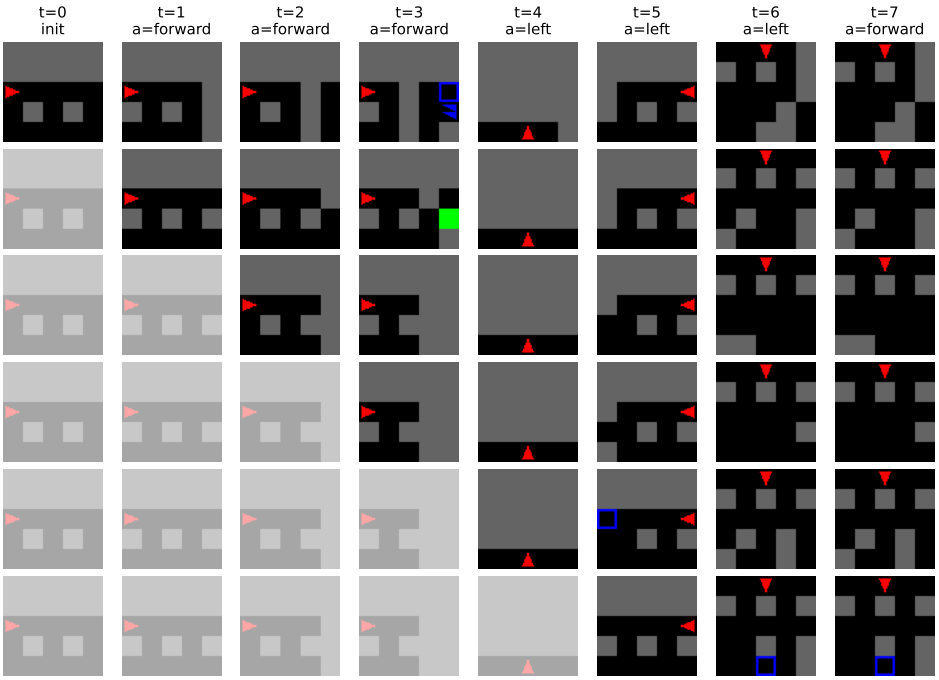


Figure 8: Key Minigrid Example Rollout

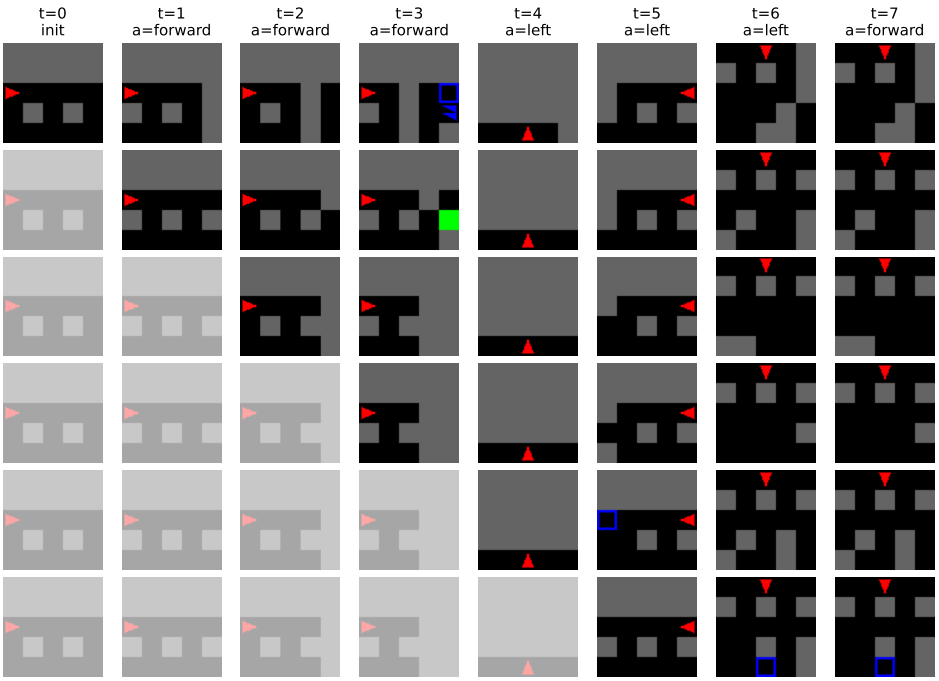


Figure 9: Minigrid Example Rollout

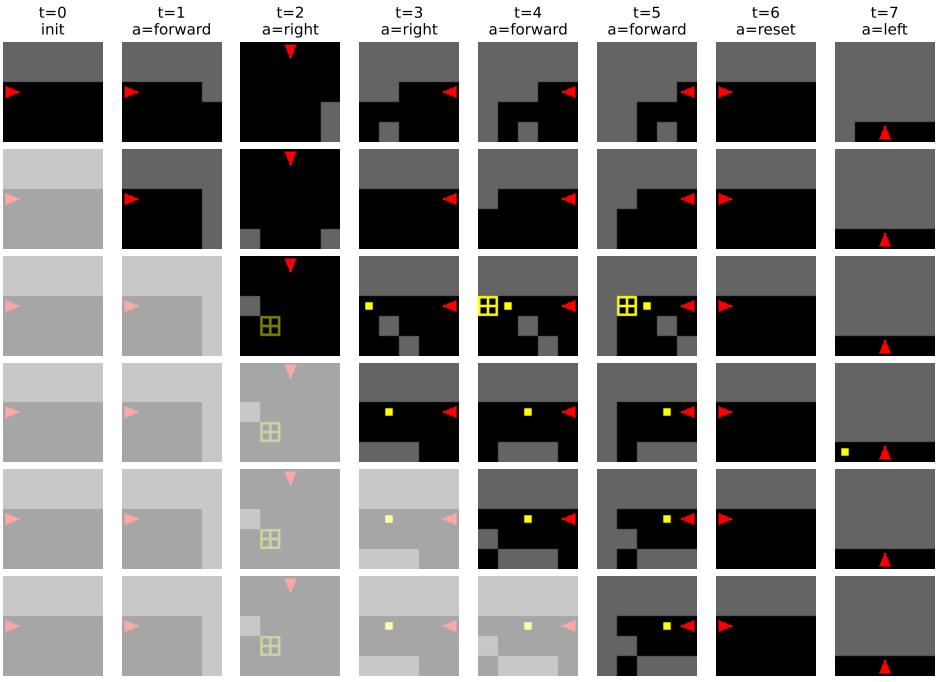


Figure 10: Sokoban Example Rollout

C.5 Evaluation Levels

Figures 11, 12 and 13 illustrate the hand design evaluation levels for key minigrid, minigrid and sokoban respectively, as used in previous work (Mead et al., 2025)

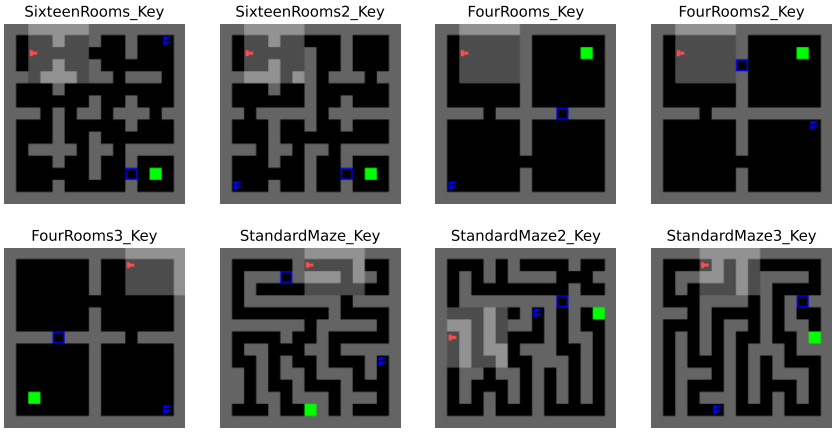


Figure 11: Hand designed evaluation levels for key minigird

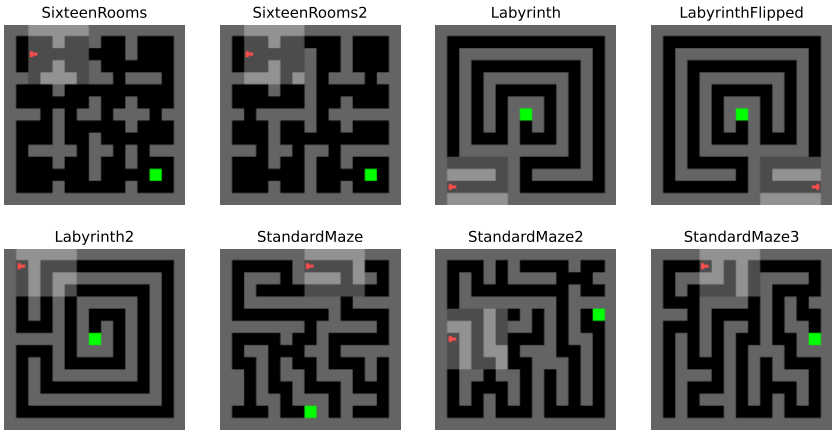


Figure 12: Hand designed evaluation levels for minigird

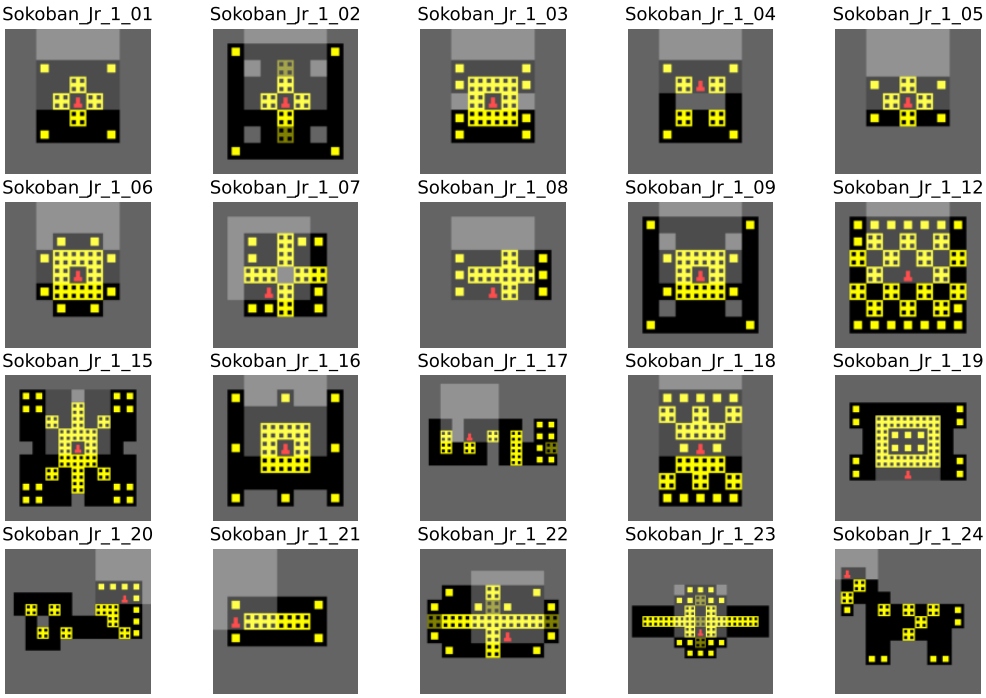


Figure 13: Hand designed evaluation levels for sokoban

C.6 Tabular Evaluation Level Results

Tables 4, 5 and 6 outline the solve rate of each method, by specific evaluation level. Results in **bold** indicate the best performing method trained in a world model. Results in *italics* indicate the best performing method overall.

Table 4: Key Minigrid Eval Level Solve Rate, showing Mean and Standard Error

Level	WM - DyMGen	WM - Random	WM - PLR	Env - DEGen	Env - DR
FourRooms2_Key	**0.89 ± 0.10**	0.25 ± 0.07	0.50 ± 0.19	*0.93 ± 0.05*	0.55 ± 0.13
FourRooms3_Key	**0.75 ± 0.12**	0.20 ± 0.09	0.48 ± 0.16	*0.98 ± 0.02*	0.54 ± 0.12
FourRooms_Key	***0.83 ± 0.10***	0.09 ± 0.05	0.39 ± 0.17	*0.83 ± 0.08*	0.28 ± 0.12
SixteenRooms2_Key	**0.69 ± 0.11**	0.28 ± 0.08	0.09 ± 0.05	*1.00 ± 0.00*	0.21 ± 0.08
SixteenRooms_Key	***0.99 ± 0.01***	0.41 ± 0.10	0.39 ± 0.09	*0.99 ± 0.01*	0.60 ± 0.11
StandardMaze2_Key	***0.54 ± 0.11***	0.41 ± 0.13	0.03 ± 0.03	0.44 ± 0.11	0.21 ± 0.08
StandardMaze3_Key	**0.84 ± 0.10**	0.00 ± 0.00	0.00 ± 0.00	*0.86 ± 0.08*	0.25 ± 0.11
StandardMaze_Key	**0.41 ± 0.10**	0.03 ± 0.03	0.00 ± 0.00	*0.64 ± 0.11*	0.06 ± 0.04
Mean	**0.74 ± 0.04**	0.21 ± 0.05	0.23 ± 0.07	*0.83 ± 0.03*	0.34 ± 0.06

Table 5: Minigrid Eval Level Solve Rate, showing Mean and Standard Error

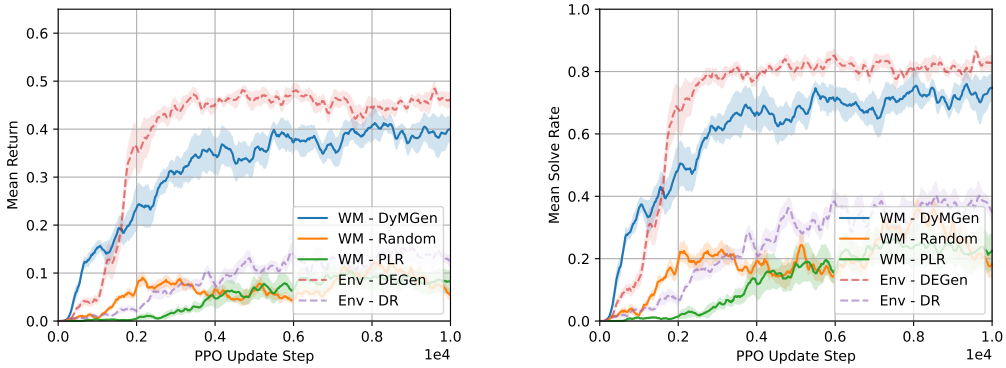
Level	WM - DyMGen	WM - Random	WM - PLR	Env - DEGen	Env - DR
Labyrinth	**0.71 ± 0.15**	0.36 ± 0.10	0.55 ± 0.10	*0.98 ± 0.02*	0.51 ± 0.16
Labyrinth2	**0.75 ± 0.13**	0.19 ± 0.08	0.20 ± 0.07	*0.79 ± 0.12*	0.40 ± 0.14
LabyrinthFlipped	**0.73 ± 0.13**	0.53 ± 0.13	0.28 ± 0.13	*0.91 ± 0.09*	0.38 ± 0.15
SixteenRooms	***1.00 ± 0.00***	***1.00 ± 0.00***	***1.00 ± 0.00***	*1.00 ± 0.00*	*1.00 ± 0.00*
SixteenRooms2	***1.00 ± 0.00***	0.99 ± 0.01	0.80 ± 0.12	*1.00 ± 0.00*	*1.00 ± 0.00*
StandardMaze	***1.00 ± 0.00***	0.91 ± 0.06	0.75 ± 0.09	*1.00 ± 0.00*	0.99 ± 0.01
StandardMaze2	0.75 ± 0.13	**0.79 ± 0.05**	0.28 ± 0.13	*1.00 ± 0.00*	0.85 ± 0.06
StandardMaze3	0.93 ± 0.07	***1.00 ± 0.00***	0.92 ± 0.08	*1.00 ± 0.00*	0.93 ± 0.04
Mean	**0.86 ± 0.05**	0.72 ± 0.03	0.60 ± 0.05	*0.96 ± 0.03*	0.76 ± 0.05

Table 6: Sokoban Eval Level Solve Rate, showing Mean and Standard Error

Level	WM - DyMGen	WM - Random	WM - PLR	Env - DEGen	Env - DR
Sokoban_Jr_1_01	**0.88 ± 0.12**	0.00 ± 0.00	0.01 ± 0.01	*1.00 ± 0.00*	0.00 ± 0.00
Sokoban_Jr_1_02	**0.86 ± 0.12**	0.00 ± 0.00	0.00 ± 0.00	*1.00 ± 0.00*	0.00 ± 0.00
Sokoban_Jr_1_03	**0.30 ± 0.13**	0.00 ± 0.00	0.00 ± 0.00	*0.49 ± 0.14*	0.00 ± 0.00
Sokoban_Jr_1_04	***0.39 ± 0.12***	0.00 ± 0.00	0.00 ± 0.00	0.35 ± 0.09	0.00 ± 0.00
Sokoban_Jr_1_05	***0.33 ± 0.11***	0.00 ± 0.00	0.00 ± 0.00	0.24 ± 0.16	0.00 ± 0.00
Sokoban_Jr_1_06	**0.54 ± 0.14**	0.00 ± 0.00	0.00 ± 0.00	*0.94 ± 0.03*	0.00 ± 0.00
Sokoban_Jr_1_07	**0.65 ± 0.13**	0.00 ± 0.00	0.00 ± 0.00	*1.00 ± 0.00*	0.00 ± 0.00
Sokoban_Jr_1_08	**0.18 ± 0.12**	0.00 ± 0.00	0.00 ± 0.00	*0.50 ± 0.15*	0.00 ± 0.00
Sokoban_Jr_1_09	**0.61 ± 0.13**	0.00 ± 0.00	0.00 ± 0.00	*0.85 ± 0.06*	0.00 ± 0.00
Sokoban_Jr_1_12	**0.03 ± 0.02**	0.00 ± 0.00	0.00 ± 0.00	*0.30 ± 0.15*	0.00 ± 0.00
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	**0.81 ± 0.12**	0.00 ± 0.00	0.00 ± 0.00	*0.95 ± 0.03*	0.00 ± 0.00
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	**0.19 ± 0.11**	0.00 ± 0.00	0.00 ± 0.00	*0.69 ± 0.14*	0.00 ± 0.00
Sokoban_Jr_1_19	**0.04 ± 0.03**	0.00 ± 0.00	0.00 ± 0.00	*0.10 ± 0.06*	0.00 ± 0.00
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	**0.88 ± 0.12**	0.00 ± 0.00	0.00 ± 0.00	*0.99 ± 0.01*	0.00 ± 0.00
Sokoban_Jr_1_22	**0.16 ± 0.09**	0.00 ± 0.00	0.00 ± 0.00	*0.30 ± 0.11*	0.00 ± 0.00
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	**0.04 ± 0.02**	0.00 ± 0.00	0.00 ± 0.00	*0.09 ± 0.03*	0.00 ± 0.00
Mean	**0.34 ± 0.06**	0.00 ± 0.00	0.00 ± 0.00	*0.49 ± 0.02*	0.00 ± 0.00

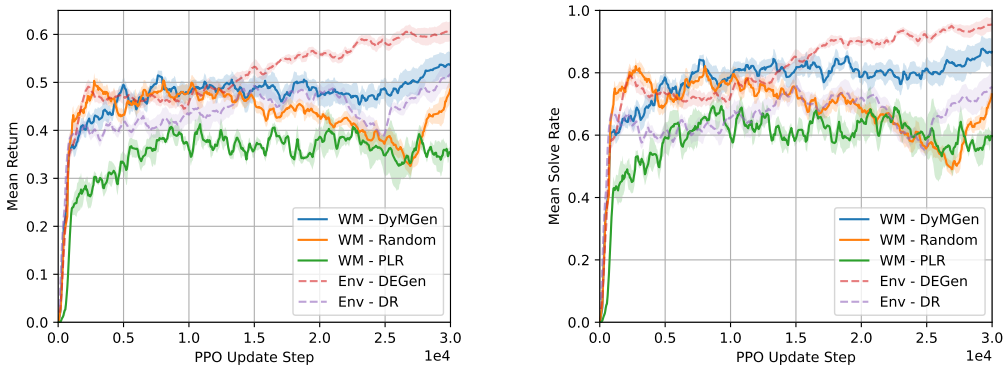
C.7 Additional Evaluation Level Results

Figures 14, 15 and 16 illustrate UED training performance, with the addtion of mean return on the training level set.



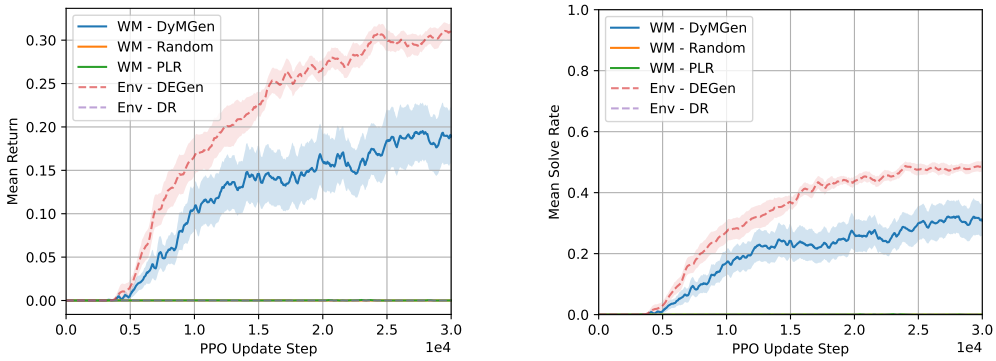
(a) Eval Levels Mean Return (b) Eval Levels Mean Solve Rate

Figure 14: Key Minigrid training plots. a) shows the mean return, and b) of each method on the hand-designed eval levels. These plots show 8 seeds for each method, with the mean and standard error.



(a) Eval Levels Mean Return (b) Eval Levels Mean Solve Rate

Figure 15: Minigrid training plots. a) shows the mean return, and b) of each method on the hand-designed eval levels. These plots show 8 seeds for each method, with the mean and standard error.

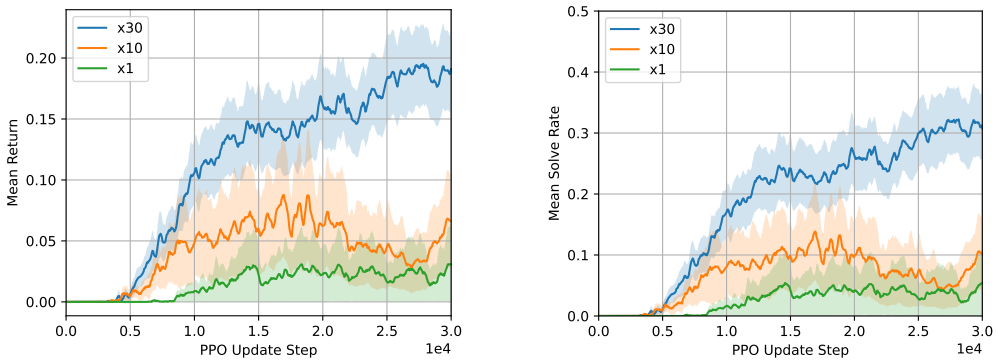


(a) Eval Levels Mean Return (b) Eval Levels Mean Solve Rate

Figure 16: Sokoban training plots. a) shows the mean return, and b) of each method on the hand-designed eval levels. These plots show 8 seeds for each method, with the mean and standard error.

C.8 Sokoban Dataset Scaling Performance

Initial testing with the sokoban environment on a dataset of equal size to that used for Key Minigrid and Minigrid showed relatively poor performance. As such, we investigated scaling the dataset size. In Figure 17, we see clear performance improvements when scaling the training dataset size from the minigrid size (x1), to 30 times larger (x30). We use this x30 world model to perform our Sokoban performance evaluations compared to baselines.



(a) Eval Levels Mean Return (b) Eval Levels Mean Solve Rate

Figure 17: Sokoban training plots, showing the effect of scaling the training dataset size. a) shows the mean return, and b) of DyMGen performed using a world model trained with each dataset size, on the hand-designed eval levels. These plots show 8 seeds for each dataset size, with the mean and standard error.

D World Model Evaluation

For evaluation of controllable world models, we aim to evaluate both world model consistency, as well as controllability.

To measure controllability, we measure the mean error rate, when the LG decoder is conditioned on the true latent generation actions $\hat{a}^g$.

$$\text{True-Action Error Rate} = \frac{1}{T}\sum_{t=0}^{T}\frac{1}{n}\sum_{i=0}^{n}\mathbf{1}\left\{\arg\max_{z^{(i)}} p_\psi(z^{(i)}|h_t, \hat{a}_t^g) \neq \arg\max_{z^{(i)}} p_\phi(z^{(i)}|o_t)\right\} \tag{35}$$

$$\hat{a}_t^g = \arg\max_{a^g} p_\psi(a^g|h_{t+1}) \tag{36}$$

We can compare this error rate between a conditioned and unconditioned world model, where the reduction in error rate in the conditioned world model indicates the amount of information contained within the latent generation actions. If these generation actions are able to impart more information to the LG decoder, this indicates greater controllability.

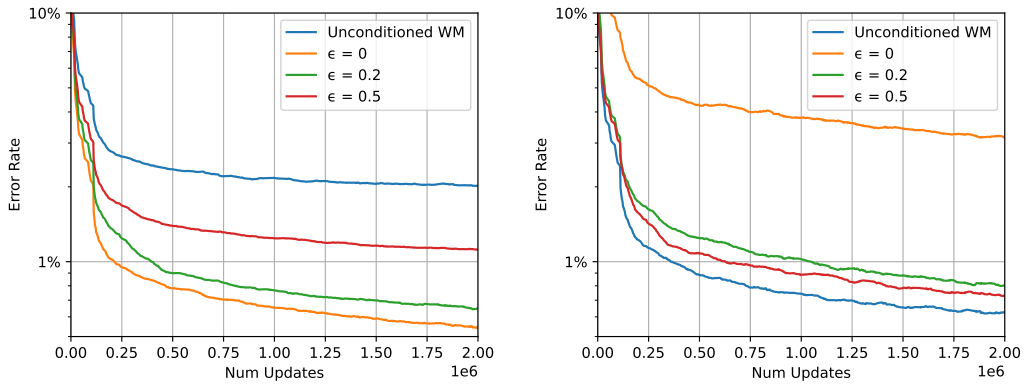
To evaluate consistency, we measure the mean error rate when the LG decoder is conditioned on random generation actions.

$$\text{Post-Exploration Error Rate} = \frac{1}{T}\sum_{t=T_e}^{T}\frac{1}{n}\sum_{i=0}^{n}\mathbf{1}\left\{\arg\max_{z^{(i)}} p_\psi(z^{(i)}|h_t, a_t^g) \neq \arg\max_{z^{(i)}} p_\phi(z^{(i)}|o_t)\right\} \tag{37}$$

$$a_t^g \sim U(a^g) \tag{38}$$

To more accurately measure consistency, we consider only the *post-exploration* error rate, by evaluating the mean error rate for $t > T_e$, where T_e is approximately how many time steps required for the agent to explore the whole environment. As we aim to evaluate consistency, we want to assess world model error rate when the full context required to accurately generate future observations is contained in the observation-action history. Therefore we only want to assess error rate after the world model has likely observed the full environment.

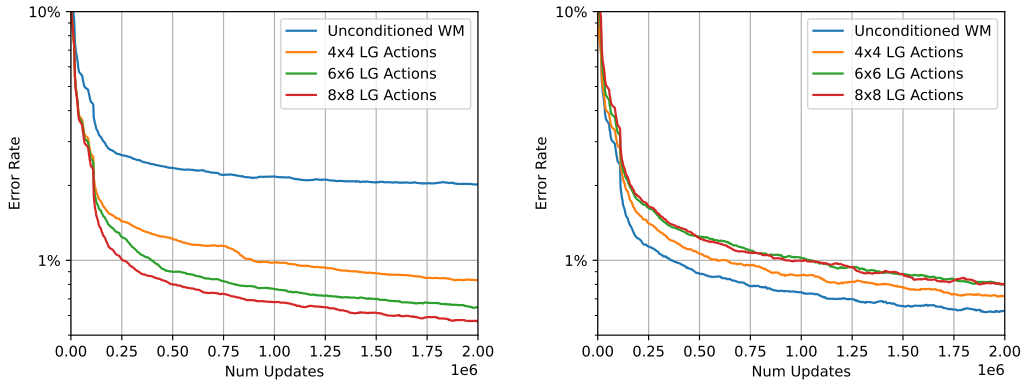
We include training plots, showing world model performance when the latent action noise and latent action size are varied, shown in Figures 18 and 19 respectively. We also include comparisons between the unconditioned, conditioned and selector models in Figure 20.



(a) True-Action Error Rate

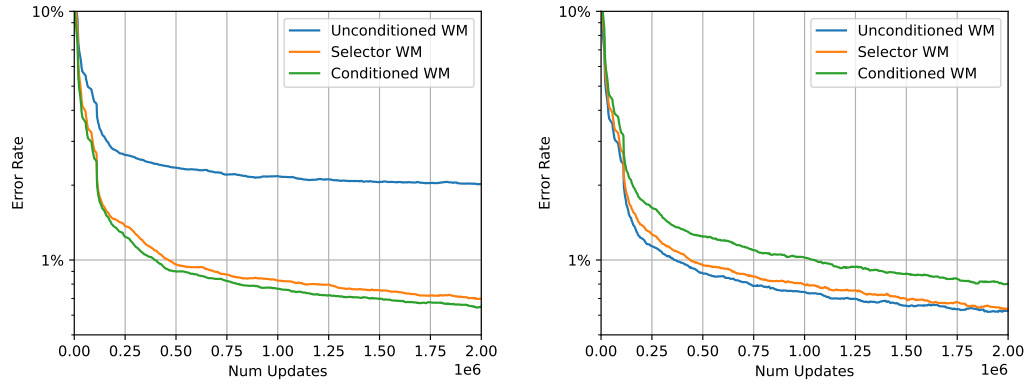
(b) Post-Exploration Error Rate

Figure 18: Comparison of True-Action Error Rate and Post-Exploration Error Rate as the latent generation action noise mixing parameter ϵ is varied.



(a) True-Action Error Rate (b) Post-Exploration Error Rate

Figure 19: Comparison of True-Action Error Rate and Post-Exploration Error Rate as the latent generation action size is varied. For all conditioned models, $\epsilon = 0.2$.



(a) True-Action Error Rate (b) Post-Exploration Error Rate

Figure 20: Latent Error Rate and Rollout-Conditioned Error Rate comparison between the unconditioned, conditioned and selector world models. For the conditioned and selector models, $\epsilon = 0.2$ and the action space is 4×4.

Figure 18(b) clearly shows the necessity for adding noise to the latent encoder distributions. When ϵ is zero, the the post-exploration error rate is significantly higher than any other model, indicating significantly worse consistency under this paramterisation.

E Additional Experiment Details

All existing methods were trained using implementations based on JaxUED Coward et al. (2024), available at `https://github.com/DramaCow/jaxued`, and DEGen Mead et al. (2025), available at `https://github.com/HarryMJMead/Dynamic-Environment-Generation-for-UED`.

E.1 Hyperparameters

We use Mead et al. (2025) as the basis for our hyperparameters.

Table 7: UED Hyperparameters.

Parameter	Key Minigrid	Minigrid	Sokoban
Student PPO			
Number of Updates	10000	30000	
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		
Anneal LR	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		
Adversary PPO			
γ	0.998		0.995
λ_{GAE}	0.95		
PPO epochs	4		
PPO minibatches per epoch	4		
PPO clip range	0.2		
Adam learning rate	1e-3		5e-4
Anneal LR	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		
PLR			
Replay rate, p	0.5		
Buffer size, K	8000		
Prioritisation	Rank		
Temperature, β	1.0		
staleness coefficient	0.3		

E.2 Compute Details

For all experiments, each run was on 1 Nvidia A40, except when marked with †, where a 4x Nvidia H100 node was used. We show the mean compute time each UED method in Table 8. As the UED runs varied in number of update steps, the UED compute times are given per 10,000 updates. We show the compute times of the world models in Table 9.

Table 8: UED Compute Time.

Method	**Compute Time (hh:mm)**
WM - DyMGen	15:14 ± 00:01
WM - Random	12:39 ± 00:01
Env - DEGen	08:18 ± 00:01
Env - DR	04:02 ± 00:00

Table 9: World Model Compute Time.

Method	**Compute Time (hh:mm)**
Observation Model	10:36
Unconditioned World Model	32:27
Conditioned World Model	32:26
Selector World Model	34:33
Selector World Model (Sokoban) †	19:45