ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2026
D. Levin and M. Chu
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 8

F3AMD: Fast FiLM-Conditioned Fourier Autoregressive Motion Diffusion

Calvin Z. Qiao [1,2†] Benjamin MacAdam[2†] Mohammadarsh Khokhar[1,2] Pranav Balaji[2] Jiaqing Hu[1] KangKang Yin[1]

[1]Simon Fraser University [2]CD PROJEKT RED

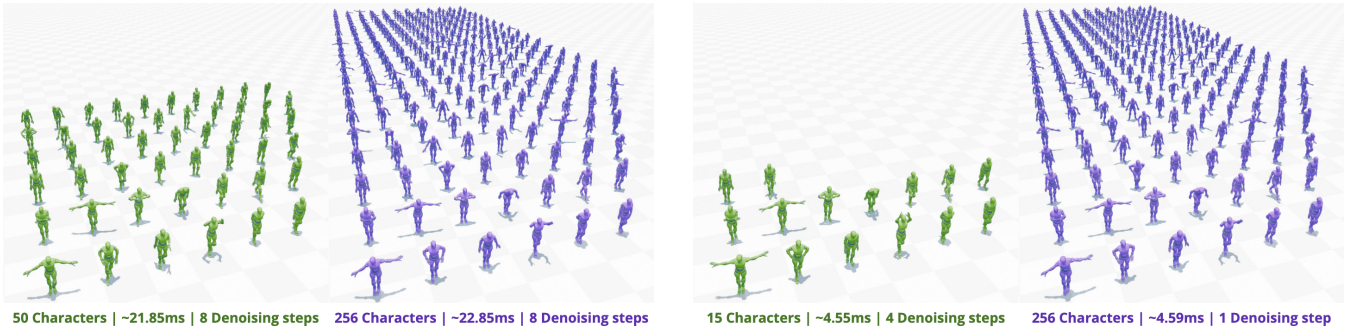


Figure 1: *We introduce F3AMD, an autoregressive diffusion framework optimized for motion synthesis for large crowds. We compare GPU runtime performance for the baseline CAMDM (green) and the proposed F3AMD-FNO-96 (purple). Left: With 8 denoising steps, both models achieve comparable visual quality. Right: F3AMD models can still produce high-quality motions with just a single denoising step, whereas CAMDM requires at least 4 denoising steps to maintain acceptable quality*

Abstract

Recent advances in generative motion synthesis have enabled realtime autoregressive generation of diverse and realistic character animations conditioned on user inputs, as demonstrated by models such as the Conditional Autoregressive Motion Diffusion Model (CAMDM). However, real-world applications (e.g., computer games) often demand faster-than-realtime performance for large numbers of characters. We introduce F3AMD (Fast FiLM-conditioned Fourier Autoregressive Motion Diffusion), a framework that achieves an order of magnitude speedup over state-of-the-art systems for multi-character animation on both GPUs and CPUs while maintaining high motion quality. Our key insight is that autoregressive motion diffusion is primarily bottlenecked by architectural and sampling inefficiencies. To address this, F3AMD employs Fourier Neural Operators (FNOs) as encoder-decoder modules, substitutes Transformer backbones with FNO blocks, replaces condition concatenation with lightweight Feature-wise Linear Modulation (FiLM), and adopts a variance-exploding noise schedule with a deterministic sampler. This design enables a substantially lower-dimensional latent space, facilitates learning in both the spectral and temporal domains, and significantly improves sample efficiency. We conduct systematic ablations of key design factors, including latent dimension, backbone type, diffusion window length, and number of denoising steps. Our recommended configuration, F3AMD-FNO-96, achieves 20x speedup over the baseline CAMDM model, while maintaining comparable motion quality.

CCS Concepts

• ***Computing methodologies*** → ***Procedural animation; Motion processing;***

† Equal contribution and corresponding authors:
zhuhanq@sfu.ca, ben.macadam@cdprojektred.com.

1. Introduction

Online human motion generation is a fundamental problem in animation, gaming, and extended reality. Recent diffusion-based methods build upon the Motion Diffusion Model (MDM) framework [TRG*23, HPD*24, VWLF*24, ZLT25, PSW*24, SWJ*24, CSH*24, LQR*24, YTY*23, MXP*25, ZCP*22], which can synthesis human motions from multi-modal user inputs. The state-of-the-art Conditional Autoregressive Motion Diffusion Model (CAMDM) further enables real-time, long-horizon motion synthesis of high quality. However, due to its Transformer backbone [VSP*17] and specific conditioning and sampling mechanism, CAMDM remains limited in scaling to large-crowd animation for super–real-time applications such as gaming.

To address these limitations, we propose a novel motion diffusion framework **F3AMD** (**F**ast **F**iLM-conditioned **F**ourier **A**utoregressive **M**otion **D**iffusion), which achieves an order-of-magnitude speedup with comparable quality to CAMDM. F3AMD introduces several key innovations. Instead of concatenating conditioning signals with motion features, we employ Feature-wise Linear Modulation (FiLM) to inject contextual information through channel-wise scaling and shifting [PSDV*18]. We further replace conventional linear encoder–decoder layers and Transformer backbones with Fourier Neural Operators (FNOs), which effectively capture long-range temporal dependencies by learning global interactions through spectral convolution in the frequency domain [LKA*21]. Together, FiLM and FNOs enable operation in a compact latent space, substantially improving performance and scalability without compromising motion quality. In addition, we replace the standard variance-preserving (VP) noise schedule and stochastic sampling in MDMs with a variance-exploding (VE) schedule and deterministic DDIM (Denoising Diffusion Implicit Models) sampler for more efficient inference.

In summary, our contributions are:

1. We introduce F3AMD, a novel autoregressive motion diffusion framework for fast and high quality character animation that utilizes FNO backbone and encoder/decoder, and FiLM conditioning.
2. We employ a VE noise schedule with the DDIM sampler to further improve sample efficiency.
3. We systematically evaluate the effect of various design choices on model performance and motion quality, including the backbone type, condition injection mechanism, encoder/decoder method, latent dimensionality, diffusion window length, and denoising step count. We show that our best model F3AMD-FNO-96 can achieve up to **20**× speedup in synthesis on both CPU and GPU, through two multiplicative factors: a ∼4x gain from the new F3AMD architecture, and a ∼5x gain from reducing the number of denoising steps from 8 to 1, while maintaining high motion quality.

2. Related Work

2.1. Conditional Autoregressive Motion Diffusion

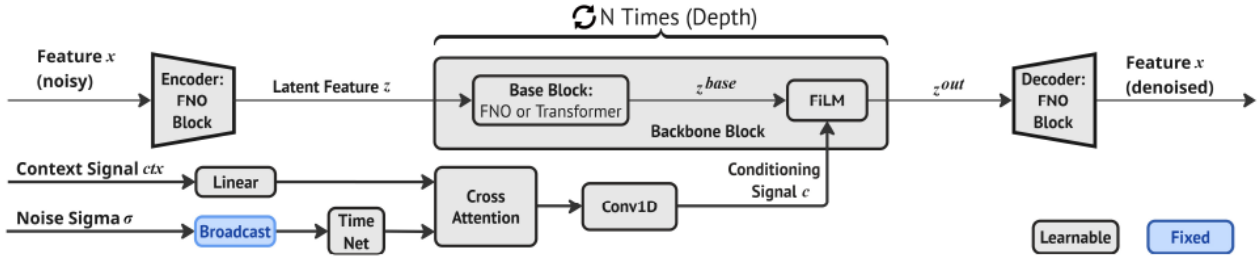
Early motion diffusion models focus on fixed-length motion synthesis from text or action labels [TRG*23, ZCP*22, STKB24], which are not suitable for real-time online motion synthesis under continuous user control. Conditional Autoregressive Motion Diffusion Models (CAMDM) address this limitation by iteratively synthesizing short motion segments (e.g., 16–48 frames) conditioned on past motion history and optional user inputs. Recent variants include DART [ZLT25], AMDM [SWJ*24], AAMDM [LQR*24], and CAMDM [CSH*24]. Among these, AMDM and AAMDM employ MLP-based backbones, which enable high responsiveness but tend to lose temporal coherence over long motion rollouts. Transformer-based architectures such as DART and CAMDM leverage extended temporal contexts to produce more stable motion. These models typically concatenate conditional inputs, such as root trajectory or style embeddings, with motion tokens along the temporal axis [CSH*24, HPD*24, LQR*24]. While Transformer-based systems are capable of animating one or a few characters in realtime, they remain insufficient for animating large crowds in super-real-time applications such as gaming.

Existing CAMDM models typically adopt the VP noise schedule with a DDPM (Denoising Diffusion Probabilistic Model) sampler [HJA20], which we find leads to step-sensitive denoising and reduced sampling efficiency. In contrast, we employ a VE schedule [SSDK*21, KAAL22, KAL*24], which, when combined with deterministic ODE-based samplers such as DDIM (Denoising Diffusion Implicit Models) [SME21], enables efficient sampling with far fewer denoising steps. This arises because VE training leads to more consistent denoising behavior across noise levels, allowing DDIM to track a stable deterministic trajectory and produce high-quality samples, even in a single step.

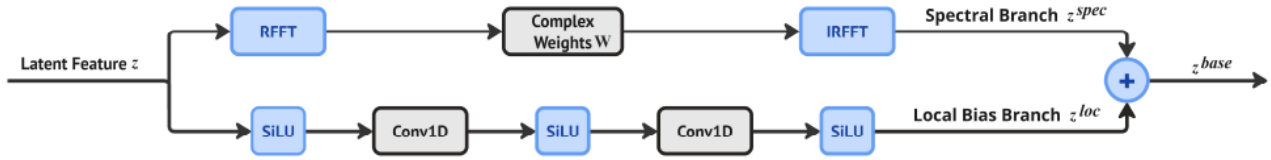
Another popular choice for super-real-time character animation is motion matching [Cla16, PAAP25]. These methods excel in motion quality and stability by retrieving and blending high-quality motion clips from large databases. In contrast, autoregressive generative models offer a more flexible framework for integrating diverse control signals and can synthesize a broader range of motions beyond direct database retrieval.

2.2. Frequency Related Animation Methods

Animation methods have leveraged frequency analysis and the inherent periodicity within human motion before. Early approaches, such as PFNN [HKS17], parameterized network weights using a phase variable synchronized with the gait cycle, enabling stable and controllable locomotion. Later this approach has been extended to model multi-modal and non-periodic motions [SZKZ20, ZSKS18, SZKS19]. DeepPhase [SMK22] integrated these ideas by learning a continuous phase manifold that produces compact, phase-aware latent codes for fine-grained motion control. This phase manifold-based formulation has since been used for motion in-betweening [SSKS23], phase-conditioned 3D pose estimation [SSY*23], and text-driven frequency-domain motion diffusion models [WHW*23]. Kry et al. [Kry12] introduced a spectral formulation based on modal analysis, decomposing articulated motion into oscillatory vibration modes where lower frequencies capture global movements and higher frequencies encode local details. These systems often employ customized architectures or pipelines that do not generalize easily, whereas FNOs offer greater flexibil-



(a) *The F3AMD architecture replaces linear encoder–decoder layers with FNO blocks. The backbone block is repeated N times, each consisting of a base block and a FiLM layer. A base block can be either an FNO or a Transformer encoder block. The FiLM layer injects condition signal by affine modulations.*



(b) *An FNO block fuses a spectral branch with a local bias branch. The spectral branch transforms features into the frequency domain via real FFT (RFFT), applies complex weights, and reconstructs the signal via inverse real FFT (IRFFT). The local branch consists of two 1D-CNNs capturing fine-grained details.*

Figure 2: *F3AMD Architecture: (a) the overall architecture. (b) the FNO block diagram.*

ity as plug-and-play layers or modules within larger neural network models.

2.3. Fourier Neural Operators

Fourier Neural Operators are a class of neural operators that learn mappings between continuous function spaces [LKA*21, KLL*23]. The FNO uses a spectral convolution to capture global features: a Real Fast Fourier Transform (RFFT) to transform the data to the frequency domain, then a learned complex transformation to each frequency feature and an Inverse Real Fast Fourier Transform (IRFFT) to return to the temporal domain. This is then summed with a local bias branch, which uses a standard convolutional neural network to capture fine-details of temporal features.

Originally developed for solving partial differential equations, this approach has achieved remarkable acceleration in large-scale physical simulations such as Darcy flow [HW97]. While FNOs have recently been explored as backbones for diffusion models [LT25, DCW25], their integration into motion diffusion models remains underexplored in visual computing. In this work, we combine FNOs with a lightweight conditioning mechanism FiLM, yielding a highly efficient diffusion framework for multi-character motion synthesis.

3. Methods

3.1. Diffusion Framework

A neural network G is trained to predict the clean motion feature x_0 from its noisy version x_t based on the noise level $\sigma(t)$ and a context signal ctx:

$$\hat{x}_0 = G(x_t, \phi(\sigma(t)), C(ctx)) \tag{1}$$

Here, $\sigma(t)$ is the noise level, $\phi(\cdot)$ embeds it into a higher-dimensional representation, and $C(\cdot)$ encodes the context. At inference, the model iteratively denoises noise to generate the next motion segment, using each newly produced segment as context for the next in an autoregressive manner.

3.2. Feature Representation

We use the 6D joint rotation representation to avoid discontinuities that may cause instability during learning [ZBL*19]. For the root joint, 3D positions and 6D orientations are included, with planar XZ positions computed relative to the first frame. Facing direction invariance is enforced through random direction augmentation during training. The final motion feature is $x \in \mathbb{R}^{C_{\text{in}} \times \mathtt{t}}$, where C_{in} is the motion feature dimension and t is the number of frames.

3.3. Conditioning Signal

Following CAMDM [CSH*24], we use three conditioning inputs to guide autoregressive motion generation:

- **History**: past motion of a fixed number of frames.
- **Future root trajectory**: target 2D root displacements (x_{root}, z_{root}) and yaw orientation in 6D (θ_{root}).
- **Motion style**: a one-hot vector specifying the desired style (e.g., natural walking, dancing).

Each input is encoded by a single-layer network into tokens of equal latent dimension, which are concatenated to form the context sequence ctx. The scalar noise level σ is broadcast over the diffusion window, and embedded by a time-embedding network [KAL*24]. Then both the time embeddings and ctx are fed into a cross-attention module to produce the conditioning signal $c \in \mathbb{R}^{C_{\text{cond}} \times \mathtt{t}}$.

3.4. System Architecture

3.4.1. Backbone Block

A backbone block in F3AMD serves as the fundamental building unit of the diffusion model, as shown in Figure 2a. Motion

features $x \in \mathbb{R}^{C_{\text{in}} \times \mathtt{t}}$ are first encoded into a latent space $z \in \mathbb{R}^{D \times \mathtt{t}}$ by an FNO encoder block, then processed through the backbone blocks **N** times, and finally decoded back to the original motion feature dimension via a final FNO decoder block. The parameter **N** is referred to as the depth of the backbone. Each backbone block consists of a base block, which can be either an FNO block or a Transformer encoder block, followed by a FiLM layer to modulate conditional features. The Transformer encoder block follows the standard self-attention and feed-forward formulation with positional encodings [VSP*17]. Hereafter, we use the naming convention **F3AMD-BaseBlock-LatentDimension** to denote a specific architectural variant.

3.4.2. FNO Block

The FNO block (Figure 2b) feeds a latent feature z to a spectral branch and a local bias branch, and returns the sum:

$$z^{\text{spec}} = \text{IRFFT}\big(\mathbf{W} \odot \text{RFFT}(z)_{[:,0:m]}\big), \tag{2}$$

$$z^{\text{loc}} = \text{SiLU}\Big(\text{Conv}_k\big(\text{SiLU}\big(\text{Conv}_k(\text{SiLU}(z))\big)\big)\Big), \tag{3}$$

$$z^{\text{base}} = z^{\text{spec}} + z^{\text{loc}}. \tag{4}$$

Here, RFFT and IRFFT denote the real Fourier transform and its inverse. The learned complex weights $\mathbf{W} \in \mathbb{C}^{d_0 \times d_1 \times m}$ modulate the first m retained frequency modes to provide global spectral transformation. In the local bias branch, two temporal convolutions capture local motion details: the first Conv_k maps dimension d_0 to d_0 and the second maps d_0 to d_1. The FNO block can be an encoder/decoder by setting $d_0 \neq d_1$, and a sequence-to-sequence model by setting $d_0 = d_1$.

3.4.3. FiLM-Style Conditioning Layer

The FiLM module applies affine modulation to the output of the base block z^{base} based on the conditioning signal c as follows:

$$[\gamma, \beta] = \text{Conv}_{1\times1}(c), \tag{5}$$

$$\tilde{z} = z^{\text{base}} \odot (1 + \tanh(\gamma)) + \beta, \tag{6}$$

$$z^{\text{out}} = \text{SiLU}\big(\text{Conv}_{1\times1}(\tilde{z})\big), \tag{7}$$

where the bounded gate $(1+\tanh(\gamma)) \in [0,2]$ stabilizes training and enables adaptive per-channel modulation.

3.5. Noise Schedule and Sampler

We use a variance-exploding (VE) noise schedule for training [KAAL22], where noise is added with a time-dependent scale $\sigma(t)$ while the signal remains unscaled:

$$x_t = x_0 + \sigma(t)\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}). \tag{8}$$

The continuous schedule is defined as

$$\sigma(t) = \big(\sigma_{\max}^{1/\rho} + t(\sigma_{\min}^{1/\rho} - \sigma_{\max}^{1/\rho})\big)^{\rho}, \tag{9}$$

where $t \in [0,1]$, $\sigma_{\max}$ and $\sigma_{\min}$ are the start and end noise levels, and ρ controls the curvature. This formulation supports flexible step counts during inference and is paired with the ODE-based DDIM sampler for efficient sampling [SME21].

4. Training and Evaluation

We train on the 100STYLE dataset, which is a standard benchmark for multi-style human locomotion [MSK22]. We resample the motions to 30 FPS, which is sufficient for learning human motions.

4.1. Loss Function

We adopt the loss function from CAMDM [CSH*24], denoted as $\mathcal{L}_{\text{baseline}}$, which consists of a reconstruction loss, position and velocity losses, and a foot contact loss (see supplementary materials). We augment this loss function with an additional term, named Inter-Round Discontinuity loss ($\mathcal{L}_{\text{IRD}}$), to penalize temporal discontinuities across autoregressive rounds. $\mathcal{L}_{\text{IRD}}$ measures spatial discrepencies at autoregressive window boundaries by minimizing the Euclidean distance between the global joint positions of the last frame of window t_k and the first frame of the next window t_{k+1}:

$$\mathcal{L}_{\text{IRD}} = \mathbb{E}\big[\big\|\text{FK}(\hat{x}_{t_{k+1}}) - \text{FK}(\hat{x}_{t_k})\big\|_2\big] \tag{10}$$

Our final training objective is $\mathcal{L} = \mathcal{L}_{\text{baseline}} + \lambda_{\text{IRD}}\mathcal{L}_{\text{IRD}}$.

4.2. Experimental Protocol

We evaluate whether our models improve efficiency while maintaining motion quality relative to the CAMDM baseline [CSH*24]. We vary three factors: (1) baseblock type (F3AMD-FNO vs. F3AMD-Transformer), (2) latent dimension (32, 48, 64, 96, 128), and (3) diffusion window length (32 vs. 64 frames). All models are trained under the same parameter settings where possible. After identifying the best quality–efficiency configuration, we further test reducing the denoising steps from 8 to (4, 2, and 1).

4.3. Metrics on Model Efficiency

We evaluate model efficiency using two metrics:

- **Synthesis Time**: Average wall-clock time in milliseconds (ms) per diffusion round, computed over 100 rounds for different character counts $\{2^i, i \in [0,\ldots,10]\}$ on GPU and $\{2^i, i \in [0,\ldots,5]\}$ on CPU.
- **Training Time**: Average time in minutes per training epoch, computed for 100 epochs for each model.

4.4. Metrics on Motion Quality

We evaluate motion quality on long-horizon autoregressive animation rollouts. Each model generates a 2,560-frame (85 seconds at 30 FPS) sequence, corresponding to 80 autoregressive rounds (32-frame models) or 40 rounds (64-frame models). All models are conditioned on a complex 2D root trajectory (with varied speeds, turns, and directions) and 100 different style labels (e.g., Aeroplane, ArmsBehindBack, Cat) to test generalization and style coverage.

We report standard metrics following MDM and CAMDM [TRG*23, CSH*24], including FID (Fréchet Inception Distance), smoothness, foot sliding, and diversity (see supplementary materials). In addition, we adapt two metrics from [CSH*24], incorporate two metrics from related works, and introduce one new metric of our own, as described below.

- **Trajectory Error (Traj.).** Mean Euclidean error per frame in meters between generated and reference root positions on the ground plane. Global angle error between the vectors pointing from the first frame to the current frame is used in [CSH*24], which does not reflect the root location accuracy well. Moreover, trajectory error is calculated for the full rollout in [CSH*24], which tends to be dominated by inter-round discontinuities around window boundaries. We calculate this error by comparing the root position changes of the current frame wrt. the first frame of each diffusion window, and rely on the IRD (to be described shortly) to evaluate inter-round discontinuities.
- **Direction Error (Dir.).** Mean root yaw angle error per frame in degrees. Similar to the trajectory error, we measure the difference of yaw angle changes wrt. the first frame of each diffusion window, rather than the global facing direction error over the full rollout as in [CSH*24].
- **Earth Mover's Distance (EMD).** A non-parametric distributional metric that measures geometric alignment without assuming Gaussianity [PW09, ACB17, ZCLS20]. Similar to FID, lower EMD means lower discrepancy to ground-truth motion distributions. Moreover, EMD values showed better consistency with perceived motion quality than FID in our experiments.
- **Style Coverage.** For each style s with generated counts n_s^{gen} and target counts n_s^{tgt} [NOU*20, MPPSD17, LKFO18]:

$$\text{Coverage} = \frac{\sum_s \min(n_s^{\text{gen}}, n_s^{\text{tgt}})}{\sum_s n_s^{\text{tgt}}} \times 100\%.$$

  Higher values indicate more diverse style coverage.
- **Inter-Round Discontinuity (IRD).** Mean Euclidean distance (in centimeters) of the global joint positions between adjacent frames at window boundaries:

$$\text{IRD} = \mathbb{E}[\|\text{FK}(\hat{x}_{t_k+1}) - \text{FK}(\hat{x}_{t_k})\|_2].$$

  Lower IRD indicates smoother transitions across diffusion windows. This new metric is introduced based on the observation that autoregressive diffusion models often exhibit noticeable discontinuities at window boundaries.

4.5. Implementation Details

All models are implemented in JAX (v0.5.0) [BFS*18], and trained for 100 epochs using an inverse-square learning rate schedule (initial value 5×10^{-4}, decay constant 10^5), batch size 512, and EMA decay 0.993. Training is performed on an NVIDIA RTX A6000 (48 GB) and an AMD Threadripper 3970X CPU. Each training sample uses 10 history frames. All loss weights are set to 1. FNO blocks use 17 frequency modes for 32-frame diffusion windows and 33 modes for 64-frame windows. Noise parameters $\sigma_{\max} = 128$, $\sigma_{\min} = 0.01$, and $\rho = 3$. For synthesis-time benchmarking, models are compiled to IREE with CUDA (GPU) and LLVM (CPU) backends. For fair comparison, we reimplemented CAMDM [CSH*24] in JAX. So throughout this paper CAMDM refers to our JAX implementation, while the implementation from the original authors of [CSH*24] is denoted as CAMDM (PyTorch). Numerical differences between the two CAMDM implementations are reported in the supplementary materials.

CAMDM uses a latent dimension of 256 with depth $\mathbf{N} = 4$. In F3AMD variants, we test reduced latent dimensions and adjust the network depth to balance capacity and efficiency as follows:

- $\mathbf{N} = 8$ for latent dimensions 32 and 48,
- $\mathbf{N} = 6$ for latent dimensions 64 and 96,
- $\mathbf{N} = 4$ for latent dimension 128,

where lower latent dimensions are coupled with greater depth to maintain expressiveness, while higher latent dimenions are coupled with lower depth to control model size.

5. Results

Our experiments show that the proposed **F3AMD** framework achieves *order-of-magnitude* efficiency gains while preserving motion quality comparable to the CAMDM baseline [CSH*24]. Among all tested configurations, **F3AMD-FNO-96** with 32-frame diffusion windows offers the best trade-off between quality and runtime for large crowds. Its single-step variant further accelerates motion synthesis by over **20**× on both GPU and CPU compared to the baseline.

5.1. Model Efficiency

Table 1 reports synthesis time per diffusion round using 8 denoising steps, and Figure 4 visualizes relative speedups. On GPU, F3AMD-FNO models scale more efficiently as the number of characters increases. For short diffusion windows of 32 frames, F3AMD-FNO and F3AMD-Transformer exhibit similar latencies for under ten characters, but performance differences widen beyond 16 characters. F3AMD-FNO reaches 4∼8× speedup over CAMDM, while F3AMD-Transformer variants attain roughly 2.5∼3.5×. For 64-frame diffusion windows, F3AMD-FNO maintains a 3∼5× advantage and F3AMD-Transformer about 2∼3× for large number of characters. On CPU, where absolute latencies are higher, the relative efficiency gains remain similar. For 32-frame diffusion windows, F3AMD-FNO achieves 5∼8× speedups for 4–8 characters and stabilizes near 3∼4× for 16–32 characters. F3AMD-Transformer variants yield lower but consistent 2∼3× performance gains.

Beyond runtime, Table 2 shows that F3AMD-FNO models train the fastest. For 32-frame models, each epoch takes about 5 minutes versus 5∼6 minutes for F3AMD-Transformer models and 12.5 minutes for CAMDM. At 64 frames, F3AMD-FNO still averages 5∼6 minutes per epoch, compared to 8∼10 minutes for F3AMD-Transformer and over 20 minutes for CAMDM. Moreover, F3AMD-FNO models typically converge in either comparable or fewer number of epochs, compared to CAMDM.

5.2. Motion Quality

Table 3 summarizes motion quality across models. With 32-frame diffusion windows, all models achieve similar EMD and comparable perceptual quality. Quantitatively, between F3AMD-FNO and F3AMD-Transformer, F3AMD-Transformer yields slightly better FID and style coverage, while F3AMD-FNO provides better IRD. These differences are visually negligible and likely stem from the

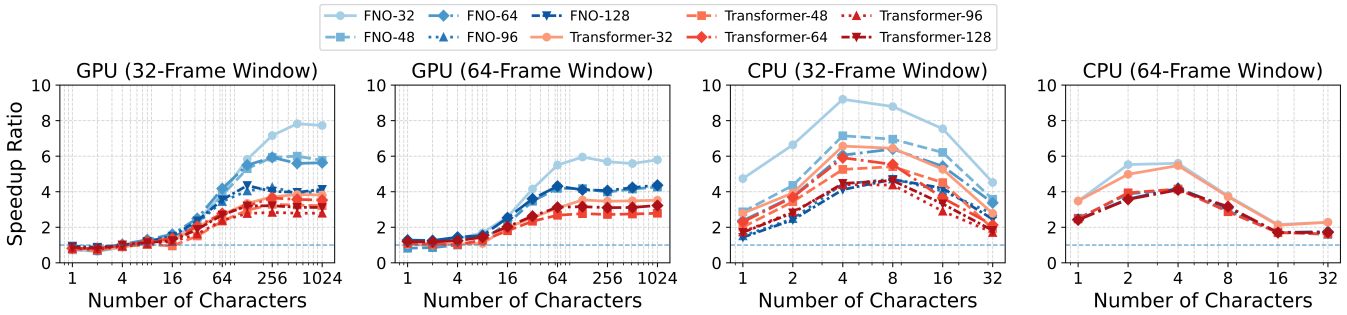


Figure 3: *Speedup over the baseline CAMDM. Each plot shows the runtime ratio t_{CAMDM}/t_{model} vs. the number of characters (log scale). Ratios above 1× indicate F3AMD-FNO-96 is faster. Blue: F3AMD-FNO variants. Red: F3AMD-Transformer variants.*

Table 1: *Synthesis time wrt. the number of characters for F3AMD-FNO, F3AMD-Transformer, and CAMDM. Timing in milliseconds is measured for one autoregressive diffusion round using 8 denoising steps. We test up to 1024 characters on GPU and 32 characters on CPU.*

Diffusion Window	Model Configuration	Number of Characters (GPU) 1	2	4	8	16	32	64	128	256	512	1024	Number of Characters (CPU) 1	2	4	8	16	32
32 frames	F3AMD-FNO-32	4.59	4.74	4.53	4.62	4.78	4.76	5.11	7.07	11.62	21.52	45.02	17.31	26.56	45.31	78.58	133.24	229.20
	F3AMD-FNO-48	4.42	4.95	4.74	4.69	4.75	4.87	5.62	7.97	14.52	28.84	61.45	28.60	40.48	58.34	99.29	161.82	289.68
	F3AMD-FNO-64	5.40	5.37	5.39	5.43	5.56	5.99	6.85	9.34	16.39	33.21	64.04	34.55	47.30	68.80	107.96	184.88	307.17
	F3AMD-FNO-96	5.58	6.31	5.87	5.92	6.46	6.68	8.29	12.70	22.85	47.24	89.47	54.11	70.98	94.05	146.56	248.74	376.07
	F3AMD-FNO-128	4.73	4.96	5.55	5.85	5.96	6.33	8.06	11.81	24.20	46.84	87.23	57.24	72.92	101.29	147.18	239.18	422.12
	F3AMD-Transformer-32	5.33	5.74	6.30	6.03	7.25	7.38	10.63	15.36	26.19	48.38	94.47	29.48	44.83	63.50	107.16	190.90	372.52
	F3AMD-Transformer-48	5.80	6.08	6.17	6.85	9.44	9.90	12.20	17.34	29.56	57.18	112.15	40.84	51.31	79.33	127.69	222.63	537.37
	F3AMD-Transformer-64	5.48	5.53	5.71	5.88	6.52	7.29	10.31	16.04	26.66	52.07	102.21	35.65	47.79	70.55	124.78	275.39	487.10
	F3AMD-Transformer-96	5.49	5.67	6.01	6.73	7.48	9.23	12.00	18.51	34.28	66.54	129.23	47.24	61.44	94.92	157.79	345.85	607.59
	F3AMD-Transformer-128	5.09	5.29	5.61	6.21	7.15	7.95	10.61	16.05	30.28	59.45	115.84	47.94	62.73	93.30	149.34	303.06	568.89
	CAMDM	4.45	4.35	5.60	7.06	8.95	14.94	28.60	51.36	97.23	185.79	360.61	82.04	176.32	416.97	690.85	1004.69	1036.66
64 frames	F3AMD-FNO-64	5.34	5.97	5.38	5.66	6.22	6.80	9.43	16.86	33.58	67.34	128.22	51.82	74.39	118.80	266.03	483.73	850.36
	F3AMD-FNO-96	7.39	7.39	7.30	7.34	7.28	8.15	12.38	24.05	47.95	90.60	174.24	71.72	106.12	160.65	342.86	605.56	1219.57
	F3AMD-FNO-128	4.75	5.04	5.06	6.20	6.35	7.85	12.03	24.42	47.00	88.86	170.23	73.62	114.54	158.89	323.74	609.88	1111.05
	F3AMD-Transformer-64	5.87	6.34	6.88	8.78	8.13	11.09	16.83	28.37	55.13	107.83	211.15	51.75	82.54	121.63	269.55	492.81	853.51
	F3AMD-Transformer-96	5.59	6.07	7.10	7.56	8.91	12.08	19.38	36.27	70.22	136.97	266.59	72.99	104.42	160.28	348.87	614.44	1194.35
	F3AMD-Transformer-128	5.09	5.36	5.87	6.62	7.91	10.76	16.60	31.76	61.57	120.79	230.40	74.62	115.23	162.06	317.21	611.90	1139.48
	CAMDM	6.07	6.32	7.30	9.54	16.07	28.25	51.87	100.36	191.04	376.25	743.21	180.06	411.18	665.34	1003.25	1041.73	1945.38

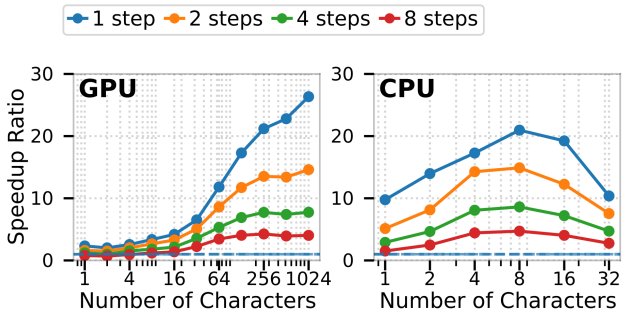


Figure 4: *Speedup of F3AMD-FNO-96 relative to the 8-step CAMDM (32-frame diffusion windows). Each plot shows the time ratio $(t_{CAMDM}/t_{F3AMD\text{-}FNO\text{-}96})$ vs. the number of characters (log scale). Ratios above 1× indicate F3AMD-FNO-96 is faster.*

Table 2: *Training time per epoch (min) on GPU (batch size 512).*

Diffusion Window	F3AMD–FNO 32	48	64	96	128	F3AMD–Transformer 32	48	64	96	128	CAMDM
32 frames	4.92	5.13	5.00	5.07	4.65	5.14	5.39	5.10	5.62	5.16	12.50
64 frames	–	–	5.80	6.48	6.05	–	–	7.95	9.50	8.44	20.60

– motion quality too poor for quantitative reporting

FNO's frequency-domain integration. The CAMDM baseline attains a better FID but a worse IRD. When increasing the diffusion window size to 64 frames, all F3AMD variants exhibit minor degradation in both perceptual and quantitative measures, whereas CAMDM deteriorates substantially, and we therefore omit its numerical results.

Recommended Model. Balancing quality and efficiency, two 32-frame variants F3AMD-FNO-96 and F3AMD-Transformer-64 emerge as the strongest candidates (bolded in Tables 1 and 3). They have comparable quality, and comparable runtime at low character counts. For large crowds, F3AMD-FNO-96 is 10% – 22% faster than F3AMD-Transformer-64 on both GPUs and CPUs.

Table 3: *Quality comparison across latent dimensions for F3AMD-FNO and F3AMD-Transformer, with CAMDM as baseline. Evaluated on a 2560-frame test sequence (80 rounds for 32-frame and 40 for 64-frame models) using 8 denoising steps per round.*

Diffusion Window	Model Configuration	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓
32 frames	F3AMD-FNO-32	0.250	16.332	87.957	0.033	2.202	2.028	1.275	20.448	2.615
	F3AMD-FNO-48	0.217	16.303	90.883	0.028	2.674	2.056	1.299	26.108	2.664
	F3AMD-FNO-64	0.220	16.460	90.169	0.010	2.973	2.277	1.326	26.088	3.621
	F3AMD-FNO-96	0.189	16.134	93.980	0.033	3.260	2.147	1.282	29.721	1.795
	F3AMD-FNO-128	0.211	16.505	91.901	0.074	4.266	2.209	1.282	28.759	1.525
	F3AMD-Transformer-32	0.165	15.677	95.729	0.037	2.135	1.753	1.307	29.770	4.497
	F3AMD-Transformer-48	0.176	16.172	95.652	0.045	2.341	1.969	1.344	33.616	3.414
	F3AMD-Transformer-64	0.147	15.919	95.290	0.051	2.485	1.831	1.281	34.253	3.359
	F3AMD-Transformer-96	0.175	16.118	96.325	0.040	2.952	1.742	1.281	33.537	4.015
	F3AMD-Transformer-128	0.165	15.959	95.752	0.046	3.062	1.727	1.274	32.825	3.888
	CAMDM	0.143	16.031	96.762	0.039	2.911	3.357	1.284	34.277	3.834
64 frames	F3AMD-FNO-64	0.272	16.127	91.043	0.180	3.947	2.438	1.279	28.717	1.993
	F3AMD-FNO-96	0.239	16.118	92.646	0.102	3.614	2.540	1.251	31.451	2.029
	F3AMD-FNO-128	0.260	16.059	90.828	0.049	3.671	2.522	1.270	28.853	2.467
	F3AMD-Transformer-64	0.236	16.220	94.732	0.182	2.331	2.307	1.349	34.927	2.491
	F3AMD-Transformer-96	0.211	16.003	94.746	0.097	3.221	2.412	1.281	36.382	2.023
	F3AMD-Transformer-128	0.222	15.923	93.642	0.044	3.434	2.150	1.267	35.050	2.104
	CAMDM	–	–	–	–	–	–	–	–	–

– motion quality too poor for quantitative reporting

Table 4: *Synthesis time in milliseconds wrt. the number of characters for the 32-frame F3AMD-FNO-96 model using different denoising steps. We test up to 1024 characters on GPU and 32 characters on CPU. 1-step and 2-step CAMDM results are omitted due to poor quality.*

	Number of Characters (GPU)											Number of Characters (CPU)					
Model Configuration	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**1**	**2**	**4**	**8**	**16**	**32**
F3AMD-FNO-96 (8 steps)	5.58	6.31	5.87	5.92	6.46	6.68	8.29	12.70	22.85	47.24	89.47	54.11	70.98	94.05	146.56	248.74	376.07
F3AMD-FNO-96 (4 steps)	3.67	4.39	3.66	3.90	4.18	4.21	5.38	7.44	12.58	25.10	46.53	28.26	37.87	51.65	80.21	139.39	219.99
F3AMD-FNO-96 (2 steps)	2.76	2.89	2.76	2.65	2.72	2.95	3.32	4.38	7.19	13.86	24.69	15.99	21.67	29.21	46.38	82.00	137.48
F3AMD-FNO-96 (1 step)	1.91	2.15	2.19	2.10	2.14	2.29	2.42	2.97	4.59	8.15	13.69	8.40	12.62	24.13	32.96	52.24	99.98
CAMDM (8 steps)	4.45	4.35	5.60	7.06	8.95	14.94	28.60	51.36	97.23	185.79	360.61	82.04	176.32	416.97	690.85	1004.69	1036.66
CAMDM (4 steps)	2.12	2.05	2.30	2.94	4.29	7.42	14.64	27.35	53.15	102.58	199.71	52.03	104.65	209.16	372.48	542.88	596.71

Table 5: *Quality comparison between F3AMD-FNO-96 and CAMDM with 32-frame diffusion windows using different denoising steps. Evaluation is performed over 80 autoregressive rounds. 1-step and 2-step CAMDM results are omitted due to poor quality.*

Model Configuration	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓
F3AMD-FNO-96 (8 steps)	0.189	16.134	93.980	**0.033**	3.260	2.147	1.282	29.721	1.795
F3AMD-FNO-96 (4 steps)	0.189	16.135	93.983	**0.033**	3.260	2.147	1.282	29.719	1.794
F3AMD-FNO-96 (2 steps)	0.184	16.078	93.909	**0.033**	3.257	**2.141**	1.285	29.649	**1.729**
F3AMD-FNO-96 (1 step)	0.184	16.079	93.895	**0.033**	3.257	**2.141**	1.285	29.648	1.730
CAMDM (8 steps)	**0.143**	**16.031**	**96.762**	0.039	**2.911**	3.357	1.284	**34.277**	3.834
CAMDM (4 steps)	0.177	16.761	95.495	0.042	3.046	2.437	**1.211**	29.786	2.808
CAMDM (2 steps)	–	–	–	–	–	–	–	–	–
CAMDM (1 step)	–	–	–	–	–	–	–	–	–

– motion quality too poor for quantitative reporting

These properties make **F3AMD-FNO-96** the more practical and deployment-oriented configuration for multi-character animation applications.

5.3. Denoising Step Analysis

Table 4 reports runtimes of the recommended F3AMD-FNO-96 model wrt. different numbers of denoising steps. On CPU, decreasing from 8 to 1 step achieves 6.4× speedup for 1 character, and 3.4× for 32 characters. On GPU, 2.9× for 1 character and 6.5× for 1024 characters. Motion quality at different steps remains close quantitatively as shown in Table 5, and nearly indistinguishable qualitatively from our perception. Figure 4 plots the relative speedups wrt. the baseline CAMDM. Compared to 8-step CAMDM, 1-step F3AMD-FNO-96 is about 4× faster for 16 characters and 25× faster for 1024 characters on GPU, with similar 10–20× speedups on CPU. Quality of the original CAMDM degrades significantly with reduced number of denoising steps, as will be discussed in more detail shortly.

6. Discussion and Future Work

F3AMD demonstrates that frequency-operator–based architectures (FNOs), when combined with lightweight FiLM conditioning and VE-based deterministic sampling, make autoregressive motion diffusion practical for crowd-scale scenarios—a regime that was not feasible with CAMDM-style Transformer architectures. Our key insight is that autoregressive motion diffusion is primarily bottlenecked by architectural and sampling inefficiencies. By co-designing these components, we enable a new operational regime: stable single-step autoregressive diffusion at super-realtime rates on GPUs for crowd-level animation.

Additional ablation studies are presented in Appendix E to examine the contribution of each component. In particular, we further analyze the effects of different backbone and encoder–decoder architectures, conditioning strategies, and latent dimensionalities. We also discuss the relative insensitivity of the VE+DDIM sampler to the number of denoising steps, and outline several directions for future improvements.

Backbone Choice. Transformers operate directly in the time domain with a computational complexity of $O(TD^2 + T^2D)$, where the quadratic term arises from pairwise attention across sequence length T. In contrast, FNOs perform feature mixing in the frequency domain using RFFT with complexity $O(T \log T\, D + mD^2)$, where m denotes the number of retained spectral modes. This replaces the quadratic attention cost $O(T^2D)$ with a quasi-linear spectral transform $O(T \log T\, D)$. In this work, we retain all frequency modes ($m = T/2+1$). Preliminary experiments with reduced modes ($m \ll T/2$) yielded an additional $\sim$ 10–15% runtime improvement for large character counts, with only mild quality degradation – indicating further performance potential for F3AMD-FNO models. Please refer to Table 8- 10 in Appendix E for more detailed comparisons.

Conditioning Method. CAMDM concatenates the condition and feature sequences along the temporal axis, expanding the input length to $T' = 2T + T_{\text{hist}} + T_{\text{style}}$, which leads to a quadratic attention cost $O(T'^2D)$, where D is the latent feature dimension. In contrast, F3AMD applies FiLM modulation through per-channel scale–shift conditioning, preserving the original input length T and introducing only a lightweight computational cost. Please refer Table 11 in Appendix E for more ablation results.

Latent Dimension. F3AMD maintains stability and quality at low latent dimensions (e.g., the F3AMD-FNO-96 configuration), when CAMDM becomes unstable due to severe mode collapses, as shown in Table 12 in Appendix E. The reduced memory footprint and computational cost at low latent dimensions yield 2–3× speedups in both training and inference for F3AMD.

Number of Denoising Steps. Each denoising step entails a full network evaluation; hence reducing the number of denoising steps lowers latency almost linearly. The speedup is slightly sublinear at small character counts due to fixed launch overheads but approaches linear scaling as the number of characters increases.

Noise Representation. Following early motion diffusion works such as MDM and CAMDM [TRG*23, CSH*24], we do not scale the loss by noise magnitude, unlike most non-motion diffusion models [HJA20, KAAL22]. Scaling the loss is not effective when representing joint rotations in $\mathbb{R}^6$. As adding noise in the Euclidean space disregards the $SO(3)$ manifold structure, thus the noise magnitudes do not correlate with rotation differences. Without loss scaling, small-noise samples contribute little to the gradient, biasing training toward higher noise levels. In future work, we plan to explore adding noise directly on $SO(3)$ [YJHS25], which may enable noise-scaled losses and improve performance at low noise.

Motion Quality. Both quantitative metrics and qualitative visual inspections indicate comparable motion quality among the F3AMD-FNO, F3AMD-Transformer, and CAMDM models when using the default 8 denoising steps. Notably, the F3AMD-FNO model maintains stable quality even with fewer denoising steps (Table 5 and Figure 1). In contrast, CAMDM requires at least 4 sampling steps to produce acceptable motion quality. Its results degrade significantly at 2 steps and become unstable at a single step, as shown in our supplementary animation demo.

The step sensitivity of CAMDM stems from its VP noise schedule combined with the DDPM sampler. Under this setup, the model learns relatively coarse and stochastic corrections at high noise levels, followed by finer corrections at low noise levels. Collapsing the sampling process to a single step skips much of the stochastic denoising trajectory and removes the low-noise refinement stage. In contrast, F3AMD uses a VE noise schedule with a DDIM sampler, as described in Section 3.5. This enables the model to learn a clean-motion denoiser over additive noise levels, while DDIM uses this prediction in a deterministic ODE-like update. As a result, the visual difference between 8-step and 1-step denoising is largely imperceptible. We further applied the same VE+DDIM setup to CAMDM and found that it can also produce acceptable motion quality with a single denoising step, achieving a 5× speedup over 8-step CAMDM. We note that although one-step DDIM may reduce diversity, it is fully deterministic only when both the conditioning signals and the initial noise are fixed. In our demos, diversity is still introduced through varying trajectories, styles, and noise samples across multiple characters.

FNO variants may lose subtle motion details such as fine-grained foot movements. In general, low-amplitude motion components are more difficult to recover in Fourier-based models because their spectral magnitudes are weak and can be overwhelmed by stronger motions from other body parts at similar frequencies. Future work will explore hybrid spatial–spectral architectures to further improve motion fidelity and better balance the tradeoffs among efficiency, smoothness, diversity, and fine motion detail.

For all models, minor discontinuities remain visible at diffusion window boundaries. Such artifacts are commonly mitigated in autoregressive methods through blending or similar post-processing techniques, which can further improve motion quality to production level for deployment. We intentionally did not apply such post-processing in order to examine the raw rollout behavior of the models; this also better motivates our inter-round discontinuity loss and metric. The IRD results show that F3AMD-FNO-96 reduces boundary errors compared with CAMDM, even though temporal FFT on a finite window implicitly assumes periodicity and may cause boundary artifacts.

Autoregressive models tend to accumulate drift in long-horizon rollouts, motivating future exploration of stabilization strategies such as scheduled sampling, condition noising, and training across multiple windows. Evaluating and comparing our approach with more efficient Transformer-style alternatives, such as linear-attention variants [TDBM22, KVPF20, WLK*20], is also a worthwhile direction for future work. This work has primarily focused on locomotion data; extending the evaluation to more heterogeneous motion types remains an important next step.

Acknowledgements This project is partially supported by Mitacs Accelerate Program IT37648/IT47444 and NSERC Discovery Grants Program RGPIN-2024-06752.

References

[ACB17] ARJOVSKY M., CHINTALA S., BOTTOU L.: Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning* (06–11 Aug 2017), Precup D., Teh Y. W., (Eds.), vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 214–223. 5

[BFS*18] BRADBURY J., FROST R., SUTSKEVER I., ET AL.: Jax: Autograd and xla. *GitHub* (2018). URL: https://github.com/google/jax. 5

[CC25] CHOU C.-A., CHEN L.-H.: Reduced-rank factorized fourier neural operator. In *ACML 2025 Conference Track* (2025). 12

[Cla16] CLAVET S.: Motion matching and the road to next-gen animation. Game Developers Conference, 2016. URL: https://www.gdcvault.com/play/1023280/Motion-Matching-and-The-Road. 2

[CSH*24] CHEN R., SHI M., HUANG S., TAN P., KOMURA T., CHEN X.: Taming diffusion probabilistic models for character control. In *ACM SIGGRAPH 2024 Conference Papers* (New York, NY, USA, 2024), SIGGRAPH '24, Association for Computing Machinery. 2, 3, 4, 5, 8, 12, 13

[DCW25] DONG X., CHEN C., WU J.-L.: Data-driven stochastic closure modeling via conditional diffusion model and neural operator. *Journal of Computational Physics 534* (2025), 114005. 3

[HJA20] HO J., JAIN A., ABBEEL P.: Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems* (2020), Larochelle H., Ranzato M., Hadsell R., Balcan M., Lin H., (Eds.), vol. 33, Curran Associates, Inc., pp. 6840–6851. 2, 8

[HKS17] HOLDEN D., KOMURA T., SAITO J.: Phase-functioned neural networks for character control. *ACM Transactions on Graphics 36*, 4 (jul 20 2017), 1–13. 2

[HPD*24] HAN B., PENG H., DONG M., REN Y., SHEN Y., XU C.: Amd: Autoregressive motion diffusion. *Proceedings of the AAAI Conference on Artificial Intelligence 38*, 3 (Mar. 2024), 2022–2030. 2

[HRU*17] HEUSEL M., RAMSAUER H., UNTERTHINER T., NESSLER B., HOCHREITER S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems* (2017), vol. 30. 12

[HW97] HOU T. Y., WU X.-H.: A multiscale finite element method for elliptic problems in composite materials and porous media. *Journal of Computational Physics 134*, 1 (1997), 169–189. 3

[KAAL22] KARRAS T., AITTALA M., AILA T., LAINE S.: Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems* (2022), Koyejo S., Mohamed S., Agarwal A., Belgrave D., Cho K., Oh A., (Eds.), vol. 35, Curran Associates, Inc., pp. 26565–26577. 2, 4, 8

[KAL*24] KARRAS T., AITTALA M., LEHTINEN J., HELLSTEN J., AILA T., LAINE S.: Analyzing and improving the training dynamics of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2024), pp. 24174–24184. 2, 3

[KLL*23] KOVACHKI N., LI Z., LIU B., AZIZZADENESHELI K., BHATTACHARYA K., STUART A., ANANDKUMAR A.: Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research 24*, 89 (2023), 1–97. 3

[Kry12] KRY P. G.: Modal vibrations for character animation. In *Motion in Games* (Berlin, Heidelberg, 2012), Kallmann M., Bekris K., (Eds.), Springer Berlin Heidelberg, pp. 66–77. 2

[KVPF20] KATHAROPOULOS A., VYAS A., PAPPAS N., FLEURET F.: Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning* (2020), vol. 119 of *Proceedings of Machine Learning Research*, PMLR, pp. 5156–5165. 9

[LKA*21] LI Z., KOVACHKI N. B., AZIZZADENESHELI K., LIU B., BHATTACHARYA K., STUART A., ANANDKUMAR A.: Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations* (2021). 2, 3

[LKFO18] LIN Z., KHETAN A., FANTI G., OH S.: Pacgan: The power of two samples in generative adversarial networks. In *Advances in Neural Information Processing Systems* (2018), Bengio S., Wallach H., Larochelle H., Grauman K., Cesa-Bianchi N., Garnett R., (Eds.), vol. 31, Curran Associates, Inc. 5

[LQR*24] LI T., QIAO C., REN G., YIN K., HA S.: Aamdm: Accelerated auto-regressive motion diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2024), pp. 1813–1823. 2

[LT25] LIU X., TANG H.: Difffno: Diffusion fourier neural operator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2025), pp. 150–160. 3, 12

[MPPSD17] METZ L., POOLE B., PFAU D., SOHL-DICKSTEIN J.: Unrolled generative adversarial networks. In *International Conference on Learning Representations* (2017). 5

[MSK22] MASON I., STARKE S., KOMURA T.: Real-time style modelling of human locomotion via feature-wise transformations and local motion phases. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 5*, 1 (may 2022). 4

[MXP*25] MENG Z., XIE Y., PENG X., HAN Z., JIANG H.: Rethinking diffusion for text-driven human motion generation: Redundant representations, evaluation, and masked autoregression. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)* (June 2025), pp. 27859–27871. 2

[NOU*20] NAEEM M. F., OH S. J., UH Y., CHOI Y., YOO J.: Reliable fidelity and diversity metrics for generative models. In *Proceedings of the 37th International Conference on Machine Learning* (13–18 Jul 2020), III H. D., Singh A., (Eds.), vol. 119 of *Proceedings of Machine Learning Research*, PMLR, pp. 7176–7185. 5

[PAAP25] PONTON J. L., ANDREWS S., ANDUJAR C., PELECHANO N.: Environment-aware motion matching. *ACM Transactions on Graphics 44*, 6 (2025), 1–18. 2

[PSDV*18] PEREZ E., STRUB F., DE VRIES H., DUMOULIN V., COURVILLE A.: Film: Visual reasoning with a general conditioning layer. *Proceedings of the AAAI Conference on Artificial Intelligence 32*, 1 (Apr. 2018). 2

[PSW*24] PINYOANUNTAPONG E., SALEEM M. U., WANG P., LEE M., DAS S., CHEN C.: Bamm: Bidirectional autoregressive motion model. In *Computer Vision – ECCV 2024* (nov 2024), Springer Nature Switzerland, p. 172–190. 2

[PW09] PELE O., WERMAN M.: Fast and robust earth mover's distances. In *2009 IEEE 12th International Conference on Computer Vision* (2009), pp. 460–467. 5

[SME21] SONG J., MENG C., ERMON S.: Denoising diffusion implicit models. In *International Conference on Learning Representations* (2021). 2, 4

[SMK22] STARKE S., MASON I., KOMURA T.: Deepphase. *ACM Transactions on Graphics 41*, 4 (7 2022), 1–13. 2

[SSDK*21] SONG Y., SOHL-DICKSTEIN J., KINGMA D. P., KUMAR A., ERMON S., POOLE B.: Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations* (2021). 2

[SSKS23] STARKE P., STARKE S., KOMURA T., STEINICKE F.: Motion in-betweening with phase manifolds. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 6*, 3 (Aug. 2023), 1–17. 2

[SSY*23] SHI M., STARKE S., YE Y., KOMURA T., WON J.: Phasemp: Robust 3D pose estimation via phase-conditioned human motion prior. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2023), pp. 14725–14737. 2

[STKB24] SHAFIR Y., TEVET G., KAPON R., BERMANO A. H.: Human motion diffusion as a generative prior. In *The Twelfth International Conference on Learning Representations* (2024). 2

[SWJ*24] SHI Y., WANG J., JIANG X., LIN B., DAI B., PENG X. B.: Interactive character control with auto-regressive motion diffusion models. *ACM Transactions on Graphics 43*, 4 (July 2024), 1–14. 2

[SZKS19] STARKE S., ZHANG H., KOMURA T., SAITO J.: Neural state machine for character-scene interactions. *ACM Transactions on Graphics 38*, 6 (Nov. 2019), 1–14. 2

[SZKZ20] STARKE S., ZHAO Y., KOMURA T., ZAMAN K.: Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics 39*, 4 (Aug. 2020). 2

[TDBM22] TAY Y., DEHGHANI M., BAHRI D., METZLER D.: Efficient transformers: A survey. *ACM Computing Surveys 55*, 6 (2022), 1–28. 9

[TRG*23] TEVET G., RAAB S., GORDON B., SHAFIR Y., COHEN-OR D., BERMANO A. H.: Human motion diffusion model. In *The Eleventh International Conference on Learning Representations* (2023). 2, 4, 8, 12

[VSP*17] VASWANI A., SHAZEER N., PARMAR N., USZKOREIT J., JONES L., GOMEZ A. N., KAISER L. U., POLOSUKHIN I.: Attention is all you need. In *Advances in Neural Information Processing Systems* (2017), Guyon I., Luxburg U. V., Bengio S., Wallach H., Fergus R., Vishwanathan S., Garnett R., (Eds.), vol. 30, Curran Associates, Inc. 2, 4

[VWLF*24] VAN WOUWE T., LEE S., FALISSE A., DELP S., LIU C. K.: Diffusionposer: Real-time human motion reconstruction from arbitrary sparse sensors using autoregressive diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2024), pp. 2513–2523. 2

[WHW*23] WAN W., HUANG Y., WU S., KOMURA T., WANG W., JAYARAMAN D., LIU L.: Diffusionphase: Motion diffusion in frequency domain. *arXiv preprint arXiv:2312.04036* (2023). URL: https://arxiv.org/abs/2312.04036. 2

[WLK*20] WANG S., LI B. Z., KHABSA M., FANG H., MA H.: Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020). URL: https://arxiv.org/abs/2006.04768. 9

[YJHS25] YU H., JIN Y., HE Y., SUI W.: Efficient task-specific conditional diffusion policies: Shortcut model acceleration and so(3) optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (June 2025), pp. 4213–4222. 8

[YTY*23] YIN W., TU R., YIN H., KRAGIC D., KJELLSTRÖM H., BJÖRKMAN M.: Controllable motion synthesis and reconstruction with autoregressive diffusion models. In *2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)* (Aug. 2023), IEEE, p. 1102–1108. 2

[Z*24] ZHU J., ET AL.: Unveiling the secret of adaln-zero in diffusion transformer. In *International Conference on Learning Representations (ICLR)* (2024). 12

[ZBL*19] ZHOU Y., BARNES C., LU J., YANG J., LI H.: On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019). 3

[ZCLS20] ZHANG C., CAI Y., LIN G., SHEN C.: Deepemd: Few-shot image classification with differentiable earth mover's distance and structured classifiers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020). 5

[ZCP*22] ZHANG M., CAI Z., PAN L., HONG F., GUO X., YANG L., LIU Z.: Motiondiffuse: Text-driven human motion generation with diffusion model. *arXiv preprint arXiv:2208.15001* (2022). URL: https://arxiv.org/abs/2208.15001. 2

[ZLT25] ZHAO K., LI G., TANG S.: Dartcontrol: A diffusion-based autoregressive motion model for real-time text-driven motion control. In *The Thirteenth International Conference on Learning Representations* (2025). 2

[ZSKS18] ZHANG H., STARKE S., KOMURA T., SAITO J.: Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics 37*, 4 (July 2018), 1–11. 2

Appendix A: More Architecture Details

For F3AMD-Transformer variants, we scale the number of attention heads with the latent feature dimension. Specifically, for $D = 32$ we use a single head, for $D \in \{48, 64\}$ we use two heads, and for $D \in \{96, 128\}$ we use four heads. This choice keeps the per-head dimensionality sufficiently large to model rich motion-dependent correlations, while avoiding an excessive number of small heads that would fragment the representation and increase computational cost without clear benefits.

Appendix B: CAMDM Loss

In CAMDM [CSH*24], the network is trained to reconstruct the clean motion x_0 from its noise-corrupted version x_τ. Let $\hat{x}_0$ denote the model's reconstruction. The primary objective is a sample reconstruction loss:

$$\mathcal{L}_{\text{samp}} = \mathbb{E}\left[\|\hat{x}_0 - x_0\|_2^2\right] \tag{11}$$

A geometric loss using forward kinematics (FK) is included, which maps joint rotations to global 3D positions:

$$\mathcal{L}_{\text{pos}} = \mathbb{E}\left[\|\text{FK}(\hat{x}_0) - \text{FK}(x_0)\|_2^2\right] \tag{12}$$

To encourage temporal consistency, a velocity loss computed over consecutive frames is included:

$$\begin{aligned}\mathcal{L}_{\text{vel}} = \mathbb{E}\Big[\|\Delta\hat{x}_0 - \Delta x_0\|_2^2\Big] \\ + \mathbb{E}\Big[\|\Delta\text{FK}(\hat{x}_0) - \Delta\text{FK}(x_0)\|_2^2\Big]\end{aligned} \tag{13}$$

A foot contact loss is also used to penalize predicted foot motion at frames where the reference foot is in contact with the ground (see [TRG*23] for more details).

The final training loss combines all the above terms through a weighted sum:

$$\mathcal{L} = \lambda_{\text{samp}}\mathcal{L}_{\text{samp}} + \lambda_{\text{pos}}\mathcal{L}_{\text{pos}} + \lambda_{\text{vel}}\mathcal{L}_{\text{vel}} + \lambda_{\text{foot}}\mathcal{L}_{\text{foot}} \tag{14}$$

Appendix C: Other Metrics Used

Here is a list of additional metrics used from CAMDM [CSH*24]:

- **Fréchet Distance (FID).** It measures the discrepancy between the mean and covariance of two distributions, approximated by multivariate Gaussians [HRU*17].
- **Temporal Smoothness.** Average magnitude of second-order finite differences (joint accelerations) normalized by the mean acceleration of the root in the ground plane.
- **Foot Sliding.** Mean horizontal displacement of toe joints during contact frames (defined by vertical height $<$ threshold 1cm), normalized by the average root horizontal displacement per frame.
- **Diversity.** Variance of local joint positions in the root frame for multiple stochastic samples under identical conditioning signals.

Appendix D: CAMDM Implementation Comparison

Table 6 compares the motion quality of our CAMDM reimplementation in JAX with the original CAMDM implementation in PyTorch [CSH*24]. FID, EMD, style coverage, foot contact, and diversity metrics match closely. The JAX version yields lower trajectory and facing-direction errors, while the PyTorch version shows slightly better smoothness but higher IRD. Overall, the visual motion quality remains comparable. Table 7 compares the performance of our CAMDM reimplementation in JAX with the original PyTorch version [CSH*24]. GPU runtimes are quite similar, but the PyTorch implementation is approximately 2–5× faster than the JAX version on CPU. This gap arises because PyTorch leverages highly optimized CPU-side attention kernels and supporting libraries that are not available in JAX. On GPU, both frameworks rely on CUDA backends, leading to comparable performance.

Appendix E: More Ablation Results

We further ablated more design choices in Table 9- 12, with the recommended configuration, F3AMD-FNO-96 with the standard FNO backbone, FiLM conditioning, FNO encioder/decoder, and m=17 retained frequency modes, as the baseline in the top row of these tables. Table 9 reports variants using FNO vs Linear encoder/decoder. FNO encoder/decoder is preferred, as the linear alternative significantly reduces style coverage and diversity.

Table 10 compares the performance with different number of retained frequency modes m in FNO backbone. m=1 performs poorly; for m>1, results are similar across metrics. The speed gain with lower m is not significant. For example, m =4 is faster than the baseline (m=17) by 3–4% at low character counts, ∼8% at moderate character counts, and 15–16% at high character counts.

Table 8 reports FNO backbone variants: discrete cosine transform (DCT); reduced-rank FNO, which uses low-rank factorization (rank=8) of spectral kernels for a tunable capacity–generalization trade-off [CC25]; and weighted FNO, which employs learnable per-mode gains on spectral coefficients (weight=0.7) that adjust how much each frequency band contributes [LT25]. All FNO variants: Standard FNO, DCT, reduced-rank, and weighted FNO perform similarly overall with small trade-offs here and there.

Table 11 compares different conditioning methods: FiLM, concatenation, AdaLN-zero [Z*24], and cross-attention. All methods are qualitatively comparable with small trade-offs here and there. The major difference is the speed. The more characters to animate, the wider the performance gap is. For 1024 characters, FiLM conditioning is 2× faster than concatenation, ∼32% faster than cross-attention, and ∼9% faster than AdaLN-zero.

Table 12 compares CAMDM-96 with F3AMD-FNO-96. At low latent dimensions, CAMDM becomes unstable due to severe mode collapses.

Appendix F: Sampler Details

Our model uses the variance-exploding (VE) forward noising process:

$$x_\sigma = x_0 + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, I),$$

where σ specifies the noise level. To apply a DDIM sampler, which is conventionally written in the variance–preserving (VP) format $\bar{\alpha}_t \in (0, 1]$, we convert each σ_t to the corresponding $\bar{\alpha}_t$.

Table 6: *Quantitative comparison between our reimplementation of CAMDM in JAX and the original CAMDM (PyTorch) implementation [CSH*24]. Evaluated on a 2560-frame test sequence (8 denoising steps per round).*

Model	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓
CAMDM (JAX)	0.143	16.031	96.762	0.039	2.911	3.357	1.284	34.277	3.834
CAMDM (PyTorch) [CSH*24]	0.142	15.831	95.855	0.178	9.882	2.499	1.249	34.375	4.426

Table 7: *Synthesis time in milliseconds wrt. the number of characters between our reimplementation of CAMDM in JAX and the original CAMDM (PyTorch) [CSH*24] (8 denoising steps per round). We test up to 1024 characters on GPU and 32 characters on CPU.*

	Number of Characters (GPU)											Number of Characters (CPU)					
Model Configuration	1	2	4	8	16	32	64	128	256	512	1024	1	2	4	8	16	32
CAMDM (JAX)	4.45	4.35	5.60	7.06	8.95	14.94	28.60	51.36	97.23	185.79	360.61	82.04	176.32	416.97	690.85	1004.69	1036.66
CAMDM (PyTorch) [CSH*24]	12.31	12.57	12.60	12.53	12.49	13.90	25.30	48.28	90.57	179.73	345.26	34.58	53.08	87.71	155.08	299.21	576.42

Table 8: *FNO backbone variants: DCT, reduced-rank, and weighted FNO as alternatives to the standard FNO in F3AMD.*

Model	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓
F3AMD-FNO-96	0.189	16.134	93.980	**0.033**	**3.260**	2.147	1.282	**29.721**	1.795
F3AMD-DCT-96	0.193	**15.534**	92.528	0.045	3.708	**1.713**	**1.240**	27.670	3.521
F3AMD-weighted-FNO-96	0.190	15.833	93.222	0.081	3.798	1.919	1.289	29.186	1.770
F3AMD-reduced-rank-FNO-96	**0.173**	15.584	**94.612**	0.081	4.328	2.279	1.282	29.437	**1.717**

Table 9: *Encoder/decoder: Linear vs FNO. Replacing the Fourier-based encoder/decoder with standard linear layers.*

Model	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓
F3AMD-FNO-96 (FNO enc/dec)	**0.189**	16.134	**93.980**	**0.033**	3.260	2.147	1.282	**29.721**	1.795
F3AMD-FNO-96 (linear enc/dec)	0.271	**15.733**	87.418	0.037	**1.907**	2.195	1.243	16.067	**1.548**
F3AMD-Transformer-64 (linear enc/dec)	0.416	15.780	74.128	0.052	2.132	**1.629**	**1.130**	4.157	3.350

Table 10: *Retained frequency mode (m) variants. F3AMD-FNO-96 with different m. The baseline uses m=17.*

Model	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓
F3AMD-FNO-96, m=17	0.189	16.134	93.980	0.033	3.260	2.147	1.282	29.721	**1.795**
F3AMD-FNO-96, m=14	0.195	15.870	93.659	**0.028**	3.293	**1.897**	1.290	29.779	3.131
F3AMD-FNO-96, m=12	0.196	15.972	93.416	0.055	3.401	2.179	1.304	30.298	1.939
F3AMD-FNO-96, m=8	**0.178**	**15.648**	93.828	0.096	3.157	2.173	1.284	**30.471**	2.270
F3AMD-FNO-96, m=4	0.181	15.657	**94.687**	0.083	3.320	2.111	**1.281**	29.416	2.006
F3AMD-FNO-96, m=1	0.203	15.790	92.912	0.065	**2.683**	2.115	1.306	26.598	2.287

Table 11: *Conditioning variants: FiLM, concatenation, AdaLN-zero, and cross-attention for F3AMD-FNO-96.*

Model	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓	Time (ms)↓
F3AMD-FNO-96 (FiLM)	0.189	16.134	93.980	0.033	3.260	**2.147**	1.282	29.721	**1.795**	**89.47**
F3AMD-FNO-96 (concatenation)	0.183	15.780	94.317	0.024	3.541	2.174	**1.267**	**30.563**	2.036	183.39
F3AMD-FNO-96 (AdaLN-zero)	**0.169**	15.700	**94.921**	0.019	3.319	2.390	1.322	30.385	2.329	97.37
F3AMD-FNO-96 (cross-attention)	0.173	**15.431**	94.703	**0.016**	**3.002**	2.441	1.290	29.493	2.009	118.40

Table 12: *CAMDM-96 vs. F3AMD-FNO-96. Quantitative comparison when the latent dimension is fixed to 96.*

Model	FID↓	EMD↓	Style%↑	Traj. (m)↓	Dir. (°)↓	Smooth↓	Foot↓	Diversity (cm)↑	IRD (cm)↓
F3AMD-FNO-96	0.189	16.134	93.980	0.033	3.260	2.147	1.282	29.721	1.795
CAMDM-96	1.860	30.743	2.000	0.394	14.560	3.046	3.357	0.496	10.882

σ to $\bar{\alpha}$ conversion:

$$\bar{\alpha}_t = \frac{1}{1+\sigma_t^2} \qquad \alpha_t = \sqrt{\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}}$$

Clean-sample prediction: Our denoising network G directly predicts the clean signal:

$$\widehat{x}_0 = G(x_t, \sigma_t)$$

The noise estimate implied by this prediction is:

$$\widehat{\epsilon} = \frac{x_t - \widehat{x}_0}{\sigma_t}$$

Deterministic DDIM update:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\,\widehat{x}_0 \; + \; \sqrt{1-\bar{\alpha}_{t-1}}\,\widehat{\epsilon}.$$