TRANSDREAMER: REINFORCEMENT LEARNING WITH STOCHASTIC TRANSFORMER WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

The Dreamer agent provides various benefits of Model-Based Reinforcement Learning (MBRL) such as sample efficiency, reusable knowledge, and safe planning. However, its world model and policy networks inherit the limitations of recurrent neural networks and thus an important question is how an MBRL framework can benefit from the recent advances of transformers and what the challenges are in doing so. In this paper, we propose a transformer-based MBRL agent, called TransDreamer. We first introduce the Transformer State-Space Model, a world model that leverages a transformer for dynamics predictions. We then share this world model with a transformer-based policy network and obtain stability in training a transformer-based RL agent. In experiments, we apply the proposed model to 2D visual RL and 3D first-person visual RL tasks both requiring long-range memory access for memory-based reasoning. We show that the proposed model outperforms Dreamer in these complex tasks.

1 INTRODUCTION

Model-based reinforcement learning (MBRL) (Sutton, 1991) provides a solution for many problems of current reinforcement learning. Its imagination-based training provides sample efficiency by fully leveraging the interaction experience via the world model (Ha & Schmidhuber, 2018), the world model can be considered a form of task-agnostic general knowledge enabling reusability of the knowledge about the environment in many downstream tasks (Sekar et al., 2020), and finally the dynamics model makes planning possible (Schrittwieser et al., 2020; Hafner et al., 2018) for accurate and safe decisions (Berkenkamp et al., 2017; Kidambi et al., 2020; Lu et al., 2020).

Among the recent advances in MBRL, a particularly notable one is the Dreamer agent (Hafner et al., 2019; 2020). Learning a world model in latent representation space, Dreamer is the first visual MBRL model that achieves performance and sample efficiency better than model-free approaches such as Rainbow (Hessel et al., 2018) and IQN (Dabney et al., 2018).

To deal with partial observability (Kaelbling et al., 1998), the dynamics models in MBRL have been implemented using recurrent neural networks (RNNs) (Hafner et al., 2019; 2020; Schrittwieser et al., 2020; Kaiser et al., 2019). However, Transformers (Vaswani et al., 2017; Dai et al., 2019) have shown to be more effective than RNNs in many domains requiring long-term dependency and direct access to memory for a form of memory-based reasoning (Ritter et al., 2020; Banino et al., 2020). Also, it has been shown that training complex policy networks based on transformers using only rewards is difficult (Parisotto et al., 2020), so learning a transformer-based world model where the training signal is more diverse may facilitate learning. Therefore, it is important to investigate how to make an MBRL agent using a transformer-based world model and to analyze the benefits and challenges in doing so.

In this paper, we propose a transformer-based MBRL agent, called TransDreamer. As implied by the name, the proposed model inherits from the Dreamer framework, but aims to bring the benefits of transformers into it. Seemingly, it may seem like a simple plug-in task to replace an RNN with a transformer. However, there are a few critical challenges to make it work. First of all, we need to develop a new transformer-based world model that supports effective stochastic action-conditioned transitions in the latent space to implement the prior and posterior of the transition model. At the same time, this model should also preserve the parallel trainability of transformers for computational efficiency. To the best of our knowledge, there is no such model yet. Also, as shown in Parisotto

et al. (2020), finding an architecture, hyperparameters, and other design choices to make a working model is particularly challenging for Transformer-based RL models.

The main contribution of this paper is the first transformer-based MBRL agent. We introduce the Transformer State-Space Model (TSSM) as the first transformer-based stochastic world model. Using this world model in the Dreamer framework, we propose TransDreamer, a fully transformer-based MBRL framework. In experiments, we show that TransDreamer outperforms Dreamer on tasks that requires long-term and complex memory interactions, and the world model of TransDreamer is better than Dreamer at predicting rewards and future frames for imagination. Furthermore, we also show that the performance of TransDreamer is comparable to Dreamer on a few simple DMC (Tassa et al., 2018) and Atari (Bellemare et al., 2013) tasks that do not require long-term memory.

2 PRELIMINARIES

Our model builds on top of Dreamer (Hafner et al., 2019; 2020), a model-based reinforcement learning framework (Sutton, 1991) for visual control in a partially observable Markov decision process (POMDP) (Kaelbling et al., 1998). Dreamer consists of three main steps: (1) world model learning, (2) policy learning, and (3) environment interaction. These steps are cycled until convergence. Specifically, a dynamics model of the environment (i.e., world model) and a reward function are learned to fit the data in an experience replay buffer. Then, an actor-critic policy is trained on imagined experience, i.e., hypothetical trajectories generated by simulating the learned world model. Lastly, to provide fresh experience to the replay buffer, the agent collects trajectory data by executing the trained policy on the real environment.

2.1 WORLD MODEL IN DREAMER

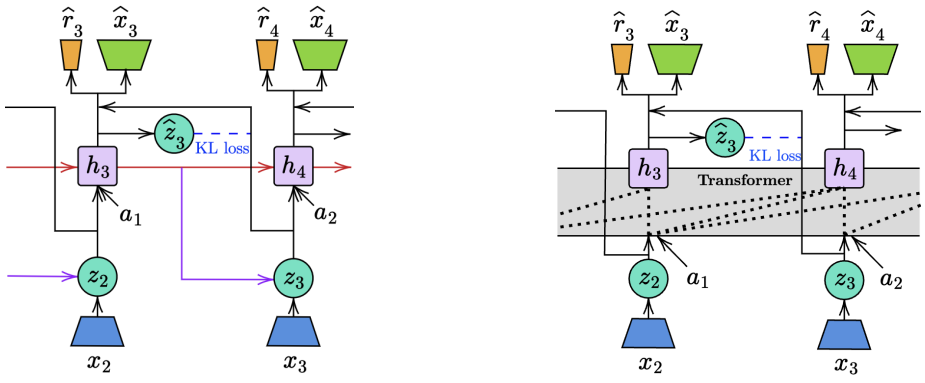
The backbone of the world model used in Dreamer is a stochastic recurrent neural network, called the Recurrent State-Space Model (RSSM) (Hafner et al., 2018). Depicted in Figure 1a, the RSSM represents a latent state s_t by the concatenation of a stochastic state z_t and a deterministic state h_t which are updated by $z_t \sim p(z_t|h_t)$ and $h_t = f(h_{t-1}, z_{t-1}, a_{t-1})$, respectively. Here, a_{t-1} is an action and the deterministic update f is the state update rule of a recurrent neural network (RNN) such as an LSTM (Hochreiter & Schmidhuber, 1997) or GRU (Chung et al., 2014). While the deterministic path helps to model the temporal dependency in the world model, it also inherits the limitations of RNNs, particularly when compared to the benefits of transformers. The stochastic state makes it possible to capture the stochastic nature of the world, e.g., for imagining multiple hypothetical future trajectories. Crucially, using the above models, rollouts can be executed efficiently in a compact latent space without the need to generate observation images.

Learning the RSSM is via the maximization of evidence lower bound (Jordan et al., 1999). The representation model $z_t \sim q(z_t|h_t, x_t)$ infers the stochastic state given an observation x_t. Whenever a new observation is provided, the current state is updated by the representation model. The observation model $p(x_t|s_t)$ and the reward model $p(r_t|s_t)$ are then used for the reconstruction of observation x_t and reward r_t from the latent state. All component models are listed in Table 1.

2.2 POLICY LEARNING IN DREAMER

After updating the world model for a number of iterations, Dreamer updates its policy $\pi_\phi(a_t|s_t)$. The policy learning is done without interaction with the actual environment; it uses imagined trajectories obtained by simulating the learned world model. Specifically, from each state s_t obtained from a batch sampled from the replay buffer, it generates a future trajectory of length H using the RSSM world model and the current policy as the behavior policy for the imagination. Then, for each state in the trajectory, the rewards $p_\theta(r_t|s_t)$ and the values $v_\psi(s_t)$ are estimated. This allows us to compute the value estimate $V(s_t)$, e.g., by the discounted sum of the predicted rewards and the bootstrapped value $v(s_{t+H})$ at the end of the trajectory. See Hafner et al. (2019) for more details and other options about the value estimation.

Learning the policy in Dreamer means updating two models, the policy $\pi_\phi(a_t|s_t)$ and the value model $v_\psi(s_t)$. For updating the policy, Dreamer uses the sum of the value estimates of the sim-



(a) RSSM **(b)** TSSM

Figure 1: RSSM and TSSM. The red arrow in RSSM makes sequential computational necessary. In TSSM, we replace this by a transformer. In addition, the purple arrow should also be removed in TSSM because it also prevents parallelizing the updates of all time steps.

ulated trajectories, $\sum_{\tau=t}^{t+H} V(s_\tau)$, to construct the objective function. While we can compute the gradient of this objective w.r.t. the parameters of the policy ϕ via a policy gradient method such as REINFORCE (Williams, 1992), Dreamer also takes advantage of the differentiability of the learned world model by directly backpropagating from the value function to the world model, and to the parameters of the policy network. This provides gradients of lower variance than that of REINFORCE. Updating the value model parameter ψ is done via temporal difference learning with the value estimate $V(s_t)$ as the target.

3 TRANSDREAMER

Transformers have been shown to outperform RNNs in many tasks in both NLP and computer vision. In particular, their ability to directly access historical states and to learn complex interactions among them, has been shown to excel in tasks that require complex long-term temporal dependencies such as memory-based reasoning (Ritter et al., 2020; Banino et al., 2020). Furthermore, they have been shown to be effective for temporal generation in both language and visual domains. Observing that both of these abilities are important and desirable properties for a robust world model, we hypothesize that a model-based agent based on transformers can outperform the RNN-based Dreamer agent for tasks requiring complex and long-term memory dependency.

3.1 TRANSFORMER STATE SPACE MODEL (TSSM)

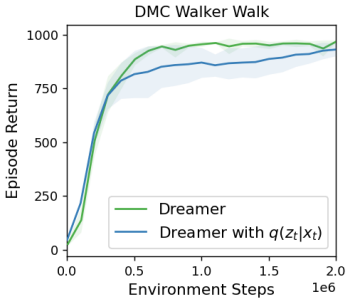
In the design of a transformer-based world model, we aim to achieve the following desiderata: (i) to directly access past states, (ii) to update the states of each time step in parallel during training, (iii) to be able to roll out sequentially for trajectory imagination at test time, and (iv) to be a stochastic latent variable model. To our knowledge, no such model is available. The RSSM does not satisfy (i) and (ii). Although we can consider a simple modification of the RSSM, a memory-augmented RSSM, by introducing direct attention to the past states in the RNN state update (Ke et al., 2018) and make (i) satisfied as well, it still does not satisfy (ii) and thus remains computationally inefficient. Traditional transformers are deterministic and thus do not satisfy (iv). This motivates us to introduce the Transformer State-Space Model (TSSM).

The TSSM is a stochastic transformer-based state-space model. Figure 1 illustrates the architectures of TSSM in comparison to RSSM. In the RSSM, the main source of the sequentially-dependent computation is the RNN-based state update $h_t = f_{\text{gru}}(h_{t-1}, z_{t-1}, a_{t-1})$, depicted in red arrows in Figure 1a. This means that all component models of the RSSM are computed sequentially because they all take the hidden state as input. To remove this sequential computation and enable direct access to and complex interaction of the historical states, we propose employing a transformer as a replacement for the RNN. Unlike the RSSM which accesses the past indirectly only via a compression h_{t-1}, the transformer is allowed to directly access the sequence of stochastic states and actions of the past at every time step, i.e., $h_t = f_{\text{transformer}}(z_{1:t-1}, a_{1:t-1})$. If all h_t can be computed in parallel in this way, then all components taking h_t as input can also be computed in parallel.

Table 1: Comparison of the Component Models of RSSM and TSSM.

	RSSM	**TSSM**
Deterministic state model	$h_t = \text{gru}(h_{t-1}, z_{t-1}, a_{t-1})$	$h_t = \text{transformer}(z_{1:t-1}, a_{1:t-1})$
Representation model	$z_t \sim q(z_t \mid h_t, x_t)$	$z_t \sim q(z_t \mid x_t)$
Stochastic state model	$\hat{z}_t \sim p(\hat{z}_t \mid h_t)$	
Image predictor	$\hat{x}_t \sim p(\hat{x}_t \mid h_t, z_t)$	
Reward predictor	$\hat{r}_t \sim p(\hat{r}_t \mid h_t, z_t)$	
Discount predictor	$\hat{\gamma}_t \sim p(\hat{\gamma}_t \mid h_t, z_t)$	

Myopic Representation Model. This hope, however, is broken due to the representation model $q(z_t|h_t, x_t)$. This is because unlike other component models listed in Table 1, the output of the representation model is used as the input to the transformer. Since a transformer should not use an output also as an input to achieve parallel training, the representation model should not be conditioned on h_t. That is, the purple arrows in Figure 1 should be removed. To this end, we propose approximating the posterior representation model by $q(z_t|x_t)$, removing h_t. Since the posterior can now be computed independently for each time step, we can compute all of the inputs $z_{1:t-1}$ simultaneously. Then, with a single forward pass through the transformer, we can obtain $h_{1:t}$.



One may argue that removing h_t and thereby ignoring all the history $z_{1:t-1}$ in the representation model may result in a poor approximation. This can be true if we use only z_t as the full state of our model. However, like the RSSM, the full state of our model is a concatenation of the stochastic state z_t and the deterministic state h_t. Therefore, we hypothesize that encoding temporal information in the stochastic states may not be essential for model performance because that information is provided by the deterministic states. Furthermore, the trajectory imagination does not require the representation model but only requires the stochastic state model $p(z_t|h_t)$ that can still use h_t. We observe that if this hypothesis is correct, a modified Dreamer using $q(z_t|x_t)$ instead of $q(z_t|x_t, h_t)$ would perform similarly to the original Dreamer and the plot on the right confirms this. Another possible yet more complex choice for the representation model is to condition the posterior representation directly on all the past observations using another transformer layer, i.e., $q(z_t|f_{\text{transformer}}(x_{1:t}, a_{1:t}))$. However, due to its increased complexity, we do not consider this model.

Imagination. During imagination, we use the prior stochastic state $\hat{z}_t \sim p(\hat{z}|h_t)$ as the input to the transformer to autoregressively generate future states as shown in Figure 6 in the Appendix. This allows the agent to imagine future states completely in the latent space but with more direct access to the historical states than the RSSM. In this way, the TSSM achieves all the desiderata discussed above. Table 1 highlights the key differences between the RSSM and the TSSM.

The **loss function** is almost the same as that of the RSSM. The difference is that we approximate the representation posterior $p(z_{1:t}|x_{1:t})$ by $\prod_{\tau=1}^{t} q_\phi(z_\tau|x_\tau)$ instead of $\prod_{\tau=1}^{t} q_\phi(z_\tau|z_{1:\tau-1}, x_{1:\tau})$ used in the RSSM. The loss function can be found in Appendix A.2 with the derivation of the ELBO.

3.2 Policy Learning and Implementation Details

Policy Learning. The policy learning in TransDreamer inherits the general framework of Dreamer described in Section 2.2. The main difference is that we replace the RSSM with the TSSM. The component models are thus based on the states from the TSSM which can capture the long-term and complex temporal dependency better. Since the TSSM is fully differentiable, we can similarly use both REINFORCE and dynamics backpropagation to train the policy. The TSSM parameters are held fixed when training the agent.

Training Stability. Transformers have notably had stability issues when used in RL settings, especially in cases where the rewards are sparse. GTrXL (Parisotto et al., 2020), in particular, adds GRU gating layers to try to alleviate this problem. In TransDreamer, however, since the transformer parameters are held fixed during agent training and only trained to predict images, rewards, and discounts, we find that we do not encounter similar stability issues even without any additional gating.

Prioritized Replay. Since the training signal for agent training is based on only the rewards the agent receives, the reward prediction in the world model is especially important for learning a good agent. Learning a good reward predictor, on the other hand, relies on the agent having a good enough policy so that it can collect trajectories with reward, especially in environments with sparse rewards. To facilitate this, we optionally weight the replay buffer so that trajectories with higher rewards are more likely to be sampled. In particular, we sample only from nonzero-reward trajectories α-percentage of the time where $\alpha \in [0, 1]$. The remaining trajectories are sampled uniformly across the replay buffer.

Number of Imagination Trajectories. Due to the increased memory requirements of transformers compared with RNNs, we find that it is not feasible to generate imagined trajectories from every state sampled from the replay buffer, as is done in Dreamer. Instead, we randomly choose a smaller subset of starting states of size K to generate imagined trajectories from. While this effectively reduces the number of trajectories the agent can learn from in any given iteration, we find that we are still able to achieve performance comparable or better than Dreamer.

4 Related Works

Transformers in RL. Transformers have been used successfully in diverse domains including NLP (Vaswani et al., 2017; Devlin et al., 2019; Radford et al., 2018; 2019; Brown et al., 2020), computer vision (Chen et al., 2020; Dosovitskiy et al., 2021; Parmar et al., 2018), and video generation (Weissenborn et al., 2020; Yan et al., 2021). Parisotto et al. (2020) address the problem of using transformers in RL and showed that adding gating layers on top of the transformers layers can stabilize training. Subsequent works addressed the increased computational load of using a transformer for an agent's policy (Irie et al., 2021; Parisotto & Salakhutdinov, 2021). Chen et al. (2021); Janner et al. (2021) take a different approach by modeling the RL problem as a sequence modeling problem and use a transformer to predict actions without additional networks for an actor or critic. Several recent works also explore long-term video generation with transformers which is related to building world models using transformer-based architectures (Wu et al., 2021; Creswell et al., 2021).

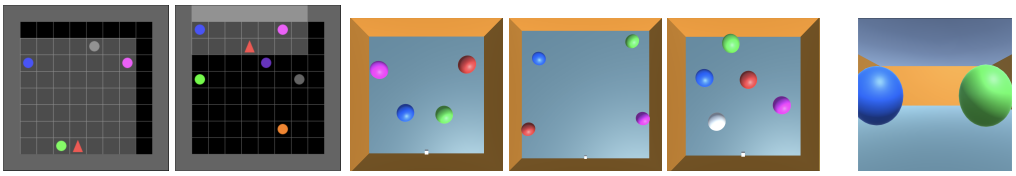
Stochastic Transformers. Stochasticity has been added to several transformer-based architectures in the context of response generation (Lin et al., 2020), sign language translation (Voskou et al., 2021), story completion (Wang & Wan, 2019), and layout generation (Arroyo et al., 2021). Martin et al. (2020) introduce the Sequential Monte Carlo Transformer which adds stochastic hidden states to the network architecture and outputs a distribution of predictions allowing for uncertainty quantification. To our knowledge, no previous work investigates stochastic transformers in the context of world models and MBRL.

Model-based RL. Dyna (Sutton, 1991) introduced a general framework for MBRL that our model is based on. SimPle (Kaiser et al., 2019) adopts this framework by making predictions at the pixel level and training a PPO agent on that model. Our work mainly builds off of the Dreamer (Hafner et al., 2019; 2020) framework. The RSSM is first introduced in PlaNet (Hafner et al., 2018) where it is used for planning in the latent space. Ha & Schmidhuber (2018) use a VAE with an RNN as the world model and learns a policy with an evolution strategy. MuZero (Schrittwieser et al., 2020) uses task-specific rewards to build a model and Monte-Carlo Tree Search to solve RL tasks.

5 Experiments

In this section, we compare TransDreamer and Dreamer on a variety of tasks, from tasks that require long-term memory and reasoning to tasks that can be solved with only short-term memory. We try to answer the following three questions: 1) How do TransDreamer and Dreamer perform in tasks that require long-term memory and reasoning? 2) How do the learned world models of TransDreamer and Dreamer compare? 3) Can TransDreamer also work comparably to Dreamer in environments that require short-term memory?

To answer the first question, we created a new set of tasks called Hidden Order Discovery that is inspired by the Numpad task (Humplik et al., 2019; Parisotto et al., 2020). We create both 2D and 3D versions of this task. The 2D environment, built with the Minigrid (Chevalier-Boisvert et al., 2018) framework, provides a top-down view of the agent navigating a room while the 3D environment,



(a) 2D 4-Ball and 6-Ball **(b)** 3D 4-Ball Dense/Sparse and 5-Ball Dense **(c)** 3D Agent View

Figure 2: Hidden Order Discovery Environments.

built with Unity (Juliani et al., 2020), provides a more partially observable and visually rich first-person view of the agent. These tasks require long-term memory and reasoning to solve. To answer the second question, we thoroughly analyze the quality of the world model that is learned in solving these tasks both quantitatively and qualitatively. Lastly, to answer the third question, we compared TransDreamer and Dreamer on some tasks in DeepMind Control Suite (DMC) (Tassa et al., 2018) and Atari (Bellemare et al., 2013). These tasks are almost fully observable and do not require long-term memory and reasoning to solve.

5.1 Hidden Order Discovery in 2D Object Room

To evaluate our model on the tasks that require long-term memory, we created a new task called Hidden Order Discovery inspired by the NumPad task (Humplik et al., 2019; Parisotto et al., 2020). In this task, there are several color balls in a 2D grid, as shown in Fig. 2a. The agent (illustrated by the red triangle) can only see the highlighted area in front of it, so this is a partially observable environment. For each episode, there is a hidden order of balls and the agent must collect the balls *in the correct order*. If the agent fails, the map is reset, and the agent needs to start from the first ball. Note that when the map is reset, the agent position and the hidden order are not changed but all the balls are reset to their initial positions. Therefore, to find the hidden ball order efficiently, the agent can benefit from remembering what it has tried in the past in the current episode. When a ball is collected in the correct order, a reward of +3 is given, but if the map is reset due to the agent collecting the incorrect ball, the rewards for balls visited in previous tries are 0. This prevents the agent from collecting a high reward from just constantly revisiting the first ball in the sequence. When the agent successfully collects every ball in the hidden order, the map and rewards are reset. The hidden order and ball positions are randomized per episode.

We evaluate in environments with 4, 5, and 6 balls where the grid size is 8x8 cells, and the agent is given 100 time steps to collect as much reward as possible. The results are shown in Figure 3. We see that TransDreamer outperforms Dreamer in all of these configurations. Since the reward for correctly collecting one ball is +3, an average reward of 3 means that on average the agent collects the first ball correctly, and an average reward of 6 means that on average the agent collects the first two balls correctly, and so on. For the 4-Ball configuration, TransDreamer reaches an episode reward of around 7 while Dreamer's episode reward is around 4. This means that in the 4-Balls setting, TransDreamer averages collecting over two balls in the correct order, whereas Dreamer only collects one ball in the correct order.

To obtain further understanding beyond the averaged behavior, we also measure the success rate of each agent, defined as the percentage of trajectories where an agent collects *all* balls in the correct order at least once. For the 4-Ball configuration, we find that TransDreamer has a success ratio of 23% and Dreamer has a success ratio of only 7%, providing further evidence that TransDreamer can better solve this task than Dreamer. A full comparison of the success ratio is reported in Appendix Table 3. The difficulty of this task increases as the number of balls increases since with more balls, there are more combinations for the agent to try before determining the hidden order. Thus, we see the performance for both degrade as the number of balls increases.

We emphasize the difficulty of this task. Because the hidden order is randomized in each episode, in the worst-case scenario, discovering the first ball in the given order would require searching through all 4 balls. Then, determining the second ball in the sequence would require searching among the remaining 3 balls, while always remembering what has happened before in order not to waste time by visiting balls already known to be incorrect. The higher scores for TransDreamer provides some evidence that the transformer-based architecture is effective in tasks that require this long-term memory and reasoning.

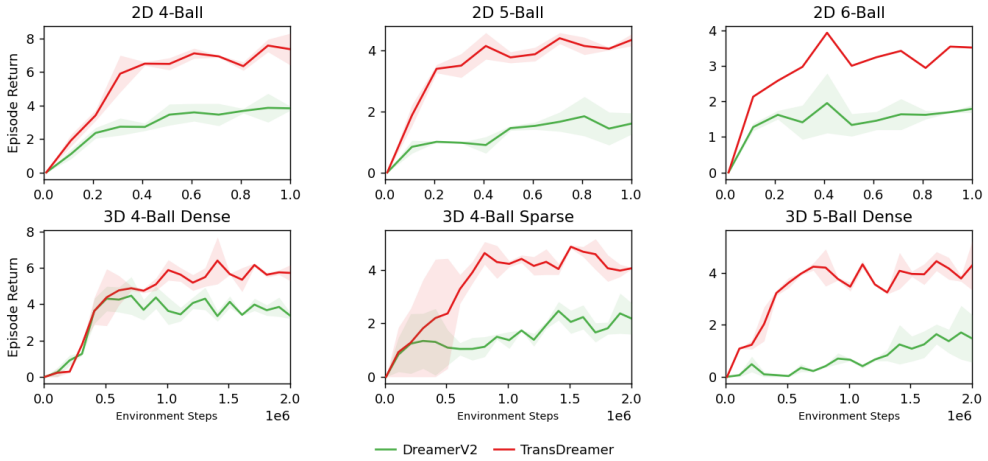


Figure 3: Comparison between DreamerV2 and TransDreamer in Hidden Order Discovery tasks

5.2 Hidden Order Discovery in 3D Object Room

To evaluate this task in a more realistic environment, we also implemented a 3D version of Hidden Order Discovery in Unity (Juliani et al., 2020). The reward structure is the same as the 2D task, but the agent view is a 3D first-person view. Figure 2b shows an overview view of the different configurations and Figure 2c shows the agent's first-person view. Compared to the 2D environment, since the environment is larger, it takes more steps to navigate to the balls, especially in the sparse setting. Therefore, with the 3D environment, we can evaluate how TransDreamer can handle long-term dependency and complex reasoning more clearly.

We implemented 3 settings for this task. The 4-Ball and 5-Ball Dense environment has 4 and 5 balls, separated by at least one ball-length each. The 4-Ball Sparse environment tests longer-term memory by increasing the distance between balls to three ball-lengths so the agent needs to navigate a longer distance between balls. The results are shown in Figure 3. Even if the 3D environment provides more severe partial observability and longer-term dependency than the 2D environment due to its degree of freedom in exploring a larger environment with first-person view, we see that TransDreamer obtains similar outperforming results as we obtained in 2D Object Room. Next, we compare the quality of the trajectories imagined by the TSSM and the RSSM by measuring the generation performance quantitatively and qualitatively.

5.3 World Model - Quantitative Results

We measure the Mean Squared Error (MSE) of the predicted images and the reward prediction accuracy during the action-conditioned generation in the 3D 5-Ball Dense configuration. Even though the image is not directly used for policy training, the quality of the predicted image can serve as a proxy for measuring latent state prediction accuracy. Reward prediction accuracy, on the other hand, is directly related to policy training, and may provide some insights into why TransDreamer performs better than Dreamer in the above environments.

Image Generation. For a fair comparison, we separately trained the TSSM and the RSSM with the same set of trajectories without any policy training. Given a trajectory of 100 timesteps, we measure the generation quality for several different context lengths and measure the per image MSE in the remaining generated steps. This allows us to measure the generation quality given different amounts of historical context and analyze how the different models utilize this context. The reported MSE is for the foreground objects (i.e., the balls), since that is where the most important information for this task is and more than 60% of the gap in overall MSE can be attributed to the foreground, despite the balls only occupying a small portion of the image most of the time (see Appendix Table 4 for overall MSE results). The results are shown in Table 2a. We see that TransDreamer generally achieves lower or comparable MSE when compared with Dreamer. As expected, more steps given in the context results in lower MSE since the agents have more opportunity to see the entire environment before making predictions. In the 4-Ball Sparse setting, the MSE between TransDreamer and Dreamer are

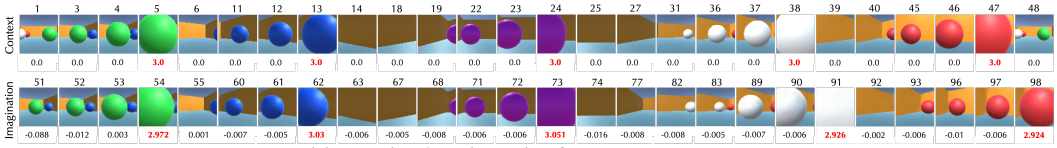
Table 2: World Model Quantitative Comparison

(a) Image Generation Foreground MSE

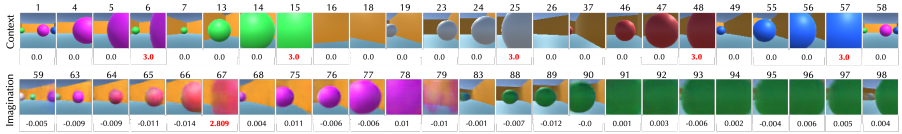
Task	Model	Context 60	70	80
4-Ball Dense	TransDreamer	**211.2**	**133.1**	**69.8**
	DreamerV2	281.9	194.2	110.8
4-Ball Sparse	TransDreamer	**195.5**	**115.2**	**56.8**
	DreamerV2	215.8	138.6	72.4
5-Ball Dense	TransDreamer	**245.2**	**163.1**	**85.0**
	DreamerV2	300.9	217.0	124.9

(b) Reward Prediction Accuracy

Task	Model	Context 60	70	80
4-Ball Dense	Transdreamer	**46.9**	**53.2**	**73.2**
	DreamerV2	28.2	34.6	50.5
4-Ball Sparse	Transdreamer	**32.4**	**36.5**	**48.6**
	DreamerV2	32.0	33.3	42.3
5-Ball Dense	Transdreamer	**17.7**	**18.1**	**32.3**
	DreamerV2	9.8	6.2	15.3



(a) Imagined Trajectories from TransDreamer



(b) Imagined Trajectories from Dreamer

Figure 4: Imagined trajectories comparison between DreamerV2 and TransDreamer

actually very comparable. This may be because when the environment is sparse, the agent sees the foreground objects less frequently.

Reward Prediction. Since the reward is zero for most timesteps and both models can predict zero reward timesteps well (see Appendix Table 5), we focus on the reward prediction accuracy for the nonzero +3 reward timesteps. Since the reward is a continuous value, in order to obtain an accuracy, we classify rewards as positive if they are predicted in the range 3 ± 0.3. The results are shown in Table 2b. We once again see that TransDreamer generally achieves more accurate reward prediction than Dreamer, with longer contexts resulting in higher accuracy. For the 5-Ball Dense environments, however, Dreamer's reward prediction does not improve much as the context increases. This can indicate that Dreamer is not fully taking advantage of the additional context when predicting rewards in these more complex settings. TransDreamer, in contrast, does continue to improve when given more context, showing that the transformer architecture can take advantage of the increased context in making more accurate predictions. A full version including zero-reward prediction accuracy is reported in Appendix Table 5.

5.4 World Model - Qualitative Comparison

In Figure 4, we show the imagined trajectories from TransDreamer and Dreamer in the 5-Ball Dense environment. We provide context timesteps from each agent's trained policy up to when all the balls are collected for the first time. After this, the agent and balls reset to their original positions (frame 48 for TransDreamer and frame 58 for Dreamer). We then imagine the rest of the trajectory up to 100 total steps for each agent. Since the context timesteps contain information about the correct order of balls, ideally the agent would revisit the balls in this order during the imagination timesteps and correctly predict the rewards when the balls are collected. Note that the context frames for Dreamer and TransDreamer are different since they are based on trajectories from their own policies. This is necessary because the world model is trained from the trajectories of each agent's policy. Providing context that is not from the agent's policy would not necessarily be in the training distribution of the respective world model. See Figure 8 in the Appendix for an example of when the same context is given to both agents. Despite this, however, we can still see some clear differences between the quality of the imagination steps as well as the reward predictions.

In particular, TransDreamer is able to predict the appearance of the environment correctly as well as the collection and subsequent disappearance of the balls in frames 54, 62, 73, 90, and 98. It also accurately predicts the reward at these timesteps of around +3 (highlighted in red). For timesteps where there is no reward, it correctly predicts a reward around 0. Dreamer, on the other hand,

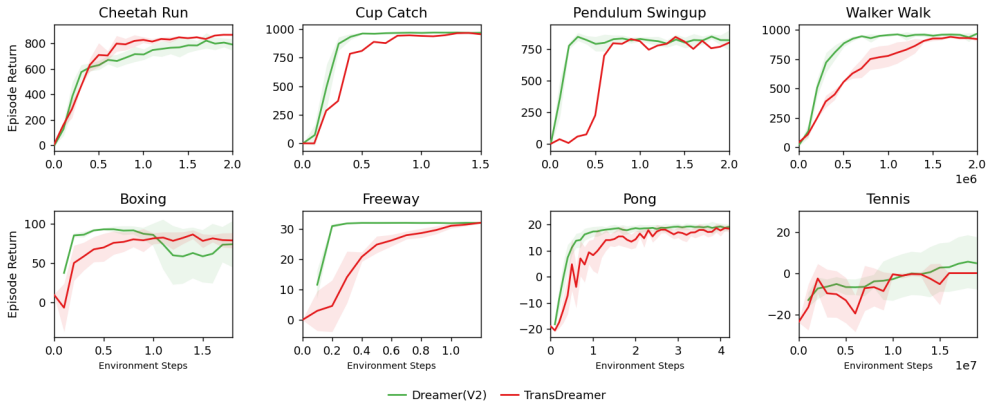


Figure 5: Comparison between Dreamer and TransDreamer on a few DMC (upper row) and Atari tasks (bottom row) for short-term memory test. As expected, TransDreamer converges to a similar return value but slowly.

predicts images that are blurrier than TransDreamer. Furthermore, the imagined trajectories are incorrect. While it does predict the collection of the red ball and the reward in frame 67, this color is incorrect since the first ball should be purple. When it subsequently predicts the collection of the purple ball in frame 78, it is again the wrong color and no reward is predicted. This error seems to compound as the dark green ball it predicts at the end of the trajectory is not even one of the possible colors in the environment. This shows that the quality of the world model is better in TransDreamer than Dreamer, especially in the later steps of imagination where the long-term memory is more important. This can be a reason why TransDreamer outperforms Dreamer in these tasks.

5.5 Short-Term Memory Tasks in DMC and Atari

As the final validation, we perform a sanity check by testing the proposed model on a few simple DMC and Atari tasks. We note that it is expected that these tasks may favor Dreamer over TransDreamer, because solving these tasks does not require long-term and complex memory interactions, but modeling the dynamics of just the last few steps can suffice[1]. Specifically, we expect that RNN-based models can learn *faster* than transformer-based models because the former has the specific inductive bias to focus on the near-term history. Nevertheless, TransDreamer is supposed to converge eventually to an accuracy similar to that of Dreamer, and it is an important step to see whether these expectations are met.

We follow the configurations used by the authors in Dreamer (Hafner et al., 2019) for DMC and DreamerV2 (Hafner et al., 2020) for Atari. The only difference is that Dreamer uses the imagined trajectory from all states sampled from the replay buffer, whereas TransDreamer uses a few randomly selected states. Configurations for Transformer are described in Appendix A.3. As shown in Fig. 5, Dreamer and TransDreamer eventually reach comparable performance as expected, but TransDreamer is slower to saturate in general than Dreamer except for a few tasks. Interestingly, TransDreamer shows slightly better performance and faster convergence for the DMC Cheetah Run.

6 Conclusion

We proposed TransDreamer, a transformer-based MBRL agent, and the Transformer State-Space Model (TSSM), the first transformer-based stochastic world model. TransDreamer shows comparable performance with Dreamer on DMC and Atari tasks that do not require long-term memory, and outperforms Dreamer on Hidden Order Discovery tasks that require long-term complex memory interactions. We also show that image generation and reward prediction of TSSM is better than Dreamer qualitatively and quantitatively. Future work may involve validating our model on more complex tasks such as Crafter (Hafner, 2021).

[1]Some Atari games require long-term memory along with a good exploration policy. We do not choose these games as we do not address the exploration problem in this work.

REFERENCES

Diego Martín Arroyo, Janis Postels, and Federico Tombari. Variational transformer networks for layout generation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 13642–13652. Computer Vision Foundation / IEEE, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Arroyo_Variational_Transformer_Networks_for_Layout_Generation_CVPR_2021_paper.html.

Andrea Banino, Adrià Puigdomènech Badia, Raphael Köster, Martin J Chadwick, Vinicius Zambaldi, Demis Hassabis, Caswell Barry, Matthew Botvinick, Dharshan Kumaran, and Charles Blundell. Memo: A deep network for flexible combination of episodic memories. *arXiv preprint arXiv:2001.10913*, 2020.

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

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

Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551*, 2017.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pp. 1691–1703. PMLR, 2020.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Antonia Creswell, Rishabh Kabra, Christopher Burgess, and Murray Shanahan. Unsupervised object-based transition models for 3d partially observable environments. *CoRR*, abs/2103.04693, 2021. URL https://arxiv.org/abs/2103.04693.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pp. 1096–1105. PMLR, 2018.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

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.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

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.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *arXiv preprint arXiv:2106.06295*, 2021.

Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. *arXiv preprint arXiv:2106.02039*, 2021.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents, 2020.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Nan Rosemary Ke, Anirudh Goyal, Olexa Bilaniuk, Jonathan Binas, Michael C Mozer, Chris Pal, and Yoshua Bengio. Sparse attentive backtracking: Temporal creditassignment through reminding. *arXiv preprint arXiv:1809.03702*, 2018.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

Zhaojiang Lin, Genta Indra Winata, Peng Xu, Zihan Liu, and Pascale Fung. Variational transformers for diverse response generation. *CoRR*, abs/2003.12738, 2020. URL `https://arxiv.org/abs/2003.12738`.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Reset-free lifelong learning with skill-space planning. *arXiv preprint arXiv:2012.03548*, 2020.

Alice Martin, Charles Ollion, Florian Strub, Sylvain Le Corff, and Olivier Pietquin. The monte carlo transformer: a stochastic self-attention model for sequence prediction. *CoRR*, abs/2007.08620, 2020. URL `https://arxiv.org/abs/2007.08620`.

Emilio Parisotto and Ruslan Salakhutdinov. Efficient transformers in reinforcement learning using actor-learner distillation. *arXiv preprint arXiv:2104.01655*, 2021.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pp. 7487–7498. PMLR, 2020.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4052–4061. PMLR, 2018. URL http://proceedings.mlr.press/v80/parmar18a.html.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Sam Ritter, Ryan Faulkner, Laurent Sartran, Adam Santoro, Matt Botvinick, and David Raposo. Rapid task-solving in novel environments. *arXiv preprint arXiv:2006.03662*, 2020.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

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.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Andreas Voskou, Konstantinos P Panousis, Dimitrios Kosmopoulos, Dimitris N Metaxas, and Sotirios Chatzis. Stochastic transformer networks with linear competing units: Application to end-to-end sl translation. *arXiv preprint arXiv:2109.13318*, 2021.

Tianming Wang and Xiaojun Wan. T-CVAE: transformer-based conditioned variational autoencoder for story completion. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 5233–5239. ijcai.org, 2019. doi: 10.24963/ijcai.2019/727. URL https://doi.org/10.24963/ijcai.2019/727.

Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rJgsskrFwH.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. Generative video transformer: Can objects be the words? In *International Conference on Machine Learning*, pp. 11307–11318. PMLR, 2021.

Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. Videogpt: Video generation using VQ-VAE and transformers. *CoRR*, abs/2104.10157, 2021. URL https://arxiv.org/abs/2104.10157.

A APPENDIX

A.1 DREAMERV2

Hafner et al. (2020) makes several additional changes to the framework that are found to improve performance on the Atari environment. First, instead of using a continuous stochastic hidden state, a discrete state is used. Second, straight-through gradients (Bengio et al., 2013) are used to differentiate through the discrete states and actions. Due to the bias induced by the straight-through estimator, REINFORCE gradient or a mixed gradient of REINFORCE and the dynamics backpropagation is used. Lastly, they use KL balancing, separately scaling the prior cross entropy and the posterior entropy in the KL loss.

A.2 TRANSDREAMER LOSS FUNCTION

We optimize the following objective, which is the negative ELBO of the action conditioned model with additional terms for predicting the reward and discount,

$$\mathcal{L}_{\text{TSSM}}(\phi) = \sum_{t=1}^{T} \Bigg(\mathbb{E}_{\prod_{\tau=1}^{t} q_{\phi(z_\tau|x_\tau)}} [-\eta_x \ln p_\phi(x_t|h_t, z_t) - \eta_r \ln p_\phi(r_t|h_t, z_t) - \eta_\gamma \ln p_\phi(\gamma_t|h_t, z_t)] \\ + \mathbb{E}_{\prod_{\tau=1}^{t-1} q_\phi(z_\tau|x_\tau)} \left[D_{\text{KL}} \left[q_\phi(z_t|x_t) \parallel p_\phi(z_t|z_{1:t-1}, a_{1:t-1}) \right] \right] \Bigg).$$

Here, η_x, η_r, and η_γ are hyperparameters used to scale the loss terms. The derivation of the ELBO can be found in the below.

A.2.1 ELBO

The generative model is $p(o_t, z_{1:T}|a_{1:T}) = \prod_t p(o_t|h_t, z_t)p(z_t|z_{1:t-1}, a_{1:t-1})$ where $o_t = (x_t, r_t, \gamma_t)$ and $h_t = f_{\text{transformer}}(z_{1:t-1}, a_{1:t-1})$. By approximating the posterior by $q(z_t|x_t)$, a variational posterior is $q(z_{1:T}|o_{1:T}, a_{1:T}) = \prod_t q(z_t|x_t)$. By the importance weighting and Jensen's inequality, we can write as follows:

$$\begin{aligned}
\ln p(o_{1:T}|a_{1:T}) &= \ln \mathbb{E}_{p(z_{1:T}|o_{1:T}, a_{1:T})} \left[\prod_{t=1}^{T} p(o_t|h_t, z_t) \right] \\
&= \ln \mathbb{E}_{q(z_{1:T}|o_{1:T}, a_{1:T})} \left[\prod_{t=1}^{T} p(o_t|h_t, z_t)p(z_t|z_{1:t-1}, a_{1:t-1})/q(z_t|x_t) \right] \\
&\geq \mathbb{E}_{\prod_{t=1}^{T} q(z_t|x_t)} \left[\sum_{t=1}^{T} \ln p(o_t|h_t, z_t) + \ln p(z_t|z_{1:t-1}, a_{1:t-1}) - \ln q(z_t|x_t) \right] \\
&= \sum_{t=1}^{T} \Bigg(\mathbb{E}_{\prod_{\tau=1}^{t-1} q(z_\tau|x_\tau)} [\ln p(o_t|h_t, z_t)] \\
&\quad - \mathbb{E}_{\prod_{\tau=1}^{t-1} q(z_\tau|x_\tau)} \left[D_{\text{KL}} \left[q(z_t|x_t) \parallel p(z_t|z_{1:t-1}, a_{1:t-1}) \right] \right] \Bigg) \qquad (1) \\
&= \sum_{t=1}^{T} \Bigg(\mathbb{E}_{\prod_{\tau=1}^{t-1} q(z_\tau|x_\tau)} [\ln p(x_t|h_t, z_t) + \ln p(r_t|h_t, z_t) + \ln p(\gamma_t|h_t, z_t)] \\
&\quad - \mathbb{E}_{\prod_{\tau=1}^{t-1} q(z_\tau|x_\tau)} \left[D_{\text{KL}} \left[q(z_t|x_t) \parallel p(z_t|z_{1:t-1}, a_{1:t-1}) \right] \right] \Bigg) \qquad (2)
\end{aligned}$$

where $p(o_t|h_t, z_t) = p(x_t|h_t, z_t)p(r_t|h_t, z_t)p(\gamma_t|h_t, z_t)$.

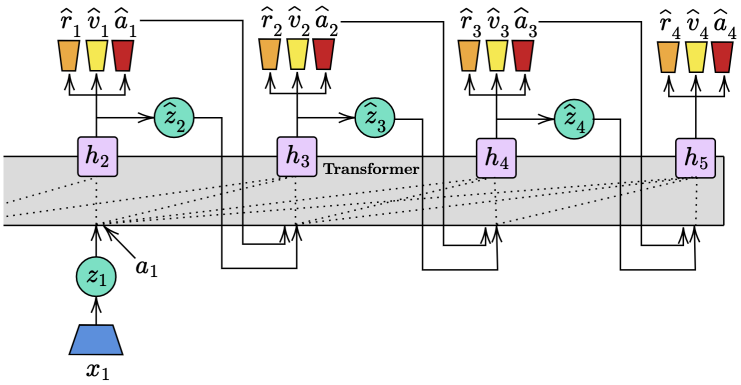


Figure 6: Transfomer-Based Trajectory Rollout for Actor Critic Learning.

A.3 DMC AND ATARI

As written in Sec. 5.5, we used almost identical configurations with Dreamer and DreamerV2 by referring to the configuration file in https://github.com/danijar/dreamerv2 (e.g., action repeat and training World Model and policy every 5 steps for DMC). The one configuration we did modify is the number of imagined trajectories, which is not configurable in Dreamer, but is necessary in TransDreamer because imagining from every state in the batch requires too much computational resources. We control this through a hyperparamter that limits the number of imagined trajectories per training sample. For DMC and Atari, we use 3 imagined trajectories per sample.

Several other hyperparmeters are specific to the TSSM. These include: whether or not to use gating or identity map reording as is done in GTrXL (Parisotto et al., 2020), the number of layers and heads to use for the transformer, the size of the hidden state for the MLP in the transformer, and whether or not to use relational positional embedding (Dai et al., 2019). For DMC and Atari, we generally use 2-layer Transformer (Vaswani et al., 2017) without dropout, gating, or identity map reordering. One exception is for Atari Pong where we did find identity map reordering performed better. We use 10 heads in the Multihead Attention and the dimensions of the hidden state for the MLP and Attention are 200 for DMC and 600 for Atari. These are the same as the dimensions used in the deterministic state in DreamerV2.

A.4 HIDDEN ORDER DISCOVERY

For 2D and 3D Hidden Order Discovery tasks, we measure the model in two aspects, the ability to deal with complex memory-based reasoning and the ability to extract long-term knowledge. Thus we design tasks either with an increased number of objects or the distances between any two objects or both. Specifically, on 2D tasks, we increase the number of balls from 4 to 6, while not changing the distance. The distance here is measured as the number of cells between any two balls. We sum the absolute difference along the x-axis and y-axis as the distance between any two balls. To control the long-term dependency, a threshold of 2 is applied to the distance of balls, i.e. the minimum distance between any two balls should not be less than 2 cells. For 3D tasks, we tested not only the reasoning complexity but also the ability to capture long-term dependency. For reasoning complexity, we compare 5-Ball Dense with 4-Ball Dense. For long term dependency, we compare 4-Ball sparse against 4-Ball Dense. The sparse setting has a larger distance between any two balls. We use the Euclidean distance as a measure of distance. Any two balls have a distance at least 4 units in the sparse setting, while in the dense setting, it is 2 units. 1 unit equals 1 ball size (diameter). Thus, in sparse setting, the distance between any two balls is at least 3 ball-size. For each task, the maximum steps for an episode is set as 100.

We implemented a 6-layer transformer with identity map reordering as TSSM for both 2D and 3D Hidden Order Discovery tasks. During imagination, only one state was randomly sampled as the starting state for imagination. We imagined till the trajectory's max step was reached. Empirically we found concatenating the intermediate output of attention layers together as h_t accelerates the converge speed, so we applied this during experiments. Other hyperparameter configurations are kept the same as DreamerV2 crafter configuration, see https://github.com/danijar/crafter/issues/1

for details. For DreamerV2, we use the same configuration as DreamerV2 for crafter, except that we imagined 30 steps for agent learning.

As explained in the paper, to help train the world model better, we used a prioritized replay buffer for Dreamer and TransDreamer with $\alpha = 0.5$. The sampling probability for each trajectory is set at the return of this trajectory divided by the overall return of the whole data buffer collected till now, thus a trajectory with higher rewards will have a higher chance to be sampled. The rest 50% of the batch are sampled uniformly from the whole data buffer.

Table 3 shows the ratio of successfully completing at least one round of the hidden order on all the 2D and 3D tasks. We deploy the well-trained agent for each task to collect 1000 trajectories and count the ratio in the whole trajectories. As we can see, TransDreamer performs better than Dreamer by this metric. Note that the episode length is limited to 100, and succeeding in this task in 100 steps is difficult. For example, in the 4-Ball case, the chance of guessing the right order is 0.04 (1 over 4!), and the agent needs to start collecting from the first ball when it collects an incorrect ball. Therefore the agent needs to start over again and again during exploration. When increasing the number of balls from 4 to 5, the chance of randomly guessing the order decreases by a factor of 5. We can observe this relation approximately on TransDreamer's performance, $23\% \rightarrow 5\%$, while Dreamer nearly fails.

Table 3: Success Rate for Complete Order Visitation

Task	2D Object Room			3D Object Room		
	4-Ball	5-Ball	6-Ball	4-Ball Dense	4-Ball Sparse	5-Ball Dense
TransDreamer	**23%**	**5%**	**1%**	**18%**	**11%**	**4%**
DreamerV2	7%	0%	0%	10%	1%	0 %

A.4.1 Full Quantitative Results

To compute the foreground MSE, we use Unity (Juliani et al., 2020) to render a foreground image, Figure 7, and from the foreground image, we infer a binary foreground mask to filter out the background from the predicted image. The full MSE result is reported in Table 4. As can be seen, more than half of the overall MSE gap between TransDreamer and Dreamer happens in the foreground. For example, in the 4-Ball Dense, 60 context setting, the overall MSE gap between TransDreamer and Dreamer is 119.8, while 70.7 of the error occurs in the foreground.

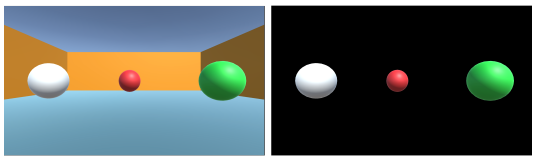


Figure 7: Image from Unity Foreground Camera

Table 5 shows the reward prediction accuracy on both zero-reward and nonzero-reward timesteps for the 3D tasks. As mentioned in the paper, to measure prediction accuracy for +3 reward case, we classify it by labeling 3 ± 0.3 as positive. For 0 reward case, we classify it by labeling ± 0.01 as positive. We can see that TransDreamer outperforms Dreamer by a large gap on nonzero-reward in the 4-Ball Dense and the 5-Ball Dense. Both models perform well generally on zero-reward. In 4-Ball Sparse setting, The gap is smaller, we hypothesis that it is because in the sparse setting, the foreground balls are seen less frequently.

Table 4: Image Generation MSE

Task	Model	60 contexts / 40 targets		70 contexts / 30 targets		80 contexts / 20 targets	
		Overall	Foreground	Overall	Foreground	Overall	Foreground
4-Ball Dense	TransDreamer	**458.0**	**211.2**	**281.9**	**133.1**	**146.0**	**69.8**
	DreamerV2	577.8	281.9	380.0	194.2	206.2	110.8
4-Ball Sparse	TransDreamer	**448.8**	**195.5**	**261.4**	**115.2**	**128.1**	**56.8**
	DreamerV2	462.6	215.8	279.7	138.6	145.1	72.4
5-Ball Dense	TransDreamer	**516.0**	**245.2**	**329.9**	**163.1**	**167.4**	**85.0**
	DreamerV2	605.1	300.9	413.8	217.0	231.6	124.9

Table 5: Reward Prediction Accuracy

Task	Model	60 contexts / 40 targets		70 contexts / 30 targets		80 contexts / 20 targets	
		Zero	Non-zero	Zero	Non-zero	Zero	Non-zero
4-Ball Dense	Transdreamer	**94.9**	**46.9**	**94.7**	**53.2**	**95.4**	**73.2**
	DreamerV2	93.7	28.2	93.6	34.6	94.2	50.5
4-Ball Sparse	Transdreamer	**96.4**	**32.4**	96.0	**36.5**	**96.6**	**48.6**
	DreamerV2	95.6	32.0	**96.2**	33.3	95.5	42.3
5-Ball Dense	Transdreamer	**92.5**	**17.7**	**93.2**	**18.1**	**93.3**	**32.35**
	DreamerV2	91.1	9.8	92.3	6.2	92.4	15.3

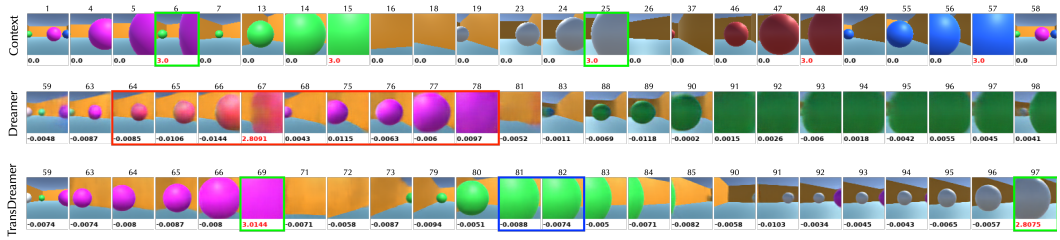


Figure 8: Imagined trajectories comparison between DreamerV2 and TransDreamer given same context

A.5 World Model Imagination with Same Context

Different from Figure 4, in Figure 8, we illustrated imagined trajectories from TransDreamer and Dreamer given the same contexts. This 5-Ball Dense sample is collected from Dreamer's agent learning process, so for TransDreamer, it is an out of distribution sample. This is the same context given to Dreamer in Figure 4. Despite being an out of distribution sample, TransDreamer can still correctly imagine the balls and predicts rewards for the purple and white balls (Green box) correctly. However, it does make a mistake predicting the reward for the green ball (blue box). Nevertheless, even in this setting, the quality of imagination in TransDreamer is better than Dreamer.