

Less is More: Improving Motion Diffusion Models with Sparse Keyframes

Supplementary Material

A. Method Details

In this section, we provide additional explanations on the methodology introduced in Sec. 4. Specifically, we describe details about employed Lipschitz MLPs [4] (Sec. A.1) and Visvalingam-Whyatt algorithm [12] for keyframe reduction algorithm (Sec. A.2), respectively.

A.1. Lipschitz Regularization

Lipschitz MLP [4] regularizes a multilayer perceptron (MLP) by bounding the Lipschitz constant c (Eq. (3)). By constraining the network’s Lipschitz constant, we enforce the network to learn smoother mapping between input and output feature space. Lipschitz constant c is calculated as the product of norms of the linear layers’ weight matrices: $\prod_i \|\mathbf{W}_i\|_p$. Since this bound depends solely on trainable parameters \mathbf{W}_i , this regularization method efficiently avoids gradient vanishing problem. Authors suggest using ∞ -norm can enhance computational efficiency.

Additionally, we follow original implementation which employs weight normalization technique for further encourage robustness of the network. Specifically, the authors conduct row-wise normalization where k -th row of matrix \mathbf{W}_i is normalized through

$$\hat{\mathbf{W}}_{i,k} = \mathbf{W}_{i,k} \cdot \min(1, \frac{\text{softplus}(\|\mathbf{W}_i\|_p)}{\|\mathbf{W}_{i,k}\|_p}), \quad (5)$$

where the function $\text{softplus}(y)$ is defined as $\log(1 + e^y)$. As a result, i -th layer of Lipschitz MLP is defined as

$$g_{\theta,i}(x) = \sigma(\hat{\mathbf{W}}_i x + b_i). \quad (6)$$

The overall Lipschitz regularization loss \mathcal{L}_{lip} of the network is then computed as the product of norms,

$$\mathcal{L}_{\text{lip}} = \prod_i \text{softplus}(\|\mathbf{W}_i\|_p). \quad (7)$$

We recommend to see the further descriptions in the original paper [4].

A.2. Keyframe Reduction Algorithm

Although our method is agnostic to specific keyframe reduction algorithms, we select the Visvalingam-Whyatt algorithm [12] as our primary method. The Visvalingam-Whyatt algorithm is a line simplification method that reduces the number of vertices in a polyline while preserving its overall shape. It works by iteratively removing the vertex with the smallest *effective area*, defined as the area of the triangle formed by that vertex and its adjacent vertices. For a

given vertex v_i with neighbors v_{i-1} and v_{i+1} (coordinates (x_{i-1}, y_{i-1}) , (x_i, y_i) , and (x_{i+1}, y_{i+1})), the effective area A_i is computed as

$$A_i = \frac{1}{2} \left| \det \begin{pmatrix} x_{i-1} & y_{i-1} & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{pmatrix} \right|. \quad (8)$$

which equals the area of triangle $\triangle(v_{i-1}, v_i, v_{i+1})$. After running this algorithm until the end, we obtain a priority list of indices, where the higher priority means the latter the point is removed. By cropping certain length from the start, we can get most reliable points that can effectively represent the input geometry. Here, the reduction rate determines the length to crop, *i.e.* the number of keyframes. We find 60% to 80% of reduction rate often extracts plausible sets of keyframes for motions with 20 Frames per second (FPS). However, we note that the optimal values can be different depending on the dataset.

To apply the Visvalingam-Whyatt algorithm to keyframe selection, we first define a suitable feature for each frame. Through empirical observation, we find that using joint positions in the root coordinate space yields effective results. Specifically, for the SMPL-based HumanML3D dataset [2], we compute local positions $\mathbf{p}_j \in \mathbb{R}^3$ for all $J = 23$ joints. Additionally, we append an additional dimension for the frame index, which leads extracting evenly distributed keyframes along temporal axis. Consequently, each frame is represented as a 64-dimensional vector, and we run the algorithm directly in this 64-dimensional space to identify keyframes.

B. Implementation Details

For reproducibility, we additionally provide detailed explanations on the implementation for Sparse Motion Diffusion Model (sMDM, Sec. B.1) and Sparse Diffusion Planner (sDiP, Sec. B.2), respectively.

B.1. Sparse Motion Diffusion Model (sMDM)

We leverage the Transformer-based architecture [11] from the baseline MDM [9]. This implementation employs Transformer layers from PyTorch [6], and provides options between Transformer encoder and decoder for denoising network. Following the original implementation, we use 8 layers of Transformer layers, while employing 4 heads for multi-head attention structure. Each transformer layer introduces Dropout layers [8] with 0.1 of dropout probability. Also, we allocate 512 dimension for the intermediate features space, *i.e.* using 512-dimensional vector for latent representation. For additional robustness, we use customized

self-attention layer, where key feature are quantized through finite scalar quantization (FSQ) [5]. Unlike to regular vector quantization [10], this scalar quantization does not employ additional loss terms. Lastly, we exclude Lipschitz regularization as the model without it produces better result.

B.2. Sparse Diffusion Planner (sDiP)

We also follow the official codebase of Diffusion Planner (DiP). As DiP largely borrows model structure from MDM [9], we use the same hyperparameters in Sec. B.1 for architectural design. Unlike regular MDM, DiP aims to generate motions in a real-time controller scenario, which demands fast inference time. Following original implementation, we train this model with 10 diffusion steps, which employing 20 frames (1 sec.) for the past trajectory while 40 frames (2 sec.) for future trajectory (denoising target). Also, for the *target-conditioning* example, we employ a linear layer with a Sigmoid Linear Unit (SiLU) activation to encode multiple conditioning joint trajectories. As demonstrated in Table 2, we exclude *dynamic mask update* during the inference, as this strategy is not much effective with the model with the small diffusion step.

C. Additional Results

C.1. Different Transformer Layers

We evaluate our approach’s generalizability across different Transformer architectures. Although our method is architecture-agnostic, we specifically validate performance improvements on Transformer encoder layers. To directly measure our method’s impact, we retrain the baseline MDM from scratch rather than using the pre-trained model. Results are shown in Table 5.

Method	R-Precision \uparrow			FID \downarrow	MM-Dist \downarrow	Div \rightarrow
	Top1	Top2	Top3			
Real	0.511	0.703	0.797	0.002	2.794	9.503
MDM (enc)	0.463	0.656	0.755	0.495	3.206	10.050
sMDM (enc)	0.503	0.697	0.793	0.284	3.007	9.955

Table 5. Text-to-motion results of MDM [9] and sMDM, implemented with Transformer encoder, on HumanML3D dataset [2]. Both models are trained with 50 diffusion steps. We use the same evaluation metrics in Table 1.

As shown in Table 5, our method consistently outperforms the baseline, confirming that our approach effectively generalizes across different Transformer architectures.

We further validate the generality of our method by applying it to MotionDiffuse[13], namely *Sparse MotionDiffuse* (*sMotionDiffuse*). To ensure fairness, we directly modified the original MotionDiffuse codebase and retrained the model from scratch. Similar to sMDM, we apply an 80%

reduction rate for keyframe selection, keeping all other settings unchanged. During implementation, we corrected an error in the original diffusion loss term, where invalid frames were not properly masked. Table 6 shows quantitative improvements after integrating our sparse keyframe approach.

Method	R-Precision \uparrow			FID \downarrow	MM-Dist \downarrow	Div \rightarrow
	Top1	Top2	Top3			
Real	0.511	0.703	0.797	0.002	2.794	9.503
MotionDiffuse	0.459	0.648	0.753	1.096	3.262	9.073
sMotionDiffuse	0.483	0.674	0.776	0.698	3.110	9.375

Table 6. Text-to-motion results of MotionDiffuse and sMotionDiffuse, implemented with Transformer encoder, on HumanML3D dataset [2].

Results indicate consistent improvements across all evaluation metrics, demonstrating that our approach is broadly applicable to Transformer-based motion diffusion frameworks, independent of the specific architectural details.

C.2. Different Interpolation Methods

As described in Section 4.1, we use linear interpolation to reconstruct the full sequence in the *feature space*. This guides the denoising network with full-length ground-truth motion and improves text-to-motion quality, as verified in our ablation study (Table 1). Our batched linear interpolation is fast, differentiable, and easily integrated into the training pipeline. Although interpolating directly in the *raw motion space* can be an alternative, we observe that it sometimes fails to recover detailed motions, likely due to the complex, nonlinear nature of raw motions. To test further interpolation option, we replace a linear interpolation with differentiable *cubic-Hermite spline* interpolation and report the results below.

Method	R-Precision \uparrow			FID \downarrow	MM-Dist \downarrow	Div \rightarrow
	Top1	Top2	Top3			
DiP	0.457	0.670	0.781	0.214	3.187	9.409
sDiP (linear)	0.469	0.677	0.789	0.186	3.130	9.419
sDiP (spline)	<u>0.466</u>	<u>0.676</u>	<u>0.786</u>	<u>0.196</u>	<u>3.146</u>	9.427

Table 7. Text-to-motion results using Diffusion Planner (DiP) with different interpolation methods.

As shown in the table, our approach consistently outperforms the baseline regardless of interpolation method. Interestingly, simple linear interpolation slightly outperforms cubic-Hermite spline. We attribute this to the fact that overly flexible spline interpolation may cause subtle distortions in the feature space, whereas linear interpolation maintains a more stable structure. This suggests that, de-

spite its simplicity, linear interpolation is highly effective for our setting.

C.3. U-Net based Architecture

We also present preliminary results to assess the generalization of our approach to motion diffusion models with U-Net-based architectures [7].

Although initial motion diffusion models, such as MDM [9] and subsequent works, primarily utilized Transformer-based architectures, recent studies have explored U-Net alternatives. Notably, Guided Motion Diffusion (GMD) [3] showed that U-Net architectures are particularly effective for motion representations involving absolute root coordinates. Specifically, GMD highlights the advantages of using U-Net when editing motion data with the conditioning trajectories in a global coordinates. Inspired by GMD, CondMDI [1] modify the suggested model with imputation technique in order to conduct flexible motion in-betweening.

Since CondMDI demonstrates the state-of-the-art performance with U-Net structure, we choose CondMDI as the baseline to test our approach. We summarize three components where CondMDI mainly differs from MDM: (1) *Architecture of Denoising Network*. Although both architectures employ attention layers, U-Net in CondMDI can not exclusively use sparse keyframes during the calculation. This is because U-Net heavily relies on 1D convolutional layers along temporal axis, which assumes dense sequence as input. (2) *Data Representation*. CondMDI uses modified representation of HumanML3D dataset [2]. It uses absolute positions rather than localized velocities for root representation. (3) *Imputation*. As CondMDI presents motion diffusion model tailored to motion in-betweening task, it explicitly employs imputation process during the training. In contrast to MDM which uses noised motions \mathbf{x}_t as input, CondMDI imputates noised frames with the clean input \mathbf{x}_0 , resulting

$$\hat{\mathbf{x}}_t = \mathbf{M} \cdot \mathbf{x}_0 + (1 - \mathbf{M}) \cdot \mathbf{x}_t, \quad (9)$$

where \mathbf{M} corresponds to the keyframe mask. Different from our approach, the authors of CondMDI propose random selections to create a keyframe mask during the training.

As U-Net with 1D convolution cannot fundamentally accept sparse frames as input, our sparse version, namely *sCondMDI*, simply replaces uninformative keyframes with the learnable embeddings. Also, we employ similar geometric losses [9] that are originally introduced to prevent foot skating or jerk. For keyframe selection, we use keyframes selected from Visvalingam-Whyatt algorithm [12] with dynamic reduction rate in a range of [90%, 95%]. We evaluate the motion in-betweening, which measures generation quality given 5 conditioning keyframes, in Table 8.

Method	R-Pre \uparrow	FID \downarrow	MM-Dist \downarrow	Div \rightarrow	Keyframe Error \downarrow	Skating Ratio \downarrow
Real	0.797	0.002	2.794	9.503	-	-
CondMDI	0.669	0.153	<u>5.127</u>	9.457	0.081	0.067
w/o geometric	0.669	<u>0.322</u>	5.157	9.031	<u>0.096</u>	0.098
sCondMDI	0.667	0.551	5.059	<u>9.075</u>	0.218	<u>0.082</u>

Table 8. Motion in-betweening results of CondMDI and sCondMDI. For evaluation, we sample keyframes from test split of HumanML3D dataset [2] using Visvalingam-Whyatt [12] algorithm.

Along with the evaluation metrics for generation (R-Precision (Top 3), FID, Multi-Modal Distance, Diversity), we additionally measure the keyframe errors and skating ratio, following the original work. As demonstrated in the table, our approach exhibits degraded performance compared to the baseline. We attribute this that our preliminary modification cannot focus on the sparse keyframes. Future work could explore further architectural modifications to more naturally accommodate sparsity.

C.4. Runtime Efficiency

Our method is computationally efficient since only a subset of frames is processed by the self-attention layers, reducing computation time during the inference. This efficiency gain is due to our use of uniform keyframe intervals at inference, eliminating the need for masking operations. Interpolation is performed only at the final diffusion step during inference. During training, we use a masking strategy for loss function, which maintains computational complexity similar to standard approaches. We report inference time comparisons for MDM (50 diffusion steps) and DiP (10 diffusion steps) when generating a batch of 64 samples (RTX 4090).

	MDM	sMDM (ours)	DiP	sDiP (ours)
Time (s)	1.937	0.725	0.174	0.164

Table 9. Inference time during the generation of 64 samples.

With the same setting, we also find that our approach reduces GPU memory usage by 230 MB and 45 MB for sMDM and sDiP, respectively. Although dynamic inference (Section 4.2) increases inference time due to the keyframe reduction algorithm, it is an optional technique mainly for models with large diffusion steps. Models with fewer steps do not require it, so our approach maintains its inference speed advantage.

C.5. User Study

We conducted a user study where participants rated 12 samples with Likert scale (1-5) based on three criteria: (1) *Naturalness* (smoothness and realism), (2) *Text-Motion Alignment* (accuracy to the prompt), and (3) *Expressiveness* (rich-

ness and vividness). We collected 35 responses, and the averaged scores are summarized below. A number inside the parenthesis indicates standard deviation.

Method	Naturalness \uparrow	Alignment \uparrow	Expressiveness \uparrow
MDM	3.14 (1.18)	2.89 (1.21)	2.81 (1.14)
MotionGPT	3.10 (1.25)	2.90 (1.42)	3.36 (1.07)
sMDM (ours)	3.64 (1.17)	4.22 (0.94)	3.92 (0.95)

Table 10. Results of user study.

The results are consistent with the quantitative evaluations (Table 1), where our model significantly improves upon baseline MDM across the conventional metrics. Our model is particularly effective at generating vivid and expressive motions, such as dance. We attribute this capability to the use of keyframes, which can efficiently summarize motion dynamics and allow the model to learn complex patterns more effectively. Please refer to supplementary videos.

References

- [1] Setareh Cohan, Guy Tevet, Daniele Reda, Xue Bin Peng, and Michiel van de Panne. Flexible motion in-betweening with diffusion models. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–9, 2024. 3
- [2] Chuan Guo, Shihao Zou, Xinxin Zuo, Sen Wang, Wei Ji, Xingyu Li, and Li Cheng. Generating diverse and natural 3d human motions from text. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5152–5161, 2022. 1, 2, 3
- [3] Korrawe Karunratanakul, Konpat Preechakul, Supasorn Suwajanakorn, and Siyu Tang. Guided motion diffusion for controllable human motion synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2151–2162, 2023. 3
- [4] Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. Learning smooth neural functions via lipschitz regularization. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–13, 2022. 1
- [5] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: Vq-vae made simple. *arXiv preprint arXiv:2309.15505*, 2023. 2
- [6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 1
- [7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention—MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015. 3
- [8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 1
- [9] Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Daniel Cohen-Or, and Amit H Bermano. Human motion diffusion model. *arXiv preprint arXiv:2209.14916*, 2022. 1, 2, 3
- [10] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 2
- [11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1
- [12] M Visvalingam and JD Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993. 1, 3
- [13] Mingyuan Zhang, Zhongang Cai, Liang Pan, Fangzhou Hong, Xinying Guo, Lei Yang, and Ziwei Liu. Motiandiffuse: Text-driven human motion generation with diffusion model. *arXiv preprint arXiv:2208.15001*, 2022. 2