

Text-to-Any-Skeleton Motion Generation Without Retargeting

Supplementary Material

Qingyuan Liu¹ Ke Lu^{1,2} Kun Dong¹ Jian Xue^{1*} Zehai Niu¹ Jinbao Wang³
¹University of Chinese Academy of Sciences ²Peng Cheng Laboratory ³Shenzhen University
{liuqingyuan23, dongkun22, niuzehai18}@mailsucas.ac.cn, {luk, xuejian}@ucas.ac.cn
wangjb@szu.edu.cn

A. Physical Plausibility Metrics

In this section, we provide detailed formulations of the specialized metrics used to evaluate the physical plausibility of generated motions, which is crucial for animation applications.

A.1 Skating Ratio

Skating artifacts occur when a character’s feet unnaturally slide across the ground surface while they appear to be in contact with it. This phenomenon breaks the illusion of physical plausibility in character animation. The skating ratio metric quantifies the proportion of frames exhibiting this artifact.

As shown in Algorithm 4, we detect skating by first identifying foot-ground contact frames based on height thresholds. When a foot is in contact with the ground but exhibits horizontal velocity exceeding a predefined threshold, we classify it as skating. The final skating ratio represents the proportion of frames where either foot is skating.

This metric is particularly valuable for evaluating locomotion animations, where proper foot planting is essential for visual realism.

A.2 Skeleton Error

Maintaining consistent bone lengths throughout a motion sequence is a fundamental physical constraint in character animation. The skeleton error metric quantifies violations of this constraint by measuring deviations from standard bone lengths.

Algorithm 2 illustrates our approach. For each frame in the motion sequence, we calculate the Euclidean distance between connected joints and compare it with the standard bone length derived from the reference skeleton. The accumulated absolute differences across all bones and frames constitute the total skeleton error.

Lower skeleton error values indicate better preservation

of the character’s physical structure throughout the animation sequence.

A.3 Mean Per Joint Position Error (MPJPE)

MPJPE is a standard metric in pose estimation that measures the average Euclidean distance between predicted and ground truth joint positions after root alignment.

As detailed in Algorithm 1, we first align both the predicted and ground truth poses by centering them at their respective root joints (typically the pelvis). Then, for each corresponding joint pair, we compute the Euclidean distance. The mean of these distances across all joints provides the MPJPE for each pose.

This metric provides a direct measure of positional accuracy, which is crucial for assessing the fidelity of generated motions compared to reference data.

A.4 Jitter Error

Natural human motion exhibits smoothness and continuity. The jitter error metric quantifies unnatural discontinuities or rapid fluctuations in motion, which manifest as visual jerkiness in animations.

Algorithm 3 presents our computation approach. We derive this metric by calculating the third derivative of position with respect to time (known as “jerk” in physics). Specifically, we compute successive differences to obtain velocity, acceleration, and finally jerk. The sum of absolute jerk values across all joints and frames constitutes the jitter error.

Lower jitter error values indicate smoother, more natural motion with fewer abrupt changes in acceleration, resulting in more visually pleasing animations.

These four metrics collectively provide a comprehensive assessment of physical plausibility in character animation, addressing critical aspects from foot-ground interaction to temporal smoothness of motion.

*Corresponding author.

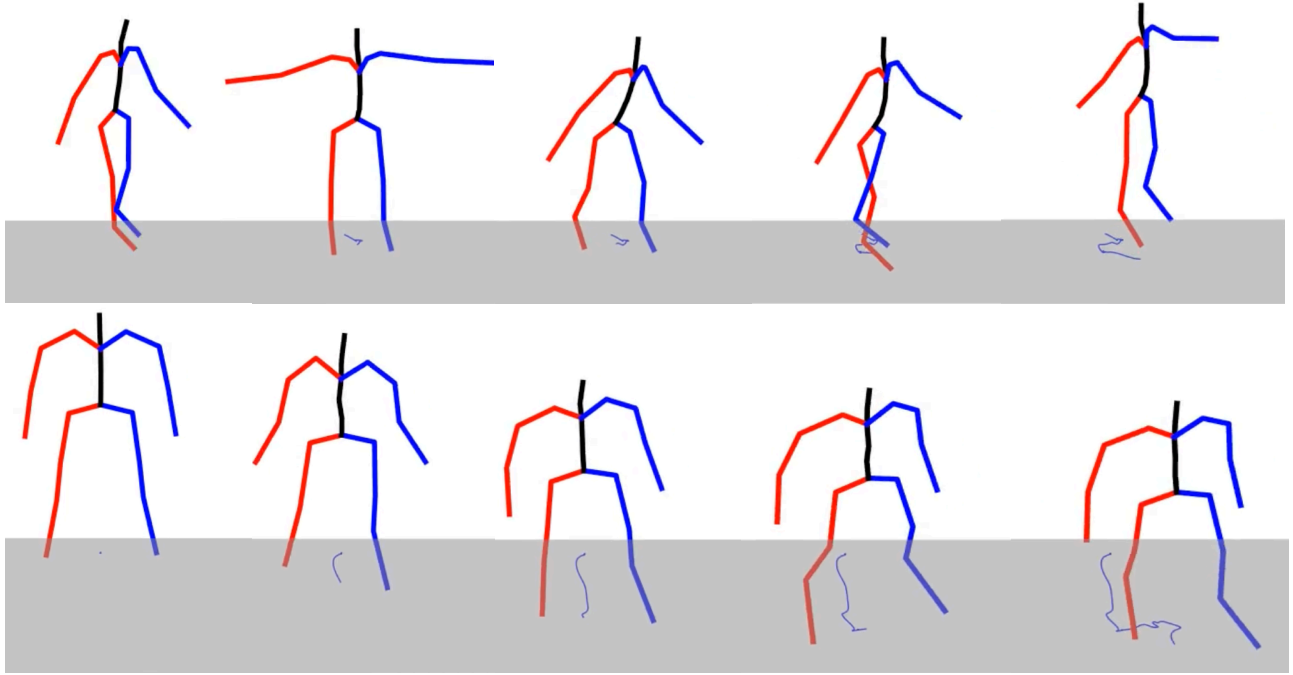


Figure S1. Examples of unconditionally generated motions from our SR-VAE model. The figure displays two different motion sequences (top and bottom rows), each showing five key frames.

B. Unconditional Motion Generation

In addition to the conditional motion generation capabilities described in the main paper, our SR-VAE framework also supports high-quality unconditional motion generation. This section details this complementary functionality and presents qualitative results.

B.1 Unconditional Generation Approach

For unconditional generation, we employ a simple yet effective approach. We set the text embedding input to zero vectors, which effectively removes textual conditioning from the generation process. Additionally, we randomly replace 30% of the tokens in the first layer of our quantized latent representation with other tokens from the same codebook. This introduces controlled stochasticity while maintaining the hierarchical structure of our latent representation, resulting in diverse yet physically plausible motion sequences.

B.2 Quality and Diversity

As shown in Figure S1, our model generates diverse and realistic motion sequences without any explicit conditioning. The generated motions span a wide range of natural human behaviors, from locomotion to gestural movements, while maintaining physical plausibility.

The discrete nature of our SR-VAE’s latent space contributes to the quality of generated motions by effectively capturing the key patterns and modes of human movement. This quantized representation has proven particularly effective at modeling the complex distribution of natural human motion, enabling high-fidelity synthesis even in unconditioned scenarios.

B.3 Applications

Unconditional motion generation has several practical applications in animation and gaming:

- **Background Character Animation:** Generating diverse ambient character motions for populating virtual environments
- **Motion Prototyping:** Rapidly exploring possible motion variations during the animation design process
- **Data Augmentation:** Expanding existing motion datasets with synthetic but realistic samples
- **Idle Animation:** Creating natural idle motions for characters in interactive applications

The ability of our SR-VAE model to support both conditional and unconditional generation makes it a versatile tool for a wide range of character animation applications, offering flexibility to content creators whether they need precise control or creative inspiration.

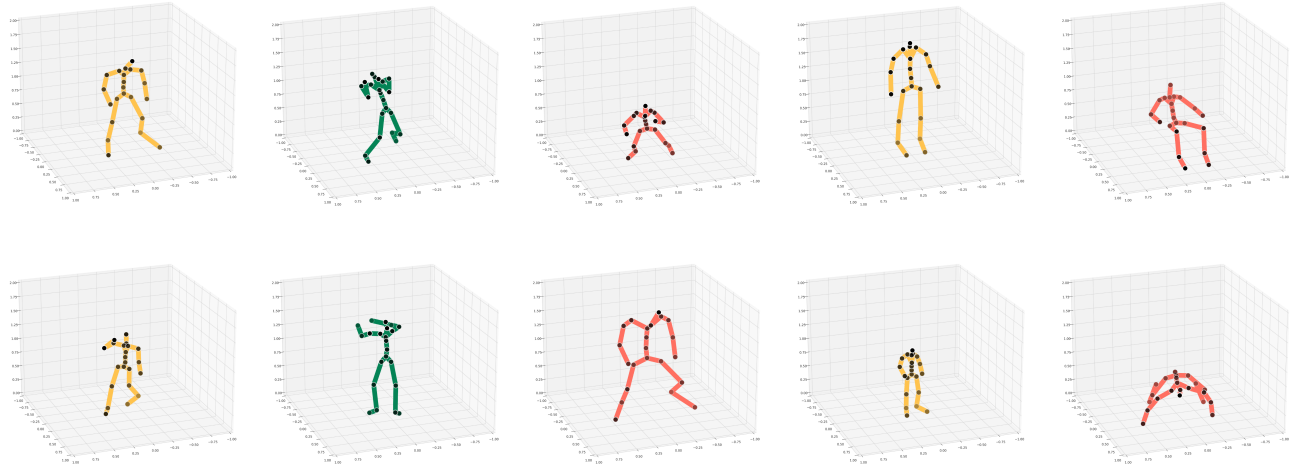


Figure S2. Examples of randomized skeletons generated using our approach. Each skeleton maintains the standard 22-joint SMPL configuration while featuring unique bone length proportions. These variations were produced by applying random scaling factors to individual bone lengths of T-pose characters extracted from the Mixamo library.

C. Dataset Generation Details

This section provides detailed information about our dataset generation process, which creates diverse skeletal motion sequences by combining standard motion patterns with randomized skeleton structures.

C.1 Skeleton Randomization

Unlike previous works that primarily focus on a standard skeleton configuration, our approach incorporates skeleton diversity to enhance the robustness of motion generation models. Our simplified skeleton randomization process includes:

1. Extracting standard T-pose skeletons from characters in the Mixamo library
2. Mapping each extracted skeleton to the standard 22-joint configuration used in the SMPL model
3. Applying random scaling factors to individual bone lengths to create anatomically varied but plausible body proportions
4. Occasionally adjusting overall height to ensure reasonable proportions

Figure S2 shows examples of our randomized skeletons, illustrating the diversity achieved through this approach. The skeletal variations include differences in limb proportions, torso length, and overall height, while maintaining anatomical feasibility.

C.2 Motion Retargeting Pipeline

To create our final dataset, we combine standard motion patterns with our randomized skeletons:

1. We use the HumanML3D dataset as our source of high-quality motion patterns, which provides a diverse range of human movements with corresponding text descriptions.
2. For each motion sequence in HumanML3D, we select the original skeleton and two randomly generated skeletons from our pool.
3. We employ Autodesk MotionBuilder to perform professional-grade motion retargeting, adapting each motion pattern to all three skeleton structures.
4. This process results in three variations of each original motion sequence, significantly expanding the diversity of our dataset while preserving the semantic meaning of the original movements.

D. Limitations and Future Work

Limitations. Despite OmniSkel’s capability to adapt to various human skeletal structures, it has not been extended to non-humanoid topologies. Our model also does not account for skin deformation and rigging factors crucial for production-ready animation. Additionally, while our approach handles variations in human bone proportions well, it may struggle with extremely exaggerated body types that deviate significantly from anatomical norms. Performance degradation is also observed when the text prompts describe highly complex or specialized movements (like specific dance styles or sports techniques) that are underrepresented in the training data.

Future Work. Our skeleton-aware architecture shows potential for integration with music-to-dance generation and multimodal motion synthesis. The tokenized representa-

tion could serve as a building block for more complex systems, including interactive applications and real-time generation. We are particularly interested in extending our framework to handle non-humanoid skeletons, which would open applications in creature animation and robotic movement planning. Another promising direction is combining our skeleton-aware motion representation with physics-based constraints to ensure physical plausibility across different body types and environments.

Algorithm 1: Mean Per Joint Position Error (MPJPE) Calculation

Input: Ground truth joint positions $J_{gt} \in \mathbb{R}^{N \times J \times 3}$
 (poses, joints, xyz), Predicted joint positions
 $J_{pred} \in \mathbb{R}^{N \times J \times 3}$
Output: MPJPE for each pose $E \in \mathbb{R}^N$

```

1 // Step 1: Validate input shapes ; if
  shape( $J_{gt}$ )  $\neq$  shape( $J_{pred}$ ) then
2   Raise Error: "Ground truth and prediction
  shapes must match" ;
3 end

4 // Step 2: Root-relative alignment
  (pelvis-centered) ; for  $n \leftarrow 0$  to  $N - 1$  do
5   // Calculate pelvis position (root joint) ;
     $P_{gt}[n] \leftarrow J_{gt}[n, 0]$  ; // Pelvis
    position in ground truth
6    $P_{pred}[n] \leftarrow J_{pred}[n, 0]$  ; // Pelvis
    position in prediction
7   // Subtract pelvis position from all joints ; for
     $j \leftarrow 0$  to  $J - 1$  do
8      $J_{gt}[n, j] \leftarrow J_{gt}[n, j] - P_{gt}[n]$  ;
      // Center ground truth
9      $J_{pred}[n, j] \leftarrow J_{pred}[n, j] - P_{pred}[n]$  ;
      // Center prediction
10    end
11  end

12 // Step 3: Calculate position error for each joint ;
  for  $n \leftarrow 0$  to  $N - 1$  do
13   for  $j \leftarrow 0$  to  $J - 1$  do
14     // Euclidean distance between predicted
    and ground truth joint ;  $D[n, j] \leftarrow$ 
     $\sqrt{\sum_{d=0}^2 (J_{pred}[n, j, d] - J_{gt}[n, j, d])^2}$  ;
15   end
16 end

17 // Step 4: Average error across all joints for each
  pose ; for  $n \leftarrow 0$  to  $N - 1$  do
18    $E[n] \leftarrow \frac{1}{J} \sum_{j=0}^{J-1} D[n, j]$  ; // Mean error
    per pose
19 end

20 return  $E$  ; // MPJPE for each pose

```

Algorithm 2: Skeleton Error Calculation

Input: Motion sequence $M \in \mathbb{R}^{T \times J \times 3}$ (frames, joints, xyz), Skeleton structure S containing parent indices and offsets

Output: Total skeleton error

```
1 // Step 1: Extract bone connections and standard
  bone lengths ;  $\mathcal{B} \leftarrow \emptyset$ ; // Initialize
  bone connections list
2  $\mathcal{L}_{standard} \leftarrow \emptyset$ ; // Initialize standard
  lengths list
3 for  $i \leftarrow 0$  to  $|S| - 1$  do
4   child  $\leftarrow i$ ; parent  $\leftarrow S[i][0]$ ; // Parent
    index from skeleton data
5   if parent  $\neq -1$  then
6     // Ignore root joint
7     Add [parent, child] to  $\mathcal{B}$ ;
8     offset  $\leftarrow S[i][3:6]$ ; // Joint offset
        from parent
9     standard_length  $\leftarrow \|\text{offset}\|$ ;
        // Standard bone length
10    Add standard_length to  $\mathcal{L}_{standard}$ ;
11  end
12 // Step 2: Initialize error accumulator ;
    $E_{total} \leftarrow 0$ ;
13 // Step 3: Calculate errors for each frame ; for
    $t \leftarrow 0$  to  $T - 1$  do
14    $P_t \leftarrow M[t]$ ; // Joint positions at
     frame t
15   for  $b \leftarrow 0$  to  $|\mathcal{B}| - 1$  do
16     [parent, child]  $\leftarrow \mathcal{B}[b]$ ;
17     // Step 4: Calculate current bone vector
       and length ;  $\vec{v} \leftarrow P_t[\text{child}] - P_t[\text{parent}]$ 
       ; // Bone vector
18      $L_{current} \leftarrow \sqrt{\sum_{d=0}^2 \vec{v}[d]^2}$ ;
        // Euclidean length
19     // Step 5: Calculate error compared to
       standard length ;
        $L_{standard} \leftarrow \mathcal{L}_{standard}[b]$ ;
        $E_{bone} \leftarrow |L_{current} - L_{standard}|$ ;
       // Absolute error
20      $E_{total} \leftarrow E_{total} + E_{bone}$ ;
       // Accumulate error
21   end
22 end
23 return  $E_{total}$  ;
```

Algorithm 3: Motion Jitter Error Calculation

Input: Motion sequence $M \in \mathbb{R}^{T \times J \times 3}$ (frames, joints, xyz)

Output: Jitter error (lower values indicate smoother motion)

```
1 // Step 1: Calculate first-order derivatives
  (velocity) ;  $V \leftarrow \emptyset$ ; // Initialize
  velocity container
2 for  $t \leftarrow 1$  to  $T - 1$  do
3    $V_t \leftarrow M_{t+1} - M_t$ ; // Position
    difference between
    consecutive frames
4   Add  $V_t$  to  $V$ ;
5 end
6 // Step 2: Calculate second-order derivatives
  (acceleration) ;  $A \leftarrow \emptyset$ ; // Initialize
  acceleration container
7 for  $t \leftarrow 1$  to  $|V| - 1$  do
8    $A_t \leftarrow V_{t+1} - V_t$ ; // Velocity
    difference between
    consecutive frames
9   Add  $A_t$  to  $A$ ;
10 end
11 // Step 3: Calculate third-order derivatives (jerk)
    ;  $J \leftarrow \emptyset$ ; // Initialize jerk
    container
12 for  $t \leftarrow 1$  to  $|A| - 1$  do
13    $J_t \leftarrow A_{t+1} - A_t$ ; // Acceleration
    difference between
    consecutive frames
14   Add  $J_t$  to  $J$ ;
15 end
16 // Step 4: Calculate average absolute jerk as the
    jitter error ;
    jitter_error  $\leftarrow \sum_{t=1}^{|J|} \sum_{j=1}^J \sum_{d=1}^3 |J_{t,j,d}|$ ;
    // Sum of absolute jerk values
17 return jitter_error ;
```

Algorithm 4: Skating Ratio Calculation

Input: Motion sequence M : joint positions across time

Output: Skating ratio, Skating velocities

```
1 // Step 1: Initialize parameters ;
  thresh_height  $\leftarrow$  0.05 ; // Height
  threshold for ground contact
2 fps  $\leftarrow$  20.0 ; // Frames per second
3 thresh_vel  $\leftarrow$  0.50 ; // Velocity threshold
  (20 cm/s)
4 avg_window  $\leftarrow$  5 ; // Frames for velocity
  averaging
5 // Step 2: Standardize input format ; if  $M$  has
  shape  $[J, 3, T]$  then
6    $M \leftarrow$  reshape  $M$  to  $[1, J, 3, T]$  ; // Add
    batch dimension
7   is_single  $\leftarrow$  True ;
8 end
9 // Step 3: Extract foot joint positions ;
   $V_{feet} \leftarrow M[:, [10, 11], :, :]$  ; // Left and
  right foot joints
10 // Step 4: Calculate horizontal plane velocity ;
   $V_{plane} \leftarrow \|V_{feet}[:, :, [0, 2], 1 :] - V_{feet}[:, :, [0, 2], : - 1]\|_2 \times \text{fps}$  ; // xZ-plane
  velocity
11 // Step 5: Apply temporal smoothing ;  $V_{avg} \leftarrow$ 
  uniform_filter1d( $V_{plane}$ , size=avg_window) ;
  // Moving average
12 // Step 6: Extract foot heights and detect ground
  contact ;  $H_{feet} \leftarrow V_{feet}[:, :, 1, :]$  ;
  // Y-coordinate (height)
13  $C \leftarrow (H_{feet}[:, :, -1] < \text{thresh\_height}) \wedge (H_{feet}[:, :, 1] < \text{thresh\_height})$  ; // Contact
  frames
14 // Step 7: Calculate skating velocities during
  contact ;  $V_{skate} \leftarrow C \times V_{avg}$  ; // Velocity
  when in contact
15 // Step 8: Identify skating frames ;
   $S \leftarrow C \wedge (V_{plane} > \text{thresh\_vel}) \wedge (V_{avg} > \text{thresh\_vel})$  ; // Skating detection
16 // Step 9: Combine detection for both feet and
  calculate ratio ;
   $S_{combined} \leftarrow S[:, 0, :] \vee S[:, 1, :]$  ; // Either
  foot skating
17 skating_ratio  $\leftarrow \sum S_{combined} / \text{length}(S_{combined})$  ;
  // Proportion of frames with
  skating
18 return skating_ratio ;
```
