MMM: Generative Masked Motion Model

Supplementary Material

A. Overview

The supplementary material is organized into the following sections:

- Section B: Inference speed, quality, and editability
- Section C: Codebook reset
- Section D: Influence of word embedding via cross attention
- Section E: Confidence-based masking
- Section F: Inference speed relative to motion length
- Section G: Impact of mask scheduling functions
- Section H: Impact of token sampling strategies
- Section I: Qualitative results including two new motion editing tasks: motion extrapolation/outpainting, motion completion
- Section J: Limitations

B. Inference speed, quality, and editability

To summarize the advantages of our method in three aspects—speed, quality, and editability—we compare AITS, R-Precision, FID, and Editability for MDM [35], MotionDiffuse [44], MLD [4], T2M-GPT [43], AttT2M [47] in Table 7. Only MDM [35], a diffusion model applied directly to motion space, and our method allows for editability. We denote this feature with a checkmark symbol, '✓'. It's worth noting that MotionDiffuse [44] is also capable of editability, although no specific application is provided. In the table, we represent this with a hyphen symbol, '—'. While applying the diffusion process directly to motion space provides editable capabilities for various applications, the inference time of diffusion models is not practical for real-time applications. The inference time is nearly threefold the duration of the generated motion sequence. Specifically, it takes 28.112 seconds on average to generate 196 frames, which is equivalent to a 9.8-second motion sequence. On the other hand, MLD [4] performs a diffusion process on a single motion latent space to speed up the generation time. However, the compression of its encoder not only results in the loss of fine-grained synthesis detail but also restricts its ability to edit the motion. T2M-GPT [43] and AttT2M [47] compress motion into multiple temporal embeddings, leading to faster generation time. However, due to their autoregressive approach, they lack the ability to see future tokens, which also results in a loss of editability, as indicated by the '✗'. Table 7 shows that our approach exhibits the highest quality and preserves editable capabilities while employing only 0.081 seconds on average to generate motion.

Table 7. Comparison of the inference speed and quality of generation on text-to-motion along with the editable capability of each model. '✓' means editable while '✗' is not and '—' refers to has-capability but no application provided. We calculate the Average Inference Time per Sentence (AITS) on the test set of HumanML3D [12] without model or data loading parts. All tests are performed on a single NVIDIA RTX A5000.

Methods	AITS (seconds) ↓	R-Precision Top-1 ↑	FID ↓	Edit
MDM	28.112	0.320	0.544	✓
MotionDiffuse	10.071	0.491	0.630	—
MLD	0.220	0.481	0.473	✗
T2M-GPT	0.350	0.491	0.116	✗
AttT2M	0.528	0.499	0.112	✗
MMM (our)	**0.081**	**0.515**	**0.089**	✓

C. Codebook Reset

During stage 1, the motion tokenizer learns discrete tokens as discussed in 3.1. Besides codebook factorization, we also adopt codebook reset to prevent codebook collapse, where the majority of tokens are allocated to only a few codes, while the rest of the codebook entries are inactive. As shown in our experiments, the codebook reset frequency directly affects codebook utilization and thus motion generation quality. For example, simply implementing codebook reset every training iteration impedes the codebook from learning motion tokens effectively, aggravating codebook collapse. As visually depicted in Figure 8, metrics such as FID score, R-precision, and Multi-modal Distance are initially learned during the early stages of training. However, when codebook collapse occurs, all the metrics significantly worsen. In contrast, resetting the unused codebook less frequently means the codebook will not be fully utilized, causing less fine-grain detail that the codebook can capture, which shows worse quality in 60 to 80 codebook reset iterations in Table 8.

Moreover, the results in **Stage 1: Motion Tokenizer** and **Stage 2: Conditional Masked Transformer** may not align. This is because the Stage 1 objective is the reconstruction task without text condition, while the Stage 2 objective is to generate motion conditioned by text. Therefore, even though resetting every 40 iterations leads to the best performance in the first stage, resetting the codebook every 20 iterations works best in the text-to-motion task.

Table 8. Ablation results on codebook reset every different number of training iterations. Codebook reset iteration in stage 2 indicates the pretrained models from stage 1.

Stage1: Motion Tokenizer

Codebook Reset Every Number of Training Iteration	R-Precision ↑			FID ↑	MM-Dist ↓	Diversity →	Loss ↓
	Top-1	Top-2	Top-3				
1 Iteration	Diverge (as shown in Figure 8)						
20 Iterations	0.503	0.698	0.793	0.075	3.027	9.697	**0.05156**
40 Iterations	**0.507**	**0.698**	**0.793**	**0.059**	**3.013**	**9.629**	0.05196
60 Iterations	**0.507**	**0.698**	**0.793**	0.075	3.019	9.710	0.05185
80 Iterations	0.505	0.697	0.793	0.079	3.022	9.658	0.05163

Stage2: Conditional Masked Transformer

Codebook Reset Every Number of Training Iterations (From 1st Stage)	R-Precision ↑			FID ↑	MM-Dist ↓	Diversity →	MModality ↑
	Top-1	Top-2	Top-3				
20 Iterations	**0.515**	**0.708**	**0.804**	**0.089**	**2.926**	**9.577**	**1.226**
40 Iterations	0.508	0.702	0.798	0.108	2.954	9.645	1.136
60 Iterations	0.518	0.710	0.805	0.0923	2.897	9.7163	1.157
80 Iterations	0.507	0.701	0.797	0.111	2.963	9.541	1.208

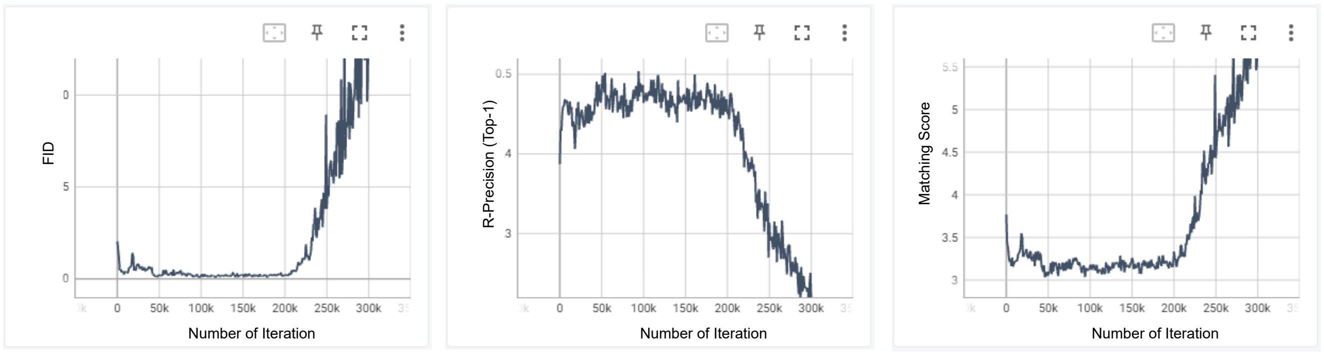

Figure 8. Codebook collapse effects in FID score, R-precision, and Multi-modal Distance metrics on the evaluation set during training using codebook reset in every training iteration.

D. Influence of Word Embedding via Cross Attention

As discussed in Section. 3.2, word embedding and sentence embedding help the model learn global and local text-to-motion relationship. We ablate the number of cross attention between word and motion embeddings to study the influence of word embedding. We replace self-attention with cross-attention at the starting layers while maintaining the same total number of 18 layers. The results presented in Table 9 indicate that a higher number of cross-attention layers leads to improved R-Precision but also worsens the FID score.

Table 9. Ablation results on the number of cross attention layers.

Number of Cross Attention Layer	R-Precision ↑			FID ↓	MM-Dist ↓	Diversity →	MModality ↑
	Top-1	Top-2	Top-3				
0 Layer	0.486	0.674	0.772	**0.082**	3.108	**9.528**	1.224
1 Layer	0.515	0.708	0.804	0.089	2.926	9.577	1.226
2 Layer	0.522	0.714	0.807	0.090	2.902	9.587	1.206
4 Layer	0.524	0.716	0.810	0.094	2.891	9.651	1.234
9 Layer	**0.527**	**0.721**	**0.816**	0.114	**2.872**	9.613	**1.325**

E. Confidence-based Masking

During the inference stage, MMM adopts confidence-based masking as shown in Figure 3. To understand the behavior of this masking strategy, we visualize the confidence levels of all sequence motion tokens in each iteration. As illustrated in Figure 9 and Figure 10, the x-axis represents the indices of temporal tokens ranging from 0 to 48 (where 49 tokens correspond to 196 frames, as 4 frames are compressed into 1 token), and the y-axis indicates the 10 iterations during the generation process. The tokens with the highest confidence are predicted and used as input conditions for the next iteration. In the initial iteration (x-axis = 0), Figure 9 suggests that the model predicts most tokens with very high confidence when it solely focuses on the text condition without being constrained by other prior tokens, as all tokens are masked. However, in the second iteration, conditioned by the highest confidence token from the first iteration, the model's confidence drops before gradually increasing in the following iterations. By the final iteration, all tokens are predicted, and no `[MASK]` tokens remain, as shown in Figure 10. Notably, we observe that the increasing confidence tends to be around the location of the previous high-confidence token and expands in the later iteration.

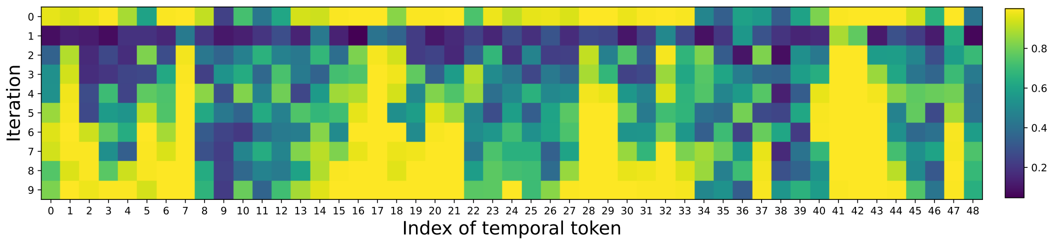

Figure 9. Visualization of the confidence in each iteration from high confidence (🟨) to low confidence(🟦)

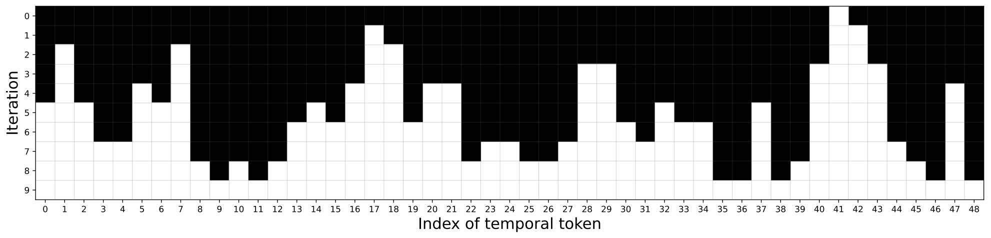

Figure 10. Visualization of mask tokens in each iteration. ■ indicates `[MASK]` tokens, and □ refers to unmasked tokens.

F. Inference Speed Relative to Motion Length

One notable advantage of our approach resides in its superior inference speed relative to the length of the generated motion. In particular, we use a dynamic masking schedule to calculate the number of `[MASK]` tokens m_t at iteration t w.r.t. maximum iteration T, the length of the motion tokens L, and the maximum number of motion tokens M that can be taken by the conditional transformer as the inputs, where M is determined by the maximum-length motion in the datasets. We compute the number of mask n_M at t iteration by a mask scheduling function as follows:

$$n_M(t) = \left\lceil \gamma\left(\frac{t}{T_{dyn}}\right) \cdot L \right\rceil . \tag{4}$$

where the total iteration of each sample, i.e., $T_{dyn} = T \cdot \frac{L}{M}$ with $L \leq M$, is proportional to its sample length L. $\gamma()$ is a decaying function. The choice of the decaying function is discussed in Section G.

As shown in Figure 11, diffusion-based approaches, MDM [35], MotionDiffuse [44], and MLD [4], exhibit the same inference time regardless of the number of generated frames. In contrast, token-based approaches such as T2M-GPT [43] and AttT2M [47] show slower inference speeds than MLD [4] for long-length sequences but faster speeds for short sequences. Our approach, on the other hand, is not only faster than all state-of-the-art approaches for long sequences but even faster for short sequence generation. Specifically, the inference speed for a 40-frame sequence can be as fast as 0.018 seconds.

This behavior offers a significant advantage for long-range generation. As discussed in Section 4, to generate long-range motion, we combine multiple short motions with transition tokens, which can be generated in parallel. Since the transitions are short and our method's inference speed is relative to the motion length, transition generation can be completed in just a single iteration, as opposed to the thousand steps required by diffusion-based models. In particular, we can generate a 10.873-minute sequence in only 1.65 seconds.

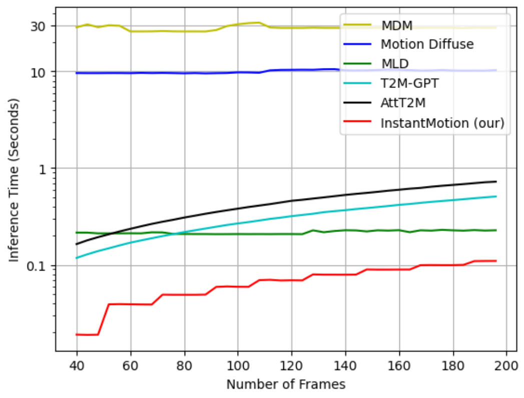

Figure 11. Comparison of Average Inference Time per Sentence (AITS) by the length of motion sequence (number of frames).

It is worth noting that the HumanML3D [12] dataset is heavily skewed towards 196-frame samples (around 1680 samples), while all other shorter samples have less than 200 samples per length. Since our method's speed is relative to the length of the generated samples, the reported average speed in Table 7 could be even significantly lower if the number of samples by length was equal. The distribution of samples by their length is visualized in Figure 12.

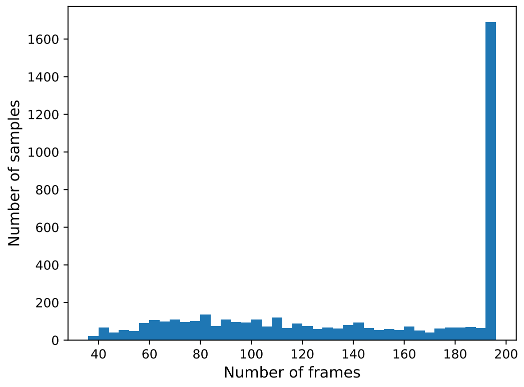

Figure 12. The number of samples in HumanML3D [12] test set by length

G. Mask Scheduling Function

As discussed in the previous section, the mask scheduling function aims to determine the number of tokens to be masked during each iteration during inference. We experiment with three mask scheduling functions to study their influence on motion generation, varying the number of iterations. Let L denote the length of generated motion and The three mask scheduling functions are described as follows:

1. **Cosine** function represents a concave dependency which can be written as $n_M(t) = L \cdot cos(\frac{1}{2}\pi \frac{t}{T_{dyn}})$
2. **Linear** function is simply an inverse relationship function of x and y: $n_M(t) = L \cdot \frac{T_{dyn}-t}{T_{dyn}}$
3. **Square Root** function is a representative convex function, expressed as $n_M(t) = L \cdot (1 - (\frac{t}{T_{dyn}})^2)$

In Table 10, we experiment with each mask scheduling function with 5, 10, 15, 20, 25, and 30 inference iterations. The results show that **Square Root** yields the worst result. **Linear** performs the best FID score at 15 iterations and can reach the best Top-1 R-precision of 0.519. While **Cosine** can achieve its best FID score of 0.089 at only 10 iterations and reach the best Top-1 R-precision at 0.518.

Table 10. Ablation results on different numbers of inference iterations with cosine, linear, and square root mask scheduling functions.

Number of Inference Iterations	R-Precision ↑			FID ↑	MM-Dist ↓	Diversity →	MModality ↑
	Top-1	Top-2	Top-3				
Cosine							
5	0.505	0.695	0.791	0.169	3.003	9.370	**1.414**
10	0.515	0.708	0.804	**0.089**	2.926	9.577	1.226
15	0.516	0.710	0.806	0.091	2.919	**9.576**	1.134
20	**0.518**	0.710	0.807	0.096	2.912	9.590	1.058
25	0.517	0.710	0.805	0.102	2.911	9.682	1.008
30	0.515	**0.711**	**0.807**	0.100	**2.908**	9.698	0.937
Linear							
5	0.505	0.693	0.789	0.195	2.995	9.399	**1.435**
10	0.514	0.707	0.804	0.090	2.920	9.584	1.190
15	0.518	0.710	0.805	**0.086**	2.910	**9.583**	1.133
20	0.519	0.713	0.808	0.091	**2.900**	9.605	1.057
25	0.517	0.713	0.808	0.096	**2.900**	9.715	1.023
30	**0.520**	**0.713**	**0.808**	0.103	2.907	9.632	0.9774
Square Root							
5	0.497	0.685	0.781	0.379	3.057	9.271	**1.511**
10	0.508	0.701	0.798	0.174	2.960	9.399	1.297
15	0.512	0.705	0.801	0.119	2.933	9.448	1.241
20	0.516	0.708	0.804	0.099	2.916	9.582	1.174
25	0.518	0.709	0.805	**0.092**	**2.909**	**9.540**	1.185
30	**0.520**	**0.712**	**0.807**	0.093	2.910	9.610	0.9815

H. Token Sampling Strategies for Parallel Decoding

After the tokens are masked out during each iteration, the masked tokens are decoded in parallel in the next iteration. The decoding process is based on stochastic sampling, where the tokens are sampled based on their prediction confidences. In particular, we investigate three sampling strategies, including temperature, top-k, and top-p, on the motion generation performance. These sampling strategies are widely adopted by natural language generation tasks, where top-p sampling generally shows superior performance compared with top-k and temperature-based sampling. However, our experiments show that the temperature-based and top-k sampling yields the best performance for the motion generation task.

Temperature Sampling. Temperature sampling generates the tokens according to the softmax distribution function shaped by a temperature parameter β, as shown below:

$$p(y_i|Y_{\bar{M}}, W) = \frac{\exp(e_i/\beta)}{\sum_{i \in E} \exp(e_i/\beta)}$$

where e_i is logits for motion code y_i from the codebook and E is the motion codebook. Low temperature gives more weight to the motion tokens with high prediction confidence. High-temperature trends to sample the tokens with equal probability.

Table 11 shows that with temperature $\beta = 1$, we can achieve the best overall generation performance measured by FID, while maintaining competitive performance for other metrics.

Table 11. Ablation results on different temperatures.

Temperature (β)	R-Precision ↑			FID ↑	MM-Dist ↓	Diversity →	MModality ↑
	Top-1	Top-2	Top-3				
.5	**0.517**	**0.714**	**0.810**	0.098	**2.893**	9.725	0.744
1	0.515	0.708	0.804	**0.089**	2.926	9.577	1.226
1.2	0.504	0.695	0.792	0.140	2.998	**9.484**	1.446
1.5	0.478	0.664	0.761	0.495	3.186	9.211	**1.884**

Top-k Sampling. Top-k sampling samples the token from the top k most probable choices. In particular, it first finds the top-k codebook entries to form a new codebook consisting of k entries, $E^k \in E$, which maximizes $\sum_{i\in E^k} p(y_i|Y_{\bar{M}}, W)$. Then, the original distribution is re-scaled to a new distribution $p'(y_i|Y_{\bar{M}}, W)$, from which the motion token is sampled. In particular, $p'(y_i|Y_{\bar{M}}, W) = p(y_i|Y_{\bar{M}}, W)/p_{sum}$, where $p_{sum} = \sum_{i\in E^k} p(y_i|Y_{\bar{M}}, W)$. "10%" refers to the sampling from only the top 10% of most probable motion tokens from the codebook entries."100%" indicates the use of all codebook entries. As shown in Table 12, $k = 100\%$ leads to the best performance, which is mathematically equivalent to temperature sampling with temperature $\beta = 1$.

Table 12. Ablation results on different probability of Top-K sampling.

Top-k %	R-Precision ↑			FID ↑	MM-Dist ↓	Diversity →	MModality ↑
	Top-1	Top-2	Top-3				
10%	0.511	0.705	0.803	**0.088**	2.930	9.636	1.180
30%	0.512	0.706	0.803	0.093	2.928	9.624	1.192
50%	0.515	0.707	0.804	0.093	**2.926**	9.637	1.208
70%	0.512	0.705	0.802	0.092	2.934	**9.506**	1.202
90%	0.514	0.706	0.803	0.091	2.927	9.594	1.159
100%	**0.515**	**0.708**	**0.804**	0.089	**2.926**	9.577	**1.226**

Top-p Sampling. The Top-P strategy, also called nucleus sampling, selects the highest probability tokens whose cumulative probability mass exceeds the pre-chosen threshold p. In particular, it first finds the top-p codebook $E^p \in E$, which is the smallest set such that $\sum_{i\in E^k} p(y_i|Y_{\bar{M}}, W) > p$. Then, the original distribution is re-scaled to a new distribution $p'(y_i|Y_{\bar{M}}, W)$, from which the motion token is sampled. In particular, $p'(y_i|Y_{\bar{M}}, W) = p(y_i|Y_{\bar{M}}, W)/p_{sum}$, where $p_{sum} = \sum_{i\in E^p} p(y_i|Y_{\bar{M}}, W)$. $p = 0.1$ refers to the sampling from only a set of probable tokens from the codebook entries whose summation of prediction confidences is larger than 0.1. $p = 1$ indicates the use of all codebook entries. As shown in Table 13, selecting only 0.1 of codebook entries can improve FID score, however, worsens R-precision, MM-Dist, and MModality.

Table 13. Ablation results on different probability of Top-P sampling.

Probability of Top-P Sampling	R-Precision ↑			FID ↑	MM-Dist ↓	Diversity →	MModality ↑
	Top-1	Top-2	Top-3				
0.1	0.511	0.704	0.802	**0.084**	2.932	**9.518**	0.060
0.3	0.516	0.711	0.807	0.093	2.909	9.589	0.480
0.5	0.516	0.711	0.807	0.091	2.905	9.622	0.739
0.7	**0.518**	**0.712**	**0.809**	0.090	**2.900**	9.645	0.963
0.9	0.516	0.710	0.806	0.089	2.910	9.636	1.079
1	0.515	0.708	0.804	0.089	2.926	9.577	**1.226**

I. Qualitative Results

I.1. Temporal Motion Editing.

In this section, we show that our method is not only able to edit the motion in-betweening task but also able to modify various temporal motion tasks such as temporal motion outpainting, temporal motion completion with text condition, and temporal motion completion without text condition, as shown in Figure 13.

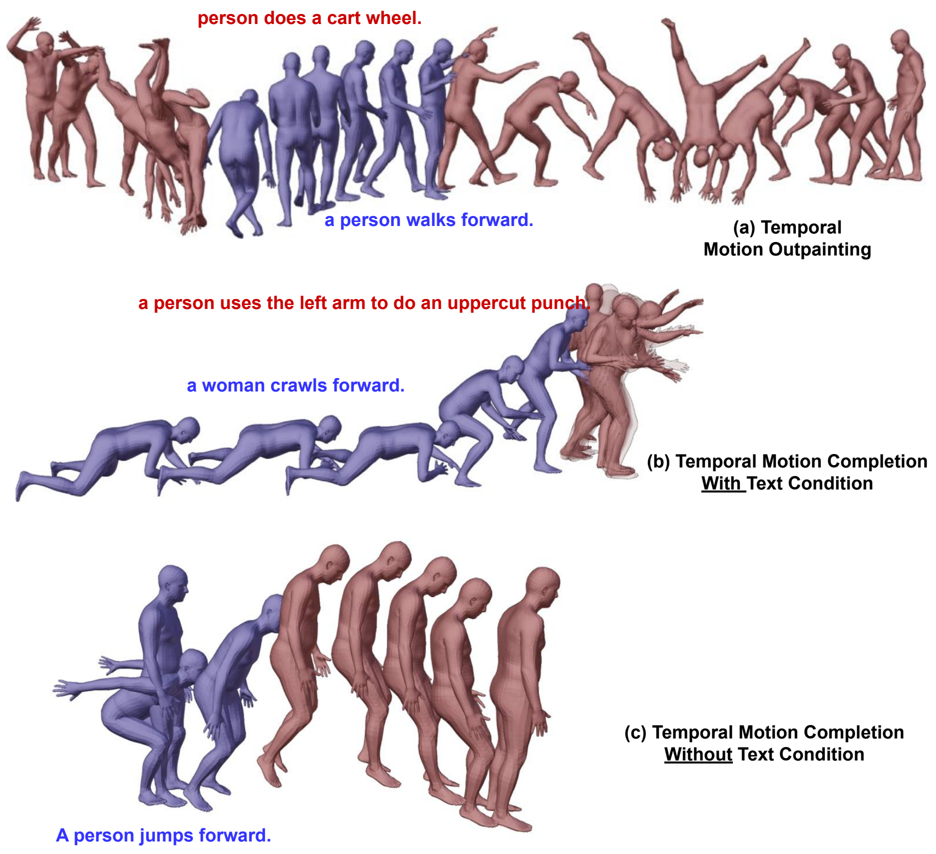
person does a cart wheel.
a person walks forward.
(a) Temporal Motion Outpainting
a person uses the left arm to do an uppercut punch.
a woman crawls forward.
(b) Temporal Motion Completion With Text Condition
(c) Temporal Motion Completion Without Text Condition
A person jumps forward.

Figure 13. Qualitative results of (a) temporal motion outpainting and (b) temporal motion completion with text condition (c) temporal motion completion without text condition where the red frames indicate generated motion and the blue frames represent conditioned frames.

I.2. Upper Body Editing

The upper body editing task combines upper and lower body parts from different prompts which can lead to the out-of-distribution issue as the joint distribution of upper and lower body parts from the prompts is not present in the dataset. We overcome this challenge by introducing [MASK] tokens into the lower body condition tokens as controllable parameters to adjust the influence of lower body parts of the generation. As illustrated in Figure 14, upper body parts are generated with text "A man punches with both hands." conditioned by lower body parts "a man rises from the ground, walks in a circle and sits back down on the ground.", showing in three cases. **Top figure** shows generated upper body parts with all masked lower body tokens which means no lower body part condition at all. The result shows uncorrelated motion between upper and lower body parts. On the other hand, **middle figure** presents the generated upper body parts with all lower part conditions without any mask. The generated motion is realistic, however, less expressive as the influence from the lower body part is too strong and the joint distribution of upper and lower body parts is unseen in the training set. To solve this problem, **the bottom figure** generates motion by conditioning only some lower body tokens. This can be achieved by applying [MASK] tokens to some of the lower body tokens. The result shows that the motion is realistic while expressing very strong corresponding motion to both upper and lower body parts.

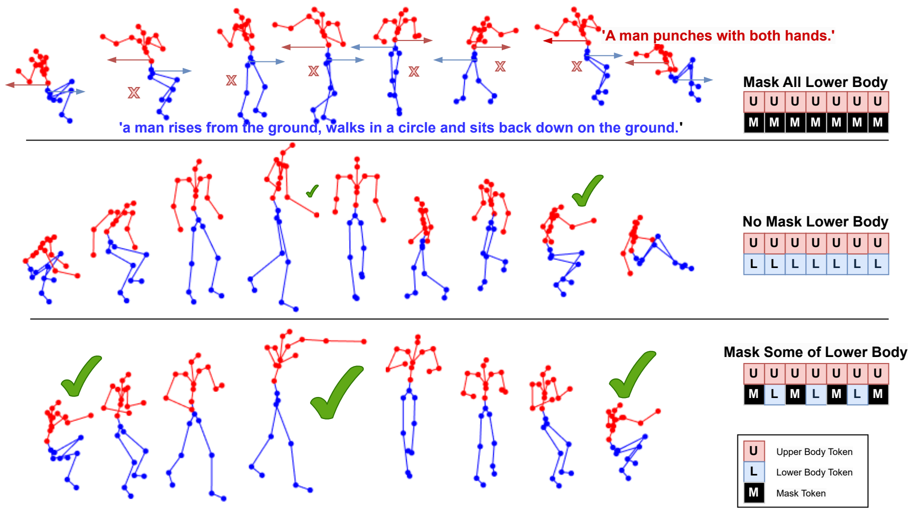

Figure 14. Effect of lower body masked tokens. "✓" shows frames that exhibit strong correlation to the textual description of the upper body part

I.3. Long Sequence Generation

In Figure 15, we generate a long motion sequence by combining multiple text prompts. First, our model generates the motion token sequence for each prompt (red frames). Then, we generate transition motion tokens (blue frames) conditioned on the end of the previous motion sequence and the start of the next motion sequence.

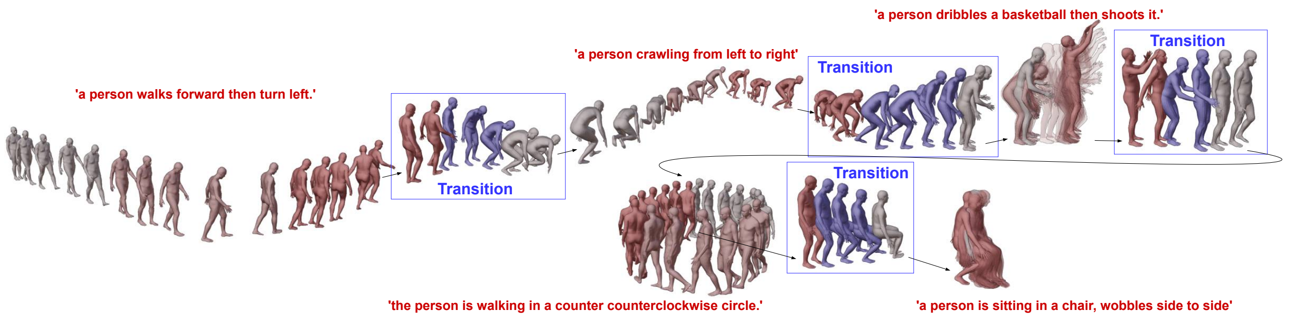
'a person walks forward then turn left.'
Transition
'a person crawling from left to right'
Transition
'a person dribbles a basketball then shoots it.'
Transition
Transition
'the person is walking in a counter counterclockwise circle.'
'a person is sitting in a chair, wobbles side to side'

Figure 15. Long Sequence Generation. An arbitrary long motion sequence is generated from multiple prompts (red frames) combined with transitions (blue frames)

J. Limitations

Our model may face challenges in rendering some fine-grain details for exceptionally long single textual descriptions. This is due to limitations in text-to-motion training datasets, which support motions up to a maximum of 196 frames. To address this, we are exploring how our model's long motion generation capabilities can be leveraged. Specifically, we aim to integrate large language models to effectively segment a lengthy texture description into several concise text prompts. Additionally, our current model does not support the generation of interactive motions involving multiple individuals. This limitation is not unique to our model but is also present in other competing methods.