# Moto: Latent Motion Token as the Bridging Language for Learning Robot Manipulation from Videos

Supplementary Material

## 7. Details on Experiment Setup

### 7.1. Benchmarks

**SIMPLER.** On the SIMPLER benchmark, we focus on three tasks concerning the Google Everyday Robot embodiment: `Pick Coke Can`, `Move Near`, and `Open/Close Drawer`, as illustrated in Fig. 13. The "Pick Coke Can" task involves grasping and lifting the empty coke can in three different orientations: horizontal laying, vertical laying, and standing. The "Move Near" task places 3 (out of 8) objects in a triangle pattern on the tabletop and instructs the robot to move a designated source object near another object as the target. The "Open/Close Drawer" task requires the robot to open or close a specific drawer from the top, middle, or down position of a cabinet.


Figure 13. Illustration of the evaluation tasks in SIMPLER [31].

**CALVIN (ABC⟶D).** The CALVIN benchmark uses a Franka Emika Panda robot embodiment. It evaluates long-horizon task completion with unconstrained language instructions. In each trial, the robot should accomplish 5 (out of 34) tasks in a row. There are four different environments (A, B, C, D), each containing a desk with a sliding door, a drawer, differently colored blocks, a button that toggles an LED, and a switch controlling a lightbulb. As shown in Fig. 14, the environments differ in the textures of the desk, and the positions of all static elements including the sliding door, the drawer, the LED button, and the lightbulb switch. We conduct experiments under the most challenging ABC⟶D setting, i.e., training on data from environments A, B, and C while testing in D.

**Real-world Robot Experiments.** We design three tasks for real-world evaluation: `Pick-Place Banana` (picking up a banana from the tabletop and placing it in a pan), `Close Laptop` (pushing the laptop's lid until it is completely closed), and `Disassembly` (removing the stick that is assembled in the slot). All tasks are executed with a FANUC LR Mate 200iD robot arm, as shown in Fig. 3. Each task is tested for 10 times, with the initial positions of objects randomized. For generalization evaluation, we consider two scenarios: (i) Novel Object: we change the color, texture, and shape of the manipulated object; (ii) Visual Distractor: we add irrelevant objects as distractors.

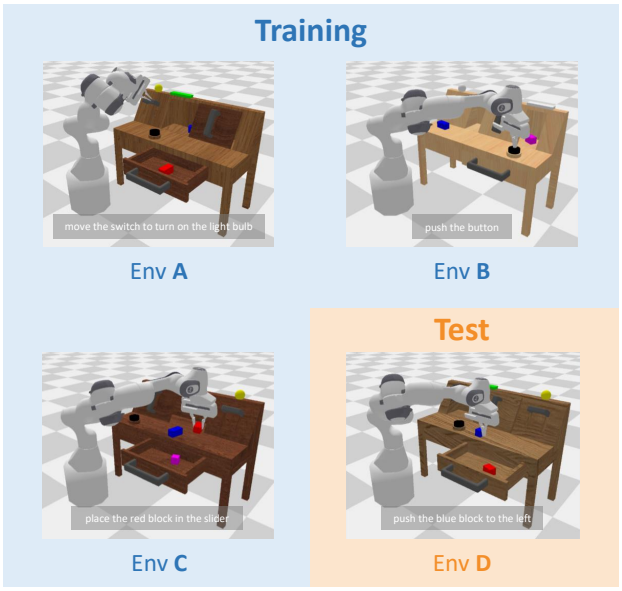


Figure 14. Illustration of the four different environments in CALVIN, adapted from the original figure in Mees et al. [39].

### 7.2. Datasets

**Training Latent Motion Tokenizer & Pre-training Moto-GPT.** On the SIMPLER benchmark, we use a subset of Open-X-Embodiment (OXE) [52] to train the Latent Motion Tokenizer and pre-train Moto-GPT, including 109k real-world trajectory videos from RT-1 Robot Action [4], Bridge [53], Task Agnostic Robot Play [40, 48], Jaco Play [13], Cable Routing [36], RoboTurk [38], NYU VINN [44], Austin VIOLA [61], Berkeley Autolab UR5 [9], and TOTO [60] datasets across various embodiments. On CALVIN, we use all the play videos from environments A, B, and C to train the Latent Motion Tokenizer. 35% data (including 18k trajectory videos) with language annotations is used for autoregressive pre-training. The real-world experiments also use OXE data for pre-training.

**Fine-tuning Moto-GPT.** On the SIMPLER benchmark, we use the 73k action-labeled real-world expert trajectories from RT-1 Robot Action [4] to fine-tune the policy model. On the CALVIN benchmark, we use the 18k demonstration trajectories with language annotations and action labels from environments A, B, and C for fine-tuning. For real-world experiments, we collect 90 demonstration trajectories (30 for each task) with teleoperation for fine-tuning.

Note that all the pre-training and fine-tuning data for SIMPLER come from the real world instead of simulation environments. This setting aims to study the model's transfer ability between real and simulation scenarios. On the other hand, the ABC-only setting for CALVIN training data aims to evaluate the model's zero-shot generalization capability to the unseen environment D.

### 7.3. Compared Models

**SIMPLER.** On the SIMPLER benchmark, we mainly compare Moto-GPT with five representative models pre-trained with Open-X-Embodiment datasets:

- **RT-1-X** [4] uses a transformer backbone to output tokenized actions with a FiLM EfficientNet to fuse language and 6 history images into token inputs.
- **RT-2-X** [62] adapts the pre-trained large vision-language model (VLM), PaLI-X (55B), into a robot policy by casting tokenized actions into text tokens.
- **Octo-Base** [41] employ a transformer architecture to process language and image tokens, with a diffusion-based action head to produce actions.
- **OpenVLA** [27] builds on a pre-trained Prismatic-7B VLM backbone for robot action prediction.
- **OpenVLA (fine-tuned)** further fine-tunes OpenVLA on the RT-1 Robot Action dataset [4], despite its action-labeled pre-training data already contains this dataset.

**CALVIN.** On the CALVIN benchmark, we select the following baseline models that leverage pre-training strategies to improve robot manipulation performance:

- **SuSIE** [3] pre-trains an image editing model to generate the goal image, which is fed into a low-level policy for action prediction.
- **RoboFlamingo** [32] is a robot policy model adapted from OpenFlamingo, a large VLM pre-trained on extensive vision-language corpus.
- **GR-1** [54] pre-trains a GPT-style transformer to directly predict the pixel values of a single-step future observation for each input observation.
- **MT-R3M** [54] is a variation of GR-1, which leverages the pre-trained robot visual encoder R3M [42] to encode observation images.

Table 4. Implementation details of the Latent Motion Tokenizer.

| Component | Parameter | Value |
|---|---|---|
| M-Former | num_queries | 8 |
| | num_layers | 4 |
| | hidden_size | 768 |
| | num_heads | 12 |
| ViT Decoder | patch_size | 16 |
| | num_layers | 12 |
| | hidden_size | 768 |
| | num_heads | 12 |
| VQ Codebook | num_codes | 128 |
| | latent_dim | 32 |

**Ablations of Moto-GPT.** We study the following variations of Moto-GPT as optional baselines:

- **Moto w/o Motion Token** shares the same backbone with Moto-GPT but is trained from scratch on action-labeled robot data without latent motion tokens.
- **Moto-IML** undergoes the same pre-training stage as Moto-GPT. It keeps latent motion tokens in the input sequence but ignores the next-motion-token-prediction loss during the fine-tuning stage.
- **Moto-DM** is pre-trained in the same way as Moto-GPT but completely discards latent motion tokens in the input sequence during fine-tuning.

## 8. Training Details

### 8.1. Latent Motion Tokenizer

The implementation details for the trainable modules of the Latent Motion Tokenizer are summarized in Table 4. We use the hyperparameters listed in Table 5 to train this model on four 40G GPUs. To facilitate the learning of latent motion tokens, we downsample the original videos in the training dataset, ensuring that the visual motion between frames is sufficiently distinct. Specifically, for videos from the OXE data, we sample one frame every three frames (i.e., $\Delta t = 3$) for videos from the RT-1 Robot-Action subset. The sampling rates for videos from other OXE subsets are adjusted based on the ratio of their fps to that of RT-1 Robot-Action videos. For human videos, $\Delta t$ is empirically set to 6. We train the Latent Motion Tokenizer for 350k steps. For videos from the CALVIN dataset, we adopt a sampling rate of one frame every five frames ($\Delta t = 5$) and train the model for 150k steps. For real-world robot experiments, we fine-tune the Latent Motion Tokenizer pre-trained on OXE videos for another 500 steps on the newly collected real-world trajectory videos.

Table 5. Training hyperparameters for Latent Motion Tokenizer.

| Parameter | Value |
|---|---|
| batch_size | 256 |
| optimizer | AdamW |
| lr_max | 1e-4 |
| lr_schedule | cosine decay |
| weight_decay | 1e-4 |
| warmup_steps | 1000 |

Table 6. Implementation details of Moto-GPT.

| Component | Parameter | Value |
|---|---|---|
| GPT backbone | num_layers | 12 |
| | hidden_size | 768 |
| | num_heads | 12 |
| Action Head | num_layers | 2 |
| | hidden_size | 384 |

Table 7. Training hyperparameters for Moto-GPT.

| Parameter | Value |
|---|---|
| batch_size | 512 |
| optimizer | AdamW |
| lr_max | 1e-4 |
| lr_schedule | cosine decay |
| weight_decay | 1e-4 |
| warmup_epochs | 1 |

### 8.2. Moto-GPT

We present the implementation details of Moto-GPT in Table 6, where the Action Head is included only during the fine-tuning phase. Moto-GPT handles a maximum video length of three frames, and the video downsampling rate applied during both the pre-training and fine-tuning stages is consistent with the rate used for training the Latent Motion Tokenizer. When fine-tuning Moto-GPT across different benchmarks, the number of action query tokens inserted after the latent motion tokens at each time step varies. Specifically, for the SIMPLER benchmark, we insert three action query tokens, whereas for the CALVIN benchmark, we insert five. For pre-training, Moto-GPT is trained for 10 epochs using eight 40G GPUs, with the relevant hyperparameters outlined in Table 7. The hyperparameters for fine-tuning remain consistent with those used during pre-training, with the exception of the number of epochs. During fine-tuning, Moto-GPT is trained for three epochs on the RT1-Robot-Action dataset and 18 epochs on the CALVIN dataset, utilizing four 40G GPUs. For real-world experiments, we start with the same pre-trained checkpoint of Moto-GPT as adopted for the SIMPLER benchmark. We further pre-train it with a combination of OXE videos and the 90 newly collected trajectory videos for 5 epochs. Then we fine-tune it with real robot action labels for 20 epochs.

## 9. Details of Experiments

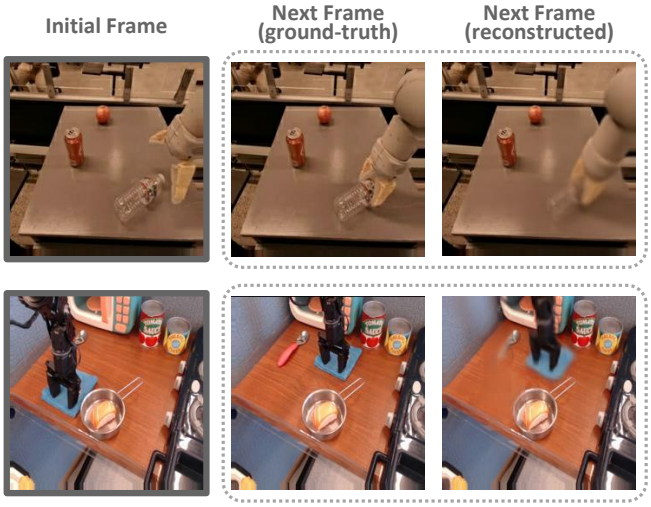


Figure 15. Qualitative examples of reconstruction results, where discrete motion tokens obtained from the Latent Motion Tokenizer based on the initial and next frame, are fed into the decoder along with the initial frame to reconstruct the target frame.

Table 8. Top-K motion token prediction accuracy of Moto-GPT in predicting ground-truth latent motion tokens from a 128-size codebook on the validation splits of the pre-training datasets.

| Dataset | Top-5 | Top-10 | Top-20 |
|---|---|---|---|
| Oepn-X-Embodiment | 0.521 | 0.698 | 0.853 |
| Calvin (ABC⟶D) | 0.298 | 0.518 | 0.768 |

## 10. Limitations & Future Work

This paper introduces Moto, a novel method that uses latent motion tokens as a "language" interface to bridge generative pre-training on video data with precise robot control. Moto opens several exciting avenues for future work.

Firstly, Moto demonstrates the feasibility of learning a unified language to interpret diverse visual dynamics from videos, eliminating the need for hardware-specific action labels. The latent motion trajectories tokenized from videos provide a rich resource for models to learn motion priors closely related to low-level actions. While we currently mainly use robot videos to train the Latent Motion Tokenizer, the learned latent motion tokens demonstrate the potential to produce consistent visual motions across varied contexts and embodiments. Notably, our preliminary

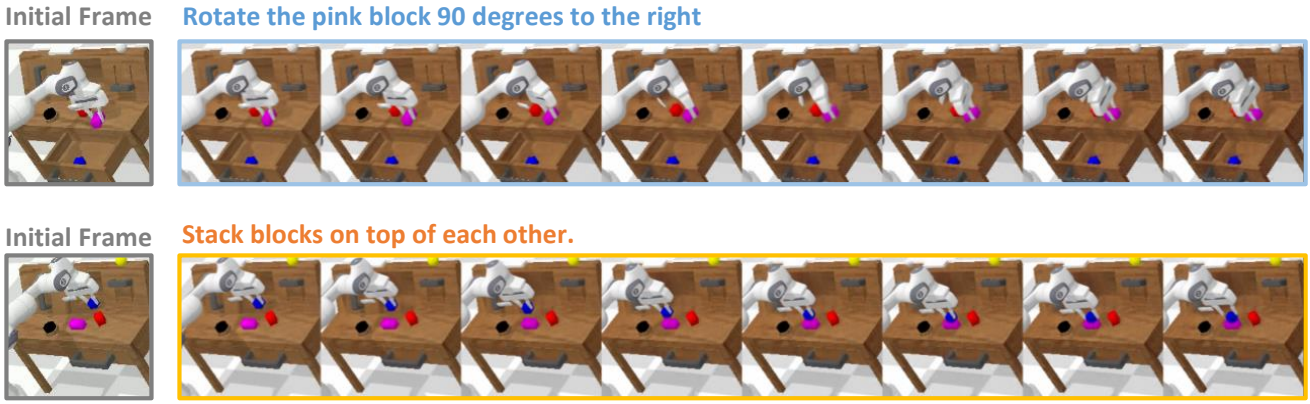


Figure 16. Predicted video trajectories by the pre-trained Moto-GPT for CALVIN tasks reflecting delicate robot actions.

experiments with SSV2 videos show promising results in human-to-robot motion transfer via latent motion tokens. We believe a similar approach could be applied to model more complex human motions, enabling robots to learn a wealth of world knowledge from Internet-scale videos.

Besides, the Moto-GPT pre-trained on videos tokenized into latent motion token sequences and fine-tuned on action-labeled trajectories, effectively transfers motion priors learned from (even human) videos to actual robot action prediction. This is particularly beneficial in low-resource scenarios. Future work could scale up pre-training video data and optimize fine-tuning to improve model performance on downstream robot tasks further.

Moreover, while Moto is primarily utilized to enhance imitation learning for robot manipulation tasks, it shows potential as a reward model for measuring trajectory rationality and as a vivid environment simulator. Future research could explore Moto's use in improving the robustness of reinforcement learning agents and extending its application to a wider range of robotic tasks, such as navigation and locomotion, to develop a more versatile robot action policy.