A DEEP REINFORCEMENT LEARNING-BASED RATE ADAPTATION OF ADAPTIVE VIDEO STREAMING*

Lam Huynh Hoc†
Department of Computer Science and Engineering
International University - HCMC
Ho Chi Minh city.
hoclamhuynh@gmail.com

Nghia Nguyen Trung
Department of Computer Science and Engineering
International University - HCMC
Ho Chi Minh city.

Phuong Vo Thi Luu
Department of Computer Science and Engineering
International University - HCMC
Ho Chi Minh city

ABSTRACT

Artificial Intelligence has always been an ambitious topic of research within Computer Science. However, recent breakthroughs in Artificial Intelligence have put it into the spotlight, regardless of public opinions on the subject matter. Reinforcement Learning is the spearhead of this advancement. Reinforcement Learning has been able to accomplish incredible feats, reaching human-level performance in various tasks such as video-game playing, image generation, communication and so on. Aside from that, many researchers can confirm the effectiveness of Reinforcement Learning algorithms on smaller scale environments.

However, there exists two approaches to Reinforcement Learning: Model-Free and Model-Based. A large majority of algorithms receiving media coverage belong to the school of Model-free and they have demonstrated massive success over recent years through a large degree of fine-tuning and optimization. Comparatively to Model-free, Model-based enjoyed less limelight while still being capable. The reason for this phenomenon is that Model-based Reinforcement Learning is more complex in its implementation and more nuanced, and thus, being reserved for tasks Model-free cannot accomplish. One fine example of this is in playing Atari games, where Model-based algorithms produce better results in multiple games. It is from this that we hypothesize whether there could be environments in which Model-based Reinforcement Learning performs better than Model-free ones.

One environment we have in mind is the rate adaptation process of streaming services/platform. In this paper, we aim to integrate a Model-based Reinforcement Learning algorithm to that environment. We will compare the effectiveness of a Model-based algorithm against cutting edge, readily available Model-free methods, namely DQN, A2C and PPO from a reliable source. Model-free Reinforcement Learning has proven to work well for rate adaptation algorithm. Thus, this paper is a documentation detailing our attempts into modifying this algorithm using Model-based Reinforcement Learning, which recently demonstrated super-human performances in various Atari games tests, with the hope of further maximizing quality of service and user experience.

*The utilization of Model-based Reinforcement Learning into an Adaptive Bitrate Algorithm for adjusting bitrate

†Author of this paper

WOODSTOCK'18, June, 2018, El Paso, Texas USA

[1] [1]: Hongzi Mao, Ravi Netravali, Mohammad Alizadeh (2017) "Neural Adaptive Video Streaming with Pensieve".

CCS CONCEPTS

• Computing methodologies • Artificial Intelligence • Distributed artificial intelligence • Intelligent agent

KEYWORDS

Reinforcement Learning, Neural Network, Video Streaming, Rate Adaptation, Model-based

1 INTRODUCTION

In 2017, a study concerning the subject matter was published [1][1]. In this research paper, the group concluded that due to rigid control rules based on simplified model of the streaming environment, the Adaptive Bitrate Algorithm (ABR) is unable to achieve optimal performance across a wide array of network conditions across the world. Additionally, the group proposes Pensieve, a system that generates an ABR using Reinforcement Learning (RL), opening the possibility of utilizing RL to improve the algorithm's functionality. To simplify the ABR's decision making process, the source video is divided into multiple parts on the server-side, and it is the ABR's goal on the client-side video player to select the most suitable bitrate for each corresponding part based on observations from the environment such as network conditions and playback capability at the moment of viewing. The group asserted that the majority of concurrent ABR algorithms develop fixed control rules for making bitrate decisions, leading to a lack of generalization for a diverse environment. Thus, the Pensieve system leverages Reinforcement Learning to learn a policy without any explicit instructions on control rules or pre-conceived assumptions of the environment, solving both proposed

problems. It is how a Reinforcement Learning agent is trained, without any external input and purely on experience alone.

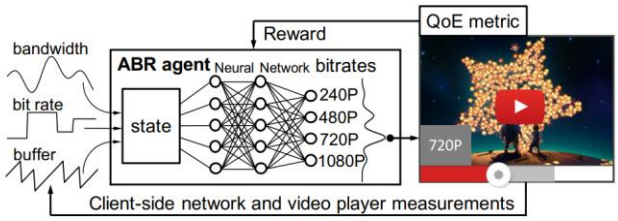

Figure 1: The blueprint application of Reinforcement Learning in a video streaming environment.

Since the implementation of Reinforcement Learning into a video streaming environment is possible, we wonder if there are algorithms available that can perform on the same environment. At the time of the publication of this paper, many popular RL algorithms, such as DQN and A2C, existed but still in their infancy. Since then, many optimizations to those algorithms have been made, creating various permutations that vastly outperform the original. Not only that, but there is also a different approach to Reinforcement Learning that recently saw numerous breakthroughs. That is to say that Reinforcement Learning has come a long way since the publication of this Video Streaming research paper.

And thus, the goal of this paper is to find out if there really is a better Reinforcement Learning algorithm capable of functioning in a video streaming environment. In particular, this paper will focus on a Model-Based Reinforcement Learning algorithm called DreamerV2 and its performance in a video streaming environment compared to various Model-Free methods. According to its original paper, DreamerV2 achieved a higher level of performance over many cutting-edge Model-Free algorithms in Atari games playing. It is evident that those environments are nowhere near similar, one is an Atari games playing environment and the other is a video streaming one, but there hasn't been any research exploring this avenue. On the other hand, various Model-Free algorithms have been confirmed to work well in a video streaming environment but very few documentations for Model-Based. These 2 contributing factors have led us to seek out answers regarding the topic. Will DreamerV2 function well in a video streaming environment and is it worth training a Model-Based algorithm over readily available Model-Free algorithms? What benefits will a Model-Based algorithm provide in this situation? We intend to provide answers to these questions with the hope of gaining insights into improving user experiences for many consumers.

2 DREAMERV2 – A MODEL-BASED REINFORCEMENT LEARNING ALGORITHM

2.1 Model-based Reinforcement Learning

Model-based Reinforcement Learning is another branch within the field. Whereas Model-Free algorithm aims to optimize its policy, its behaviour within the environment in order to maximize cumulative reward, Model-Based algorithm aims to "understand" the environment, thus training the agent based on this understanding. In a less vague manner, Model-Based algorithm will construct a model based on the agent's observation and will then try to predict the future states and their reward within this model of the environment, constructing a policy regarding that. This is the main reason why Model-Based is much harder to implement as a slight error in model learning will mislead the agent entirely. Many techniques can be implemented to mold the model and its prediction accuracy. As for the agent's learning process, Model-Based agent still uses algorithms from Model-Free. If successful, Model-Based algorithm can speed up the learning process by orders of magnitude, having higher sampling efficiency than Model-Free algorithms.

2.2 DreamerV2

In 2021, in a collaboration between Deep Mind and University of Toronto, spearheaded by Danijar Hafner and his team, a new Model-based algorithm was created called DreamerV2 [2] [2]. DreamerV2 builds upon the foundation of PlaNet (Deep Planing Network) and Dreamer to create a reinforcement learning agent that rivals, or even surpasses, average human capability in Atari games.

The basic idea of DreamerV2 consists of a model capable of "dreaming". For each state, the algorithm will have the model "imagine" possible future state and reward. Just as a human capable of thinking ahead, devising logic and patterns for the world based on experience, so too will DreamerV2 be able to analyze what will happen in the future based on previous observations and episodes. Essentially, the model acts as the algorithm's view of the world, a "dream" of the environment. The next step, then, is to minimize the difference between "dream" and reality in order to obtain a relatively accurate view of the environment.

DreamerV2 combines many components into its model creation to achieve near identical state-space representation of its environment over many training steps. Since the goal is to create as close a model as the environment itself, all aspects involved in the learning process, all the interactions between the environment and the agent must be replicated. Those components are:

$$\text{RSSM} \begin{cases} \text{Recurrent model:} & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Representation model:} & z_t \sim q_\phi(z_t \mid h_t, x_t) \\ \text{Transition predictor:} & \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t) \\ \text{Image predictor:} & \hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t) \\ \text{Reward predictor:} & \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t) \\ \text{Discount predictor:} & \hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t \mid h_t, z_t). \end{cases}$$

Figure 2: The world model of DreamerV2

Where the Representation model is tasked with generating a stochastic state with the knowledge of the input image, the

[2] [2]: Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, Jimmy Ba (2021) "Mastering Atari With Discrete World Models".

Transition predictor guesses the image without such knowledge, the Image predictor is tasked with encoding an input image for model state incorporation or decoding the model state into an output image, the Reward predictor estimates the reward of a given model state and the Discount predictor estimates the probability of an episode ending when learning behaviors from model predictions, since an Atari game can abruptly end at any given time.

The main model used in DreamerV2 is RSSM (Recurrent State Space model). State-space model (SSM) [3] [3], as a concept, proposes that there is a compact state representation that captures all essential aspects of the environment on an abstract level. It describes the probabilistic dependence between latent state variables and observation, providing a general framework for analyzing deterministic and stochastic components observed through a stochastic process. The recurrent state-space model (RSSM) used within this algorithm is first introduced in PlaNet, a precursor to Dreamer and DreamerV2, containing both deterministic and stochastic components for predicting forward via latent space representation. For each step, utilizing the concept of latent hidden state, the algorithm will pass high-dimensional information (images in this case) towards an encoder network, extracting hidden states. A decoder within will try to reconstruct the original based on these hidden states, try to match the original as close as possible by means of KL-balancing. Upon reaching near identical reconstruction, those hidden states now have the capability to predict accurate future states.

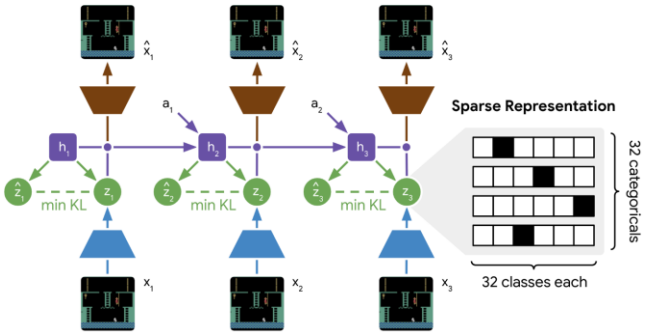

Figure 3: DreamerV2's model learning process

To further explain the training process transcribed into the picture above, each parameters present will be given more detailed information. The training sequence of images x_t is encoded using the Convolutional Neural Network. The RSSM uses a sequence of deterministic recurrent states h_t. Despite the potentially vast number of variables taken from the input image, the image is compressed into a number of categories, in this case 32, representing the number of variables, and a number of classes representing their associate values, in this case 32. In essence, 32 variables with 32 values each will be taken from input image as information. For each recurrent state of each step, using GRU cells, the RSSM computes the posterior stochastic state z_t that incorporates information regarding the input image x_t and the prior stochastic state $\hat{z}_t$ that will "imagine" the original image, essentially predicting what the posterior will look like without access to the input image. The KL loss from these posterior and prior states help regulates how much information should be incorporated into the posterior while also training the prior. As stated by the research paper, this regularization increases robustness to novel inputs. Upon performing KL loss balancing, the algorithm will decode information from deterministic state h_t and stochastic state z_t to reconstruct the original input image, for the Image Predictor training purpose.

The algorithm will proceed to optimize the world model and its components according to the following loss function:

$$\mathcal{L}(\phi) \doteq E_{q_\phi(z_{1:T} \mid a_{1:T}, x_{1:T})}\Big[\sum_{t=1}^{T} -\ln p_\phi(x_t|h_t,z_t) - \ln p_\phi(r_t|h_t,z_t) - \ln p_\phi(\gamma_t|h_t,z_t) + \beta\, KL\left[q_\phi(x_t|h_t,z_t) \parallel p_\phi(z_t|h_t)\right]\Big]$$

Where $-\ln p_\phi(x_t|h_t,z_t)$ is the image log loss, $-\ln p_\phi(r_t|h_t,z_t)$ is the reward log loss, $-\ln p_\phi(\gamma_t|h_t,z_t)$ is the discount log loss and $\beta\, KL\left[q_\phi(x_t|h_t,z_t) \parallel p_\phi(z_t|h_t)\right]$ is the KL loss.

The agent in DreamerV2 employs Actor-Critic learning algorithm for behavior learning. Actor-Critic provides the robustness of a policy-oriented algorithm, capable of learning stochastic and deterministic environment, and the sampling efficiency of a value-oriented algorithm. Actor aims to determine appropriate action in posterior state which will lead to states that maximizes reward while Critic aims to estimate sum of potential returns achieved from actions stated by Actor.

For Actor, DreamerV2 leveraged Reinforce algorithm to output action that maximizes predicted reward. However, Reinforce is known to produce fairly unbiased but high variance gradient, leading to long training period and sampling inefficiency. DreamerV2 combines the high variance Reinforce gradient with biased, low variance straight-through gradients for dynamic backpropagation. The use of entropy regularizer will encourage exploration whenever possible.

$$\mathcal{L}(\psi) \doteq E_{p_\phi, p_\psi}\Big[\sum_{t=1}^{H-1}\Big(-\rho \ln\, p_\psi(\hat{a}_t|\hat{z}_t)\, sg\left(V_t^\lambda - v_\xi(\hat{z}_t)\right) - (1-\rho)V_t^\lambda - \eta H[a_t \mid \hat{z}_t]\Big)\Big]$$

$$V_t^\lambda \doteq \hat{r}_t + \hat{\gamma}_t \begin{cases}(1-\lambda)v_\xi(\widehat{z_{t+1}}) + \lambda V_{t+1}^\lambda, & if\ t < H\\ v_\xi(\widehat{z_H}), & if\ t = H\end{cases}$$

[3] [3]: Lars Buesing, Théophan Weber, Sébastien Racanière, et al (2018) "Learning and Querying Fast Generative Models for Reinforcement Learning".

For Critic, DreamerV2 utilized temporal-difference learning to estimate discounted sum of future rewards. Since the RSSM can generate imaginary future state once it has learnt the environment and its policy, the world model can project multiple states ahead, enabling the use of n-step targets for faster Critic learning. The number of possible future states lookahead is determined by the horizon H set beforehand.

Prediction based on prior and posterior stochastic states requires a process called KL-balancing for optimization. The model's loss function can be interpreted as an ELBO, where the imaginary predicted state is considered an approximate temporal posterior and the predicting process (Neural Network, GRU cells, … or also known collectively as transition predictor) can be considered a temporal prior. By minimizing KL loss, we can train the prior towards the posterior and regularize posterior towards prior. In other words, the predicted state can more closely resemble what can be observed from the environment. However, since predicted state is dependent on the accuracy of the predicting process, we need to avoid regularizing predicted state towards a faulty prior, potentially causing a lengthy training process. Thus, KL loss is minimized faster with respect to prior. Hence why it is called KL balancing.

$$\mathcal{L}(\xi) \doteq E_{p_\phi,\, p_\psi}\left[\sum_{t=1}^{H-1} \frac{1}{2}\left(v_\xi(\widehat{z}_t) - sg\left(V_t^\lambda\right)\right)^2\right]$$

3 ADAPTIVE BITRATE STREAMING ENVIRONMENT AND DATASET

Adaptive Bitrate (ABR) Streaming is a technique utilized in streaming multimedia content over HTTP. An ABR algorithm will feature an encoder for encoding the source content to fit a variety of network conditions. Multiple versions of the same source content compatible with different bitrates are created. After the encoding process is done, the content is segmented into smaller files a few seconds in length, usually called chunks. In a real streaming environment, the video is sequentially transmitted and shown in these chunks, rather than the whole video in a set bitrate. When the next chunk is in line for transmission, the ABR algorithm will take into account the network bandwidth and CPU processing power, as discussed above, and determine the bitrate of the source content suitable for the concurrent condition. Many video players will start the video requesting the lowest bitrate chunk possible and then working their way up throughout the streaming process. The final product will be the content streamed over HTTP made up of chunks in various quality in order to maximize user experience in a number of devices available. Among the vast majority of ABR algorithms, Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH, is the most prominent.

We will utilize a gym compatible DASH environment [4][4] for Reinforcement Learning model training and testing. Gym, also known as Gymnasium since the entity responsible renamed the product recently, is a common Reinforcement Learning environment framework, establishing a standard environment structure for many algorithms training and deployment. The framework's nomenclature simplifies the interaction between environment and agent, providing readability to developers and standardized structure for a wide variety of environments. In the case of a DASH environment, the Action Space pertains to the number of quality levels available for a video. As for the Observation Space, it is shaped as a Dictionary with 6 keys:

- Vectorized estimated network speed..
- Vectorized chunk's size.
- Vectorized download time of previous chunk ins seconds.
- Scalar value of current buffer size normalized by 10.
- Scalar value indicating the number of remaining chunks normalized by the episode's length.
- Categorical value indicating the last chunk's quality level.

At every step, the agent will choose a quality level based on perceived observation and be rewarded with a corresponding value dictated by a Reward function. The function features mechanisms for bitrate utility, quality switch penalty and rebuffering penalty, making up the Reward function below:

$$r_n = q(B_n) - \mu_s |\, q(B_n) - q(B_{n-1}) \,| - \mu_\phi \phi_n$$

$$q(B_n) = \ln\left(\frac{B_n}{B_{min}}\right)$$

$$|\, q(B_n) - q(B_{n-1}) \,|$$

$$\phi_n = \max(0, d_n - G_n)$$

Where $q(B_n)$ corresponds to utility downloading chunk at step n and B_n to the bitrate of the chunk at step n, $\mu_s |\, q(B_n) - q(B_{n-1}) \,|$ corresponds to the quality switch penalty between the quality of chunk n and the previous chunk, $\mu_\phi \phi_n$ corresponds to the rebuffering penalty, d_n corresponds to the download time taken for the current chunk n and G_n corresponds to the buffer size at the start of step n.

[4] [4]: Long M. Luu, Nghia T. Nguyen, Phuong L. Vo, Tuan-Anh Le "Deep Reinforcement Learning - based Bitrate Adaptations in Dynamic Adaptive Streaming over HTTP".

As for the data used for training and testing models, samples were taken from similar sources to the original Pensieve research paper. Video streaming capabilities of DreamerV2 were trained under real-world circumstances using data collected by the Federal Communications Commission (FCC) from various American ISPs to emulate a broadband environment and data from two major Irish mobile operators to emulate a cellular/4G LTE environment.

FCC dataset [5][5]: The original dataset contains over 1 million throughput traces at a 10 second granularity. Via random selection and concatenation of traces from the "download speed" category, as opposed to the original Pensieve paper, which selected from the "web browsing" category instead, the dataset containing 1000 traces was constructed. Since "web browsing" covers all content of a page which includes more than just the video, selecting throughput traces from "download speed" will better capture the download process of video chunks.

4G LTE dataset [6][6]: The original dataset contains 135 traces at a 1 second granularity across 5 different mobility patterns: static, pedestrian, car, bus and train. Using similar technique in the Pensieve research paper, to match the number of traces within the dataset above, 1000 traces were generated using a sliding window across the whole dataset, each spanning 320 seconds.

Among each of these two datasets, traces were chosen, from a 80 to 20 percentiles split, for training or testing respectively. It should be noted that in order to build a robust, generalized model for video streaming, we decided to combine the train sets of both FCC and LTE together and let the algorithm develop a policy that encompasses both. Test sets are left separate to evaluate the performance of the algorithm in each situation.

4 IMPLEMENTATION AND EVALUATION

4.1 Objective

We used a Pytorch implementation of DreamerV2 written by Raj Ghugare [7][7] as our base for experimentation and evaluation. Training and evaluation will be conducted using primarily Cuda-compatible NVIDIA GPU, type GeForce RTX 3060 Laptop GPU.

During training, once an episode is concluded, the algorithm will decide whether to save the current model and agent through a mean score comparison. Mean score is computed by averaging the latest 100 episodes rewards including the current one. If the calculated mean is higher than the recorded best mean, the algorithm will save the current model configuration as a .pth file, waiting for evaluation. This process will continue until the time step limit has been reached.

Another way the algorithm can save a model configuration is by passing a certain time step benchmark, predefined by hyperparameters. In this instance, we set the value to 50000 steps, meaning that every time the algorithm progresses past the 50000 steps mark, it will save the model configuration as a separate .pth file each time. The reason for this decision is to monitor and record potential model configurations as the best model during training (with best mean score) won't always be the best during evaluation. Once training is done, we will have a model_best file with many model files recorded at every time steps benchmark.

Comparison between Model-free and Model-based Reinforcement Learning algorithms is the crux of this paper. For the other half of this objective, implementation of DQN, A2C and PPO from Stable Baselines 3 Reinforcement Learning library will serve our purpose. Stable Baselines 3 features industry-standard Model-free RL algorithms which are trustworthy, suitable for comparison. By using a recognized library, we ensure that the algorithms will be justified baselines for comparison.

Only through evaluation can we ascertain the effectiveness of our trained models as training records only illustrate partial findings. Trained models will be tested using clean test data from FCC and LTE dataset. We will evaluate each algorithm separately, under separate test datasets; each evaluation attempt will be dedicated to a corresponding training attempt. For each evaluation attempt, trained models of that training attempt will undergo 200 evaluation episodes and trained models are measured through a mean score value averaged from all 200 episodes. The best model will be the representative of that training attempt. Once the best evaluated model is determined, this process will be repeated for an arbitrary number of times (8 in this case) and the final best_mean_score will be the mean of all 8 evaluation attempts.

4.2 Implementation

The common problem plaguing every Reinforcement Learning algorithm, except PPO, is the choice of hyperparameters. Hyperparameters define the structure of many moving components within and there is no surefire way to predict the optimal hyperparameters at the onset. The only solution is to home in on the correct ones through rigorous trial and error. For Stable Baselines 3 implementations, default hyperparameters were chosen [8][8]. Since DreamerV2 was proven to work well in Atari games playing environment, we figured that using those same hyperparameters is a good starting point [9][9].

	FCC best mean	LTE best mean	At time steps
DreamerV2_Default	0.882 ± 0.005	0.568 ± 0.026	50000

It is clear from this trained model that the original Atari games hyperparameters are unsuitable for video streaming, that changes

[5] [5]: Federal Communications Commission (2021) "Measuring Fixed Broadband – Tenth Report".
[6] [6]: Darijo Raca, Jason J. Quinlan, Ahmed H. Zahran, et al (2018) "Beyond throughput: a 4G LTE dataset with channel and context metrics".
[7] [7]: Raj Ghugare "Dreamer -v2 Pytorch": https://github.com/RajGhugare19/dreamerv2.
[8] [8]: Default hyperparameters are listed in the official documentation and written below: https://stable-baselines3.readthedocs.io/en/master/.
[9] [9]: Default hyperparameters are written below.

must be made if we are to develop a competent Reinforcement Learning algorithm.

In order to discover the theoretical optimal hyperparameters, multiple training attempts, with adjustments to core hyperparameters, were done. In the end, the optimal hyperparameters for the video streaming environment were found after numerous fine-tuning and training [10][10] shown below. To ensure optimality, hyperparameters were adjusted with the supposed optimal configuration at the time of comparison as the base.

Regarding the RSSM configuration, the size of the deterministic state will dramatically impact the quality of resulting stochastic states since information of deterministic state are needed to both construct the posterior, with additional information taken from encoded observation, and prior stochastic states. The following showcases the quality of trained models with differing deterministic size:

	FCC best mean	LTE best mean	At time steps
Deter_size = 100	0.763 + 0.004	0.406 + 0.010	1250000
Deter_size = 200 (optimal)	0.948 ± 0.003	0.693 ± 0.012	450000
Deter_size = 300	0.920 ± 0.004	0.577 ± 0.011	400000

Another aspect of the RSSM involves the stochastic states, specifically how should those states treat input observations. Namely, how much should those input observations be compressed into categories and classes to maximize learning potential. We are concerned with the size of those variables in this case. Multiple training attempts were carried out to evaluate changes of this hyperparameter:

	FCC best mean	LTE best mean	At time steps
Class_size, Category_size = 7	0.865 ± 0.009	0.546 ± 0.027	200000
Class_size, Category_size = 16	0.864 ± 0.003	0.603 ± 0.012	800000
Class_size, Category_size = 20 (optimal)	0.948 ± 0.003	0.693 ± 0.012	450000
Class_size, Category_size = 25	0.854 ± 0.020	0.540 ± 0.034	250000
Class_size, Category_size = 32	0.633 ± 0.005	0.325 ± 0.019	450000

In regard to model training, we chose two hyperparameters that have an effect: the number of train steps, effectively the frequency of training, and the number of horizons collected for training. As for training, adjusting the number of train steps will directly impact the quality of the models. However, since the video streaming environment has consistent a step count for each episode, different from the Atari game environment, another choice is to train the model at the end of an episode. Batch size and sequence length were also adjusted to match the train steps for each choice (60 for end of episode training). The results are as follow:

	FCC best mean	LTE best mean	At time steps
TrainSteps = 30	0.952 ± 0.007	0.644 ± 0.018	150000
TrainSteps = 90	0.856 ± 0.006	0.582 ± 0.013	150000
TrainSteps = End Eps (optimal)	0.948 ± 0.003	0.693 ± 0.012	450000

Whenever we train a policy-based Reinforcement Learning algorithm, gradient is always the focus point of the optimization process. Since Actor-Critic is a hybrid, part of the algorithm heavily utilizes gradient for optimization. It is unknown whether there are any vanishing or exploding gradient phenomenon occurring during training. To evaluate this, variations of the Grad_clip hyperparameter value were chosen:

	FCC best mean	LTE best mean	At time steps
Grad_clip = 100	0.937 ± 0.005	0.606 ± 0.012	150000
Grad_clip = 0.5 (optimal)	0.948 ± 0.003	0.693 ± 0.012	450000

The recorded training reward curve of DreamerV2 with the optimal hyperparameters configuration has the following shape:

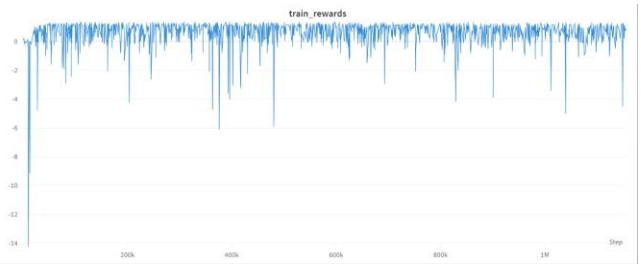

Figure 4: A chart showcasing rewards obtained from each episode in an approximate 1100000 time steps of the DreamerV2

4.3 Evaluation

Our DreamerV2 trained model will be evaluated against Stable Baselines 3 implementations of DQN. A2C and PPO trained in the same environment using default configurations available in their documentation. Those implementations are supplied with the same

[10] [10]: Optimal hyperparameters are written below.

source of data as DreamerV2. As a reference, here are the results obtained from identical evaluation process of DQN, A2C and PPO:

	FCC best mean	LTE best mean	At time steps
DQN	0.935 ± 0.001	0.691 ± 0.009	1000000
A2C	0.950 ± 0.002	0.620 ± 0.004	1000000
PPO	0.993 ± 0.002	0.643 ± 0.005	1000000

Overall, DQN offered the best performance on LTE dataset while PPO offered the best on FCC one. Among the three, A2C left a lot to be desired, achieving second best mean on FCC but the worst on LTE.

After many training and evaluation attempts of DreamerV2, we have produced a relatively competent DreamerV2 model on our pre-discussed video streaming environment. It is inconclusive whether this is the best possible DreamerV2 model for this type of environment, as the number of hyperparameters combinations is astronomical, but with the current configuration, we can conclude that DreamerV2 is perfectly functional on an environment very different from its original concept, which is Atari games playing. Model-based Reinforcement Learning is applicable to more than just their original intended environments. However, the algorithm will not be the definitive solution to ABR. There will be situations where certain Model-free methods will perform better as they are specialized for these types of environments.

4.3.1 Comparison between DreamerV2 and A2C.

This is an important comparison since DreamerV2 utilizes Actor Critic as its agent learning algorithm. Essentially, we are comparing a Model-based implementation of Actor Critic against a Model-free variant. The records of the best mean reward achieved during training are as follows:

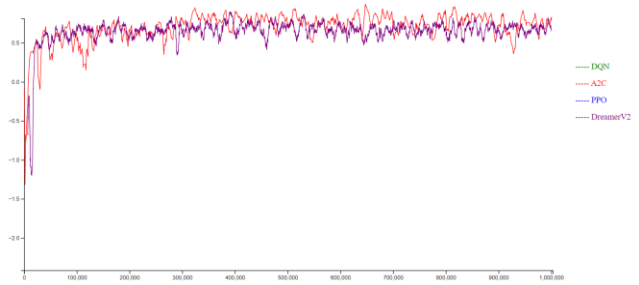

Figure 5: The chart illustrating the best mean reward achieved (the y axis) during training plotted over time steps (the x axis) of DreamerV2 and A2C.

Model-free A2C and DreamerV2 both have very similar trajectory. As for the evaluation results, both trained models representing each approach yield the following:

	FCC best mean	LTE best mean	Average
DreamerV2	0.948 ± 0.003	0.693 ± 0.012	0.821
A2C	0.950 ± 0.002	0.620 ± 0.004	0.785

While the results on FCC test dataset are comparable to one another, there is a vast difference between the results on LTE test dataset. Evaluation scores are the decider on whether an algorithm will perform better or not. Thus, we can definitively conclude that the Model-based Reinforcement Learning algorithm, DreamerV2, outperforms cutting-edge Model-free A2C currently available in public.

4.3.2 Comparison between DreamerV2 and DQN.

On the other hand, DQN was seen to have incredible test LTE result, but we also have this current DreamerV2 model that performed remarkably on the same test set. According to the training record, these are the obtained mean reward.

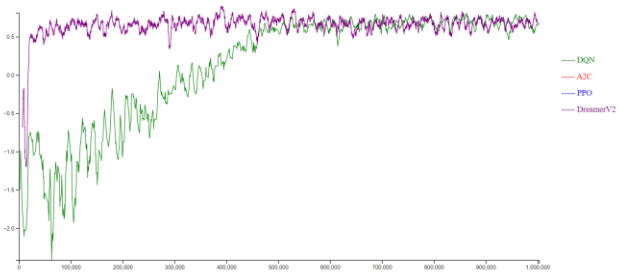

Figure 6: The chart illustrating the best mean reward achieved (the y axis) during training plotted over time steps (the x axis) of DreamerV2 and DQN

Such trajectory is characteristic of DQN, but both seem to converge on similar mean reward value after the halfway point. Below are the evaluation results:

	FCC best mean	LTE best mean	Average
DreamerV2	0.948 ± 0.003	0.693 ± 0.012	0.821
DQN	0.935 ± 0.001	0.691 ± 0.009	0.813

Contrary to Stable Baselines 3 A2C, the difference here is on the other parameter. DreamerV2 and DQN scored similarly on LTE dataset, but DreamerV2 achieved higher noticeable results on FCC dataset. In conclusion, DreamerV2 is an improvement over DQN.

4.3.3 Comparison between DreamerV2 and PPO.

PPO is the default Reinforcement Learning algorithm for OpenAI, meaning that the algorithm is extremely effective. The evaluation results of PPO have also proven this point. We are

curious how DreamerV2 will measure to this state-of-the-art Reinforcement Learning algorithm.

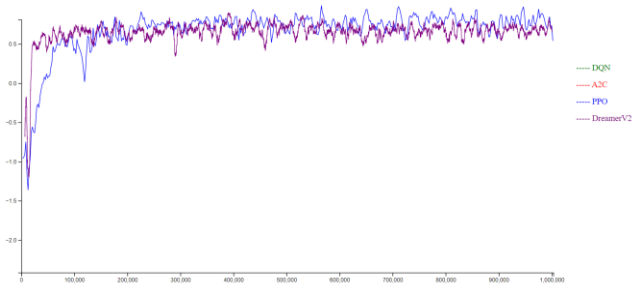

Figure 7: The chart illustrating the best mean reward achieved (the y axis) during training plotted over time steps (the x axis) of DreamerV2 and PPO.

From this chart, we can determine that PPO trajectory is slightly higher than DreamerV2 after the halfway point. The evaluation results are thus:

	FCC best mean	LTE best mean	Average
DreamerV2	0.948 ± 0.003	0.693 ± 0.012	0.821
PPO	0.993 ± 0.002	0.643 ± 0.005	0.818

The distinction is less clear here than with DQN and A2C. While DreamerV2 evaluated better on LTE dataset, PPO achieved higher performance on FCC dataset in equal measure. These 2 algorithms specialize in different aspects of video streaming.

4.4 Summary

Overall, DreamerV2 leverages the efficiency of DQN on LTE network conditions while still maintaining its A2C FCC performance, which is the 2nd best among those 3 Stable Baselines 3 implementations. However, PPO and DreamerV2 offer different values to the video streaming environment. PPO specializes in FCC network conditions while DreamerV2 specializes in LTE network conditions. It is inaccurate to conclude that one is better than the other.

5 Conclusion

In conclusion, we have proven that Model-based Reinforcement Learning can indeed function outside of their designed environment. This paves the future to further Model-based implementations to various Model-free tasks. To assess which is more effective and which is less, that is for future research to elucidate. All we have concluded is that the potential of viable Model-based models within Model-free tasks should not be overlooked. Model-based Reinforcement Learning can function beyond just Atari games playing, even though those represent the highest performance ceiling of Model-based Reinforcement Learning at the moment. Based on our findings, DreamerV2 will make for an excellent ABR algorithm in DASH video streaming environment.

There are many paths we can take from here regarding Model-based Reinforcement Learning. We could explore more environments that Model-based Reinforcement Learning algorithms can potentially function in. Since Model-based was proven to be effective in DASH video streaming environment, perhaps we could use this as a testing ground for other Model-based Reinforcement Learning algorithms as it is an ever-growing field of study. On the other hand, Reinforcement Learning has advanced so tremendously that better algorithms can be discovered in a matter of days. The spotlight has stimulated rapid growth for the branch of study.

ACKNOWLEDGMENTS

Ever since the beginning, it is with the guidance of my advisor, Dr. Vo Thi Luu Phuong, and other members of the laboratory, especially Mr. Nguyen Trung Nghia, that this paper can reach its completion. It is with deep gratitude and respect that I acknowledge their continuous support throughout. Without their expertise and insights, I could not have cultivated ideas to foster the growth of my paper. Additionally, I am grateful to the faculty of the School of Computer Science of the International University for providing technical support and guidelines to follow with. Gratitude is also expressed to the members of my reading and examination committee.

REFERENCES

[1] Hongzi Mao, Ravi Netravali, Mohammad Alizadeh (2017) "Neural Adaptive Video Streaming with Pensieve".
[2] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, Jimmy Ba (2021) "Mastering Atari With Discrete World Models".
[3] Lars Buesing, Théophan Weber, Sébastien Racanière, et al (2018) "Learning and Querying Fast Generative Models for Reinforcement Learning".
[4] Long M. Luu, Nghia T. Nguyen, Phuong L. Vo, Tuan-Anh Le "Deep Reinforcement Learning - based Bitrate Adaptations in Dynamic Adaptive Streaming over HTTP".
[5] Federal Communications Commission (2021) "Measuring Fixed Broadband – Tenth Report".
[6] Darijo Raca, Jason J. Quinlan, Ahmed H. Zahran, et al (2018) "Beyond throughput: a 4G LTE dataset with channel and context metrics".
[7] Raj Ghugare "Dreamer -v2 Pytorch": https://github.com/RajGhugare19/dreamerv2.
[8] Stable Baselines 3 official documentation: https://stable-baselines3.readthedocs.io/en/master/.

APPENDIX

Hyperparameters	Value
Batch_size	50
Seq_len	50
Embedding_size	200
Rssm_node_size	200
Rssm_info	field(default_factory=lambda:{'deter_size':600, 'stoch_size':256, 'class_size':32, 'category_size':32, 'min_std':0.1})
Grad_clip	100
Discount	0.995
Lambda_	0.95

Horizon	15
Lr	field(default_factory=lambda:{'model':2e-4, 'actor':4e-5, 'critic':1e-4})
Loss_scale	field(default_factory=lambda:{'kl':0.1, 'reward':1, 'discount':5})
Kl	field(default_factory=lambda:{'use_kl_balance':True, 'kl_balance_scale':0.8, 'use_free_nats':False, 'free_nats':0})
Slow_target_update	100
Slow_target_fraction	1
Actor	field(default_factory=lambda:{'layers:4, 'node_size':400, 'dist':'one-hot', 'min_std':1e-4, 'int_std':5, 'mean_scale':5, 'activation':nn.ELU})
Critic	field(default_factory=lambda:{'layers':4, 'node_size':400, 'dist':'normal', 'activation':nn.ELU})
Expl	field(default_factory=lambda:{'train_noise':0.4, 'eval_noise':0, 'expl_min':0.05, 'expl_decay':7000, 'expl_type':'epsilon_greedy'})
Actor_grad	'reinforce'
Actor_grad_mix	1
Actor_entropy_scale	1e-3
Obs_encoder	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':None, 'activation':nn.ELU, 'kernel':3, 'depth':16})
Obs_decoder	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':'normal, 'activation':nn.ELU, 'kernel':3, 'depth':16})
Reward	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':'normal', 'activation':nn.ELU})
Discount	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':'binary', 'activation':nn.ELU, 'use':True})

Figure 8: Default hyperparameters of DreamerV2

Hyperparameters	Value
Batch_size	60
Seq_len	60
Embedding_size	200
Rssm_node_size	200
Rssm_info	field(default_factory=lambda:{'deter_size':200, 'stoch_size':200, 'class_size':20, 'category_size':20, 'min_std':0.1})
Grad_clip	0.5
Discount	0.99
Lambda_	0.95
Horizon	10

Lr	field(default_factory=lambda:{'model':2e-4, 'actor':4e-5, 'critic':1e-4})
Loss_scale	field(default_factory=lambda:{'kl':0.1, 'reward':1, 'discount':5})
Kl	field(default_factory=lambda:{'use_kl_balance':True, 'kl_balance_scale':0.8, 'use_free_nats':False, 'free_nats':0})
Slow_target_update	100
Slow_target_fraction	1
Actor	field(default_factory=lambda:{'layers:3, 'node_size':100, 'dist':'one-hot', 'min_std':1e-4, 'int_std':5, 'mean_scale':5, 'activation':nn.ELU})
Critic	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':'normal', 'activation':nn.ELU})
Expl	field(default_factory=lambda:{'train_noise':0.4, 'eval_noise':0, 'expl_min':0.05, 'expl_decay':7000, 'expl_type':'epsilon_greedy'})
Actor_grad	'reinforce'
Actor_grad_mix	0
Actor_entropy_scale	1e-3
Obs_encoder	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':None, 'activation':nn.ELU, 'kernel':3, 'depth':16})
Obs_decoder	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':'normal, 'activation':nn.ELU, 'kernel':3, 'depth':16})
Reward	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':'normal', 'activation':nn.ELU})
Discount	field(default_factory=lambda:{'layers':3, 'node_size':100, 'dist':'binary', 'activation':nn.ELU, 'use':True})

Figure 9: Optimal hyperparameters of DreamerV2

Hyperparameters	Value
Policy	"MultInputPolicy"
Learning_rate	0.0001
Buffer_size	1000000
Learning_starts	50000
Batch_size	32
Tau	1
Gamma	0.99
Train_freq	4
Gradient_steps	1
Replay_buffer_class	None
Replay_buffer_kwargs	None
Optimize_memory_usage	False
Target_update_interval	10000
Exploration_fraction	0.1

Exploration_initial_eps	1
Exploration_final_eps	0.05
Max_grad_norm	10

Figure 10: Default hyperparameters of Stable Baselines 3 DQN

Hyperparameters	Value
Policy	"MultInputPolicy"
Learning_rate	0.0007
N_steps	5
Gamma	0.99
Gae_lambda	1
Ent_coef	0
Vf_coef	0.5
Max_grad_norm	0.5
Rms_prop_eps	1e-05
Use_rms_prop	True
Use_sde	False
Sde_sample_freq	-1
Normalize_advantage	False

Figure 11: Default hyperparameters of Stable Baselines 3 A2C

Hyperparameters	Value
Policy	"MultInputPolicy"
Learning_rate	0.0003
N_steps	2048
Batch_size	64
N_epochs	10
Gamma	0.99
Gae_lambda	0.95
Clip_range	0.2
Clip_range_vf	None
Normalize_advantage	True
Ent_coef	0
Vf_coef	0.5
Max_grad_norm	0.5
Use_sde	False
Sde_sample_freq	-1
Target_kl	None

Figure 12: Default hyperparameters of Stable Baselines 3 PPO