Understanding the Effects of Discrete Representations in Model-Based Reinforcement Learning

An analysis on the effects of categorical latent space world models on the MinAtar Environment

Mihai Mitrea[1]
Supervisor(s): Frans A. Oliehoek[1], **Jinke He**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Mihai Mitrea
Final project course: CSE3000 Research Project
Thesis committee: Frans A. Oliehoek, Jinke He, Mathijs de Weerdt

Abstract

While model-free reinforcement learning (MFRL) approaches have been shown effective at solving a diverse range of environments, recent developments in model-based reinforcement learning (MBRL) have shown that it is possible to leverage its increased sample efficiency and generalisation abilities to solve highly complex tasks with fewer resources and environment interactions. The introduction of discrete latent states through categorical distributions allowed DreamerV2, a MBRL approach, to surpass the state-of-the-art MFRL Rainbow algorithm on the Arcade Learning Environment. Despite the successes of this approach, it is not yet understood why discretization improves performance. This paper investigates how the discretization of the latent space through categorical distribution affects planning performance in a deterministic environment. Further investigations are conducted on the model's generalization abilities and the impact of the latent space's shape on performance. By using a dataset of experiences instead of directly interacting with the environment, the models are trained in an offline setting. Results show that the discrete world model underperforms compared to a continuous latent space model while being significantly harder to train. Further investigations concluded that the number of categorical distributions has a high influence on performance and that in the considered setting the discrete world model can generalize better than the continuous baseline but it does so by sacrificing small gains in important metrics.

1 Introduction

Reinforcement learning agents, like humans, learn by interacting with their environment. To be successful, the agents need to understand the rules and dynamics that govern the outcome of their actions [1]. To achieve this, two approaches can be used: Model-Free Reinforcement Learning (MFRL) and Model-Based Reinforcement Learning (MBRL). Model-free approaches aim to choose the best action from their understanding of the current state of the world. On the other hand, model-based approaches integrate planning, the idea of "trying things in your head" [2], and an internal world model to look into the future and select the best option [3]. It is widely regarded that in structured environments, MBRL achieves a higher data efficiency through its ability to generalize [4, 5], requiring less training time and resources.

Generally, such methods have not been able to compete with state-of-the-art model-free approaches [5, 6], however, recent developments have shown the first promising signs that MBRL can match or even outperform their counterparts. Algorithms such as AlphaGo Zero [7], MuZero [8], and DreamerV2 [6] have managed to achieve superhuman performance on notoriously difficult tasks, including Go, Chess, and Atari.

For high-dimensional observations, it is useful to extract low-dimensional and meaningful representations of the world [9, 10], commonly known as latent space representations, to abstract away unnecessary details. One way of doing this is through autoencoders, a specific architecture of neural networks trained to encode and reconstruct the input while maintaining its properties [11]. Subsequently, the agent can use these representations to predict the outcome of their actions [10, 12, 6, 5, 13].

Recently, several attempts have been made at utilising different representation models for the latent space encoding of the world. Most prominent, are stochastic models which encode a given input as a multivariate Gaussian [10, 12, 13, 5] or Categorical distribution [6] and subsequently sample the latent state representation. Furthermore, promising results have been achieved by discretising such latent spaces [6, 14, 13]. Despite the successes of

these approaches, there is a lack of evidence explaining why such representations improve the model's ability to understand the dynamics of the environment.

This paper investigates *how the discretization of the latent space through categorical distributions affects the world model's performance when planning in a deterministic environment and an offline setting.* To guide this research the following sub-questions were used:

- How does a world model using discrete embeddings compare to a one using continuous latent states?
- How does the number and size of the categorical distributions affect the discrete model's performance?
- How does the discretization of the latent space affect the model's ability to generalize?

All investigations were conducted on a modified version of Breakout from the MinAtar learning environment [15] with terminating episodes, deterministic transitions, full observability, and no redundant actions. By using a dataset of experiences instead of directly interacting with the environment, the models are trained in an offline setting [16]. The dataset is obtained by a Deep Q-Network (DQN) agent [17] learning to play optimally.

Results show that the discrete world model underperforms compared to a continuous latent space model while being significantly harder to train. When considering how the composition of the latent space affects the model's performance, this study finds that the number of categorical distributions is the main driver behind performance rather than their size. Further investigations concluded that in the considered setting the discrete world model can generalize better than the continuous baseline but it does so by sacrificing small gains in important metrics.

2 Related Works

World Models [10] The paper describes an approach that uses images to predict the next latent space without having to reconstruct the input. To this end, they use a Variational Autoencoder (VAE) [18] to create multidimensional Gaussian distributions from which the latent space is sampled. To operate in their partially observable environment the model incorporates an LSTM which, over time, integrates information about the environment.

Simulated Policy Learning (SimPLe) [5] The authors introduce a new architecture that is successfully applied on a multitude of Atari environments. The presented architecture serves as a "predictive model" to train a Proximal Policy Optimization (PPO) agent to operate optimally in the real environment. Unlike other approaches, SimPLe cannot predict the next frame while remaining in latent space, requiring the reconstruction of the input data even during inference. This approach can learn the dynamics of both stochastic and deterministic environments.

A novel feature introduced by this approach is the discretization into bits of the latent variables for the stochastic case. Another novelty introduced by the paper is that training is restricted to 100K interactions with the environment, demonstrating the high sample efficiency of their approach.

Discrete World Models through VQ-VAE [13] The proposed approach builds on top of World Models, SimPLe and DreamerV2 by integrating the latent space prediction approach presented by World Models [10], the limited number of interactions with the environment of SimPLe [5], and the idea of discretizing the latent space proposed by DreamerV2 [6]. The proposed architecture utilizes a Vector Quantized Variational Autoencoder [19] to discretize the latent space, a different approach than the one used by [6, 14]. The proposed model achieves a similar performance when compared to SimPLe while maintaining a much smaller architecture and the ability to act entirely in latent space.

DreamerV1, DreamerV2, and DreamerV3 [12, 6, 14] The Dreamer models present an architecture that follows the idea of predicting in latent space while using varying embedding techniques. At the core of the model stands the Recurrent State-Space Model (RSSM) introduced by PlaNet [20]. This architecture allows the world model to operate in partially observable environments through an internal state containing past information.

Initially, DreamerV1 used Gaussian latents to represent the environment states but in subsequent iterations (V2 and V3) the embeddings would be represented by one-hot samples from categorical distributions. This novel approach significantly improved the model's performance, allowing it to achieve superhuman performance on the Atari benchmark [21]. However, this discretization technique altered the loss landscape to require domain-specific hyperparameter tuning in the form of weights for some components of the loss function. The authors explain in DreamerV3's paper that complex environments require higher weights and that simpler domains would require lower ones [14].

DreamerV3 stabilized the model by introducing domain-agnostic techniques such that it can be used for multiple environments without extensive fine-tuning, a feat proven by collecting diamonds in the more challenging MineRL environment [22]. One proposed technique is the introduction of a lower bound on the previously weighted components of the loss function to prevent degenerate solutions.

All iterations of the algorithm are trained online on a relatively high amount of data consisting of 50M frames and it is unclear how the introduced robustness techniques would fair in an offline environment under a low data regime. Furthermore, the authors state that it is not yet clear why the discretization of the latent space improves performance and provide four hypotheses.

3 World Model Description

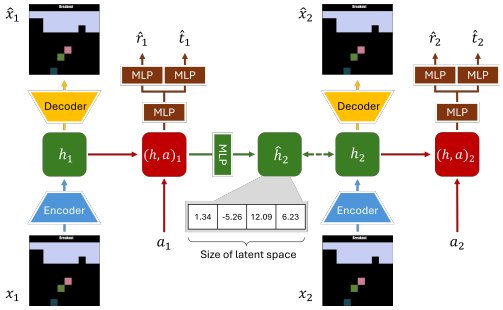
This section introduces the architecture behind the world model and provides an overview of the continuous and discrete latent space representations. The training and inference procedures are also described.

3.1 Architecture

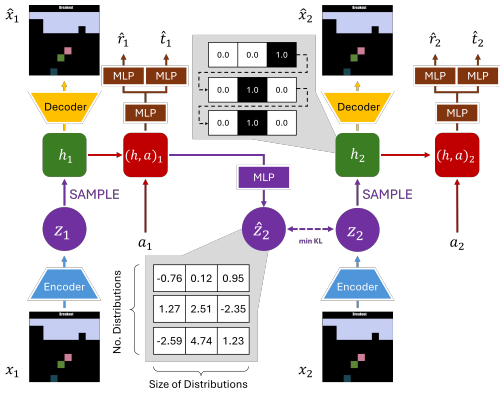
The architecture as a whole is heavily influenced by that of the Dreamer models [12, 6, 14] featuring prediction in latent space without the need for reconstruction. The most significant change is the replacement of the Dreamer's RSSM with a simple dynamics model suited for the fully observable setting.

The world model is structured into three components. The representation component encodes the input frame x_i into its latent space representation h_i. The reconstruction network uses this embedding to decode the latent state into an approximate frame $\hat{x}_i$ which

resembles, as closely as possible, the original input. Both components are implemented as Convolutional Neural Networks (CNN) [23] with the representation model also using a Multi-Layer Perceptron (MLP). The dynamics component, consisting of multiple MLPs, uses an action a_i and the latent embedding h_i to predict the reward r_i, termination t_i, and latent state of the transition's next frame $\hat{h}_{i+1}$. All model components and their relations are presented in Figure 1.



(a) Continuous model architecture. The representation and dynamics models directly learn the continuous latent representations h_i and $\hat{h}_{i+1}$.



(b) Discrete model architecture. The representation and dynamics models learn the log probabilities of multiple categorical distributions in the form of z_i and $\hat{z}_{i+1}$. The latent embedding h_i is obtained by sampling all the distributions in z_i and one-hot encoding the results.

Figure 1: Continuous and Discrete world models architectures. The input image x_i is encoded into a latent space representation h_i through an encoder network and then reconstructed into $\hat{x}_i$. This latent space representation is also used, together with an action a_i, by the dynamics model to predict the reward $\hat{r}_i$, termination $\hat{t}_i$, and the latent state representation of the next frame $\hat{h}_{i+1}$.

3.2 Latent Space Types

To allow for continuous and discrete latent representations, the embedding h_i and its approximation $\hat{h}_i$ are always represented as a vector of floating point numbers. This allows for a common interface between all components of the world model regardless of the type of representation used.

In the continuous case, the output of the representation and dynamics components will directly be the latent embedding of a frame, as shown in Figure 1a. On the other hand, when considering a discrete representation, the output of both components will be the log probabilities of multiple categorical distributions of the same size denoted by z_i in Figure 1b. One sample is drawn from each distribution and the results are converted into arrays of 1s and 0s where the only non-zero term corresponds to the index of the drawn class, a technique called one-hot encoding. The resulting binary array represents the latent embedding h_i.

3.3 Training Procedure

The world model and its components are optimized using the loss function shown in Equation 1 with the Adam optimizer [24]. The function's three components ensure the model learns meaningful latent representations while enabling transitions through the dynamics model.

To deal with the inability of backpropagating gradients through samples obtained from a categorical distribution, the discrete model is trained using the Gumber-Softmax Estimator [25] with constant temperature $\tau = 1$. This distribution estimates the categorical distribution while also one-hot encoding samples.

$$\mathcal{L}(\phi) = \mathcal{L}_{pred}(\phi) + \beta_{dyn}\mathcal{L}_{dyn}(\phi) + \beta_{rep}\mathcal{L}_{rep}(\phi) \tag{1}$$

The prediction loss $\mathcal{L}_{pred}$, shown in Equation 2, trains the model towards latent space representations which can be used to reconstruct the input image, predict the transition's reward, and determine whether the episode terminates. Mean Squared Error (MSE) is used for the reconstructed image and transition reward, while Binary Cross Entropy (BCE) with logits is used for the episode termination.

$$\mathcal{L}_{pred}(\phi) = \text{MSE}(\hat{x_i}, x_i) + \text{MSE}(\hat{r_i}, r_i) + \text{BCE}(\hat{t_i}, t_i) \tag{2}$$

To effectively predict the embedding of subsequent states, the dynamics network is optimised using $\mathcal{L}_{dyn}$, shown in Equation 3. When the latent space is continuous, the dynamics loss is represented by the MSE between the predicted embedding $\hat{h}_{i+1}$ and the actual embedding h_{i+1}. Conversely, when the latent space is discrete, the dynamics loss is represented by the Kullback-Leibler (KL) divergence between the predicted distribution $\hat{z}_{i+1}$ and z_{i+1}.

$$\mathcal{L}_{dyn}(\phi) = \begin{cases} \text{MSE}(\text{sg}(\hat{l_i}), l_i) & \text{, continuous latent space} \\ \max(kl\_clip, \text{KL}(\text{sg}(\hat{z_i}), z_i)) & \text{, discrete latent space} \end{cases} \tag{3}$$

While the prediction loss function $\mathcal{L}_{pred}$ ensures that the model learns meaningful representations, they may be too complex for the dynamics network to predict subsequent states effectively. With this in mind, the representation loss is introduced to simplify the actual encoding and to bring it closer to the predicted one. As can be seen in Equation 4, the only difference from $\mathcal{L}_{dyn}$ is the placement of the stop gradient operator $\text{sg}(\cdot)$, informing the optimizer about which gradients to consider.

$$\mathcal{L}_{rep}(\phi) = \begin{cases} \text{MSE}(\hat{l_i}, \text{sg}(l_i)) & \text{, continuous latent space} \\ \max(kl\_clip, \text{KL}(\hat{z_i}, \text{sg}(z_i))) & \text{, discrete latent space} \end{cases} \tag{4}$$

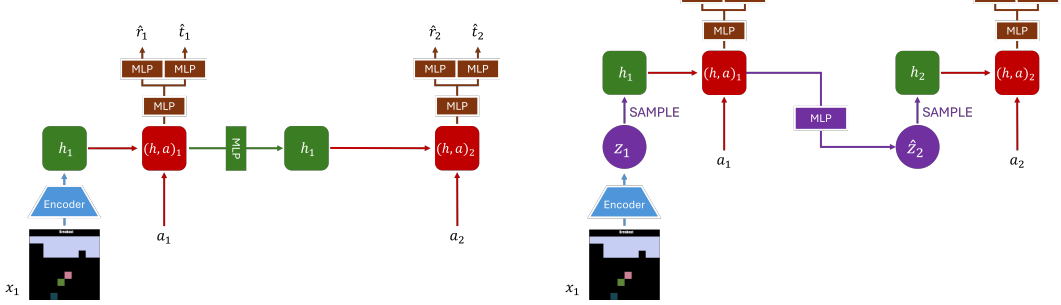
The introduction of weights β_{dyn} and β_{rep} aims to balance the embedding's level of detail with the difficulty of learning the transition function. Thus, high weights would encourage a simple embedding which is easy to predict but not detailed enough for reconstruction and low weights would produce complex latent states which are too hard for the dynamics model to learn. Either one of those cases describes a degenerate solution.

The more common degenerate solution is the one in which the latent embedding is simple and the dynamics and representation losses are minimized. To stabilize the training and avoid such scenarios, the $\mathcal{L}_{dyn}$ and $\mathcal{L}_{rep}$ losses are clipped to a value kl_clip, allowing the model to focus on $\mathcal{L}_{pred}$ once the other two losses are sufficiently minimized. The continuous model does not require clipping because of the simpler nature of its embedding and the different values yielded by the MSE loss function.

3.4 Inference

The model's ability to predict the outcome of actions enables the usage of planning algorithms to determine the agent's next move. The reconstruction of predicted frames is not required because of the dynamics network's ability to work completely in latent space. Figure 2 describes the architecture of both world models.

During inference, the representation model encodes the latest frame x_i into a latent embedding h_i. This representation is used by the dynamics network, along with an action a'_i to predict reward $\hat{r'}_i$, termination $\hat{t'}_i$, and the latent representation of the next state $\hat{h'}_{i+1}$. Because future game states are unknown, the predicted embedding is used as an approximation of the true embedding h'_{i+1}.



(a) Continuous model during inference. The predicted latent state is used as the actual latent state of the next frame given an action.

(b) Discrete model architecture. The predicted categorical distribution is sampled to obtain the latent state of the future frame considering the input action.

Figure 2: Continuous and Discrete world models during inference. The current frame x_i is encoded into a latent space representation h_i. This latent space representation is used by the dynamics model, along with an action a'_i, to predict the reward $\hat{r'}_i$, termination $\hat{t'}_i$, and the latent state representation of the next frame $\hat{h'}_{i+1}$. Unlike during training, the next frame is unknown and the model uses a recurrent logic to predict $\hat{h'}_{j+1}$ from $\hat{h}_j$ given an action.

4 Experimental Setup

This section presents the environment of choice as well as the modifications applied to it, the data-gathering process for the world model training, and the performance metrics used to evaluate models.

4.1 Learning Environment

Although all the previously proposed algorithms were trained and evaluated on the widely used Arcade Learning Environment (ALE) [21], this study analyses the effects of discrete latent representations on the MinAtar environment [15] due to its simpler yet faithful implementation of the ALE. Given the study's comparative nature, a fair correlation between the type of latent space and performance can be drawn, no matter the environment, as long as both approaches are trained and evaluated equally.

Due to time constraints, this study will only consider the Breakout game of MinAtar with fully deterministic transitions. The environment was modified to remove unnecessary actions (fire and vertical movements) and infinite-length episodes to further simplify the game for increased experimental speed. A detailed description of all the modifications can be found in Appendix A.

4.2 Training Dataset

The full training dataset [1] contains 5 million frame, action, reward, and termination tuples obtained by a DQN agent learning to play the game using an ϵ-greedy strategy. Additionally, two intermediate datasets, containing the first 10,000 and 30,000 episodes of the learning process, are used throughout this study for reduced computational costs on more expensive experiments. An analysis on the properties of these datasets can be found in Appendix B.

To assess the model's ability to generalize, a dataset containing transitions present in a separate optimal policy dataset but not in the training one will be compiled. A transition is defined as a tuple of frames, action, subsequent frame, termination, and reward with equality being defined by the two frames and action. Due to the high computational cost of constructing this dataset, only 100,000 transitions are being considered.

4.3 Performance Metrics

A model is assessed through its ability to be used by a Monte Carlo Tree Search (MCTS) algorithm to plan while interacting with the real environment. In each step, the MCTS will encode the last frame and use the model's inference mode to plan and decide its next action. The score of the assessment is represented by the mean of rewards over 100 episodes.

The performance of a world model is defined by the mean and standard error of the mean (SEM) over the scores of three independent models trained on different seeds.

A model's ability to generalize will be defined by its loss when predicting the future frame, reward, and termination of a transition given an initial image and action. The loss is computed as the mean of all unseen transitions using the prediction loss function $\mathcal{L}_{pred}$ as defined in section 3.

5 Experiments

This section discusses the goal, approach, and results of each experiment. First, the difficulties of training a world model with discrete latent embeddings are described, followed by a performance comparison with the continuous baseline model. Furthermore, investigations are conducted into how the latent space's size affects the model's performance and whether each model's ability to generalize can be linked to its general performance.

5.1 Hyperparameters, Learning Curves, and Training Instabilities

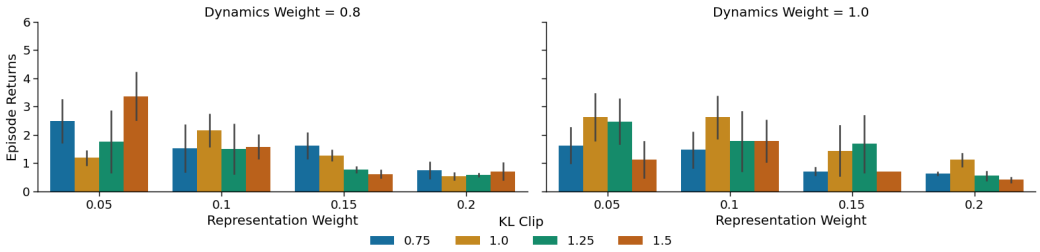
This section discusses the instabilities observed during fine-tuning and how slight variations in the loss function's hyperparameters can drastically affect the model's learning process.

Fine-tuning was performed using three seeds for the discrete model and, due to resource limitations, two seeds for the continuous one. To accommodate the simplified environment considered in this study, the dimensions of the discrete latent state were 16x16 and 128 for

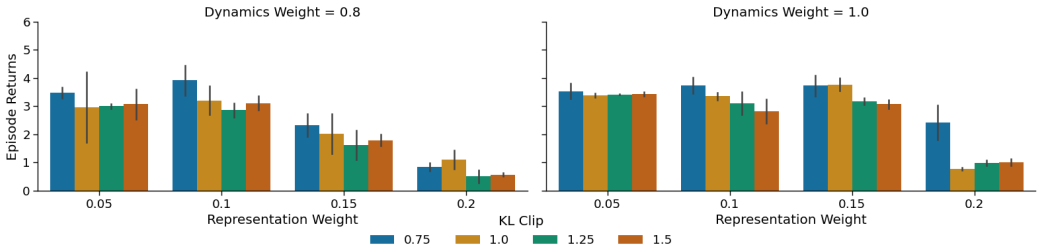
[1] The dataset can be found under the name "Bachelor Thesis Mihai Mitrea Dataset at " 4TU.ResearchData

the discrete and continuous models, respectively. All training runs were performed on the 30k episode dataset due to its similarity to the full dataset, however, the 10k dataset was also considered for a comparative nature due to its limited number of late-game transitions.

In the discrete case, the loss weight values were inspired by the ones proposed by Dreamer [6, 14], while for the continuous model a broader range was investigated. Furthermore, the representation loss weight was never higher than that of the dynamics one, which, itself was not allowed to exceed 1. Partial results for dynamics weights of 0.8 and 1.0 for the discrete world model can be seen in Figure 3 with the full results available in Appendix C.



(a) Partial results for the discrete model on the 10k dataset.

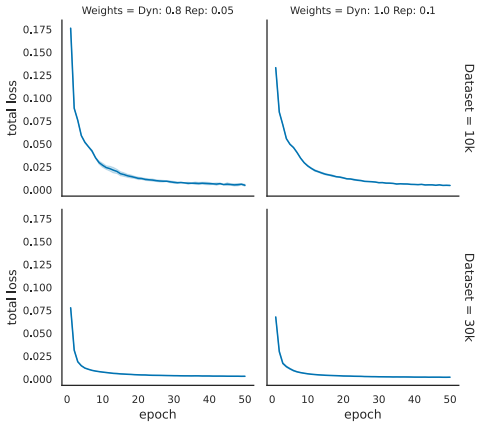


(b) Partial results for the discrete model on the 30k dataset.

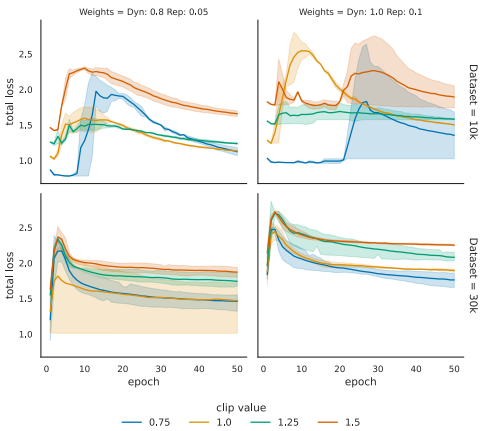
Figure 3: Partial results of the hyperparameter tuning for the discrete world model using the 10k and 30k datasets. Only dynamics weights of 0.8 and 1.0 out of 0.7, 0.8, 0.9, and 1.0 are shown. Barplots show the mean and standard error of means (SEM) of three independent models trained on different seeds for each tuple of hyperparameters.

The results on the 30k dataset indicate that a large value for the representation loss's weight guides the learning process towards simple embeddings which cannot be used effectively to predict future states. Furthermore, a trend where lower KL clip values allow for better-performing models can be observed for almost all weight values. On the smaller dataset, this trend does not seem to hold as strongly, indicating that in a low-data scenario, higher clip values might be needed to avoid a degenerate solution.

Figure 4 compares the learning curves of the continuous and discrete world models to understand why different clip values matter for discrete embeddings when considering multiple datasets.



(a) Total loss of the continuous world model when trained for 50 epochs on the 10k and 30k datasets using two different hyperparameter settings.



(b) Total loss of the discrete world model when trained for 50 epochs on the 10k and 30k datasets using two different hyperparameter settings.

Figure 4: Total losses of the continuous and discrete world models when trained on the 10k and 30k datasets. The line plots show the mean and 95% confidence interval of three independent models trained on different seeds.

By analyzing the learning curves of the discrete and continuous world models under the same conditions it becomes clear that the discretization of the latent space destabilizes the training process.

The continuous model's loss gradually decreases and levels off. For the discrete case, however, the losses are characterized by an initial low value, situated at or around the KL clip, followed by a sudden and drastic increase and a slow decrease. Furthermore, in low-data scenarios, these spikes appear to not always happen at the beginning of the training process, if they happen at all, a behavior which is less frequent when operating on larger datasets.

The behavior of the discrete model could represent its tendency to minimize the dynamics and representation losses at the expense of its ability to predict rewards and episode terminations. Breaking down the loss function into its components, it has been observed that these sudden spikes are mainly influenced by increases in the dynamics and representation losses, signaling a transition between simple and easy to predict embeddings towards more detailed ones.

Therefore, the need for a higher clip value when operating in a low-data regime could be explained by the relatively smaller increase in the loss function required for the transition to happen. Conversely, when there is sufficient data to force this transition, lower clip values may allow the model to achieve a better understanding of the game's dynamics.

5.2 Performance Comparison

This experiment aims to compare the performance of the discrete latent space world model to that of the continuous baseline and to answer the following sub-question: "How does a world model using discrete embeddings compare to a continuous approach?"

To ensure a fair comparison, both model types were fine tuned for the considered domain. This process was performed in two phases: finding the right values for the loss function and determining the optimal size for the latent space. This method was chosen due to resource limitations making the exploration of all combinations of embedding sizes and loss function parameters infeasible.

The first phase and its challenges are described in subsection 5.1. To find the right latent space size two seeds were used to train individual models. For the continuous approach, sizes of 64, 128, and 256 were considered. For the discrete model, the size and number of the categorical distributions could be one of the following: 8, 16, 24, 32, and 40.

As shown in Figure 5, results indicate that the discrete world model is outperformed by the continuous one, with means of approximately 4 and 6.5, respectively.

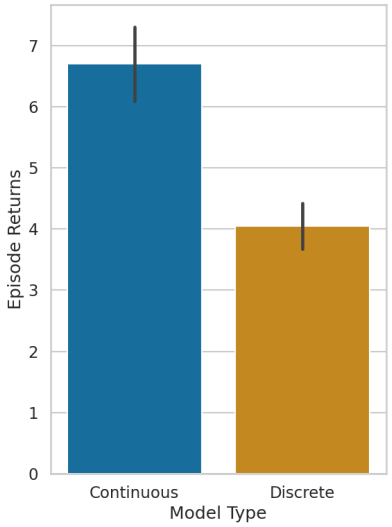


Figure 5: Results comparing the continuous and discrete world models' performance.

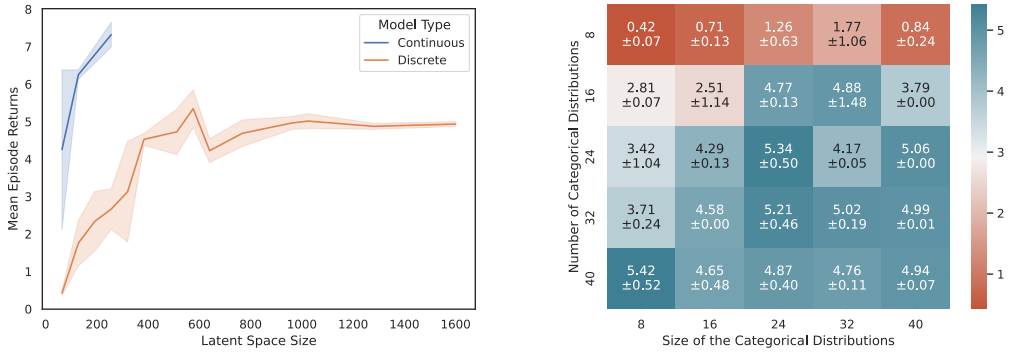
It is not clear why the discrete world models is worse than the baseline, but one hypothesis could be that the stochastic nature of the approach might add an extra layer of complexity which is not necessary, or even potentially harmful, in a deterministic environment.

5.3 Effects of Embedding Size on Performance

This experiment aims to better understand how the size and shape of the latent space affect both models' performance and provide insights into why the continuous model outperforms the discrete one. Furthermore, this experiment also aims to answer the following sub-question: "How do the number and size of the categorical distributions affect the discrete model's performance?" Due to resource limitations, the focus of this experiment is on the discrete model with a more limited investigation conducted on the continuous one.

All investigations were performed on the 30k dataset using two seeds for both the discrete and continuous world models. For the discrete approach, latent space sizes of 64, 128, and 256 were considered. In the continuous case, the size and number of categorical distributions could be 8, 16, 24, 32, or 40. Larger values were not considered due to their higher resource requirements. All other hyperparameters are consistent with the ones identified during the hyperparameter tuning process.

Figure 6 provides a comparison on the performance of both types of models as well as an analysis on the outcomes of different shapes for the discrete model's embedding.



(a) Means and SEM of both types of world models when considering varying latent space sizes.

(b) Heatmap showing the mean and SEM of the discrete model based on the size and number of categorical distributions.

Figure 6: Results showcasing the effect of the latent space's size on model performance.

Based on the results shown in Figure 6a, it can be concluded that the continuous model dominates the discrete one. Even when considering different sizes, the continuous model can achieve comparable results to an optimal discrete model while using a significantly smaller embedding size.

Another observation is that the discrete model seems to require a specific latent space size to achieve optimal results with further increases not correlating with better performance. This has not been observed for the continuous model, but a theoretical upper bound corresponding to the size of the input should exist.

Figure 6b shows that the number of categorical distributions is much more important to the model's performance than their size. However, when considering a small number of distributions, larger sizes can somewhat improve performance, although not to the same degree.

The need for a minimum number of distributions could reflect the environment's characteristics and therefore the identified values might differ on other Atari games. In more complex games it could be the case that a larger number of distributions would be needed to achieve good performance.

5.4 Correlation Between Generalization and Performance

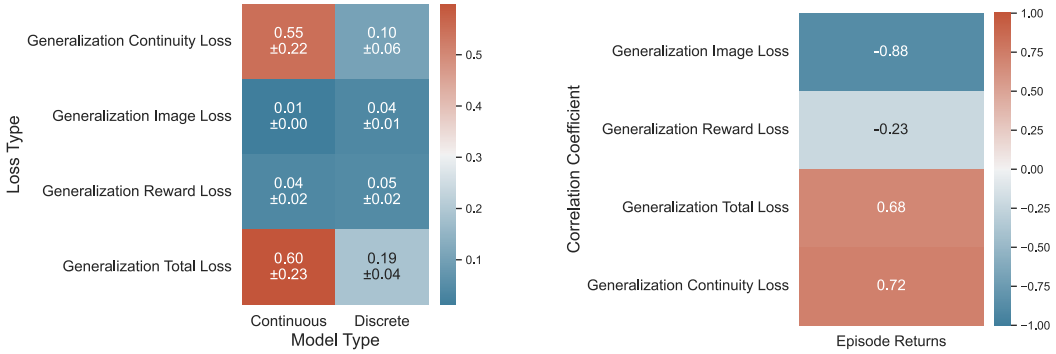
This experiment aims to understand how each model deals with unseen transitions and whether the ability to generalize is beneficial during planning. At the core of this experiment is the following question: "How does the discretization of the latent space affect the model's ability to generalize?" Because of that, the main focus will be on the discrete model, with the baseline used as a reference.

To conduct this experiment, for each type of latent embedding, three independent models with optimal hyperparameters and latent space sizes were trained on the 10k dataset. Their planning performance was evaluated according to the procedure established in section 4 whilst their generalization performance was assessed using a dataset containing unseen transitions.

A model's generalization performance is defined by the $\mathcal{L}_{pred}$ loss of the predicted reward, termination, and next frame given an initial frame and action. Split by components, the

image reconstruction loss aims to asses whether the model is able to correctly predict the next state of the game, while the other two losses aim to establish whether the model can correctly predict rewards and episode terminations.

The results of this experiment can be observed in Figure 7. The mean of the losses over the independent models are presented in Figure 7a, while a statistical correlation between each component of the loss and the episode returns is presented in Figure 7b.



(a) Generalization losses for both model types when assessed on a dataset of 100k unseen transitions following an optimal playing strategy.

(b) Correlation between generalization losses and episode returns. A positive correlation is considered undesirable when aiming to minimize losses.

Figure 7: Results showcasing the generalization losses and the correlation between each loss component and episode returns for models trained on the 10k dataset.

Given that the continuous model outperforms the discrete one despite a larger total generalization loss, it seems that certain components of $\mathcal{L}_{pred}$ are more important in this environment than others. Considering the by-component losses and their correlation with episode returns, the image generalization loss (encompassing the ability to correctly predict the next frame) and reward generalization loss are much more important than the continuity generalization loss.

An important observation would be that the generalization continuity loss and the total loss have a positive correlation with episode returns, meaning that an increase in the loss would lead to higher returns. This could be the case because the discrete model chose to sacrifice smaller gains in more important loss components in favor of larger gains in less important ones.

Considering that the MCTS decides the agent's moves based on the returns of a trajectory, it could be hypothesized that the correct prediction of rewards and subsequent game states may take precedence over that of episode terminations.

6 Discussion

This study analyzed the impact of discrete latent representation in an offline setting and a deterministic environment from multiple points of view.

First of all, the overall performance of world model using discrete latent embeddings was compared to one using continuous latent states, showing that in the considered setting, the

discrete model is outperform by the continuous one. Furthermore, the discretization of the latent space has been shown to introduce severe instabilities in the training procedure, which become more pronounced in a low-data regime. One hypothesis behind the underperformance of the discrete model would be the mismatch between the stochastic nature of the algorithm and the deterministic one of the environment.

Second of all, this study found that the main driver behind the discrete model's performance is the number of categorical distributions and not their size. However, it is not entirely unfounded to believe that certain environments might be better described by a smaller number of larger distributions. Examples of such environments might be ones with few elements which may have multiple possible states (e.g. the paddle from Atari Breakout which may take multiple positions or the cars from Atari Freeway). Results also show that there is an upper limit to the size of the latent embedding, after which no further performance gains can be achieved.

Lastly, it seems that the discrete model is overall better at generalizing to unseen transitions, however, it does so by sacrificing small gains in important metrics. This might also be one of the reasons behind the discrete model's under-performance.

The results of this analysis suggest that two of the original hypotheses proposed by [6] regarding why a discrete latent space yields better results may be true. First, the underperformance of the discrete model in a deterministic setting could suggest that the flexibility offered by categorical distributions over Gaussians in stochastic environments is the reason behind DreamerV2's increased performance. Second, the overemphasis on minimizing the continuity component of the loss function could suggest that categorical variables are better at describing "the non-smooth aspects of Atari games" and thus easier to optimize. In this specific case, the world model could use one of the categoricals to represent whether the ball is at the bottom of the screen (done) or not (not done).

An important consideration that has to be made is the offline setting of the study and the influence that the dataset has on the model's performance. It can be observed that the discrete model performs better on the 30k dataset (see Figure 6b) than on the full dataset (see Figure 5) despite the former being wholly included in the latter.

7 Responsible Research

This section discusses the reproducibility of the study and discusses potential ethical considerations involving MBRL. A description of the usage of Large Language Models throughout this study is presented in Appendix E.

7.1 Reproducibility

Ensuring the reproducibility of results is a crucial aspect of research that needs thorough consideration and appropriate design choices when dealing with the inherent randomness of Machine Learning algorithms. Studies have shown that unpublished codebases make replication difficult [26] and changes in the runtime environments can lead to different outcomes [27].

To this end, this study aims to mitigate these issues by providing the codebase [2], datasets [3], and thoroughly explaining the model's architecture and training procedure. Furthermore, the

[2] Author's GitHub: Miha5092

[3] The datasets can be found under the name "Bachelor Thesis Mihai Mitrea Dataset at " 4TU.ResearchData

codebase was designed with the inherent randomness of Reinforcement Learning algorithms in mind, allowing for fixed-seed experiments to be conducted.

The values for each hyperparameter of the world models and MCTS can be found in Appendix F as well as throughout the paper where appropriate. The seeds used throughout this study were 0, 42, and 420, in this particular order. The models were trained on nVidia A100, V100, and RTX 3060 GPUs.

7.2 Ethical Considerations

By looking at the setting considered in this study, MinAtari Breakout, no negative ethical implications arise. However, by broadly considering advancements in Model-based Reinforcement Learning, the idea that such algorithms might be used to cause harm requires additional attention. Even though this paper and related ones use simple and harmless environments, there are no signs that the proposed methods could not be used in a real-world setting. As [14] has shown, state-of-the-art MBRL algorithms can generalise to diverse environments without requiring fine-tuning, lowering the requirements of a would-be malicious party. On the other hand, [28] discusses instances in which Reinforcement Learning algorithms have already been shown capable of learning market manipulation tactics. With the higher sample efficiency of MBRL more and better such agents could be employed in the future and multiple areas of everyday life.

8 Conclusions

This study investigated how the discretization of the latent space through categorical distributions affects the performance of a model-based reinforcement learning algorithm when used for planning in a deterministic environment. More specifically, experiments were conducted to analyze modifications to the loss landscape, the effects of the latent embedding's size and shape, and this approach's overall performance and generalization ability.

The architecture used for this investigation was heavily influenced by that of the Dreamer models but modified to account for the fully observable environment presented by the MinAtar game of Breakout. To further simplify the environment, actions introduced by MinAtar's interface for more complex games were removed. The model was trained offline using a dataset obtained by a DQN agent and performance was assessed as the mean over multiple game episodes controlled by an MCTS planning inside the world model.

To investigate how the discretization and loss function parameters affect the loss function's landscape, multiple models were trained on a range of values around previously proposed norms. The results show that the training process becomes more unstable when using discrete embeddings, leading to degenerate solutions or periods of stagnating losses. When considering how the composition of the latent space affects the model's performance, results show that the number of categorical distributions is the main driver behind performance rather than their size. Overall, the discrete approach under-performed compared to the deterministic baseline due to its emphasis on over-optimizing less important components of the loss function.

Limitations

The main limitation of this study is that all explorations have been performed only on a single game of the MinAtar environment. Despite the usage of multiple seeds to eliminate outliers,

the obtained results might be heavily biased towards the characteristics of Breakout. Thus, this analysis represents only a starting point which could be used for a more comprehensive study covering multiple or all of the Atari games.

Furthermore, computational resources were a severely limiting factor when conducting experiments and because of that the full training dataset was rarely used. Unfortunately, a close inspection of the fine-tuning process reveals that the best hyperparameters for a low-data regime might not lead to the best-performing model in a high-data one, or vice-versa. Therefore, the optimal models used throughout this study might not show the full potential of each type of world model.

Future Work

A more diverse set of games Considering the limitations of this study, future works should aim at investigating the effects of discrete latent space embeddings in a broader setting than MinAtar's Breakout.

Stochastic Environment Given the probabilistic nature of the discrete world model, it is recommended that future studies are performed in a stochastic environment. Furthermore, such studies should also consider whether a continuous latent space embedding is suitable in such an environment, or a more complex model, such as a Gaussian one, should be used.

Curriculum Learning for Stability Considering the influence of the KL clip on the stability of the training process, curriculum learning strategies [29] could be employed to negate instabilities while allowing for more refined dynamics predictions. Such strategies could start with high KL clip values which would be gradually reduced as the model learns meaningful representations.

References

[1] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction (2nd ed.)*. The MIT Press, 2018.

[2] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2, 1991.

[3] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[4] Kenny Young, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber. The benefits of model-based generalization in reinforcement learning. *arXiv preprint arXiv:2211.02222*, 2022.

[5] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari, 2024.

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

[7] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.

[8] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588, 2020.

[9] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *Advances in neural information processing systems*, 30, 2017.

[10] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. https://worldmodels.github.io.

[11] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

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

[13] Jan Robine, Tobias Uelwer, and Stefan Harmeling. Discrete latent space world models for reinforcement learning. *arXiv*, 2020.

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

[15] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

[16] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[18] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[19] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[20] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[21] 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.

[22] William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[26] Matthew Hutson. Artificial intelligence faces reproducibility crisis. *Science*, 359:725–726, 2 2018.

[27] Harald Semmelrock, Simone Kopeinik, Dieter Theiler, Tony Ross-Hellauer, and Dominik Kowald. Reproducibility in machine learning-driven research. *arXiv preprint arXiv:2307.10320*, 2023.

[28] European Parliament, Directorate-General for Parliamentary Research Services, J. Fox-Skelly, E. Bird, N. Jenner, A. Winfield, E. Weitkamp, and R. Larbey. *The Ethics of Artificial Intelligence – Issues and Initiatives*. European Parliament, 2020.

[29] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. *International Journal of Computer Vision*, 130(6):1526–1565, 2022.

A Environment Modifications

An undesirable property of the MinAtar environment is that all of its games present the agent with 6 possible actions (left, right, up, down, fire, and do nothing) even though, games such as Breakout require only a subset (left, right, and do nothing). To this end, the environment was modified to allow for the removal of such extra actions. This change should not affect performance as in the original setting the agent would only need to learn to ignore these extra actions at the cost of more training time.

Furthermore, the only terminating condition for Breakout episodes in the classic MinAtar environment is if the agent loses by allowing the ball to slip past the paddle at the bottom of the screen. If all of the blocks are destroyed, new ones are spawned as if the episodes restarted. With no other termination condition, episodes have a theoretically infinite length without introducing any potentially novel scenarios for the agent. Because of this, the environment was further modified to terminate the episode once all the blocks have been destroyed.

B Training Data

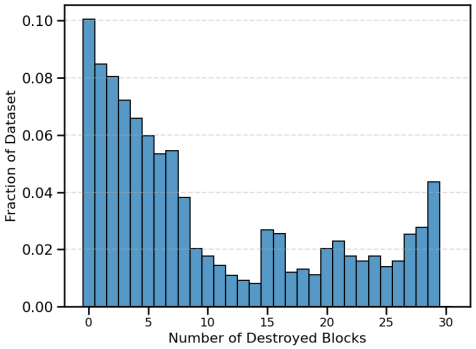
This appendix discusses the properties of the data packets used to train and evaluate the world models. The "learning" datasets represent transitions the DQN Agent encountered while learning to play optimally with intermediate saves after 10.000 and 30.000 episodes. The agent is trained using an ϵ-greedy strategy with linear annealing from 1.0 to 0.1. The "optimal" datasets contain the transitions the same DQN agent encountered after already having 5M interactions with the environment. Due to limitations regarding available computational resources, only the first 10.000 episodes of the optimal strategy were considered. Table 1 presents the distribution of actions and terminals for all of these datasets.

Table 1: Distribution of actions and terminals within all the considered datasets.

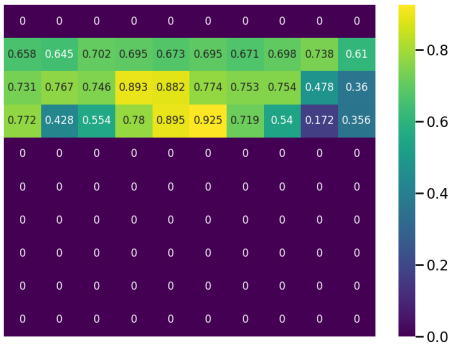
	Actions Distribution			**Terminals Distribution**	
Dataset	Nothing	Left	Right	Done	Not Done
Learning Full	31.61%	34.31%	34.08%	0.96%	99.04%
Learning 30k	31.35%	34.32%	34.33%	1.08%	09.02%
Learning 10k	30.70%	34.04%	35.26%	2.43%	97.57%
Optimal 10k	32.41%	33.34%	34.25%	0.23%	99.77%

B.1 Full Dataset Analysis

The full dataset was only used to train and evaluate both world models after fine-tuning them on the smaller 30k dataset. Figure 8 presents the distribution of the number of surviving blocks in each frame.



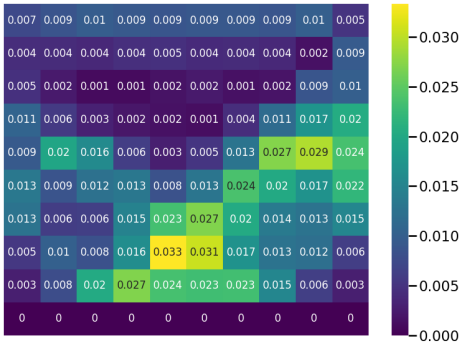
(a) Distribution of frames over the number of existing blocks.



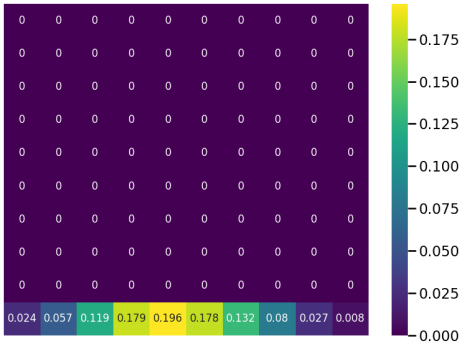
(b) Fraction of frames in which each block appears in.

Figure 8: Analysis of the number of existing blocks and their location in each of the frames of the full learning dataset.

Because of the predefined starting location and direction of the ball, the optimal strategy learned by the DQN agent skews the training dataset towards certain areas of the map. Figure 9 presents the most frequented location for the ball and paddle, respectively.



(a) Proportion of frames containing the ball in each of the game's tiles.



(b) Proportion of frames containing the paddle in each of the game's tiles.

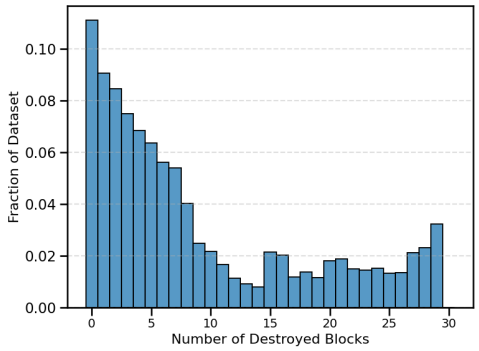
Figure 9: Heatmaps showcasing the most frequented locations for the ball and paddle, respectively.

It can be observed that the training dataset contains many more frames in which the paddle is located in a central position with a slight preference for the left side of the screen. Furthermore, the ball tends to follow a trajectory going towards the right side of the screen with very rare incursions towards the centre.

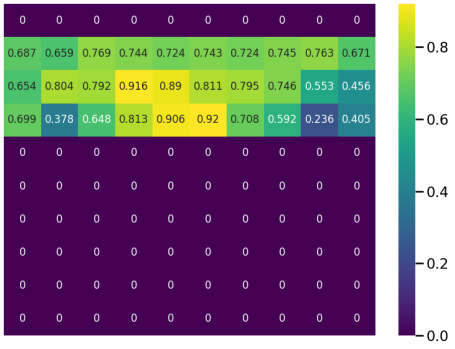
B.2 30k Dataset Analysis

The 30k dataset is the most used one throughout this study because of its similarity when compared to the full training dataset while maintaining a reduced size. This dataset contains

approximately 2.2 million of the total 5 million frames. Figure 10 presents the distribution of the blocks in the dataset's frames.



(a) Distribution of frames over the number of existing blocks.

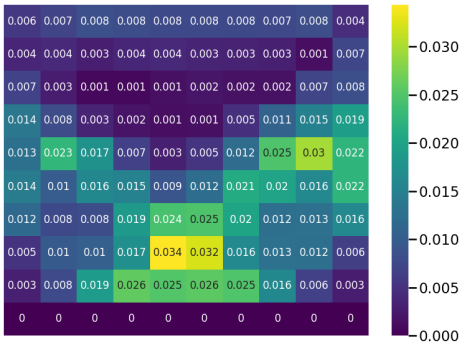


(b) Fraction of frames in which each block appears in.

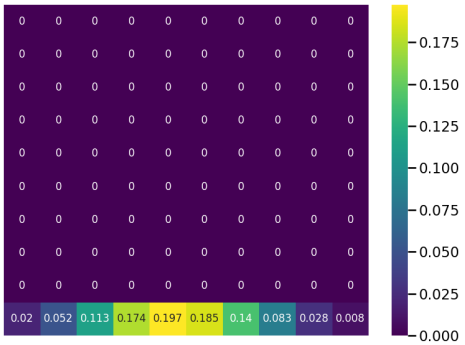
Figure 10: Analysis of the number of existing blocks and their location in each of the frames of the 30k episode learning dataset.

As can be observed, the distribution of the number of surviving blocs in Figure 10a is very similar to the one in Figure 8a, motivating the usage of this dataset.

When considering the positioning of the ball and paddle, the same trends as in the full dataset are also observed here as shown in Figure 11.



(a) Proportion of frames containing the ball in each of the game's tiles.



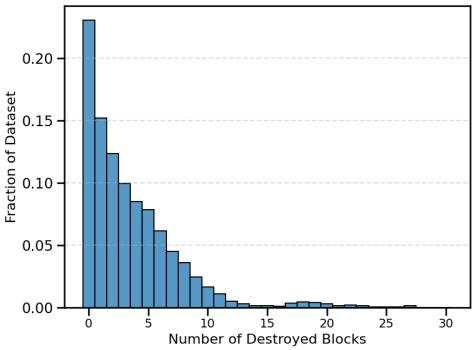
(b) Proportion of frames containing the paddle in each of the game's tiles.

Figure 11: Heatmaps showcasing the most frequented locations for the ball and paddle, respectively.

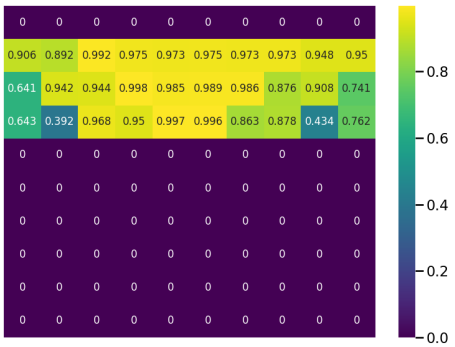
B.3 10k Dataset Analysis

This dataset was used during the loss function hyperparameter tuning process and to evaluate the models' ability to generalize. This dataset is much smaller than the 30k one, containing only around 440,000 frames. Furthermore, these transitions are from the early stages of the

DQN's learning process, meaning that, most probably, they do not represent an optimal game strategy. The distribution of the blocks in the dataset's frames can be seen in Figure 12.



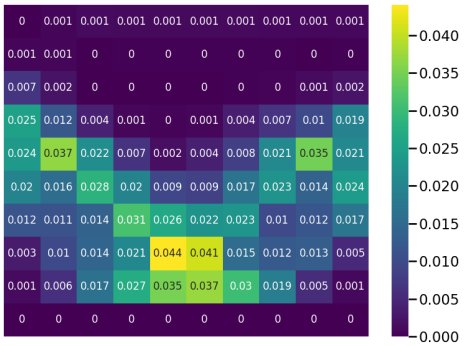
(a) Distribution of frames over the number of remaining blocks.



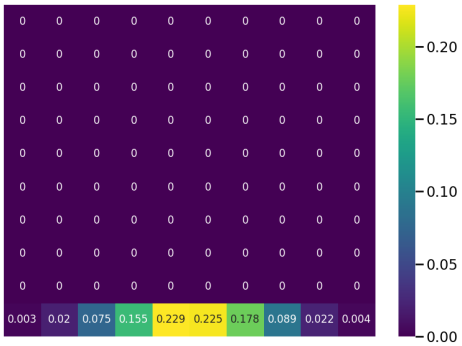
(b) Fraction of frames containing a block in each of the game's tiles.

Figure 12: Analysis of the number of existing blocks and their location in the 10k dataset's frames.

Unlike the other two datasets, the agent's playstyle is less pronounced, offering a more even placement of blocks on the game's tiles. Furthermore, this can also be seen through the symmetry in Figure 13.



(a) Proportion of frames containing the ball in each of the game's tiles.



(b) Proportion of frames containing the paddle in each of the game's tiles.

Figure 13: Heatmaps showing the most frequented locations by the ball and paddle for the 10k dataset.

C Hyperparameter Tuning

Model Type	Dyn Weights	Rep Weight	KL Clip
Discrete	0.7, 0.8, 0.9, 1.0	0.05, 0.1, 0.15, 0.2	0.75, 1.0, 1.25, 1.5
Continuous	0.1, 0.3, 0.5, 0.7, 1.0	0.05, 0.1, 0.2, 0.3	N/A

Table 2: Table presenting all the values considered for the loss function during the hyperparameter tuning process for both world models.

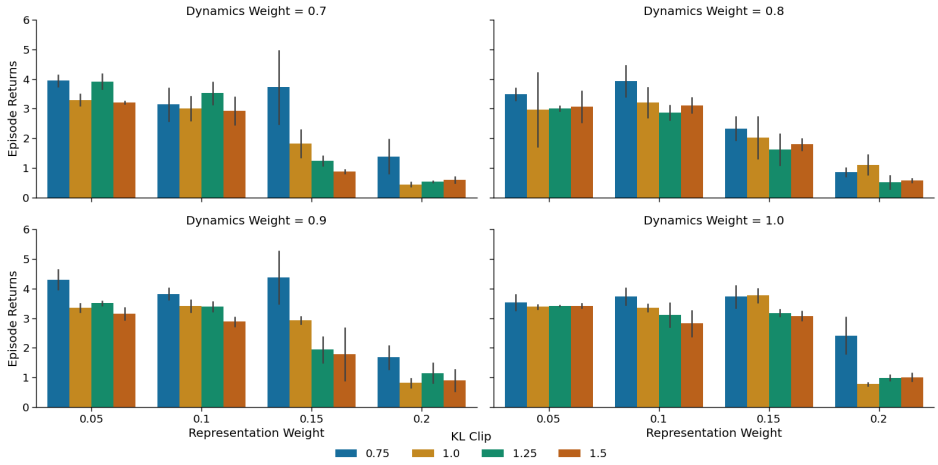


Figure 14: Hyperparameter tuning results for the discrete world model when trained on the 30k dataset.

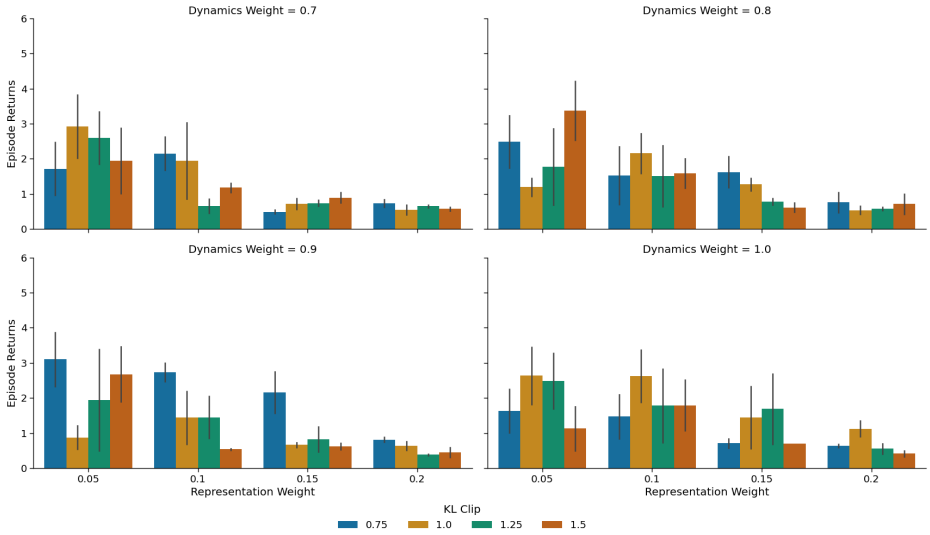


Figure 15: Hyperparameter tuning results for the discrete world model when trained on the 10k dataset.

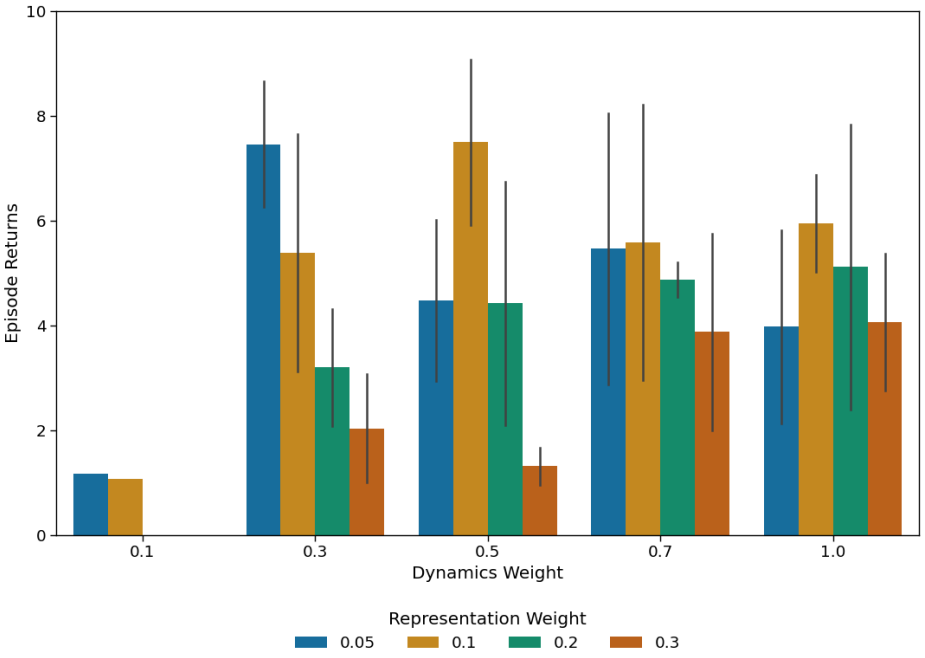


Figure 16: Hyperparameter tuning results for the continuous world model when trained on the 30k dataset.

D Loss landscape

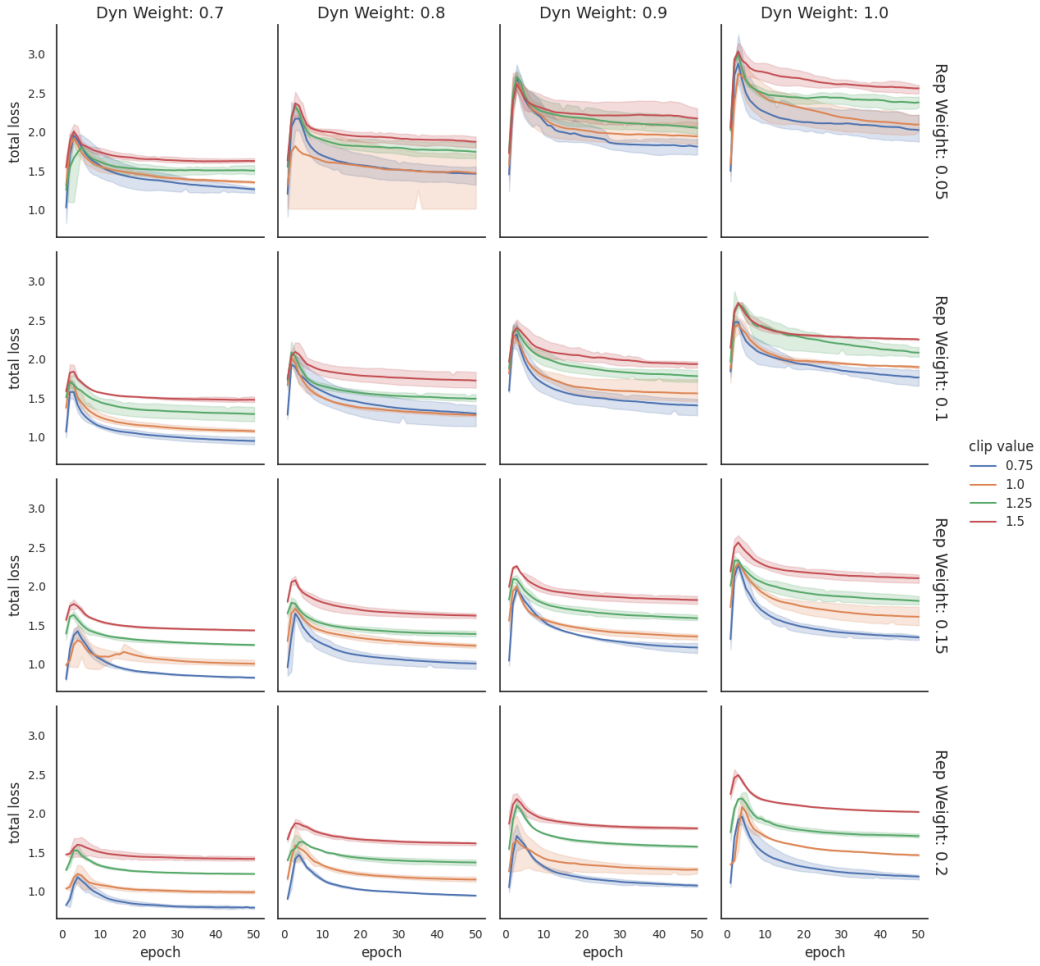


Figure 17: Total loss for the discrete world model when trained on the 30k dataset using a multitude of weights and clip values.

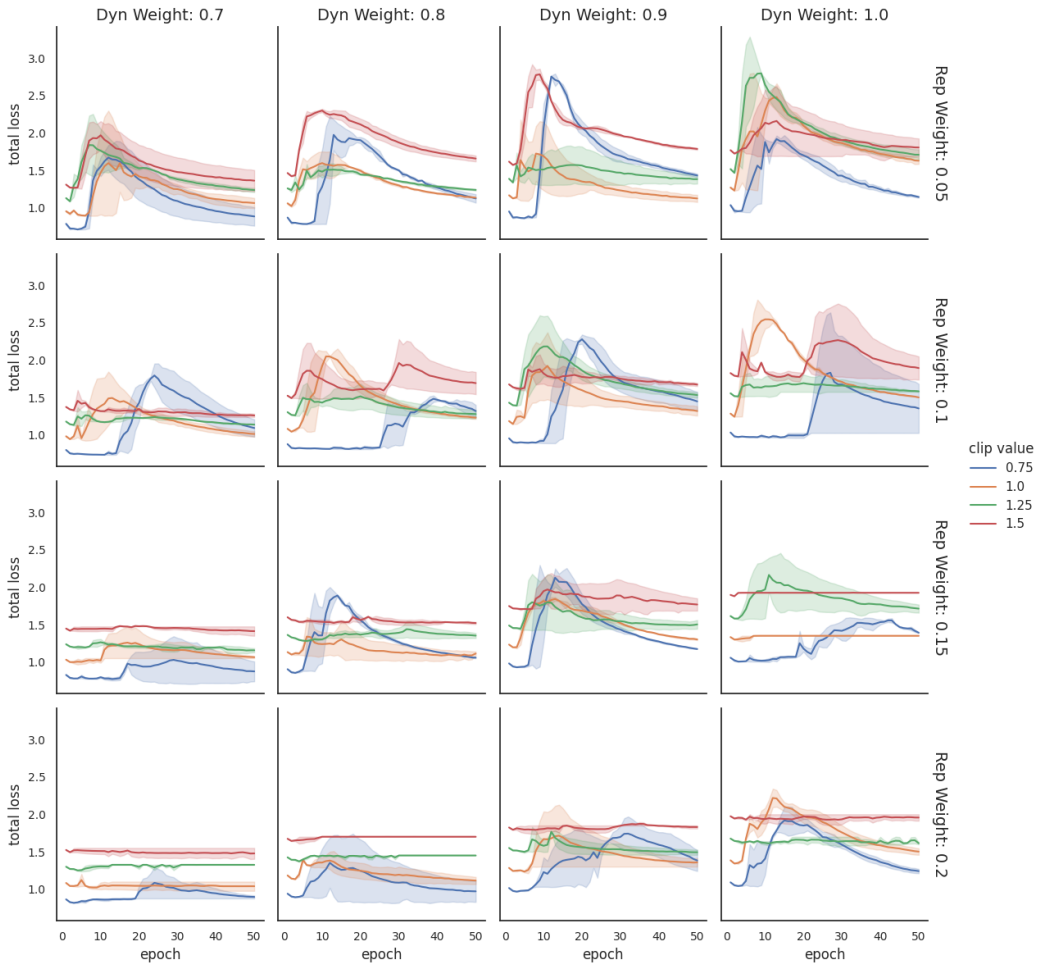


Figure 18: Total loss for the discrete world model when trained on the 10k dataset using a multitude of weights and clip values.

E Usage of LLMs

This study used ChatGPT4o and ChatGPT4 for a multitude of application regarding improvements for evaluation speed and plotting.

For example, to increase the evaluation speed of a model, the evaluation function was modified to use mpi4py [4] with the help of ChatGPT4o. This improvement was done by providing the original implementation file along with the prompt "Could you make this code use mpi4py?" The output prompt was the fully modified code which had to be slightly adapted due to mistakes regarding variable names.

Another example would be the improvement of the used plotting techniques. ChatGPT4o was prompted with "I have a dataset which represents the training loss of a lot of models. Therefore I want to somehow display the loss over the epochs. However I also have three varying hyperparameters whose impact I need to display. How would you recommend doing that?" to which it responded with a recommendation to use Seaborn's FaceGrid along with an example code. This response was integrated into the codebase and used to create the figures shown in Appendix D.

Sometimes, the model was used to help with LaTeX formatting, mostly regarding mathematical equations and tables. One example of such an instance is the following prompt which provided the current formatting of all mathematical equations for the loss functions presented in section 3:

[4] https://mpi4py.readthedocs.io/en/latest/intro.html

Please fix any errors you encoutner here:

```
\begin{math}
   \mathcal{L}(\phi) = \mathcal{L}_{pred}(\phi) +
\beta_{dyn}\mathcal{L}_{dyn}(\phi) +\beta_{rep}\mathcal{L}_{rep}(\phi)
\end{math}

\begin{math}
   \mathcal{L}_{pred}(\phi) = MSE(\hat{x_i}, x_i) + MSE(\hat{r_i}, r_i) +
BCE(\hat{t_i}, t_i)
\end{math}

\begin{math}
   \mathcal{L}_{dyn}(\phi) = \left \{
      \begin{array}{ll}
         MSE(sg(\hat{h_i}), h_i) & deterministic \\
         max(1, KL(sg(\hat{h_i}), h_i)) & discrete \\
      \end{array} \right
\end{math}

\begin{math}
   \mathcal{L}_{rep}(\phi) = \left \{
      \begin{array}{ll}
         MSE(\hat{h_i}, sg(h_i)) & deterministic \\
         max(1, KL(\hat{h_i}, sg(h_i))) & discrete \\
      \end{array} \right
\end{math}
```

Figure 19: Prompt to ChatGPT40 about formatting of LaTeX mathematical equations.

F Hyperparameters

Table 3: Training hyperparameters

Hyperparameter Name	**Value**
Batch Size	128
Unroll steps	1
Learning Rate	0.00025
Weight Decay	0
Aggregate Method	Sum

Table 4: Neural Networks architecture

Hyperparameter Name	**Value**
Activation Function	SiLU
Hidden Layer Size	128
Conv. Layer No.	1
Conv. Kernel Size	3
Conv. Layer Stride	1
Conv. Layer Channels	16
Representation Net. Linear Layer No.	3
Reward Net. Linear Layer No.	3
Done Net. Linear Layer No.	3
Embedding Net. Linear Layer No.	4
Dynamics Net. Linear Layer No.	4
Deconv. Layer No.	3
Deconv. Kernel Size	4
Last Deconv. Kernel Size	6
Deconv. Stride	2
Deconv. Padding	1
Deconv. Channel No.	64

Table 5: MCTS Hyperparameters

Hyperparameter Name	**Value**
Exploration Constant	1.0
Discount Factor	0.97
Planning Horizon	32
No. Simulations	128