World Models On A Visualized Bin Packing Problem

1st Mohammed Farhaan Shaikh
DKE (of Aff.)
Otto von Guericke University (of Aff.)
Magdeburg, Germany
mohammed.shaikh@st.ovgu.de

2nd Shivani Anil Hegde
DKE (of Aff.)
Otto von Guericke University (of Aff.)
Magdeburg, Germany
shivani.hegde@st.ovgu.de

3rd Diana Carolina Guzman Rodriguez
DKE (of Aff.)
Otto von Guericke University (of Aff.)
Magdeburg, Germany
diana.guzman@st.ovgu.de

4th Vishnu Jayanand
Digital Engineering (of Aff.)
Otto von Guericke University (of Aff.)
Magdeburg, Germany
vishnu.jayanand@st.ovgu.de

5th Asad Karim
DKE (of Aff.)
Otto von Guericke University (of Aff.)
Magdeburg, Germany
asad.karim@st.ovgu.de

6th Shweta Pandey
DKE (of Aff.)
Otto von Guericke University (of Aff.)
Magdeburg, Germany
shweta.pandey@st.ovgu.de

***Abstract*—Deep Reinforcement Learning (DRL) can play a promising role in developing automated tools for computer systems to interact efficiently with the real world. However, it is challenging to actually train DRL models on real-world systems as they come across the problem of sample inefficiency, and require a lot of time to train. In our paper, we discuss on how to overcome this problem by reducing the interaction with the real world, while the optimal behaviour is being learnt. In specific we study a DRL model called World Models, which enables an agent to learn from a dream state, without real-world interaction. In our paper we first evaluate changing the hyper parameters for the World Models, when solving a standard car racing problem, and measure the performance of the solution, to find the impact of the configuration. We then use the bin-packing optimization problem to test the World Models in a new computer system environment, not reported in the literature. To this end we develop a visual representation of the problem, and compare the performance of agents with the use of different methods like sum square errors and best fit. Taken together, our study sets a starting point for researching in depth on the application of World Models for computer system scenarios.**

***Index Terms*—Deep Reinforcement Learning, World Models, Bin Packing, DQN, Rainbow Agent**

I. Introduction

Deep Reinforcement Learning (DRL) is the combination of Reinforcement Learning and Deep learning or Artificial Neural Networks (ANNs). ANNs take a state as input and can learn a Q-value approximation for each action. Deep Reinforcement Learning provides solutions to a broad range of complex decision-making tasks that were previously impossible for a machine to solve with human-like acumen and perception. In DRL, we find models like Deep Q networks, which are generally applicable to many domains. They are based on Q-learning algorithms and replace traditional Q-tables from Q-learning.

The landscape of algorithms in modern Reinforcement Learning can be categorized mainly into two categories: Model-Free and Model-Based. One of the key questions in Reinforcement Learning is whether the agent has access to a model about the transitions between states in the environment. A model helps the agent to think forward and see a range of possible solutions to the problem and decide which option is the best. Algorithms which rely on models are called model-based methods and those which do not are known as model-free. In this paper, main focus is on model-based approaches specifically '*World Models*', and try to find out whether world models are a good alternative for improving the training runtime on some representative environments. The impact of the hyperparameters on learning with World Models for given use cases is also considered. Then it is tested on a new use case which is the Bin Packing optimization problem. Lastly, we compare to what extent the World Models are faster with regard to other approaches. Based on the previously discussed concepts, the car racing environment is studied in order train the car to remain on the road for as long as possible. We see how this approach is able to converge to a high reward with small changes in the components of the controller during training and then show that they can outperform other models including the state of the art at the time. One of the many benefits of the world models is the assurance to substantially improve sample efficiency which is a problem of DRL [1]. One such example is Simulated Policy Learning (SimPLE) [2]. SimPLE is a model-based RL algorithm that consists of alternating between learning a model and then using this model to optimize a policy with model-free reinforcement learning. The learning process is iterated around 15 times, where in each step the agent is trained inside the latest world model using PPO for creating a policy, which is used afterwards to collect data in the real environment and the world model is trained again with this new collected data. SimPLe is significantly more sample efficient than a highly tuned version of the state of the art Rainbow algorithm [3].

To test the world models on a different environment, we chose the Bin Packing problem using the online stochastic variant. In Bin Packing, items arrive one at a time with item sizes drawn from a fixed distribution, the task is to pack (or assign) items into capacity-bounded bins, in a way that reduces waste. Previous research has shown the DRL models to work in a competitive manner for this problem [4].For the use of World Models to such a representative chosen environment, it was necessary to adapt the bin packing problem to include a visualization representation.

The main contributions from this project are to measure the impact of changing hyper-parameters from the world models in a known case that is car racing, designing a visual representation for the Bin Packing problem and also test how the world models behave on the novel Bin Packing problem. These serve as starting points towards continuing researching the applicability of World Models for computer system problems.

This paper is structured as follows: In the Section II, a review of Deep Reinforcement learning and World Models is given. In the Section III, our design for the world models on unseen environments is explained. The Bin Packing problem that we worked on is explained on the Section IV, with the parameters and required changes. The experiments and the results obtained are shown in Section V and finally in Sections VI and VII we cover the conclusions and further work.

II. Deep Reinforcement Learning (DRL)

Reinforcement Learning (RL) can be expressed as the task of learning how an agent can perform a series of actions A on an environment that it perceives, in order to maximize the long-term cumulative reward obtained [5]. This scenario assumes episodic interactions of an agent with an environment. At each time step, the agent perceives the state s_t of the environment and performs an action a_t over the environment, which causes the environment to change to a new state s_{t+1} and to give the agent a reward r_{t+1}. The ideal strategy of agents is to obtain higher rewards through time. More precisely, agents are tasked with finding a policy, which is usually how actions are chosen at each step.

Deep Reinforcement Learning (DRL) combines Reinforcement Learning and Deep Learning in order to try to solve decision-making tasks in high dimensional spaces, which were not possible to solve earlier with other methods [6].

The deep learning approach on DRL consists on the use of deep neural networks, which are formed by a succession of multiple processing layers. The first layer contains the input information and each following layer consists of non-linear transformation of the data. All the layers from the deep neural network are trained in order to minimize the empirical error. Some of the commonly used layers in DRL are Convolutional layers for image and Recurrent layers for sequential data, with traditional fully connected layers too.

Over high dimensional state-spaces, traditional RL has difficulties mainly due to storage consumption and limited prior knowledge. DRL, on the other hand has been successful for tasks that are complicated due to the ability to learn (from experience) different levels of abstractions to represent the environment states and to use less storage space [6]. The abstractions in representing the environment are important, since they enable agents to have some estimated knowledge on how to act for unknown states, based on the similarity of those to states already visited in the past.

Apart from adapting to high dimensional state-spaces, RL faces other challenges, among them the exploration versus exploitation and the credit assignment problem. The first one has to do with the need of designing an agent that balances well the competing goals of exploring an environment to learn more about the impact of actions, while at the same time gaining high rewards. The credit assignment problem refers to the challenge of designing agents able to properly identify which choice (from the past actions) determined more crucially the reward obtained.

To address the challenges of RL, there are the two family of models for RL/DRL, which are are model-based and model-free approaches. Algorithms that make use of a model, for how the environment transitions from one state to another, are called model-based methods and the ones that do not have such kind of model are called model-free.

A. MODEL-FREE APPROACHES

Model-free approaches are comparatively more popular as they tend to be easier to implement and tune. However, they go without the advantage of sample efficiency.

The agent can be trained and represented with model-free RL by Policy Optimisation, Q-Learning or hybrids. Under policy optimisation, methods represent a policy explicitly as π_θ . The parameters θ are optimized either directly by gradient ascent on the performance objective J(π_θ), or indirectly, by maximizing local approximations of J(π_θ). Here each update only uses data aggregated while acting according to the most recent version of the policy, this is called on-policy optimization. Policy optimization learns an approximator V$_\phi(s)$for the on-policy value function V$^\pi(s)$, that is used in figuring out how to update the policy. An example of policy optimization method is the Proximal Policy Optimization (PPO) [7].

Under Q-Learning, methods learn an approximator $Q_\theta(s, a)$ for the optimal action-value function, $Q^*(s, a)$. This optimization is performed off-policy, which means during training each update uses data collected at any point, regardless of how the agent was exploring the environment when the data was acquired. The action of the Q-learning agent is commonly given by $a(s) = argmax_a Q_\theta(s, a)$, which means that the agent is expected to act consistently with respect to its beliefs about the values of actions. Apart from following the best action, Q-learning agents can act with some randomness, for better exploring the environment. An example of a Q-learning method includes the so-called Deep Q Network (DQN) agent [8].

B. MODEL-BASED APPROACHES

In the previous section, we talked about RL methods that are based on model-free approaches. In most cases learning a model of an environment can be tricky and demanding. Every aspect cannot be easily simulated with reliability, and these simulations are not accurate enough in many real-world use cases such that a trained model can be deployed to use them in our target environment. In this section we will be discussing model-based approaches, which seek to overcome the difficulty of reliably modeling real-world transition data.

Model-based methods, have (or learn) a model of the system transitions or a representation of the model. In contrast to model-free reinforcement learning, model-based approaches don't have an easy-to-define group of methods.

Model-based methods can use pure planning techniques like Model-Predictive Control (MPC), for the selection of actions [9]. In MPC, at each time step t the agent observes an environment, and it computes a plan which is optimal for the model. The plan depicts all actions to take over a fixed window of time after the present. Then the agent executes the first action of the plan and discards the rest of it. Each time that the agent interacts with the environment, it computes a new plan. This is done to avoid using an action from a plan with a shorter-than-desired planning horizon.

C. WORLD MODELS

One model-based method that was introduced by Ha and Schmidhuber [10] is called World Models. It basically creates a dream environment in an unsupervised way to learn a compressed spatial and temporal representation of the environment, where the agent can learn a policy to solve a task in its own dream and transfer such a policy back to the real environment. In this section we are going to present a summary of this work.

Large RNNs can learn different representations of data which makes them profoundly expressive. However, many model-free RL strategies in the literature often use small neural networks with few parameters.The credit assignment problem is one of the bottlenecks of the RL algorithms, which make it difficult for traditional RL algorithms to learn huge weights of a large model. Thus, in practice, in order to iterate faster to a fairly good policy during training, smaller networks are used.

To train the large RNN-based agents efficiently we use the backpropagration algorithm. This helps us do the training by dividing the agent into a large world model and a small controller model. At the beginning, we train the neural network to learn a model of the agent's world in an unsupervised manner. Then we train the controller model to learn to perform a task using this world model which helps the algorithm used for training to focus on the credit assignment problem on a small search space by keeping the capacity and expressiveness through the large world model.The goal of training the agent through the view of world model helps us give a high compact policy to perform the task.

1) Agent Model: The agent model is inspired by the human cognitive system. It comprises of a visual sensory component or a VAE (V) model that compresses what it sees into a small representative code z. The agent gets a high dimensional input observation as a 2D image frame from the environment at each time step and maps it to such small codes.

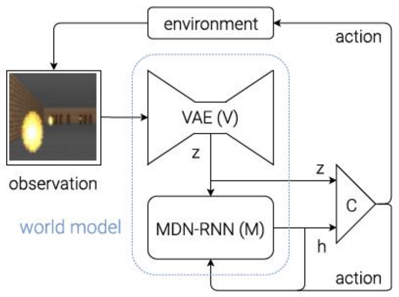

Fig. 1. Flow diagram of agent from World Model [10]

The agent model also has a memory component or a MDN-RNN (M), to make future code predictions based on the history of the agent. The *M* model acts as a predictive model of the future z vectors that V is expected to produce. As many environments have a stochastic nature, the RNN is trained to output a probability density function $p(z)$ rather than a deterministic prediction of z. The probability $p(z)$ is approximated as a mixture of Gaussian distributions, and the output of the RNN is the probability distribution of the next latent vector z_{t+1} given the current and historical information that is available. The RNN models $P(z_{t+1}|a_t, z_t, h_t)$, where a_t is the action taken at time t and h_t is the hidden state of the RNN at time t. The temperature parameter τ can be adjusted to control model uncertainty which is useful to train the controller later.

Lastly, the agent also consists of a decision-making component, or the Controller (C) Model, to decide the required actions to take based solely on the representations created by its vision and memory components. An overview of the agent is presented in figure 1. The C model is made small and simple and is trained separately from V and M so that most of the agent's complexity inhabits in the world model (V and M). The C model is a single layer linear model that maps z_t and h_t directly to action a_t at each time step as follows $a_t = W_c[z_t h_t] + b_c$.

On this paper we want to know what is the impact of changing the hyperparameters used on the agent on learning with World Models compare to the experiments conducted on the World Model paper?

2) Evaluations:

- Car Racing Experiment:
  The first experiment conducted was a car racing experiment, where an agent is trained to solve a car racing task. A predictive world model helps in extracting useful representations of space and time and these features are used as controller inputs and the controller is trained in a compact and minimal way to perform continuous control task, such as how to drive from pixel inputs for a top-down car racing environment called CarRacing-v0.
  In the car racing experiment, the tracks are generated randomly every time and the agent is rewarded as many tiles as possible in the minimum time. The agent controls

three actions such as steering left/right, acceleration, and brake.

The V model is trained by collecting a dataset of 10,000 random rollouts of the environment. The agent randomly explores the environment several times and records the random actions a_t taken and the resulting observations from the environment. The VAE encode each frame into low dimensional latent vector z by minimizing the difference between a given frame and the reconstructed version of the frame produced by the decoder from z. This trained V model can be used to pre-process each frame at time t into z_t to train the M model. The pre-processed data along with the recorded random actions a_t taken are used for training the MDN-RNN to model $P(z_{t+1}|a_t, z_t, h_t)$ as a mixture of Gaussians.

The world model (V and M) has no knowledge about the actual reward signals from the environment. The controller (C) model has the sole access to the reward information from the environment. The VAE can be used to reconstruct each time frame using z_t at each time step to visualize the quality of the information the agent actually sees during a rollout.

The results obtained on this experiment were:

- **V model:** Previous works have showcased that a good set of hand-crafted information such as LIDAR information, positions, angles and velocities can train a small feed forward network easily. Hence, the agent was first tested by handicapping C so that it only has access to V but not M. So the controller is defined as $a_t = W_c[z_t] + b_c$. While the agent can still navigate the race track, we can see that it wobbles around and misses the tracks on sharper corners.

  In line with the performance of other agents on OpenAI Gym's leaderboard and traditional Deep RL methods such as A3C, this handicapped agent achieved an average score of **632±251** over 100 random trials. An addition of a hidden layer to C's policy network helps improving the results to **788±141**, but it is not enough to solve this environment.
- **Full world model V and M:** The representation z_t given by V model only captures a representation at a moment in time and does not have much predictive power. In contrast to this, M can predict z_{t+1} really well. Combining z_t with h_t gives the controller C a good representation of both the current observation, and what to expect in the future. This is so because M's prediction of z_{t+1} is produced from the hidden state of the RNN h_t at time t, this vector is a good set of learned features that can be given to the agent. Allowing the agent to access the both z_t and h_t improves its driving capability. The driving gets more stable and the agent becomes able to attack the sharp corners and avoid wobbling. While making fast reflective driving decisions during a car race, the agent does not need to plan ahead and roll out or expect hypothetical scenarios of the future. The agent can just queries the RNN instinctively to guide its action decisions because h_t has information about probability distribution of the future. The agent can predict when and where to navigate in the heat of the moment. The agent is able to achieve a score of **906±21** over 100 random trials, efficiently solving the task and attaining new state of the art results. This world model takes in a stream of raw RGB pixel images and directly learns a spatial-temporal representation.

- Viz-Doom Experiment:
  As a second experiment, a mimic of a Viz-Doom environment was created in order to train an agent to learn inside its own dream instead of the actual environment and see how it behaves once the policy learned is transferred back to the actual environment.

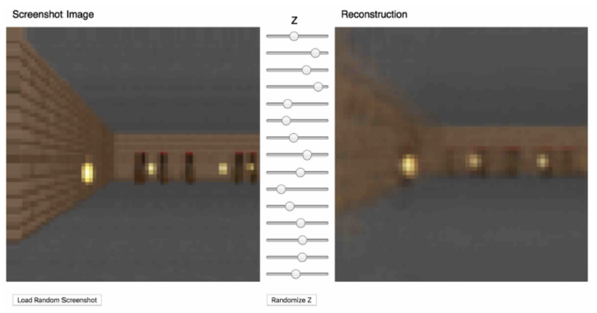

Fig. 2. Comparison image from the real model vs reconstructed image from World Model [10]

In the world model, a room with monsters shooting at the agent was created as shown in figure 2 and the cumulative reward was established as the number of time steps in which the agents stays alive. For the experiment, the setup was similar to the car racing task but differs in the sense that the M component predicts not only Z_t but also if the agent dies or not in the next frame.

For the experiment, the V component does not need to encode frames during the dreaming process as the agent train over the virtual environment, which is built over the M component and it is trained to mimic the whole game environment, including game logic, enemy behavior and physics.

This experiment consisted of 6 steps:

- Collect 100.000 rollouts from a random policy
- Train the V model to have the space representation
- Train the M model
- Define the controller
- Train the controller to maximize the survival time
- Transfer the policy learned on the actual environment.

In the virtual environment it was possible to make the game more challenging by changing parameters like

temperature, which showed that agents that performs well on higher temperature setting have a better performance in the normal setting. Once the agent trained on the virtual environment was tested on the original environment, it outperformed the scores that was obtaining on the virtual environment. This indicates that even though the virtual environment does not capture all of the details of the real environment, it is possible to learn a good policy on it.
On the experiment authors also found out that by increasing the temperature, it is possible to create a more stochastic version of model M and prevent the controller from taking advantages of imperfections of the world model. It means that with the normal setting on the world model, the World Model generates some trajectories that do not follow the laws on the actual environment. Besides that, the model M creates the virtual dream environment and the controller has accesses to the hidden states of M. Therefore, the agent can find some ways to "cheat" the environment to get higher rewards.
According to this, the temperature was set as a hyperparameter. If the temperature is too low, the monsters fail to shoot at the agent and even though the agent has the highest reward on the virtual model, it can not perform good on the actual environment. On the other hand, when the temperature is too high, it is too hard, and the agent won't be able to learn from it.

3) Iterative training procedure: The tasks made on the two experiments were simple and in case of trying to make more complicated tasks, it is necessary that the agent explore the world by collecting iteratively new observations (new rollouts) over time in order to refine the world model.

With the approach presented, it is necessary that the model *M* not only predicts the next observation but also whether the agent dies or not. The reward and action for the next time step is predicted so that the world model can imitate at some point its own model *M* to try to solve more complex tasks.

4) Discussion: With this work, it is demonstrated that it is possible to train an agent to perform a task inside of a simulated dream world, which allows to use less compute resources by using the simulated environment (which is less expensive) and it is possible to transfer the policies back to the real world.

Some limitations of the models are the use of VAE for the V model, which can possibly encode parts that are not relevant for the task and fail to reproduce task-relevant parts. The capacity of the world model is limited is not able to store all the information inside its weight connections, which can lead to issues like catastrophic forgetting.

III. DESIGN

In this section, we intent to answer how can world models be migrated to a new use case and to what extent are the world models better when compared to other approaches on a given use case.

A. Creating Dream Worlds for Unseen Environments

Dream Worlds for Unseen Environments can be created with the model M trained on the environment. The pseudo-code for the algorithm, which returns future vectors z, is as follows:

Algorithm 1 Sample Next z

```
1: procedure SAMPLENEXTZ($environment, action$)
2:     $outwidth \leftarrow RNN.outputSequenceWidth$
3:     $prevX \leftarrow np.zeros(outwidth)$
4:     $inputX \leftarrow concatenate(prevX, outwidth)$
5:     $feed \leftarrow \{inputX, environment.state\}$
6:     $outputRNN \leftarrow RNN.run(environment, feed)$
7:     $maxLogmix \leftarrow max(outputRNN.logmix/temperature)$
8:     $logmix2 \leftarrow exp(outputRNN.logmix/temperature - maxLogmix)/outputRNN.logmix.sum()$
9:     $mixture \leftarrow np.zeros(outwidth)$
10:    $chosenMean \leftarrow np.zeros(outwidth)$
11:    $chosenLogStd \leftarrow np.zeros(outwidth)$
12:    for j in range(outwidth) do
13:        $idx = getPiIdx(environment, logmix2[j])$
14:        $mixture[j] = idx$
15:        $chosenMean[j] = outputRNN.mean[j][idx]$
16:        $chosenLogStd[j] = outputRNN.logstd[j][idx]$
17:    end for
18:    $randGaussian \leftarrow environment.randn * sqrt(temperature)$
19:    $nextZ \leftarrow chosenMean + exp(chosenLogstd) * randGaussian$
20:    return $nextZ$
21: end procedure
```

The generic step function for training on real world systems follows the following pattern -

1. Act on Real System
2. Observe new state
3. Observe Reward
4. Determine whether training is done
5. Return
6. Add non-deterministic system interaction times.

The time consumption of the above process may be stochastic. The last step signifies that we can simulate non-deterministic interaction times.

For training on a dream system,

1. Sample next state from memory based on action.
2. Get no reward.
3. Process is never done until time is complete.
4. Return

B. Evaluating the Goodness of Dreaming

The goodness of Dreaming states how well the model performs with respect to training. This score is primarily accounted by the Reward *R* and the System Interaction Time *t*, wherein incremental in *R* should be proportional to the time taken for training. The implication of this logic tells you that increases in training time should result in an increased *R*. Assuming that the number of steps during dreaming and after dreaming are the same, we can formulate the Goodness of Dreaming, *G* as:

$G \doteq \frac{R_d}{R_t}$

where R_d is the average Reward after dreaming and R_t is the average Reward after training.

IV. EXPERIMENTAL SETUP

A. Benchmark Problems and Parameters

In order to test world models in a different problem to the experiments conducted by [10], we decided to choose the well-known Bin Packing problem using the online stochastic variant as in [4]. Bin packing is one practical application of online stochastic optimisation problems, where items arrive one at a time with item sizes drawn from a fixed distribution. The task is to optimize the placement of items into capacity-bounded bins, such as to reduce waste, improving as much as possible the utilization of existing bins. There are many areas where variants of Bin Packing problem are used such as the order assignment problem (where we assign orders to fulfilment resources), the tote packing problem (where we fill items as they arrive into totes for shipment), and the trailer truck packing problem. In computing, Bin Packing comes into play, where virtual machines with varying memory and CPU requirements are allocated to servers with fixed capacity.

In the online stochastic optimisation version of Bin Packing, items arriving online, one in each time period $t \in \{1,\dots,T\}$. Items can be of different types $j \in \{1,\dots,J\}$. The size of the type j is s_j and the probability that an item is of type j is p_j. Without losing its generality, we make an assumption that item types are indexed in the increasing order of their size: $s_1 < s_2 < ... < s_J$. Each bin with size B (we assume that $s_J < B < \infty$) allocates the items upon arrival. Allocation of items are considered feasible if the total size of the items packed in each bin does not exceed the bin size. Our aim is to find a feasible packing which minimises the number of bins used to pack all of the items arriving within the time horizon. We assume the item sizes s_j and bin size *B* are integers. We assume the number of bins one can open is boundless and we denote the sum of item sizes in a bin *k* as *level* h_k. After *t* items have been packed, we denote the number of bins at some level *h* as $N_h(t)$, where $h \in \{1, ..., B\}$.

We can minimise the total waste in partially filled bins (i.e empty space) by reducing the number of non-empty bins. In real applications (e.g. packing trucks or virtual machines) it costs while using these resources, so at any time horizon our objective is to minimise total waste $\sum_{t=0}^{T} W(t)$, where
$W(t) \doteq \sum_{h=1}^{B-1} N_h(t)(B-h)$.

We use $W_F^A(t)$ to denote the total waste after step *t* of algorithm *A* when the input samples come from distribution *F* . For RL, we define the cumulative reward up to time step *t* to be $W_F^R L$ (t). The Bin Packing problem is formulated as an Markov Decision Process, where the state $S_t \in \boldsymbol{S}$ is the current item with size s_j and the number of bins at each level is $N_h(t)$, where $h \in 1, ..., B$. The action a to be performed is to pick a bin level which can fit the item. Thus, the number of actions can be done is *B* with one action for each level and action 0 corresponds to opening a new bin. An episode characterises the start and end of a simulation. At the beginning, all the bins are empty. The reward R_t is the negative of incremental waste as each item is put into a bin s_j . If the item is put into an existing bin, the incremental waste will reduce by item size. If the item is put into a new bin, the waste increases by the empty space left in the new bin. *T* items need to be placed in the bins, after which the episode ends.

At each time t a state for the environment was defined as an array of size $B+1$, where each i from the first B elements represents the amount of bins that are full up to level i and the last element represents the size of the new item that arrived on time t.

In order to use the world models for the Bin Packing problem described in this section, it was necessary to create a visual representation of the state so the VAE Model from the agent can interpret it. For this purpose a 64x64x3 bit image was created with the state s at time t. figure 3 represents an example with a bin size of 30. The state is [0, 18, 70, 51, 82, 0, 65, 29, 0, 183, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4]. The last element in this state is the size of a new item and the rest represent how many bins are full up to that level. This array is encoded into an image in the following way - each number from the state is represented as a row of height 2px from bottom to top except the bottom most row, of height 4px, which represents the size of a new item. As the number of bins full up to each level increases, the color of the box shifts from blue to white and the row extends to the right. The last row encodes new item as a box of size 6x4px and the box is placed according to the weight. if the weight of the new item is 1 then the box is placed on the left most side and each item of higher weight is incrementally placed to the right.

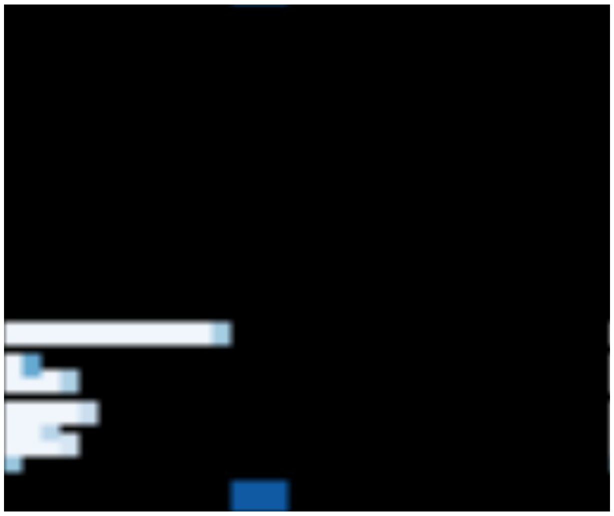
Fig. 3. Bin Packing visual representation

B. Model Parameters

The parameters chosen for the Bin Packing problem used in this project were:

- **Bin size:** 30
- **Number of different items:** 9. For each one of this items the size and probability was defined.

- **Item sizes:** 1, 2, 3, 4, 5, 6, 7, 8, 9
- **Item probabilities:** 0.06, 0.11, 0.11, 0.22, 0, 0.11, 0.06, 0, 0.33
- **time horizon:** 1000

C. *Hardware and Software Details*

We have used high-end machines to run the experiments to avoid any bottlenecks on the hardware side. Our processors included two 'Intel(R) Xeon(R) Gold 6130 CPU @ 2.10G' with 1MB L1 cache, 16MB L2 cache and 22MB L3 cache memory. The machine included six pairs of '64GiB DIMM DDR4 Synchronous 2666 MHz' RAM while having two GV100GL [Nvidia Tesla V100 SXM2 32GB] GPUs. Linux Ubuntu Machine was used with conda environment set up to run the experiments.

V. RESULTS

The experiments conducted on this project were divided on changes of the hyperparameters from the car racing problem from the World Model paper and experiments over the use of the World Model for the Bin Packing problem.

A. *Hyperparameter changes on Car racing problem*

For the car racing problem the hyperparameter that was changed in order to test the influence of the parameter on the performace was the population size for the controller component. In figure 4 is possible to observe the initial results using the same hyperparameters as in the paper with a population of 64, for images 5 and 6 the population size was changed to 32 and 16 respectively.

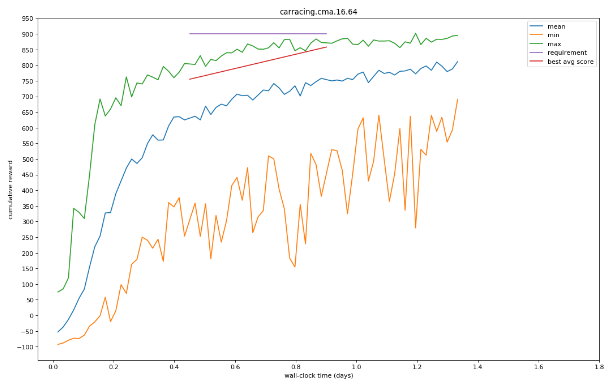

Fig. 4. Results obtain with initial hyperparameters

For each set up, the model trains for about 1 day. We observe that the difference between results from the models trained with a population size of 64 and 32 is not big. In both cases the agent achieves a cumulative reward of 850 in about 14 hours of execution. What we can observe is that the agent is exploring less for smaller population size, Therefore, the gap between the best and the worst line is small for both models with population sizes 32 and 16.

For the original World Model paper a population size of 64 was used and the requirement was that the agent achieve an average score above 900, which was achieved after running

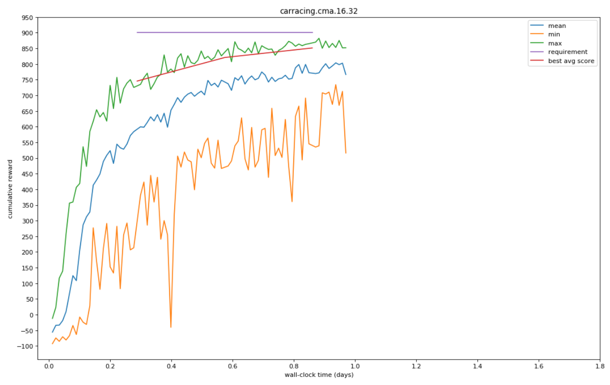

Fig. 5. Results obtain with population size of 32

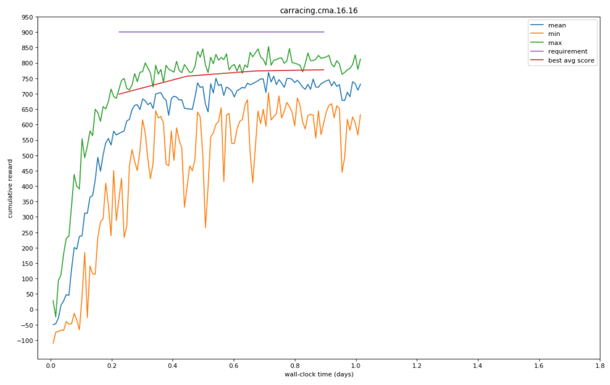

Fig. 6. Results obtain with population size of 16

the model for over a week and the reward after training for one day was approximately 850. Therefore an average score of 850 is considered as good and for all of the set ups that we run follows the same behavior.

B. *Bin Packing problem with World Models*

The results obtained by running the Bin Packing problem with World Models are shown in the image 7, where the mean cumulative reward after training for 5 hours reaches a value of -112. After that point the mean cumulative reward does not suffer greater variations.

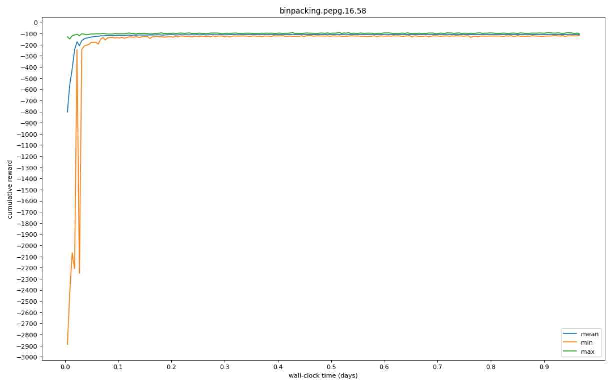

Fig. 7. Results obtain for Bin Packing with World Models

After running the trained model for the Bin Packing problem with 1000 trials, it gives an average of -104.48.

To get an idea of how good the results form the Bin packing problem with World Models are, we run the problem with the same parameters in two baseline best-fit and Sum of squares and a Rainbow agent.

As a result for the best-fit for the same 1000 trials, the average total reward was -26.047. For the Sum of squares, the average total reward obtained was -26.857. In comparison with the results obtained from the World models it is around four times worse than the two base lines.

For the Rainbow agent the visualized version of the Bin Packing problem was used. On the image 8 is possible to observe the reward obtain on the 1000 trials. The average reward was -96 which is around 8% better than the World Models.

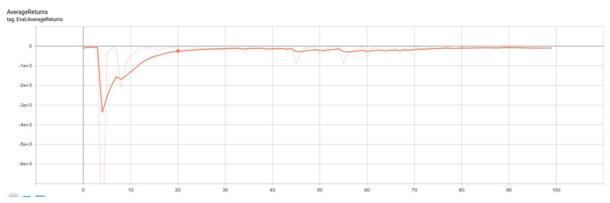

Fig. 8. Results obtained for Bin Packing with Rainbow agent

VI. Conclusion

Our study is based on the application of World Models that serve as DRL model in order to improve the efficiency of computer system applications. This approach exploits the ability of these models to train in a simulation environment so that the cost incurred during the training with real world systems is reduced to a large extent. The role of the hyperparameters in the controller model is evaluated and for the Bin Packing problem the World Model is tested on a computer system environment.

From our results, we find that the population size hyperparameter for the car racing problem is related to the hardware where the training is executed and the amount of population can not exceed the number of available cores. On the tests made, it was observed that as the population size increases, the maximum cumulative reward that the agent achieves in the same time window also rises. The difference in terms on cumulative reward for setups of 32 and 64 population size is relatively small, therefore it could be concluded that a population size of 32 can be used to get good results on the car racing problem without the need to use more resources.

A major limitation regarding the experiment was the long time of training approximately more than one day for running all the tests over the car racing and the Bin Packing problem and the requirement of high-end machines. Another drawback on the execution of the project was that World Models expect only images from the environment and the bin packing problem is not a visual problem. Consequently, it was necessary to create a visual representation so that the World Model could work with the Bin Packing problem.

We can conclude that the results obtained from the Bin Packing problem with World Models are not as good as the results obtained from the baseline methods such as Sum of Squares and Best fit which are without the visual representation. When compared to Rainbow agent that has the same visual representation as Bin Packing problem the difference between them is not significant. The reason being that the visual representation may not have been the best or our model should have trained for a longer time in order to get better results.In order to find out the reason behind this we can try running our visual Bin Packing approach with other RL agents to test how good the visualization performs or try to create a different visualization for the problem.

VII. Discussion and Future Work

As future work an interleaving between training the agent on the actual environment and the dream environment should be performed. According to the paper, the training can happen over the actual environment or over the dream environment, hence it would be interesting to test the impact of interleaving both training.

In order to make this interleaving between the two types of training for the car racing problem is necessary to change the Memory model so it will return not only the next z value but also the reward to mimic completely the real environment.

To be able to prove how good the world model is in a problem like Bin Packing, it is desired that a comparison of the results of the Bin Packing problem with the visual component is also tested in other DRL approaches like PPO or DQN.

Acknowledgments

We would like to thank our supervisor Gabriel Campero Durand for his thoughtful feedback and time spent with us in discussing the topics related to it. We are immensely grateful for his guidance and valuable insights on this project which has helped us improve our research and final results.

References

[1] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.

[2] 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, 2019.

[3] 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, 2017.

[4] Bharathan Balaji, Jordan Bell-Masterson, Enes Bilgin, Andreas Damianou, Pablo Moreno Garcia, Arpit Jain, Runfei Luo, Alvaro Maggiar, Balakrishnan Narayanaswamy, and Chun Ye. Orl: Reinforcement learning benchmarks for online stochastic optimization problems, 2019.

[5] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[6] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018.

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

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[9] Sanket Kamthe and Marc Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1701–1710, Playa Blanca, Lanzarote, Canary Islands, 09–11 Apr 2018. PMLR.

[10] David Ha and Jürgen Schmidhuber. World models, 2018.