ENTER THE VOID: EXPLORING WITH HIGH ENTROPY PLANS

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-based reinforcement learning (MBRL) offers an intuitive way to increase the sample efficiency of model-free RL methods by simultaneously training a world model that learns to predict the future. These models constitute the large majority of training compute and time and they are subsequently used to train actors entirely in simulation, but once this is done they are quickly discarded. We show in this work that utilising these models at inference time can not only boost performance but also sample efficiency. We propose a novel approach that anticipates and actively seeks out high-entropy states using the world model's short-horizon latent predictions, offering a principled alternative to traditional curiosity-driven methods that chase once-novel states well after they were stumbled into. While many model predictive control (MPC) based methods offer similar alternatives, they typically lack commitment, synthesising multiple multi-step plans at every step. To mitigate this, we present a hierarchical planner that dynamically decides when to replan, planning horizon length, and the commitment to searching entropy. While our method can theoretically be applied to any model that trains its own actors with solely model generated data, we have applied it to Dreamer to illustrate the concept. Our method finishes Miniworld's procedurally generated mazes 50% faster than base Dreamer at convergence and in only 60% of the environment steps that base Dreamer's policy needs; it displays reasoned exploratory behaviour in Crafter, achieves the same reward as base Dreamer in a third of the steps; planning is shown to accelerate and improve even Deepmind Control performance.

1 INTRODUCTION

In recent years, reinforcement learning (RL) has achieved remarkable success across a variety of domains, from mastering Go (Silver et al., 2017b) to racing drones at high speed (Kaufmann et al., 2023). However, these successes often rely on dense reward signals and highly structured environments. In real-world applications such as autonomous navigation, exploration, and disaster response, rewards are sparse and environments are stochastic and partially observable. In these conditions, achieving efficient exploration and good sample efficiency remain an active research problem. Curiosity-based bonuses tend to pursue aleatoric novelty, whereas MPC-style planners replan myopically at each step, offering little commitment at substantial computational cost.

Another avenue to address these challenges is model-based reinforcement learning (MBRL), where models of the environment (world models (Ha & Schmidhuber, 2018)) are concurrently trained to predict transitions. In this work, we propose to augment Dreamer (Hafner et al., 2020), a prominent and efficient world model, with a planner to anticipate and seek informative states *as soon as they are about to occur* to drive reasoned exploration. To do this, we leverage the world model by combining it with the greedy actor to generate a selection of high scoring rollouts, from which we choose the rollout whose predicted states have the highest entropy. We also introduce a lightweight Proximal Policy Optimisation (PPO) based hierarchical planner that dynamically decides when to commit to a pre-selected rollout and when to discard it to replan. While our experiments focus on Dreamer, our method is model-agnostic and can be applied to any MBRL framework; the mathematical formulation of our method places no such restrictions.

Our contributions are:

- Recast world-model training as a min–max objective that couples model learning with entropy-seeking exploration, improving information gain and sample efficiency.
- Use short-horizon latent predictions at inference time to proactively target high-entropy states, yielding active and reasoned exploration.
- Introducing a reactive hierarchical planner that dynamically selects between committing to a plan and replanning based on new information received, making the method more efficient, committal, and decisive.

The rest of the paper is structured as follows. In Section 2, we review related work in intrinsic motivation, planning, and hierarchical RL. In Section 3, we provide background on Dreamer and its training formulation. Section 4 introduces our entropy-seeking planner and reactive hierarchical policy. Section 5 details our experimental setup, evaluates performance on Miniworld's procedurally generated 3D maze environment, Crafter, and DMC's vision based control environment. Section 6 concludes this work and outlines limitations.

2 Related Work

2.1 Intrinsic Reward

Intrinsic motivation methods can be grouped into *retrospective* and *anticipatory* approaches. Retrospective methods assign reward after experience has occurred, using prediction error (Pathak et al., 2017), state novelty (Burda et al., 2018), episodic novelty (Badia et al., 2020), or representation surprise (Raileanu & Rocktäschel, 2020). While simple and broadly compatible with model-free RL, they are vulnerable to the white-noise problem (attraction to aleatoric uncertainty) and detachment (Burda et al., 2018; Ecoffet et al., 2019). Anticipatory methods instead steer agents toward potentially novel states using short-horizon predictions of epistemic uncertainty (Shyam et al., 2019; Sekar et al., 2020; Chua et al., 2018). When combined with reward conditioning they better avoid aleatoric traps, but typically introduce multi-step planning components that increase architectural and computational complexity.

2.2 Planning

Planning in RL ranges from tree search to trajectory optimization. Monte Carlo Tree Search (MCTS) has proved effective in games (Coulom, 2006; Silver et al., 2016; 2017a) but assumes (near) full observability and discrete action spaces, limiting its applicability in stochastic, partially observed environments. Path-integral / MPPI-style control samples and reweights trajectories under learned dynamics (Gómez et al., 2014; Williams et al., 2015); TD-MPC and TD-MPC2 pair this with TD learning for vision-based control (Hansen et al., 2022; 2023), but the planner's actions can drift from the policy network, risking distribution shift and value overestimation. Closer to our setting, Look Before You Leap prefers low-entropy, high-reward states, which can suppress exploration early in training (Wang et al., 2018); MaxEnt-Dreamer biases the actor towards toward high entropy states but where this method could have been reactive, it instead is retrospective (Svidchenko & Shpilman, 2021); RAIF optimizes posterior over prior uncertainty but this necessitates evaluating gains retrospectively, blind to emerging novelty (Nguyen et al., 2024).

2.3 Hierarchical Policies

Hierarchical RL introduces temporal abstraction via high- and low-level controllers. Option-Critic learns options and termination conditions end-to-end (Bacon et al., 2017), while HiPPO runs PPO at two temporal scales (Li et al., 2019). These methods improve long-horizon credit assignment, but fixed intervals or frequent replanning can limit adaptability; excessive termination also reduces effective commitment. Our planner differs by explicitly learning when to replan versus commit, driven by signals computed from imagined rollouts.

3 PRELIMINARIES

Dreamer (Hafner et al., 2019; 2020; 2023) learns a compact latent dynamics model and performs policy optimization entirely in latent space. In this work, we use DreamerV3 as it is the most recent and advanced formulation of the Dreamer series of models. At its core, Dreamer relies on a RSSM that factorizes the environment into deterministic recurrent states and stochastic latent representations. While our method is model agnostic, we choose to use dreamer as it is the most general and well performing method that still makes use of an GRU based model, facilitating planning. The RSSM is described by the following formulations:

$$\begin{aligned} \text{Recurrent model:} \quad & h_t = f_\phi(h_{t-1}, \mathrm{z}_{t-1}, \mathrm{a}_{t-1}) \\ \text{Transition predictor (prior):} \quad & \hat{\mathrm{z}}_t \sim p_\phi(\cdot \mid h_t) \\ \text{Representation model (posterior):} \quad & \mathrm{z}_t \sim q_\phi(\cdot \mid h_t, \mathrm{x}_t) \\ \text{Predictors (Image, Reward, Discount):} \quad & \hat{\mathrm{x}}_t, \hat{r}_t, \hat{\gamma}_t \sim p_\phi(\cdot \mid h_t, \mathrm{z}_t) \end{aligned}$$

Here, x_t is the observation at time t, and h_t is the deterministic recurrent state. At each step, Dreamer generates a prior latent state $\hat{\mathrm{z}}_t$ from the deterministic recurrent state h_t via $p_\phi(\cdot \mid h_t)$, and updates it into a posterior $q_\phi(\cdot \mid h_t, \mathrm{x}_t)$ once the new observation x_t has been received. The KL divergence between the prior and posterior is minimized to train the model:

$$\mathcal{L}_{\mathrm{KL}} = D_{\mathrm{KL}}\Big(q_\phi(\mathrm{z}_t \mid h_t, \mathrm{x}_t) \,\|\, p_\phi(\cdot \mid h_t)\Big). \tag{1}$$

Dreamer's policy network is trained using imagined trajectories generated by the world model. This ensures that policy training remains effectively on-policy. The buffer that the world model trains from is populated by a naive ϵ-greedy actor. We show in this work that filling the buffer with high-entropy transitions can lead to greater training efficiency and improved exploration.

4 METHOD

4.1 ENTROPY

DreamerV3 models latent states as factorised discrete variables. For exposition only, we analyse an equivalent case with a diagonal Gaussian prior/posterior to make the maximum-entropy argument transparent; all experiments however use DreamerV3's discrete RSSM. The same reasoning carries over by replacing differential entropy with categorical entropy of the latent logits (summing per-factor entropies) and using prior predictive entropy along short imagined rollouts. Thus the latent state in this method section is represented by:

$$\hat{z}_t \sim \mathcal{N}(\mu_t, \Sigma_t) \tag{2}$$

The conclusions derived from this will apply to the discrete case as well. We use σ_p to refer to the standard deviation of the prior's latent state, and σ_q to refer to the standard deviation of the posterior's latent state, which can also be seen as a rough uncertainty measure.

Training the model involves minimizing a KL divergence loss between the prior and posterior distributions (equation 1); the same KL divergence loss can also be interpreted as the model's information gain (IG) about the environment at the current timestep (Quinlan, 1986).

$$\mathrm{IG} = \frac{1}{2}\left(k\frac{\sigma_q^2}{\sigma_p^2} + \frac{\|\mu_p - \mu_q\|^2}{\sigma_p^2} - k + k\log\left(\frac{\sigma_p^2}{\sigma_q^2}\right)\right) \tag{3}$$

When the model has trained long enough such that the reconstruction loss and the KL loss are below a reasonable threshold, the model should set σ_p and μ_p such that any rise in $\|\mu_p - \mu_q\|$ should be counterbalanced by a proportional increase in σ_p to minimise KL loss. Conversely, it is probable that $\|\mu_p - \mu_q\|$ is high where σ_p is low.

To increase information gain, we can increase $\mu_p - \mu_q$, or the σ_q, or σ_p. If we are to use an anticipatory approach so that the model can react to novel interesting states as they emerge, the posterior of the state is not available to us. Thus, to increase information gain, we must attempt to find states where

σ_p is high, as this implies that the model expects $\mu_p - \mu_q$ to be high too. We can find these states by querying the world model to estimate prior uncertainty by generating a short horizon rollout of states.

Rather than optimising for raw standard deviation, we choose the entropy of the state distribution as it is a good descriptive statistic. The entropy of the state's distribution therefore can be taken as an estimate for the model's uncertainty, as:

$$H(state) = \frac{k}{2}\log(2\pi e) + \sum_{i=1}^{k}\log(\sigma_{p,i}) \tag{4}$$

where k is the dimensionality of the state.

Thus, the objective becomes to maximise prior entropy:

$$\mathcal{J} = \max H(state) \tag{5}$$

There are two primary failure modes for this kind of uncertainty-based exploration. The first arises in environments with high aleatoric uncertainty, where a state has multiple plausible successors due to stochasticity inherent to the environment. In these cases, even a well-understood state s_t may produce a high-entropy predictive distribution despite all potential outcomes being familiar, purely because there are multiple potential outcomes. This is due to the RSSM's unimodal predictive structure: the prior predictor must approximate a multi-modal distribution comprised of several low-variance potential states using a single Gaussian with an artificially inflated variance:

$$\hat{p}(s_{t+1} \mid s_t, a_t, h_{0:t}) \sim \mathcal{N}(\mu, \sigma^2), \quad \text{where } \sigma^2 > \sigma_i^2 \text{ for all } i$$

Here $h_{0:t}$ denotes the history of recurrent hidden states up to time t. In this case, elevated entropy arises from aleatoric uncertainty rather than epistemic uncertainty. To avoid over-exploration in these regions, we condition planning not solely on entropy but jointly on both entropy and predicted reward. This is done via using the greedy actor to generate all candidate trajectories, thus implicitly conditioning the trajectories toward reward before they are used to choose the one with the highest entropy.

The second failure mode arises in environments with latent transitions that require specific, rarely executed actions. In such cases, the ideal transition distribution remains:

$$p^*(s_{t+1} \mid s_t, a_t) = \sum_{i=1}^{n} w_i \cdot \mathcal{N}(s_{t+1} \mid \mu_i, \sigma_i^2) \tag{6}$$

Here, each mode corresponds to a distinct possible outcome, with weights w_i representing their respective likelihoods. The weight associated with the common transitions, denoted w_C, satisfies $w_C \gg w_{\not C}$, where $w_{\not C}$ is the total weight of the rare transitions. If the agent has only encountered the high-probability transitions, the learned model will be ignorant of the rare outcomes and estimate:

$$\hat{p}(s_{t+1} \mid s_t, h_{0:t}) \sim \mathcal{N}(\mu, \sigma^2), \quad \text{where } \sigma^2 \ll 1$$

In these cases, a state's uncertainty may be chronically underestimated and subsequently under-explored. This form of hidden epistemic uncertainty cannot be resolved by naïvely increasing entropy-seeking behavior globally or by enabling random exploration at all states. Addressing this likely requires mode-seeking mechanisms (option discovery, social learning, teacher-student learning). While this is outside the scope of the present method, it is possible to amplify this method with some future work.

4.2 Reactive hierarchical planner

At each environment step we generate N short-horizon imagined rollouts (we use $N{=}64$) starting from the current latent state using the greedy actor. Because proposals are actor-guided, they are already reward-seeking; among these candidates we select the plan whose latent prior has the highest cumulative entropy:

$$\hat{\tau}^\star \;=\; \arg\max_{\hat{\tau}\in\{\text{actor rollouts}\}} \;\sum_{t'=t}^{t+H-1} H_{\text{prior}}(\mathrm{z}_{t'}).$$

We then execute the selected action prefix unless the meta-policy decides to replan (below). This focuses compute on informative futures without having to trade off reward in the score itself.

We use a light PPO head to control when to commit to old plans and when to generate a new plan: The meta planner outputs a categorical over $p_t \in \{0, 0.25, 0.5, 0.75, 1\}$ which we squashed as p_t^2 to decide replanning: draw $u_t \sim \mathcal{U}(0, 1)$ and replan if $u_t < p_t^2$. We find in practice that excessive replanning is a problem that frequently plagues such planners, a finding echoed across the hierarchical RL literature (Klissarov et al., 2017; Chunduru & Precup, 2022; Johnson & Weitzenfeld, 2025) which is why we change the enaction condition from $\mathrm{u}_t < p_t$ to $\mathrm{u}_t < p_t^2$, discouraging excessive replanning without explicitly punishing replanning or rewarding commitment.

Importantly, planning decisions are re-evaluated at every environment step, allowing for flexible replanning without commitment if needed. Pseudocode for our planning algorithm is given in Appendix B.

As input to the PPO policy, we provide the encoder embedding; the RSSM feature vector; the current step number normalized by the episode time limit; the greedy action proposed by the actor; the position within the current plan (normalized); a binary in-plan flag; and the "final" RSSM feature that is predicted to be observed if the current plan is followed to the end.

We maintain replay buffers of meta-transitions with fields: ppo observation, ppo action, ppo sample log prob, implemented flag (whether a replan signal was sent), per-step entropy, next base reward, and done. We form length-L sequences and compute a shaped scalar reward for PPO:

$$r^{\text{meta}} \;=\; \tfrac{1}{2L}\Big(\underbrace{\sum_t \text{base\_reward}_t}_{\text{env return}} \;+\; \underbrace{\sum_t H_{\text{prior}}(\mathrm{z}_t)}_{\text{latent entropy}} \Big),$$

with L set to 32 for most experiments conducted here. This is because cancelling or replanning can have temporally far reaching consequences. Since the maximum rollout here is roughly 16 steps, we use double this length's rewards so that long term behaviour is adjusted for rather than short term gain. We then compute advantages with GAE(γ, λ) and optimize a clipped PPO objective (separate actor/critic) with a naive entropy bonus, using Adam for both policy and value heads. The PPO head trains on all collected transitions. To encourage early behavioral diversity we use He initialization and bias the PPO head's initial logits toward intermediate p_t values.

5 Experiments

We evaluate across three regimes that stress different aspects of decision making: procedurally generated 3D mazes in MiniWorld for long-horizon navigation under partial observability and sparse reward, Crafter (Hafner (2021)) for survival-style open-world play with diverse subgoals, and vision-based control tasks from the DeepMind Control suite (Tassa et al. (2018)) for closed-loop continuous control. We report means with variability over 5 seeds for MiniWorld [0, 409, 412, 643, 996] and DMC [0, 413, 604, 765, 891], and 3 seeds for Crafter [0, 920, 11]. Because MiniWorld and Crafter are procedurally generated, training and test distributions coincide; we therefore report training curves only. MuJoCo-based DMC tasks have no train/test split under our setup, so we also report training performance.

We compare with PLan2Explore (Sekar et al. (2020)) as a baseline; it supplies an anticipatory exploration baseline that scores novelty via ensemble disagreement, giving a contrast between disagreement-based uncertainty and our inference-time entropy signal from a single world model. We also add PPO on MiniWorld as a model-free reference (Schulman et al., 2017). For Crafter and DMC, competitive pixel-based model-free methods (augmented SAC/DrQ-style agents) are known to be sensitive to implementation and tuning (Engstrom et al., 2020; Henderson et al., 2018); to avoid confounding factors and keep compute comparable, we omit them here (Kostrikov et al., 2020; Yarats et al., 2021; Laskin et al., 2020; Srinivas et al., 2020). Dreamer is a widely adopted and strong pixel-based MBRL agent that already surpasses earlier model-based and many model-free methods in visual control (Hafner et al., 2019; 2020; 2023). Since our contribution is an inference-time planner that augments a learned world model, we instantiate it on Dreamer to illustrate benefits. Our method is however model-agnostic and could be plugged into other MBRL backbones.

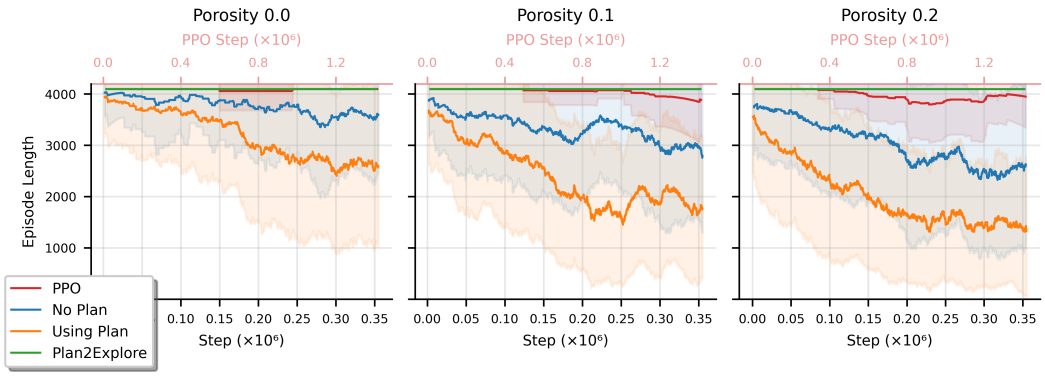


Figure 1: Episode lengths across different porosity levels. Lower porosity increases maze difficulty.

We train MiniWorld mazes for 350,000 environment steps (approximate convergence for Dreamer under our setup). On DMC and Crafter we use a fixed 24-hour wall-clock budget per method to ensure compute parity. Metrics are task-appropriate: episode time to completion for MiniWorld (shorter is better) and undiscounted training return for DMC and Crafter. All curves use a rolling mean (window 10%) with shaded ± 1 s.d. across seeds.

5.1 Maze Explorer

We use a 3D maze environment adapted from MiniWorld (Chevalier-Boisvert et al., 2023), where each episode presents a new random layout. The agent receives RGB image observations and performs continuous actions to locate three goal boxes. We introduce a *porosity* parameter that controls wall density to vary exploration difficulty. Observations are augmented with a binary spatial map that encodes visited regions and current orientation, providing a simple episodic memory. The reward function combines exploration, proximity, and goal rewards to encourage both coverage and task success. Full environment details are in Appendix C.

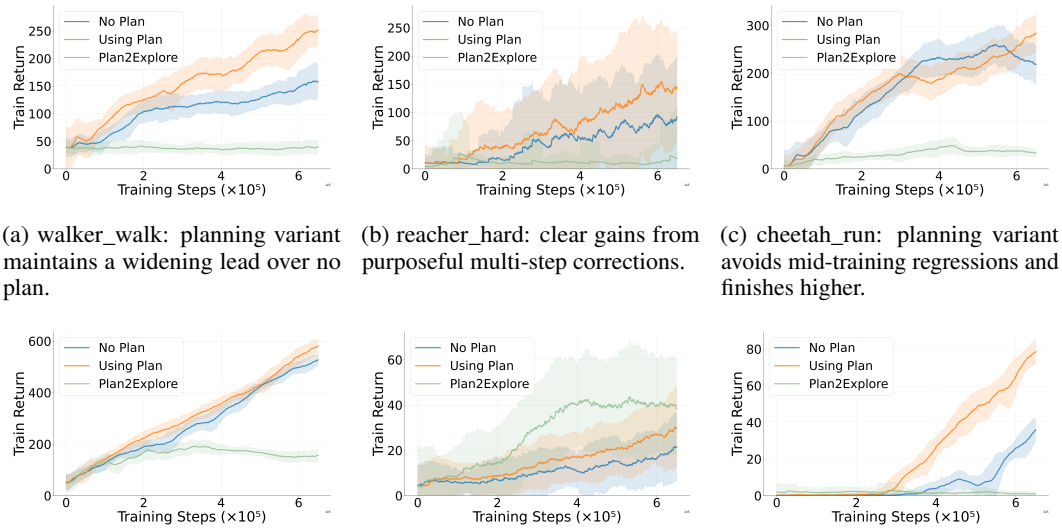
5.1.1 Task Difficulty

Varying the porosity parameter controls maze difficulty (low porosity leads to difficult mazes). Visual examples of mazes at different porosity levels are provided in Appendix D. Figure 1 shows that our method maintains low episode lengths even in denser mazes, outperforming both Dreamer and PPO. PPO underperforms across all settings, likely due to its lack of memory and long-horizon reasoning. Dreamer's performance degrades under low porosity, where rewards are harder to reach, while our method shows robust and consistent performance. Our method also exhibits lower variance, suggesting more consistent behavior across seeds.

Under the most difficult condition (porosity = 0), where only a single path exists between the agent and three goals, both our method and Dreamer perform worse. However, our approach still outperforms Dreamer, achieving 20% shorter episode lengths on average, albeit with higher variance due to the increased exploration burden. Plan2Explore underperforms here, which we hypothesize stems from frequent replanning without commitment; ensemble disagreement identifies novelty but does not enforce trajectory-level persistence.

5.2 Vision-based control (DMC-Vision)

We evaluate six pixel-control tasks from the DeepMind Control Suite: `cartpole_swingup`, `walker_walk`, `cheetah_run`, `reacher_hard`, `acrobot_swingup`, and `hopper_hop`. Rather than sweeping the entire benchmark, we select a compact set that spans complementary regimes. Agents observe 64×64 RGB frames and act in continuous spaces. For each task we train (i) a base Dreamer agent without planning ("no plan"), (ii) our commit-aware planning variant ("planning variant"), and (iii) Plan2Explore, all under identical step budgets; curves report mean ± 1 s.d. across seeds (rolling window 10%). To isolate *reasoned* exploration, we use a single environment



(a) walker_walk: planning variant maintains a widening lead over no plan.

(b) reacher_hard: clear gains from purposeful multi-step corrections.

(c) cheetah_run: planning variant avoids mid-training regressions and finishes higher.

(d) cartpole_swingup: Not much change between planning and no planning variants.

(e) acrobot_swingup: planning variant is on par or slightly better, but plan2explore greatly improves upon both here.

(f) hopper_hop: Our method starts finding the control sequences that will make the hopper hop much earlier than the no planning variant.

Figure 2: DMC-Vision learning curves (return vs. environment steps) for no-plan, the planning variant, and Plan2Explore. Shaded bands: ± 1 s.d. across seeds. The planning variant helps most when dynamics/observations induce higher variance and remains competitive elsewhere.

instance (no vectorization), since heavy parallelism can introduce "free" random exploration. For the same reason, we omit sparse-reward variants (e.g., `cartpole_swingup_sparse`), where neither base Dreamer nor our planner is expected to reliably discover narrow reward regions within the given budget.

Across four of six tasks, the planning variant improves sample efficiency and final return. Gains are largest in contact-rich or higher-variance control (Figures 2a, 2f), where short commitments reduce dithering and stabilize control under pixel noise while collecting informative trajectories. On smoother, lower-variance dynamics (Figures 2e, 2d), improvements are smaller but positive on average. Despite `hopper_hop` often benefiting from increased parallelism or longer training, our method discovers effective hopping sequences earlier than no-plan (Figure 2f). Plan2Explore underperforms on most tasks here but is strong on `acrobot_swingup`, suggesting that disagreement-based novelty aligns with that underactuated swing-up; by contrast, our approach provides broader improvements across the suite.

We did not expect an entropy-aware planner to dominate dense-control tasks; accordingly, gains are modest but consistent. A plausible mechanism is that short, commit-aware exploratory rollouts improve representation coverage and reduce vacillation, yielding small yet reliable sample-efficiency gains under pixel observations.

5.3 Open-ended survival (Crafter)

Crafter stresses long-horizon exploration and routine formation. We train for 300k environment steps (approximately 24 hours on a GeForce RTX 5090 GPU) and report means with variability over 3 seeds. The budget was chosen such that compute usage remained efficient while highlighting interesting behaviours. We run a single environment here as well (no parallel rollouts), which is the default for Crafter.

Overall, the planning variant is about 20% higher in average return and reaches comparable thresholds in roughly 50% of the steps that base Dreamer takes (Fig. 3). Gains are concentrated in routine-forming achievements such as collecting wood, placing tables, and defeating zombies where short, commit-aware exploratory rollouts appear to reduce dithering and stabilize representation learning

Figure 3: Episode returns during Crafter training.

(Figs. 4a, 4b, 4c). In contrast, deeper crafting branches (make wooden sword/pickaxe) remain low within 300k steps, especially for the planning variant (Figs. 4g, 4h). We hypothesise this is because the agent attempts these actions early game and finds they do nothing consistently (a nearby table and correct materials in the inventory are necessary to make tools), so the corresponding states become high-confidence low-reward states and are not attempted again. Even so, the planning variant remains better than or competitive with the baseline across the panel (Fig. 4).

Tasks like collect wood and place table are repeatable and reward-dense; plan commitment converts them into habits, yielding steady slopes and higher returns (Figs. 4a, 4b). Combat against zombies sits between routine and opportunistic: once wood/table routines are established, the planner's broader coverage increases encounter rate, so the zombie curve rises earlier and higher than no-plan but still exhibits spikes (Fig. 4c). Making tools remains low for both agents; the planning variant is especially conservative (Figs. 4g, 4h). It is interesting that even though we do not explicitly optimise for reward in the planner, the inherent bias toward rewarding rollouts results in what is effectively zombie and tree farming behaviour (Figs. 4a, 4c). Collect drink, collect sapling and eat cow (Figs. 4a, 4b) are all roughly matched between the no plan variant and the planning variant, indicating . Plan2Explore does not do well in Crafter either, as it requires committed exploration instaed of naive one step exploration.

5.4 Ablation Study

To isolate the contributions of individual components, we compare:

Base Dreamer: The standard agent without planning.

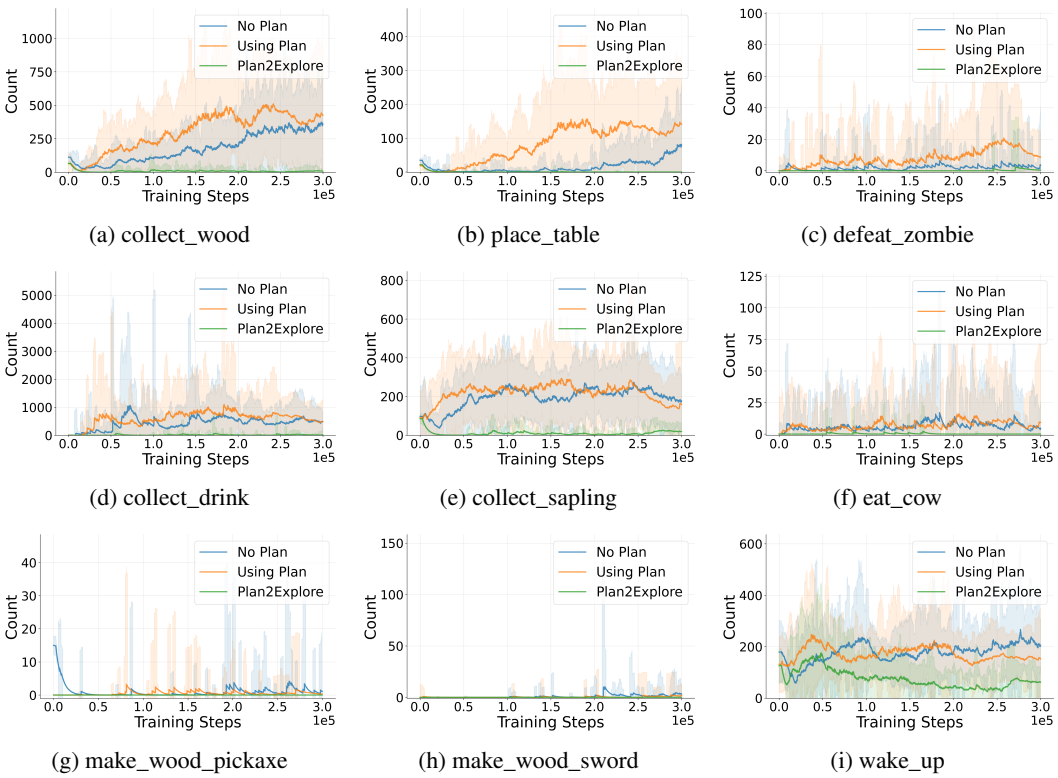
MPC Style: To evaluate the value of our underlying planner, we modify the meta planner to replan at every step, mimicking MPC behaviour.

We run all MPC ablation variants for 90% of the normal experiments' step count to account for the fact that MPC style experiments are slower to run than the full experiments. This threshold was chosen to reduce training time while still capturing meaningful differences in learning dynamics.

Figure 5a demonstrates the value of committing to plans over extended horizons. Myopic planning in a maze will result in the agent constantly choosing between different paths and never committing enough to either path to finish exploration. Myopic planning in the Crafter environment, as shown in Figure 5b, yields the same relative performance as it did in the maze environment as Crafter also benefits from committing to planned trajectories. MPC style is also not as efficient as the full planner, showing that committing to plans yields efficiency gains.

6 Conclusion

We present a robust method with a strong theoretical foundation and fast convergence to drive structured exploration in MBRL, suggesting it can generalize across domains and model architectures. This is not without limitations: inflating the KL objective in this way can lead to instabilities, so reinforcing the model is recommended. We also note that an inherent limitation of our method is that the actor must be trained purely with world model generated states rather than through experience replay, as this method biases collection of experiences towards high model entropy, leading to a distributional shift between the actor's policy and the actual behaviour policy.

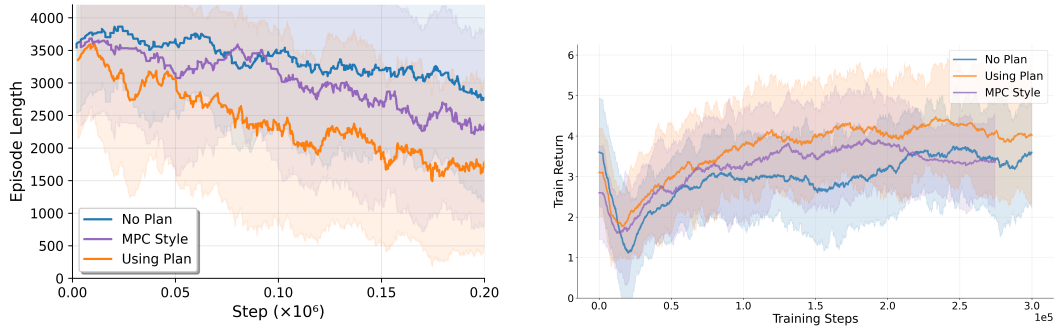


(a) collect_wood (b) place_table (c) defeat_zombie

(d) collect_drink (e) collect_sapling (f) eat_cow

(g) make_wood_pickaxe (h) make_wood_sword (i) wake_up

Figure 4: Crafter achievement counts over 300k steps (3 seeds) for no plan vs the planning variant vs plan2explore. The planning agent forms reliable routines (wood, table, zombie) and improves sample efficiency, while deeper crafting remains conservative within this budget.



(a) ObjectNav: Episode lengths during training for different ablations. The full planner outperforms both the base Dreamer and MPC-style variants, highlighting the benefit of plan commitment.

(b) Crafter: Episode returns during training for different ablations. The full planner has greater performance and takes less time than the MPC variant, signalling efficiency benefits of plan commitment.

Figure 5: Comparison of episode lengths during training for ObjectNav (left) and Crafter (right) across different ablations.

Reproducibility Statement. We document datasets, preprocessing, and step-by-step training and evaluation procedures in the Experiments section. Random seeds used for all runs are listed in the experiments section. We will release the complete codebase on GitHub upon acceptance; in the meantime, the paper and appendix provide all details needed to reimplement our results. Configuration settings are located in Appendix E.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Raviteja Chunduru and Doina Precup. Attention option-critic. *arXiv preprint arXiv:2201.02628*, 2022.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:2005.12729.

Vicenç Gómez, Hilbert J Kappen, Jan Peters, and Gerhard Neumann. Policy search for path integral control. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I 14*, pp. 482–497. Springer, 2014.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021. URL `https://arxiv.org/abs/2109.06780`.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

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

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. doi: 10.1609/aaai.v32i1.11694.

Brendon Johnson and Alfredo Weitzenfeld. Hierarchical reinforcement learning in multi-goal spatial navigation with autonomous mobile robots. *arXiv preprint arXiv:2504.18794*, 2025.

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.

Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks. *arXiv preprint arXiv:1712.00004*, 2017.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020. NeurIPS 2020 camera-ready.

Alexander C Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. *arXiv preprint arXiv:1906.05862*, 2019.

Viet Dung Nguyen, Zhizhuo Yang, Christopher L Buckley, and Alexander Ororbia. R-aif: Solving sparse-reward robotic tasks from pixels with active inference and world models. *arXiv preprint arXiv:2409.14216*, 2024.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, 2020.

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

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International conference on machine learning*, pp. 8583–8592. PMLR, 2020.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pp. 5779–5788. PMLR, 2019.

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*, 2017a.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017b.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*. PMLR, 2020. URL https://proceedings.mlr.press/v119/laskin20a.html.

Oleg Svidchenko and Aleksei Shpilman. Maximum entropy model-based reinforcement learning. *arXiv preprint arXiv:2112.01195*, 2021.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. URL https://arxiv.org/abs/1801.00690.

Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 37–53, 2018.

Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

A LLM / AI TOOLING DISCLOSURE

We used AI-assisted tools during this project as follows.

Tools.

- **Cursor (AI coding assistant):** Used to generate boilerplate code, suggest refactorings, produce docstrings and unit-test skeletons, and surface API idioms. All produced code was reviewed, modified as needed, and verified by the authors.
- **ChatGPT (writing assistant):** Used for copy-editing, grammar and style suggestions, tightening wording, expanding or condensing paragraphs on request, and clarifying phrasing. We did not use it for ideation, technical contributions, or to generate substantive claims.

The authors reviewed and verified all AI-assisted outputs for correctness and originality, and accept full responsibility for the code and text included in the paper. Any code suggested by tools was tested and adapted to our setting before inclusion.

B PLANNING ALGORITHM

Algorithm 1 Entropy Seeking Anticipatory Planning

Input: Observation o_t
Meta-policy computes discrete planning probability $p_t \in \{0, 0.05, 0.25, 0.55, 1\}$ (via squaring sampled values from $\{0, 0.25, 0.5, 0.75, 1.0\}$)
Sample $u_t \sim \mathcal{U}(0, 1)$
if $u_t < p_t$ **then**
    Greedy actor samples C (256) candidate actions $a_1, \ldots, a_C$ from o_t
    for $i = 1$ to C **do**
        Roll out trajectory τ_i of length H (maximum rollout length, 16 here) using world model and greedy actor
        Compute $E_H = \frac{1}{H} \sum_{t=1}^{H} E_t^{(i)}$
    Select trajectory $\tau_{best} = \arg\max E_H$
    Set plan to τ_{best}
Continue interacting with environment; repeat planning check at next step

C MINIWORLD ENVIRONMENT AND REWARD SCHEME

We extend the MiniWorld Maze environment(Chevalier-Boisvert et al., 2023) with several task relevant augmentations. The maze environment is a 3 dimensional procedurally generated maze of size 8x8 (using recursive backtracking) where the agent can take continuous actions along three dimensions - forward/back (step size attenuated if moving backwards to encourage progress), strafe left/right, turn left/right. Each observation consists of a forward-facing RGB image of size (64x64x3). Each episode ends when the time limit (4096) is reached, or the three goal boxes have been found. Each training run samples a new maze structure every episode to prevent memorization. No regions of the maze are sectioned off from the rest of the maze and all the goal states are reachable.

To promote structured exploration, we introduce a porosity parameter that controls wall density: with probability p, wall segments are randomly removed during generation. This provides a tunable complexity gradient for navigation tasks by creating variable maze connectivity.

An auxiliary binary 2D map of size (64x64x3) that records agent visitation over the course of an episode has been concatenated to the observation. This map records visited coordinates as 1s whereas unvisited coordinates are kept at 0. The position of the agent and the direction it is looking in is also visible on the map. This serves as episodic spatial memory that enables agents to reason about coverage and connect their actions to the current observation. This mirrors plausible real-world capabilities that can be enacted through GPS tracking or odometry.

The reward function consists of three components:

Exploration Reward: A positive reward is granted when the agent visits a previously unvisited cell in its binary exploration map. The reward magnitude is proportional to the number of newly visited cells within a square region around the agent, the size of which is controlled by the *blur* parameter, given by b. While this reward introduces non-Markovian dynamics by incorporating visitation history, the inclusion of a binary map in the observation allows memory-less model-free agents such as PPO to perform effectively in this environment.

$$\text{exploration reward} = \begin{cases} \frac{\Delta_t}{b^2} & \text{if } b > 1 \\ \Delta_t & \text{otherwise} \end{cases}$$

$$\text{where } \Delta_t = \text{number of newly explored cells at time } t$$

Proximity Reward: A smoothly decaying signal is emitted by each goal object, with exponentially scaled rewards given when the agent is within an x-unit radius. This mimics real-world analogs such as bluetooth signals or radio signals for search and rescue, animal noises for ecological monitoring, or semantic hints for more advanced exploration. This reward takes the form of two bars in the center of the image - if the agent is near a goal box of a particular colour (red, green, or blue), the bars will turn that colour with intensity varying with distance.

$$\text{Proximity reward} = \begin{cases} 0 & \text{if } \Delta < 0 \text{ or } \Delta > 10 \\ (10 - \Delta)^2 \cdot p_{mul} & \text{otherwise} \end{cases}$$

$$\text{where } \Delta = \text{dist} - (r_{\text{agent}} + r_{\text{box}} + s) \text{ and } p_{mul} = 0.03$$

Goal Reward: The agent gets a reward for moving into a coloured box. It gets 50 per box and then 150 when it gets the third box.

Thus the overall reward is composed of these three elements summed onto the baseline of -10. The lower limit of reward gained in an episode is $-T$ where T is the time limit, and the upper limit is 0.

D Maze Images

To visualize the effect of varying porosity on maze complexity, we provide top-down views of generated mazes at increasing porosity levels, see Fig. 6. As porosity increases, more internal walls are removed, resulting in more open environments. These top-down maps reflect the structural differences that influence planning difficulty.

To contextualize the agent's perspective within these mazes, we also provide an example of the full map layout and a corresponding visual observation seen by the agent, as given in Figure 7.

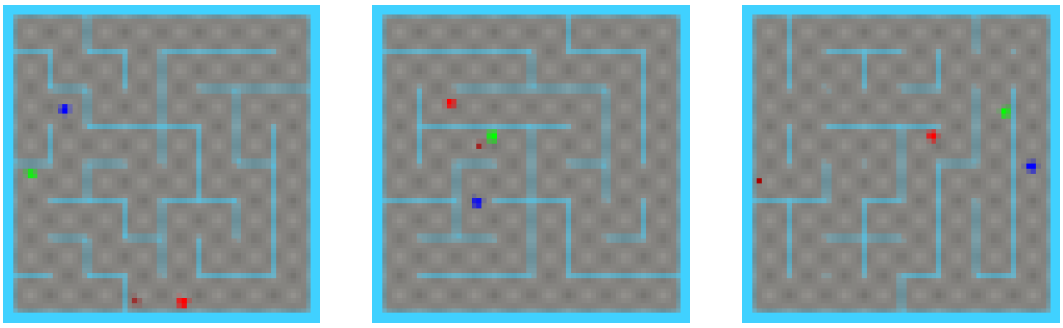
E Default Configuration and Code Base

E.1 Default Configuration

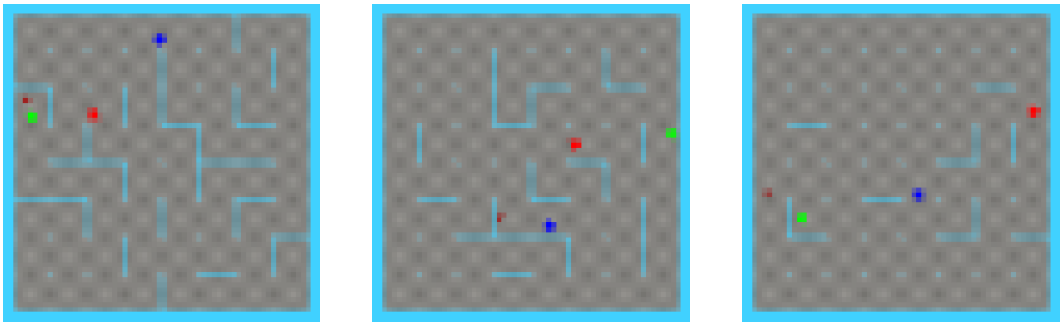
The following listing provides the default hyperparameters and settings used in our experiments.

```
use_plan: True

logdir: null
traindir: null
evaldir: null
offline_traindir: ''
offline_evaldir: ''
seed: 0
deterministic_run: False
steps: 1e6
parallel: False
```

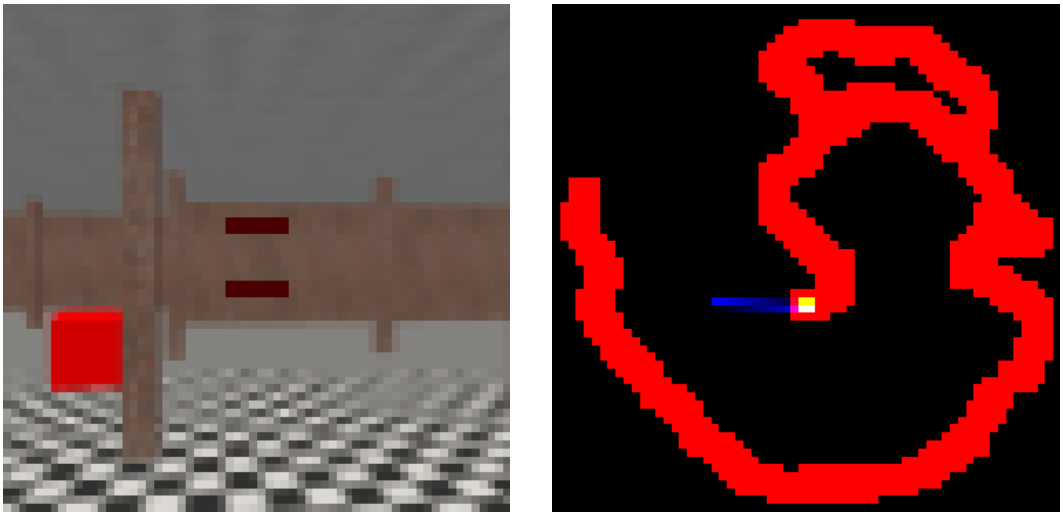


(a) Porosity 0.0 (b) Porosity 0.1 (c) Porosity 0.2



(d) Porosity 0.4 (e) Porosity 0.6 (f) Porosity 0.8

Figure 6: Top-down maze layouts at selected porosity levels. Higher porosity values remove more internal walls, increasing openness and reducing planning difficulty.



(a) Agent's visual observation (b) Agent's map observation

Figure 7: Image of what the agent perceives - the visual observation (left) and of the map observation (right).

```
eval_every: 1e4
eval_episode_num: 10
log_every: 1e4
reset_every: 0
device: 'cuda:0'
compile: True
precision: 16
```

```
debug: False

# Environment
task: 'dmc_walker_walk'
size: [64, 64]
# envs: 1
# action_repeat: 1
time_limit: 1000
grayscale: False
prefill: 2500
reward_EMA: True

# Model
dyn_hidden: 512
dyn_deter: 512
dyn_stoch: 32
dyn_discrete: 32
dyn_rec_depth: 2
dyn_mean_act: 'none'
dyn_std_act: 'sigmoid2'
dyn_min_std: 0.1
grad_heads: ['decoder', 'reward', 'cont', 'entropy']
units: 512
act: 'SiLU'
norm: True
encoder:
  {mlp_keys: '$^', cnn_keys: 'image', act: 'SiLU', norm: True, cnn_depth: 64, kernel_size: 4,
decoder:
  {mlp_keys: '$^', cnn_keys: 'image', act: 'SiLU', norm: True, cnn_depth: 32, kernel_size: 4,
actor:
  {layers: 2, dist: 'normal', entropy: 3e-4, unimix_ratio: 0.01, std: 'learned', min_std: 0.1,
Q:
  {layers: 2, dist: 'symlog_disc', slow_target: True, slow_target_update: 1, slow_target_fract
critic:
  {layers: 2, dist: 'symlog_disc', slow_target: True, slow_target_update: 1, slow_target_fract
reward_head:
  {layers: 2, dist: 'symlog_disc', loss_scale: 1.0, outscale: 1.0} #CHANGE
entropy_head:
  {layers: 2, dist: 'symlog_disc', loss_scale: 1.0, outscale: 1.0} #CHANGE
cont_head:
  {layers: 2, loss_scale: 1.0, outscale: 1.0}
dyn_scale: 0.5
rep_scale: 0.1
kl_free: 1.0
weight_decay: 0.0
unimix_ratio: 0.01
initial: 'learned'

# Training
batch_size: 16
batch_length: 64
train_ratio: 512
pretrain: 100
model_lr: 1e-4
opt_eps: 1e-8
grad_clip: 1000
dataset_size: 1000000
opt: 'adam'

# Behavior.
discount: 0.997
discount_lambda: 0.95
imag_horizon: 15
imag_gradient: 'dynamics'
imag_gradient_mix: 0.0
```

```
eval_state_mean: False

# Exploration
expl_behavior: 'greedy'
expl_until: 0
expl_extr_scale: 0.0
expl_intr_scale: 1.0
disag_target: 'stoch'
disag_log: True
disag_models: 10
disag_offset: 1
disag_layers: 4
disag_units: 400
disag_action_cond: False

# plan_behavior:
plan_max_horizon: 16
plan_choices: 256
plan_train_every: 32
sub_batch_size: 64
num_epochs: 30
buffer_size: 32768
clip_epsilon: 0.2
gamma: 0.99
lmbda: 0.95
entropy_eps: 0.1
num_cells: 256
lr: 0.003
seq_length: 8
buffer_minimum: 512
meta_action_quant: 5                    # used in CategoricalSpec
num_meta_action_lwr: 2                    # used in CategoricalSpec
ent_multiplier: 1.0                     # multiplier for entropy in _flow method
rew_multiplier: 1.0                   # multiplier for reward in _flow method

dmc_vision:
  steps: 1e6
  action_repeat: 2
  envs: 1
  train_ratio: 512
  video_pred_log: false
  encoder: {mlp_keys: '$^', cnn_keys: 'image'}
  decoder: {mlp_keys: '$^', cnn_keys: 'image'}

crafter:
  task: crafter_reward
  step: 1e6
  action_repeat: 1
  envs: 1
  train_ratio: 512
  video_pred_log: false
  dyn_hidden: 1024
  dyn_deter: 4096
  units: 1024
  encoder: {mlp_keys: '$^', cnn_keys: 'image', cnn_depth: 96, mlp_layers: 5, mlp_units: 1024}
  decoder: {mlp_keys: '$^', cnn_keys: 'image', cnn_depth: 96, mlp_layers: 5, mlp_units: 1024}
  actor: {layers: 5, dist: 'onehot', std: 'none'}
  value: {layers: 5}
  reward_head: {layers: 5}
  cont_head: {layers: 5}
  imag_gradient: 'reinforce'
```

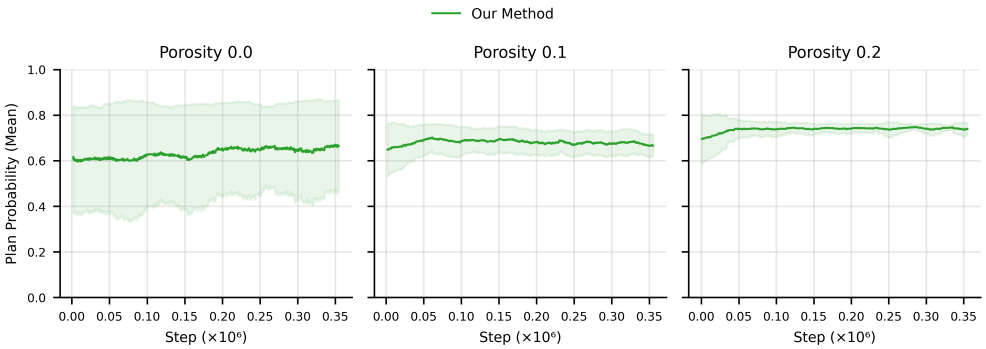


Figure 8: Replanning probabilities across porosities. While mean values remain comparable, variance across seeds diminishes faster at higher porosities.

E.2 Code Base

Our implementation is forked from `https://github.com/NM512/dreamerv3-torch/blob/main/dreamer.py`. We adapt this code to our setting while retaining the default configuration listed above.

F Other Results

In this section, we present additional metrics and quantities tracked during our experiments. While these results are not central to the main text, they provide further insight into the model's internal behavior and performance dynamics.

From the porosity comparison study, we highlight the internal planning parameters set by the meta-policy. Specifically:

- The probability of replanning is shown in Figure 8, with its intra-episode standard deviation in Figure 9.

Shaded intervals in all graphs represent variation across different random seeds, while intra-episode standard deviations are shown in dedicated plots.

We explore the replanning probability, which captures how often the agent updates its plan mid-episode. Figure 8 shows the average replanning probabilities over training for each porosity level. While the means are broadly similar, a clear reduction in variability across seeds is observed at higher porosities, reflecting more deterministic behavior under lower uncertainty. Figure 9 complements this by plotting the intra-episode standard deviation of the replanning probability. Here, we observe a steady decline over training, indicating growing confidence in the agent's planning under all conditions.

Figure 10 shows KL divergence values between the prior and posterior. Higher KL indicates that the world model is encountering states it cannot yet predict, which can reflect either model failure or valuable learning. Given that our method leads to better downstream performance, we interpret this as a sign of active, informed exploration.

Figure 11 shows that our method maintains approximately 10% higher prior entropy across training, indicating greater predictive uncertainty of the world model and a broader exploration strategy.

Figure 12 presents the actor loss across different porosities, indicating that our method does not significantly affect the actor component itself, but instead the gains in performance are driven by directly optimising the world model. Interestingly, a divergence between the planned and unplanned variants becomes apparent toward the end of training in the 0.2 porosity setting. This may suggest that, as the world model converges and its predictive uncertainty decreases, the actor also stabilizes and its loss declines. Additionally, the standard deviation of the actor loss is notably lower for the

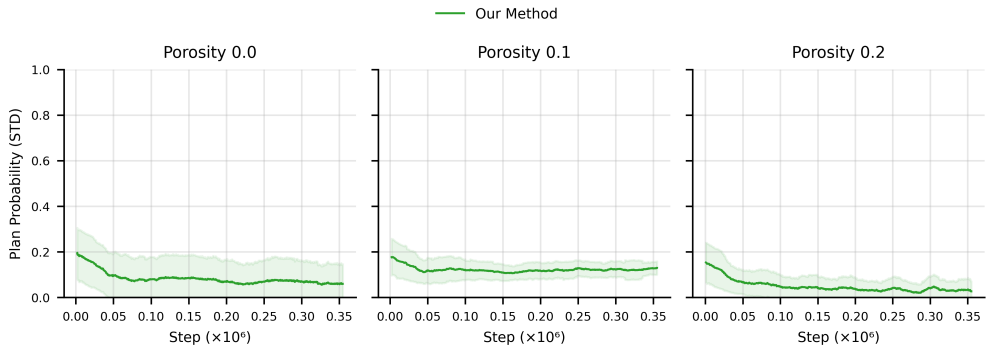


Figure 9: Intra-episode standard deviations of replanning probabilities. These deviations drop quickly as meta-policies converge.

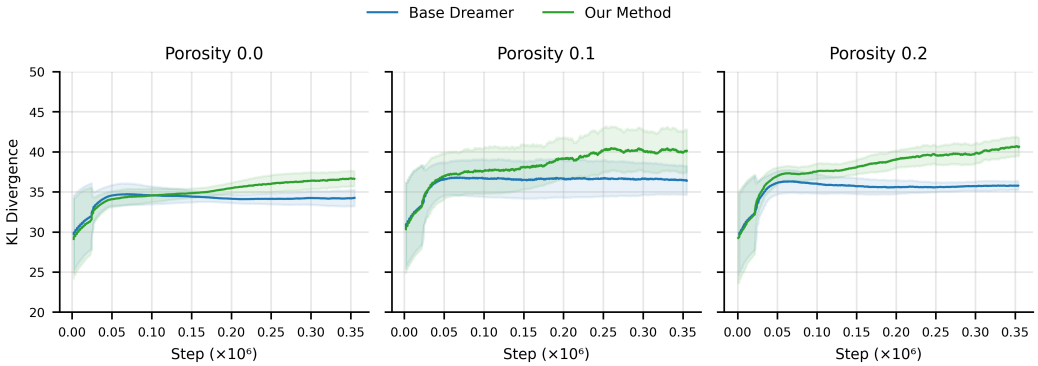


Figure 10: KL divergence across porosity levels. Higher porosity results in a more marked difference between our method and base Dreamer.

planned agent in higher-porosity environments, which is likely a consequence of reduced epistemic uncertainty in the world model.

Figure 13 presents the length of a plan before replanning occurs. This measure is generally stable across porosities, but the variance across seeds is higher in more difficult (low porosity) settings. Interestingly, as shown in Figure 14, the intra-episode standard deviation remains low, suggesting that while different seeds may converge to distinct stable values, intra-run variability remains small. The variation between seeds in Figure 14 is higher in the hardest setting, as noted in previous plots. This could be because of the world model not reaching convergence as easily.

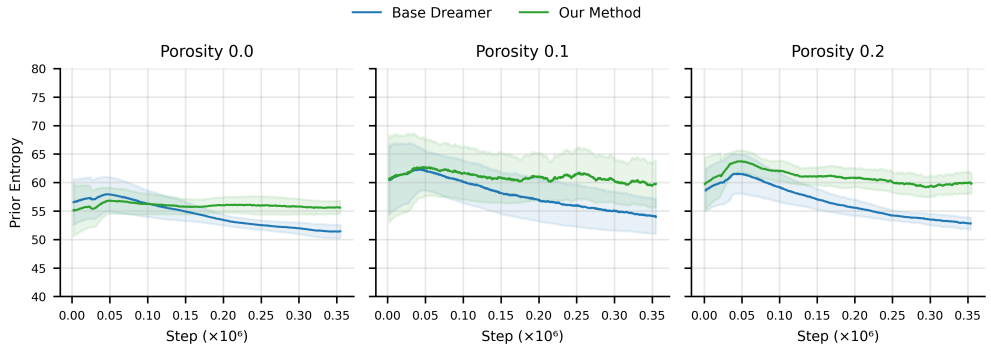


Figure 11: Prior entropies through different porosities, reflecting the model's estimated uncertainty.

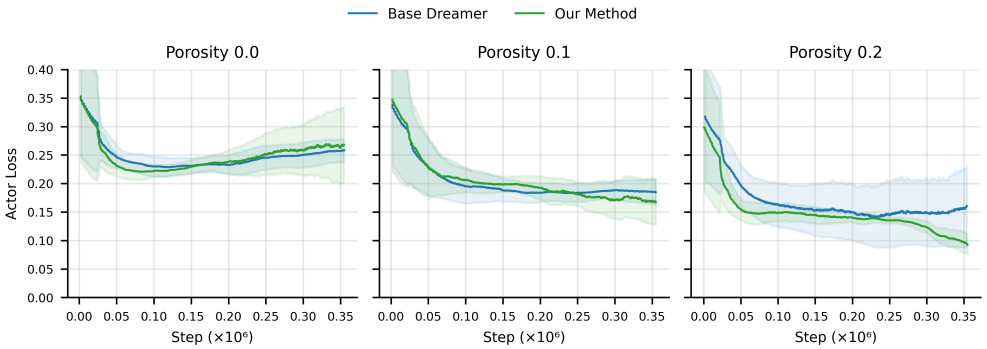


Figure 12: Actor loss across porosities. The actor remains stable across planning variants, with some divergence occurring later in training for 0.2 porosity.

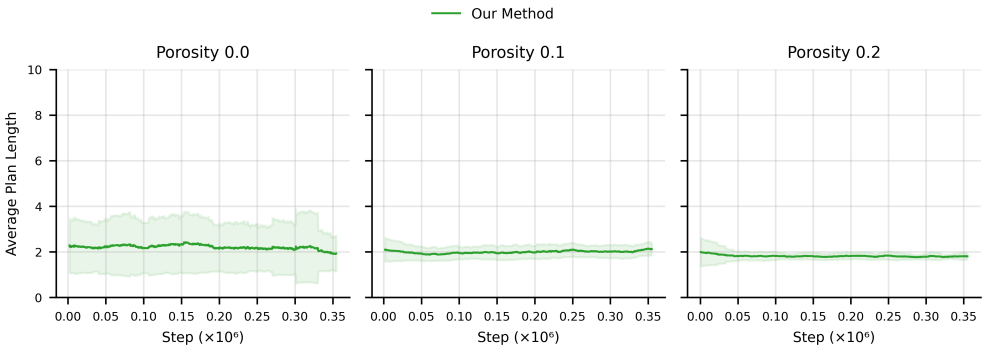


Figure 13: Length of plan before replanning across porosities. Stability is observed overall, with higher inter-seed variation in low-porosity settings.

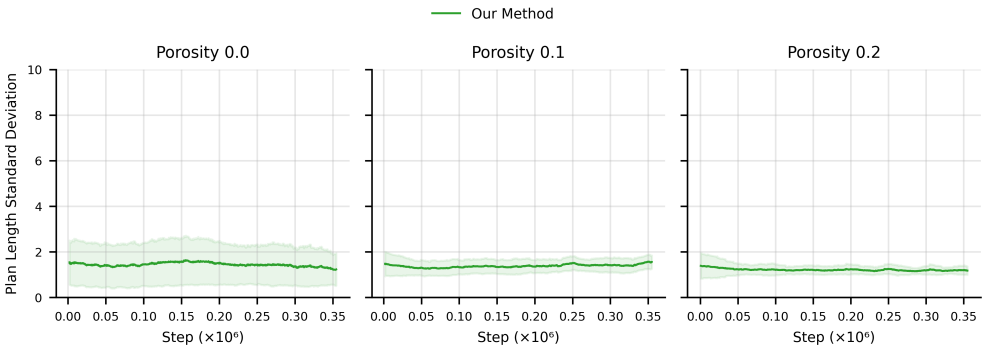


Figure 14: Intra-episode standard deviation of plan length before replanning. Values remain low across porosities, suggesting consistent behavior within each run.