

A More Related Works

World Model for Agent Learning. Developing world models for training agents has been a long-standing research focus, aimed at enhancing policy learning within simulated environments rather than solely achieving high-fidelity reconstructions of observations. This research involves two primary stages: 1) modeling the training environment by reconstructing observations, rewards, and continuation signals, often through a recurrent state-space model; and 2) utilizing this model to predict future states, enabling reinforcement learning to optimize robust policy functions. Studies indicate that this method provides sample efficiency gain of over 1000% compared to directly learning policies from real environments, shows resilience across diverse domains, and can outperform fine-tuned expert agents on a range of benchmarks and data budgets [7]. Key contributions in this area include Recurrent World Models [39], Dreamer (v1 [40], v2 [3], and v3 [7]), TD-MPC (v1 [41] and v2 [42]), DayDreamer [43], SafeDreamer [44], and MuDreamer [45]. Notably, MuZero [2] runs the self-play of Monte Carlo tree search to build world models for Atari, Go, chess and shogi, without external data.

B Details in Experiments

B.1 DiT Backbone

The DiT backbone is adapted from the publicly available DiT models [19]. It consists of a patch embedding module, a caption embedding module, a timestep embedding module, 32 DiT blocks, followed by a linear output head with LayerNorm [20]. The followings provide details of each module within the DiT backbone.

The Patch Embedding Module. The patch embedding module employs a 3D convolution with a kernel size of $1 \times 2 \times 2$, followed by a reshape operation. Thus, the convolution can effectively process the video latent from the VAE encoder, and the reshape operation can further transform the feature into a sequence of tokens with 2,048 feature size. By using a 3D convolution, the module captures both spatial and temporal features, ensuring that the token sequence retains essential information from the video data.

Caption Embedding Module. The caption embedding module takes the caption token sequence encoded by the T5 model and further processes it through a two-layer FFN. Both the hidden feature size and the output feature size are set to 2,048, allowing the module to generate rich and high-dimensional representations of the caption data.

Timestep Embedding Module. The timestep embedding module is implemented as a sinusoidal embedding module followed by a two-layer FFN. Both the hidden feature size and the output feature size of this FFN are set to 2,048.

DiT Block. Each DiT block includes a self-attention layer operating on network features, a cross-attention layer linking conditions with self-attention outputs, and an FFN layer composed of two linear layers with a GELU activation [21] in between.

B.2 Training Details

Upon obtaining the base DiT model, the training process consists of four distinct stages: (1) warm-up on unlabeled *Source*, (2) training of the Interactive Module, (3) fine-tuning using Swin-DPM, and (4) Stream Consistency Model distillation. Below, we first outline the common training configurations utilized across all stages, followed by a detailed description of each individual phase.

Common Settings. All training procedures were executed with an overall batch size of 32 and a learning rate of 1×10^{-5} . Mixed-precision training was employed using bfloat16 to enhance computational efficiency. During preprocessing, all video inputs were resized to a resolution of 1280×720 pixels and set to 16 FPS. For sequences exceeding 25,200 frames in length, we used the Deepspeed Ulysses sequence parallelism strategy [46], distributing the sequence across 8 GPUs to manage memory and computational demands effectively.

Warm-Up on Unlabeled Source Dataset. In the initial warm-up stage, we fine-tuned all linear layers of the base DiT model using Low-Rank Adaptation (LoRA) to tailor the model to the source data distribution [22]. The LoRA rank was set to 128, and the model was trained for 20,000 steps. This adaptation ensures that the model parameters are suitably adjusted to the characteristics of the unlabeled source dataset before advancing to subsequent training phases.

Training of Interactive Module. The second stage focuses on training the Interactive Module, each of which is integrated after every two consecutive DiT blocks, totaling 16 Interactive Module. During this phase, the parameters of the base DiT model were frozen to concentrate the training solely on the Interactive Module. This stage was conducted over 20,000 training steps, enabling the Interactive Module to effectively interface with the base model without altering its foundational parameters.

Fine-Tuning Using Swin-DPM. The third stage involves comprehensive fine-tuning of all model parameters, including both the base DiT model and the Interactive Module, utilizing the Swin-DPM approach. This extensive fine-tuning was carried out over 60,000 steps, allowing for the refinement and optimization of the entire model to better capture data intricacies and enhance overall performance.

Consistency Model Distillation. In the final stage, consistency model distillation was performed using the model from the preceding fine-tuning phase as the teacher model. The student model was initialized with the teacher’s weights to facilitate knowledge transfer. During distillation, we employed a one-stage guided distillation technique [47], incorporating Classifier-Free Guidance (CFG) into the student model. For the Ordinary Differential Equation (ODE) solver within the consistency distillation framework, we utilized the Euler solver with a single-step size of 25/1000. This distillation process was conducted over 10,000 training steps.

C The Source Dataset

C.1 The GameData Platform

We build a framework, *GameData* Platform, for data collection. The framework consists of three components: Controlling, Simulation, and Observation.

Controlling. In most games, we need to control a character to go to different scenes and make interactions. Intrinsically, the collected data can be reconstructed with initial states of game worlds and a series of control signal. In order to make the collected data clean and meaningful, instead of being stuck in one corner, we designed two different control systems, namely the automatic one and the manual one. For the automatic control system, we use Cheat Engine for pivotal data access, such as XYZ coordinates in games. These data can be used to determine whether the game has been stuck for some time. We detect the coordinates of a past period of time and determine whether they are covered in a circle of a given size. If the game is detected as stuck, we will reset the game state and restart the recording. Generally, the automatically generated control signals will move randomly, change direction, and change perspective. This is good enough for games that move on a 2D-like surface. However, for games moving in a 3D space, random signals will struggle with generating meaningful content, so we have to change to the manual system. Since our game is running and captured on cloud servers, human data collectors will observe the game through a low-definition streaming and control manually. Signals (from keyboards, mice, and gamepads) are translated and delivered through the socket server, and cloud servers will generate keyboard events through the virtual keyboard. Here, the latency between the control signals and the OBS screen recording is crucial. We eventually found that the control signals recorded on the cloud sever and the actual action responses in the recorded videos were generally no more than three frames apart. and in general, this delay is stable and can be subtracted directly from the timeline.

Simulation. The game runs directly on the cloud servers. we can directly copy the server images to get a large number of running instances. The recorded videos and control signals will be uploaded to the data center. We set up a series of video quality checks to filter out samples of low quality (still or overly noisy videos, and some undefined scenes). All games run at the highest quality while ensure the OBS screen recording does not get stuck. In order to avoid overly complicated situations, we

487 removed the NPCs and running vehicles in the game. We use the Reshade to adjust the game scenes
488 to make it more reality-like.

489 **Observation.** We use OBS as the screen recorder. One can use scripts to control OBS for automatic
490 recording. We recorded the game at native resolution of 2560×1600 (higher resolutions may cause
491 the game and recording to lag). For the reality of the recorded videos, we removed GUIs and texts
492 in the game through a Reshade plugin, namely ReshaderEffectShaderToggler.¹ It can turn off the
493 rendering of GUI related shaders in the game while left the native video untouched.

494 **Forza Horizon 5.** In Forza Horizon 5, a telemetry mechanism can be used for game status retrieving.
495 We can access the real-time game data through socket after checking on the telemetry option in
496 settings. An example script for data listening can be found here.² We can access XYZ coordinates,
497 velocities and accelerations. We use these data for stuck detection and sample filtering. Since Forza
498 Horizon 5 is a game that mainly takes on 2D area, we apply automatic pipeline that randomly walking
499 on different game scenes (like dessert, grassland, the watery and the snowy areas). Control signals are
500 simplified to going forward, turning left and turning right. During the data collection stage, if XYZ of
501 is still for several seconds, the controller will try to move back. And if the 40 position points collected
502 during the last 40 seconds can be covered with a circle with radius of 80 meters, the controller will
503 try to teleport the car to a random position. After raw sample collection, we apply some strategies to
504 filter out samples of low quality. We use the acceleration data to detect if the car has collided with
505 anything, and drop these video clips with collision. Sometimes the car is moving backward while
506 the controlling input is moving forward, this is because the direction of movement in the game is to
507 provide acceleration. We filter out data with a large angle between acceleration and velocity. Due to
508 some problems in the game itself, the video often changes suddenly at some time. We filter out video
509 clips with large average error between any two adjacent frames.

510 **Cyberpunk 2077.** Cyberpunk 2077 is a game that offers realistic visuals and lighting effects. Due
511 to the complexity of the game terrain, we have to choose the manual pipeline. For simplicity, the
512 actions in game are reduced to two separated inputs. The first one makes the character move forward
513 or stop. And the second one makes the direction of the character’s sight move up, down, left and
514 right. We disable the NPCs and moving vehicles with game mod. During the data collection, players
515 observe the game through low-definition OBS streaming and send control signals. The signals are
516 then mapped into “W” (moving forward) / “U” “D” “L” “R” (up, down, left and right) on the cloud
517 servers. We access and record the XYZ coordinates of player through Cheat Engine. These coordinate
518 sequences are then used for filtering out video clips where collisions occur between the character and
519 the game scene.

520 C.2 The Source Dataset

521 We present the *Source* dataset from three perspectives: basic information, the annotation method
522 used to convert the original data from *GameData Platform* to our desired format, and the filtering
523 method applied to remove undesirable data.

524 C.2.1 Basic Information

525 The *Source* comprises data from both Forza Horizon 5 and Cyberpunk 2077. For Forza Horizon 5,
526 we collected approximately 1,200,000 pairs of video and control signals, while for Cyberpunk 2077,
527 we gathered around 1,000,000 such pairs. All collected videos have a duration of approximately
528 6 seconds, recorded at 60 FPS. For Forza Horizon 5, we specifically collected data from multiple
529 scenes, including deserts, oceans, water bodies, grasslands, and fields. The videos from different
530 scenes are illustrated in Figure A1, along with the distribution of data volume for each scene Figure
531 A2(a). For Cyberpunk 2077, we focused on gathering data from urban environments that feature a
532 significant number of tall buildings.

533 In Forza Horizon 5, the dataset includes only three distinct control signals: “moving forward”
534 (denoted by “D”), “moving forward and turning left” (denoted by “DL”), and “moving forward and
535 turning right” (denoted by “DR”). In contrast, the data for Cyberpunk 2077 encompasses five different

¹<https://github.com/4lex4nder/ReshadeEffectShaderToggler>

²<https://github.com/jasperan/forza-horizon-5-telemetry-listener>



Figure A1: Examples of Horizon5 across different scenarios.

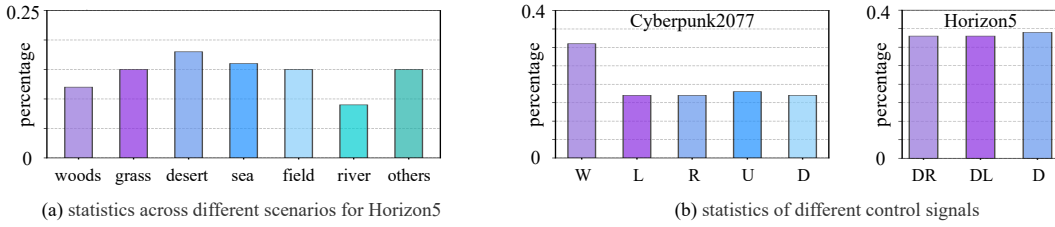


Figure A2: (a) The statistics for Horizon5 across different scenarios include woods, grass, desert, sea, fields, rivers, and others. The results indicate that the quantity of data across these scenarios is relatively balanced. (b) The statistics of different control signals for Cyberpunk 2077 and Horizon5. In Cyberpunk 2077, the percentage of the "move forward" signal is relatively high, while other steering control signals are distributed more evenly. In Horizon5, all three control signals are evenly distributed.

control signals: "moving forward" (denoted by "W"), "turning left" (denoted by "L"), "turning right" (denoted by "R"), "looking upward" (denoted by "U"), and "looking downward" (denoted by "D").

C.2.2 Annotation Methods

The original data from **GameData Platform** typically has a duration of around 10 minutes, which is excessively long for training *The Matrix*. Therefore, we use FFmpeg [48] to segment these videos into 6-second clips. Next, we extract the corresponding control signals from the complete set of signals. We then use InternVL [49] to generate captions based on 12 uniformly extracted key frames from the videos. After the captioning process with InternVL, we perform manual corrections on the generated captions to eliminate obvious errors.

C.2.3 Filtering Methods

After the annotation step, a significant amount of undesired data remains, which could disrupt the training of *The Matrix*. To address this, we employ five filtering methods to eliminate these problematic data points, which we introduce as follows. Note that for Cyberpunk 2077, since we utilize human data collection rather than automatic methods, many of the following issues do not exist.

Balance Control Signals. Balancing the number of different control signals is beneficial for the training of *The Matrix*. The process of balancing control signals consists of three steps: 1) First, we analyze the distribution of control signals for each 6-second video and record the results. 2) Next, we assess the overall distribution of control signals across the entire dataset to identify the most frequently occurring control signal. 3) Finally, we remove some data points that contain the highest proportion of this predominant control signal. We repeatedly implement the second and third steps until the distribution is relatively balanced. The distribution results for Forza Horizon 5 and Cyberpunk 2077 are reported in Figure A2 (b). We provide the pseudocode for the algorithm in Algorithm 1.

Algorithm 1 Control Signal Balancing Algorithm

Require: Dataset D containing control signals from 6-second videos

Ensure: Balanced Dataset B

```
1: Initialize  $B$  as an empty set
2: for each video  $v$  in  $D$  do
3:   Analyze the distribution of control signals in  $v$ 
4:   Record the results for  $v$ 
5: end for
6: while not isBalanced( $B$ ) do
7:   overallDistribution  $\leftarrow$  Assess the overall distribution of control signals in  $D$ 
8:   mostFrequentSignal  $\leftarrow$  Identify the most frequently occurring control signal from overallDis-
      tribution
9:    $D \leftarrow$  Remove data points from  $D$  that contain mostFrequentSignal
10: end while
11: Set  $B \leftarrow D$ 
12: return  $B$ 
```

560 **Detect and Remove the Data with Collisions.** In Forza Horizon 5, randomly generated control
561 signals often cause the car to collide with walls or rocks. Additionally, the car may be struck by other
562 vehicles. These collisions can severely disrupt the training process, making it essential to identify
563 and remove collision-affected data. Our analysis revealed that collisions consistently result in abrupt
564 changes in acceleration over a very short time. Thus, we use significant variations in acceleration as a
565 reliable indicator of collision events and discard any corresponding data to maintain the integrity of
566 the training process.

567 **Detect and Remove Stuck Data.** In Forza Horizon 5, after colliding with walls or rocks, the car often
568 gets stuck; even when the “D” key is pressed, the car fails to move. This stuck situation complicates
569 the training data and negatively impacts the performance of *The Matrix*. Therefore, we need to detect
570 and remove such instances. Detecting when the car is stuck is relatively straightforward—we simply
571 calculate the distance the car has traveled within the video. If this distance falls below a certain
572 threshold, we conclude that the car is stuck and discard the corresponding data.

573 **Detect and Remove the Data with Mismatched Motion and Control.** As introduced in ap-
574 pendix C.1, to quickly resolve a stuck situation, the car will move backward when stuck. As a result,
575 it is possible for the car to still move backward at a slower speed even when the “D,” “DL,” or “DR”
576 keys are pressed. Similar situations may arise when “DL/DR” is pressed for a long period and then
577 switched to “DR/DL.” Although the acceleration is directed to the right/left, the car may continue to
578 move in the opposite direction for a brief period. We refer to this as mismatched motion and control,
579 which complicates the training process. To address this issue, we calculate the directions of both
580 the acceleration and the car’s movement. If the angle between these two directions is too large, we
581 discard the corresponding data.

582 **Detect and Remove Artifacts.** In Forza Horizon 5, visual artifacts can occur when a car collides
583 with obstacles like trees, introducing distortions into the generated videos. To filter out such corrupted
584 data, we detect variations in pixel values across consecutive frames. Our analysis shows that applying
585 a high threshold effectively identifies all videos containing these artifacts, enabling their removal.

586 C.3 The DROID dataset

587 C.3.1 Basic Information

588 DROID is a large, diverse robot manipulation dataset containing 76k demonstration trajectories
589 (350 hours of interaction) collected across 564 scenes and 86 tasks over 12 months by 50 collectors
590 worldwide. It aims to improve the performance, robustness, and generalization of robotic manipulation
591 policies. DROID uses the same hardware setup across all 13 institutions to streamline data collection
592 while maximizing portability and flexibility. The setup consists of a Franka Panda 7DoF robot arm,
593 two adjustable Zed 2 stereo cameras, a wristmounted Zed Mini stereo camera, and an Oculus Quest 2

594 headset with controllers for teleoperation. Everything is mounted on a portable, height-adjustable
595 desk for quick scene changes.

596 C.3.2 Filtering Methods

597 **Remove Overly Complex Scenes and Balance Different Scenes.** DROID is a collaborative effort
598 involving multiple laboratories and contains data from 11 different environments, including domestic
599 scenes like kitchens and bedrooms, as well as industrial settings such as factories and laboratories.
600 Due to the complexity of these scenes, which poses challenges for subsequent captioning and video
601 learning, we first classify the data based on scene labels. For each category, we manually select 50
602 less complex scenes. We then use DINO to encode and extract semantic features to calculate the
603 mean, removing outliers within each scene based on this semantic mean. To balance the number
604 of training samples across scenes, we ensure that the final dataset contains an approximately equal
605 number of samples from each scene after outlier removal.

606 **Filtering Frames Without Arm Presence and Removing Failed Executions.** Since some frames
607 in the videos do not contain the robotic arm or have only a small visible area, we use Grounding
608 DINOv2 [50] to remove such frames. If more than 20% of the frames in a video meet this condition,
609 the entire video is discarded. Additionally, to ensure accuracy in control, we remove data where the
610 robotic arm fails to follow the intended trajectory successfully. Finally, we use the spatial position of
611 the robot gripper as a condition for each frame in the training.

612 D More Examples

613 D.1 Generalization

614 We provide more results on the generalization ability of *The Matrix* in Figure A3.

615 D.2 Long Video Generation

616 We provide several long video generation demos in the *Supplementary Video files*. Please check them
617 after Unzip. **All videos are heavily compressed to satisfy the supplementary file size limit.**

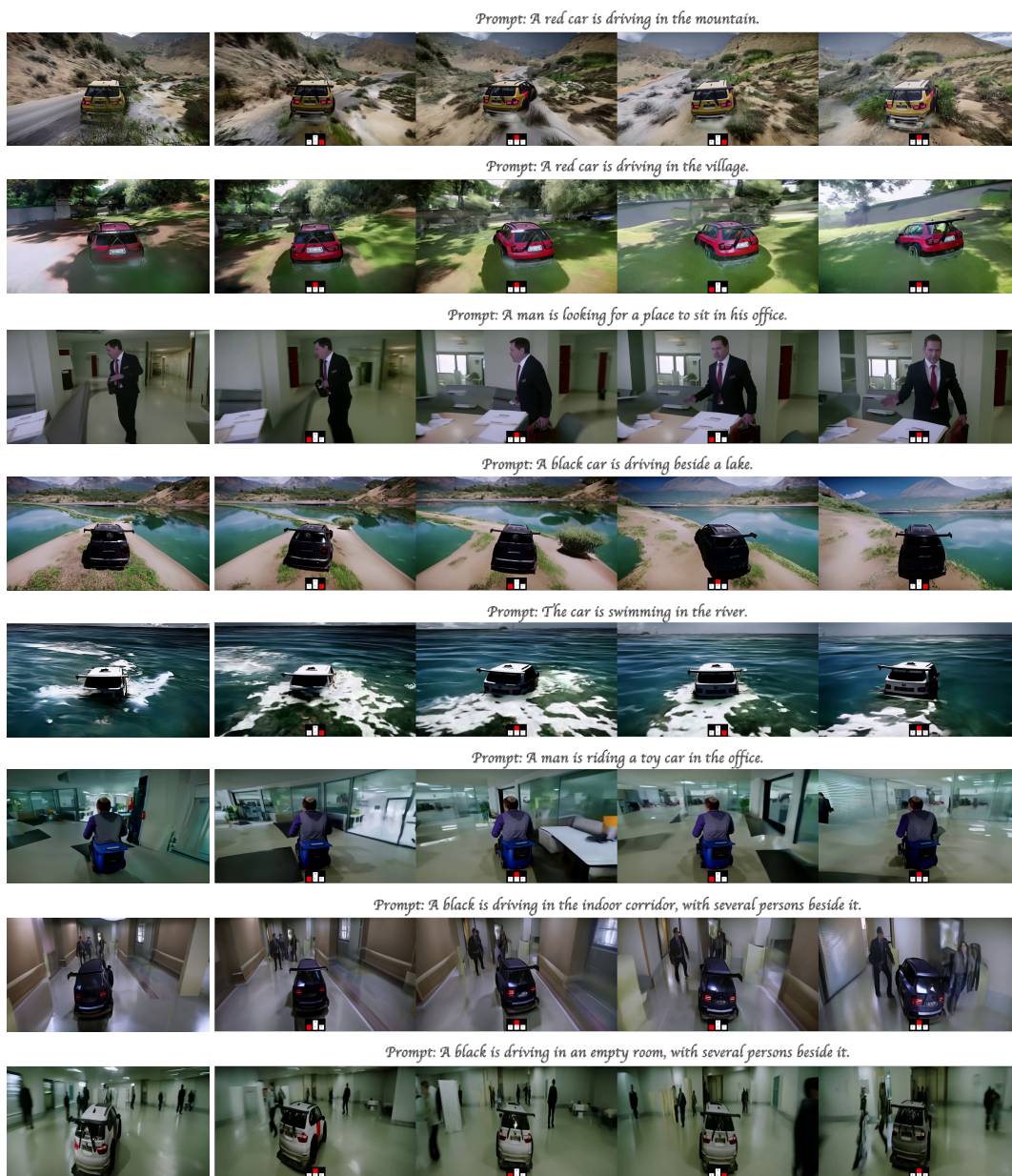


Figure A3: More generalization results of *The Matrix* on unseen scenes and objects.