Learning Long-term Motion Embeddings for Efficient Kinematics Generation

Supplementary Material

A. Additional Evaluation Details

LIBERO Trajectory Prediction We compare the accuracy of our model's predicted motions against both discriminative and generative baselines in Tab. E. Discriminative methods output a single set of trajectories for a given start frame and text prompt, while generative methods output distributions of possible trajectories. Across all tasks, our model significantly outperforms both discriminative and generative baselines, demonstrating superior understanding of the scene dynamics and the task at hand. For the generative methods, we report the Min MSE metric from WHN [9] for $k = 8$ samples (*Min*) and for fair comparison with discriminative methods, also for $k = 1$ (*Single*).

We closely follow the evaluation protocol of previous methods [9, 45, 47]: We split the videos into temporal windows of length T. Per temporal window, we extract $n = 32$ trajectories that exceed a certain variance threshold, i.e., are non-static. Given the start frame of the temporal window and the n starting positions, models have to predict the trajectories for the following T frames. All existing methods use a temporal window size of $T = 16$ [9, 45, 47]. Since our model predicts 64 frames, we interpolate the baseline window of 16 frames up to 64 frames during training. At inference, we subsample our model's prediction from 64 frames down to 16 frames to match the evaluation protocol. Pseudo-GT tracks are obtained through the exact preprocessing pipeline used by baselines [9, 45, 47]. The MSE metrics are computed on a resolution of 128×128. We train a single model jointly on both task suites (LIBERO-90 and LIBERO-10) and both views (side and effector).

LIBERO Action Prediction We train a policy head, also sometimes referred to as *Inverse Dynamics Module*, to predict robot actions from our generated motion embeddings. In the LIBERO setting, actions are 7-dimensional vectors. We instantiate the policy head as a shallow 6-layer transformer with dimensionality 768. During policy head training and policy rollouts, we perform 10 sampling steps for the motion planner to generate the conditional motion embeddings. The policy head cross-attends only to these motion embeddings, thus we can be sure that the planning is actually done by the motion planner. Following ATM [45] and Tra-MoE [47], we train the policy head with a simple L2 regression loss, effectively performing Behavioural Cloning (BC). We train the policy head for 12 hours on 16 H200s. For both the ATM [45] and Tra-MoE [47] baselines, we also follow their respective data configuration for training the motion planner. Unlike ATM [45], we train a single joint motion planner across all LIBERO task suites, which further increases task diversity and planning complexity for our model. To increase robustness of the motion planner during policy rollouts, we lock the DINOv2 image encoder and apply noise augmentation to the start frame during training. Further, we condition the motion planner on the current episode timestep and randomly drop out the start frame as an additional incentive to follow the task conditioning.

Model	LIBERO-90		LIBERO-10	
	Side	Effector	Side	Effector
Discriminative				
ATM [45]	47.82	123.01	59.10	131.58
Tra-MoE [47]	39.77	116.47	50.37	131.75
Generative				
$\text{WHN}_{\text{Single}}$ [9]*	17.89	57.64	26.18	63.47
$\text{Ours}_{\text{Single}}$	**8.83**	**45.23**	**10.73**	**46.27**
WHN_{Min} [9]*	10.99	32.01	13.86	35.93
Ours_{Min}	**5.96**	**27.78**	**7.43**	**25.80**
WHN_{Mean} [9]*	18.32	60.47	26.71	66.35
$\text{Ours}_{\text{Mean}}$	**9.18**	**45.71**	**9.05**	**46.55**

Table E. Comparison on text-conditioned trajectory prediction on the LIBERO [27] dataset, measured by MSE and following the evaluation setup of [9]. Numbers marked with an asterisk * are taken directly from the original paper, as no official checkpoints or training code were released. All other results were reproduced using the official checkpoints. The upper part of the table reports regression-based methods, while the lower part compares generative methods. The $_{\text{Single}}$ metric is the MSE for the first sample. All distributional metrics are computed over $k = 8$ samples per trajectory.

Video Models We compare our approach against two state-of-the-art generative video models that offer start and end frame conditioning, making them suitable baselines for our goal-conditional motion generation task. For Wan [43], we use the `Wan-AI/Wan2.1-FLF2V-14B-720P-Diffusers` implementation from Hugging Face and run it locally. Since Veo 3 is a closed-source model, we obtain samples via the fal.ai API. Due to the high computational and monetary cost associated with sampling from these models, we focus our evaluation on a curated subset of the Pexels dataset [40], selecting 68 videos that feature unconstrained, real-world motion diversity.

To ensure a meaningful evaluation of motion generation,

we carefully selected videos in which the primary moving objects remain clearly visible throughout the entire sequence. Our curated set features a diverse range of scenes, spanning various object categories including humans engaged in different activities, animals, vehicles, natural landscapes, and urban environments. The number of moving objects varies across videos, encompassing both single-object and multi-object scenarios. We further ensured diversity in motion by including clips with a wide spectrum of motion magnitudes, ranging from subtle, fine-grained movements to pronounced, large-scale motions.

Each original Pexels video has a framerate between 24 and 30 fps. We subsample each video with a frame skip of 1, selecting a contiguous window of 64 frames, corresponding to roughly 4-5 seconds of video. To establish ground-truth motion, we randomly sample 1024 query points in the first frame of each video and track them forward throughout the sequence. From these, we randomly select 40 dynamic (i.e., non-static) tracks, which serve as our final ground-truth trajectories for evaluation.

For both video models, we condition generation on the first and last frame of each clip, aligning the setup with our poke-conditional evaluation. The Wan model produces 81 frames at 12 fps, while Veo 3 generates 96 frames at 24 fps. To account for stochasticity and fairly assess distributional metrics, we sample $K = 8$ generations per start–end frame pair, resulting in a total of $68 \cdot 8 = 544$ generated video samples per model.

We then track the same 40 query points in each generated video using the same tracking procedure as was used to establish the ground truth tracks. Since both models output more frames than our ground truth sequences, we downsample the resulting tracks to match the 64-frame ground truth length. Finally, we compute the evaluation metrics outlined in the experiments section, comparing the predicted tracks from the generated videos to the ground truth to assess the quality and diversity of motion generation.

B. Implementation Details

We train our model in two stages: a variational autoencoder (VAE) that compresses tracking data into a latent grid representation, and a flow matching model that generates these latent representations conditioned on task specifications. Table Tab. F summarizes the key hyperparameters for both stages. Both models are trained on 10M video clips from diverse open-set videos, using TAPNext [52] to extract 1024 randomly sampled tracking positions per clip, with poke coordinates normalized to the range [-1,1].

The VAE employs a dual-encoder architecture with 12 layers each for self-attention and cross-attention processing, where tracking tokens attend to image tokens via cross-attention. Image features are extracted using a pretrained DINOv2 ViT-B/14 encoder [31], which remains frozen during initial training and is unlocked later when scaling to larger batch sizes. Tracker points are first Fourier embedded and processed with an MLP before passing them to the transformer. The VAE uses 3D RoPE [12, 37] as positional encoding, compressing the tracking data into a 16×16 latent grid with 16 channels. We set the KL divergence weight to $\beta = 1.0 \times 10^{-7}$ to regularize the latent space and use an L1 reconstruction loss. During early experiments, we evaluated alternative reconstruction losses (L2 and Huber) and alternative prediction targets (auto-regressive deltas and offsets to the start location). Directly predicting absolute coordinates in our normalized space with an L1 loss consistently gave the best performance. The VAE is trained with AdamW [28] using a constant learning rate of 1.0×10^{-4} and stable decay (WSD) [20], gradually scaling the batch size from 64 to 256 over 800k steps.

The second stage motion planner is a 24-layer transformer with 1024-dimensional self-attention and cross-attention, trained to perform flow matching in the learned latent space. Unlike the VAE, this model processes both image tokens and latent grid tokens through shared self-attention layers, while task specifications (either Fourier-embedded pokes or text embeddings) are provided via cross-attention. The motion planner uses 2D RoPE for positional encoding, as it operates on the 2D latent grid output by the VAE. For ablation experiments, we also evaluate using 3D RoPE in the motion planner in case the latent grid has a temporal dimension. Training follows a similar schedule to the VAE, scaling from 512 to 2048 batch size over 700k steps with a constant learning rate. Both models use RMSNorm[49], SwiGLU activations [33], and an FFN expansion factor of 3, and are trained in bfloat16 mixed precision on 16 to 64 Nvidia H200 GPUs, each requiring approximately 3 days of training time.

We conduct two ablations on the model design. Fig. J demonstrates that explicitly arranging latent tokens in a grid structure leads to slightly better performance compared to treating them as an unstructured sequence. Fig. K shows the performance of the flow matching model across different numbers of function evaluations (NFEs), demonstrating that the model achieves decent performance even with as few as 10 sampling steps, making it practical for more time-sensitive applications. All ablation models use identical hyperparameters to the main models but are trained on only 4 Nvidia H200 GPUs for 24 hours to enable rapid experimentation.

C. Track Densification and Inpainting

Trajectories provide a sparse sampling of motion and do not capture every pixel in a video. However, some application require or benefit from a dense a dense motion representa-

81 frames is the minimum number of frames that this version of Wan can generate. Shorter videos are not possible.

Parameter	VAE	Motion Planner
Dataset	Open-Set Videos	Open-Set Videos
Number of clips	10M	10M
Tracker	TAPNext [52]	TAPNext [52]
Tracker positions	1024 random	1024 random
Poke scale	$[-1, 1]$	$[-1, 1]$
Batch size	$64 \rightarrow 256$	$512 \rightarrow 2048$
Optimizer	AdamW [28]	AdamW [28]
Peak LR	1.0×10^{-4}	1.0×10^{-4}
LR schedule	constant with WSD [20]	constant
Betas	$(0.9, 0.95)$	$(0.9, 0.95)$
Warm-up steps	300	300
Total Steps	800k	700k
Precision	bfloat16	bfloat16
Total Parameters	340M	530M
GPUs	$16 \rightarrow 64$ Nvidia H200	$16 \rightarrow 64$ Nvidia H200
Training Time	3 days	3 days
Depth	12 & 12	24
SA width	768	1024
CA width	768	1024
Normalization	RMSNorm [49]	RMSNorm [49]
FFN expand factor	3	3
Activation	SwiGLU [33]	SwiGLU [33]
Positional encoding	3D RoPE [12, 37]	2D RoPE [12, 37]
Image size	224×224	224×224
Image encoder	DINOv2 ViT-B/14 [31]	DINOv2 ViT-B/14 [31]
Latent width	16	16
Latent size	16×16	16×16
KL loss β	1.0×10^{-7}	-

Table F. Hyperparameters for our first stage VAE and motion planner. Ablation models use the same parameters, but were only trained on 4 Nvidia H200 GPUs for 24 hours.

tion. Our approach naturally supports track densification, as illustrated in Fig. L.

This capability arises from two key properties of our framework. First, the motion planner (Sec. 3.2) always generates a dense latent motion grid, regardless of how dense or sparse the poke conditioning is, due to our training setup. Second, our first-stage autoencoder (Sec. 3.1) can be queried at arbitrary spatial locations. Thus, by converting available tracks into pokes and conditioning the motion planner on these inputs, we can generate a dense latent grid and subsequently obtain densely inpainted tracks by querying the autoencoder at all pixel locations. This procedure enables our model to reconstruct dense motion fields from sparse tracker inputs, providing high-resolution inpainting of trajectories where needed.

D. Effects of Temporal Compression

We extend our analysis of the temporal compression factor's influence on generation performance from Fig. 4. To that end, we show motion generation performance over the course of training in two compute-matched settings in Fig. M: matching step time across models and matching batch size across models. As discussed in the main paper, the number of tokens to be denoised is inversely proportional to the temporal compression factor. To illustrate, the model with $t_c = 2$ denoises $32\times$ as many tokens as the model with $t_c = 64$, which drastically increases the compute per sample both during training and inference. Results clearly demonstrate that our strong temporal compression is beneficial for motion generation performance, while also being much more efficient at inference time (see Fig. 4).

Train \ Eval	**TapNext**	**CoTracker3**
TapNext	96.8	97.0
CoTracker3	96.3	97.3

Table G. Cross-tracker reconstruction accuracy δ^{avg}.

Ablation	$\delta^{\text{avg}} \uparrow$
Ours	96.8
+ track dropout	94.0
+ filter occlusions	93.2

Table H. Reconstruction accuracy degradation under systematic tracker degradation during training.

E. Tracker Model Ablation

To directly assess whether the learned motion space inherits tracker-specific biases, we ablate both the supervision source and its quality. Since the first stage embeds trackers into a smooth space, we report averaged reconstruction PCKs (δ^{avg}). Training with TapNext versus CoTracker3 yields near-identical reconstruction accuracy, and models generalize well when evaluated with the other tracker (Tab. G), indicating limited sensitivity to tracker choice.

We also stress-test supervision quality by degrading tracks during training. Dropping out tracks or training only on non-occluded tracks leads to graceful degradation on an evaluation split that includes occlusions (Tab. H), indicating robustness to imperfect supervision. While we acknowledge that tracker quality theoretically upper-bounds performance, our method outperforms baselines on downstream tasks (Tab. 2) and exceeds motion prediction performance of models trained with flow (Tab. 1) or RGB supervision (Tabs. 3 and 4).

F. Why CoTracker for Video Evaluation

We use CoTracker3 to obtain tracks from the generated videos in our SOTA evaluation (Sec. 4.3) because TapNext can lose tracks, resulting in missing values. This makes an unbiased challenging as it is unclear how to deal with these missing tracks. For completeness, we rerun the evaluation with tracks obtain from TapNext where we inpainting missing track values with the last observed position. This

Model	Min MSE ↓	Mean MSE ↓	EPE ↓
DAVIS [32]			
Motion I2V [34]	222.2	307.0	16.37
Ours	**155.1**	**233.0**	**0.83**
PhysicsIQ [29]			
Motion I2V [34]	177.8	225.1	12.4
Ours	**90.60**	**143.65**	**0.76**

Table I. Dense Track prediction results on Davis *(top)* and PhysicsIQ *(bottom)*. We condition both models on the maximum number of pokes from the start to the end frames.

worsens metrics for video models (Min MSE↓: Wan 29→37, Veo 3 36→48).

G. Additional Track Prediction Results

We focus on static scenes to avoid conflating detailed object motion since heavy camera motion would dominate the metrics.We further evaluate our model on the DAVIS 2017 dataset, comprising 150 videos with often significant camera motion, as well as the solid mechanics split of Physics IQ, focusing on physical understanding. In both settings, we outperform Motion I2V, our strongest direct competitor, in its strongest Dense setting (Tab. I).

H. Addional Qualitative Examples

We provide additional qualitative examples in Figs. H and I.

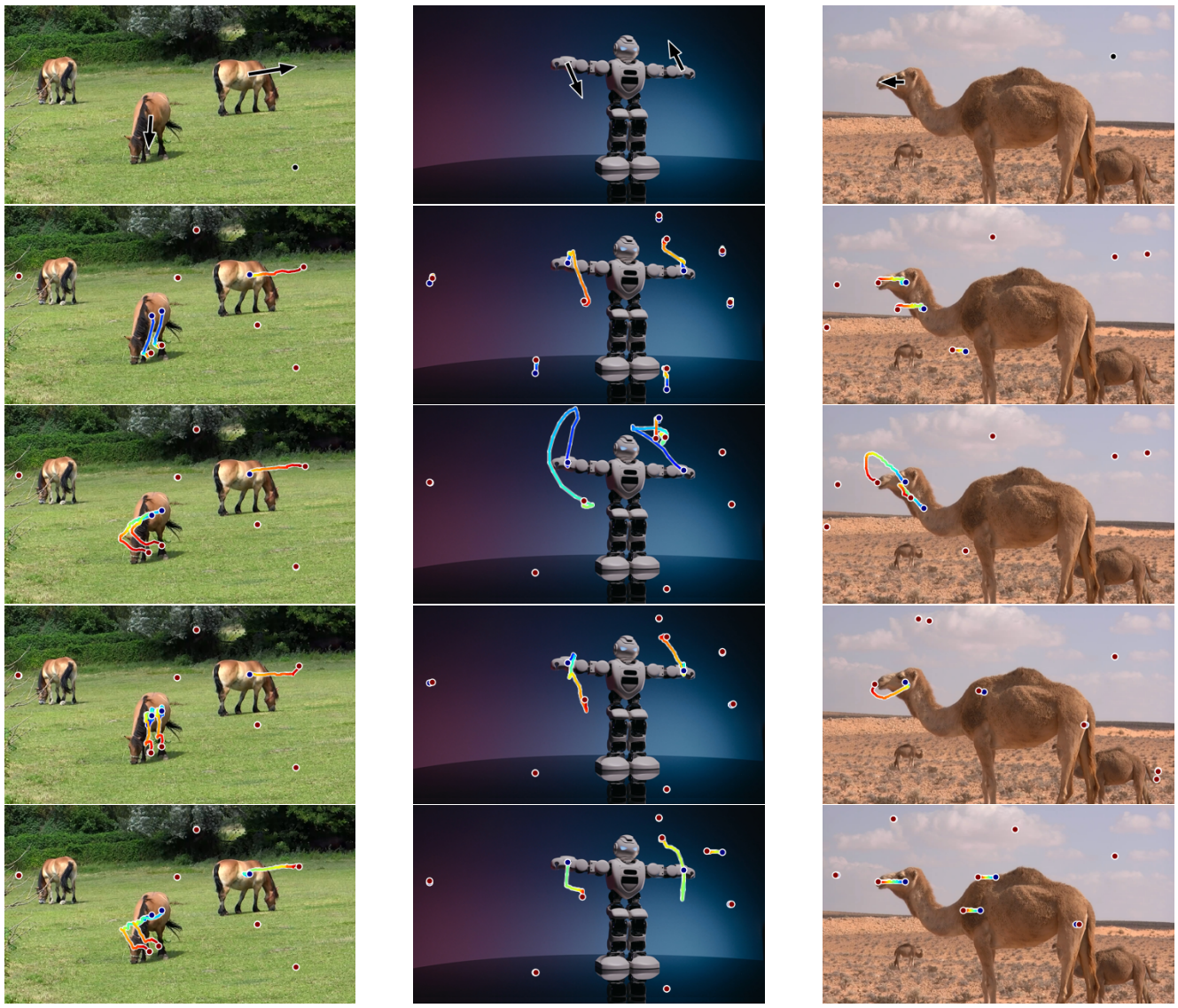
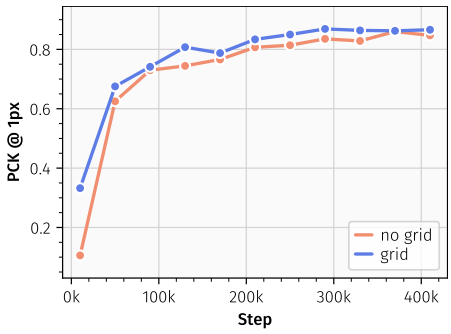
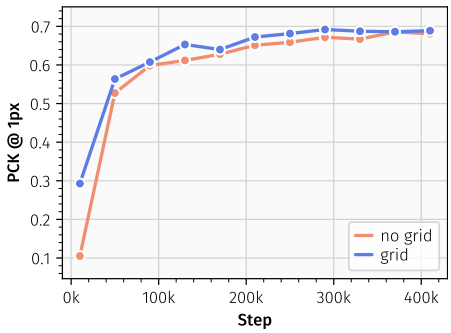


Figure H. **Additional qualitative samples.** The first image in every column is the prompt, with arrows indicating pokes, i.e., where a specific start position should end. The remaining four images are samples from our models, illustrating trajectories of randomly selected query points. The color gradient of the trajectories acts as an indicator of how fast the motion is happening. The gradient starts at blue, moves to green, yellow, and finally ends at red.

Figure I. **Additional qualitative samples.** The first image in every column is the prompt, with arrows indicating pokes, i.e., where a specific start position should end. The remaining four images are samples from our models, illustrating trajectories of randomly selected query points. The color gradient of the trajectories acts as an indicator of how fast the motion is happening. The gradient starts at blue, moves to green, yellow, and finally ends at red.



(a) AE PCK



(b) MAE PCK

Figure J. We ablate the effect of arranging latent tokens in a grid using 2D RoPE. We measure PCK using a 1 pixel threshold and report reconstruction performance for encoded positions (AE setting), as well as unseen positions (MAE setting). We find that providing positional information slightly increases performance.

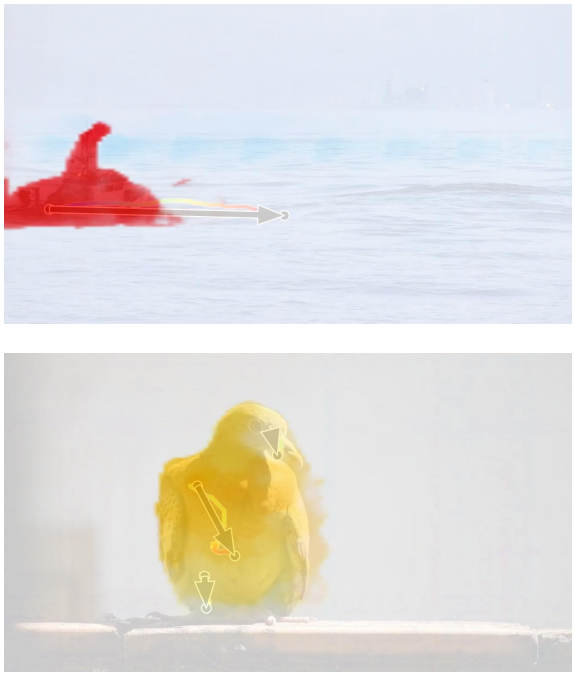


Figure L. Our model enables dense motion inference from sparse tracker inputs by converting observed tracks (pokes) into conditioning signals that specify desired endpoints in the final frame. The visualizations show dense flow toward the goal frame, illustrating how our approach infers globally coherent motion. Only the endpoint for each poke is provided; the resulting dense prediction demonstrates our model's ability to interpolate and inpaint plausible trajectories across the entire scene.

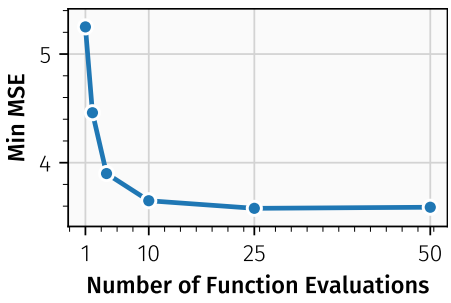
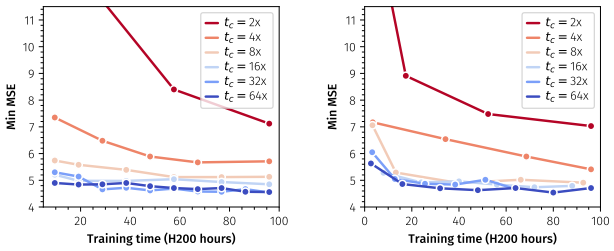


Figure K. Motion Generation performance (Min MSE) across different numbers of function evaluations (NFEs). Trajectories are densely conditioned on target positions, sampled $k = 128$ times per start frame.



(a) matched step time (b) matched batch size

Figure M. Motion generation quality over training for different temporal compression factors, measured against wall-clock time to account for varying computational cost. Matching step time (left) and batch size (right) both show that higher temporal compression yields more efficient learning and faster emergence of realistic motion.