Planning from Pixels in Environments with Combinatorially Hard Search Spaces

Anonymous Authors[1] ,

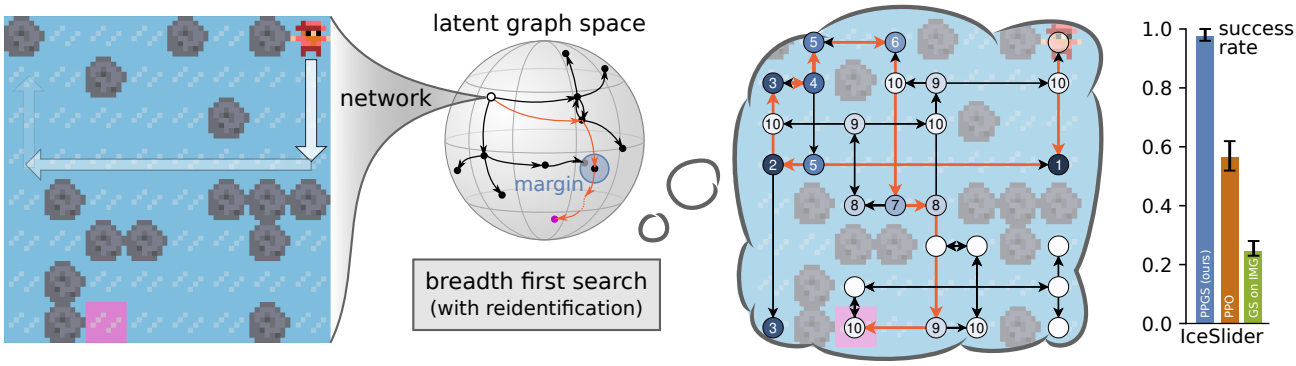


Figure 1: Planning from Pixels with Graph Search. Our method leverages learned latent dynamics to efficiently build and search a graph representation of the environment. Resulting policies show unrivaled performance across a distribution of hard combinatorial tasks.

Abstract

The ability to form complex plans based on raw visual input is a litmus test for current capabilities of artificial intelligence, as it requires a seamless combination of visual processing and abstract algorithmic execution, two traditionally separate areas of computer science. A recent surge of interest in this field brought advances that yield good performance in tasks ranging from arcade games to continuous control; these methods however do not come without significant issues, such as limited generalization capabilities and difficulties when dealing with combinatorially hard planning instances. Our contribution is two-fold: (i) we present a method that learns to represent its environment as a latent graph and leverages state reidentification to reduce the complexity of finding a good policy from exponential to linear (ii) we introduce a set of lightweight environments with an underlying discrete combinatorial structure in which planning is challenging even for humans. Moreover, we show that our methods achieves strong empirical generalization to variations in the environment, even across highly disadvantaged regimes, such as "one-shot" planning, or in an offline RL paradigm which only provides low-quality trajectories.

1. Introduction

Decision problems with an underlying combinatorial structure pose a significant challenge for a learning agent, as they require both the ability to infer the true low-dimensional state of the environment and the application of abstract reasoning to master it. A traditional approach for common logic games, given that a simulator or a model of the game are available, consists in applying a graph search algorithm to the state diagram, effectively simulating several trajectories to find the optimal one. As long as the state space of the game grows at a polynomial rate with respect to the planning horizon, the solver is able to efficiently find the optimal solution to the problem. Of course, when this is not the case, heuristics can be introduced at the expense of optimality of solutions.

Learned world models (Ha and Schmidhuber, 2018; Hafner et al., 2019) can learn to map complex observations to a lower-dimensional latent space and retrieve an approximate simulator of an environment. However, while the continuous structure of the latent space is suitable for training reinforce-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

ment learning agents (Hafner et al., 2020; Chua et al., 2018) or applying heuristic search algorithms (Schrittwieser et al., 2020), it also prevents a straightforward application of simpler graph search techniques that rely on identifying and marking visited states.

Our work follows naturally from the following insight: a simple graph search might be sufficient for solving visually complex environments, as long as a world model is trained to realize a suitable structure in the latent space. Moreover, the complexity of the search can be reduced from exponential to linear by reidentifying visited latent states.

The method we propose is located at the intersection between classical planning, representation learning and model-based reinforcement learning. It relies on a novel low-dimensional world model trained through a combination of opposing losses without reconstructing observations. We show how learned latent representations allow a dynamics model to be trained to high accuracy, and how the dynamics model can then be used to reconstruct a *latent graph* representing environment states as vertices and transitions as edges. The resulting latent space structure enables powerful graph search algorithms to be deployed for planning with minimal modifications, solving challenging combinatorial environments from pixels. We name our method **PPGS** as it **P**lans from **P**ixels through **G**raph **S**earch.

We design PPGS to be capable of generalizing to unseen variations of the environment, or equivalently across a distribution of *levels* (Cobbe et al., 2020). This is in contrast with traditional benchmarks (Bellemare et al., 2013), which require the agent to be trained and tested on the same fixed environment.

We can describe the main contributions of this paper as follows: first, we introduce a suite of environments that highlights a weakness of modern reinforcement learning approaches, second, we introduce a simple but principled world model architecture that can accurately learn the latent dynamics of a complex system from high dimensional observations; third, we show how a planning module can simultaneously estimate the latent graph for previously unseen environments and deploy a breadth first search in the latent space to retrieve a competitive policy; fourth, we show how combining our insights leads to unrivaled performance and generalization on a challenging class of environments.

2. Method

For the purpose of this paper, each environment can be modeled as a family of fully-observable deterministic goal-conditioned Markov Decision Processes with discrete actions, that is the 6-tuples $\{S_i, A_i, T_i, , G_i, R_i, \gamma_i\}_{1...n}$ where S_i is the state set, A_i is the action set, T_i is a transition function $T_i : S_i \times A_i \rightarrow S_i$, G_i is the goal set and R_i is

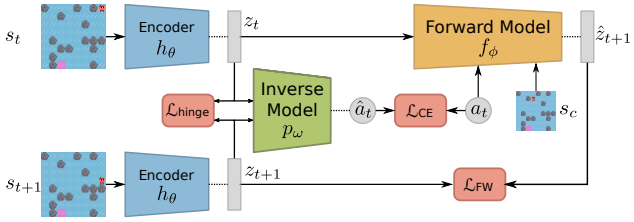


Figure 2: Architecture of the world model. A convolutional encoder extracts latent state representations from observations, while a forward model and an inverse model reconstruct latent dynamics by predicting state transitions and actions that cause them. The notation is introduced in Sec. 2.1

a reward function $R_i : S_i \times G_i \rightarrow \mathbb{R}$ and γ_i is the discount factor. In particular, we deal with families of procedurally generated environments. We refer to each of the n elements of a family as a *level* and omit the index i when dealing with a generic level.

In our work the reward simplifies to an indicator function for goal achievement $R(s, g) = \mathbf{1}_{s=g}$ with $G \subseteq S$. Given a goal distribution $p(g)$, the objective is that of finding a goal-conditioned policy π_g that maximizes the return

$$\mathcal{J}_\pi = \mathop{\mathbb{E}}_{g \sim p(g)} \left[\mathop{\mathbb{E}}_{\tau \sim p(\tau|\pi_g)} \sum_t \gamma^t R(s_t, g) \right] \tag{1}$$

where $\tau \sim p(\tau|\pi_g)$ is a trajectory $(s_t, a_t)_{t=1}^T$ sampled from the policy.

Our environments of interest should challenge both perceptive and reasoning capabilities of an agent. In principle, they should be solvable through extensive search in hard combinatorial spaces. In order to master them, an agent should therefore be able to (i) identify pairs of bisimilar states (Zhang et al., 2021), (ii) keep track of and reidentify states it has visited in the past and (iii) produce highly accurate predictions for non-trivial time horizons. These factors contribute to making such environments very challenging for existing methods. Our method is designed in light of these necessities; it has two integral parts, the world model and the planner, which we now introduce.

2.1. World Model

The world model relies solely on three jointly trained function approximators: an encoder, a forward model and an inverse model. Their overall orchestration is depicted in Fig. 2 and described in the following.

2.1.1. ENCODER

Mapping highly redundant observations from an environment to a low-dimensional state space Z has several benefits (Ha and Schmidhuber, 2018; Hafner et al., 2019). Ideally, the projection should extract the compressed "true state" of the environment and ignore irrelevant visual cues, discarding all information that is useless for planning. For this purpose, our method relies on an *encoder* h_θ, that is a neural function approximator mapping each observed state $s \in S$ and a low-dimensional representation $z \in Z$ (*embedding*). While there are many suitable choices for the structure of the latent space Z, we choose to map observations to points on an d-dimensional hypersphere taking inspiration from Liu et al. (2017).

2.1.2. FORWARD MODEL

In order to plan ahead in the environment, it is crucial for an agent to estimate the transition function T. In fact, if a mapping to a low-dimensional latent space Z is available, learning directly the projected transition function $T_Z : Z \times A \to Z$ can be largely beneficial (Ha and Schmidhuber, 2018; Hafner et al., 2019). The deterministic latent transition function T_Z can be learned by a neural function approximator f_ϕ so that if $T(s_t, a_t) = s_{t+1}$, then $f_\phi(h_\theta(s_t), a_t) := f_\phi(z_t, a_t) = h_\theta(s_{t+1})$. We refer to this component as *forward model*. Intuitively, it can be trained to retrieve the representation of the state of the MDP at time $t + 1$ given the representation of the state and the action taken at the previous time step t.

Due to the Markov property of the environment, an initial state embedding z_t and the action sequence $(a_t, \dots, a_{t+k})$ are sufficient to to predict the latent state at time $t + k$, as long as z_t successfully captures all relevant information from the observed state s_t. The amount of information to be embedded in z_t and to be retained in autoregressive predictions is, however, in most cases, prohibitive. Take for example the case of a simple maze: z_t would have to encode not only the position of the agent, but, as the predictive horizon increases, most of the structure of the maze.

Invariant Structure Recovery To allow the encoder to only focus on local information, we adopt an hybrid forward model which can recover the invariant structures in the environment from previous observations. The function that the forward model seeks to approximate can then include an additional input: $f_\phi(z_t, a_t, s_c) = z_{t+1}$, where $s_c \in S$ is a generic observation from the same environment and level. Through this context input the forward model can retrieve information that is constant across time steps (e.g. the location of walls in static mazes). In practice, we can use randomly sampled observation from the same level during training and use the latest observation during evaluation. This choice allows for more accurate and structure-aware predictions, as we show in the ablations in Suppl. A.

Given a trajectory $(s_t, a_t)_{t=1}^{T}$, the forward model can be trained to minimize some distance measure between state embeddings $(z_{t+1})_{1\dots T-1} = (h_\theta(s_{t+1}))_{1\dots T-1}$ and one-step predictions $(f_\phi(h_\theta(s_t), a_t, s_c))_{1\dots T-1}$. In practice, we choose to minimize a Monte Carlo estimate of the expected Euclidean distance over a finite time horizon, a set of trajectories and a set of levels. For a given level l drawn from the training distribution $p(l)$, we extract K trajectories of length H from each level with a uniform random policy π and we minimize

$$\mathcal{L}^l_{\text{FW}} = \frac{1}{H-1} \sum_{h=1}^{H-1} \mathop{\mathbb{E}}_{a_h \sim \pi} \Big[\| f_\phi(z^l_h, a_h, s^l_c) - z^l_{h+1} \|_2^2 \Big]. \quad (2)$$

where the superscript indicates the level from which the embeddings are extracted.

2.1.3. INVERSE MODEL AND COLLAPSE PREVENTION

Unfortunately, the loss landscape of Equation 2 presents a trivial minimum in case the encoder collapses all embeddings to a single point in the latent space. As embeddings of any pair of states could not be distinguished in this case, this is not a desirable solution. We remark that this is a known problem in metric learning and image retrieval (Bellet et al., 2013), for which solutions ranging from siamese networks (Bromley et al., 1993) to using a triplet loss (Hoffer and Ailon, 2015) have been proposed.

The context of latent world models offers a natural solution that isn't available in the general embedding problem, which consists in additionally training a probabilistic *inverse model* $p_\omega(a_t | z_t, z_{t+1})$ such that if $T_Z(z_t, a_t) = z_{t+1}$, then $p_\omega(a_t | z_t, z_{t+1}) > p_\omega(a_k | z_t, z_{t+1}) \forall a_k \neq a_t \in A$. The inverse model, parameterized by ω, can be trained to predict the action a_t that causes the latent transition between two embeddings z_t, z_{t+1} by minimizing multi-class cross entropy.

$$\mathcal{L}^l_{\text{CE}} = \frac{1}{H-1} \sum_{h=1}^{H-1} \mathop{\mathbb{E}}_{a_h \sim \pi} \Big[-\log p_\omega(a_h | z^l_h, z^l_{h+1}) \Big]. \quad (3)$$

Intuitively, $\mathcal{L}_{\text{CE}}$ increases as embeddings collapse, since it becomes harder for the inverse model to recover the actions responsible for latent transitions. For this reason, it mitigates unwanted local minima. Moreover, it is empirically observed to enforce a regular structure in the latent space that eases the training procedure, as argued in Sec. A of the Appendix. We note that this loss plays a similar role to the reconstruction loss in Hafner et al. (2019). However, $\mathcal{L}_{CE}$ does not force the encoder network to embed information that helps with reconstructing irrelevant parts of the observation, unlike training methods relying on image reconstruction (Chiappa et al., 2017; Ha and Schmidhuber, 2018; Hafner et al., 2019; 2020; 2021).

While $\mathcal{L}_{\text{CE}}$ is sufficient for preventing collapse of the latent space, a discrete structure needs to be recovered in order to deploy graph search in the latent space. In particular, it is still necessary to define a criterion to reidentify nodes during the search procedure, or to establish whether two embeddings (directly encoded from observations or imagined) represent the same true low-dimensional state.

A straightforward way to enforce this is by introducing a margin ε, representing a desirable minimum distance between embeddings of non-bisimilar states (Zhang et al., 2021). A third and final loss term can then be introduced to encourage margins in the latent space:

$$\mathcal{L}^l_{\text{margin}}=\frac{1}{H-1}\sum_{h=1}^{H-1}\max\Big(0,1-\frac{\|z^l_{h+1}-z^l_h)\|_2^2}{\varepsilon^2}\Big). \quad (4)$$

We then propose to reidentify two embeddings as representing the same true state if their Euclidean distance is less than $\frac{\varepsilon}{2}$.

Adopting a latent margin effectively constrains the number of margin-separated states that can be represented on an hyperspherical latent space. However, this quantity is lower-bounded by the kissing number (Wikipedia contributors, 2021), that is the number of non-overlapping unit-spheres that can be tightly packed around one d dimensional sphere. The kissing number grows exponentially with the dimensionality d. Thus, the capacity of our d-dimensional unit sphere latent space (d=16 in our case with margin ε=0.1) is not overly restricted.

The world model can be trained jointly and end-to-end by simply minimizing the expected value of a linear combination of the three loss functions over a training level distribution $p(l)$:

$$\mathcal{L}=\mathop{\mathbb{E}}_{l\sim p(l)}\Big[\alpha\mathcal{L}^l_{\text{FW}}+\beta\mathcal{L}^l_{\text{CE}}+\mathcal{L}^l_{\text{margin}}\Big]. \quad (5)$$

To summarize, the three components are respectively encouraging accurate dynamics predictions, regularizing latent representations and enforcing a discrete structure for state reidentification.

2.2. Planning Regimes

A deterministic environment can be represented as a directed graph G whose vertices V represent states $s\in S$ and whose edges E encode state transitions. An edge from a vertex representing a state $s\in S$ to a vertex representing a state $s'\in S$ is present if and only if $T(s,a)=s'$ for some action $a\in A$, where T is the state transition function of the environment. This edge can then be labelled by action a. Our planning module relies on reconstructing the *latent graph*, which is a projection of graph G to the latent state Z.

In this section we describe how a latent graph can be build from the predictions of the world model and efficiently searched to recover a plan, as illustrated in Fig. 3. This method can be used as a one-shot planner, which only needs access to a visual goal and the initial observation from a level. When iterated and augmented with online error correction, this procedure results in a powerful approach, which we refer to as *full planner*, or simply as PPGS.

One-shot Planner Breadth First Search (BFS) is a graph search algorithm that relies on a LIFO queue and on marking visited states to find an optimal path $O(V+E)$ steps. Its simplicity makes it an ideal candidate for solving combinatorial games by exploring their latent graph. If the number of reachable states in the environment grows polynomially, the size of the graph to search will increase at a modest rate and the method can be applied efficiently.

We propose to execute a BFS-like algorithm on the latent graph, which is recovered by autoregressively simulating all transitions from visited states. As depicted in Fig. 3, at each step, the new set of leaves L is retrieved by feeding the leaves from the previous iteration through the forward model f_ϕ. The efficiency of the search process can be improved as shown in Fig. 4, by exploiting the margin ε enforced by equation 4 to reidentify states and identify loops in the latent graph. We now provide a simplified description of the planning method in Algorithm 1, while details can be found in Suppl. C.2.

Algorithm 1 Simplified one-shot PPGS

Input: Initial observed state s_1, visual goal g

$z_1,z_g=h_\theta(s_1),h_\theta(g)$ ▷ project to latent space Z
$L,V=\{z_1\}$ ▷ sets of leaves and visited vertices
for T_{MAX} steps **do**
  $L=\{f_\phi(z,a,s_1):\exists z\in L,a\in A\}$ ▷ grow graph
  if $z^*\in L$ can be reidentified with z_g **then**
    return action sequence from z_1 to z^*
  end if
  $L=L\setminus V$ ▷ reidentify and discard visited vertices (details in Suppl. C.2)
  $V=V\cup L$ ▷ update visited vertices
end for

Full Planner The one-shot variant of PPGS largely relies on highly accurate autoregressive predictions, which a learned model cannot usually guarantee. We mitigate this issue by adopting a model predictive control-like approach (Garcia et al., 1989). PPGS recovers an initial guess on the best policy $(a_i)_{1,\dots,n}$ simply by applying one-shot PPGS as described in the previous paragraph and in Algorithm 2. It

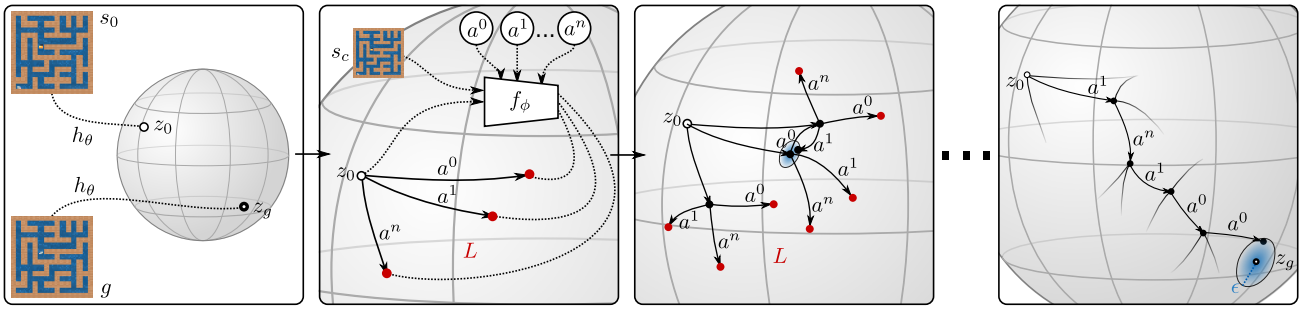


Figure 3: Overview of latent-space planning. One-shot planning is possible by (i) embedding the current observation and goal to the latent space and (ii) iteratively growing a latent graph until a vertex is reidentified with the goal.

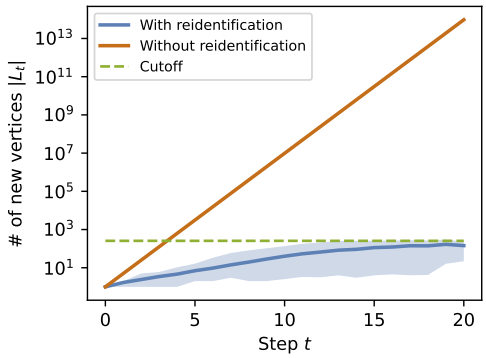


Figure 4: Number of leaf vertices when planning in ProcgenMaze, averaged over 100 levels, with 90% confidence intervals.

then applies the policy step by step and projects new observations to the latent space. When new observations do not match with the latent trajectory, the policy is recomputed by applying one-shot PPGS from the latest observation. This happens when the autoregressive prediction of the current embedding (conditioned on the action sequence since the last planning iteration) can not be reidentified with the embedding of the current observation. Moreover, the algorithm stores all observed latent transitions in a lookup table and, when replanning, it only trusts the forward model on previously unseen observation/action pairs. A detailed description can be found in Suppl. C.2.

3. Environments

In order to benchmark both perception and abstract reasoning, we empirically show the feasibility of our method on three challenging procedurally generated environments. These include the Maze environment from the procgen suite (Cobbe et al., 2019), as well as DigitJump and IceSlider, two combinatorially hard environments which stress the reasoning capabilities of a learning agent, or even of an human player. The last two environments are available as supplementary materials (Anonymous, 2021). We invite the interested reader to test them out in an interactive way on the wepsite to directly experience their complexity. More details on their implementation are included in Suppl. D.

ProcGenMaze The ProcgenMaze environment consists of a family of procedurally generated 2D mazes. The agent starts in the bottom left corner of the grid and needs to reach a position marked by a piece of cheese. For each level, an unique shortest solution exists, and its length is usually distributed roughly between 1 and 40 steps. This environment presents significant intra-level variability, with different sizes, textures, and maze structures. While retrieving the optimal solution in this environment is already a non-trivial task, its dynamics are uniform and actions only cause local changes in the observations. Moreover, ProcgenMaze is a forgiving environments in which errors can always be recovered from. We use these insights to choose the additional environments.

IceSlider IceSlider is in principle similar to ProcgenMaze, since it also consists of procedurally generated mazes. However, each action propels the agent in a direction until an obstacle (a rock or the borders of the environments) is met. We generate solvable but unforgiving levels that feature irreversible transitions, that, once taken, prevent the agent from ever reaching the goal.

DigitJump DigitJump features a distribution of randomly generated levels which consist of a 2D 8x8 grid of handwritten digits from 1 to 6. The agent needs to go from the top left corner to the bottom right corner. The 4 directional actions are available, but each of them causes the agent to move in that directions by the number of steps expressed by the digit on the starting cell. Therefore, a single action can easily transport the player across the board. This makes navigating the environment very challenging, despite the reduced cardinality of the state space. Moreover, the game presents many cells in which the agent can get irreversibly stuck.

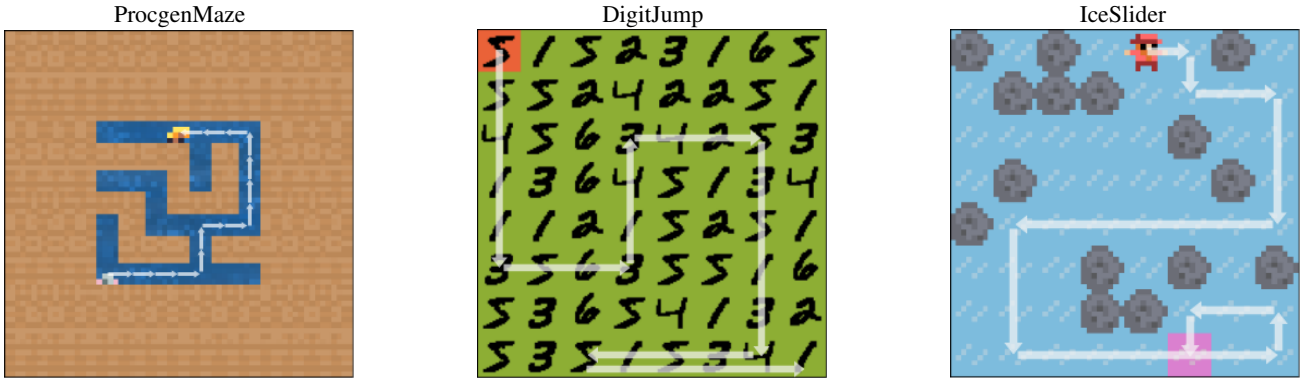


Figure 5: Environments. Initial observations and one-shot PPGS's solution (arrows) of a random level of each of the three environments. ProcgenMaze is from (Cobbe et al., 2019). DigitJump and IceSlider are proposed by us and can be accessed at (Anonymous, 2021).

4. Related Work

World Models and Reinforcement Learning The idea of learning to model an environment has been widely explored in recent years. Work by Oh et al. (2015) and Chiappa et al. (2017) has argued that modern machine learning architectures are capable of learning to model the dynamics of a generic environment reasonably well for non-trivial time horizons. The seminal work by Ha and Schmidhuber (2018) built upon this by learning a world model in a low-dimensional latent space instead of conditioning predictions on observations. They achieved this by training a VAE on reconstructing observations and a recurrent network for sampling latent trajectories conditioned on an action sequence. Moreover, they showed how sample efficiency could be addressed by recovering a simple controller acting directly on latent representations through an evolutionary approach.

This initial idea was iteratively improved along two main directions. On one hand, some subsequent works focused on learning objectives and suggested to jointly train encoding and dynamics components. Hafner et al. (2019) introduced a multi-step variational inference objective to encourage latent representations to be predictive of the future and propagate information through both deterministic and stochastic paths. On the other hand, authors proposed to learn to act in the latent space by using zero-order methods (Hafner et al., 2019) such as CEM (Rubinstein, 1997) or policy gradient techniques (Hafner et al., 2020; 2021). These improvements gradually led to strong model-based RL agents capable of achieving very competitive performance in continuous control tasks (Hafner et al., 2020) and on the Atari Learning Environment (Bellemare et al., 2013; Chen et al., 2020; Hafner et al., 2021).

Relying on image reconstruction can however lead to vulnerability to visual noise: to overcome this limitation Okada and Taniguchi (2020) and Zhang et al. (2021) forgo the decoder network, while the latter proposes to rely on the notion of bisimilarity to learn meaningful representations. Similarly, Gelada et al. (2019) only learn to predict rewards and action-conditional state distributions, but only study this task as an additional loss to model-free reinforcement learning methods. Another relevant approach is that of (Zhang et al., 2020), who propose to learn a discrete graph representation of the environment, but their final goal is that of recovering a series of subgoals for model-free RL.

A strong example of how world models can be coupled with classical planners is given by MuZero (Schrittwieser et al., 2020). MuZero trains a recurrent world model to guide a Monte Carlo tree search by encouraging hidden states to be predictive of future states and a sparse reward signal. While we adopt a similar framework, we focus on recovering a discrete structure in the latent space in order to reidentify states and lower the complexity of the search procedure. Moreover, we do not rely on reward signals, but only focus on learning the dynamics of the environment.

Neuro-algorithmic Planning In recent years, several other authors have explored the intersection between representation learning and classical algorithms. This is the case, for instance, of Kuo et al. (2018); Kumar et al. (2019); Ichter and Pavone (2019) who rely on sequence models or VAEs to propose trajectories for sampling-based planners. Within planning research, Yonetani et al. (2020) introduce a differentiable version of the A* search algorithm that can learn suitable representations from images with supervision. The most relevant line of work to us is perhaps the one that attempts to learn representations that are suitable as an input for classical solvers. Within this area, Asai and Fukunaga (2018); Asai and Muise (2020) show how symbolic representations can be extracted from complex tasks in an end-to-end fashion and directly fed into off-the-shelf

solvers. More recently, Vlastelica et al. (2021) frames MDPs as shortest-path problems and trains a convolutional neural network to retrieve the weights of a fixed graph structure. The extracted graph representation can be solved with a combinatorial solver and trained end-to-end by leveraging the blackbox differentiation method (Pogančić et al., 2020).

Visual Goals A further direction of relevant research is that of planning to achieve multiple goals (Nair et al., 2018). While the most common approaches involve learning a goal-conditioned policy with experience relabeling (Andrychowicz et al., 2017), the recently proposed GLAMOR (Paster et al., 2021) relies on learning inverse dynamics and retrieves policies through a recurrent network. By doing so, it can achieve visual goals without explicitly modeling a reward function, an approach that is sensibly closer to ours and can serve as a relevant comparison.

5. Experiments

The purpose of the experimental section is to empirically verify the following claims: (i) PPGS is able to solve challenging environments with an underlying combinatorial structure and (ii) PPGS is able to generalize to unseen variations of the environments, even when trained on few levels. We aim to demonstrate that forming complex plans in these simple-looking environments is beyond the reach of the best suited state-of-the-art methods. Our approach, on the other hand, achieves non-trivial performance. With this in mind, we did not insist on perfect fairness of all comparisons, as the different methods have different type of access to the data and the environment. However, the largest disadvantage is arguably given to our own method.

While visual goals could be drawn from a distribution $p(g)$, we evaluate a single goal for each test level matching the environment solution (or the only state that would give a positive reward in a sparse reinforcement working framework). This represents a very challenging task with respect to common visual goal achievement benchmarks (Paster et al., 2021), while also allowing comparisons with reward-based approaches such as PPO (Schulman et al., 2017). We mainly evaluate the success rate, which is computed as the proportion of solved levels in a set of 100 unseen levels. A level is considered to be solved when the agent achieves the visual goal (or receives a non-zero reward) within 256 steps.

Choice of Baselines Our method learns to achieve visual goals by planning with a world model learned on a distribution of levels. To the best of our knowledge, no other method in the literature shares these exact settings. For this reason, we select three diverse and strong baselines and we make our best efforts for a fair comparison within our computational limits.

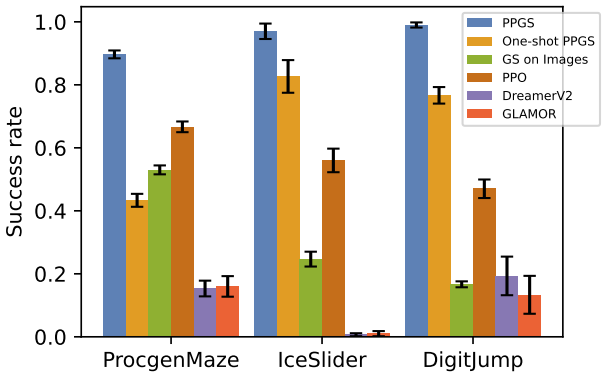


Figure 6: Success rates across the three environments. One-shot planning is competitive with the full method on shorter time horizons.

PPO (Schulman et al., 2017) is a strong and scalable policy optimization method that has been applied in procedurally generated environments (Cobbe et al., 2019). While PPGS requires a visual goal to be given, PPO relies on a (sparse) reward signal specializing on a unique goal per level. DreamerV2 (Hafner et al., 2021) is a model-based RL approach that also relies on a reward signal, while GLAMOR (Paster et al., 2021) is more aligned with PPGS as it is also designed to reach visual goals in absence of a reward.

While we restrict PPGS to only access an offline dataset of low-quality random trajectories, all baselines are allowed to collect data on policy for a much larger number of environment steps. More considerations on these baselines and on the fairness of our comparison can be found in Suppl. B. Furthermore, we also consider a non-learning naive search algorithm (GS ON IMAGES) thoroughly described in C.3.

A comprehensive ablation study of PPGS can be found in Section A of the Appendix.

5.1. Comparison of Success Rates

Our first claim is supported by Figure 6. PPGS outperform its baselines across the three environments. The gap with baselines is smaller in ProcgenMaze, a forgiving environment for which accurate plans are not necessary. On the other hand, ProcgenMaze involves long-horizon planning, which can be seen as a limitation to one-shot PPGS. As the combinatorial nature of the environment becomes more important, the gap with all baselines increases drastically.

PPO performs fairly well with simple dynamics and long-term planning, but struggles more when combinatorial reasoning is necessary. GLAMOR and DreamerV2 struggle across the three environments, as they likely fail to generalize across a distribution of levels. The fact that GS ON IMAGES manages to rival other baselines is a testament to

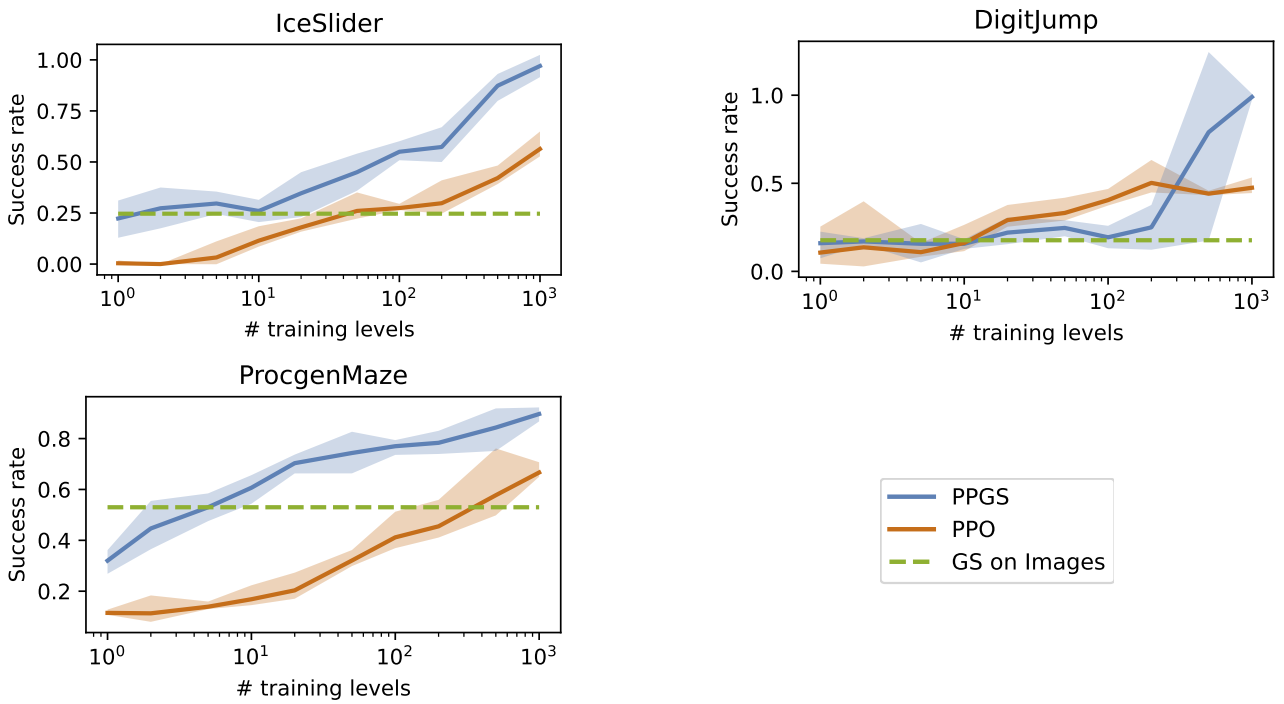


Figure 7: Solution rates of PPGS and PPO as a function of the cardinality of the set of training levels.

the harshness of the environments.

5.2. Analysis of Generalization

The inductive biases represented by the planning algorithm and our training procedure ensure good generalization from a minimal number of training levels. In Fig. 7, we compare solution rates between PPGS and PPO as the number of levels available for training increases. Our method generally outperforms its baselines across all environments. In ProcgenMaze, PPGS achieves better success rates than PPO after only seeing two orders of magnitude less level, e.g. 10 levels instead of 1000. Note that PPGS uses only 400k samples from a random policy whereas PPO uses 50M on-policy samples. Due to the harshness of the remaining environments, PPO struggles to find a good policy and its solution rate on unseen levels improves slowly as the number of training levels increases. In IceSlider, PPGS is well above PPO for any size of the training set and a outperforms GS ON IMAGES when only having access to 2 training levels. While having a comparable performance to PPO on small training sets in DigitJump, our method severely outperforms it once approximately 200 levels are available. On the other hand, PPO's ability to generalize plateaus. These results show that PPGS quickly learns to extract meaningful representations that generalize well to unseen scenarios.

6. Discussion

Hard search from pixels is largely unexplored and unsolved, yet fundamental for future AI. In this paper we presented how powerful graph planners can be combined with learned perception modules to solve challenging environment with a hidden combinatorial nature. In particular, our training procedure and planning algorithm achieve this by (i) leveraging state reidentification to reduce planning complexity and (ii) overcoming the limitation posed by information-dense observations through an hybrid forward model. We validated our proposed method, PPGS, across three challenging environments in which we found state-of-the-art method to struggle. We believe that our results represent a sensible argument in support of the integration of learning-based approaches and classical solvers.

References

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.

pdf.

Anonymous. https://sites.google.com/view/ppgs-url-icml, 2021.

M. Asai and A. Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6094–6101. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16302.

M. Asai and C. Muise. Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to STRIPS). In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2676–2682. ijcai.org, 2020. doi: 10.24963/ijcai.2020/371. URL https://doi.org/10.24963/ijcai.2020/371.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6:737–744, 1993.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

S. Chiappa, S. Racanière, D. Wierstra, and S. Mohamed. Recurrent environment simulators. *CoRR*, abs/1704.02254, 2017. URL http://arxiv.org/abs/1704.02254.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing (NeurIPS)*, 2018.

K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. *CoRR*, abs/1912.01588, 2019. URL http://arxiv.org/abs/1912.01588.

K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25 (3):335–348, 1989.

C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2170–2179. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/gelada19a.html.

D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf.

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/hafner19a.html.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, pages 84–92. Springer, 2015.

B. Ichter and M. Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4 (3):2407–2414, 2019. doi: 10.1109/LRA.2019.2901898.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gax6VtDB.

R. Kumar, A. Mandalika, S. Choudhury, and S. Srinivasa. Lego: Leveraging experience in roadmap generation for sampling-based planning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1488–1495, 2019. doi: 10.1109/IROS40897.2019.8968503.

Y.-L. Kuo, A. Barbu, and B. Katz. Deep sequential models for sampling-based planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6490–6497. IEEE, 2018.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. Sphereface: Deep hypersphere embedding for face recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6738–6746, 2017. doi: 10.1109/CVPR.2017.713.

A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/7ec69dd44416c46745f6edd947b470cd-Paper.pdf.

J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/6ba3af5d7b2790e73f0de32e5c8c1798-Paper.pdf.

M. Okada and T. Taniguchi. Dreaming: Model-based reinforcement learning by latent imagination without reconstruction, 2020.

K. Paster, S. A. McIlraith, and J. Ba. Planning from pixels using inverse dynamics models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=V6BjBgku7Ro.

M. V. Pogančić, A. Paulus, V. Musil, G. Martius, and M. Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(96)00385-2. URL https://www.sciencedirect.com/science/article/pii/S0377221796003852.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

M. Vlastelica, M. Rolínek, and G. Martius. Neuro-algorithmic policies enable fast combinatorial generalization, 2021.

Wikipedia contributors. Kissing number — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Kissing_number&oldid=1020394961. [Online; accessed 9-May-2021].

R. Yonetani, T. Taniai, M. Barekatain, M. Nishimura, and A. Kanezaki. Path planning using neural a* search, 2020.

A. Zhang, R. T. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-2FCwDKRREu.

L. Zhang, G. Yang, and B. C. Stadie. World model as a graph: Learning latent landmarks for planning, 2020.

Supplementary Material for: Planning from Pixels in Environments with Combinatorially Hard Search Spaces

A. Ablation Study

PPGS relies on crucial architectural choices that we now set out to motivate. We do so by performing an ablation study and questioning each of the choices individually to show its contribution to the final performance.

World Model We evaluate the impact of different choices on the world model by retraining it from scratch and reporting the success rate of the full planner in Table 1. We also compute two latent metrics, which are commonly used to benchmark latent predictions (Kipf et al., 2020), see below.

In particular, given a planning horizon K, we first collect a random trajectory $(s_t,a_t)_{t=1\dots L}$ of length L=20 and extract latent embeddings $\{z_t\}_{t=1,\dots L}$ through the encoder h_θ. We then autoregressively estimate the embedding z_{K+1} using only the first embedding z_1 and the action sequence $(a_t)_{t=1\dots K}$, obtaining a prediction $\hat{z}_{K+1}$.

We repeat this process for N trajectories, obtaining N sequences of latent embeddings $(z_t^n)_{t=1\dots L}^{n=1\dots N}$ and N predictions $\{\hat{z}_{K+1}^n\}^{n=1\dots N}$. We compute $\texttt{rank}(\hat{z}_{K+1}^n)$ as the lowest k such that $\hat{z}_{K+1}^n$ is in the k-nearest neighbors of z_{K+1}^n considering all other embeddings $\{z_t^n\}_{t=1,\dots L}$. We can finally compute $H@K{=}\frac{1}{N}\sum_{n=1}^{N}\mathbf{1}_{\texttt{rank}(\hat{z}_{K+1}^n)=1}$ and MMR@K$=\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\texttt{rank}(\hat{z}_{K+1}^n)}$.

We found that training an inverse model is crucial for learning good representations. Even if the uninformative loss L_{margin} introduced in Equation 4 already helps with avoiding the collapse in the latent space, we were not successful in training the forward model to high predictive accuracy unless the inverse model was jointly trained, despite further hyperparameter tuning. We can hypothesize that the inverse model enforces a regular structure in the latent space, which is in turn helpful for training the rest of the world model.

On the other hand, we find that the contribution of an hybrid forward model is more environment-dependent. We ablate this by introducing a forward model that only takes the state embedding z_t and an action a_t as input to predict the next embedding z_{t+1}, without having access to a contextual observation s_c. In this case, the forward model can be implemented as a layer-normalized MLP with 3 layers of 256 neurons. When adopting this fully latent forward model, predictive accuracy drops sensibly. While success rate also drops sharply in DigitJump, this is not the case for ProcgenMaze. We believe that this can be motivated by the fact that several levels of ProcgenMaze could be solved in few steps, only knowing the local structure of the maze with respect to the starting position of the agent. In DigitJump, on the other hand, the agent can easily move across the environment and needs global information to plan accurately.

Planner Evaluating the planning algorithm does not require retraining the world model. We first show the importance of the lookup table. Without correcting the world model's inaccuracies, the planner is not able to recover from incorrect trajectories. As a result, the success rate is comparable to that of one-shot planning. Finally, we empirically validate the importance of state reidentification: when disabled, the BFS procedure is forced to randomly discard new vertices due to the exponential increase in the size of the graph. Because of this, promising trajectories cannot be expanded and the planner is only effective on simple levels which only require short-term planning.

Failure Cases We now present a visual rendition of failure cases for one-shot PPGS in Fig. 8, together with the correct policy retrieved by the full planner.

B. Choice of Baselines and Fairness

After introducing the fundamental reasons behind our choice of baselines in Sec. 5, we present our reasoning and experimental setup with respect to each of the methods.

PPO PPO (Schulman et al., 2017) is a strong model-free RL algorithm. Unlike PPGS, PPO requires a reward signal instead of visual goals. We grant PPO an advantageous setting by allowing on-policy data collection for 50M environment steps, which is in stark contrast to the offline dataset of 400k random transitions that PPGS is trained on. We use the implementation and hyperparameters presented by Cobbe et al. (2019) for the Procgen suite, due to its similarity to the rest of the environments. While PPGS is tuned on ProcgenMaze and keeps its hyperparameters fixed across environments, we favor PPO by tuning the number of timesteps per rollout according to the environment to account for the possibility of getting stuck in a funnel state.

GLAMOR GLAMOR (Paster et al., 2021) learns inverse dynamics to achieve visual goals in Atari games. Similarly to PPGS, GLAMOR does not require a reward signal but needs to receive a visual goal. The only difference with PPGS in terms of settings is that we allow GLAMOR to collect data on-policy and for more interactions (2M). At evaluation time we deploy a strictly more forgiving scheme for GLAMOR, which is described in the original paper (Paster et al., 2021). As GLAMOR is designed to approximately reach its goals, we also accept trajectories that terminate near the actual goal as viable solutions. Hyperparameters

Table 1: Ablations. We evaluate the success rate on two environments when removing important components of our world model and planner. For the world model modifications, we also report metrics for predictive accuracy explained in the text. All results are averaged over 3 seeds.

	ProcgenMaze / DigitJump				
	Success %	H@1	H@10	MMR@1	MMR@10
Our method	0.91/1.00	1.00/1.00	0.92/0.99	1.00/1.00	0.94/1.00
without inverse model	0.38/0.23	1.00/1.00	0.77/0.23	1.00/1.00	0.81/0.33
with fully latent forward model	0.80/0.34	0.98/1.00	0.73/0.53	0.98/1.00	0.81/0.63
without lookup table	0.39/0.92	-	-	-	-
one-shot, with reidentification	0.43/0.76	-	-	-	-
one-shot, without reidentification	0.27/0.31	-	-	-	-

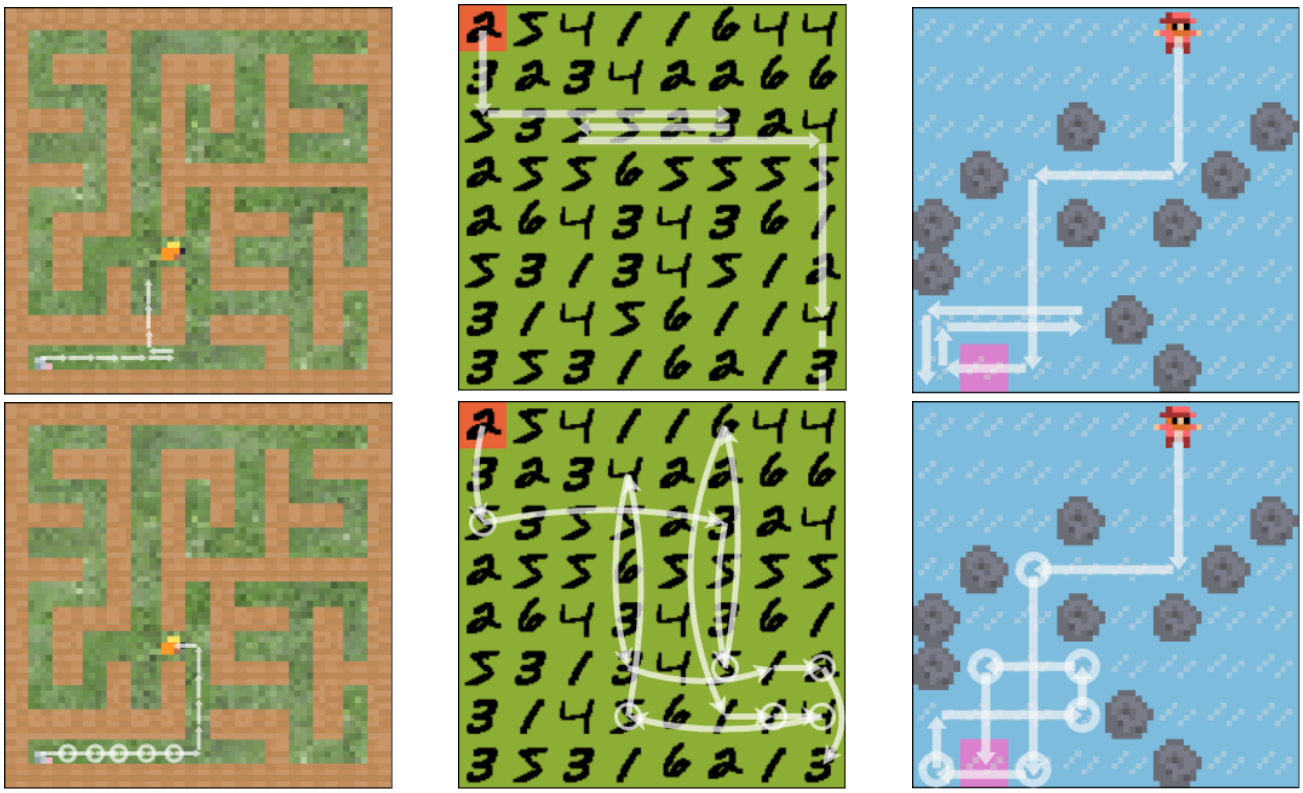


Figure 8: Failure cases. On the first row, a level from each environment that one-shot PPGS fails to solve (the white arrows represent the policy). On the second row, the policies corrected by a full planner, which is able to solve all levels. A white circle is drawn when PPGS recomputes its policy.

for GLAMOR were tuned by the original authors in Atari, which is a visually comparable setting.

DreamerV2 DreamerV2 (Hafner et al., 2021) is a model-based RL approach reaching state-of-the-art performance in discrete games and continuous control domains. We use the original implementation for Atari environments. Due to its large computational requirements, we are only able to run DreamerV2 for a reduced number of steps, totaling 4M. We remark that while this is not enough for performance on Atari to converge, it is shown by the original authors to be sufficient for solving a significant number of games.

All cited codebases that we use are publicly available under a MIT license.

C. Further Implementation Details

In this section we report relevant implementation choices for PPGS. In any case, we refer to our code (Anonymous, 2021) for precise details.

C.1. World Model

Encoder. The encoding function h_θ is learned by a convolutional neural network. The output of the convolutional backbone of a ResNet-18 (He et al., 2015) is fed through a single fully connected layer with d units, where d=16 is the size of the latent space Z. The output of the network is normalized to unit L2 norm.

Forward Model. From an architectural perspective, our hybrid forward model transforms the state embedding z_t through a deconvolutional network and concatenates it to the RGB observation s_c and a batchwise one-hot tensor representing the action. The result is processed through a second ResNet-18 to predict the next embedding. We found it to be irrelevant whether to train the network to predict a state representation z_{t+1} or a latent offset $z_{t+1}-z_t$: for our experiments we choose the former.

Similarly to Hafner et al. (2020) we find that, in practice, explicitly encouraging representations to be predictive for longer horizons (for instance through a multi-step loss) does not appear to be helpful. For this reason, we only train for one-step predictions, as noted in Equation 2.

Inverse Model. To enforce a simpler structure in the latent space, we implement the inverse model p_ω as a low-capacity one-layer MLP with ReLU activations, 32 neurons and layer normalization (Ba et al., 2016).

Hyperparameters Hyperparameters are fixed in all experiments unless explicitly mentioned. More importantly, we deploy the same set of hyperparameters across all three environments, after tuning them on ProcgenMaze via grid search. The latent margin ε is set to 0.1 and the dimensionality of the latent space d is set to 16. The world model we propose is optimized using Adam (Kingma and Ba, 2014) with learning rate λ=0.001 for all components and parameters ε=0.00001,β_1=0.9,β_2=0.999. All components are trained for 40 epochs with a batch size of 128. The losses are combined as in Equation 5 with weights α=10,β=1, although our architecture shows robustness to this choice. Training the world model takes approximately 20 hours on a single NVIDIA ampere GPU.

C.2. Planner

In this subsection, we present both planners (one-shot and full) more in detail.

One-shot Planner Algorithm 2 offers a more thorough description of the one-shot planner introduced in Algorithm 1. Given a visual goal g and an initial observed state s, maximizing discounted rewards corresponds to recovering the shortest action sequence $(a_i)_{1,\dots,n}$ such that s_n=g.

For this purpose, Algorithm 2 efficiently builds and searches the latent graph. It has access to an initial high-dimensional state s and a visual goal g; it keeps track of a list of visited vertices V and of a set of leaf vertices L. For each visited latent embedding, the algorithm stores the action sequence leading to it in a dictionary D. A key part of the algorithm is represented by the `filter` function. The `filter` function receives as input the new set of leaves L', from which vertices reidentifiable with visited states have already been discarded. The function removes elements of L' until no pair of states is too close. This is done by building a secondary graph with the elements of L' as vertices and edges between all pairs of vertices at less than $\frac{\epsilon}{2}$ distance. A set of non-conflicting elements can then be recovered by approximately solving a minimum vertex cover problem. If the state space of the environment grows exponentially with the planning horizon, or if the world model fails to reidentify bisimilar states, L' can still reach impractically large sizes. For this reason, after resolving conflicts, if its cardinality is larger than a cutoff C=256, $|L_{t+1}|-C$ elements are uniformly sampled and removed.

Full Planner The full planner used by PPGS introduces the possibility of online replanning in an MPC approach. It autoregressively computes a latent trajectory T conditioned on the action sequence P retrieved by one-shot planning. At each step, the current observation is projected to the latent space to check if it can be reidentified with the predicted embedding in T. When this is not possible, the action sequence P is recomputed. Moreover, the planner gradually fills a latent transition buffer B. Forward predictions are then computed according to $\hat{f}_\theta(z,a,s)$, which returns z' if $(z,z',a)\in B$, otherwise it queries the learned forward model f_θ. As a side note, when replanning while using the full planner, the planning horizon T_{max} is set to 10 steps. We report the method in full in Algorithm 3.

C.3. GS ON IMAGES

Our baselines include a graph search algorithm in observation space that does not involve any learned component. We refer to this algorithm as GS ON IMAGES and it can be seen as a measure of how hard an environment is when relying on reconstructing the state diagram to solve it. GS ON IMAGES assumes solely on the deterministic nature of the environment. Given a starting state s, a goal state g and the action set A, GS ON IMAGES plans as shown in Algorithm 4. It relies on a dictionary A_{left} which stores, for each visited state, the set of actions that have not been attempted yet, and on a

Algorithm 2 One-shot PPGS

Input: s,g
Output: action sequence $(a_i)_{1,\dots,n}$

z,z_g=$h_\theta(s)$,$h_\theta(g)$ ▷ project to latent space
V=L=$\{z\}$
D=$\{z$:[]$\}$
while for T_{MAX} steps **do**
  L'=$\emptyset$
  for $z \in L$ **do** ▷ grow the latent graph
    for $a \in A$ **do**
      z'=$f_\phi(z,a,s)$
      if $\min_{v \in V} \|z'-v\|_2 > \frac{\varepsilon}{2}$ **then** ▷ skip if already visited
        L'=$L' \cup \{z'\}$
        $D[z']$=$D[z]$+$[a]$
      end if
    end for
  end for
  L=`filter`(L') ▷ select the largest group of elements such that no pair is too close
  $z^\star$=$\text{argmin}_{z \in L} \|z-z_g\|_2$
  if $\|z^\star - z_g\|_2 \leq \frac{\varepsilon}{2}$ **then** ▷ if $z^\star$ can be reidentified with the goal
    return $D[z^\star]$
  end if
  V=$V \cup L$ ▷ add leaves to visited set
end while

graph representation of the environment G=$(V$,$E)$, where V is the set of visited states and E contains the observed transitions between states and labeled by an action.

C.4. Data Collection

As our method solely relies on accurately modeling the dynamics of the environment, the only requirement for training data is sufficient coverage of the state space. In most cases, this is satisfied by collecting trajectories offline according to a uniformly random policy. The ability to leverage a fixed dataset of trajectories draws PPGS closer to off-policy methods or even batch reinforcement learning approaches.

In practice, unless specified otherwise, we collect 20 trajectories of 20 steps from a set of n=1000 training levels, for a total of $400k$ environment steps. One exception is made for ProcgenMaze, for which we also set a random starting position at each episode, since uncorrelated exploration is not sufficient to cover significant parts of the state.

D. Environments

In this section, we present a few remarks on the environments chosen. For ProcgenMaze, we choose what is reported as the *easy* distribution in Cobbe et al. (2019). This corresponds to grids of size $n \times n$, with $3 \leq n \leq 15$; each cell of the grid is either a piece of wall or a corridor. In IceSlider, the agent always starts in the top row and needs to descend to a goal on the bottom row. It is not sufficient to slide over the goal, but the agent needs to come to a full stop on the correct square. In DigitJump, the handwritten digits are the same across training and test levels. Their frequency and position does of course change.

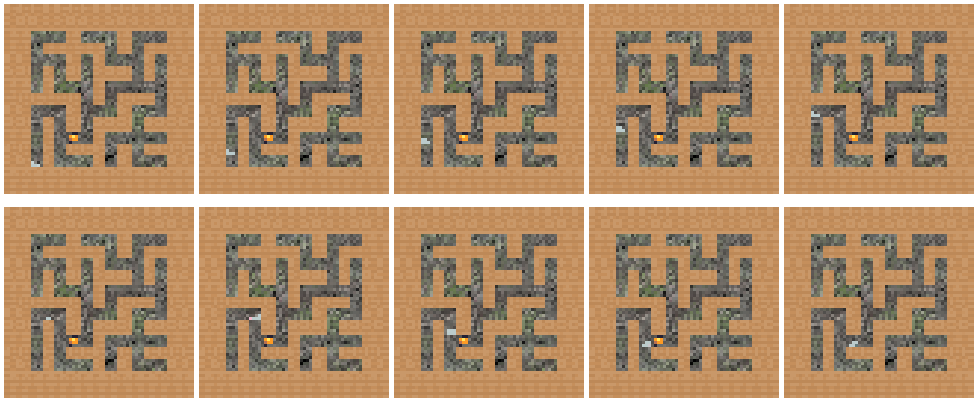
All environments return observations as 64x64 RGB images. ProcgenMaze and IceSlider are rendered in a similar style to ATARI games, while the DigitJump is a grid of MNIST (LeCun et al., 1998) digits that highlights the cell at which the agent is positioned. The action space in all cases is restricted to four cardinal actions (UP, DOWN, LEFT, RIGHT) and a no-op action, for a total of 5 actions.

Examples of expert trajectories are shown in Fig. 9. For more information on the environments, we refer the reader to the supplementary material (Anonymous, 2021).

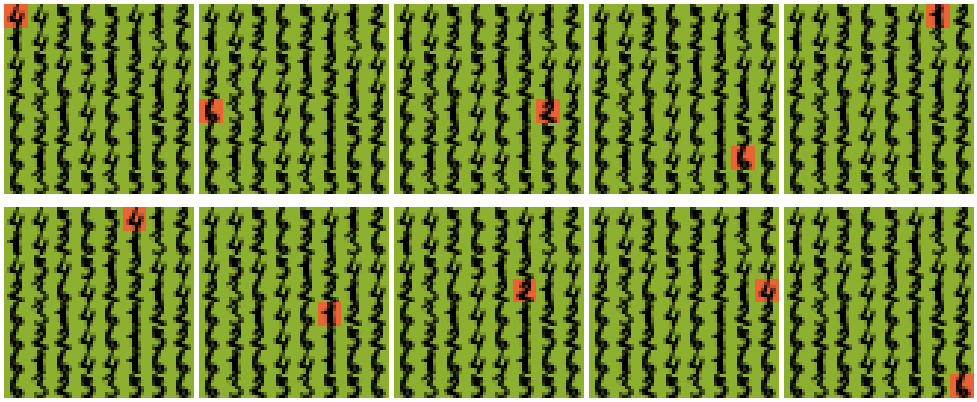
E. Numerical Results

We finally include the full numerical results from Fig. 6 and Fig. 7 in Table 2.

ProcgenMaze



DigitJump



IceSlider

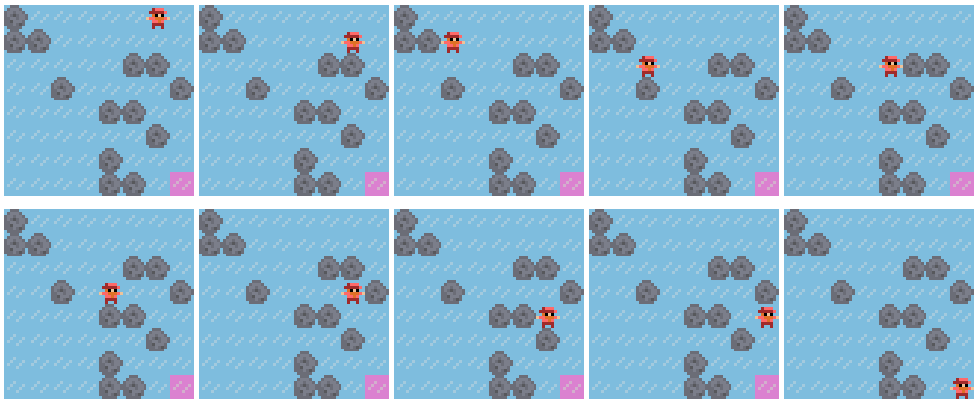


Figure 9: Expert trajectories for a level extracted from each of the environments.

Algorithm 3 PPGS (full planner)

Input: s,g

z=$h_\theta(s)$
B=$\emptyset$ ▷ set of observed latent transitions
P=`one_shot_PPGS`(s,g) ▷ retrieve initial policy
T=$[z]$
for a in P **do**:
    T=T+$[\hat{f}_\phi(T[-1],a,s)]$ ▷ autoregressively predict latent trajectory T
end for
T.pop(0) ▷ discard first embedding
while $s \neq g$ **do**
    a=P.pop(0) ▷ take first action and remove it from the action list
    take action a and reach state s'
    z'=$h_\theta(s')$
    B=$B \cup \{(z,z',a)\}$
    z_{pred}=T.pop(0) ▷ retrieve predicted embedding
    if $\|z_{\text{pred}} - z'\|_2 > \frac{\varepsilon}{2}$ or A = [] **then** ▷ if the latent trajectories does not match predictions
        A=`one_shot_PPGS`(s',g) ▷ replan
        T=$[z]$
        for a in P **do**:
            T=T+$[\hat{f}_\phi(T[-1],a,s)]$ ▷ autoregressively predict latent trajectory T
        end for
        T.pop(0) ▷ discard first embedding
    end if
    s=s'
    z=z'
end while

Algorithm 4 GS ON IMAGES

Input: s,g

A_{left}=$\{s$:$A\}$
V=$\{s\}$
E=$\emptyset$
while $s \neq g$ **do**
    if $A_{\text{left}}[s]$=$\emptyset$ **then**
        if $\exists s' \in V$ such that $A_{\text{left}}[s'] \neq \emptyset$ and s' is reachable from s **then**
            find and apply action sequence to reach closest $s' \in V$
            s=s'
        else
            return
        end if
    else
        $a \sim \mathcal{U}(A_{\text{left}}[s])$ ▷ uniformly sample among remaining actions
        $A_{\text{left}}[s]$=$A_{\text{left}}[s] \setminus \{a\}$
        take action a and reach state s'
        if $s' \notin V$ **then**
            V=$V \cup \{s'\}$
            $A_{\text{left}}[s]$=A
        end if
        E=$E \cup (s,s',a)$
    end if
end while

Table 2: Generalization results. This table presents the numerical results used to produce Fig. 6 and Fig. 7. All metrics are averaged over 3 random seeds.

	ProcgenMaze - Success % on unseen levels when training on n levels									
	n=1	2	5	10	20	50	100	200	500	1000
PPGS	0.320	0.447	0.530	0.607	0.703	0.743	0.770	0.783	0.843	0.897
PPO	0.115	0.113	0.139	0.168	0.203	0.321	0.417	0.455	0.577	0.667
DreamerV2	-	-	-	-	-	-	-	-	-	0.153
GLAMOR	-	-	-	-	-	-	-	-	-	0.100

	IceSlider - Success % on unseen levels when training on n levels									
	n=1	2	5	10	20	50	100	200	500	1000
PPGS	0.223	0.273	0.296	0.260	0.347	0.450	0.550	0.573	0.873	0.970
PPO	0.004	0.000	0.032	0.115	0.179	0.260	0.274	0.298	0.421	0.564
DreamerV2	-	-	-	-	-	-	-	-	-	0.007
GLAMOR	-	-	-	-	-	-	-	-	-	0.010

	DigitJump - Success % on unseen levels when training on n levels									
	n=1	2	5	10	20	50	100	200	500	1000
PPGS	0.160	0.170	0.157	0.157	0.220	0.247	0.193	0.250	0.790	0.990
PPO	0.107	0.137	0.109	0.161	0.291	0.331	0.405	0.502	0.441	0.475
DreamerV2	-	-	-	-	-	-	-	-	-	0.193
GLAMOR	-	-	-	-	-	-	-	-	-	0.133