Frontiers in Computer Science

TYPE Original Research
PUBLISHED 11 April 2025
DOI 10.3389/fcomp.2025.1549693

EDITED BY
Caroline Larboulette,
Université Bretagne Sud, France

REVIEWED BY
Nicolas Pronost,
Université Claude Bernard Lyon 1, France
Alexandre Meyer,
Université Claude Bernard Lyon 1, France

*CORRESPONDENCE
Anthony Belessis
✉ up1066445@ac.upatras.gr

RECEIVED 21 December 2024
ACCEPTED 26 March 2025
PUBLISHED 11 April 2025

CITATION
Belessis A, Loi I and Moustakas K (2025) Advanced articulated motion prediction. *Front. Comput. Sci.* 7:1549693. doi: 10.3389/fcomp.2025.1549693

Advanced articulated motion prediction

Anthony Belessis*, Iliana Loi and Konstantinos Moustakas

Department of Electrical and Computer Engineering, Visualization and Virtual Reality Group, University of Patras, Patras, Greece

Motion synthesis using machine learning has seen rapid advancements in recent years. Unlike traditional animation methods, utilizing deep learning to generate human movement offers the unique advantage of producing slight variations between motions, similar to the natural variability observed in real examples. While several motion synthesis methods have achieved remarkable success in generating highly varied and probabilistic animations, controlling the synthesized animation in real-time while retaining stochastic elements remains a serious challenge. The main purpose of this work is to develop a Conditional Generative Adversarial Network to generate real-time controlled motion that balances realism and stochastic variability. To achieve this, three novel Generative Adversarial models were developed. The models differ in the architecture of their generators that utilize: a Mixture-of-Experts method, a Latent-Modulated Noise Injection technique, and a Transformer-based architecture respectively. We consider the latter to be the main contribution of this work, and we evaluate our method by comparing it to the other models on both stylized locomotion data and complex, aperiodic dance sequences, assessing its ability to generate diverse, realistic motions, being able to mix between different styles while responding to motion control. Our findings highlight the trade-offs between motion quality, variety and motion generalization in real-time synthesis by comparing by exploring the advantages and disadvantages of each architecture, contributing to the ongoing development of more flexible and varied animation techniques.

KEYWORDS

motion synthesis, GANS, transformers, mixture of experts, deep learning, character control

1 Introduction

As the demand for 3D animation grows across several domains, from video games or virtual reality applications to robotics, there is an increasing need for motion generation techniques that can capture the full range of human motion variability. In this work, we develop a novel probabilistic deep learning architecture in pursuit of generating highly stochastic and varied movements controlled in real-time.

Conditional Generative Adversarial Networks or cGANs (Mirza and Osindero, 2014) are used for their stochastic nature and relatively low computational cost during inference compared to other generative models. We propose a conditional GAN model relying on a transformer-based generator and a discriminator using a Mixture-of-Experts technique to generate realistic yet diverse motion. We show the model's capabilities by training it on highly varied dancing motion data, which deterministic models fail to replicate, as well as different styles of locomotion that we attempt to blend using style mixing. We compare our transformer-based model to two different GANs, each based on well-established and successful architectures: a Mixture-of-Experts (MoE) approach, and a model that injects the modulated latent code directly into the intermediate layers. These models are trained in a subset of the 100STYLE dataset from Mason et al. (2023), which is a comprehensive collection of motion capture information, which consists of 5 distinct styles of locomotion.

The contributions of this work are summarized as follows:

1. We developed a novel generative transformer-based architecture using dot product self-attention for motion generation in real-time without using a latent vector (we use the terms latent vector and latent code interchangeably).
2. Two additional stochastic models, built upon existing successful methods, are used in order to assess our main framework's performance. These different approaches to real-time stochastic motion synthesis, trained alongside the same novel discriminator network, are explored and evaluated in order to compare and achieve a balance between realism and motion variability.
3. A final, hybrid model is developed and explored, utilizing a transformer architecture alongside Latent-Modulated Noise Injection for varied stochastic motion generation.

The models are evaluated using objective metrics, namely Maximum Mean Discrepancy (MMD) introduced by Gretton et al. (2012), which calculates the distance between the probability density functions of the generated data and the training data, and MiVo by Arnout et al. (2021), which calculates the mean of incoming and the variance of outgoing samples to evaluate the realism and variety of produced samples related to the training data. We also evaluate our models using metrics more common to motion matching such as Dynamic Time Warping (DTW), which measures the distance between two temporal signals, and L2 distance between the bones of the generated pose versus the target pose. Finally, a subjective qualitative evaluation of the motion is discussed, as this is the final judge of whether a generated result is satisfactory.

2 Related work

In this section, we discuss different motion synthesis techniques that utilize deep learning. We focus separately on deterministic and probabilistic methods of generating motion and analyze the benefits and detriments of each approach.

2.1 Deterministic motion synthesis

Deterministic approaches to motion synthesis have shown success in producing coherent and controllable character animations. Usually, recurrent networks like LSTMs are used for time-series data like motion. One such example is by Aristidou et al. (2023) which utilizes a three-level framework for dance animation that preserves the global structure and choreography of specific dance genres. The three levels are (1) an auto-conditioned LSTM to generate temporally coherent poses, (2) a motif level that ensures short movements belong to specific motion clusters using a motion perceptual loss, and (3) a choreography level that controls the global structure of the dance by selecting motifs to match the overall dance signature.

However, LSTMs are not without their drawbacks. Despite being effective at capturing temporal dependencies, they tend to suffer from error accumulation over long sequences as noted by Mourot et al. (2022). LSTMs have also shown limited success in controlled motion synthesis in real-time, displaying undesired behavior like foot-skating. Architectures like the Phase-Functioned Neural Network (PFNN) by Holden et al. (2017) overcome this limitation, producing realistic motion, controlled in real-time, by utilizing a simple network that is fed by the character's state for a current frame, and produces the state for the next frame. This method uses a single-phase parameter to modulate network weights, effectively utilizing multiple parallel neural networks that blend their weights to produce varied movements for user-specified tasks. This concept was further developed in the Neural State Machine from Starke et al. (2019), which introduced a Mixture of Experts (MoE) approach, replacing the phase function with a gating network for more adaptive state transitions.

Subsequent advancements, such as Starke et al. (2020), improved motion quality and complexity by introducing separate local motion phase features for each body segment. The phase feature was calculated by the contacts of the character with the environment, using a genetic algorithm in addition to convolutions and low-pass filtering. This approach enabled the synthesis of asynchronous motion for different body parts, significantly enhancing the realism of the produced motion. Further work by the same authors in Starke et al. (2022) changes the phase feature so that it no longer depends on the contacts of the character with the geometry, instead being represented as a multi-dimensional phase space extracted from the character's motion curves.

As highlighted in the work from Loi et al. (2023), while deterministic models offer reliability and ease of training, they often struggle to generate varied character behaviors, potentially resulting in repetitive animations. The same work showcases that probabilistic motion synthesis methods do not suffer from this problem.

2.2 Probabilistic motion synthesis

Probabilistic approaches to motion synthesis aim to generate diverse yet contextually plausible motions, which can result in different movements even with identical input conditions. GANs have been widely adopted in this domain. For instance, the system proposed in Men et al. (2022) uses GANs to synthesize character reactions to another virtual character's motion by utilizing a recurrent network that uses an attention mechanism. Another GAN approach in Mourot et al. (2021) uses a deep convolutional architecture for both its generator and discriminator to enhance the quality of animated skeletal representations in a 2D character as well as predict the locations of joints absent in the original input. A common practice in generative networks is upsampling at multiple stages of the feed forward network of the generator, starting with processing the coarse features and progressively tune the finer details of the output by increasing the resolution of the layers. A prime example is Li et al. (2022), which uses multiple discriminators - one for each resolution of the progressive generator network. This allows the framework to be able to produce realistic motion by training on a single sequence instead of a large dataset.

Impressive results have also been achieved by MoGlow in Henter et al. (2020), a generative and autoregressive controllable motion-data model based on normalizing flows. The model generates highly varied locomotion using a pre-determined path as a control input. This is achieved by iteratively producing the next pose of the character's movement by drawing a random sample from a simple distribution and then passing it through a neural network for a non-linear but invertible transformation. The method uses both auto-regression and a hidden state from an LSTM, which ensures stability and realistic outputs.

Variational Autoencoders (VAEs) have also shown promise in probabilistic motion synthesis. The work in Hassan et al. (2021) aims to synthesize character movement for realistic interaction and navigation in indoor scenes with complicated geometry. The framework consists of three main components: two conditional VAEs, one autoregressive model utilizing a MoE method similar to Starke et al. (2020) for generating stochastic motion sequences, and a second for predicting goal positions and orientations on object surfaces respectively. The third component is a path planning algorithm which computes a navigation mesh from the scene geometry. This system reinforces that MoE methods are very successful with motion generation, even in non-deterministic approaches.

A recent trend in probabilistic motion synthesis is the use of diffusion models. These approaches, such as Tevet et al. (2022), offer great learning capacity and can express complex motion given text or actions as input. Further improvements to this work were made by Karunratanakul et al. (2023) by adding spatial constraints to the motion, for example defining a trajectory, effectively introducing a better-defined and objective level of control, although not in real-time. A similar work using diffusion is by Kulkarni et al. (2024), which uses an object-centric interaction field to guide the diffusion generator's output for automatic complex character-object interactions depending on the object geometry. The process of diffusion consists of several forward passes, making it unsuitable for real-time control, since inference time is usually more than a few seconds long.

Several of the previous approaches, both probabilistic and deterministic, utilize an attention mechanism. Attention layers, mainly popularized for Natural Language Processing (NLP) problems, have shown impressive results in learning complex relationships between vectors, by assigning uni-directional weighted parameters that connect all vectors to each other. Transformer architectures (Vaswani et al., 2017) in deep neural networks exploit the attention layers, alongside fully-connected components, layer normalization and residual connections to produce a meaningful representation of the data without changing the dimensions of the input, making them a powerful tool for encoder or decoder networks.

Due to their increased complexity, probabilistic approaches to motion synthesis are more computationally expensive than deterministic approaches in general, making inference a slow operation. There is a noteworthy lack of approaches that can produce novel movement not present in the training dataset (e.g., mixing different styles of motion from the dataset), and are controlled by the user in real-time, responding to both previous character's states and to user input. This is a limitation that this work addresses, by experimenting with several generative frameworks that generate motions that respond instantly to conditions provided by the user input in a probabilistic manner.

3 Methodology

In this section, we discuss a general overview of our framework, first focusing on the dataset that was used, the pre-processing of the training data, and the network inputs. Then, we analyze our conditional transformer-based GAN framework that was developed for the task of stochastic generation of motion data, alongside the models that are used for evaluation. All generator architectures were trained using the same discriminator model.

3.1 Dataset

The 100STYLE dataset that was chosen for training encompasses more than 4 million frames while the subset we used contains about 170 thousand frames or 90 minutes of footage. The 100STYLE dataset showcases 100 distinct locomotion styles, all performed by a single actor. For each style, the dataset includes several different movements like forward, backward or sidestepping locomotion, as well as walking or running variations. The subset of the dataset that we used consists of 5 styles, namely angry, skipping, and hopping locomotion, as well as movements where the character moves while mimicking a chicken and finally, mimicking an airplane with both arms raised as wings—tilting from side to side.

3.2 Pre-processing and network input

The motion processing module of this work was heavily based on the framework from Starke et al. (2020), and has the role of extracting features for every frame *i* of the raw motion capture files. These features are:

- Root Trajectories X_i^T: the positions, directions and velocities for the hips of the character for the current frame in the XZ plane (the Y axis being the up-direction).
- Action Labels X_i^A: the action labels include states for standing, moving, character speed, left-hand height, and right-hand height–which are all calculated automatically–and the manually set labels for the five styles. Standing and moving actions are floating point variables that always add up to one and depend on the speed of the character root. The speed action calculates the value of the root's speed by dividing the total distance traveled in each time window by the time value, and the hand heights calculate the vertical distance of the hands from the hips.
- Contacts for feet X_i^C: for every frame the contacts for each foot joint with the floor plane are captured and a binary state for each sensor is created.
- Local Phases X_i^P: the local phases are calculated by normalizing, filtering, and then fitting the contacts to a sinusoidal function using a genetic algorithm, in an identical fashion to Starke et al. (2020).

These values, along with the character's state X_i^S, which describes the position, rotation, and velocity for each character

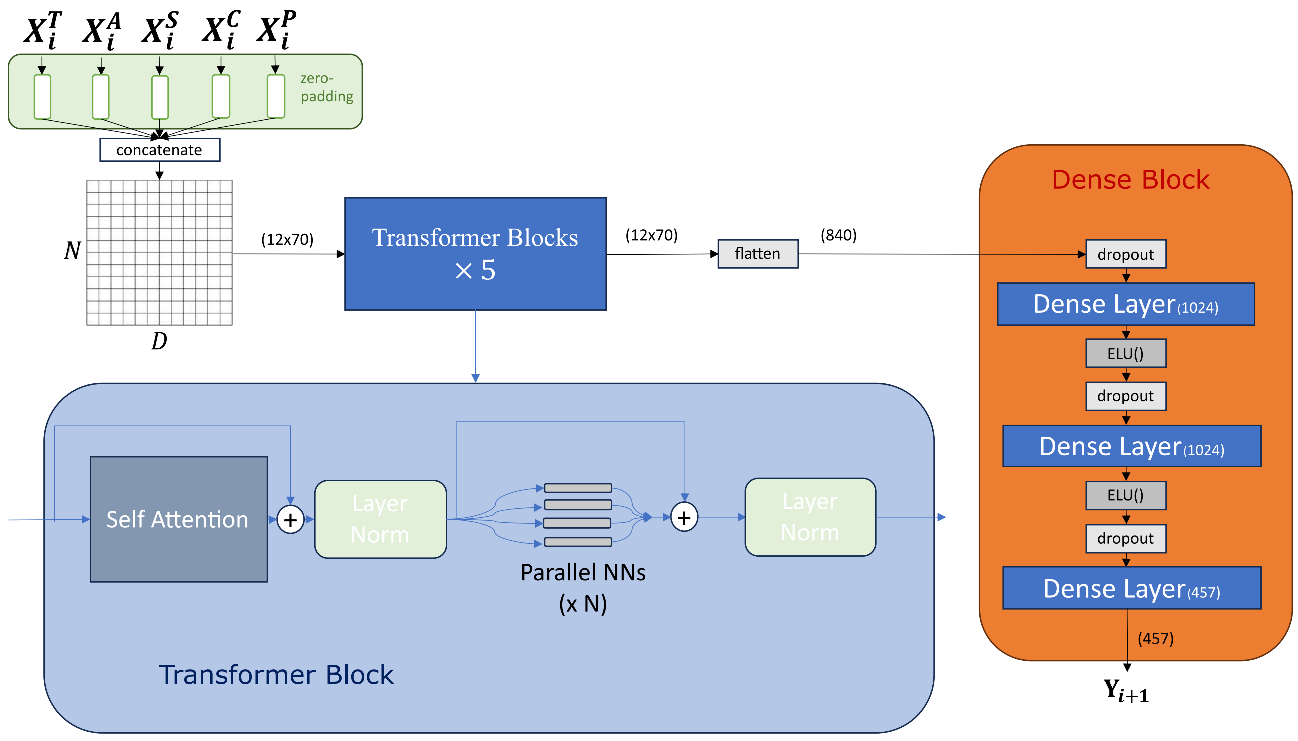

FIGURE 1
The transformer-based generator learns the dependencies between the vectors of the input and passes the transformed tensor to the final dense layers.

joint at frame i are exported in two binary files, as inputs and outputs. The conditional inputs X_i describe the features for a window of 13 frames centered at i. The 13 frames choice was also inherited from Starke et al. (2020) since it balances a relatively small amount of data (necessary for real-time inference) while also being enough to represent a reasonably long time for expressing motion (about 0.5 seconds). The corresponding outputs Y_{i+1} contain the updates for all the same information, for the next frame $i + 1$, without containing information for the past frames since those are not useful for prediction. Using X_i as the condition and Y_{i+1} as real/target samples, our models can be trained to generate synthetic samples $\hat{Y}_{i+1}$ that approach the probability distribution of the real samples for a given condition X_i.

3.3 Transformer generator

Our proposed framework was designed to generate diverse motion in real-time, enabling fast inference while simultaneously having the ability to extrapolate a meaningful understanding of the relationships between input vectors to produce realistic motion. We considered a transformer's architecture using dot product self-attention to be ideal for that purpose. The architecture begins with an input processing stage where various motion features—that consist of root trajectories, action labels, character's states, contacts, and phases—are separated and uniformly formatted. These inputs are zero-padded to create a fixed-size tensor of dimensions $N \times D$, effectively representing N input vectors, each of length D. In the case of our dataset, $D = 12$ and $N = 70$.

At the core of our architecture lie five sequential transformer blocks, each employing self-attention mechanisms with a single head. This design choice allows the model to efficiently capture intricate dependencies between different aspects of motion, enabling a deep understanding of complex movement patterns. The use of transformer blocks represents a departure from previous approaches to real-time motion generation, offering a more flexible and potentially more powerful way to model motion dynamics.

Following the transformer blocks, (as shown in Figure 1) the architecture flattens the output tensor into a single vector. This flattened representation then passes through a series of three dense layers.

3.4 Mixture-of-experts generator

Methods using MoE techniques, in essence utilizing a gating component that uses a meaningful parameter to modulate the weights of the main network, have proven successful for motion synthesis. This method can be thought of as emulating multiple parallel neural networks—dubbed "experts" each specializing in different cases of the gating component's input; this component is typically a multi-class classification neural network, which produces an output activated with a SoftMax function. The gating network's output dictates the blending ratio for each set of expert weights, effectively controlling the degree to which each expert

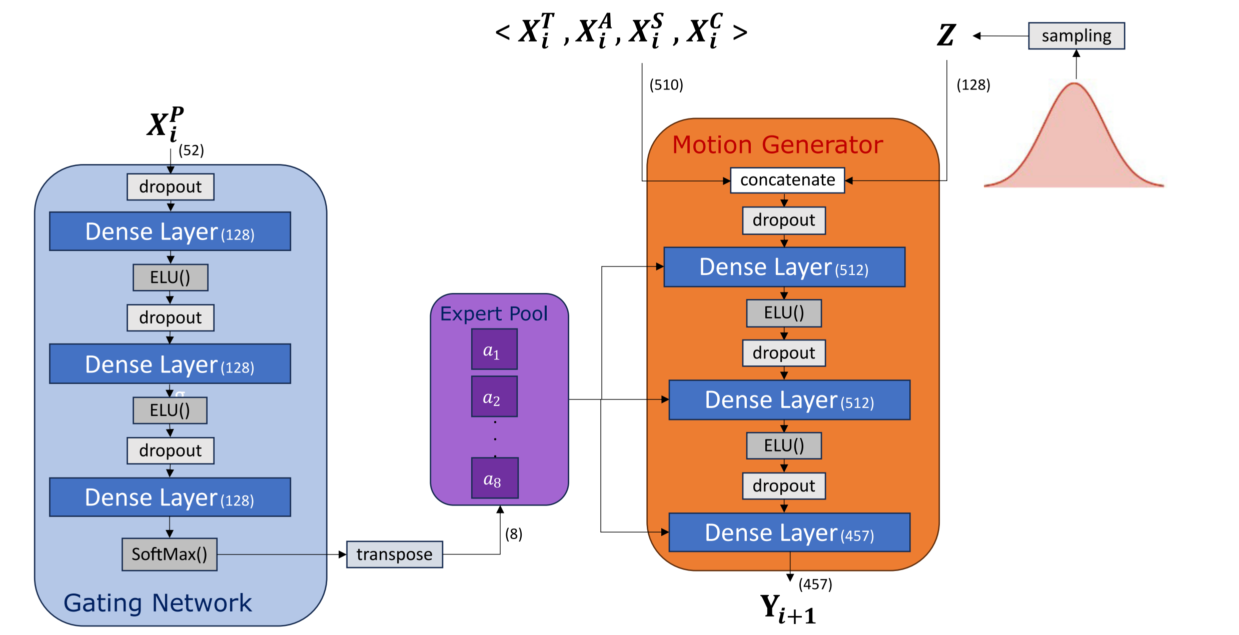

FIGURE 2
The MoE generator utilizes a gating network that uses the phase parameter to decide how much each expert of the motion generator contributes to the produced motion.

influences the final outcome. To simulate multiple experts, the weights of the main component of the neural network usually have an additional dimension, equal to the number of experts.

The MoE generator (Figure 2) we use to evaluate our proposed method, uses a latent vector Z which is sampled from a Gaussian distribution $\mathcal{N}(0, 1)$. This latent vector is what makes this network probabilistic in nature, while the generator's training being based on the discriminator's predictions, by minimizing Binary Cross-entropy loss as showcased in the original GAN paper by Goodfellow et al. (2020) (instead of training directly from a loss function) is the main cause of its generative properties. The vectors $\{X_i^T, X_i^A, X_i^S, X_i^C\}$ are the condition of the generator and are concatenated along with Z to produce the full input of the main component of our generator. The local phase feature X_i^P is the input of the gating network, making it an implicit condition. The gating component itself is deterministic, as we chose to not have the latent code affect it directly, instead having the stochastic factor only affect the motion component. Before each layer, a dropout operation is performed during training to prevent over-fitting.

3.5 LMNI generator

The MoE generator is stochastic in nature, however, the latent vector is a lot smaller than the condition, making a minor contribution to the final output. For our second approach, we opted to increase the stochasticity of our generator, by injecting the latent code (sampled from a normal distribution as before) directly into the generator's layers instead of the input. This is exactly the approach introduced by Karras et al. (2019), creating a very successful framework for image generation. Our second architecture aims to use similar techniques, with a different approach to account for the differences between time series data and images, to produce highly stochastic motion. We refer to this model as our Latent-Modulated Noise Injection (LMNI) approach.

A difficult balance to achieve in our motion synthesis framework is one between stochasticity and cohesion. While movement tha32,t has stochastic characteristics is preferable to purely deterministic motion that has no variation between iterations, the pose Y_{i+1} of a frame must necessarily be highly dependent on the pose of the previous frame (the condition X_i) for the motion to be cohesive. This is why using the condition as the input of the generator network was considered the best solution (Figure 3), instead of it being concatenated with the latent vector. For the same reason, we opted to use dense layers with a constant number of parameters throughout the network, instead of convolutions with upscaling like the original StyleGAN implementation.

The mapping network (consisting of 8 layers) learns to extract a meaningful representation W of the latent code Z which is then appropriately transformed linearly for each layer of the main network. The weights of the main network are controlled by the intermediate latent space W through the AdaIN operation. Before each layer, learned noise scaling is applied to the network to add controlled randomness and stochastic variation to the generated motion. Furthermore, dropout is applied before each dense layer, after the linearly transformed vector W, modulates and normalizes the previous layer's output. The affine transforms' weights are initialized by sampling from a normal distribution while the learned noise scaling weights are initialized to zero.

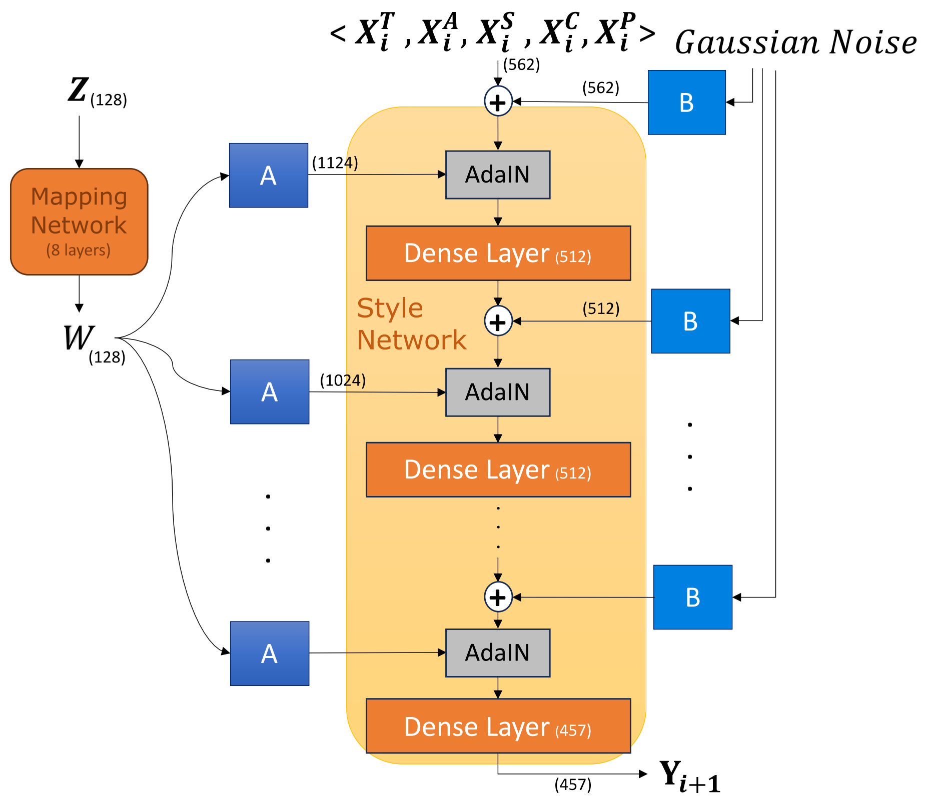

FIGURE 3
The LMNI generator uses a mapping network in addition to affine transforms (notated as "*A*") to modulate each layer separately according to the latent code *Z*. A total of 9 dense layers were used in the main network.

3.6 Hybrid generator

Lastly, we experimented using a novel architecture (Figure 4) inspired both by our main contribution and the LMNI generator inspired by the work in Karras et al. (2019). Our reasoning behind this choice of hybrid architecture was to see the transformer's behavior if a stochastic input element is introduced.

By combining key elements from both the transformer and LMNI architectures, we hope to gain a better understanding of their respective contributions. This approach not only enhances our insight of each component's role but also provides deeper insights into the structural differences and broader implications of various machine learning architectures.

3.7 The discriminator

We designed the discriminator to somewhat mimic the MoE generator's operations by using the phase condition of the GAN to blend the weights of the next layer which takes the motion condition as an input (Figure 5). This layer's output also blends the final layer's weights, this time the input being the motion vector Y_{i+1} which can be a motion vector sampled from the training set, or synthesized by the generator. The discriminator finally performs a 1D Convolution followed by a dense layer using a sigmoid activation function that results in a binary classification output. Using convolutional layers is a traditional operation for discriminator networks and experimentally, it proved necessary for the network to perform the classification task well. The discriminator has a far simpler task than the generator, having to only perform binary classifications instead of synthesizing complex data. This is our reasoning behind choosing to have only one layer for each expert blending operation.

4 Results

4.1 Implementation choices

Our inference environment is built using the Unity Engine (Haas, 2014). During runtime, locomotion is controlled in real-time via the keyboard, allowing the user to adjust movement direction and toggle between running and walking motions (increasing/decreasing the speed by a factor of 3). Motion styles can be customized within the engine environment, supporting floating-point values from 0 to 1, determining the weight for each style. The network has only been trained on data where one style is 1 and the remaining are 0, thus the consequences of intermediate values for the styles during inference are limited and we restrict ourselves to only the edge values for our tests.

All the models were trained for 100 epochs on a desktop computer with an NVIDIA GeForce RTX 3060 with 12GB of video

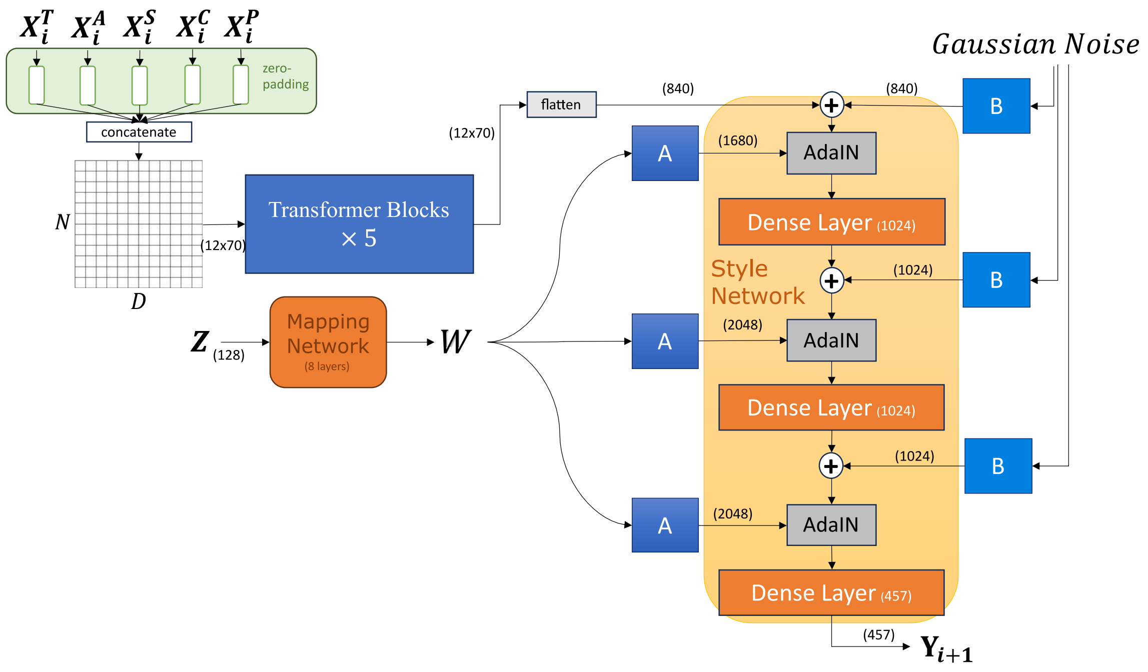

FIGURE 4
The Hybrid generator uses a mapping network for the latent code *Z* similar to the LMNI model, but passes the condition through five transformer blocks in identical fashion to our main generator model. Due to the increased computational needs of the transformer architecture, only two intermediate dense layers were used for the main component (notated as “Style Network”).

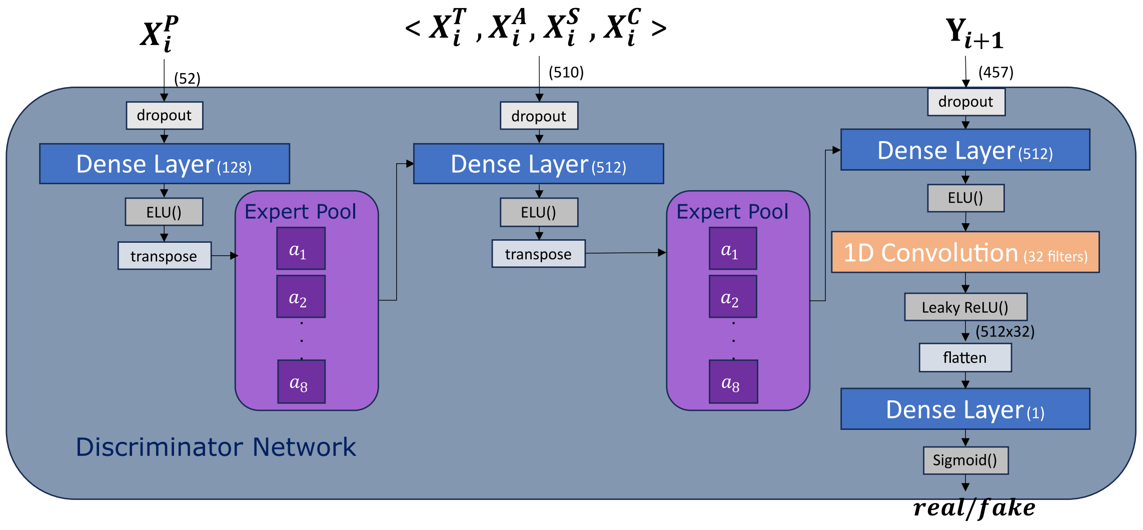

FIGURE 5
The network architecture of the Discriminator utilizes two separate expert blends: one for the conditions similar to the MoE generator, and a second for the motion vector to be classified as real or fake.

memory. The batch size was equal to 128 for both the LMNI and transformer-based model and 64 for the MoE model, due to its increased memory requirement. The generator’s learning rate for all models was initialized to 0.0001, while the discriminator’s learning rate was set lower by a factor of ten. All the architectures were trained with the Adam (Adaptive Moment Estimation) optimization algorithm by Kingma and Ba (2014) using a binary cross-entropy loss.

The dense layers of both the MoE and LMNI main components have a width of 512 units (being the closest power of 2 to the input size), while the Transformer and Hybrid architectures have a width of 1,024 units to accommodate the increased input size. It is generally considered common practice for the intermediate layers to be wider than the input and output layers. The larger width proved to work much better experimentally as well. All layers incorporate both weights and biases. The Transformer

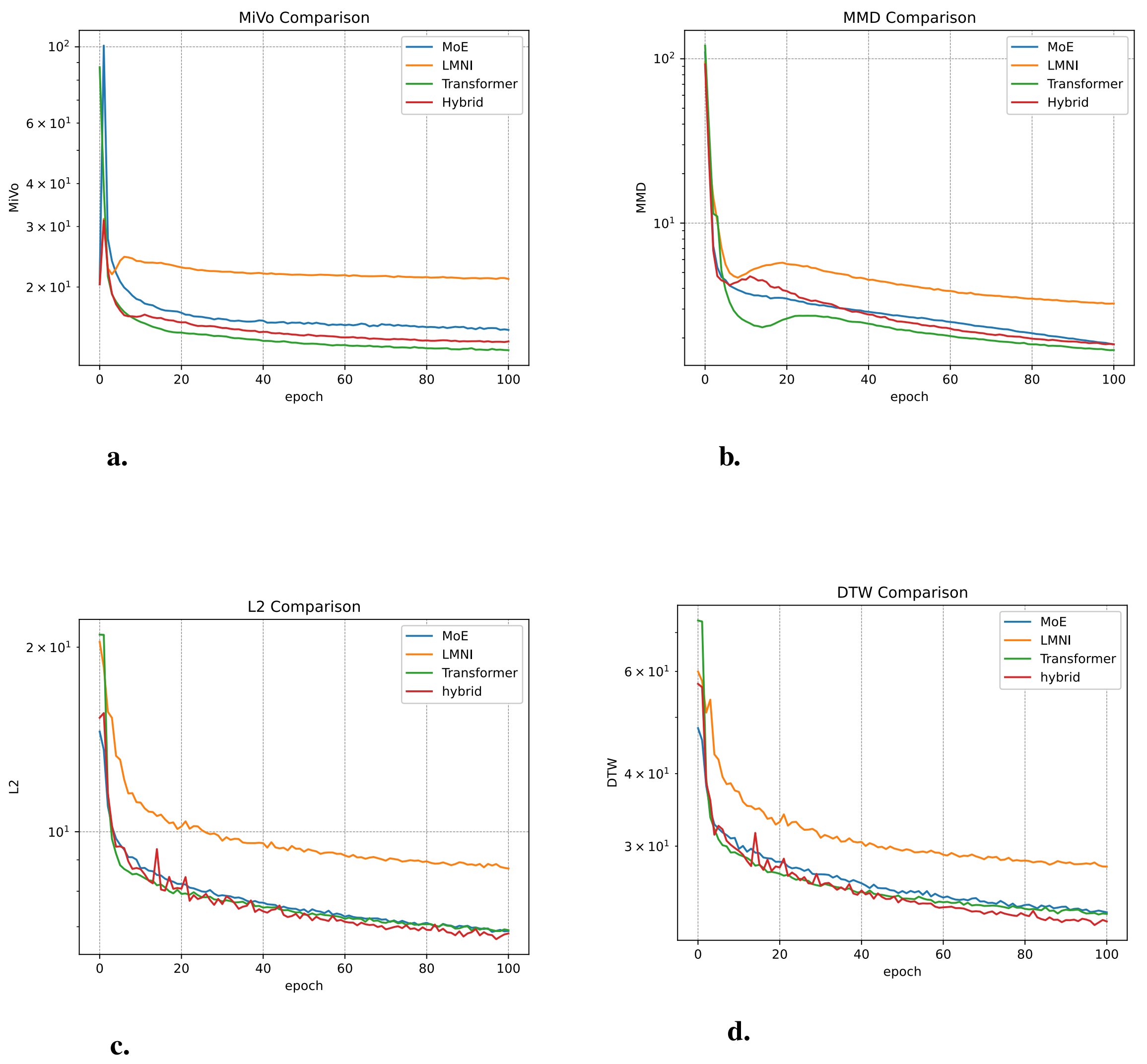

FIGURE 6
(a) The MiVo metric during training for all the GAN architectures on 100 epochs (lower values indicate better results). The LMNI architecture seems to stay relatively flat on this metric in particular, presumably due to its poor quality in finer motion features. **(b)** Mean Maximum Discrepancy for all models tested for 100 epochs of training (lower scores indicate a better performance). All models except the MoE approach, display a slight overshooting in the early epochs before converging, indicating that the generator tends toward mode dropping, before the discriminator learns to better distinguish between the real and generated samples. This hypothesis is also supported by the generator's and discriminator's loss scores (See Appendix). **(c)** L2 distance for the four models on all 100 epochs. The measurements were done after training by inferencing the model for every epoch. The distance is calculated for the pose segment of the output vectors Y^S_{i+1}. Smaller scores mean closer distance between generated and target samples **(d)** Two-dimensional Dynamic Time Warping (DTW) for all models on all 100 epochs. The measurements were done after training by inferencing the model for every epoch. DTW is used on only the vectors that express a future state for the model (Y^T_{i+1}, Y^P_{i+1}). Lower values express closer distance between generated and target samples.

model was designed with five blocks, as experiments demonstrated this configuration to be the most effective without significantly slowing down during inference. Additionally, we tested various discriminator architectures, including those utilizing transformer blocks and convolutional layers. However, across all generator architectures, the MoE discriminator consistently outperformed all attempted alternatives, which is why we chose to keep it for all frameworks. Despite having a similar architecture with the first generator architecture, we noticed no related bias during training.

4.2 Observations

According to the objective metrics MMD, MiVo (See Appendix) the transformer-based model surpasses the performance of all other architectures, with the hybrid model closely behind followed by the MoE method (Figures 6a, b, Table 1). For the distance-measuring metrics (L2 distance and DTW) the results are similar; however, the hybrid architecture achieves slightly lower (i.e., better) scores (Figures 6c, d). Training times—although varied between experiments—tend to be shorter for the LMNI and

TABLE 1 The metrics scores for all architectures after 100 epochs of training.

	MoE	LMNI	Transformer	Hybrid
MMD	1.831	3.241	**1.687**	1.825
MiVo	15.007	21.116	**13.103**	13.890
DTW	23.077	27.664	22.932	**22.257**
L2	6.889	8.712	6.909	**6.825**

The lowest (best) score for each metric is displayed in bold text.

TABLE 2 The timings for training all architectures.

	MoE	LMNI	Transformer	Hybrid
Training time	23h 14m	**10h 6m**	10h 10m	10h 45m
Inference time	**37ms**	50ms	42ms	53ms

The fastest times for inference and training are displayed with bold text. The inference time is calculated by taking the average inference time from 60 seconds of total runtime.

Transformer architectures (Table 2), as the MoE method uses a lot more parameters. The MMD score seems to overshoot in the first few epochs for most networks, indicating a tendency toward mode dropping in the early training. This may result in lower (i.e., better) values in the MMD metric but not necessarily in the MiVo score, since the latter also takes sample variety into account. The pure LMNI architecture, while producing highly stochastic results, falls behind the rest in producing realistic motion. For the subjective qualities of the resulting animation, apart from Figure 7, please watch the accompanying video for a more comprehensive demonstration. Some basic observations about the different architectures are the following:

- The MoE model produces realistic motion while retaining stochastic characteristics, although it requires a larger amount of video memory and was generally shown to train slower than the other approaches. The balance between the generator and the discriminator also was found to be hard to maintain during training, especially for smaller batch sizes.
- The LMNI approach results in highly stochastic movements but falls short of realism, instead producing artifacts such as foot skating. This method proved to be comparatively lightweight for training, but performance during inference is not optimal due to its increased depth requirements and the time-consuming noise injection process using the AdaIN operation.
- The transformer-based model produces both realistic motion and is capable of combining different styles of locomotion specified by the user, to produce realistic samples that strongly differ from any examples present in the training set, which we consider the main contribution of this work. This approach is also relatively computationally inexpensive, making it faster to train and to implement during inference.
- The hybrid model, using elements of both the transformer and the LMNI approach is very stochastic in nature, and by far the most stable to train. However, it does not converge to as low a minimum as the pure transformer approach. The motion produced is more realistic than the LMNI model, but still suffers from occasional foot-skating and an inability to smoothly transition between different styles.

4.3 Ablation studies

Some ablation studies were performed to test the robustness of our framework.

- We opted to remove the transformer blocks entirely, to see the performance of just the dense layers, and assess the transformer blocks' contribution.

  Surprisingly, even without the transformer blocks, the network can score very close to the original contribution. However, the final produced motion is lacking in energy, fails in style-mixing, and is stiff-looking in general. Due to the attention mechanism of the transformer blocks, it is better suited to encode the intricate relationships between the vectors describing the motion. We believe this is why the transformer blocks are required for the ability to mix styles.

  We theorize that the two generators score similarly as the metrics used may not be able to entirely capture perpetual quality, especially not fine-grained features and variations in motion. It is also worth noting that a single sample only contains about 0.5 seconds of temporal data, so this might be another contributor to the metrics failing to capture the transformer's contribution in this instance.
- We ran experiments using different latent vector sizes on the MoE generator to evaluate the differences in the final result. Specifically, we experimented with a vector of size 256, and a lack of a latent vector altogether.

  All architectures scored similarly in the objective metrics. However, the model without a latent vector, while generating relatively realistic motion, exhibited repetitive patterns, particularly noticeable in recurring artifacts during locomotion cycles (such as unnaturally prolonged durations in certain phases). The architecture with a larger latent vector (256) produced more energetic motion, but was more prone to unnatural angles of the joints and was subjectively considered to be the least realistic of the three. It is important to note that training the network is not a deterministic process, meaning that multiple runs with the same hyperparameters may lead to slight differences in final scores.
- The mapping network was discarded in order to confirm its contribution to the LMNI framework.

  Without the mapping network, with the latent vector being linearly transformed and fed directly into the network, the LMNI generator is entirely unable to train. Our hypothesis is that this is due to the GAN model suffering from mode collapse, due to the unstructured nature of the unmapped latent vector.

5 Discussion and limitations

Going into further detail, as is shown in Figure 7, the generator with the transformer architecture generally has a better structured motion, showing very few unnatural artifacts. In the rightmost

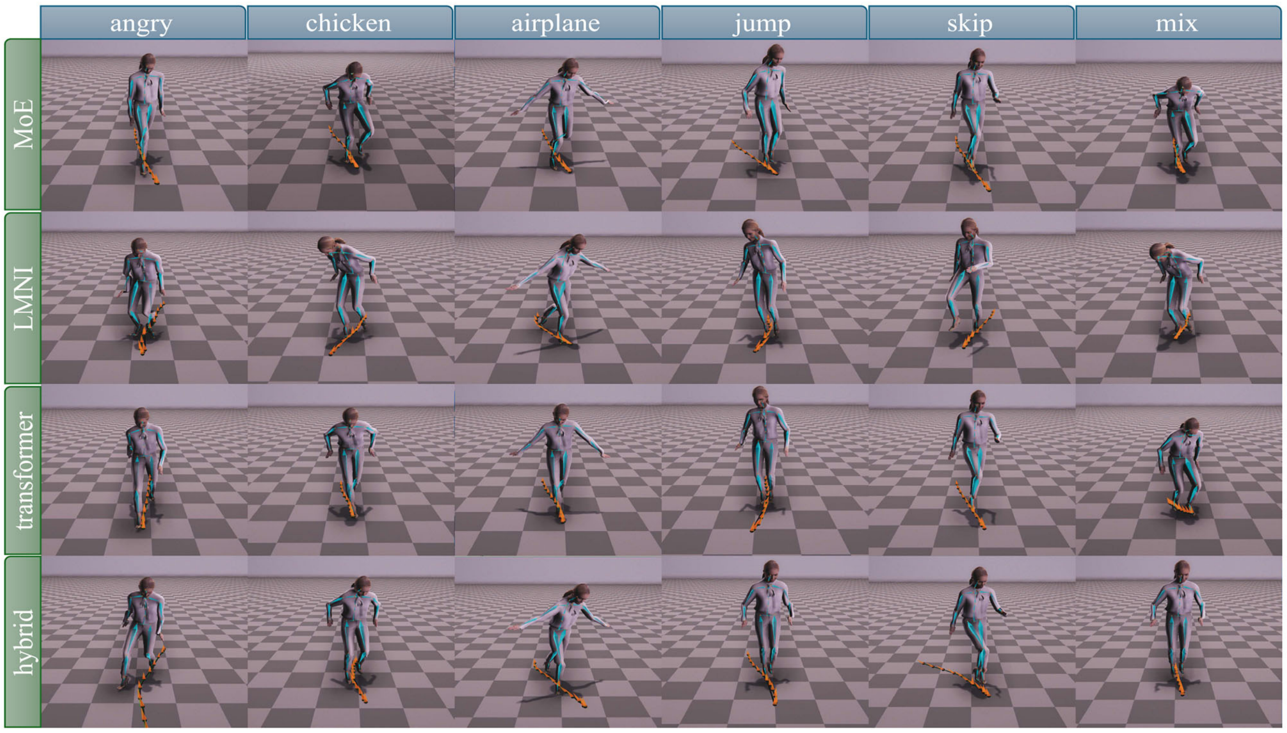

FIGURE 7
Samples from every motion style including a mix of styles (chicken and jump) by all architectures for comparison. The rows indicate the architecture while the columns describe the style of motion. We can see that the MoE and transformer networks produce relatively similar poses in regards to realism. The LMNI architecture is expressive, but often results in unrealistic or exaggerated poses. One notable feature of the networks utilizing a transformer, is their ability to produce the mixed motion instead of replicating only one of the styles unlike the MoE and LMNI architectures.

column, it is also notable that only the transformer and hybrid models can mix between styles, while the first two architectures can only arbitrarily choose between one of the two styles. As can bee seen in the figure, the LMNI architecture seems the most expressive of the four. However, in the accompanying video the motion appears less refined than the rest, with fluctuations in speed and unnatural transitions between different styles. Some slightly unnatural joint positions can also be seen in the MoE network, however the motion is quite smooth, producing an overall realistic result as is indicated in the objective metric scores. Finally, in the figure, we see that most networks had a difficult time replicating running motion in the style of the first column, tending to prefer walking motion instead. This was a general tendency in all of our experiments. The transformer surpasses the rest of the networks in that regard, indicating its capability of generating samples that better fit the condition.

Our results are also compared with similar deterministic methods for motion generation in real-time. Specifically, we trained the framework from Starke et al. (2020) with the same dataset for 100 epochs. The regressive model produces livelier samples, that are generally considered more realistic and can seamlessly transition between styles. However, unlike our transformer-based model, it fails to mix between styles, as is showcased in the accompanying video and Figure 8.

Some additional qualitative tests were performed on the frameworks of Starke et al. (2020), Li et al. (2022), and Henter et al. (2020) respectively, alongside with our framework as is showcased in Table 3. We also experimented with training our transformer model on a single sequence (Figure 9), however, the motion produced is relatively stiff and unvaried, unresponsive to control inputs, and scores much lower on objective metrics than GANimator (Li et al., 2022), a model specialized for this task, which however does not offer motion control, or generation in real-time, unlike our proposed framework.

Since the transformer-based architecture does not utilize a latent vector to generate motion, it can be trained as a regression model using Mean Square Entropy loss in a similar fashion to Starke et al. (2020). The experiment produced relatively realistic motion but failed to surpass the original regressive MoE method lacking its vigor and fast, energetic movements. The models were also trained on several Greek and Cypriot traditional dances from the Dance Motion Capture Database (Stavrakis et al., 2012) of the University of Cyprus. The dataset consists of aperiodic, highly energetic and impulsive dance sequences. The motion of the dataset proved to be a real challenge for the networks, which either failed to produce realistic results or generated a small periodic subset of the motion of the dataset, indicating mode collapse. A possible workaround would be to examine an alternative pre-processing method for the phase feature similar to Starke et al. (2022).

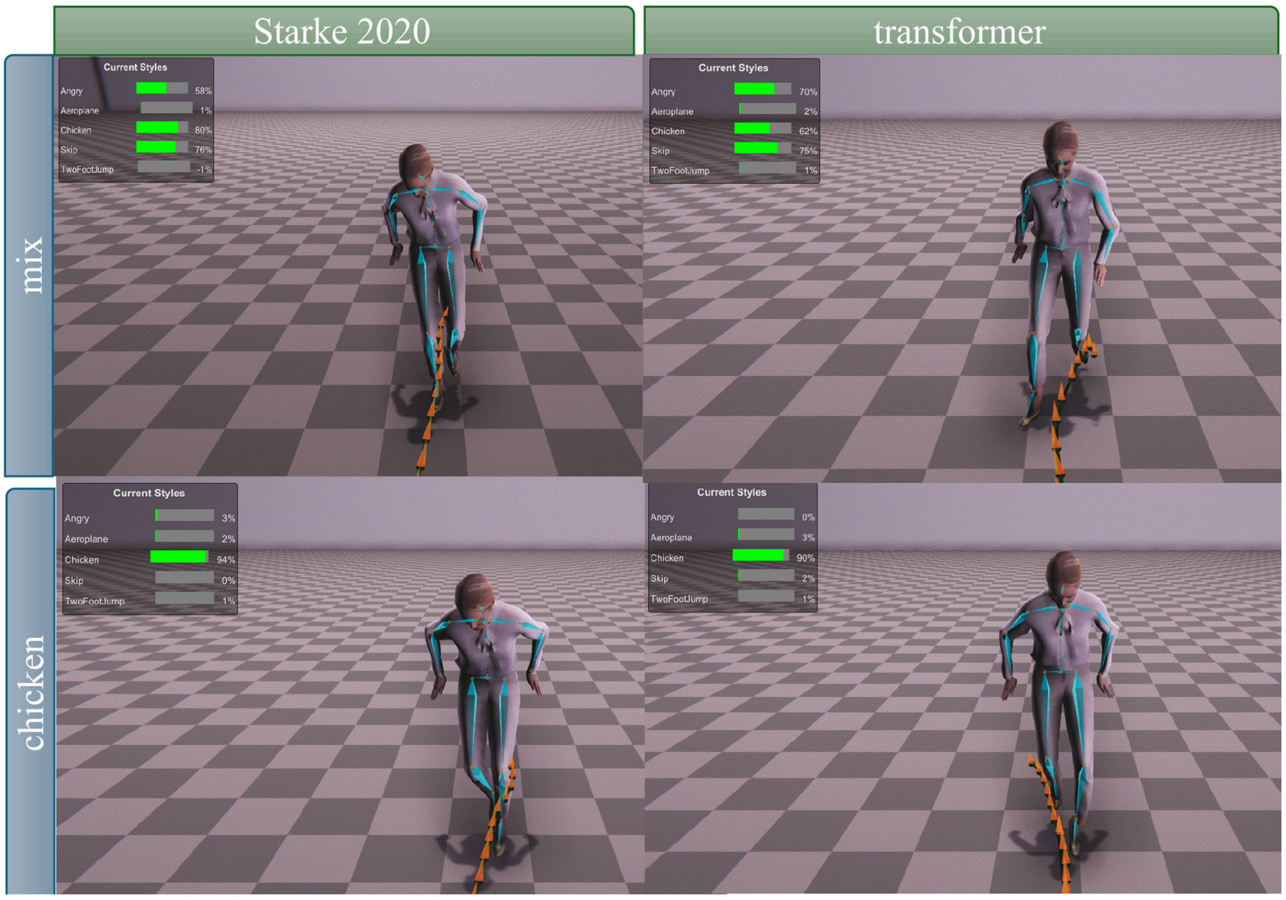

FIGURE 8
Comparison between the transformer network and the framework in Starke et al. (2020). Our transformer model is capable of mixing between three different styles of motion, producing a relatively novel set of poses, while the motion produced by the deterministic framework replicates only the "chicken" style. It is important to note that, during mixing, all three styles were assigned a value of 1. However, since the network was trained with only one active style per sample, it tends to predict lower values when multiple styles are present, resulting in the lower percentages seen in the "Current Styles" sliders. These values help showcase how our network distributes style influences and contribute to a better understanding of its behavior.

TABLE 3 Qualitative comparison between some motion synthesis frameworks that were tested to evaluate our model's performance.

	Real-time generation	Style-mixing	Single-sequence training	Controllable
Ours	✓	✓	✗	✓
GANimator	✗	✓	✓	✗
Starke 2020	✓	✗	✗	✓
MoGlow	✗	✓	✗	✓

The frameworks are compared on their ability to perform certain tasks, thus showcasing the unique capabilities of our own GAN model.

6 Conclusions and future work

We developed a novel GAN architecture, utilizing dot product self-attention transformers for generating plausible and stochastic motions that are controlled in real-time. We compare our model against two other architectures based on Mixture of Experts, and Latent Modulated Noise Injection respectively. We explored the advantages and disadvantages of each architecture and showed that the transformer-based architecture surpasses the other methods in metric scores and, more importantly, is able to combine different styles of motion to produce novel movements not present in the training data. A limitation our method is its inability to generate realistic motion for highly aperiodic and varied motions such as traditional dances, and its failure to train on limited data (e.g., a single motion sequence) highlighting the strong case-specific use of artificial neural network architectures.

In future work, we aim to generalize the use of this network for pure motion prediction, given only the previous state of the character, as well as explore different pre-processing frameworks to achieve realistic and varied results regardless of the nature and periodicity of the motion provided in the training data. We also aim to investigate training on larger sets of data, in order to be able

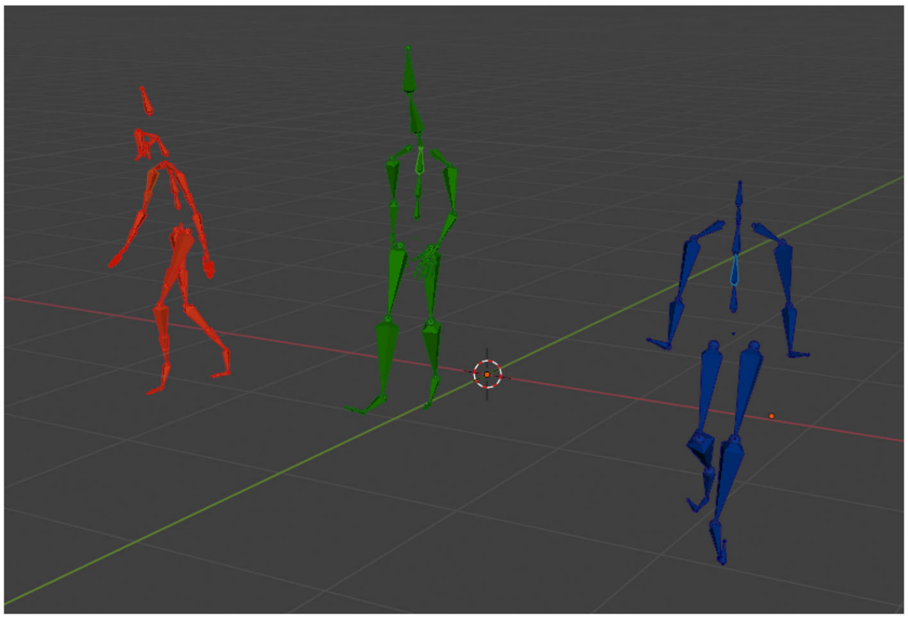

FIGURE 9
Comparison between the training sequence (green armature), the output recorded from our framework (red) and the framework of Li et al. (2022) (blue). Both frameworks were trained on just the training sequence, but our model scored worse on objective metrics, indicating an inability to perform well on specialized tasks such as training on limited data.

to synthesize our own styles of locomotion using a latent vector, generated by compressing the style of the training data.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: 100STYLE dataset: https://www.ianxmason.com/100style. Dance Motion Capture Database of the University of Cyprus: http://dancedb.cs.ucy.ac.cy.

Author contributions

AB: Conceptualization, Software, Writing – original draft, Writing – review & editing, Methodology. IL: Conceptualization, Supervision, Writing – review & editing. KM: Project administration, Supervision, Writing – review & editing.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. The research project is implemented in the framework of H.F.R.I call Basic research Financing (Horizontal support of all Sciences) under the National Recovery and Resilience Plan Greece 2.0 funded by the European Union -NextGenerationEU (H.F.R.I. Project Number: 16469).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Gen AI was used in the creation of this manuscript.

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fcomp.2025.1549693/full#supplementary-material

References

Aristidou, A., Yiannakidis, A., Aberman, K., Cohen-Or, D., Shamir, A., and Chrysanthou, Y. (2023). Rhythm is a dancer: music-driven motion synthesis with global structure. *IEEE Trans. Vis. Comput. Graph.* 29, 3519–3534. doi: 10.1109/TVCG.2022.3163676

Arnout, H., Bronner, J., and Runkler, T. (2021). "Evaluation of generative adversarial networks for time series data," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–7. doi: 10.1109/IJCNN52387.2021.9534373

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2020). Generative adversarial networks. *Commun. ACM* 63, 139–144. doi: 10.1145/3422622

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *J. Mach. Learn. Res.* 13, 723–773. doi: 10.48550/arXiv.0805.2368

Haas, J. K. (2014). *A History of the Unity Game Engine.* Technical report. Worcester, MA.

Hassan, M., Ceylan, D., Villegas, R., Saito, J., Yang, J., Zhou, Y., et al. (2021). "Stochastic scene-aware motion prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 11374–11384. doi: 10.1109/ICCV48922.2021.01118

Henter, G. E., Alexanderson, S., and Beskow, J. (2020). Moglow: probabilistic and controllable motion synthesis using normalising flows. *ACM Trans. Graph.* 39, 1–14. doi: 10.1145/3414685.3417836

Holden, D., Komura, T., and Saito, J. (2017). Phase-functioned neural networks for character control. *ACM Trans. Graph.* 36:3073663. doi: 10.1145/3072959.3073663

Karras, T., Laine, S., and Aila, T. (2019). "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4401–4410. doi: 10.1109/CVPR.2019.00453

Karunratanakul, K., Preechakul, K., Suwajanakorn, S., and Tang, S. (2023). "Guided motion diffusion for controllable human motion synthesis," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2151–2162. doi: 10.1109/ICCV51070.2023.00205

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *CoRR, abs/1412.6980.*

Kulkarni, N., Rempe, D., Genova, K., Kundu, A., Johnson, J., Fouhey, D., et al. (2024). "Nifty: neural object interaction fields for guided human motion synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 947–957. doi: 10.1109/CVPR52733.2024.00096

Li, P., Aberman, K., Zhang, Z., Hanocka, R., and Sorkine-Hornung, O. (2022). Ganimator: neural motion synthesis from a single sequence. *ACM Trans. Graph.* 41, 1–12. doi: 10.1145/3528223.3530157

Loi, I., Zacharaki, E. I., and Moustakas, K. (2023). Machine learning approaches for 3d motion synthesis and musculoskeletal dynamics estimation: a survey. *IEEE Trans. Vis. Comput. Graph* 30, 5810–5829. doi: 10.1109/TVCG.2023.3308753

Mason, I., Sarkar, A., Sasaki, T., and Boix, X. (2023). Modularity trumps invariance for compositional robustness. *arXiv preprint arXiv:2306.09005.*

Men, Q., Shum, H. P., Ho, E. S., and Leung, H. (2022). Gan-based reactive motion synthesis with class-aware discriminators for human–human interaction. *Comput. Graph.* 102, 634–645. doi: 10.1016/j.cag.2021.09.014

Mirza, M., and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784.*

Mourot, L., Clerc, F. L., Thébault, C., and Hellier, P. (2021). "Jumps: joints upsampling method for pose sequences," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 1096–1103. doi: 10.1109/ICPR48806.2021.9412160

Mourot, L., Hoyet, L., Le Clerc, F., Schnitzler, F., and Hellier, P. (2022). A survey on deep learning for skeleton-based human animation. *Comput. Graph. Forum* 41, 122–157. doi: 10.1111/cgf.14426

Starke, S., Mason, I., and Komura, T. (2022). Deepphase: periodic autoencoders for learning motion phase manifolds. *ACM Trans. Graph.* 41:3530178. doi: 10.1145/3528223.3530178

Starke, S., Zhang, H., Komura, T., and Saito, J. (2019). Neural state machine for character-scene interactions. *ACM Trans. Graph.* 38:3356505. doi: 10.1145/3355089.3356505

Starke, S., Zhao, Y., Komura, T., and Zaman, K. (2020). Local motion phases for learning multi-contact character movements. *ACM Trans. Graph.* 39:3392450. doi: 10.1145/3386569.3392450

Stavrakis, E., Aristidou, A., Savva, M., Himona, S. L., and Chrysanthou, Y. (2012). "Digitization of cypriot folk dances," in *Proceedings of the 4th International Conference on Progress in Cultural Heritage Preservation, EuroMed'12* (Berlin, Heidelberg: Springer-Verlag), 404–413. doi: 10.1007/978-3-642-34234-9_41

Tevet, G., Raab, S., Gordon, B., Shafir, Y., Cohen-Or, D., and Bermano, A. H. (2022). Human motion diffusion model. *arXiv preprint arXiv:2209.14916.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). "Attention is all you need," in *Advances in Neural Information Processing Systems*, 30.

Appendix

MiVo

The MiVo metric (Arnout et al., 2021) was designed specifically for evaluating GANs that generate time series data. It can evaluate the performance of a GAN by taking in account both the similarity between the two series, as well as the diversity of the samples produced. This is achieved by first computing the matrix that denotes the distances d_{ij} between the two sets of time series S_g and S_r:

$$D = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \dots & d_{nn} \end{bmatrix} \quad \text{(A1)}$$

A row of the matrix indicates the distance between a generated series in the set S_g and every series of S_r, while a column is the distance between a real-time series and every generated series. MiVo can be finally calculated as follows:

$$MiVo(Sr, Sg) = \mu(D_1) + \sigma^2(D_2) \quad \text{(A2)}$$

where D_1 and D_2 are the set of minimal distances over the rows and columns of D respectively. By keeping the mean value of D_1 low, we can surmise that the generated outputs are realistic as they remain close to at least one real-time series of Sr. Similarly, having the variance of D_2 low, means every real-time series must be close to at least one corresponding generated sample, thus ensuring that the generator produces diverse data, that covers as much of the training set as possible.

MMD

Maximum Mean Discrepancy is equal to the following:

$$MMD(X, Y) = ||\mu_X - \mu_Y|| \quad \text{(A3)}$$

where μ_X and μ_Y are the embedded mean values of the probability distributions in the Reproducing Kernel Hilbert Space (RKHS) H. Since we don't have access to the underlying probability distributions of the data, we can map any sample to the RKHS using a kernel function. In this feature space, we can calculate the distance between the embedded means of the two mapped sample sets. The larger this distance, the bigger the difference between the two distributions. For this application we use the gaussian kernel function: $k(X, Y) = e^{-||X-Y||^2}$. Given samples $x = \{x_0, x_1, ..., x_n\}, y = \{y_0, y_1, ..., y_n\}$ belonging to the distributions X and Y respectively, the MMD between the two distributions can be computed as:

$$MMD^2(X, Y) = \frac{1}{m^2}\sum_{ij}^{\infty} k(x_i, x_j) + \frac{1}{n^2}\sum_{ij}^{\infty} k(y_i, y_j) - \frac{2}{mn}\sum_{ij}^{\infty}(x_i, y_j) \quad \text{(A4)}$$

GAN training loss

In this section the balance between the generator and discriminator are discussed during training. A strong correlation between a large batch size and stability between the generator and discriminator was noticed, as with a lower batch size, all models would diverge sooner in the training. This can also be noticed in Figure A1, as the MoE network, which is the only one with a smaller batch size (64 instead of 128), was shown to be the more unstable of the four. Another noteworthy observation is that all of our tests, including different discriminator architectures, ended in the discriminator loss decreasing and the generator's lost increasing over time, meaning that after a sufficient amount of training, all discriminator architectures managed to tell apart generated samples from real ones.

As with Goodfellow et al. (2020), all of our GAN networks are trained by minimizing Binary Cross-entropy loss, described by the equation:

$$\mathcal{L}_{\mathrm{BCE_{GAN}}} = \mathbb{E}_{Y\sim p_{\mathrm{data}}(Y)}[\log D(Y|X)] + \mathbb{E}_{Z\sim p_Z(Z)}[\log(1 - D_Y(G(Z|X)))]$$

where the functions $D(Y|X)$ and $G(Z|X)$ represent the process of the discriminator and generator respectively.

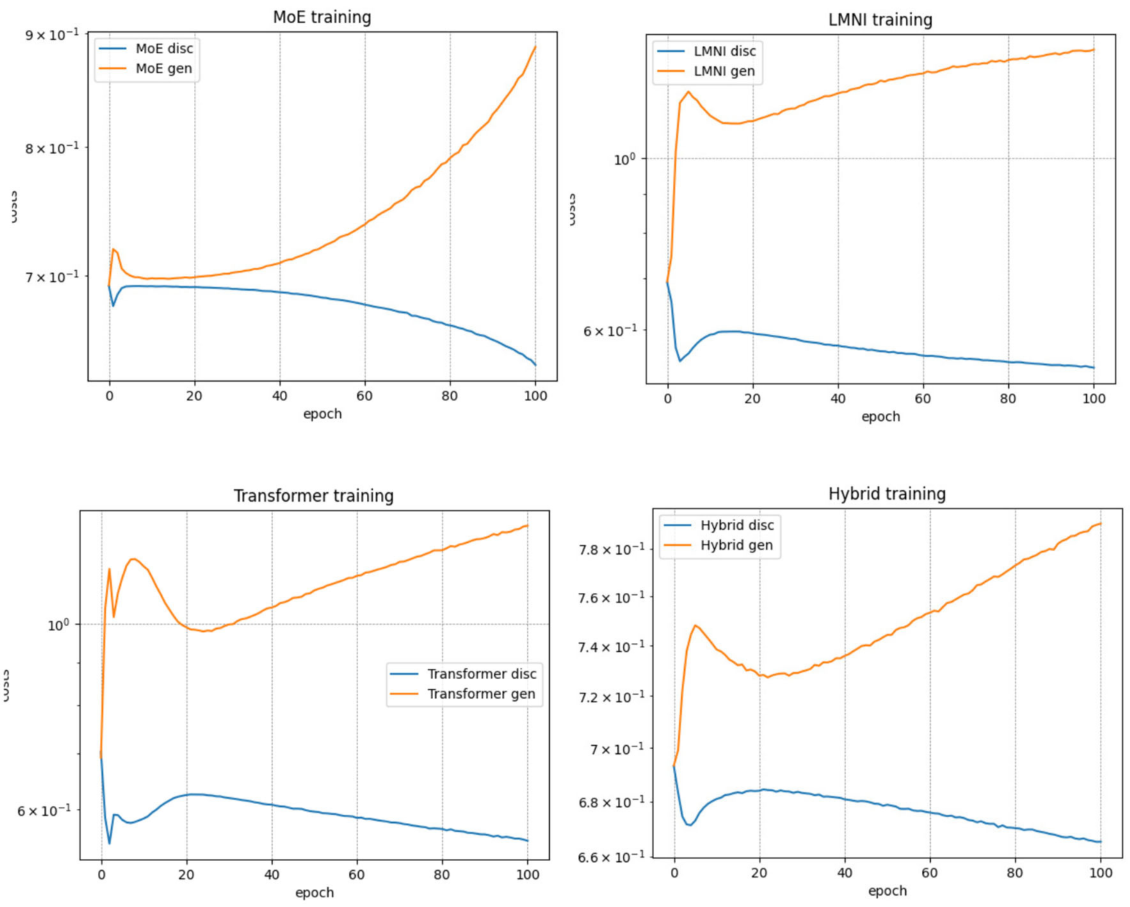

FIGURE A1
The costs for the generators and discriminators across all of our frameworks during the training of 100 epochs.