PRIMAL: Physically Reactive and Interactive Motor Model for Avatar Learning

Supplementary Material

Contents

7. More Discussions on PRIMAL

7.1. Motivation, Contribution, and Benefits Justification

We aim to build a "motor system" for digital avatars enabling them to move perpetually and react promptly, like real humans. This goes beyond existing game engines that animate characters with canned motions, increasing realism. Inspired by recent advances in motion generation, PRIMAL generates perceptually realistic motions, generalizes across body shapes, supports efficient adaptation, and makes the avatar reactive to impulses. Our experiments and demos present its potential for game production, and show that physical simulation is not necessary to produce character animations that appear physically realistic.

The key novelty is the formulation that generates 0.5-second motion given a single initial state. This contrasts with prior work that generates a long future motion conditioned on a past motion. Its benefits include reducing overfitting, making model training easier, and making the avatar reactive to impulses and classifier-based guidance. Also, our ControlNet's advantages over existing ones are demonstrated in Sec. 4, experiments, and the SupMat video.

To better understand the benefits of our formulation, we compare two identical settings except the motion length, where 'ours' generates 15 frames given 1 frame, and 'baseline' generates 40 frames given 20 frames. We replace in-context with cross-attention to handle multi-frame conditioning. Both models are successfully overfit to a ballet sequence with 229 frames, and they can reproduce the ballet conditioned on the start frames. For testing, we first generate 780 future frames given the end frame(s) of that ballet sequence. We find 'ours' produces ballet stably, whereas 'baseline' gradually fails as time progresses. ASRs (Line475) are 0.08 ('ours') and 0.12 ('baseline'). Second, we generate 156 frames, with conditions from another walking sequence. We find 'ours' produces natural transitions to ballet, whereas 'baseline' produces severe artifacts. ASRs are 0.06 ('ours') vs 0.3 ('baseline'). These results indicate *our setting makes the model more generalizable w.r.t. motion length and semantics*. Fig. A1 shows some frames; videos are online on our website.

7.2. Relations to Hierarchical Approaches

Hierarchical approaches like CLoSD [71] and InsActor [62] first use a diffusion-based motion planner to generate a future motion, and then actuate an agent to track the generated motion in simulation. *We think this hierarchical setting is not suitable for interactive digital avatars.* First, sim-

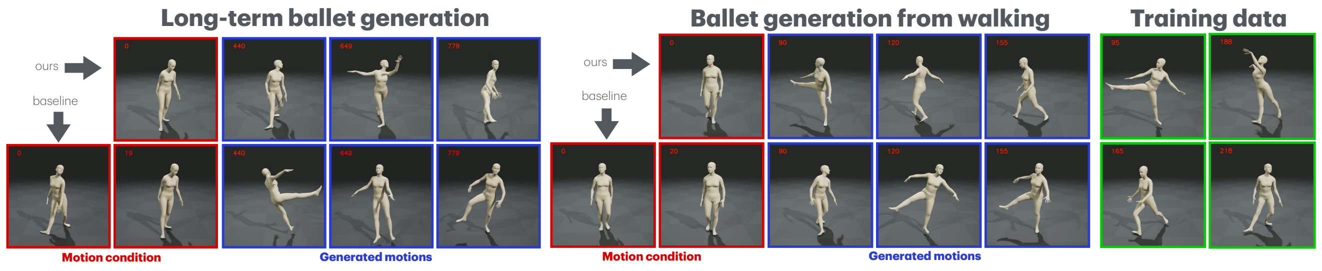


Figure A1. Our half-second atomic action setting helps generalization and transition.

ulation tends to produce less natural looking motion than diffusion-based methods, which is shown by CLoSD's BPR in Tab. 1 and the **Supplementary Video** 6:33. Second, simulation can bring extra computational costs and complicate the character animation workflow. PRIMAL only uses the diffusion model, but models much shorter motions. Unlike CLoSD (generates 40 given 20 frames) and InsActor (models more than 100 frames; see their code and SupMat), PRIMAL enables faster avatar responses.

7.3. On Physical Realism Without Simulation

We clarify that we do *not* ensure physical realism. Rather we generate *perceptually realistic* motions that look physically realistic. For example, the motion should not contain *visible* artifacts such as foot skating or ground penetration. Also, the generated motion should be human-like and aligned with user preferences. By learning purely from data, PRIMAL enables physics-like interactions with the agent (e.g. pushing and pulling). It generalizes to disruptions to the first frame that are out of distribution and responds naturally in many situations. As mentioned in the conclusion (Sec. 6), non-physical motions can occur, which could be overcome by using more training data, e.g. of people getting up from the floor.

8. More Method Demonstrations

8.1. Neural Architecture

We formulate our denoising network as $G(t, \mathbf{X}^t, \boldsymbol{x}_0)$ with transformers [55, 77], and investigate two architectures that differ in how the initial state is conditioned, as illustrated in Fig. A2. The first architecture leverages in-context conditioning, in which the diffusion time embedding and the initial state embedding are first added and then concatenated with the embedding of the noisy motion segment. We use SiLU [16, 24, 59] in the feed-forward layers. The second architecture follows the DiT network [55] with adaptive layer normalization [57].

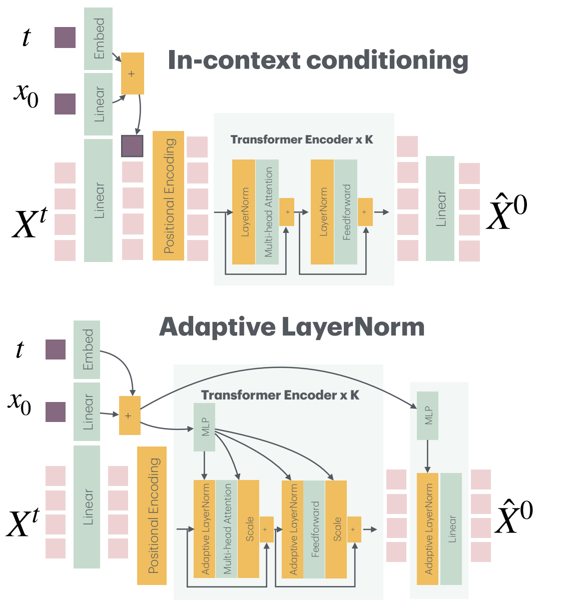


Figure A2. The diffusion network is formulated as $\hat{\boldsymbol{X}}^0 = G(t, \boldsymbol{X}^t, \boldsymbol{x}_0)$. The tea-green layers contain trainable parameters and the orange blocks are non-learnable operations. The pink squares denote the tokens at individual frames.

8.2. Inference

8.2.1. The Inference Algorithm

Our motion model learns the distribution of $p(\boldsymbol{X}|\boldsymbol{x}_0)$, following the notations in Sec. 3. Given an initial state, the model generates a motion segment starting with this initial state, and works recursively to produce an arbitrarily long motion sequence. This generation process can be refined and controlled on the fly. The overall inference approach is summarized in Alg. 1.

8.2.2. Test-time Processing

To further improve the motion quality, we propose the following three processing steps before or after generating each motion segment. Although they can be used at each individual DDPM denoising step, this introduces extra computational costs, and we did not observe advantages.

Algorithm 1: The algorithm for generating a motion segment. Long-term motions can be generated by applying this approach repeatedly.

```
Result: A new motion segment
Init: An initial state $\boldsymbol{x}_0$, guidance losses (optional);
Step 1: canonicalize the initial state;
    1.1 transform the initial state to body-centric coordinates;
    1.2 joint re-projection (optional);
Step 2: condition embedding;
Step 3: generation process;
    3.1 DDPM reverse diffusion process;
    for each denoising step do
        predict the clean motion segment;
        derive a Gaussian distribution to sample;
        compute the gradients of the guidance losses (optional);
        update the previous Gaussian distribution (optional);
        sample and get a less noisy motion segment;
    end
    3.2 snapping to ground (optional);
    3.3 inertialization blending (optional);
Step 4: transform the motion segment back to world coordinates;
```

Joint re-projection. As time progresses, the predicted joints can drift from the SMPL-X body. Inspired by [86], in each initial state we re-compute the joint locations based on the predicted SMPL-X parameters, and replace the predicted joint locations with these re-projected values. We then use this modified initial state to generate the current motion segment.

Snapping to the ground. A key aspect of motion realism is good foot-ground contact. When the model jumps, it may not fully return to the ground, resulting in foot sliding in future frames. To address this, we determine the lowest body joint over a 0.5 second motion segment. Given our hang-time assumption, in this time frame, some joint must be in contact with the ground. We then translate the entire segment along the vertical direction to "snap" the body on the ground. Since the motion segment spans roughly the hanging time of jump, this does not degrade the avatar's ability to jump.

Inertialization blending. In some cases, especially after snapping, discontinuities between the initial state and the generated motion segment can appear, leading to jittering artifacts. Therefore, we perform inertialization [8, 27] to smooth the transitions.

Perceptually, these three processing steps reliably improve the motion quality. Their quantitative evaluations are referred to Tab. A5.

8.2.3. Avatar Reactions to Induced Impulses

A sign of the emergence of physical effects is that the avatar can react to external impulses promptly and naturally. In this case, we can control the avatar with impulses to generate certain actions in a principled manner. For example, we can generate a kick by giving an upward impulse to the joints on the leg, or generate a run by pushing the avatars from the back.

action	*perturbed joints*	*perturbing velocities*
kick, left leg	left knee, left ankle	$(0, 0, 0.5)$
kick, right leg	left knee, left ankle	$(0, 0, 0.5)$
forward run	pelvis, neck, shoulders	$(0, 0, 1)$
back flip	head	$(0, 0, -1)$
forward roll	head, shoulders, elbows	$(0, -0.5, 0.5)$

Table A1. Action generation based on intuitions and straightforward principles. These perturbing velocities are in the body-centric coordinate, and are added to the mentioned joints in the initial states.

In this work, we induce impulses and forces to interact with the avatar, by intervening the initial velocities before generating a new motion segment. Based on our intuitions, we find some actions can be reliably generated, shown in Tab. A1. Their experimental results are presented in Sec. 5.1.2. Automatic solutions to discovering action principles are interesting to explore in the future.

9. More Experimental Details

9.1. Baselines for Motion Realism Evaluation

We compare our methods with two groups of baselines. The *off-the-shelf* group includes the pretrained DiP model [71] for online text-to-motion generation. We run their released code to produce 240 sequences of SMPL joint locations at 20fps, based on the test set of HumanML3D [20]. For fair comparison, we upsample the generated joint locations to 30fps via cubic interpolation and trim them to 240 frames to compute the quantitative metrics. For the perceptual study, we fit gender-neutral SMPL-X bodies to the joints via optimization, and render the videos with the same pipeline as ours. We also run the full version of CLoSD [71] based on their given test set, and obtain the SMPL joint locations. Then we process and evaluate these results as for the pretrained DiP. In addition, we run the off-the-shelf MotionLCM [13], which is a SOTA text-to-motion approach that runs in realtime. Since its base model focuses on text-to-motion and is not autoregressive, we use its default motion lengths for testing to ensure the motion quality. Likewise, we upsample the generated joint trajectories to 30fps before computing the metrics.

The *"our implemented"* group in Tab. 1 contains the DiP and the DART [89] diffusion model. Their motion representations and pretraining strategies are identical to ours. To focus on learning the body motion, we remove their text encoders but keep their key designs. Specifically, our DiP version conditions the time embedding via cross atten-

Methods	*ASR* ↓	*ARD* $\times 10^{-3}$ ↓	err. vel. ↓	err. dir. ↓
DART diffusion	0.118	3.622	2.532	1.022
DiP	0.223	4.235	1.341	0.596
InContext-8f	0.139	0.773	2.519	0.262
InContext	**0.109**	**0.252**	**1.040**	**0.140**

Table A2. The results of the CBG-based control. 'err. vel.' and 'err. dir.' denote the averaged L2 distances between the generated velocity/facing direction to their targets.

tion. Based on 20 frames in the past, it generates 40 future frames. Our DART diffusion model generates 8 future frames based on 2 past frames, suggested in [89]. Neither VAE tokenizers nor scheduled sampling are used. Their training losses are also based on Eq. (4), except that the reconstructed past motions are ignored, following their original settings. During inference, the three test-time processing steps in Sec. 3.4 (or Sec. 8.2.2) are also applied, in order to reduce artifacts such as ground interpenetration and jittering.

9.2. CBG-based Control of Speed and Direction

We evaluate the CBG method proposed in Sec. 3.4.2, which is based on intuition of the movement speed and the direction. The guidance weights of $\mathcal{L}_{move}$ and $\mathcal{L}_{facing}$ (see Sec. 3.4.2) are set to 50 and 25, respectively. We specify 4 target directions to move, i.e. (forward, left, right, back), and each direction has 2 target speeds of (1m/s, 4m/s). The target directions are applied to both moving and facing. Based on each individual initial state and target, we generate a 3-second motion, leading to 360 generated motions for each method. To evaluate the performance, we compute the L2 distance between the generated values and the target values, as well as the motion realism metrics. Considering the transition phase, we compute these L2 distances after 1 second. The results are shown in Tab. A2. We find that DART and *InContext-8f* are not responsive to the guidance on the mean velocity, probably because the average velocity in 0.25 second is highly ambiguous. The DiP model has a similar problem. Also, it tends to align the semantics between the past and the future motion, and hence increases the risk of mismatch between the motion history and the goal, leading to unrealistic movements. Fig. A3 illustrates our CBG control process. We can see that the avatar can react to the facing direction guidance within one second. See Supplemental Video.

9.3. Model Adaptation

We adapt PRIMAL-*InContext* to the tasks of semantic action generation and spatial target reaching. We evaluate two approaches: 1) We add the control embedding (i.e. the embedding of y in Fig. 3) to the time embedding, and fine-tune the pretrained model directly. This is also called as *injection* in some literature. 2) We implement the motion ControlNet proposed in [81, 90], in which the control signal is fused to the input of the ControlNet only once. An exponential moving average is also applied in the adaptation phase. We denote them as *finetuning* and *OmniControlNet*, respectively.

Methods	*ASR* ↓	*ARD* $\times 10^{-3}$ ↓	R-prec. ↑
scratch training	0.314	0.059	**0.96**
finetuning	0.253	**0.060**	0.87
OmniControlNet [81]	0.299	0.063	0.90
ours	**0.215**	0.064	**0.96**

Table A3. Results of few-shot adaptation to action generation. All ANCR results are very close to 1.

9.3.1. Personalized Action Generation

We use Mocapade3.0 [19] to capture 5 stylized action classes from video at 30fps (Fig. 6). Each class contains about 5-7 sequences of 5 seconds. We adapt the base model to this dataset using AdamW [36, 48] with a learning rate of 0.0001, and set the batch size to 16. The training terminates after 1000 epochs, leading to 19K iterations. In addition to test our mentioned baselines, we also train our model, i.e. the one used for *finetuning*, on this dataset from scratch.

To evaluate the action conditioning, we train an action label and motion feature extractor via contrastive learning [12], following the evaluation metrics of [12]. The motion encoder is a transformer with 4 blocks, 256 latent dimensions, and 8 heads. The action encoder is a single embedding layer. These encoders are trained based on our customized action data. Since our task is few-shot adaptation instead of large-scale generalization, we compute the top-1 R-precision (equivalent to classification accuracy here), and ignore the FID metric that is sensitive to the data size. We also measure the motion realism as before.

We draw 45 initial states from SFU [1], and generate a 3-second motion for each individual action. Results are shown in Tab. A3. We can see that all methods are effective. Training from scratch and our adaptation method perform best to reproduce the personalized actions, whereas our method is more realistic. Another advantage of the ControlNet-based adaptation is that it supports CFG to manipulate the intensity of the motion style. See the **Supplementary Video** for more details.

9.3.2. Spatial Target Reaching

We leverage the same training datasets with *InContext* of PRIMAL. The learning rate is fixed to 0.0001, and the batch size is set to 128. The adaptation phase terminates after 100 epochs or 6.3K iterations.

We draw 45 initial states from SFU as in Sec. 5.1.2. For each initial state, we specify 8 spatial targets w.r.t. the initial pelvis location, i.e. $(\pm1.5, 0)$, $(0, \pm1.5)$, $(\pm1, 0)$, $(0, \pm1)$ in

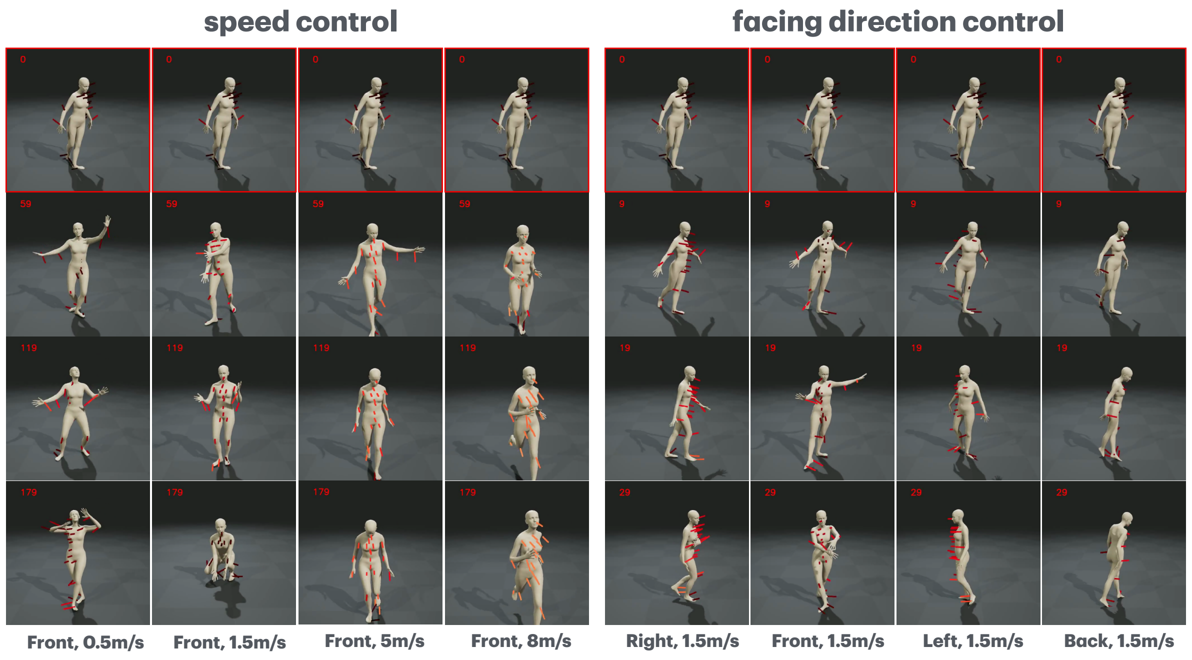


Figure A3. Illustrations of CBG-based control. Each column visualizes a motion sequence. The texts at the bottom denote the targets to achieve. The bars show the generated joint velocities, with darker/brighter color denoting smaller/larger velocity norms.

Methods	*ASR* ↓	*ARD* $\times 10^{-3}$ ↓	err. dist. ↓
DiP-goal [71]	0.3	-	0.177
finetuning	0.259	0.399	0.071
OmniControlNet [81]	**0.249**	**0.366**	0.106
ours	0.338	0.393	**0.031**

Table A4. Results of adaptation to spatial target reaching. The ANCR values of all methods are close to 1. Since SMPL-X fitting is not applied to DiP results, its ARD value is not available.

meters. Besides the above two baselines, we run the off-the-shelf DiP model [71] conditioned on the above pelvis goal locations. Each pelvis target also has 45 different motions. We zero mask the text embedding to let the model focus on the spatial target conditioning. We use this DiP model to produce sequences of joint locations, and upsample them to 30 fps via cubic interpolation.

For each target and each initial state, we generate a 2-second motion, leading to 360 sequences per method. We compute the minimal distance between the generated pelvis 2D trajectory and the target location for evaluation, as well as the motion realism metrics.

The results are shown in Tab. A4. We can find that directly finetuning is an effective approach, and has comparable performance with the two ControlNet-based methods. Compared to OmniControlNet, our adaptation method leads to a smaller distance error but slightly worse motion realism, indicating that the control signal has more impact.

9.4. Ablation Studies of PRIMAL

Tab. A5 shows the ablation study results of our PRIMAL methods, in which different model instances are separately trained from scratch. In the version of *separate tokens*, we follow CAMDM [11] to concatenate the time embedding and the initial state embedding in the context dimension instead of adding them, in which the activation function is ReLU. Since *InContext* consistently outperforms *AdaLN*, our ablation studies are mainly based on *InContext*.

Inspired by [10, 76], we add Gaussian noise with 0.01 standard error to the joint locations and velocities, but no advantages are observed in our trials. Additionally, the following approaches could not bring consistent better performances: 1) replacing SiLU with ReLU; 2) lifting the attention latent dimension from 256 to 512; 3) predicting 8 frames rather than 15 frames; 4) the *separate tokens* scheme proposed in [11].

We further investigate the influences of our test-time processing approaches. Joint re-projection can slightly increase the skating ratio and reduce the distances to the AMASS pose manifold. Inertialization can reduce skating and eliminate jittering artifacts effectively. Leveraging both of them is a good practical balance.

9.5. Default Implementations

By default, in PRIMAL the transformer has 10 blocks and the dropout ratio is 0.1. In each transformer block, the self-attention layer has the latent dimension 256 and 8

	HumanEva			SFU		
Methods	*ASR* ↓	*ANCR* ↑	*ARD* $\times 10^{-3}$ ↓	*ASR* ↓	*ANCR* ↑	*ARD* $\times 10^{-3}$ ↓
InContext	**0.017**	1.0	0.059	**0.027**	1.0	**0.081**
AdaLN	0.031	1.0	0.109	0.039	1.0	0.167
InContext w/ noise	0.020	1.0	0.089	**0.022**	1.0	0.098
InContext, ReLU	0.028	1.0	0.058	0.042	1.0	0.101
InContext, ReLU, separate tokens [11]	0.037	1.0	**0.053**	0.040	1.0	0.082
InContext, 512d	0.023	1.0	0.072	0.027	1.0	0.105
InContext, 8 frames	0.036	1.0	0.122	0.073	1.0	0.096
InContext						
no proc.	0.022	1.0	**0.055**	0.037	1.0	0.082
+ reproject	0.034	1.0	**0.055**	0.042	1.0	**0.078**
+ inertialize	**0.015**	1.0	0.069	0.028	1.0	0.089
+ reproject + inertialize	0.017	1.0	0.059	**0.027**	1.0	0.081

Table A5. Ablation studies on the base models. The initial state and every generated motion segments are snapped to the ground. The first part compares different model architectures. The second part shows the influences of the inertialization and joint re-projection.

Setting	Step 1	Step 2	Step 3	Step 4	Total
50 denoise steps	8.195	0.138	152.745	3.695	164.773
50 denoise steps + test-time proc.	8.909	0.129	152.635	3.425	165.099
50 denoise steps + test-time proc. + CBG	9.348	0.138	192.789	3.607	205.881
50 denoise steps + test-time proc. + CBG + ControlNet	10.687	0.723	349.729	3.540	364.679
10 denoise steps + test-time proc. + CBG + ControlNet	10.479	0.725	71.660	3.465	86.329

Table A6. Results of runtime analysis to generate a 0.5-second motion segment. The numbers are in millisecond. Step 1-4 are according to our inference algorithm 1.

heads. The feed-forward intermediate layer has 2048 dimensions, and its activation function is SiLU [16, 24, 59]. The DDPM process has 50 steps during training, following [11, 64, 71, 89]. We use AMASS [50] to pretrain our diffusion models [3]. All sequences are downsampled to 30fps first. During training, 15-frame motion segments are randomly extracted from a batch of sequences at each iteration. We use AdamW [36, 48] optimizer and keep the learning rate at 0.0001. The batch size is fixed to 512, and the training terminates after 30K epochs, leading to 480K iterations in total. Exponential moving average (EMA) with decay 0.999 starts to apply after the 1500-th epoch. We use a single NVIDIA H100 GPU for training, taking 3-4 days to pretrain a specific version of PRIMAL. During inference, we use the checkpoints of EMA. By default, the reverse diffusion process takes 50 steps, and the snapping-to-ground operation (see Sec. 8.2.2) is used.

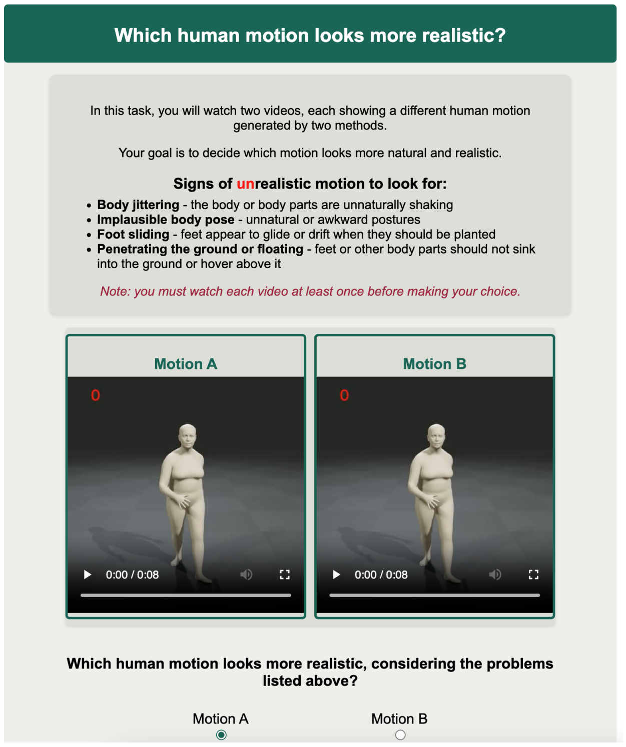
[3]Our training sets are ACCAD [2], BMLmovi [18], BMLrub [74], CMU [9], DFaust [7], Eyes Japan Dataset [49], HDM05 [51], PosePrior [3], SOMA [17], MoSh [47], SSM.

9.6. Runtime Analysis

We implement our methods with a single NVIDIA RTX 6000 GPU to measure the runtime of inference (see Algo. 1). We generate a 1200-frame motion sequence by generating motion segments recursively, then we compute the average runtime of each individual step. The results are shown in Tab. A6. We can see that all settings can produce a motion segment within 0.5 second, allowing us to stream the generated frames to the frontend (e.g. Unreal Engine) in real time. Additionally, our test-time processing is efficient. Enabling the CBG-based control brings trivial computation cost, due to their analytical gradients. Our ControlNet-based adaptor can largely slow down the diffusion, which can be overcome by reducing the denoising steps. In our trials, we find using 10 denoising steps does not bring visibly inferior results.

9.7. Perceptual User Studies

We conduct perceptual studies on the Amazon Mechanical Turk platform for both of our experiments mentioned in Sec. 5.1.1 and Sec. 5.1.2. In summary, we present users with paired results—one from our method and one from a


Which human motion looks more realistic?

In this task, you will watch two videos, each showing a different human motion generated by two methods.

Your goal is to decide which motion looks more natural and realistic.

Signs of unrealistic motion to look for:

- **Body jittering** - the body or body parts are unnaturally shaking
- **Implausible body pose** - unnatural or awkward postures
- **Foot sliding** - feet appear to glide or drift when they should be planted
- **Penetrating the ground or floating** - feet or other body parts should not sink into the ground or hover above it

Note: you must watch each video at least once before making your choice.

Motion A | Motion B

Which human motion looks more realistic, considering the problems listed above?

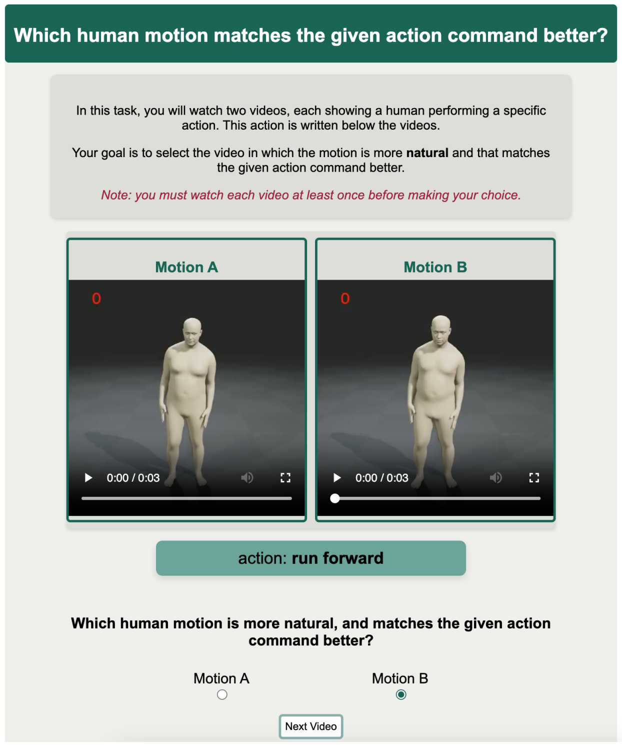
[✓ Motion A] Motion B

Figure A4. Layout of the motion realism perceptual study. Participants are presented with two videos of rendered motions and need to pick the more realistic one, according to the instructions.

baseline method. Users choose the result they prefer, and we report the percentage of cases in which the baseline is preferred. The layouts of the two perceptual studies are shown in Figs. A4 and A5.

Besides only allowing experienced and highly rated participants, we take several precautions in our study protocol to ensure reliable results. Each assignment contains 57 comparisons, i.e. pairs of videos. The first 3 are intended as warm-up tasks, and the answers to these are not considered during evaluation. There are 4 catch trials too. These are intentionally obvious comparisons that help us identify participants who are providing random inputs. We discard all submissions where even a single one of the four catch trials is failed: 6 out of a total of 55 assignments.

To eliminate bias, the order of the other 50 actual comparisons is shuffled within an assignment, and the two sides of each comparison are randomly swapped too. All methods use the same rendering pipeline.


Which human motion matches the given action command better?

In this task, you will watch two videos, each showing a human performing a specific action. This action is written below the videos.

Your goal is to select the video in which the motion is more **natural** and that matches the given action command better.

Note: you must watch each video at least once before making your choice.

Motion A | Motion B

action: **run forward**

Which human motion is more natural, and matches the given action command better?

Motion A [✓ Motion B]

Next Video

Figure A5. Layout of the action generation perceptual study. Below the instructions, participants are presented with two videos of rendered motions and a common action label.