NatyaAI: A transformer-based approach for Odissi dance choreography

Shriharshith Keshav
Dept of Electronics and Communication Engineering
PES University
Benagluru, India
shriharshithkeshav@gmail.com

Anushka Singh
Dept of Electrical and Electronics Engineering
PES University
Benagaluru, India
anushkasingh1064@gmail.com

Pragathi Prasad
Dept. of Biotechnology
PES University
Benagluru, India
hipragathiprasad@gmail.com

KS Srinivas
Dept. of AI and ML
PES University
Benagaluru, India
srinivasks@pes.edu

Shwetha KN
Dept. of AI and ML
PES Univeristy
Benagaluru, India
shwethakn@pes.edu

***Abstract*—Odissi is an Indian classical dance form historically performed as a spiritual dance to tell stories about the pastimes of Lord Jagannath, who is a form of the Hindu god Lord Krishna, as well as of other Gods. It is a lyrical dance form which depicts human and divine aspects of life. The use of AI to mimic this dance form is particularly thought-provoking due to challenges encountered in the field of pose detection of complex body movements. Our paper aims to construct a transformer encoder-decoder model that aims to predict the dance steps from a dataset of Odissi dance performance clips and create a stick figure simulation that has the ability to generate dance steps for a piece of input music according to the predicted joint angles.**

***Index Terms*—Odissi, Transformers, Movement, AI, Classical dance, Stick figure, Simulation**

I. Introduction

For centuries, Indian classical dance forms have become the means to communicate stories of the divine pastimes of gods, cosmic principles, and complex human behaviour and emotions. Odissi, one of the eight classical dance forms of India from the state of Odisha, is known for its intricate movements, sculptural poses, and graceful expressions. Characterized by fluid movements and intricate footwork, Odissi incorporates two major postures: the tribhangi and the chowk. The tribhangi divides the body into three bends at the neck, torso, and knee, creating a sinuous, graceful line, while the chowk is a square-like position that symbolizes masculinity and a warrior stance. Both postures are foundational, allowing dancers to convey a wide range of emotions and stories.Odissi dancers use a combination of these postures to create dynamic and expressive movements, including intricate footwork (tatkar), hand gestures (mudras), and facial expressions (abhinaya) to convey various emotions and narratives. The dance is also distinguished by its use of the torso's lateral movements, adding to the complexity and beauty of the performance. The music accompanying Odissi is based on classical ragas and talas, with traditional instruments like the mardala, harmonium, and flute. Dancers undergo rigorous training in basic steps, hand gestures, and facial expressions, with performances typically including invocation, pure dance, expressive dance, and a concluding dance symbolizing spiritual liberation. Odissi originated over 2000 years ago in the temples of Odisha, was suppressed during colonial times, but was revived in the mid-20th century through extensive research and reconstruction efforts [1]

AI-driven dance choreography has seen significant advancements in the context of Indian classical dances, leveraging machine learning models to analyze and generate traditional movements. Various classical dance forms, including Bharatanatyam, Kathak, and Odissi, have been the focus of research, where AI models like PoseNet and MediaPipe are used to capture and analyze dance poses and sequences. LSTM (Long Short-Term Memory) networks, known for their ability to process sequential data, have been employed to predict and generate sequences of dance movements, preserving the intricate patterns of these classical styles. Transformers, with their attention mechanisms, offer robust performance in understanding and generating complex choreography by capturing long-range dependencies in movement sequences. Additionally, GANs (Generative Adversarial Networks) have been utilized to create realistic and innovative dance routines by learning from extensive datasets of dance videos. These AI models not only aid in preserving traditional dance forms but also offer new avenues for creativity and exploration in choreography. The work of Li et.al [2] talks about a system that generates 3D dance movements conditioned on music using the AIST++ dataset. The system leverages deep learning techniques to create realistic dance sequences that align with the rhythm and style of the given music. It was one of the most significant works in the field of AI-driven dance choreography. However, the AIST++ dataset is limited to only certain dance forms and moves, therefore making it a weaker choice for working on certain types of dance forms.

As a result, many other methods have been employed as

well to work with other kinds of dance forms and tackle challenges in extracting dance steps from a variety of other dance forms. Works such as those of Qi et.al [3] employ a model that uses a Seq2Seq architecture combining Long Short-Term Memory (LSTM) and Self-Attention (SA) mechanisms to map the correlation between music and dance movements, ensuring the generated dance is rhythmically aligned with the music and exhibits natural variations. The dataset for the model was created from music and corresponding dance clips, where the dance movements were captured using motion capture technology to obtain 3D joint positions. The dance movements were captured through OpenPose, focusing on 18 key body parts. The music features were extracted using Mel Frequency Cepstral Coefficients (MFCC), and the dataset includes synchronized pairs of music and dance sequences, providing rich information for training the model to generate coherent dance movements conditioned on music.

In the field of Indian classical dance generation, the works of Jadhav et.al [4][5] seem to have been quite significant. The works introduces a system to automate Bharatanatyam dance choreography using genetic algorithms. The model represents dance poses with a 30-attribute vector encompassing the head, hands, waist, and legs. This automated system generates numerous dance step combinations for choreographers, evaluated by a fitness function to ensure adherence to traditional Bharatanatyam aesthetics.

We specifically explore Odissi dance for the following reasons:

- Odissi's unique hip movements and distinct poses provide a different set of joint angles and poses compared to Bharatanatyam's more linear and angular movements, which have already been explored extensively in the works of Jadhav et.al [4][5]
- Kathak's focus on spins and fast footwork contrasts with Odissi's emphasis on sculptural poses and intricate body lines, offering a wider variety of dance movements for modelling. Unlike Kathak, it basically helps establish a middle ground in pose detection where it isn't too difficult to cap certain poses but still has the gracefulness component.
- Odissi's classical, slow, and graceful movements differ significantly from Hip Hop's dynamic and energetic style, which may require different approaches in joint angle predictions and pose generation.
- Ballet's high extensions and jumps contrast with Odissi's grounded and fluid poses, presenting a distinct set of body dynamics for generating accurate dance simulations.

As far as Odissi itself is concerned, the work of Saha et.al [8] focuses on introducing an algorithm for identification of dance video by recognizing posture from each frame. The paper uses a kinetic sensor for pose detection. In our case, we aim to detect the poses using popular pose section libraries from publically available Odissi performances. In this paper, we intend to explore the field of Odissi dance generation for a piece of input music using a custom transformer coupled with certain LSTM layers. We intend to predict the joint angles in our case instead of joint coordinates, unlike in the case of Qi et.al [3] where joint coordinates are predicted, The reason being that joint angles help predict relativistic positions of joints that can make the model more flexible in predicting the dance movements. It removes the ambiguity of the positions as they are more sensitive to the person's physical stature, camera, angle, etc.

We propose this custom transformer model with LSTM layers for Odissi dance generation because it combines both audio and video features to produce dance poses that are synchronized with the rhythm and dynamics of the music. The LSTM and Transformer layers allow the model to capture complex temporal dependencies and contextual relationships between audio cues and dance movements, ensuring that the generated poses reflect Odissi's intricate patterns. This approach leverages the strength of sequential models to handle the temporal aspects of dance while integrating the rich audio context for accurate pose generation.

II. Preliminary tasks

A. Dataset preparation

The dataset we used for training our model was created using a variety of clips of Odissi performances collected from the online video platform YouTube. We collected around 9 individual performances of dancers, all with a resolution of 1080p. A Python script was used to snip the videos to create 30-second clips at 30 fps, resulting in around 900 frames per video. Around 182 clips were created which were then pre-processed.

B. Data pre-processing

The collected clips were thoroughly pre-processed to extract useful pose and audio features. We specifically focused on extracting the joint angles in our case as using angles instead of coordinates ensures consistency in analyzing and generating dance moves. This approach helps us detect positions more efficiently and accurately because of it's relativistic nature. It also helps standardize movement representation across different dancers, enhancing the model's accuracy and realism, which become crucial when we try to simulate the dance moves generated. The pose data in the form of joint angles of 12 joints including shoulders, elbows, wrists, hips, knees, and ankles. These angles were extracted using the popular Python library MediaPipe. Following this, stick figure simulation codes were written to map the joints of the stick figure to the extracted values in the .npy files of the pose data. This is crucial to verify that the joint angles in the files are taken correctly and the simulation can be made possible. Two stick figures were created, one with wider hips and one with more narrower hips. The intention behind both was to understand how variations in body proportions influence dance performance and style, ultimately aiding in the creation of more realistic and personalized dance animations.

Librosa was used to extract audio features. We extracted tempo, beat, MFCC, chromagram, spectral contrast, tonnetz,

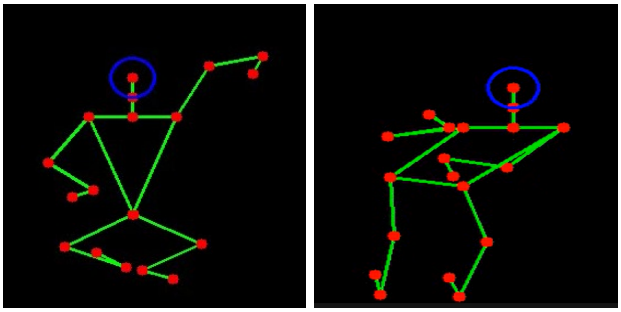


Fig. 1: Stick figures constructed using matplotlib that has the ability to move according to joint angles captured by mediapipe

mel-spectrogram, RMS, ZCR, spectral centroid, spectral bandwidth, and spectral rolloff. Following this, the dataset was thoroughly cubic interpolated to get a constant shape of 900 frames in the case of both pose and audio. This way we maintained a consistent shape across our dataset. The data was normalised to maintain consistency and adding stability to our model. We used sine-cosine normalisation for our angles as it would be easier for us to reverse the process for our simulations. We used min-max normalisation for audio normalisation, our primary goal was to preserve the relative order and distance of our data. Following this, the normalized pose and audio feature data were synchronised, which would be used to train the transformer model.

III. The Transformer

A transformer is a type of neural network architecture that has become widely popular in natural language processing and other sequence-based tasks. We used a Transformer Encoder Decoder along with LSTM layers in our model.

A transformer encoder-decoder is an architecture used in sequence-to-sequence tasks, where the goal is to transform an input sequence into an output sequence. This architecture is especially effective for tasks like machine translation, summarizing, and text generation.

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address some of the limitations of traditional RNNs, particularly their difficulties with learning long-term dependencies in sequences

LSTMs are great at capturing temporal features of a sequence, and Transformers are brilliant at recognising long term dependencies. We are now leveraging the features from both end to bring about a model that holds preserves it's temporal features and is consistent due to it's ability to recognise and adapt to the environment.

A. Specifics

Our entire model has the following features:

1) **Audio Encoder**: Linear layer
   - Input: 155 features
   - Output: 256 features
2) **Positional Encoding**: PositionalEncoding layer
3) **LSTM Layer**:
   - Input: 256 features
   - Output: 256 features
   - Layers: 3
   - Batch First: True
   - Dropout: 0.1
4) **Transformer Encoder**:
   - 6 Transformer Encoder Layer
     - **Self-Attention**:
       - Input: 256 features
       - Output: 256 features
     - **Feed Forward Network**:
       - Linear layers: 256 $\rightarrow$ 2048 $\rightarrow$ 256
       - Dropout: 0.1
     - **LayerNorm**: Applied twice
5) **Transformer Decoder**:
   - 6 Transformer Decoder Layers
     - **Self-Attention**:
       - Input: 256 features
       - Output: 256 features
     - **Multihead Attention**:
       - Input: 256 features
       - Output: 256 features
     - **Feed Forward Network**:
       - Linear layers: 256 $\rightarrow$ 2048 $\rightarrow$ 256
       - Dropout: 0.1
     - **LayerNorm**: Applied three times
6) **Fully Connected Layer**:
   - Input: 256 features
   - Output: 24 features
7) **Dropout Layer**: Dropout rate 0.1
8) **LayerNorm**: Applied after the fully connected layer

B. Architecture

The model starts with a linear layer that projects 155-dimensional audio features to 256 dimensions. This is followed by positional encoding to incorporate sequential information. The LSTM layer, with three layers and a dropout rate of 0.1, processes these features while capturing temporal dependencies.

The Transformer Encoder, consisting of six layers, applies self-attention and a feed-forward network with dropout and layer normalization to refine these features. The Decoder, also with six layers, uses self-attention, multihead attention, and similar feed-forward and normalization techniques to generate output. Finally, a fully connected layer reduces the 256-dimensional features to 24 dimensions, supported by dropout and layer normalization to ensure robust training.

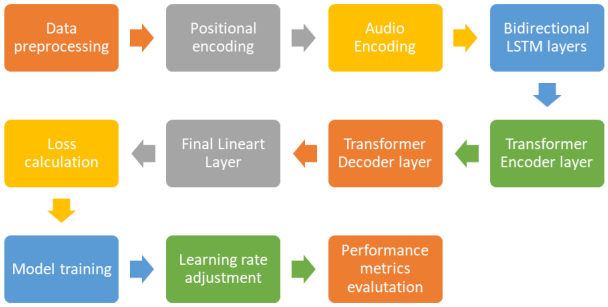


Fig. 2: Architecture of our model

C. Mathematical aspects of different layers

1) Positional encoder: It is typically defined using sine and cosine functions of different frequencies. For a position pos and dimension i, the positional encoding is:

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

where d_{model} is the dimensionality of the model.

2) Audio Encoder: It transforms the audio feature vectors into a format suitable for further processing. Each audio feature vector is linearly transformed using a weight matrix and a bias term. This transformation adjusts the dimensionality of the features and prepares them for integration with the rest of the model. The audio features are linearly transformed using:

$$E_{audio} = X_{audio}W_{audio} + b_{audio}$$

where X_{audio} represents the input feature vectors, W_{audio} is the weight matrix, and b_{audio} is the bias term.

3) Bidirectional LSTM: It aims to capture dependencies in sequential data from both directions—past to future and future to past—enhancing the model's ability to understand context. The input sequence is fed into two LSTM layers - one that processes the data in the forward direction and another in the backward direction. Each LSTM layer maintains a hidden state that updates at each time step based on the input data and the previous hidden state. The outputs from the forward and backward LSTM layers are concatenated or combined to provide a comprehensive representation of the sequence, incorporating information from both directions. For the forward LSTM, the hidden state h_t at time step t is updated as:

$$h_t^{\text{forward}},\ c_t^{\text{forward}} = \text{LSTM}_{\text{forward}}(x_t, h_{t-1}^{\text{forward}}, c_{t-1}^{\text{forward}})$$

For the backward LSTM, the hidden state h_t is updated as:

$$h_t^{\text{backward}},\ c_t^{\text{backward}} = \text{LSTM}_{\text{backward}}(x_t, h_{t+1}^{\text{backward}}, c_{t+1}^{\text{backward}})$$

The final output for each time step is typically a combination of the outputs from both directions:

$$h_t = [h_t^{\text{forward}};\ h_t^{\text{backward}}]$$

The Bidirectional LSTM processes sequential data and maintains long-term dependencies through:

- Input Gate: Controls how much of the new information to add.
- Forget Gate: Controls how much of the old information to retain.
- Output Gate: Controls the output of the LSTM cell state.

The cell state C_t is updated as:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

where f_t is the forget gate, i_t is the input gate, and $\tilde{C}_t$ is the candidate cell state.

4) Transformer Encoder: The Transformer Encoder is a key component of the Transformer model architecture, designed to process sequences of data efficiently by leveraging self-attention mechanisms. It consists of several layers, each comprising two main subcomponents:

- Self-Attention Mechanism:

$$A = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

The attention output is:

$$\text{Attention}(X) = A \cdot V$$

where Q, K, and V are the queries, keys, and values, respectively, and d_k is the dimensionality of the keys.

- Feed-Forward Network:

$$FFN(X) = \text{ReLU}(XW_1 + b_1)W_2 + b_2$$

where W_1 and W_2 are weight matrices, and b_1 and b_2 are bias terms.

5) Transformer Decoder: The Transformer Decoder is a crucial component of the Transformer architecture, designed to generate output sequences from encoded representations. It consists of multiple layers, each featuring three main subcomponents -

- Masked Self-Attention Mechanism:

$$\text{Masked\_Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T - M}{\sqrt{d_k}}\right)V$$

where M is a mask matrix.

- Encoder-Decoder Attention Mechanism:

$$\text{Encoder\_Decoder\_Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Feed-Forward Neural Network:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where W_1 and W_2 are weight matrices, and b_1 and b_2 are bias vectors.

6) Final Linear Layer: Converts the decoder output to the final video frame dimensions.

$$Y_{\text{video}} = X_{\text{decoder}} W_{\text{video}} + b_{\text{video}}$$

where W_{video} is the weight matrix that maps the decoder output to the video frame space, and b_{video} is the bias added to the transformation.

7) Loss Calculation: Combined Loss helps measure how well the predicted video frames match the target frames and other factors like temporal consistency and diversity.

$$\text{Loss} = \text{MSE}(Y_{\text{video}}, Y_{\text{target}}) + \lambda_1 \cdot \text{Temporal Loss} + \lambda_2 \cdot \text{Diversity Loss} + \lambda_3 \cdot \text{SSIM Loss}$$

where:

- MSE: Mean Squared Error between predicted and target frames.
- Temporal Loss: Measures consistency over time.
- Diversity Loss: Encourages diverse outputs.
- SSIM Loss: Measures similarity in structural information between predicted and target frames.

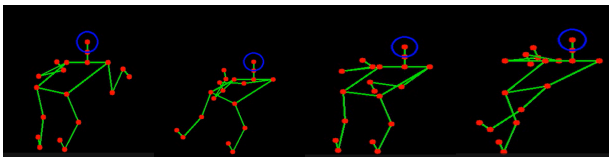
IV. Results

A. Model results

The model was trained for 50 epochs. The following results were obtained upon completion of training:

- Train Loss: 0.2892, Test Loss: 0.5385
- Train MAE: 0.3421, Test MAE: 0.4615
- Train R2: 0.3691, Test R2: 0.1053
- Train Accuracy: 0.1799, Test Accuracy: 0.1460
- Train Rhythm Matching: 0.6763, Test Rhythm Matching: 0.6481
- Train Style Consistency: 0.6946, Test Style Consistency: 0.6758
- Train SSIM: 0.6561, Test SSIM: 0.6376
- Train Fréchet Distance: 0.0466, Test Fréchet Distance: 0.0417

The model shows signs of overfitting as indicated by lower training loss compared to test loss. Performance metrics (MAE, R2, Accuracy) are relatively low, suggesting the model struggles slightly with both training and test data. This could mainly be due to the dataset size we have used in our case. Specific metrics related to dance (Rhythm Matching, Style Consistency, SSIM) are moderate, indicating moderate accuracy in predicting the dance movements. The low test Fréchet Distance implies the generated frames are more realistic compared to the training data.



(a) Jaya (b) Jaya (c) Deva (d) Hare

Fig. 3: Stick figure poses for a small music excerpt from a popular Odia devotional song 'Shritha Kamalakucha Mandala,'[6]

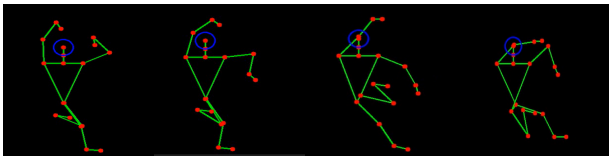


Fig. 4: Stick figure poses for a small music excerpt from the instrumental part of a Mangalacharan song [7]

B. Simulation results

The simulation takes place according to the predicted angles. However, the smoothness and realism of the simulation vary from the original videos. While the limbs such as wrists, ankles and shoulders move effectively, the movement of the hips and movement of the figure within the plane is not seen due to limitations in the model. The movements are consistent through the frames, with considerable repetitions.

V. Limitations

a) The hip angles were not considered during pre processing, and neither were other aspects such as mudras and expressions. More advanced preprocessing techniques such as OpenPose can be used to work on these details to generate more precise dance moves

b) Due to limited computational power and resources, we could only get the stick figure to move in a static position. More advanced models can be constructed that can help generate a figure that can move within 2D-3D planes.

c) A simple stick figure was constructed in our case to showcase dance moves. More realistic 3D human figures can be constructed using modules such as EDGE.

VI. Conclusion

Our custom dance transformer has shown to have been able to successfully train on Odissi data to generate static dance moves. The consideration of joint angles instead of coordinates helps create more flexibility in generating moves and therefore helps create more probabilities and variety in the field of dance generation, which can become a pivotal driver for creative AI dance choreography.

REFERENCES

[1] "Odissi Dance - Origin, Techniques, Costumes and Instruments," Testbook. https://testbook-com.cdn.ampproject.org/v/s/testbook.com/amp/ias-preparation/odissi?amp_gsa=1&amp_js_v=a9&usqp=mq331AQIUAKwASCAAgM%3D#amp_tf=From%20%251%24s&aoh=17218850332991&referrer=https%3A%2F%2Fwww.google.com&ampshare=https%3A%2F%2Ftestbook.com%2Fias-preparation%2Fodissi(accessedJul.24,2024)

[2] R. Li, S. Yang, D. Ross, and A. Kanazawa, "AI Choreographer: Music Conditioned 3D Dance Generation with AIST++," Oct. 2021, doi: https://doi.org/10.1109/iccv48922.2021.01315.

[3] Y. Qi, Y. Liu, and Q. Sun, "Music-Driven Dance Generation," IEEE Access, vol. 7, pp. 166540–166550, 2019, doi: https://doi.org/10.1109/access.2019.2953698.

[4] S. Jadhav, A. Aras, M. Joshi, and J. Pawar, "An Automated Stick Figure Generation for BharataNatyam Dance Visualization," Oct. 2014, doi: https://doi.org/10.1145/2660859.2660917.

[5] S. Jadhav, M. Joshi, and J. Pawar, "Art to Smart: An Automated Bharatanatyam Dance Choreography," Applied Artificial Intelligence, vol. 29, no. 2, pp. 148–163, Feb. 2015, doi: https://doi.org/10.1080/08839514.2015.993557.

[6] "Srita Kamala," kksongs.org. https://kksongs.org/songs/s/sritakamala.html(accessedAug.03,2024).

[7] natyasutra, "Mangalacharan (Ganesh Vandana) Odissi by Sujata Mohapatra, Choreography Guru Kelucharan Mohapatra," YouTube, Jul. 27, 2017.https://www.youtube.com/watch?v=llIxMQT4MUM(accessedAug.03,2024).

[8] natyasutra, "Mangalacharan (Ganesh Vandana) Odissi by Sujata Mohapatra, Choreography Guru Kelucharan Mohapatra," YouTube, Jul. 27, 2017.https://www.youtube.com/watch?v=llIxMQT4MUM(accessedAug.03,2024).

[9] S. Saha, S. Ghosh, A. Konar and R. Janarthanan, "Identification of Odissi dance video using Kinect sensor," 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Mysore, India, 2013, pp. 1837-1842, https://doi.org/10.1109/ICACCI.2013.6637461.

[10] Y. Liu and M. Sra, "DanceGen: Supporting Choreography Ideation and Prototyping with Generative AI," Designing Interactive Systems Conference, Jul. 2024, doi: https://doi.org/10.1145/3643834.3661594.

[11] H. Bhuyan, J. Killi, J. K. Dash, P. P. Das and S. Paul, "Motion Recognition in Bharatanatyam Dance," in IEEE Access, vol. 10, pp. 67128-67139, 2022 doi:https://doi.org/10.1109/ACCESS.2022.3184735

[12] Y. Cheng and Y. Wang, "Transformer-Based Two-level Approach for Music-driven Dance Choreography," Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2024, doi:https://doi.org/10.5220/0012434500003660.

[13] D. Huang, Y. Zhang, Z. Li, and J. Liu, "TG-Dance: TransGAN-Based Intelligent Dance Generation with Music," Lecture notes in computer science, pp. 243–254, Jan. 2023, doi:https://doi.org/10.1007/978-3-031-27077-2_19.

[14] L. Siyao et al., "Bailando++: 3D Dance GPT With Choreographic Memory," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 12, pp. 14192-14207, Dec. 2023, doi:https://doi.org/10.1109/TPAMI.2023.3319435.

[15] B. Wallace, K. Nymoen, J. Torresen, and Charles Patrick Martin, "Breaking from realism: exploring the potential of glitch in AI-generated dance," Digital Creativity, pp. 1–18, Mar. 2024, doi:https://doi.org/10.1080/14626268.2024.2327006.