Department of Creative Informatics
Graduate School of Information Science and Technology
THE UNIVERSITY OF TOKYO

Master's Thesis

Multi-Agent Simulated Environments Generated by a Transformer-Based Generative Model
トランスフォーマーを応用した生成モデルによるマルチエージェントシミュレーション環境の生成

Masao Taketani
竹谷 昌夫

Supervisor: Associate Professor Hideki Nakayama

January 2024

Abstract

Current simulators are mainly created by human experts such as graphics and algorithm engineers, which requires lots of time. Furthermore, even though graphics technologies have been improving, it is still near impossible to create realistic simulated environments that are indistinguishable from the real world by human eyes. To look for an effective way to create realistic simulators, deep generative models that recently have been gaining the ability to generate photo-realistic images or even videos come to our mind. Considering the current advancement and popularity of generative models, they have the potential to create simulated environments akin to the real world and can be trained in a data driven manner, instead of humans implementing the logic of the real world dynamics. Some studies previously and successfully created simulated environments using generative models. Although the quality of generated images has been ameliorated since the first model of this field was proposed, we can still see the results lacking temporal consistency in the created simulators. Looking at the architectures, all of the current state-of-the-art models rely on LSTMs to retain temporal information, which is recently known to have a strong alternative, Transformer that has been well known to effectively handle temporal information since the advent of Transformer-based language models. Thus, one of the objectives of our study is to investigate how the temporal consistency of the generated dynamics is affected by using a Transformer-based architecture, instead of LSTM. Also, the current created environments are only applicable to single agent, which does not resemble the real world. Oftentimes agents have to cooperate, compete or mix of the two in reality. And in order to train them, we need to have an environment for multiple agents. Therefore, another objective is to make generated environments adaptable to multi-agents. Our conducted evaluations show that our architecture demonstrates convincing temporal dynamics at least as well as the current best model with single-agent datasets and by far the best results in terms of multi-agent datasets both quantitatively and qualitatively. Furthermore, in this research, we have also tested that ours is capable of handling the number of agents up to four without compromising the quality of generated environments.

概要

今現在のシミュレータはグラフィックスやアルゴリズムのエンジニアなど, 主に人間の専門家によって作られており, 作成に多大な時間を要している. また, グラフィックス技術が以前と比べ向上しているとはいえ, それでも未だ現実世界と見分けがつかないくらいのシミュレータを作成することは不可能に近い. そこで, よりリアルなシミュレータを作成するための効果的な方法を探るべく, 我々は近年フォトリアルな画像や動画さえも生成する能力を獲得しつつある深層生成モデルに着目する. 生成モデルの現在の進歩や高い評価を考慮すると, 現実世界に近いシミュレーション環境を作成できる可能性があり, 現実世界の動的ロジックを人間が実装する代わりに, データ駆動型の方法で学習させ獲得できる可能性を秘めている. これまでの先行研究では, 生成モデルを用いて模擬環境を作成することに成功した例がいくつかある. それらから生成された環境の質を見るに, 生成された画像の質は徐々に改善されているものの, 時間的に整合性を欠く結果が見られる. 使用されているアーキテクチャを見ると, 全ての既存提案手法において, 時間情報を保持するために古くからある LSTM を活用している. そこで時間的一貫性を向上させる目的として我々の研究の目的の一つは,LSTM の代わりに近年言語モデルの発展により注目される Transformer ベースのアーキテクチャを用いることにより, 生成されたダイナミクスの時間的一貫性にどのような影響を与えるかを調べることである. また, 現在作成されている手法は単一エージェント環境にしか適用しておらず, 現実の世界とは似て非なる. 現実世界では複数のエージェントで協力したり, 競争したり, あるいはその二つを兼ね合わせたりする必要がある. そしてそれらのエージェントを学習させるためには, 複数エージェント用の環境が必要である. したがって, 本研究のもう一つの目的は, 生成された環境をマルチエージェントに対応させることである. 我々の行った研究結果は, 我々の提案手法が単一エージェント環境生成において少なくとも現在最高精度のモデルと同じぐらいの時間的整合性のあるダイナミクスを生成し, またマルチエージェント環境の生成において定量・定性の両面で群を抜いて良い結果を示した. 更に本研究では, 我々の提案手法が生成した環境の質を落とすことなく, 最大４つのエージェントまで拡張させれることも確認した.

Contents

Chapter 1

Introduction

1.1 Background

Since the advent of deep learning, there have been several kinds of learning methods that incorporate the technology being introduced. Most notable ones are supervised learning, unsupervised learning, and reinforcement learning. Among them, reinforcement learning requires an environment to train an agent with a view to create artificial intelligent systems, which differs from supervised and unsupervised learning in a significant way that supervised learning requires inputs with its corresponding labels and unsupervised learning needs inputs. In this approach, an agent tries to achieve a goal through trial and error inside the environment while storing data such as images and rewards as past experiences and aiming to maximize its future cumulative rewards given by the environment. In short, an environment has to be provided for reinforcement learning algorithms to work although inputs or pairs of an input and label data prepared in advance are unnecessary.

Among the learning methods we mentioned above, we believe reinforcement learning is the most naturalistic way of learning. Having said that, we also believe reinforcement learning is not necessarily the most optimal learning method, but instead the optimal learning method may be something similar to it. Whether we have a clear objective in our lives or not, come to think of how we humans grow up in this world, one of the biggest parts of our learning seems to be interaction with the environment, in our case the Earth, other people, animals, and plants living on the Earth, objects such as trees, walls, appliances and so on. Thus, we have been questioning ourselves that what if AI could also live in the world so that AI could also gain the same concept of the surrounding environment as humans do? In reality, though, it would be unquestionably controversial to place underdeveloped AI agents in our environment and let them explore at their own will, just like it would if we let children freely walk around the environment without parental supervision. Due to the fact, we started thinking that what if we could create a simulated environment of our world for AI agents to play with and learn as a substitute for it? That is how we became interested in creating environments. Since then, as we have been investigating various approaches, we have found out so many things on the way.

As can be seen from recent studies [14, 49, 33, 26, 61, 3], simulation environments are mostly created by humans and, by looking at the environments, those are still far from realistic. On top of that, we also found out that creating simulated environments takes an enormous amount of time and calls for specialists such as graphics and algorithm engineers. Thus, needless to say, hefty amount of money must be invested as well. As a way to tackle the situation, there have been several successful attempts aiming to create simulated environments with deep generative models [19, 31, 30]. What they basically try to do is to replace ground truth environments with pseudo environments that are generated by the generative models. Fig. 1.1 shows a simplified version of what a generative model (in the figure, GameGAN is the generative model) aims to do. As can be seen from Fig. 1.1, all the environments created by previous generative models only tries to emulate gaming environments except for the most recent work [30], which has successfully created simulated environments that are based on real-human-living environments. All the works require collection of images and their corresponding action log to train the models, which, in turn, leads to the models being able to generate a future image based on current and previous images and actions. In this methodology, simulators are created in a data-driven manner, which means human experts don't need to implement the graphics and logic of the dynamics of the engine, which gives rise to saving significant amount of time in order to create the environments. Furthermore, although even the most recent work [30] still can not generate realistic graphics to the extent that humans are unable to differentiate generated environments from real ones, looking at the current rapid pace of progress of deep generative models [23, 63, 56, 52, 57, 77], deep generative models certainly have the potential to be able to generate more realistic environments.

However, aside from the quality of generated graphics (we do not focus on this topic in our research), there are still many issues left unsolved judging by the generated results from the previous works. First of all, all the works lack temporal consistency in generated results from time to time. To give an example, let's say we are trying to create a driving environment in which an agent drives on a road. The environment should be able to generate proper and consistent driving scenes. Suddenly, the driving agent sees a truck ahead of it for a certain period of time, say until time t, but sees a human in stead of the truck at time $t + 1$. In that case, what the agent needs to do greatly differs from what it should do when it seas a truck. In other words, whenever the agent sees a person right in front, it immediately has to stop whereas it needs to keep moving forward if there is a truck moving in the same direction as the agent in front. This kind of fatal error happens if the generated environment is not consistent in handling temporal information, which makes the generated environments unreliable and not worth using.

Second, although current works are created to handle single-agent actions, there must be environments where multiple agents can train at the same time if the goal is to create realistic environments. In the real world, we have a number of occasions where multiple agents have to cooperate, compete or mix of the two to achieve a goal. For example, if the goal of the multiple agents is to go on patrol and save somebody in trouble, when one agent finds a person falling down, one of the things it needs to do right away is to check up on what is happening to the person and provide emergency treatment if needed while letting another agent know the situation and have it call for help such as an ambulance if the situation is serious. And another agent has to know how to call, where to call, and what to convey based on the given piece of information from the first agent. This is one example of cooperation, and as a means to train multiple agents cooperatively, we need training environments for them.

Other issues are that models cannot generate unseen situations that markedly diverge from training data distribution, there is no sound, there is no language interaction, etc. However, we leave those for future research and only focus on the two issues described in the previous paragraphs for our research.

1.2 Objectives

Our future goal is to eventually create realistic environments, but due to the fact that we did not have the resources to take advantage of real-human-living data, we conduct our research using human-created environments as ground truth environments.

As stated in the previous section, we aim to address the two problems, improvement

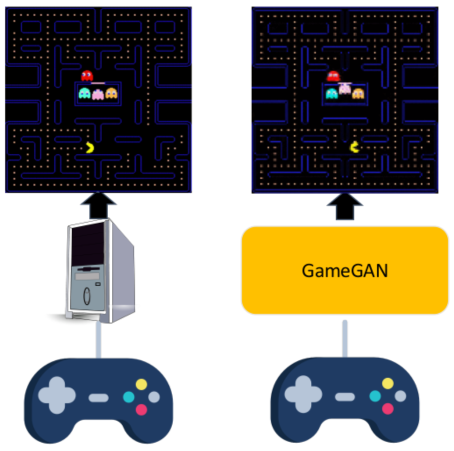

Fig. 1.1. One of the previous works, GameGAN, illustrates the overview of how a created environment by a generative model can be replaced with ground truth one on the right side, as opposed to directly using the ground truth game engine on the left side. Cited from [31].

of temporal consistency and incorporation of multiple-agent actions into generated environments. As for the capability to generate temporal consistent images, all the works previously done rely on LSTM-based modules [19, 31, 30]. Nevertheless, after Transformer [73] was introduced, many of the models that formerly had made use of LSTM started replacing it with Transformer and gained better results [73, 50, 12, 1, 48]. Therefore, here comes a motivation to incorporate Transformer into this field of research with the hope of our proposed model handling temporal dynamics better than the previous works.

In regard to multi-agent environments, as inspired by multimodal transformers [65, 8, 48, 80], we aim to incorporate multi-agent actions by tweaking the idea of the architectures. With this method, a user can easily scale up the number of agents (in our experiments, we made an attempt to scale up to four agents. For more details, see Chapter 5).

As a way to evaluate the quality of generated environments, we conduct both quantitative and qualitative evaluations to see how our method performs compared to the state-of-the-art (SOTA) ones using both single-agent and multiple-agent datasets.

1.3 Structure of Thesis

After this chapter, the thesis is organized as follows.

Chapter 2 Preliminaries
: We introduce the preliminary knowledge regarding reinforcement learning environments and deep generative models.

Chapter 3 Related Work
: We provide several leading works that have attempted to create simulated environments with generative models.

Chapter 4 Proposal
: We propose our approach with a view to effectively handling temporal dynamics and multi-agent actions.

Chapter 5 Experiments
: We conduct both quantitative and qualitative evaluations using both single-agent and multiple-agent datasets and analyse prominent features of our model.

Chapter 6 Conclusion and Future Work
: We conclude the work with the final chapter and talk about some other possible directions to create realistic simulators.

Chapter 2

Preliminaries

2.1 Reinforcement Learning Environments

2.1.1 Single-agent Environments

As discussed in chapter 1, we need to prepare an environment for an agent to train in reinforcement-learning (RL). The way it works in the environment is that the environment renders a state to the agent (or agents for multi-agent RL case. We will discuss multi-agent case in 2.1.2). Then the agent decides what action to take based on the given state information by the environment. In a normal case, there is also something called reward given by the environment, but we omit it since it is not considered in our research. Fig. 2.1 shows simplified interaction between an agent and an environment.

From now, we will discuss something related to one of the core concepts of our research when it comes to creating an environment. A state given by an environment at each time step is based on previous states and actions. Suppose that previous states that already have been rendered by the environment and actions that so far have been taken by an agent living in the environment start at time 0, s_0 and a_0, and go through all the way up to at time t, s_t and a_t. The environment now has to render a temporally consistent state based on $s_0, \dots, s_t$ and $a_0, \dots, a_t$. The environment can be either deterministic or stochastic. In mathematical notation, it is written as below for the deterministic case.

$$s_{t+1} = f(s_0, \dots, s_t, a_0, \dots, a_t), \tag{2.1}$$

where f is the deterministic function that decides the next state given previous states and actions. As for the stochastic one, it is written as follows.

$$s_{t+1} \sim p(s_{t+1} | s_0, \dots, s_t, a_0, \dots, a_t), \tag{2.2}$$

where $p(s_{t+1} | s_0, \dots, s_t, a_0, \dots, a_t)$ is the probability of the next state conditioned by the previous states and actions. This way of thinking is crucial in order to create plausible environments because a created model that emulates the real environment should also behave based on the same condition. As a way to incorporate the functionality, sequence prediction model is usually applied. As to what kinds of sequence models are used, We will discuss the details in 2.2.4.

2.1.2 Multi-agent Environments

Extensive RL algorithms have been invented in the past, but most of the well-known algorithms are for single agent [76, 66, 45, 72, 75, 2, 44, 58, 59]. In contrast, there is another movement called multi-agent reinforcement learning (MARL) [46, 38], which is a branch of RL. As the name suggests, MARL is RL for multiple agents. As for an environment of

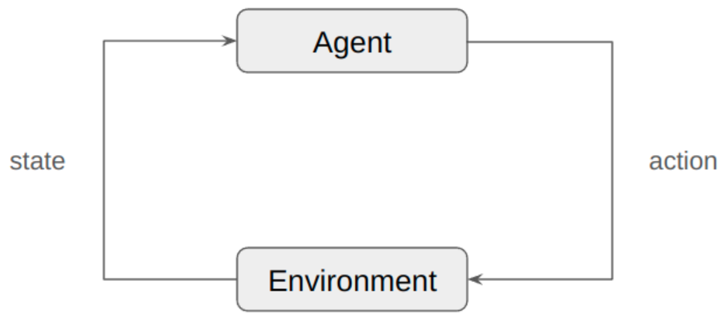

Fig. 2.1. Interaction between an environment and an agent. The environment provides a state to the agent, and the agent reply with its action in return.

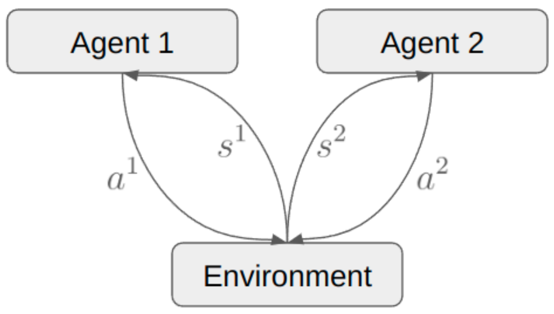

Fig. 2.2. Interaction between an environment and multiple agents. This picture shows an example of having two agents in the environment. s^1 and s^2 are the states given to the agent 1 and agent 2 respectively and a^1 and a^2 are the actions taken by agent 1 and agent 2 respectively. s^1 and s^2 can also be the same for some environments. We only handle the latter for our research.

MARL, it emits a state or states to multiple agents that interact with the environment at each time step. Fig. 2.2 depicts an interaction between a MARL environment and multiple agents (the example shows a case that there are two agents in the environment). Here comes a new concept that also applies to RL for single agent. What kind of state is given depends on whether the entire environment is fully or partially observable. There are technical terms for each type. It is called a state if the environment is fully observed and anther one is called an observation if it is partially observed. Regarding our research, we just focus on handling the former, which results in an environment given to the agents being fully-observable, which, in turn, means all the images given to the agents are equal for each time step. The reason why we chose the former is that, to the best of our knowledge, no one has ever even made an attempt to generate either of the types for a multi-agent setting and the former has to be completed first to move on to more complicated latter environments. Fig. 2.3 exhibits an example for each type.

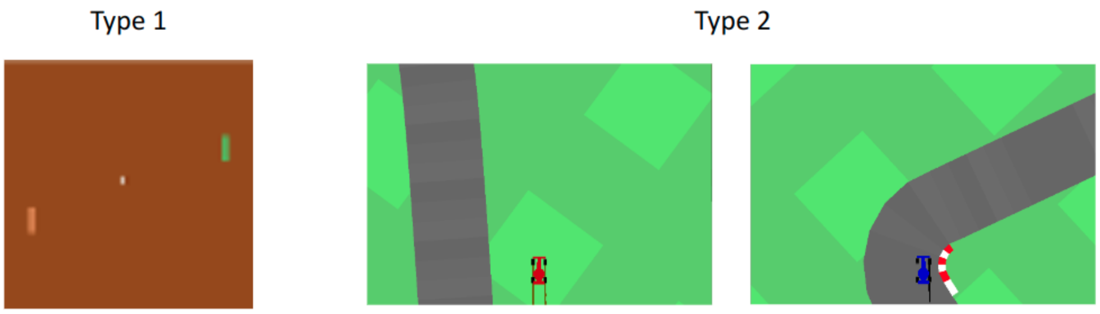

Fig. 2.3. Examples of two types of MARL environments. Type 1 shows an example that an entire environment is fully observable and shared with all the agents. Cited from [69]. Type 2 shows another case that a partial part of the state of the environment is observable to each agent. Cited from [60]. We only use type 1 environments for our research.

2.2 Deep Generative Models

2.2.1 Generative Model Overview

As mentioned in 1.1, the quality of recent deep-learning-based generative models of images and videos has been impressive. Furthermore, so many kinds of acclaimed models have been introduced with the development of deep learning, such as Variational Autoencoders (VAE) [32, 55], Autoregressive Models (AR) [71, 47, 73], Generative Adversarial Networks (GAN) [16], Flow-based Models [54, 13], Energy-based Models [68, 34], and ones that are recently favored by many, Diffusion Models [72, 23]. Each model has its own strengths and weaknesses [4, 5]. Therefore, users have to wrap their heads around the facts and consider what requirements need be met in order to achieve their goals before deciding which algorithms to adopt. As for our research, considering the fact that a generator has to sample a frame at each time step, having faster sampling speed, such as real time or even faster, is preferred. Other requirements are having decent quality of generated images and consistently handling temporal information. Based on the criteria and also related works of this field, we discuss VAE, GAN, and AR in this section.

2.2.2 Variational Autoencoders

VAE is one of the most well-known generative models that aims to obtain and utilize latent variables, which are not explicitly given by data, in order to generate pseudo data that is similar to real data. To do that, two modules named probabilistic encoder and probabilistic decoder are used. What probabilistic encoder does is to encode observable variables into latent variables and what probabilistic decoder does is to generate pseudo data utilizing latent variables. The reason that a term probabilistic is used for each module is that the outputs of both of the modules are probability distributions rather than deterministic values. In mathematical terms, let us denote the output probability of the encoder as $q_\phi(\mathbf{z}|\mathbf{x})$, where ϕ is the parameters of the encoder, $\mathbf{z}$ is the latent variables, and $\mathbf{x}$ is observable variables. Next, we denote the output probability of the decoder as $p_\theta(\mathbf{x}|\mathbf{z})$, where θ is the parameters of the overall generative model. So as to train the encoder and the decoder so that the decoder can generate plausible data utilizing encoded latent variables extracted by the encoder after training, we need to have

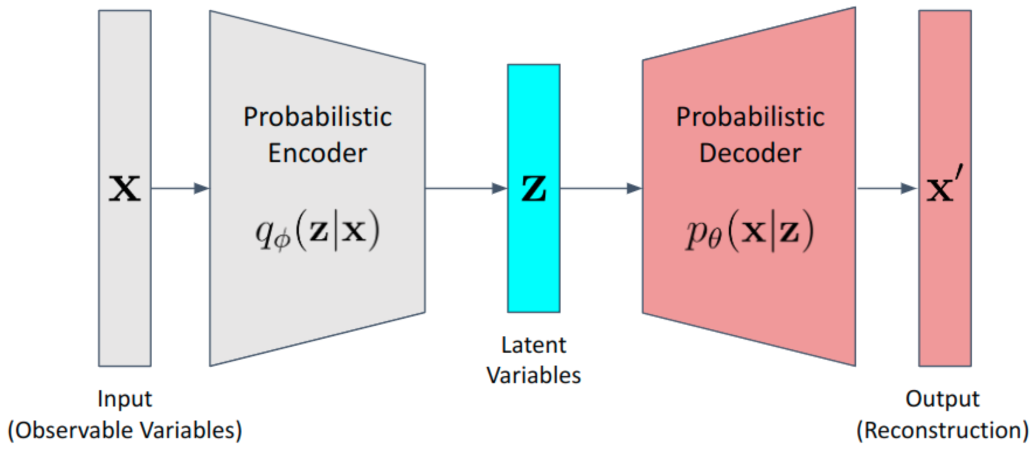

Fig. 2.4. VAE architecture. Probabilistic encoder $q_\phi(\mathbf{z}|\mathbf{x})$ encodes an input image $\mathbf{x}$ into latent variables $\mathbf{z}$. Probabilistic decoder $p_\theta(\mathbf{x}|\mathbf{z})$ generates an image $\mathbf{x}'$ aiming to reconstruct the original image.

an objective. The objective used for VAE is log-likelihood maximization. Here, let us describe the probability of observable variables as $p_\theta(\mathbf{x})$. Our goal is to maximize the probability. Since calculation becomes easier, let us take a logarithm of that probability. Then, we can derive the following equation.

$$
\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} && (2.3)\\
&= \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} && (2.4)\\
&= \log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] && (2.5)\\
&\geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] && (2.6)\\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right] && (2.7)\\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{z}) \right] && (2.8)\\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] - D_{KL} \left(\log q_\phi(\mathbf{z}|\mathbf{x}) || \log p_\theta(\mathbf{z}) \right), && (2.9)
\end{aligned}
$$

where D_{KL} is the Kullback–Leibler divergence. The inequality at (2.6) comes after applying Jensen's inequality. The term $p_\theta(\mathbf{z})$ called prior is usually set as standard normal distribution based on a prior assumption and calculation convenience. What the derivation tells us is that we first started with the objective of maximizing the log-likelihood of $p_\theta(\mathbf{x})$. However, because it is too intractable to calculate the marginal distribution, the objective is shifted from maximizing the log-likelihood to maximizing the lower bound of the log-likelihood using Jensen's inequality. The lower bound is named as the evidence lower bound (ELBO). In order to maximize the ELBO, we need to come up with higher value for the fist right-hand side (RHS) term of equation (2.9), which is the probability of observable variables when latent variables are given, and lower value for the second RHS term of it, which is the Kullback–Leibler divergence between the probability of latent variables given observable variables, also called posterior, and the prior. Fig. 2.4 depicts the overall architecture of VAE.

2.2.3 Generative Adversarial Networks

GAN is another kind of prominent generative model. The mechanism is that there are two roles as a way to produce a generative model. One role is called generator, which tries to generate pseudo data as the name suggests, and another one is called discriminator, which exerts itself to detect which data comes from real and which one comes from fake. We will eventually get a well-trained generator and discriminator while letting them compete against each other if the training goes well. To be more specific about the concept, let us denote the generator as $G(z)$, where z represents noise, and the discriminator as $D(x) \in [0, 1]$, where x is real data, such as images. Then, the objective of GANs comes down to optimize the following value function $V(G, D)$.

$$\min_G \max_D V(G,D) = \min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_{\text{z}}(z)}[\log(1 - D(G(z)))], \quad (2.10)$$

where $p_{\text{data}}(x)$ is the real data distribution and $p_{\text{z}}(z)$ is distribution of noise, which is commonly defined as either standard normal distribution or uniform distribution. The discriminator is trained to output 1 if the given input comes from real data and 0 if it comes from fake one, which leads to the maximization of equation (2.10) with respect to D. As for the generator, since its job is to generate realistic data to the extent that it would deceive the discriminator as if data created by the generator came from the real one, minimization of equation (2,10) with respect to G is a way to optimize the generator.

In terms of the value function, we can rewrite it as follows.

$$\begin{aligned} V(G,D) &= \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_{\text{z}}(z)}[\log(1 - D(G(z)))] && (2.11)\\ &= \int_x p_{\text{data}}(x)\log D(x)dx + \int_z p_{\text{z}}(z)\log(1 - D(G(z)))dz && (2.12)\\ &= \int_x p_{\text{data}}(x)\log D(x) + p_g(x)\log(1 - D(x))dx. && (2.13) \end{aligned}$$

Equation (2.13) can be obtained after applying change of variables with an assumption such that $p_g(x)dx = p_{\text{z}}(z)dz$. Then, by taking a derivative of (2.13) with respect to D and setting the derivative equal to 0, we get the optimized discriminator $D^*(x)$ as below.

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}. \quad (2.14)$$

Inserting the optimized discriminator into the value function, we get the following derivation.

$$\begin{aligned} V(G,D^*) &= \int_x p_{\text{data}}(x)\log D^*(x) + p_g(x)\log(1 - D^*(x))dx && (2.15)\\ &= \int_x p_{\text{data}}(x)\log\frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}dx + \int_x p_g(x)\log\frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)}dx && (2.16)\\ &= -\log 4 + \int_x p_{\text{data}}(x)\log\frac{p_{\text{data}}(x)}{\frac{p_{\text{data}}(x)+p_g(x)}{2}}dx + \int_x p_g(x)\log\frac{p_g(x)}{\frac{p_{\text{data}}(x)+p_g(x)}{2}}dx && (2.17)\\ &= -\log 4 + D_{KL}(p_{\text{data}}(x)||\frac{p_{\text{data}}(x) + p_g(x)}{2}) + D_{KL}(p_g(x)||\frac{p_{\text{data}}(x) + p_g(x)}{2}) && (2.18)\\ &= -\log 4 + 2D_{JS}(p_{\text{data}}(x)||p_g(x)), && (2.19) \end{aligned}$$

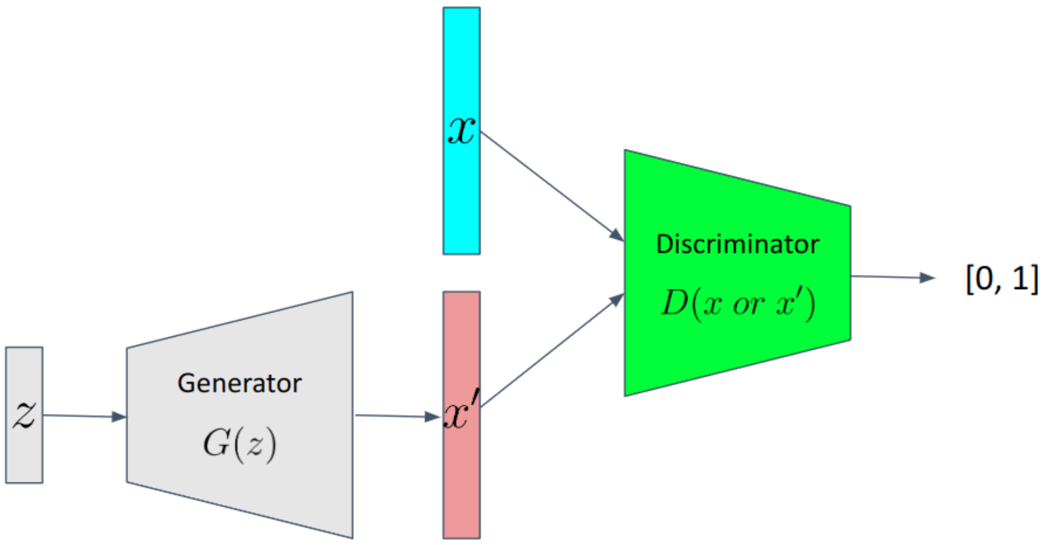

Fig. 2.5. GAN architecture. Generator $G(z)$ tries to generate a realistic data x' accepting noise z. Discriminator strives to discern real data x from x'. The output of Discriminator ranges from 0 to 1, indicating the closer the value is to 1, the more Discriminator thinks the data comes from real distribution, and the closer it is to 0, the more Discriminator believes the data comes from fake distribution.

where D_{JS} denotes Jensen–Shannon divergence. In order to get the optimized generator G^*, we need to minimize (2.19). As a matter of fact, we can observe that GAN includes Jensen–Shannon divergence as a part of the optimization, whereas VAE has Kullback–Leibler divergence as a part of it. See Fig. 2.5 for the overall architecture of GAN.

2.2.4 Autoregressive Models

Autoregressive Model Overview

AR models are created to handle sequential data, such as language and time-series data. They output probability distribution of having data at time T conditioned on the previous time steps $< T$, where $< T$ means all the time steps before T. In mathematical notation, it can be derived as follows.

$$p(\mathbf{X}) = p(\mathbf{x}_0)p(\mathbf{x}_1|\mathbf{x}_0)...p(\mathbf{x}_T|\mathbf{x}_0, ..., \mathbf{x}_{T-1}) \tag{2.20}$$

$$= p(\mathbf{x}_0)\prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{X}_{<t}), \tag{2.21}$$

where $\mathbf{X} = \mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_T$ and $\mathbf{X}_{<t} = \mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{t-1}$. As for this section, We discuss more details of two major AR models that are frequently used and also related to our research. Long Short Term Memory (LSTM) and Transformer.

Long Short Term Memory

LSTM [24] is one specific type of recurrent neural networks (RNN). LSTM demonstrates several advantages over vanilla RNNs in that LSTM possesses several gates and a cell state as a way to avoid gradient vanishing and retain long-term information. The equations of LSTM is described as follows with an input gate $\mathbf{i}_t$, a forget gate $\mathbf{f}_t$, an output gate $\mathbf{o}_t$, and a cell state $\mathbf{c}_t$.

$$\begin{bmatrix}\mathbf{i}_{tmp}\\ \mathbf{f}_{tmp}\\ \mathbf{o}_{tmp}\\ \mathbf{c}_{tmp1}\end{bmatrix} = \begin{bmatrix}\mathbf{W}_i\\ \mathbf{W}_f\\ \mathbf{W}_o\\ \mathbf{W}_c\end{bmatrix}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \begin{bmatrix}\mathbf{b}_i\\ \mathbf{b}_f\\ \mathbf{b}_o\\ \mathbf{b}_c\end{bmatrix} \tag{2.22}$$

$$\mathbf{i}_t = \sigma(\mathbf{i}_{tmp}), \mathbf{f}_t = \sigma(\mathbf{f}_{tmp}), \mathbf{o}_t = \sigma(\mathbf{o}_{tmp}), \mathbf{c}_{tmp2} = \tanh(\mathbf{c}_{tmp1}) \tag{2.23}$$

$$\mathbf{c}_t = \mathbf{i}_t \odot \mathbf{c}_{tmp2} + \mathbf{f}_t \odot \mathbf{c}_{t-1} \tag{2.24}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{2.25}$$

where $\mathbf{W}$s are weights, $\mathbf{b}$s are biases, $\sigma(\cdot)$ is a sigmoid function, $\tanh(\cdot)$ is a hyperbolic tangent function, and $\odot$ is an element-wise multiplication, which is also called Hadamard product.

Transformers

Transformer first appeared as a language translation model [73]. Since then, not only has it been applied to language-related tasks [50, 12, 51, 9], but it also has been utilized to fields such as vision, speech, audio, and time-series analysis, and more [77, 37, 18, 15, 36]. Some of the advantages of Transformer are that it can globally and explicitly attend to every token given to the model, can simultaneously process all tokens rather than sequentially (if a task requires to do autoregressive prediction, it needs to process each token step by step during inference), which leads to a faster processing speed, can easily and ideally be scaled up if the amount of training data suffice, and so on. There are also some disadvantages such that it requires lots of memory because it explicitly holds previous-step information and due to that, it at times has to cut off some of previous tokens with a view to restrict the memory footprint, which, by the way, is done to our proposed architecture, and it also requires huge amount of data in order for it to be well-trained without overfitting.

Transformer can be roughly decomposed into input encoding, self-attention, position-wise feed-forward network, residual connection plus normalization layer, and attention masking (there is also another type of masking for padding to a maximum token length, but since it is not related to this research, we will not explain it). As for the input encoding, input data such as sentences or an image has to be split into pieces, which are called tokens. After the input is split, each split piece, then, needs to be embedded into latent space and summed with positional encoding. [73] proposed a way of the encoding using sinusoidal functions. In mathematical terms, the equations can be written as follows.

$$\mathrm{PE}_{(t,2i)} = \sin(\frac{t}{10000^{\frac{2i}{d}}}) \tag{2.26}$$

$$\mathrm{PE}_{(t,2i+1)} = \cos(\frac{t}{10000^{\frac{2i}{d}}}), \tag{2.27}$$

$$\tag{2.28}$$

where t is a time step, i is i-th dimension of the embedding, and d is the total number of embedding dimension. Learnable positional encoding can also be introduced instead such as [12].

Self-attention module (SA) comes after the input encoding. The encoded input is linearly transformed into three vectors, which are a query $\mathbf{Q} \in \mathbb{R}^{d\times d_q}$, key $\mathbf{K} \in \mathbb{R}^{d\times d_k}$ and value $\mathbf{V} \in \mathbb{R}^{d\times d_v}$, where $d_q = d_k$. Then, those vectors are used to obtain semantically meaningful embedding as follows.

$$SA(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}. \tag{2.29}$$

The another part is the position-wise feed-forward network (FFN). The module usually comes with two linear layers and one activation function in between them. In mathematical terms, let's say the input embedding vector for FFN is denoted as $\mathbf{E}_{\text{FFN}}$. Then, we can write the module as follows.

$$FFN(\mathbf{E}_{\text{FFN}}) = \sigma(\mathbf{E}_{\text{FFN}}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{2.30}$$

where $\mathbf{W}_1$ and $\mathbf{b}_1$ are the weights and biases for the first linear layer, $\mathbf{W}_2$ and $\mathbf{b}_2$ are the weights and biases for the second linear layer, and $\sigma(\cdot)$ is an activation function.

Residual connection and layer normalization (RL) are used for SAs and FFNs so that they prevent Transformer from getting vanishing or exploding gradient problems. Let $\mathbf{M}$ be a module of either SA or FFN and input embedding of RL be $\mathbf{E}_{\text{RL}}$. Then, we can write the function as follows.

$$RL(\mathbf{E}_{\text{RL}}) = Norm(\mathbf{M}(\mathbf{E}_{\text{RL}}) + \mathbf{E}_{\text{RL}}), \tag{2.31}$$

where $Norm$ stands for a normalization layer. Sometimes, a block that sequentially handles an embedding input in the order of SA, RL, FFN, RL is called a transformer block that is applied for a predefined number of times inside of Transformer.

The last part is attention masking. The masking is used to prevent Transformer from paying attention to future information that is supposed to predict by itself. We apply this masking for our research because our transformer architecture is set to simultaneously predict a one-time-step-ahead token for each given token, and without the masking it would cheat on predicting future tokens by referring to the given future tokens during training.

Chapter 3

Related Work

3.1 Simulators by Generative Models

There were some successful attempts aiming to create simulated environments with a generative model. All the works require collection of images and their corresponding action log to train the models, which, in turn, leads to the models being able to generate a future image based on current and previous images and actions.

World Model [19] uses two modules to create a simulated environment. One module is called Vision which utilizes Variational Autoencoder (VAE) [32, 55] to process images into latent codes, and vice versa. The other is called Memory which utilizes LSTM [24] to predict a latent code of a future image when a pair of a current image and an action are given. There are two training phases to create a simulated environment. The first training phase is to train Vision module and the second training phase is to train Memory module.

GameGAN [31] is the first work to incorporate conditional Generative Adversarial Network (GAN) [41] to create simulated environments. On top of that, an external memory module based on Neural Turing Machine (NTM) [17] is optionally applicable for the cases that require to remember places where an agent has visited before. Other notable thing of this work is that it only requires to train the entire modules at once.

DriveGAN [30] applies VAE-GAN [35] to encode and decode latent codes. To disentangle image information, VAE strives to generate latent codes for content and theme and StyleGAN-based [28, 29] decoder aims to generate a future image based on the disentangled latent codes. Since the latent codes are editable at any time thanks to the information disentanglement by VAE, you can generate specific situations to test a trained agent in those kinds of circumstances. As for the training phases, it has two training phases as presented in World Model.

3.2 Sequence Prediction

In order for generative models to capture and convincingly generate dynamics of simulated environments, they at least have to possess some kind of sequential models. In reference to the sequential processing functionality, all the previous works employ LSTM-based modules [19, 31, 30]. Among them, DriveGAN's architecture is composed of complicated LSTM modules. Not only do their work utilize a vanilla LSTM module, but also a convolutional LSTM [62] one is applied so that it also can contain spatial dynamics as a way to adapt to 3D view-point shift. With the two types of LSTMs, the architecture further disentangles the latent codes into action-dependent and action-independent codes in order to generate consistent style and dynamics of the environment.

Our works differs from the previous works in that we apply a totally different architecture when it comes to handling temporal dynamics, which is Transformer [73].

3.3 Multimodal Transformer

Since the conception of Transformer, it has, nowadays, also been known to handle multimodal data [65, 8, 48, 80]. Creating simulated environments can be also considered as multimodal since the model needs to incorporate an image input along with (an) action input(s) in order to predict a plausible future image. Thus, we think multimodal transformer may be effective as a way to manage multimodal inputs. Among the Transformer-based multimodal models, a model named Episodic Transformer (ET) [48] caught our attention. The model which is used for vision and language navigation task effectively accepts language, vision and action inputs and predicts a proper next action to follow a given human instruction. ET expectedly outperformed LSTM-based models in their research. Since our model is also supposed to take on vision and action inputs, the ET-like architecture might have the potential to perform well even for our research. Our proposed model differs from ET in that our model does not handle a language input, but multi-agent actions, instead. Furthermore, the task of our model is to predict a future image in lieu of an action, which is much harder to predict.

Chapter 4

Proposal

4.1 Overview

In this work, as a way to create novel simulated environments, we propose two things. First, we incorporate Transformer for our base model to handle temporal information instead of LSTMs, which are previously used for all the related works [19, 31, 30]. Second, our architecture adopts actions from multiple agents instead of single agent. For the best of our knowledge, the two things haven't been conducted in the past.

In terms of architecture of our proposal, we incorporate the same holistic architecture as [19, 30], meaning employing an image encoder and decoder, and a dynamics engine. Here, we call the dynamics engine as Trans Learner (The word Trans comes from both Transformer and Transition). Fig. 4.1 shows the overview of our proposed architecture. In order to save GPU memory during training so that we can utilize larger batch size to improve the convergence speed, we set two training phases as depicted in Fig. 4.2. The first training phase is to train the encoder and the decoder models. After that, we proceed to the second training phase, which is to train Trans Learner. Separating the first training from second training would save huge amount of memory because we don't need to load the encoder, the decoder and Trans Learner all at once and latent codes, which are compressed versions of raw images, are handled during the second training phase.

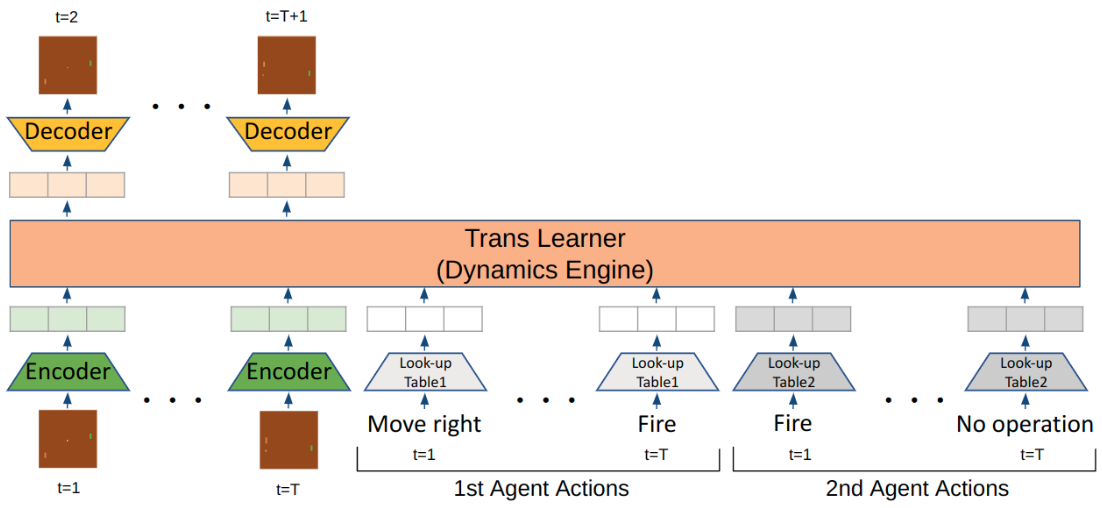

Fig. 4.1. Overall architecture of our model. The architecture accepts image information along with action information from multiple agents in order to predict one-time-step-ahead future images.

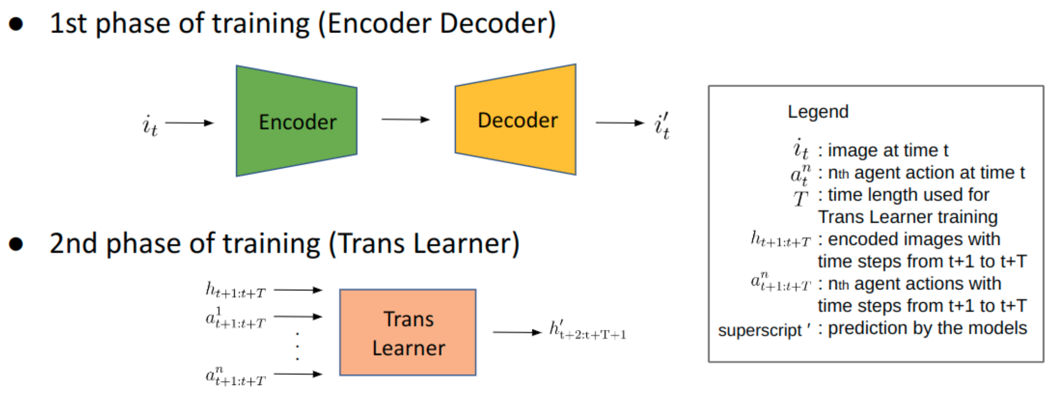
• 1st phase of training (Encoder Decoder)
i_t
Encoder
Decoder
i'_t
• 2nd phase of training (Trans Learner)
$h_{t+1:t+T}$
$a^1_{t+1:t+T}$
$a^n_{t+1:t+T}$
Trans Learner
$h'_{t+2:t+T+1}$
Legend
i_t : image at time t
a^n_t : nth agent action at time t
T : time length used for Trans Learner training
$h_{t+1:t+T}$: encoded images with time steps from t+1 to t+T
$a^n_{t+1:t+T}$: nth agent actions with time steps from t+1 to t+T
superscript ′ : prediction by the models

Fig. 4.2. There are two training phases. The first training phase is to train the encoder and the decoder. The second training phase is to train Trans Learner.

4.2 Transformer-based Dynamics Handling

The first proposal is to design a temporally stronger model. Generated environment results from previous works seemed plausible on their papers at first glance, but after we carefully investigated the temporal dynamics of their models, we found that their models may not have strong capability of handling temporal information. For example, as we have discussed in Chapter 1, objects, such as a car, suddenly changes its appearance to another objects, such as a truck. In another case, the number of driving lanes where an agent is driving on suddenly changes, e.g., from two lanes to single lane. And you can easily imagine how dangerous it would be if those kinds of things happened in real life. Thus, we need to have temporally consistent environments in order for agents to properly learn to behave. As a way to achieve that, one architecture that is most frequently wielded by AI researchers came to our mind, which is Transformer. Looking at the recent trends, Transformer-based models have been outperforming LSTM-based ones in many kinds of fields, such as language translation, speech recognition, vision-and-language navigation, etc. [73, 50, 12, 1, 48]. Because of the facts, we start having an intuition that it might also work well in this field.

When it comes to the training objective of Trans Learner, we adopt the same loss function used to train the dynamics engine of [30], which is denoted as follows.

$$L_{\mathrm{TL}} = L_{\mathrm{adv}} + \lambda_{\mathrm{feat}} L_{\mathrm{feat}} + \lambda_{\mathrm{recon}} L_{\mathrm{recon}} + \lambda_{\mathrm{action}} L_{\mathrm{action}} + \lambda_{\mathrm{GP}} L_{\mathrm{GP}}, \tag{4.1}$$

where L_{TL} is the combined loss to train Trans Learner, L_{adv} is the adversarial loss, L_{feat} is the reconstruction loss of features that are obtained in the middle of a discriminator layers, L_{recon} is the reconstruction of latent codes of images, L_{action} is the reconstruction of action inputs, and L_{GP} is the gradient penalty (R1 regularization [40] is used) and each λ_* denotes a multiplier to its corresponding loss. Here, we make one modification to the loss function for the purpose of improving the temporal consistency. In order to get the output of latent codes and the features that are used to calculate the reconstruction losses explained above, we input ground truth latent codes for Trans Learner all the time during training, which is called Teacher-forcing, as opposed to the previous works [31, 30], both of which autoregressively use generated images or latent codes as inputs even during training while gradually reducing the number of steps to use ground truth inputs (in their papers they call the technique as warm-up phase). Because of our training mechanism, one assumption comes up. The more inputs Trans Learner is conditioned on, the more accurate the predictions of the model should be. Hence, to embody the assumption into the loss, we impose linearly weighted reconstruction loss for each time step between 1 to a maximum time step N, which is the limit of tokens Trans Learner can accept and is discussed in 5.5.1. The following equation depicts the modification.

$$S = \sum_{i=1}^{N} \frac{2i}{N+1} (\mathbf{h}_i - \hat{\mathbf{h}}_i)^2, \tag{4.2}$$

where S is the sum of reconstruction loss of either latent codes of images or the features, $\mathbf{h}_i$ is a ground truth at time i, and $\hat{\mathbf{h}}_i$ is a prediction by the model at time i. The multiplier $\frac{2}{N+1}$ comes from the fact that the expectation of the total sum of the modified reconstruction loss approximately becomes equal to the original one with the multiplier, assuming each squared error $(\mathbf{h}_i - \hat{\mathbf{h}}_i)^2$ has the same value on average. If the expectation is approximately the same, the relation of the reconstruction loss to the other losses is still the same, in which case we can still use previously well-proven hyperparameters without conducting hyperparameter search.

We also incorporate other techniques proposed by [31, 30], such as single latent discriminator and temporal action-conditioned discriminator. For more details please refer to A.2.1 and [31, 30].

4.3 Incorporation of Multi-agent Actions

The Second proposal is to create simulated environments for multiple agents. Current works have successfully created environments for single agent so far. However, how the current state-of-the-art models scale up to multi-agent cases is unknown. In fact, in reality, there are many situations where multiple agents need to cooperate, compete or mix of the two. One agent working together with other agents to help someone in trouble is an example of cooperation, an agent playing tennis against another agent is an example of competition, and a company employee trying to innovate something new with coworkers to pull ahead of other companies' employees is an example of the mix of cooperation and competition (in this case, competition occurs against other companies' employees). Therefore, there also should be generative models that can create simulators for multi-agent systems. Inspired by prominent results of multi-modal Transformers form previous works [65, 8, 48, 80], we took the hint of applying the architecture to this field. Our task is multi-modal in the sense that the temporal information handling module, Trans Learner, accepts latent codes for vision and action inputs from multiple agents in order for it to predict convincing images based on the given inputs. Fig. 4.3 shows how those inputs are handled.

4.4 Attention Masks

We use two types of attention masks. The picture of them are depicted in Fig. 4.4. In the figure, it illustrates the case of having two agents. Type I is used for both single-agent and multi-agent datasets and type II is used only for multi-agent datasets. Notations are explained as follows in the figure. Vs stand for vision embeddings, As stand for action embeddings, embeddings with a subscript q indicate query embeddings, and ones with a subscript k are key embeddings, and the superscripts 1 and 2 are used to denote 1st and 2nd agent, respectively. The blue-colored blocks indicate where a vision query attends to, green-colored ones and yellow-colored ones are where actions of 1st and 2nd

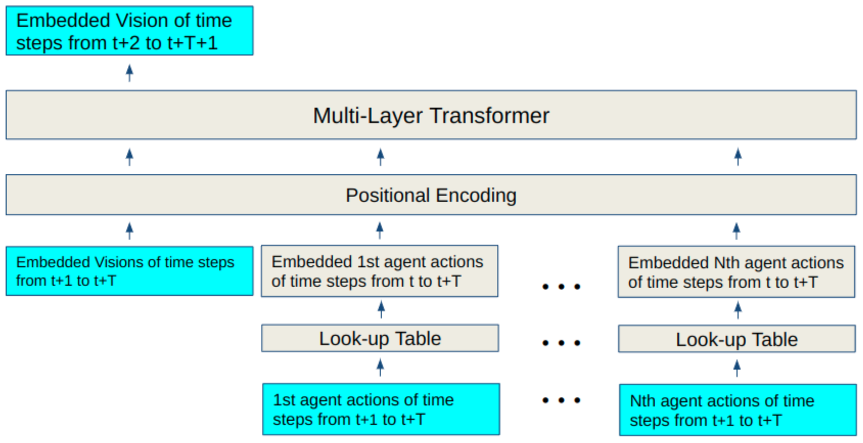
Embedded Vision of time steps from t+2 to t+T+1
Multi-Layer Transformer
Positional Encoding
Embedded Visions of time steps from t+1 to t+T
Embedded 1st agent actions of time steps from t to t+T
Embedded Nth agent actions of time steps from t to t+T
Look-up Table
Look-up Table
1st agent actions of time steps from t+1 to t+T
Nth agent actions of time steps from t+1 to t+T

Fig. 4.3. Trans Learner architecture that handles temporal information. This module accepts latent codes of images as well as multiple agent actions for specific time span.

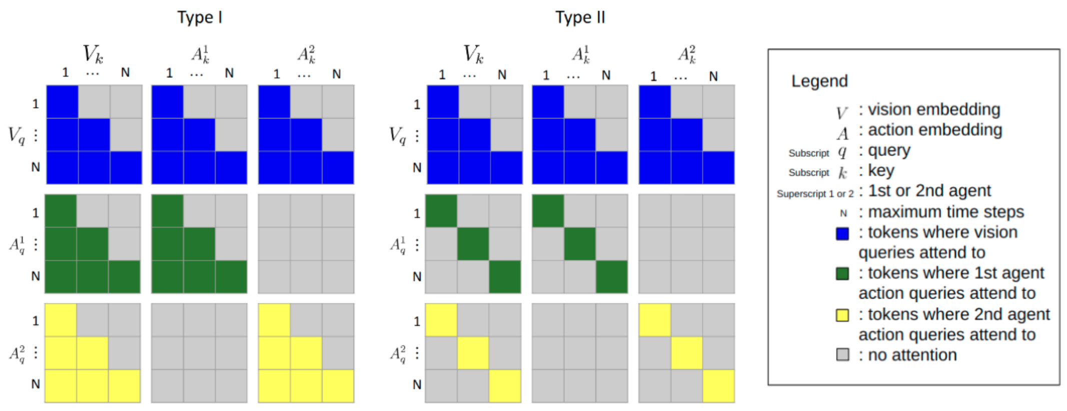

Fig. 4.4. We apply two types of attention masks. Type I is used for both single-agent and multi-agent datasets. Type II is used only for multi-agent datasets.

agent attend to, respectively, grey-colored ones indicate masked parts, and N denotes a predefined maximum time step that Trans Learner can handle, which is discussed in 5.5.1. We started experimenting only using type II for both single and multiple agent datasets at first, but it turned out that we obtained better results when Type I is used for single-agent dataset. We suspect the reason why is that the single-agent datasets we have experimented come from driving environments and due to the fact that the environments of driving scenes also depend on acceleration caused by the sequence of input actions, the action queries have to attend the sequence of previous actions in order to extract convincing features of actions that would help Trans Learner generate decent future latent codes. In terms of multi-agent datasets, we could not see much difference between two masks. For evaluations discussed in 5.5.1 and 5.5.2, we only use type II for multi-agent datasets.

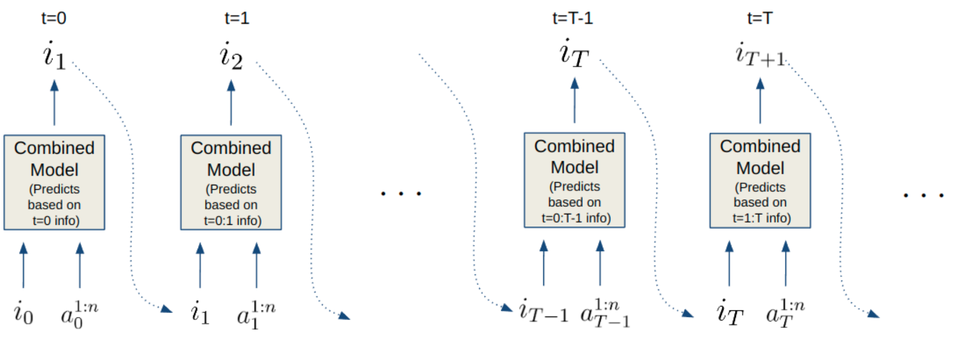
t=0
t=1
t=T-1
t=T
i_1
i_2
i_T
i_{T+1}
Combined Model (Predicts based on t=0 info)
Combined Model (Predicts based on t=0:1 info)
Combined Model (Predicts based on t=0:T-1 info)
Combined Model (Predicts based on t=1:T info)
i_0 $a_0^{1:n}$
i_1 $a_1^{1:n}$
i_{T-1} $a_{T-1}^{1:n}$
i_T $a_T^{1:n}$
. . .

Fig. 4.5. This is how our proposed method performs during inference. At an initial step, a ground truth image needs to be provided in order to predict a future image. After the initial step, predicted images are used as inputs. It continues up until predefined maximum time steps. The model slides input windows as a way to predict a future image after reaching to the maximum time steps. A notation (beginning):(ending) in the figure means our model considers information of time steps between (beginning) and (ending).

4.5 Inference

During inference, all the trained modules, which are the encoder, the decoder and Trans Learner, are combined as one and the combined model consumes an image and single-agent or multiple-agent action(s) all at once at each time step in order to predict a one-time-step-ahead image. Here, there is one constraint during inference time. At an initial step, we need to feed a ground truth image, but after the first step, generated images are used as input images, which means the combined model predicts next images in an autoregressive manner afterwards. It goes on until the number of steps reaches to predefined maximum time steps that Trans Learner can handle. After reaching to the maximum time steps, Trans Learner discards the oldest time step information and accepts the newest one to predict a one-time-step-ahead image. See Fig. 4.5 how our proposed method behaves during inference.

Chapter 5

Experiments

5.1 Tasks

In this experiment, we evaluate our architecture in two approaches. The two approaches are to test our model and previous works [19, 31, 30] using single-agent and multiple-agent datasets. The first approach is done by scaling down the multiple-agent inputs of our model to single agent ones, which can be easily carried out considering our Trans Learner architecture. As for the second approach, since all the previous works are not capable of handling actions from multiple agents, we make some modifications to their works so that they can also wield multi-agent actions. More details about it are discussed in 5.3.

5.2 Datasets

As explained in 5.1, we employ two types of datasets, which are single-agent and multi-agent datasets. The number of pixels of each dataset is resized to 64×64 (height × width) except for Grand Theft Auto V (GTAV) [20] which has 48×80 image size. Here, we briefly explain each dataset.

5.2.1 Single-agent Datasets

As for single-agent datasets, we use two datasets. GTAV and CARLA [14].

Grand Theft Auto V

GTAV dataset we use comes from driving scenes from the game. There is a black car, which is an agent of this environment, divining on a street. Besides the car running on the street, it interacts with non-agent cars and also there are walls on both sides of the road. Thus, there exist other objects which are supposed to affect the transitions of the agent. As for the action commands of the agent, they have discrete values, such as 0: move left, 1: no operation, and 2: move right. The default movement of the agent is to go forward, which means the car keeps moving forward if the agent is not instructed any command.

CARLA

CARLA is also an driving simulator, but this environment has much complicated scenarios. First of all, there exist so many types of cars for the agent. For example, the agent can be a passenger car, a truck, or even a motorcycle. With respect to the surroundings, there are way more non-agent cars that run as the same direction as the agent or coming toward the agent in the opposite lane. With regard to the action space of CARLA, it is continuous, meaning there are two parameters, speed and angular velocity. As for the speed, the higher value is, the faster the agent is moving forward. The lower value is, the slower the agent is moving forward. As for the angular velocity, the higher value is, the more the agent is moving to the right. The lower value is, the more the agent is moving to the left. This dataset is created by [30] and originally has higher number of pixels (256×256), but due to the fact that the encoder and the decoder we use in this experiment are not well capable of generating the high resolution, we resized it to 64×64.

5.2.2 Multi-agent Datasets

We use a library called PettingZoo [69] as ground truth environments. PettingZoo includes several environments, but we only use Atari environments for our research. Among them, we use environments called Pong and Boxing. Action commands for both of the environments are discrete.

Pong

Multiple agents both play against each other and cooperate with each other in Pong environment, depending on how many agents are playing. The environment allows you to choose between two and four agents. If the number of agents is two, the agents compete with each other by hitting a ball to the opponent side. If it is four, two of the agents play as a team and compete with the other team, which also has two agents as a team. We conduct our research for both cases. There are originally six actions available, but for our research, we restrict the number of actions to four due to the fact that the two actions we have decided not to include are not frequently used. Thus, available actions are as follows. 0: no operation, 1: fire, 2: move right, and 3: move left.

Boxing

Multiple agents only play against each other and the number of agents is fixed as two in Boxing environment. Each agent throws punches to hit the opponent. There are originally 18 actions available, but once again, we restrict the total number of actions to six for our research. Available actions are as follows. 0: no operation, 1: fire, 2: move up, 3: move right, 4: move left, and 5: move down.

5.3 Implementation

As for our image encoder, decoder, and discriminator models, we incorporate the same architectures as GameGAN that have Convolutional-network-based and ResNet-based architectures [21] because those two modules are not our focus, but our dynamics engine is. Thus, we can almost directly compare how well-equipped our proposed architecture of dynamics engine is to handle temporal information and multi-agent actions without being affected by generated image quality compared to previous works. The implementation details are described in A.1.1. DriveGAN has a better architecture when it comes to the encoder and the decoder, but since we are only using small number of pixels, which is 64×64 (DriveGAN is not tested using GTAV data because the image encoder and decoder can only handle square images), they presumably does not affect the end results much.

When it comes to the dynamics handling, we utilize Transformer-based architecture. As to the architecture details, we use Transformer layers and we set the number of layers as 24, embedding dimension as 512, the number of heads used for multi-head attention as 8, feed-forward network dimension as 512, the activation function as ReLU, and the ratio of dropout which is applied after the softmax functions of attention calculations and activation functions used in the feed-forward network as 0.1. In regard to positional encoding, we apply the sinusoidal functions originally proposed by [73].

As mentioned in 5.1, since previous works are not capable of handling multi-agent actions, we make a modification to their works. As for World Model, since the input dimension of MDN-RNN is hidden dimension plus action dimension, we just multiply the action dimension by the number of agents. When it comes to GameGAN and DriveGAN, before the action embedding is concatenated with hidden dimension, an action input is transformed by a linear layer and leaky ReLU activation [39]. Thus, instead of just using the linear layer for single agent action, we separately created a linear layer for each agent action in order to get each action embedding, which is concatenated all together after leaky ReLU activation is applied.

Tab. 5.1. shows the number of parameters for each model. The parameters listed in the table are obtained from 64×64 image size version for each model. DriveGAN has much higher number of parameters compared to the other models. More than half of the total parameters come from their encoder and decoder. Our dynamics engine has slightly higher number of parameters compared to the DriveGAN counterpart.

Table 5.1. The number of parameters for each model. Enc-Dec denotes an encoder and a decoder and DE stands for a dynamics engine. DriveGAN has the largest number of parameters although ours has slightly higher number of parameters as to dynamics engine.

Number of Parameters (in millions)				
Model	World Model	GameGAN	DriveGAN	Ours
Enc-Dec	4.4M	21.0M	50.1M	21.0M
DE	1.7M	6.8M	33.5M	37.9M
Sum	6.1M	27.9M	83.7M	58.9M

We use single NVIDIA RTX A6000 GPU to train each model. With the GPU having 48GB of GPU memory, we utilize the memory as much as we can to train each model except [19], which does not require large memory size to train.

5.4 Metrics

First, we operate Fréchet Video Distance (FVD) [70] which can not only measure quality and diversity of videos, but also temporal coherence by comparing the distribution of the generated videos with the distribution from ground truth ones. Fréchet Inception Distance (FID) [22] is already a widely used metric in the matter of image generation task. FVD applies the concept of FID into video generation field. To be more specific, they utilize a 3D convolution model named Inflated 3D Convnet (I3D) [6], which, as the name suggests, inflates 2D feature extraction into 3D counterpart. Therefore, it also takes temporal information into account for downstream tasks. Instead of ImageNet [10] dataset that is used for an image classification task and on which the original Inception architecture is trained, I3D pretrains with a dataset called Kinetics-400 [6], which is associated with an action recognition task. According to the experiments of [70], the authors mentioned that obtained scores from FVD most closely agree with human evaluations compared to already well-known metrics, such as Peak Signal to Noise Ratio (PSNR) or the Structural Similarity (SSIM) [74]. Thus, the metric is suitable to measure the difference between previous works and our proposed method in terms of temporal consistency.

In addition, to evaluate the quality of generated simulators, we need a metric to see how much generated images precisely follow the conditioned actions. As for a good simulator,

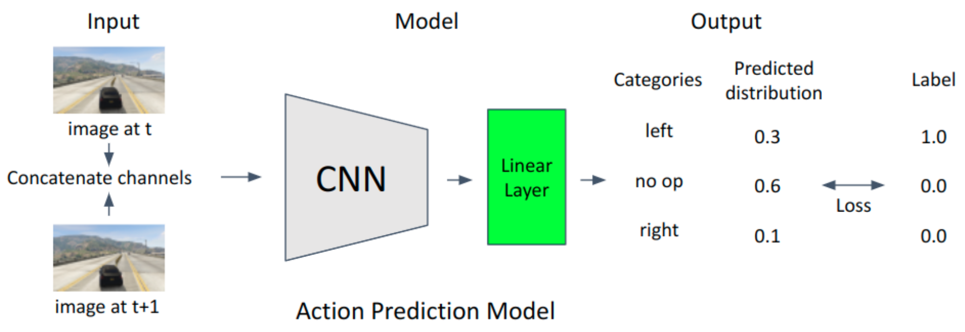

Fig. 5.1. Overview of action prediction model. Two consecutive images are concatenated channel-wise and passed to the model so that the model can predict the most appropriate action between the images. After training, loss value calculated by the model is used as a metric.

it has to generate a state which faithfully reflects (a) given action(s). To measure the action-consistency quantitatively, an action(-consistency) prediction model [30], which accepts two sequential images and predicts a proper action taken between the images, is created. Due to the fact that the authors of [30] did not provide any architectural details, we create it on our own using ResNet based architecture. The overall architecture of the model is shown in Fig. 5.1 and the details of the architecture are depicted in A.3. Note that we also need to calculate the loss for ground truth images because some actions may not be properly predicted by just looking at two consecutive images. For instance, if a car is standing at the edge of a road where there is a wall on its right, moving toward the right doesn't affect anything on the generated image because a car is not supposed to go through the wall. Thus, there always should be some error even if the given images come from ground truth. Hence, we use the loss of ground truth videos as the lower bound of the loss. We utilize this model for only single-agent datasets. The reason that we only conduct this evaluation to single-agent datasets is that we could not appropriately train the model using the multiple-agent datasets at all. We found that, after carefully investigating the ground truth environments of PettingZoo step by step, the environments do not immediately reflect given actions (they take several time steps to reflect given actions), hence, the model presumably could not predict proper actions from two consecutive images.

5.5 Results

There is one thing that needs to be mentioned for evaluation sections. Since all of the models require a ground truth image at an initial step and a sequence of action inputs, we provide identical initial images to all the models along with the same sequence of actions and see how each model transitions.

5.5.1 Quantitative Evaluations

We evaluate our proposed model along with [19, 31, 30] using FVD and the action prediction model discussed in 5.4. As one of our objectives is to measure temporal consistency, we specify the same maximum time steps during training in order to train either previous works' LSTMs or our Transformer so that all the models are trained on the same condition in terms of sequential length of inputs. We set the predefined steps for each dataset as follows. GTAV: 128, CARLA: 64, Pong: 128, and Boxing: 64.

Tab. 5.2. and Tab. 5.3. display the quantitative results of previous works and ours. In regard to FVD, ours shows superior results compared to previous works except for CARLA, in which ours is the second best. Especially, the results show that ours outperforms previous works when it comes to multi-agent datasets. This indicates that our transformer-based architecture handles multi-agent inputs far better than LSTM-based architectures from previous works, which includes a carefully thought-out architecture utilized in DriveGAN, where it applies not only LSTM, but also convolutional LSTM (ConvLSTM) [62] to handle a spatiotemporal sequence and action-dependent and action-independent features.

With respect to the action prediction loss metric, ours could not obtain the best results in either of the single-agent datasets. As mentioned in 5.2.1, inputs of GTAV are discrete numbers and ones from CARLA are continuous numbers and based on the data types, we have slightly different results. Ours shows a better result compared to GameGAN when the inputs are continuous, which describes the situation more accurately and the fact might have helped ours produces more accurate images, although GameGAN shows a far better score than ours when the inputs are discrete.

Table 5.2. Comparison of our model with previous works [19, 31, 30] using FVD. The lower the score is, the better. DriveGAN is not trained on GTAV dataset because the encoder and the decoder of DriveGAN is only capable of generating square images.

	FVD ↓			
	Single-agent		**Multi-agent**	
Models	GTAV	CARLA	Pong	Boxing
World Models	1923.00	1716.42	355.81	5181.42
GameGAN	580.52	613.16	613.18	1947.52
DriveGAN	-	**215.54**	14.87	638.40
Ours	**210.37**	333.93	**10.04**	**17.48**

Table 5.3. Comparison of our model with previous works [19, 31, 30] using action prediction loss. The lower the score is, the better. DriveGAN is not trained on GTAV dataset because the encoder and the decoder of DriveGAN is only capable of generating square images.

	Action Prediction Loss ↓	
Models	GTAV	CARLA
World Models	0.6060	16.6893
GameGAN	**0.0050**	4.2635
DriveGAN	-	**3.1303**
Ours	0.3450	3.6601
Ground Truth	0.0045	0.3735

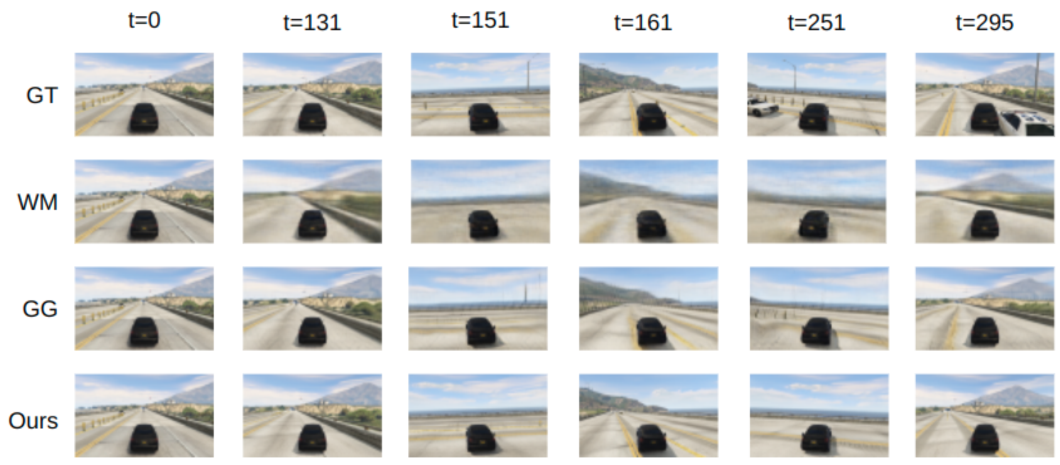

Fig. 5.2. Generated environment example using GTAV dataset. Although ours shows the best image quality of the environment, which may have led our model to have the best score in FVD, GameGAN outperforms ours in terms of reflecting action inputs.

5.5.2 Qualitative Evaluations

In this section, we evaluate results by looking at generated environments and we also analyze the reason why our model achieved better results compared to previous works in some aspects and why ours underperformed in others. In each figure for this section, used labels are as follows. GT: ground truth, WM: World Model, GG: GameGAN, and DG: DriveGAN.

Single-agent Datasets

■GTAV Fig. 5.2 displays the generated results from each model except DriveGAN (DriveGAN is not created to process non-square images). For this dataset, all the environments created by ours and previous works, except World Model, have temporally consist results, so we conduct analysis focused exclusively on image quality and how each model handles action inputs because those two things affect FVD and the action prediction metric. Compared to the other models, although ours generate the best image quality having clearly-displayed yellow lines on the ground and background, which is also shown in our FVD score, ours does not reflect action inputs as well as GameGAN according to the scores obtained from the action prediction metric. Qualitative evaluation also agrees with the scores. For example, the ground truth image at time step 161 displays that the agent car is almost parallel to the yellow line and the result from GameGAN also generates the same angle to the yellow line as opposed to ours, which is slightly off the line. At 295, which is the last time step, both the ground truth and GameGAN images have the yellow line on the left side of the agent, but ours has the agent on the line, indicating ours doesn't handle given actions to the same degree as the ground truth.

■CARLA Fig. 5.3 and Fig. 5.4 show two episodes of generated examples from each model trained on CARLA dataset. Looking at the results, we can say that our proposed model has good temporal consistency on par with the current best model, DriveGAN. For example, there is a red car at the beginning of Fig. 5.3 and only ours and DriveGAN consistently keep transitioning in a natural way (the car suddenly disappears in several

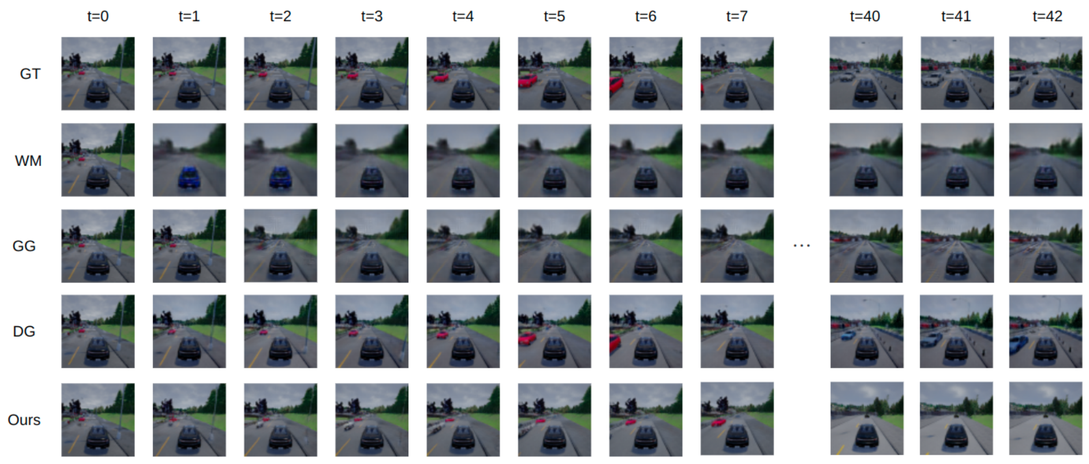

Fig. 5.3. Generated environment example 1 using CARLA dataset. Only ours and DriveGAN show strong temporal consistency. Since DriveGAN even includes carefully designed stochastic modules, it shows high diversity of generated images even at later time steps.

frames after the initiation for World Model and GameGAN). The same goes for Fig. 5.4. There is a red car in front of an agent, and only ours and DriveGAN consistently maintain it although it eventually disappears at around 11th time step (once again, the car suddenly disappears in several frames for World Model and GameGAN). Even for DriveGAN, there is an unnatural transition for the agent, a motorcycle, between 25th and 27th time steps gradually becoming a car while it stays the same for the other models.

Although the generated results show that temporal consistency of ours is at least as good as one from DriveGAN, DriveGAN outperforms ours in FVD. We suspect the reason behind it is that DriveGAN employs well-suited stochasticity backed up by their intricate architecture utilizing reparameterization steps [32] not only for the encoder, but also three other parts in the dynamics engine. Because of that, the generated results of DriveGAN show high diversity. Since FVD takes into account not only temporal consistency, but also diversity, the score of DriveGAN is presumably better than the score of ours. Fig. 5.3 evidently shows that even the time steps around 40th, the generated images of DriveGAN include some random non-agent cars while there is no non-agent car appeared on the other images from the other models around that time step. Same holds true for Fig. 5.4. At around 25th time step, there is no non-agent car except the generated results from DriveGAN despite the shape and color of the non-agent car being gradually changing.

Multi-agent Datasets

■Pong Fig. 5.5 shows the results based on Pong dataset. Since Pong environment slowly transitions for each time step, we list sequence of images having ten-time-step intervals until it gets to 70th time step. As can be seen from the results, World Model and GameGAN do not handle the transitions well at all. The white ball disappears within 30 time steps for World Model and 10 time steps for GameGAN. Furthermore, both of the agents' locations largely differ from the ones in ground truth for World Model, indicating World Model does not properly process given actions considering the same sequence of actions is given to all the models including the ground truth environment. Ours and DriveGAN, on the other hand, produce convincing sequence of images and both of them seem to accurately handle given actions. However, after several hundred time steps, the results from DriveGAN started to show massive degradation. The ball appeared at 231st time step

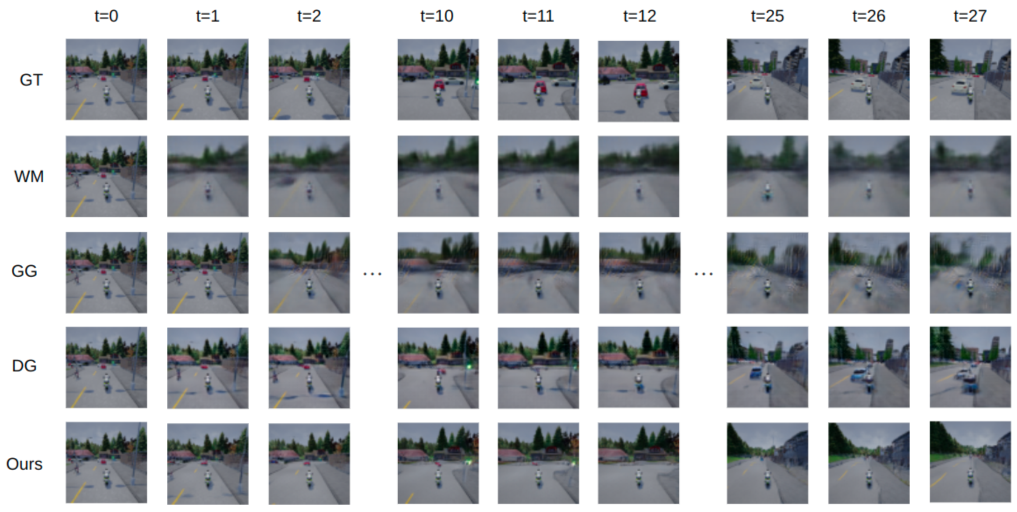

Fig. 5.4. Generated environment example 2 using CARLA dataset. Ours shows the best result in terms of temporal consistency, but DriveGAN demonstrates more diversified results compared to the others thanks to their crafted architecture.

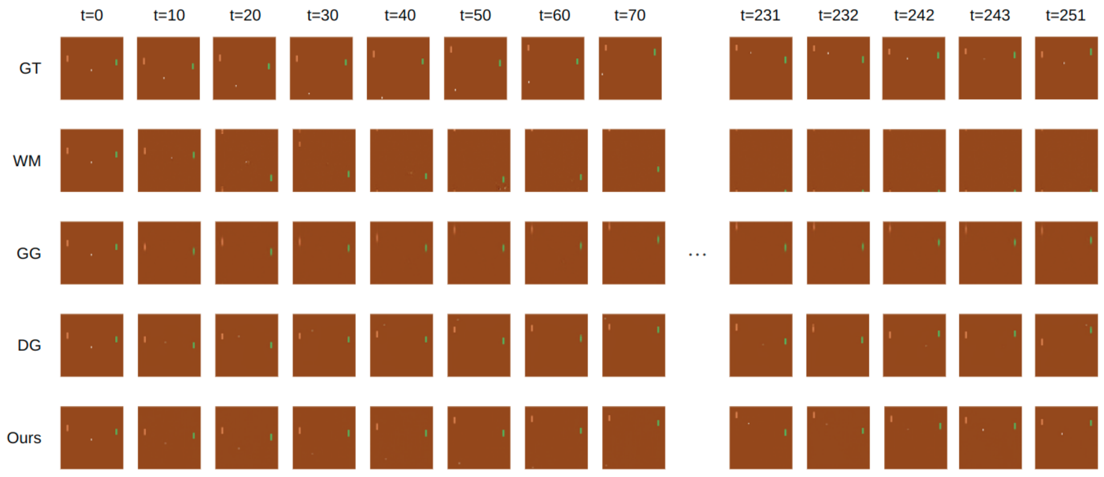

Fig. 5.5. Generated environment example using Pong dataset. Only ours accurately transitions between frames for long span of time. DriveGAN initially seems to produce good transitions, but it does not keep the quality for long time steps.

suddenly disappears at 232. Then, all of a sudden, it re-emerges at 242 and once again disappears at 243. After that, it reappears in a weird location at 251. On the contrary, ours closely replicate the ground truth environment, meaning ours processes given actions almost as close as the ground truth environment does and the produced trajectory of the ball exceptionally seems natural compared to the other results.

Regarding FVD scores, although generated results gradually degrade for DriveGAN, it obtained good score, 14.87, which is almost as good as ours, 10.4. We reckon it likely comes from the fact that the ball itself is a really small part of the entire image even though the trajectory of it is obviously unnatural to humans. Therefore, it seemingly did not affect much to the FVD score.

In addition, because PettingZoo Atari environments accommodate up to four agents, we have also tried emulating Pong environment with four agents. Fig. 5.6 displays three

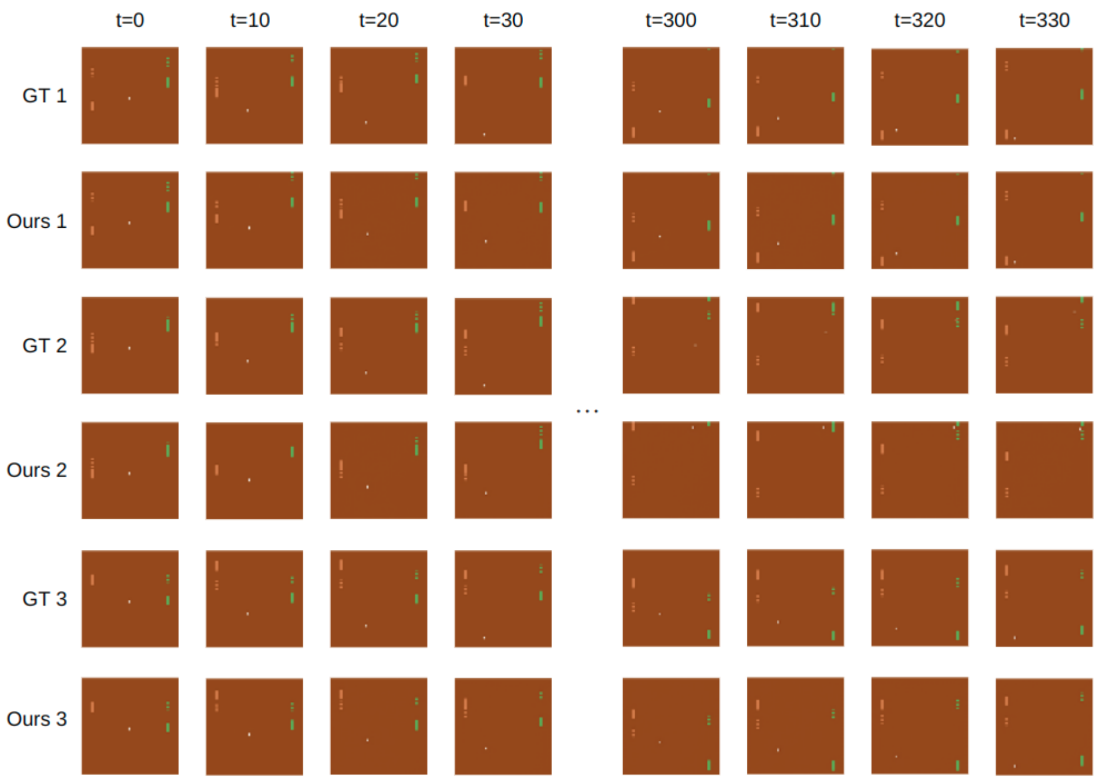

Fig. 5.6. Three generated examples from ours emulating Pong environment with four agents are compared with the same situations from ground truth environment. The results show that ours can successfully scale up the number of agents faithfully reflecting all the given four-agent actions.

episodes of our generated examples directly compared with the exact same situations of ground truth environment. The figure shows how our model can successfully scale up to more number of agents. As for FVD, ours with four agents obtained 10.00 which is almost the same score as ours with two agents, 10.04. Thus, it quantitatively shows that our proposed model is not affected by the number of agents with this environment in terms of generated image quality and temporal dynamics. Furthermore, looking at the position of each agent, ours generates each agent at almost exactly the same location as the one in ground truth, meaning ours even faithfully reflects all the four-agent action inputs to generate images.

■**Boxing** Fig. 5.7 shows an episode of generated example from each model trained on Boxing dataset. Once again, due to the environment transitions being certainly slow, we list sequence of images having five-time-step intervals for this example. Looking at the results, it is obvious that ours generates the most convincing transitions between frames. Transitions of ours have almost the same transitions of the ground truth. And ours is the only one that faithfully operates given actions. GameGAN seems to be operating the actions at the beginning, but results after 55th time step show that it gradually ceases to do so. As for World Model, two agents even disappear between 65th and 70th time steps.

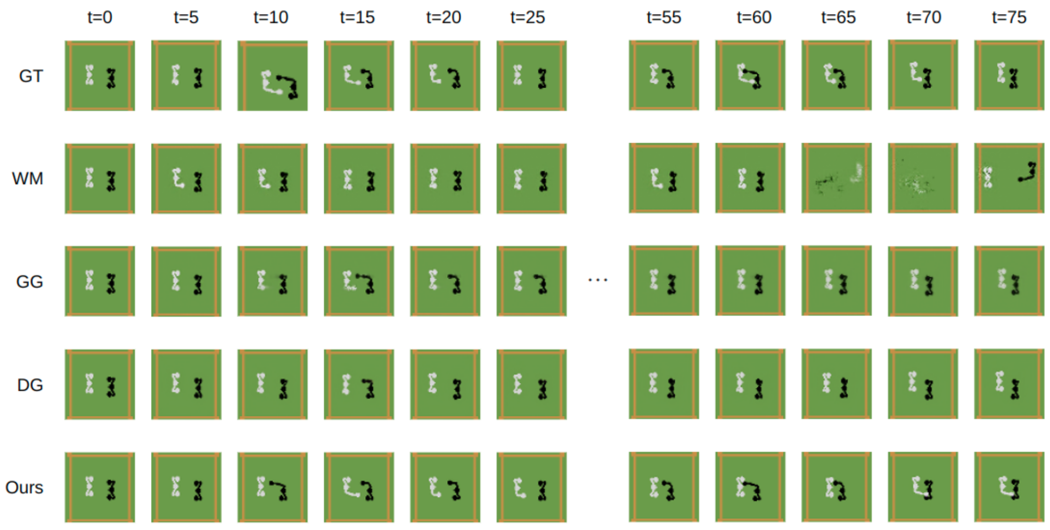

Fig. 5.7. Generated environment example using Boxing dataset. Only ours properly handles given actions.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have proposed a Transformer-based architecture to handle temporal information and multi-agent inputs with a view to create realistic environments in the future. After careful investigation, we find out that previous works have weakness in temporal consistency and they can only handle actions from single agent. Aiming to generate more temporally reliable environments that are smoothly adaptable to multiple agents, we have decided to incorporate Transformer architecture into our proposal, which empirically demonstrates that it successfully handles temporal information and also multimodal inputs. Our results suggest that, although we could not provide strong evidence that the temporal consistency of our proposed model is superior to all the previous works, our Transformer-based model generates temporally consistent transitions at least as well as the current best model. In terms of handling multi-agent inputs, ours quantitatively and qualitatively outperforms previous works by a large margin. Moreover, by scaling up the number of action inputs, our proposed model has also successfully emulated one of the environments with four agents.

In order for a generative model to create realistic environments, we still have so many obstacles to overcome, but we hope our research will motivate other researchers make a future contribution in this field.

6.2 Future Work

As discussed in Chapter 2, the environments can be either deterministic or stochastic. When it comes to our proposed architecture, it is a deterministic model because the output of our model produces the same values all the time if the temporal condition and all the given inputs are the same. We aimed to extend our model into stochastic one, but due to the time constraints of master research, we could not finish it on time. Therefore, that is one of our future plans to proceed from this research. In fact, [11] has an interesting approach when it comes to incorporating sequential stochasticity in a model, hence, we may be motivated to design a new type of architecture inspired by their work in the future.

We also could not extensively search for the hyperparameters and conduct tuning of our Transformer-based model due to the time restriction, which includes obvious tuning such as increasing the embedding dimension, the number of heads used for multi-head-self-attention, incorporating warm-up and decay of learning rate, and so on. Thus, we will work on them for our future research.

We are also interested in extending our research into type 2 environments as we briefly discuss it in 2.1.2. To achieve that, not only does a dynamics engine have to keep local information around agent(s), but also it has to consistently hold global information. [7]

proposes some interesting ideas. Their proposed architecture generates a 2D scene layout grid that is globally persistent and can be extended during inference time. Their work also applies volume rendering to the layout to utilize 3D information in order to generate each frame of an environment. We suspect some of the methods introduced on their research may be applicable to create type 2 environments.

In addition, considering the rapid development of foundation models [53, 43, 64, 27, 67], Transformer-based architecture is likely capable of generating multimodal outputs for environments such as sound and language along with an image at each time step, which would make environments far more realistic. Since Transformer architecture can be easily scaled up, it would be no surprise that one may create the type of environments utilizing huge amount of multimodal data.

Lastly, we also mentioned some existing issues in Chapter 1 if the goal is to create realistic environments. Thus, we also keep these things in our mind as potential future directions.

References

[1] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[2] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

[3] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[4] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[5] Hanqun Cao, Cheng Tan, Zhangyang Gao, Yilun Xu, Guangyong Chen, Pheng-Ann Heng, and Stan Z Li. A survey on generative diffusion model. *arXiv preprint arXiv:2209.02646*, 2022.

[6] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[7] Lucy Chai, Richard Tucker, Zhengqi Li, Phillip Isola, and Noah Snavely. Persistent nature: A generative model of unbounded 3d worlds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20863–20874, 2023.

[8] Shizhe Chen, Pierre-Louis Guhur, Cordelia Schmid, and Ivan Laptev. History aware multimodal transformer for vision-and-language navigation. *Advances in neural information processing systems*, 34:5834–5847, 2021.

[9] Alexis CONNEAU and Guillaume Lample. Cross-lingual language model pretraining. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[11] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1174–1183. PMLR, 10–15 Jul 2018.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Lin-

guistics.

[13] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[14] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.

[15] Yuan Gong, Yu-An Chung, and James Glass. AST: Audio Spectrogram Transformer. In *Proc. Interspeech 2021*, pages 571–575, 2021.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[17] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[18] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[19] 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`.

[20] Sentdex Harrison, Daniel Kukieła, and Nidhal Baccouri. GANTheftAuto. `https://github.com/Sentdex/GANTheftAuto`, 2021.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[23] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

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

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[26] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[27] Qihao Liu Zehuan Yuan Xiang Bai Song Bai Junfeng Wu, Yi Jiang. General object foundation model for images and videos at scale, 2023.

[28] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[29] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, 2020.

[30] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. Drivegan: To-

wards a controllable high-quality neural simulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5820–5829, 2021.

[31] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2020.

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

[33] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.

[34] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543, 2008.

[35] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1558–1566, New York, New York, USA, 20–22 Jun 2016. PMLR.

[36] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

[37] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12009–12019, June 2022.

[38] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[39] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[40] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018.

[41] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[42] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[43] David Mizrahi, Roman Bachmann, Oğuzhan Fatih Kar, Teresa Yeo, Mingfei Gao, Afshin Dehghan, and Amir Zamir. 4M: Massively multimodal masked modeling. In *Advances in Neural Information Processing Systems*, 2023.

[44] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[45] 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.

[46] Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

[47] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[48] Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic Transformer for Vision-and-Language Navigation. In *ICCV*, 2021.

[49] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018.

[50] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[51] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[52] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[53] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[54] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[55] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[56] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

[57] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM SIGGRAPH 2022 conference proceedings*, pages 1–10, 2022.

[58] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[60] Wilko Schwarting, Tim Seyde, Igor Gilitschenski, Lucas Liebenwein, Ryan Sander, Sertac Karaman, and Daniela Rus. Deep latent competition: Learning to race using visual control policies in latent space. In *Conference on Robot Learning*, 2020.

[61] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[62] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

[63] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan

Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022.

[64] Samuel Stevens, Jiaman Wu, Matthew J Thompson, Elizabeth G Campolongo, Chan Hee Song, David Edward Carlyn, Li Dong, Wasila M Dahdul, Charles Stewart, Tanya Berger-Wolf, Wei-Lun Chao, and Yu Su. Bioclip: A vision foundation model for the tree of life. 2023.

[65] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7464–7473, 2019.

[66] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[67] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[68] Yee Whye Teh, Max Welling, Simon Osindero, and Geoffrey E Hinton. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4(Dec):1235–1260, 2003.

[69] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

[70] Thomas Unterthiner, Sjoerd Van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

[71] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

[72] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[74] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[75] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[76] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[77] Lijun Yu, Yong Cheng, Kihyuk Sohn, José Lezama, Han Zhang, Huiwen Chang, Alexander G Hauptmann, Ming-Hsuan Yang, Yuan Hao, Irfan Essa, et al. Magvit: Masked generative video transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10459–10469, 2023.

[78] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, vol-

ume 97 of *Proceedings of Machine Learning Research*, pages 7354–7363. PMLR, 09–15 Jun 2019.

[79] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[80] Luowei Zhou, Hamid Palangi, Lei Zhang, Houdong Hu, Jason Corso, and Jianfeng Gao. Unified vision-language pre-training for image captioning and vqa. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13041–13049, 2020.

Acknowledgements

First of all, I would like to appreciate Associate Professor Hideki Nakayama and lab members for giving me valuable feedback in order to finish my research. I've learned and experienced so much in pursuit of master's degree. I promise to do my best to make a valuable contribution to our society in the future utilizing the things I've learned at this lab and UTokyo.

I would also like to express my gratitude towards associate Professor Hideki Nakayama for letting me use a large amount of GPU resources and Mr. Kai Katsumata for facilitating smooth GPU sever usage. Without their help, I could not achieve the things I've done for my master research.

Last but not least, I am indebted to Ms. Rie Itabashi and Ms. Masae Konta for collecting feedback messages for my Rinko presentations and showing me around I-REF building.

I am grateful for being a part of the lab.

A Implementation Details

A.1 1st Training Phase

A.1.1 Encoder, Decoder, and Discriminator

Encoders, decoders, and discriminators used for our training phase 1 have the same architectures as [31, 20]. The datasets we use for our experiments have two image sizes, which are 64×64 and 48×80, and based on the sizes, we have two types of encoders, decoders, and discriminators. Tables from A.1 to A.6 show the architectures of encoders, decoders, and discriminators based on each image size, respectively. Followings are label explanations that describe each layer. Conv2d is a 2D convolutional operation, ReLU is a rectified linear unit activation, LeakyReLU is a leaky ReLU activation [39], Linear is a linear transformation, Reshape is a reshaping-a-given-tensor function, ResBlock is a residual block, SA is a self-attention module [78], and SNLinear and SNConv2d represent Linear and Conv2d with spectral normalization [42], respectively. As for ResBlock, different types of residual blocks are used for decoder and discriminator. For more details, please refer to [31]. As to dimension order of input and output shape, when a tensor is 3D, the order of dimension is (channel, height, width). When a tensor is 1D, it indicates channel.

Table A.1. Encoder architecture for 64×64 image size.

Encoder		
Image Size	64×64	
Layers	Input Shape	Output Shape
Conv2d	(3, 64, 64)	(64, 63, 63)
LeakyReLU	(64, 63, 63)	(64, 63, 63)
Conv2d	(64, 63, 63)	(64, 31, 31)
LeakyReLU	(64, 31, 31)	(64, 31, 31)
Conv2d	(64, 31, 31)	(64, 15, 15)
LeakyReLU	(64, 15, 15)	(64, 15, 15)
Conv2d	(64, 15, 15)	(64, 7, 7)
LeakyReLU	(64, 7, 7)	(64, 7, 7)
Reshape	(64, 7, 7)	(3136)
Linear	(3136)	(512)
LeakyReLU	(512)	(512)

Table A.2. Encoder architecture for 48×80 image size.

Encoder		
Image Size	48×80	
Layers	Input Shape	Output Shape
Conv2d	(3, 48, 80)	(64, 47, 79)
LeakyReLU	(64, 47, 79)	(64, 47, 79)
Conv2d	(64, 47, 79)	(64, 47, 79)
LeakyReLU	(64, 47, 79)	(64, 47, 79)
Conv2d	(64, 47, 79)	(128, 24, 40)
LeakyReLU	(128, 24, 40)	(128, 24, 40)
Conv2d	(128, 24, 40)	(128, 24, 40)
LeakyReLU	(128, 24, 40)	(128, 24, 40)
Conv2d	(128, 24, 40)	(256, 12, 20)
LeakyReLU	(256, 12, 20)	(256, 12, 20)
Conv2d	(256, 12, 20)	(64, 6, 10)
LeakyReLU	(64, 6, 10)	(64, 6, 10)
Reshape	(64, 6, 10)	(3840)
Linear	(3840)	(512)
LeakyReLU	(512)	(512)

Table A.3. Decoder architecture for 64×64 image size.

Decoder		
Image Size	64×64	
Layers	Input Shape	Output Shape
SNLinear	(512)	(32768)
Reshape	(32768)	(512, 8, 8)
ResBlock	(512, 8, 8)	(256, 16, 16)
ResBlock	(256, 16, 16)	(128, 32, 32)
ResBlock	(128, 32, 32)	(64, 64, 64)
SA	(64, 64, 64)	(64, 64, 64)
ReLU	(64, 64, 64)	(64, 64, 64)
SNConv2d	(64, 64, 64)	(3, 64, 64)

A.1.2 Loss Function

The loss function of our 1st training phase is also inherited from [31]. The following is the equation.

$$L_{\text{1st}} = L_{\text{adv}} + \lambda_{\text{percept}} L_{\text{percept}} + \lambda_{\text{GP}} L_{\text{GP}}, \tag{A.1}$$

where L_{1st} is the loss of 1st training phase, L_{adv} is an adversarial loss [16], L_{percept} is a perceptual loss [79], L_{GP} is R1 gradient penalty loss [40], and each λ with a subscript is a weight for each corresponding loss.

Table A.4. Decoder architecture for 48×80 image size.

Decoder		
Image Size	48×80	
Layers	Input Shape	Output Shape
SNLinear	(512)	(46080)
Reshape	(46080)	(768, 6, 10)
ResBlock	(768, 6, 10)	(384, 12, 20)
ResBlock	(384, 12, 20)	(192, 24, 40)
SA	(192, 24, 40)	(192, 24, 40)
ResBlock	(192, 24, 40)	(96, 48, 80)
SA	(96, 48, 80)	(96, 48, 80)
ResBlock	(96, 48, 80)	(96, 48, 80)
ResBlock	(96, 48, 80)	(96, 48, 80)
ReLU	(96, 48, 80)	(96, 48, 80)
SNConv2d	(96, 48, 80)	(3, 48, 80)

Table A.5. Discriminator architecture for 64×64 image size.

Discriminator		
Image Size	64×64	
Layers	Input Shape	Output Shape
Conv2d	(3, 64, 64)	(128, 64, 64)
LeakyReLU	(128, 64, 64)	(128, 64, 64)
ResBlock	(128, 64, 64)	(256, 32, 32)
ResBlock	(256, 32, 32)	(512, 16, 16)
ResBlock	(512, 16, 16)	(512, 8, 8)
ResBlock	(512, 8, 8)	(156, 4, 4)
ResBlock	(156, 4, 4)	(512, 2, 2)
ResBlock	(512, 2, 2)	(512, 1, 1)
Reshape	(512, 1, 1)	(512)
Linear	(512)	(512)
LeakyReLU	(512)	(512)
Linear	(512)	(1)

A.2 2nd Training Phase

A.2.1 Discriminator

The discriminator used for our 2nd training phase has the same architecture as [30], except some minor modifications. Tab. A.7 shows a base architecture of the discriminator and the output of which is used for single latent discriminator and temporal action-conditioned discriminator. Tab. A.8 shows single latent discriminator architecture and Tab. A.9 and Tab. A.10 depicts temporal action-conditioned discriminator architectures based on each maximum time steps (MTS). BatchNorm1d listed in layers denotes 1D batch normalization layer [25]. There are four branches as for the temporal action-conditioned discriminator. An output from each branch is passed to another SNConv2d layer to get the final prediction output, which is judged whether the output comes from real or fake dataset. 2D tensor of input and output shape means (sequence length, channel). As to

Table A.6. Discriminator architecture for 48×80 image size.

Discriminator		
Image Size	48×80	
Layers	Input Shape	Output Shape
Conv2d	(3, 48, 80)	(128, 48, 80)
LeakyReLU	(128, 48, 80)	(128, 48, 80)
ResBlock	(128, 48, 80)	(256, 24, 40)
ResBlock	(256, 24, 40)	(512, 12, 20)
ResBlock	(512, 12, 20)	(512, 6, 10)
ResBlock	(512, 6, 10)	(156, 3, 5)
ResBlock	(156, 3, 5)	(512, 2, 3)
ResBlock	(512, 2, 3)	(512, 1, 2)
ResBlock	(512, 1, 2)	(512, 1, 1)
Reshape	(512, 1, 1)	(512)
Linear	(512)	(512)
LeakyReLU	(512)	(512)
Linear	(512)	(1)

1D and 3D tensors, refer to A.1.1.

Table A.7. Base discriminator architecture. The output of this model is used for both single latent discriminator and temporal action-conditioned discriminator.

D_{base}		
Layers	Input Shape	Output Shape
SNLinear	(MTS, 512)	(MTS, 1024)
BatchNorm1d	(MTS, 1024)	(MTS, 1024)
LeakyReLU	(MTS, 1024)	(MTS, 1024)
SNLinear	(MTS, 512)	(MTS, 1024)
BatchNorm1d	(MTS, 1024)	(MTS, 1024)
LeakyReLU	(MTS, 1024)	(MTS, 1024)
SNLinear	(MTS, 512)	(MTS, 1024)
BatchNorm1d	(MTS, 1024)	(MTS, 1024)
LeakyReLU	(MTS, 1024)	(MTS, 1024)
SNLinear	(MTS, 512)	(MTS, 1024)
BatchNorm1d	(MTS, 1024)	(MTS, 1024)
LeakyReLU	(MTS, 1024)	(MTS, 1024)

A.3 Action Prediction Model Architecture

The action prediction model accepts the image size of either 64×64 or 48×80. As for the input of the model, two temporally consecutive images are concatenated along channel dimension. The output dimension is an action space of a dataset. Tab. A.11 and Tab. A.12 show the architectures of the model with image size of 64×64 and 48×80, respectively. ResBlock used here has the same components as one used in the discriminator in 1st training phase.

Table A.8. Single latent discriminator architecture.

D_{single}		
Layers	Input Shape	Output Shape
SNLinear	(MTS, 512)	(MTS, 1024)
BatchNorm1d	(MTS, 1024)	(MTS, 1024)
LeakyReLU	(MTS, 1024)	(MTS, 1024)
SNLinear	(MTS, 512)	(MTS, 1)

Table A.9. Temporal action-conditioned discriminator for maximum time steps of 64. The outputs from 1st to 4th branches are used, respectively, to discriminate if the data comes from real or not.

D_{temporal}		
MTS	64	
Layers	Input Shape	Output Shape
1st Branch		
SNConv2d	(2048, 64, 1)	(256, 31, 1)
LeakyReLU	(256, 31, 1)	(256, 31, 1)
2nd Branch		
SNConv2d	(256, 31, 1)	(512, 15, 1)
LeakyReLU	(512, 15, 1)	(512, 15, 1)
3rd Branch		
SNConv2d	(512, 15, 1)	(1024, 7, 1)
LeakyReLU	(1024, 7, 1)	(1024, 7, 1)
4th Branch		
SNConv2d	(1024, 7, 1)	(1024, 3, 1)
LeakyReLU	(1024, 3, 1)	(1024, 3, 1)

Table A.10. Temporal action-conditioned discriminator for maximum time steps of 128. The outputs from 1st to 4th branches are used, respectively, to discriminate if the data comes from real or not.

D_{temporal}		
MTS	128	
Layers	Input Shape	Output Shape
1st Branch		
SNConv2d	(2048, 128, 1)	(512, 63, 1)
LeakyReLU	(512, 63, 1)	(512, 63, 1)
2nd Branch		
SNConv2d	(512, 63, 1)	(1024, 31, 1)
LeakyReLU	(1024, 31, 1)	(1024, 31, 1)
3rd Branch		
SNConv2d	(1024, 31, 1)	(2048, 15, 1)
LeakyReLU	(2048, 15, 1)	(2048, 15, 1)
4th Branch		
SNConv2d	(2048, 15, 1)	(2048, 7, 1)
LeakyReLU	(2048, 7, 1)	(2048, 7, 1)

Table A.11. Action prediction model architecture for 64×64 image size.

Action prediction model		
Image Size	64×64	
Layers	Input Shape	Output Shape
Conv2d	(6, 64, 64)	(128, 64, 64)
ReLU	(128, 64, 64)	(128, 64, 64)
ResBlock	(128, 64, 64)	(256, 32, 32)
ResBlock	(256, 32, 32)	(512, 16, 16)
ResBlock	(512, 16, 16)	(512, 8, 8)
ResBlock	(512, 8, 8)	(156, 4, 4)
ResBlock	(156, 4, 4)	(512, 2, 2)
ResBlock	(512, 2, 2)	(512, 1, 1)
Reshape	(512, 1, 1)	(512)
Linear	(512)	(512)
ReLU	(512)	(512)
Linear	(512)	(action space)

Table A.12. Action prediction model architecture for 48×80 image size.

Action prediction model		
Image Size	48×80	
Layers	Input Shape	Output Shape
Conv2d	(6, 48, 80)	(128, 48, 80)
ReLU	(128, 48, 80)	(128, 48, 80)
ResBlock	(128, 48, 80)	(256, 24, 40)
ResBlock	(256, 24, 40)	(512, 12, 20)
ResBlock	(512, 12, 20)	(512, 6, 10)
ResBlock	(512, 6, 10)	(156, 3, 5)
ResBlock	(156, 3, 5)	(512, 2, 3)
ResBlock	(512, 2, 3)	(512, 1, 2)
ResBlock	(512, 1, 2)	(512, 1, 1)
Reshape	(512, 1, 1)	(512)
Linear	(512)	(512)
ReLU	(512)	(512)
Linear	(512)	(action space)