Proceedings of the 2nd International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/114/2024.18318

AI-Based Music to Dance Synthesis and Rendering

Bohan Jin

Bancroft School, Worcester, USA

ilovemcmrcreeper@gmail.com

Abstract. AI Choreographer is a deep learning model that is able to generate dance motions according to music easily. However, several difficulties and weaknesses in the model still make it difficult to use. For example, the model generates realistic motions, but sometimes the motions are repetitive or do not respond to the audio correctly. Also, the model does not have a usable render that allows it to directly animate the provided models with generated data. We improved our base model to generate more realistic and better dance motions, and we also created a usable automated render pipeline to directly render the generated motions into an animation of the human models provided by the user. We improved the generation quality by introducing more audio features into the model so that the models can utilize more features for better results. Also, we overcame different difficulties in the rendering process, including applying the AI-generated numpy motion data to provided SMPL models and converting the animated SMPL models into usable FBX models. In general, the improved model generates motions that are more diverse and realistic than the base model, which provides dance motions that have higher quality.

Keywords: Dance Generation, Motion Rendering, Deep Learning, Artificial Intelligence for Art, Artificial Intelligence Generated Content (AIGC).

1. Introduction

Dance has already become a prevalent culture worldwide, as many people use it to express and share their ideas. However, dance is still a skill for people, and we must put much time and effort into practicing it to perform well. Fortunately, dance can also be treated as a language, and people can try to use AI models to help with dance motion generation, just like how AI models treat natural languages. That's why some dance generation models, like AI Choreographer, appeared. Those models could use a piece of music as an input and automatically generate the dance motions according to the music.

Unfortunately, AI Choreographer still has weaknesses that make it difficult for people to use. The AI Choreographer did not provide an official render to use the generated data to generate an animated model that could directly be used in many industries. For improvement, we added a usable render that can directly render the generated motion onto a user provided model, and output a ready-to-use FBX model. This generated FBX model can directly be used by most of the popular 3D rendering software and game engines.

Moreover, the AI Choreographer uses audio features, which are the identity that could describe audio, as tokens to compose the audio transformers. However, the amount of audio features used in this model is not sufficient to capture the rich audio information in the music clip. After our investigation, we classified all audio features into two main categories: Spectral Features and Rhythm features. With the

DOI: 10.54254/2755-2721/114/2024.18318

two main categories of features captured, the model generates dance motions that better match the beats and rhythms of the input music.

2. Related Works

2.1. 3D Human Motion Generation

Although human motion generation is still challenging in robotics, computer vision, and graphics, this topic is already well-studied, and there are some well-developed solutions to help deal with human motion generation. With the development of deep learning, people can use deep neural networks to build and train models to generate diverse motions, such as the CNN neural network [1,2] or the Transformer model [3]. AI Choreographer [4] is a music-conditioned 3D dance generation model with the help of a new dancing motion dataset: AIST++ [5]. This model allows the generation of dance motions according to different music using Transformer models to study, analyze, and generate different dance motions by extracting and utilizing the features from music and building relationships between different audio features and poses. EDGE (Editable Dance Generation) is another dance generation model built with a transformer-based diffusion model with a powerful music features extractor. This model also allows for generating "realistic and physically plausible dances." [6].

2.2. The Transformer Models

Dance motion generation is similar to natural language processing [7]; different motions are connected in a particular sequence. The Transformer Model is an excellent way to train and generate dance motions, like natural language analysis and generation. The Transformer Model is a deep learning [8] model that uses attention to calculate the relationship between tokens from the given model. When generating, the Transformer Model will always use its arguments trained from datasets to make predictions on what mostly will the next token be. This process allows computers to do natural language analysis and processing or, more extended, work on image [9], audio [10], or dance motion generation [4,6]. The Transformer Model here will receive an audio clip and generate dance motions according to the audio clip given. The model will output matrixes to indicate joint positions and rotation in a 3D space, which could used in either video rendering or robot motions.

2.3. Audio Features Extraction

Music will hugely affect the dance's motion. Hence, it is also very important to use the features of music to help with the model training process. By identifying features such as envelope [11], MFCC [12], and Chroma [13], we can tell the models what type of music they are and what their rhythms are, which will help classify and produce more precise arguments while going through the attention mechanism in the model. Librosa [14] is a Python library that helps with audio feature extraction, which allows us to utilize different audio features to improve model generation quality.

3. Backgrounds

3.1. The Transformer Model

The main building block for this model is the so-called Scaled-dot-Product Attention [3]. The Scaled-dot Product Attention function can calculate vectors Q (Query), K (Key), and V (Value) simultaneously and calculate the weights of the values, which represent the relationships of the values. The dot product of vector Q with all keys is performed, and the dot product is divided by the square root of the vector's dimension d_k, and the SoftMax of the product is calculated. Finally, the SoftMax value will perform another multiplication with vector V and calculate the values' weights. Here is the general formula for the Scaled Dot-Product Attention function:

$$\text{Attention}(Q,K,V)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

DOI: 10.54254/2755-2721/114/2024.18318

Figure 1 also explains this formula visually:

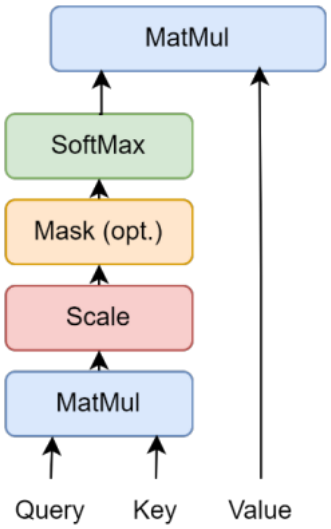

Figure 1. The visual model of Scaled Dot-Product Attention [3].

Multiple Scaled Dot-Product Attentions get together and formed Multi-Head Attentions, which is the most important building block of a transformer model, which we will explain later. The Multi-head Attention gets Query, Key, and Value vectors and perform a linear transformation for each of those. The multi-head attention will apply the Q, K and V vectors into each layer of scaled dot-product attention, and finally concatenate all results and perform another linear transformation, which is represented in the graph below:

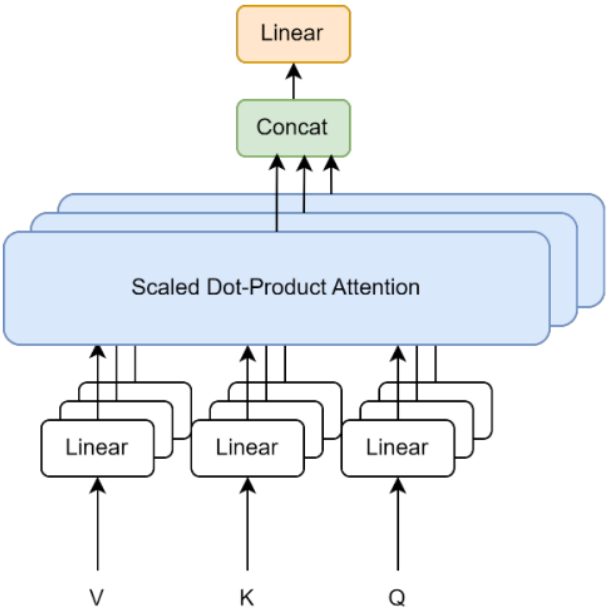

Figure 2. The Visual Model of Multi-Head Attention [3]

DOI: 10.54254/2755-2721/114/2024.18318

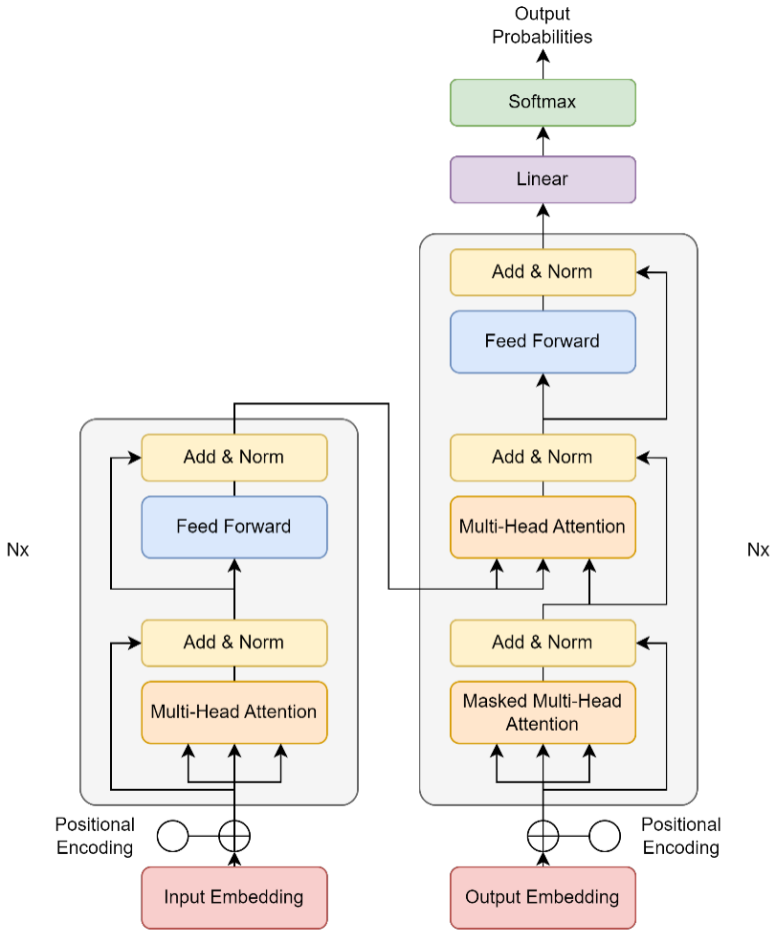

Figure 3. The Visual Model of the Transformer Model [3]

For the transformer model, just like other neural transduction models, has an encoder and decoder. During the encoding process, the model will map the series of input into a continuous representation, and the decoder will use the generated continuous representations to generate the output sequence. For each sublayer of the encoder, it contains a multi-head attention and a simple and fully connected feed-forward network. The structure of the sublayer of the decoder is similar but the decoder contains an extra masked multi-head attention. A transformer model contains the same number of encoder layers and decoder layers, and usually the number of layers is 6. Figure 3 below shows the visualization of the transformer models:

3.2. Skinned Multi-Person Linear Model (SMPL)

SMPL [15] is the main model used in this project to visualize generated motions, which are represented by the rotation angles of all joints. Here are two formulas that are essential in the process of calculation. The first formula is used to calculate the transformation matrix of joints according to a vector of turning angles related to a particular part of the model, which uses the Rodrigues formula to convert the vectors into rotation matrixes as follows:

$$\exp(\vec{\omega}_j) = I + \hat{\vec{\omega}}_j \sin(\| \vec{\omega}_j \|) + \hat{\vec{\omega}}_j^{\,2} \cos(\| \vec{\omega}_j \|) \tag{2}$$

Where $\vec{\omega}_j$ represents the relative rotation angle of a joint related to its parents in the kinematic tree and I represents the 3x3 identity matrix.

DOI: 10.54254/2755-2721/114/2024.18318

In order to calculate the world positions of the joints during the transformation, we also need to calculate the transformation of each joint relative to their T-pose positions with the formula as follows:

$$G'_k(\vec{\theta}, J) = G_k(\vec{\theta}, J) G_k(\overrightarrow{\theta^*}, J)^{-1} \tag{3}$$

Where the T-pose of the model is represented by $\overrightarrow{\theta^*}$ in the formula and $\vec{\theta}$ represents the rotations of joints in that frame k, and $\boldsymbol{J}$ represents the matrix contains the transformation information of each joint of the model. With the help of formula 2 and formula 3, the absolute positions and rotations of each joint in the model can be easily calculated. Please refer to the paper of SMPL model for the absolute positions' calculation formula [15].

Also, different individuals have different body shapes. For example, some people are taller than others or some people have larger stomachs than others. SMPL models have the compatibility to express different body shapes with the following formula:

$$B_S(\mathrm{c}; S) = \sum_{n=1}^{|\vec{\beta}|} \beta_n S_n \tag{4}$$

Where the $\vec{\beta}$ represents the linear shape coefficient, and S_n represents the orthonormal principal components of shape displacements, which are the simplified and standardized data from the original shape displacements [15]. With this formula, SMPL model is compatible with expressing a variety types of body shapes.

Because of the different body shapes, the joint locations are also different from each other, and it is very important to automatically adjust the joint locations in different body shapes. Otherwise, there would be rendering issue while SMPL is trying to do the skinning for the skeleton. Hence the formula below will automatically calculate different joint locations:

$$J(\vec{\beta}; \mathcal{J}, \overline{T}, S) = \mathcal{J}\left(\overline{T} + B_S(\vec{\beta}; S)\right) \tag{5}$$

Where $\overline{\boldsymbol{T}}$ represents the template location matrix and $\mathcal{J}$ represents the matrix that transforms rest vertices into rest joints [15]. The matrix $\mathcal{J}$ contains example poses from many different people with variety types of poses so that it will make sure that the SMPL model is compatible with all types of body shapes with different joint locations and with different poses. The formulas above are some of the most essential formulas that make sure the model will render and work properly.

3.3. Music Features

Music features are also very important information while generating dance poses, especially in this project; we used more music features to optimize the training and generation results. Music features are the characteristics of music, which use data to represent those characteristics and help with the music classification while training and generating the dance poses. According to the APIs Librosa [8] provides, we classified all the extractable music features into two major types: Spectral Features and Rhythm features. Spectral Features, for example, the chroma and MFCC of a piece of music, are the features that can be represented with a graph. Rhythm features, like the tempo of music, are the features that are more related to the rhythm and beats of the music. Using those features while training, the music can be classified into a more detailed genre so that the model can generate more appropriate dances with the detailed classifications.

4. Methods

4.1. Music Features Extraction

We used the publicly available audio processing toolbox Librosa [8] to extract music features and optimize training results. We extract the envelope of a music clip to see the changes in amplitude and frequency over time. We also extract the MFCCs (Mel Frequency Cepstral Coefficients) [12] and

DOI: 10.54254/2755-2721/114/2024.18318

chroma [13] of the music clip to extract the music's characteristics and melodic features, which the graphs of MFCC and the chroma extracted by Librosa are shown and explained as below:

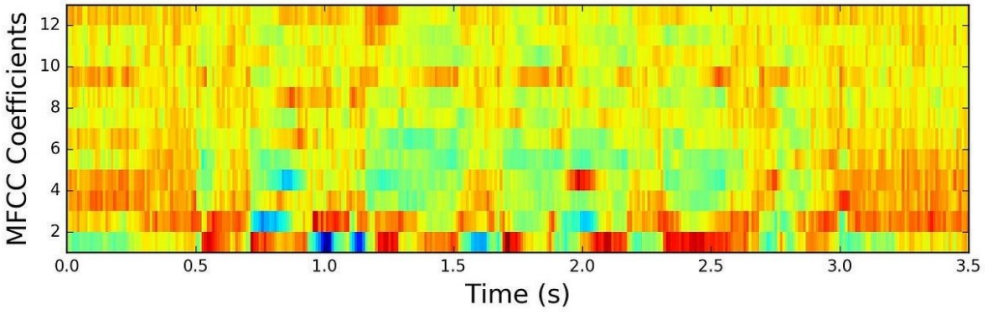

Figure 4. The MFCC feature of An Audio Represented in Graph [16,17]

The MFCC can be calculated with the following process [17]:

1. Pre-Processing: First amplifying higher frequencies, then divide the signal into small, overlapping frames, and finally apply a Hamming Window to soften edges of each frame.
2. Fast Fourier Transformer (FFT): Convert the time domain signal to the frequency domain.
3. Mel-filterbank: Separate into different frequency bands and emphasize important frequencies.
4. Logarithm: Take the logarithm of output from the Mel-filterbank, which compresses the dynamic range of audio and more closely matches human sound intensity.
5. Discrete Cosine Transform (DCT): Highlights the most significant features of the sound in each frame, which effectively captures the characteristics of a sound.

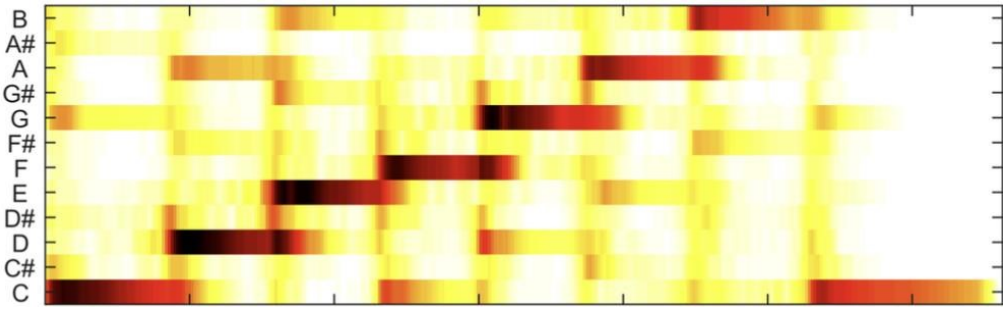

Figure 5. The Chroma Feature of an Audio Represented by Graph [18], where the X-axis represents the time and the Y-axis represents the different chroma of audio.

The basic formula that calculates the chroma of audio is represented as below:

$$chroma[k,t] = \sum_{m \in F(k)} S[m,t] \quad (6)$$

Where k represents the pitch class at time frame k, $S[m,t]$ represents the value of power spectrogram value at frequency bin m and time frame t, and $F(k)$ represents the frequency bins of the particular audio.

One-hot peaks and one-bot beats are also used to extract the pattern of rhythms. The tempo of the music is also extracted, which is also a rhythmic feature to see the speed of the music. The zero-crossing rate is also used in optimization and can recognize any part with no volume. The Spectral Centroid and Spectral Bandwidth features are also used in optimization, which indicates the frequency of energy a clip of music concentrates and shows how wide the energy spreads. All of these features above, which represent the melodic, rhythmic features, and characteristics of a music clip, help optimize the training result by having more features for the models to classify different types of music.

DOI: 10.54254/2755-2721/114/2024.18318

4.2. Dance Poses Generation

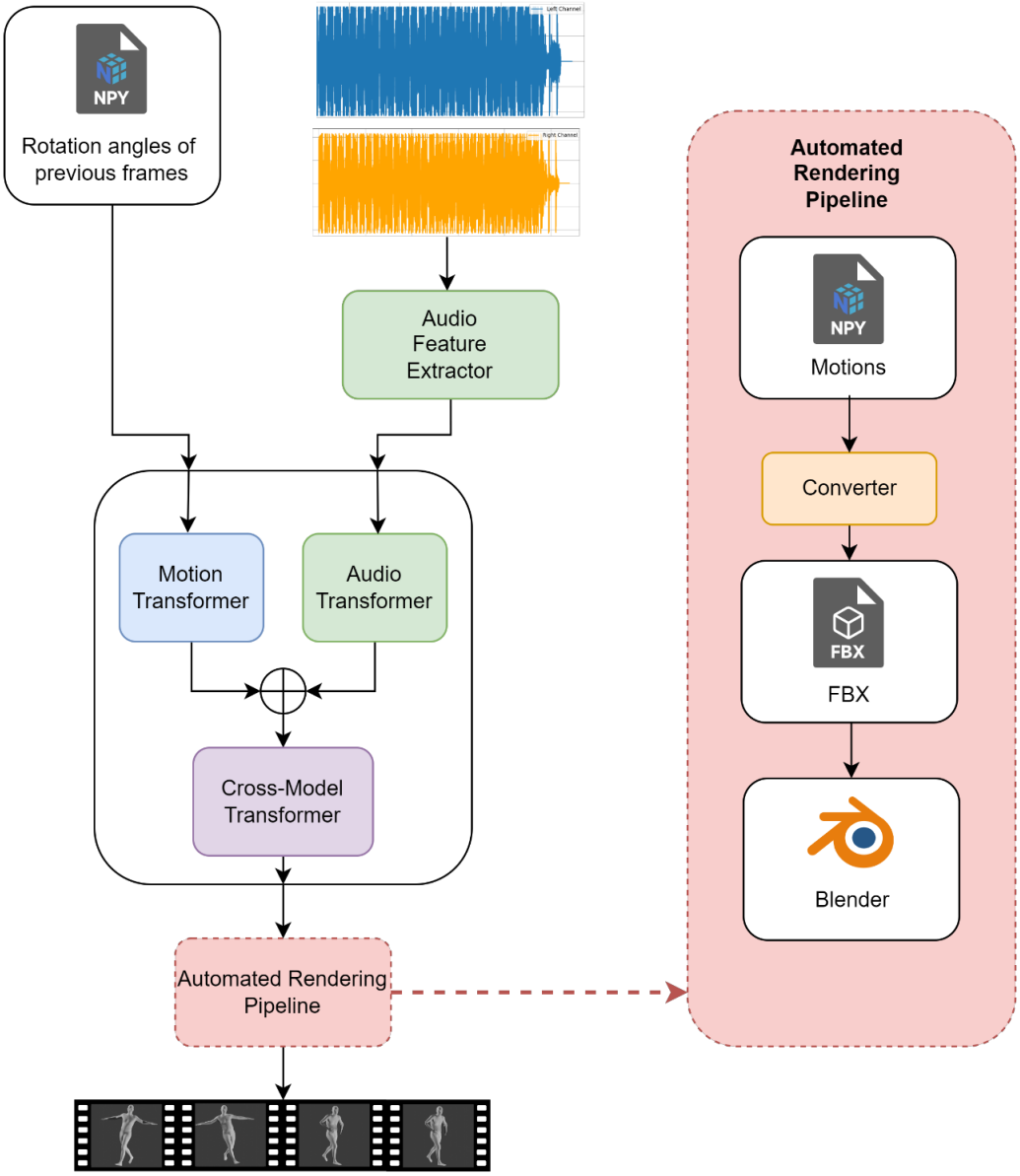

Figure 6. The main processes of Dance Poses Generation. Rotation angles and extracted audio features are input into the model to generate new dance motions, and the generated dance motion in the form of NPY file will be fed into the automated rendering pipeline for model rendering. The automated rendering pipeline will apply the generated motion onto an FBX model and render inside Blender.

The figure above shows the dance pose generation processes, including the rendering part, which has not been officially released by the developer of the AI Choreographer. First, the audio extractor will extract the essential audio features from the audio file provided and send the extracted features to the AI Choreographer. The model includes two transformers: the audio transformer, which will help with processing the audio features given from the extractor, and the motion transformer, which will process the dance motions according to the rotation angles from the previous frames. With the combination of

DOI: 10.54254/2755-2721/114/2024.18318

the two, the model will come up with a cross-model transformer to generate the dance motion in the future frames, which will finally output as a numpy file.

4.3. 3D Animation Rendering

The rendering part will start with the numpy file generated by the AI Choreographer by applying the turning angles into the prepared FBX model converted from the SMPL model, and the converter will output an animated FBX model. The animated FBX model can directly imported into any popular 3D modeling or rendering software, which we are using Blender here, to see the result.

One of the improvements we made here is providing a usable dance pose renderer using the numpy array results generated by the model. Based on the source code of Blender Plugin – SMPL-to-FBX (https://github.com/softcat477/SMPL-to-FBX)– we created a new plugin that takes the numpy file generated by the model and any SMPL model. The plugin will automatically apply the numpy file to the SMPL file as animation and convert the animated SMPL file into FBX. The animation application will follow the joint map below for conversion:

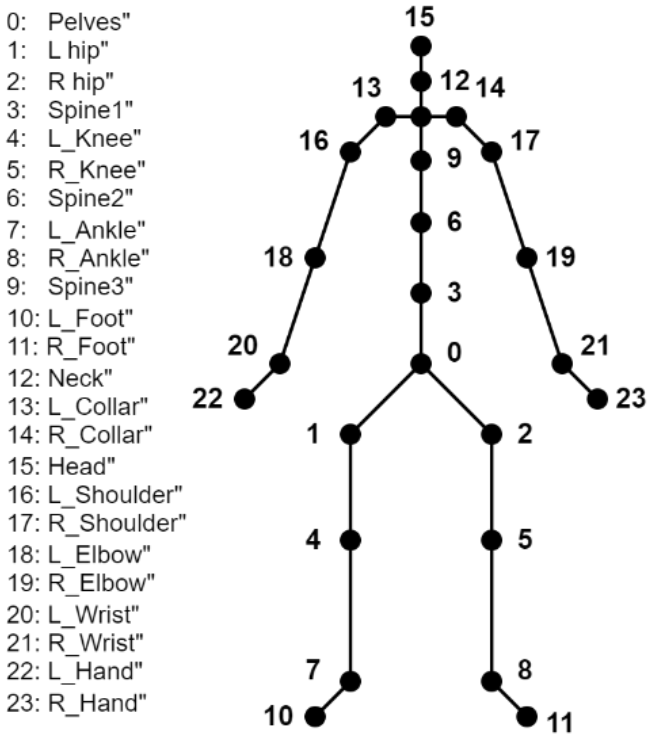

Figure 7. Joints used by the converter to incorporate the generated motions into FBX models. The corresponding joint and the numbering are listed on the left, and the positions of each joint are presented on the right.

The whole converting process can be visualized as below:

DOI: 10.54254/2755-2721/114/2024.18318

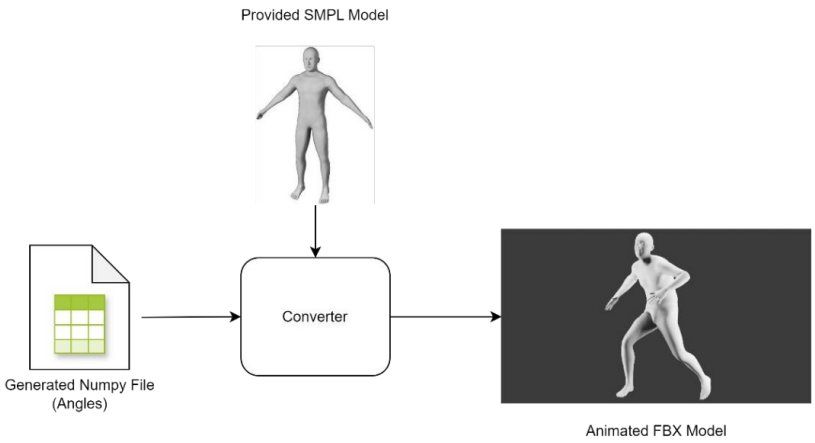

Figure 8. Visual Representation of Converter. With the provided SMPL model and generated motions, the converter will automatically convert into FBX and render into animation.

5. Results

We compared our optimized model with the baseline model to see the differences and improvements in the dance generation results. Firstly, we performed a FID score calculation to compare the generated results between different models in a quantitative way. The test results of the base model, other dance generation model, and our optimized model are shown in the table below:

Table 1. FID Scores Between Different Dance Generation Models

Model Name	Motion Quality (FID_k)
Li *et al.* [19]	86.43
Dancenet [20]	69.18
DanceRevolution [21]	73.42
FACT (Base model)	35.35
FACT-improved (ours)	33.48

Note: The realism and quality of generation can be measured by FID scores. Lower FID scores represent more realistic motion. The data of our base model and other models are from the paper of the base model.

As we can see from the table above, our optimized FACT model has the lowest FID_k score. A lower FID score means the motions we generated have smaller differences compared to the videos danced by real humans, which means the dance motions generated by our improved model are the most realistic among the different models we are comparing here. Also, you can see the differences in the dance poses between the base model and the generated models:

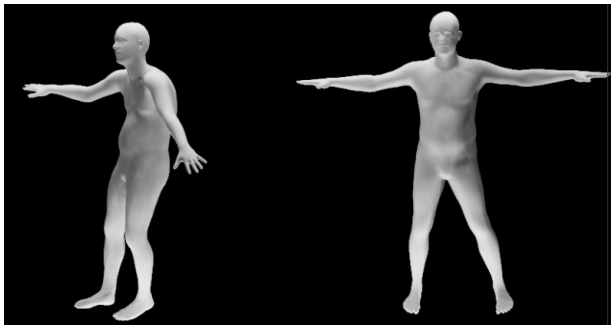

Figure 9A. Motion Comparison between the base model (right) and our improved model (left) on the same frame

DOI: 10.54254/2755-2721/114/2024.18318

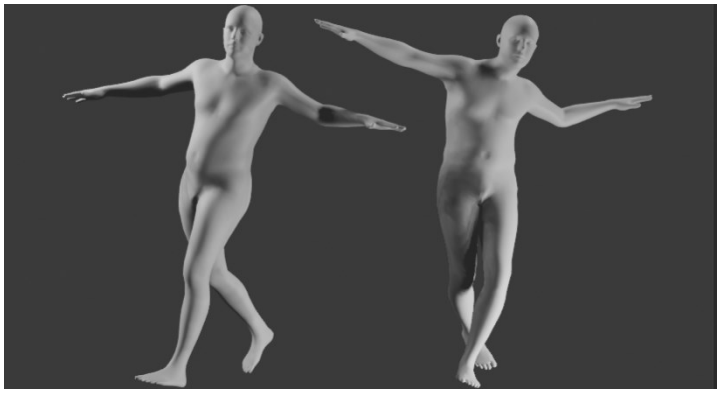

Figure 9B. Motion Comparison between the base model (left) and our improved model (right) on the same frame

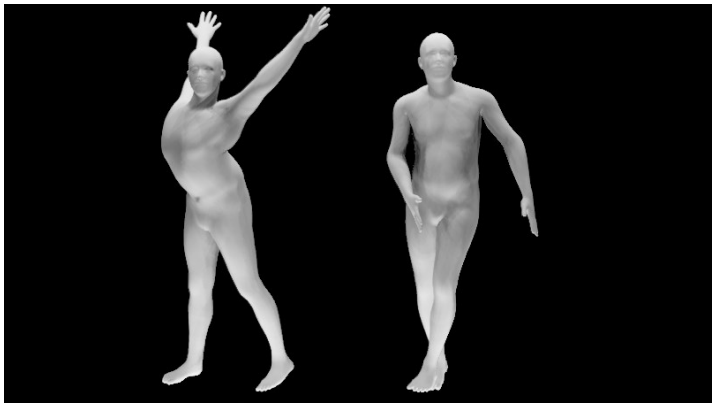

Figure 9C. Motion Comparison between the base model (right) and our improved model (left) on the same frame.

From the series of comparisons on different keyframes of motions, dance motions generated on our models have more responses to the beats of the music, and the motions are more diverse, obvious, and realistic than the dance motions generated by the base model. Dance motion generated by the base model sometimes tends to be steady, or the amplitude of the dance movement is insufficient that people might think the model is standing at the same location or drifting in the air without foot movement, as shown in Figure 9A, but after adding different audio features into the model, the model started to be more sensitive to the music and the dance motions have response to any of the beats, and also the poses and transitions are more suitable for the music clip, as shown in figure 9C. Also, dance motions generated by our model tend to be more diverse in poses. The motions tend to vary from the initial pose as time goes on, which also makes the dance motions more realistic, interactive, and interesting, as shown in Figure 9B.

6. Conclusion

Overall, the AI Choreographer, the AI dance generation model, provides people a way to generate dance motions more easily, and with our enhancement of this model, the AI Choreographer is even stronger and easier to use. The improvements we brought into this model include bringing more audio features to the model to analyze and generate more realistic and detailed dance motions and bringing a usable render that allows for directly generating dance motion videos, reducing the effort to manually render the generated dance motions. Our provided dance generation pipeline is such a powerful tool for people that this model is applicable to many fields and industries, including dance design, game development, films, and television. Our provided dance generation pipeline can also ignite individuals' creative

DOI: 10.54254/2755-2721/114/2024.18318

processes to create and enhance their personal art projects and let people get in touch with dance, this important culture, and art language in an easier and more direct way.

References

[1] O'shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." arXiv preprint arXiv:1511.08458 (2015).

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." Communications of the ACM 60.6 (2017): 84-90.

[3] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[4] Li, Ruilong, et al. "Ai choreographer: Music conditioned 3d dance generation with aist++." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021. (AI Choero)

[5] Ruilong Li*, Shan Yang*, David A. Ross, Angjoo Kanazawa. AI Choreographer: Music Conditioned 3D Dance Generation with AIST++ ICCV, 2021.

[6] Tseng, Jonathan, Rodrigo Castellon, and Karen Liu. "Edge: Editable dance generation from music. " Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023.

[7] Eisenstein, Jacob. Introduction to natural language processing. MIT press, 2019.

[8] Sarker, Iqbal H. "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions." SN computer science 2.6 (2021): 420.

[9] Chen, Hanting, et al. "Pre-trained image processing transformer." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021.

[10] Verma, Prateek, and Chris Chafe. "A generative model for raw audio using transformer architectures." 2021 24th International Conference on Digital Audio Effects (DAFx). IEEE, 2021.

[11] “Envelope (Music).” Wikipedia, 5 Feb. 2023, en.wikipedia.org/wiki/Envelope_(music)#:~:text=In%20sound%20and%20music%2C%20an.

[12] “Mel-Frequency Cepstrum.” Wikipedia, 21 Dec. 2019, en.wikipedia.org/wiki/Mel-frequency_cepstrum.

[13] “Chroma Feature.” Wikipedia, 9 Mar. 2021, en.wikipedia.org/wiki/Chroma_feature.

[14] McFee, Brian et al. “librosa: Audio and Music Signal Analysis in Python.” SciPy (2015).

[15] Loper, Matthew, et al. "SMPL: A skinned multi-person linear model." Seminal Graphics Papers: Pushing the Boundaries, Volume 2. 2023. 851-866.

[16] Kiran, Uday. “MFCC Technique for Speech Recognition.” Analytics Vidhya, 13 June 2021, www.analyticsvidhya.com/blog/2021/06/mfcc-technique-for-speech-recognition/.

[17] Nair, Pratheeksha. “The Dummy’s Guide to MFCC.” Prathena, 27 July 2018, medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd.

[18] GeeksforGeeks. “Mel-Frequency Cepstral Coefficients (MFCC) for Speech Recognition.” GeeksforGeeks, GeeksforGeeks, 26 June 2024, www.geeksforgeeks.org/mel-frequency-cepstral-coefficients-mfcc-for-speech-recognition/#what-are-mfccs.

[19] “Chroma Feature.” Wikipedia, 9 Mar. 2021, en.wikipedia.org/wiki/Chroma_feature.

[20] Jiaman Li, Yihang Yin, Hang Chu, Yi Zhou, Tingwu Wang, Sanja Fidler, and Hao Li. Learning to generate diverse dance motions with transformer. arXiv preprint arXiv:2008.08171, 2020.

[21] Wenlin Zhuang, Congyi Wang, Siyu Xia, Jinxiang Chai, and Yangang Wang. Music2dance: Music-driven dance generation using wavenet. arXiv preprint arXiv:2002.03761, 2020.

[22] Ruozi Huang, Huang Hu, Wei Wu, Kei Sawada, Mi Zhang, and Daxin Jiang. Dance revolution: Long-term dance generation with music via curriculum learning. In International Conference on Learning Representations, 2021.

DOI: 10.54254/2755-2721/114/2024.18318

Appendices

Appendix A: Source Code of Converter
Convert.py

```
from scipy.spatial.transform import Rotation as R
# mathutils are only available for py3
#from mathutils import Matrix, Vector, Quaternion
from FbxReadWriter import FbxReadWrite
from SmplObject import SmplObjects
import argparse
import tqdm
import sys
sys.path.append('~')

def getArg():
    parser = argparse.ArgumentParser()
    parser.add_argument('--input_pkl_base', type=str, required=True)
    parser.add_argument('--fbx_source_path', type=str, required=True)
    parser.add_argument('--output_base', type=str, required=True)

    return parser.parse_args()

if __name__ == "__main__":
    args = getArg()
    input_pkl_base = args.input_pkl_base
    fbx_source_path = args.fbx_source_path
    output_base = args.output_base

    smplObjects = SmplObjects(input_pkl_base)
    print("start")
    for pkl_name, smpl_params in tqdm.tqdm(smplObjects):

        #try:
        fbxReadWrite = FbxReadWrite(fbx_source_path)
        fbxReadWrite.addAnimation(pkl_name, smpl_params)
        fbxReadWrite.writeFbx(output_base, pkl_name)
        print("done")
        #except Exception as e:
        #    fbxReadWrite.destroy()
        #    print ("- - Distroy")
        #    raise e
            #pass
        #finally:
        fbxReadWrite.destroy()
```

Appendix B: Source code for SMPL Object
This file is essential for the converter to work properly

```
import numpy as np
import glob
import pickle
import os
from scipy.spatial.transform import Rotation as R
#from mathutils import Matrix, Vector, Quaternion
```

DOI: 10.54254/2755-2721/114/2024.18318

```
from typing import Dict
from typing import Tuple

from PathFilter import PathFilter

class SmplObjects(object):
    joints = ["Pelvis"
    ,"L_Hip"
    ,"R_Hip"
    ,"Spine1"

    ,"L_Knee"
    ,"R_Knee"
    ,"Spine2"

    ,"L_Ankle"
    ,"R_Ankle"
    ,"Spine3"

    ,"L_Foot"
    ,"R_Foot"
    ,"Neck"

    ,"L_Collar"
    ,"R_Collar"

    ,"Head"
    ,"L_Shoulder"
    ,"R_Shoulder"

    ,"L_Elbow"
    ,"R_Elbow"
    ,"L_Wrist"
    ,"R_Wrist"
    ,"L_Hand"
    ,"R_Hand"]

    def __init__(self, read_path):
        self.files = {}

        # For AIST naming convention
        # paths = PathFilter.filter(read_path,
dance_genres=["gBR"],  dance_types=["sBM"], music_IDs=["0"])
        paths = PathFilter.filter(read_path, dance_genres=None, dance_types=None,
music_IDs=None)
        for path in paths:
            filename = path.split("/")[-1]

            # load npy file
            if filename.endswith(".npy"):
                with open(path, 'rb') as f:
                    data = np.load(f)
                    data = np.array(data)  # (N, 225)
                    f.close()
                trans = data[:, 6:9]
```

DOI: 10.54254/2755-2721/114/2024.18318

```
                poses = data[:, 9:]
                poses = R.from_matrix(poses.reshape(-1, 3,
3)).as_rotvec().reshape(-1, 72)

                self.files[filename] = {"smpl_poses": poses,
                                        "smpl_trans": trans}

            # load pkl file
            else:
                with open(path, "rb") as fp:
                    data = pickle.load(fp)
                self.files[filename] = {"smpl_poses": data["smpl_poses"],
                                        "smpl_trans": data["smpl_trans"]}

        self.keys = [key for key in self.files.keys()]
    # def __init__(self, read_path):
    #     self.files = {}
    #
    #     # For AIST naming convention
    #     #paths = PathFilter.filter(read_path,
dance_genres=["gBR"],  dance_types=["sBM"], music_IDs=["0"])
    #     paths = PathFilter.filter(read_path,
dance_genres=None,  dance_types=None, music_IDs=None)
    #     for path in paths:
    #         filename = path.split("/")[-1]
    #         with open(path, "rb") as fp:
    #             data = pickle.load(fp)
    #         self.files[filename] = {"smpl_poses":data["smpl_poses"],
    #                                 "smpl_trans":data["smpl_trans"] /
(data["smpl_scaling"][0]*100)}
    #     self.keys = [key for key in self.files.keys()]

    def __len__(self):
        return len(self.keys)

    def __getitem__(self, idx:int) -> Tuple[str, Dict]:
        key = self.keys[idx]
        return key, self.files[key]
```