MoDi: Unconditional Motion Synthesis from Diverse Data Supplementary Materials

Sigal Raab [1] Inbal Leibovitch [1] Peizhuo Li [2]
Kfir Aberman [3] Olga Sorkine-Hornung [2] Daniel Cohen-Or [1]

[1] Tel-Aviv University [2] ETH Zurich [3] Google Research
sigal.raab@gmail.com

1. Outline

This Supplementary adds details on top of the ones given in the main paper. While the main paper stands on its own, the details given here may shed more light.

In Sec. 2 we provide more details regarding our model; considerations that led us to choose 3D convolutions and edge rotation representation, a description of skeleton-aware models in existing works, and a recap on the StyleGAN architecture. Sec. 3 describes the $\mathcal{W}+$ space used for inversion, and the computation of the *gral* score, used for latent motion editing. In addition, it provides a qualitative and quantitative comparison with applications in other works. Lastly, in Sec. 4 we elaborate on our experiments; we describe the datasets that we use, provide implementation details such as hyper-parameters, detail metrics, and show additional ablation and additional qualitative results. In particular, we demonstrate how conditional networks can be fine-tuned based on a one-time unconditional training, and present quantitative and qualitative results to support this approach. Finally, we provide a comparative analysis of unconditional methods.

2. Model – Additional Details

2.1. Structure-aware Neural Modules – Additional Details

In this section we first describe our motivation for using 3D convolutions; then, we depict the way these convolutions are used by our networks. Finally, for information completeness, we describe skeleton-aware neural modules from existing works and the way we use some of them in a 3D convolutional setting.

3D convolutions – motivation Described below are two ways to design the filters of *skeleton-aware* convolutions [2]. Recall that E, T, ℓ, and U denote the number of entities and frames, the hierarchical level index, and the kernel width, respectively.

- Existing works use 3D filters of dimension $(K_{\ell+1} \cdot E_{\ell+1}) \times (K_\ell \cdot E_\ell) \times U$. Such filters are applied by the neural model using 1D or 2D convolutions, over the time axis or time and joint-channels axes, respectively.
- Our work uses 5D filters of dimension $K_{\ell+1} \times K_\ell \times E_{\ell+1} \times E_\ell \times U$. These filters are applied by the neural model using 3D convolutions, over the time and joint axes, and an additional axis that holds the output joints, to be described next.

The convolutions in existing works combine the joints and the channels into the same dimension, yielding a non-intuitive representation that adds complexity to coding. For example, the output joints are received as channels, and require reshaping to be represented as joints. It is common practice in most neural architectures to hold a dedicated dimension for the channels. Moreover, 3D filters introduce complications when combined with the StyleGAN algorithm, for two distinct reasons:

1. StyleGAN uses modulation, which is difficult to apply if the channels and the joints share the same dimension, as the style is injected to each channel separately (see Sec. 2.3). By using 3D convolutions, *i.e.* 5D filters, we place the channels in their own dedicated dimension, so modulation becomes simple.
2. StyleGAN uses transposed convolutions, in which the axes are swapped such that the output and input channels switch places. Managing such a swap becomes straightforward when the channel dimensions are separated from the joint dimensions.

Note that it *is* possible to keep using 3D filters as done in other works. However, such usage, combined with StyleGAN's components, adds complexity (multiple data reshapes, weights reshapes, and dimension swaps).

3D convolutions – details Figure 1 describes the way in which our networks use 3D convolutions. As explained in the main paper, we dedicate separate kernels to each joint. To convolve each separate kernel with the data, we use different dimensions for the input and output joints. The output joints axis is created by expanding the data by one dimension followed by zero padding, such that sliding the filters along the new axis enables using different weights for each output joint. Once convolution is completed, the result holds the joints data in the output joint axis, and the input joint axis becomes degenerated (of size 1) and is removed.

Recap existing neural modules The modules described here are skeletal in-place convolution and skeletal pooling [2, 8, 22, 24]. In our work, we create a 3D version of the skeletal in-place convolutional filter and replace the skeletal pooling by our novel convolutional scaler filter.

In Fig. 2 we show a skeletal pooling procedure. Pooling is done by averaging the features of two entities, hence, it is equivalent to a convolution with weights of 0.5. Our new filter applies a convolution with learned weights, generalizing the pooling functionality, and allowing the network the freedom to choose the optimal weights.

In Fig. 3 we depict our 3D version of a skeleton-aware convolutional filter. Unlike our novel convolutional scaler filter, this filter is an in-place one, which means it retains the dimensions of its input, and cannot scale it.

2.2. Motion Representation Considerations

Some methods generate a sequence of 3D poses [8], where each location is specified by the 3D coordinates of each joint. However, the resulting representation is incomplete, since it does not reflect a rotation of a bone around its own axis. In particular, it does not contain all the information necessary to drive a rigged virtual 3D character, and the temporal consistency of the skeleton's bone lengths is not guaranteed. While joint rotations may be recovered from joint positions via inverse kinematics (IK), the solution is not unique and thus ill-posed. Furthermore, models that predict positions tend to yield a temporally jittery output and require a post-processing smoothing stage. Due to these considerations, we follow numerous recent works that are based on joint rotation representation [15, 16]. The motion generated by MoDi can be directly converted into an animation sequence without the need to apply either IK or temporal smoothing.

Our network is trained on a single set of bone lengths. Once a motion is generated, it can be retargeted to any other set of bone lengths using existing motion retargeting methods [2, 3, 5].

2.3. Generative Network Architecture in Detail

In this section, we provide further details regarding the architectural building blocks of MoDi. Some of the descriptions recap StyleGAN [14] and are given here for information completeness.

Generator In Fig. 4 we show additional details related to the motion generator. In particular, we depict the usage of modulation and demodulation [14], which has been shown to be safer compared to AdaIN [12] in terms of certain artifacts. The AdaIN block processes *data*, namely normalizes it and applies a new standard deviation. The modulation/demodulation block performs an equivalent (in expectation) operation on the *weights*. Let u denote a weight value within a filter, and let i, j, and k denote the input channel index, output channel index, and filter spatial index, respectively. Instead of multiplying the data by a new standard deviation, we *modulate* the weights:

$$u'_{ijk} = s_i \cdot u_{ijk}, \tag{1}$$

and instead of normalizing the data, we *demodulate* the weights:

$$u''_{ijk} = u'_{ijk} \Big/ \sqrt{\sum_{i,k} {u'_{ijk}}^2}. \tag{2}$$

Discriminator Our discriminator, as well as its role in the training procedure, is depicted in Fig. 5. Our discriminator holds the reverse architecture of the synthesis network. That is, it receives a generated or real motion and processes it in neural blocks that gradually decrease in size. Like the motion synthesis network, our discriminator is based on structure-aware neural modules. In each hierarchical level, the skeletal topology becomes coarser using skeletal convolutions.

Losses The generative network is trained with several losses. Our main loss is adversarial. In addition, we regularize the generator with foot contact and with path length and regularize the discriminator with $R1$. All losses except for foot contact are used by StyleGAN too, and for completeness, we describe them here.

Adversarial loss We train our GAN with a non-saturating adversarial loss [9],

$$\mathcal{L}^G_{adv} = - \mathop{\mathbb{E}}_{z\sim\mathcal{Z}} \left[\log D(G(z))\right], \tag{3}$$

$$\begin{aligned} \mathcal{L}^D_{adv} &= - \mathop{\mathbb{E}}_{m\sim\mathcal{M}_{nat}} \left[\log D(m)\right] \\ &= - \mathop{\mathbb{E}}_{z\sim\mathcal{Z}} \left[\log(1 - D(G(z)))\right]. \end{aligned} \tag{4}$$

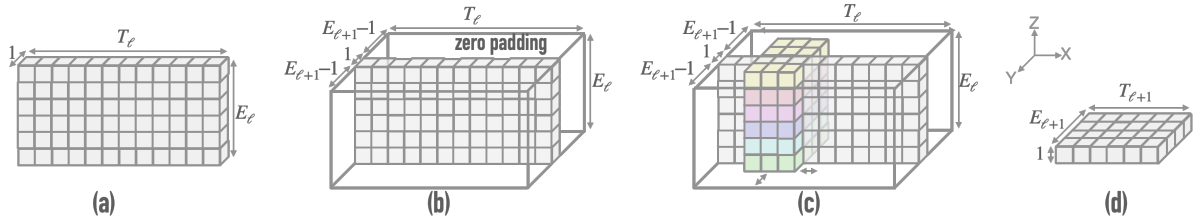

Figure 1. Down sampling with our new filter (depicted 3D out of 4D, no channels): (a) Data at hierarchical level ℓ: Dimensions are $K_\ell \times E_\ell \times T_\ell$. We expand it by one dimension in preparations for 3D convolution. (b) Level ℓ data is further padded by zeros, and its new dimensions are $K_\ell \times (2E_{\ell+1}{-}1) \times E_\ell \times T_\ell$. (c) 3D convolution: The filter is slid within the data block. Sliding is along the x and y axes only, as the z axis' filter height is identical to the data height. (d) Resulting data: The extra dimension of size 1 is dropped such that final dimensions at level $\ell + 1$ are $K_{\ell+1} \times E_{\ell+1} \times T_{\ell+1}$.

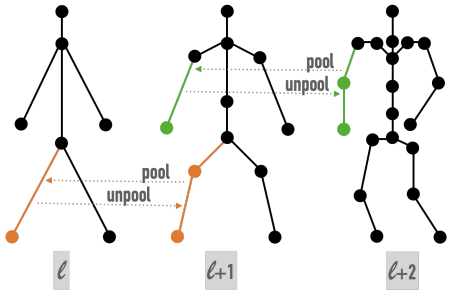

Figure 2. Skeletal pooling, not used by MoDi. The pooling operation merges two adjacent edges and removes the joint between them. The unpooling operation splits an edge into two, and adds a joint between the newly created edges. We denote skeletal hierarchy levels with $\ell, \ell + 1, \ell + 2$, and demonstrate pooling and unpooling on selected joints in orange (levels $\ell, \ell{+}1$), and in green (levels $\ell + 1, \ell + 2$).

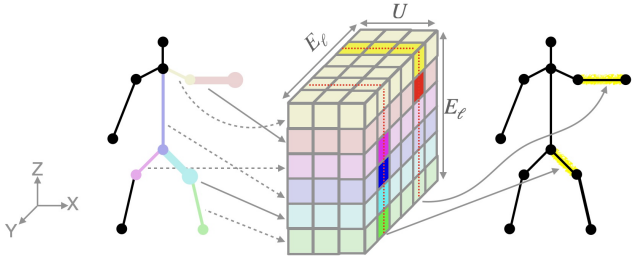

Figure 3. Our 3D version of a skeleton-aware in-place convolutional filter. Each horizontal slice (xy plane) is related to one entity in the input character (left), and each vertical slice (xz plane) is related to one entity in the output character (right). Each entity in the output character "sees" only weights related to its neighboring entities, emphasized with saturated colors in the filter. We demonstrate convolutions on the left thigh and on the left forearm, marked yellow in the output character. Note that each of these entities is affected by its immediate neighbors and ignores entities that do not neighbor it. Our filter is 5D and since we can only visualize 3D, we omit the channels. Recall that E, ℓ, and U denote the number of entities, the hierarchical level index, and the kernel width, respectively.

Path length loss This loss [14] requires that a fixed-size step in $\mathcal{W}$ results in a non-zero, fixed-magnitude change in the generated motion.

$$\mathcal{L}^G_{path} = \mathop{\mathbb{E}}_{w\sim\mathcal{W},r\sim\mathcal{R}} \left[\left\|J_w^T G(w) * r\right\|_2 - a\right]^2, \qquad (5)$$

where $\mathcal{R}$ is a unit Gaussian space normalized by the number of joints and frames, $J_w = \partial G(w)/\partial(w)$, and a is the accumulated mean gradient length.

R1 loss This loss [17] improves the functioning of the discriminator:

$$\mathcal{L}^D_{R1} = \mathop{\mathbb{E}}_{m\sim\mathcal{M}_{nat}} \left[\|\nabla_m D(m)\|_2^2\right]. \qquad (6)$$

foot contact loss The foot contact losses, $\mathcal{L}^G_{tch}$ and $\mathcal{L}^G_{fcon}$ are described in the main paper.

Altogether, the generator and discriminator losses are

$$\mathcal{L}^G = \mathcal{L}^G_{adv} + \lambda_{tch}\mathcal{L}^G_{tch} + \lambda^G_{fcon}\mathcal{L}^G_{fcon}, \qquad (7)$$
$$\mathcal{L}^D = \mathcal{L}^D_{adv}. \qquad (8)$$

We activate the regularizations $\mathcal{L}^G_{path}$ and $\mathcal{L}^D_{R1}$ in a lazy fashion, as done by Karras *et al*. [14].

2.4. Encoder – Description of $\mathcal{W}+$

Our inversion method uses the $\mathcal{W}+$ space, which is an expansion of the latent space $\mathcal{W}$, proposed by Abdal *et al*. [1]. A tensor in $\mathcal{W}+$ is a concatenation of several different $w \in \mathcal{W}$ vectors, one for each layer of the synthesis network. Each vector in $\mathcal{W}+$ is used as a modulation input to a different layer. In contrast, when using $\mathcal{W}$, the same w vector is used for all layers. Abdal *et al*. [1] show that $\mathcal{W}$ is limited and an inversion from arbitrary images is

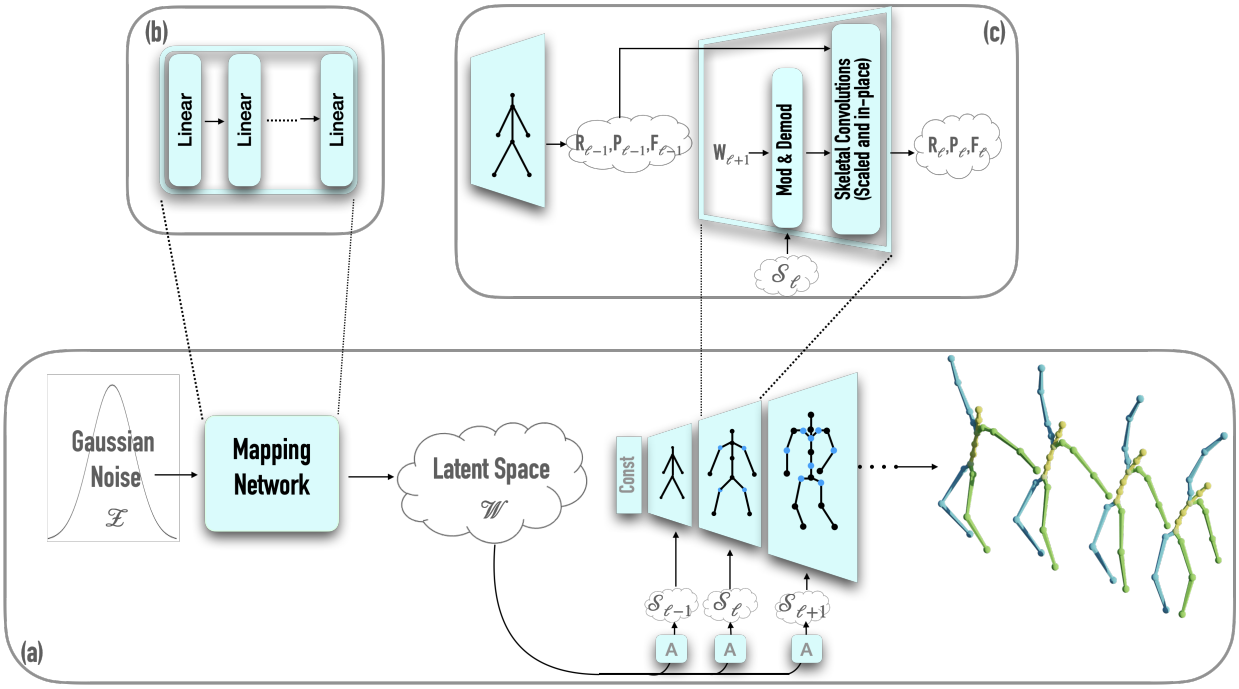

Figure 4. Our motion generator in detail. (a) Adding style injection information to the architecture depicted in the main paper. $\boxed{A}$ denotes a learned affine transformation. This transformation is applied on the latent code w to produce a style code $\mathcal{S}_\ell$, where ℓ is the hierarchical level index. A different style code is injected to each layer. (b) Zoom in on the mapping network, which is an MLP with several linear layers. (c) Zoom in on the motion synthesis network, where a style code $\mathcal{S}$ modulates the layer's weight. The styled weight is then used for a transposed convolution of the layer features. Recall that $\mathbf{R}_\ell$, $\mathbf{P}_\ell$ and $\mathbf{F}_\ell$ denote the features in level ℓ of rotations, root positions and foot contact labels, respectively. A transposed skeletal convolution applies the modulated weights on the data features from the previous (coarser) hierarchical level. Since the convolution is transposed, it results with larger dimensions, both in the temporal axis and in the joints axis.

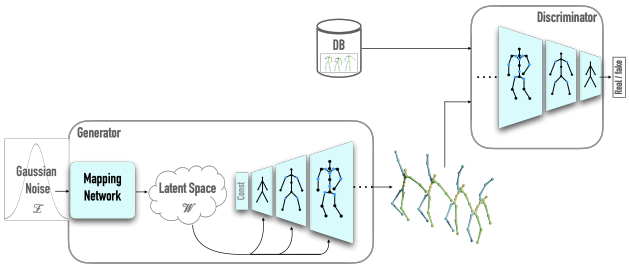

Figure 5. Our discriminator holds the reverse architecture of the synthesis network. It receives a generated or real motion, and learns to output whether the input motion is real of fake. Using structure-aware neural modules, in each hierarchical level the skeletal topology becomes coarser and the number of frames is divided by 2.

much more accurate when using $\mathcal{W}+$. In our experiments, we have witnessed that this approach works for the motion domain as well.

3. Applications – Additional Details

3.1. Latent Interpolation – Additional Details

In this section we elaborate on interpolation in the latent space, referred to in the main paper. Figure 6, also shown in the main paper, is copied here so we can further describe it.

Let $\bar{w}$ be the mean of all $w \in \mathcal{W}$, and let *mean motion* denote $G(\bar{w})$, the motion generated by it. The mean motion is depicted at the bottom row of Fig. 6(a). This motion is similar for all variations of trained networks and is what we intuitively expect: an idle standing, front-facing character.

We demonstrate the linearity of the latent space $\mathcal{W}$ by interpolating between the latent values and observing the motions generated out of the interpolated values. A special case, called *truncation*, is when the interpolation target is $\bar{w}$. In the imaging domain, truncation has an important role in regularizing out-of-distribution images. We show that truncation works well in our model too. A truncated sequence is denoted by $w_i = \hat{w} + \frac{i}{C}(\bar{w} - \hat{w})$, where $\hat{w} \in \mathcal{W}$, C is the number of interpolation steps, and $i \in [0 \dots C]$. Clearly $w_0 = \hat{w}$ and $w_C = \bar{w}$. We can replace $\bar{w}$ by any sampled $\tilde{w} \in \mathcal{W}$, and then the sequence is called interpolated rather

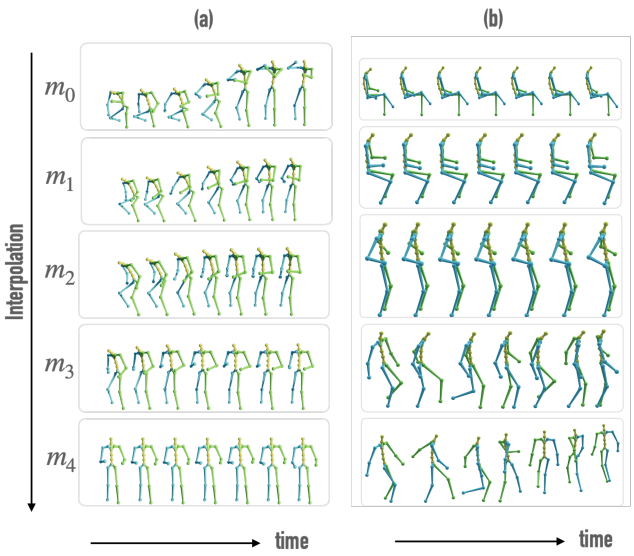

Figure 6. Interpolation in the latent space: (a) interpolation to the mean motion (truncation); (b) From sitting to walking: Note the gradual change from crossed legs sitting, to regular sitting, standing, small-step walking, and regular walking. Each motion is natural, despite interpolating between sitting and standing, which cannot be achieved by interpolating between joint values.

than truncated. Let $m_i = G(w_i)$ denote the motion generated out of each w_i. Fig. 6 (a) and (b) shows the motions created out of truncation and interpolation, respectively.

We observe favorable characteristics in all interpolation sequences. First, m_i is semantically similar to m_{i-1}, but it also changed towards the semantics of the target m_C. When dealing with truncation, m_i is always milder than m_{i-1}.

Second, we notice that the interpolation is between whole sequences rather than frames. For example, if in m_{i-1} the character jumps occasionally, then in m_i the character jumps in a similar frequency, but unnecessarily on the same frames.

Lastly, there are no unnatural motions in the sequence, although using simple geometric joint interpolation would have resulted in unnatural motions. Fig. 6(b) demonstrates this, where our latent interpolation yields natural motions at all stages. A näive geometric interpolation of edge rotations would result in an abnormal pose between sitting to standing, with a vertical spine (see supplementary video).

For comparison, Fig. 8 (top) demonstrates interpolation between "sit" to "walk" in ACTOR's [18] latent space. As can be seen, the resulting intermediate motion is "lift dumbbell", questioning the linearity of their latent space.

3.2. Computing the *gral* Score

Our classifier computes the *gral* (gradual right arm lifting) score in the following way. Let $m = [R, S, F]$ be a selected motion. Recall R represents the rotation angles of the motion. Let $R_{rs,t}$ and $R_{re,t}$ denote the rotations of the right shoulder and the right elbow at time t, respectively. Let $[R_{rs,t}, ..., R_{rs,t+8}]$ be a temporal window of size 8. A similar window is created for R_{re}. We compute the average angle in each window and slide the window with a stride of 4. Altogether we get the average computed $T/4$ times for both the right shoulder and the right elbow. Denote the sequence of average angles by α_{rs} and α_{re}. The next step is to compute the difference between each element to the one preceding it and obtain

$$score_{rs_i} = \begin{cases} 1, & \text{if } \alpha_{rs_i} > \alpha_{rs_{i-1}} \\ 0, & \text{otherwise} \end{cases}, \tag{9}$$

$$score_{re_i} = \begin{cases} 1, & \text{if } \alpha_{re_i} > \alpha_{re_{i-1}} \\ 0, & \text{otherwise} \end{cases}, \tag{10}$$

where $i \in [1, T/4 - 1]$.

Clearly, if all scores are one, the arm is going up, and if they are all zero, the arm is going down. The average of all the values in the two score vectors is used as the final attribute score.

	SNR ↑
Aberman *et al.* [2]	18.89
Ours	16.16

Table 1. Quantitative results for the denoising application. Our results are comparable to Aberman *et al*.

3.3. Spatial Editing – Qualitative Comparison

To qualitatively compare our spatial editing application with other works, we run the spatial editing process on state-of-the-art ACTOR [18]. In Fig. 8 (bottom) we manually raise both hands and lower them at mid-motion, yielding a non-continuous unnatural motion. When passing the edited motion through ACTOR's autoencoder, the result is unfaithful to the given motion. This is in contrast to the smooth and natural results of MoDi, shown in the main paper.

3.4. Denoising – Quantitative Comparison

To evaluate the performance of our encoder in the denoising task, we construct a 4:1 training-to-validation split and re-train our model only on the training set. We analyze the test set by adding Gaussian noise to a given motion m, resulting in m'. Then, $G(I(m'))$ is applied to all motions in the test set, resulting in denoised motions. Finally, we compute the mean signal-to-noise ratio (SNR) between the ground truth and the denoised motions. To assess the performance of our method, we compare our SNR results with the ones computed for Aberman *et al*. [2] on the same training-to-validation split. As can be seen in Tab. 1, the obtained SNR values are comparable.

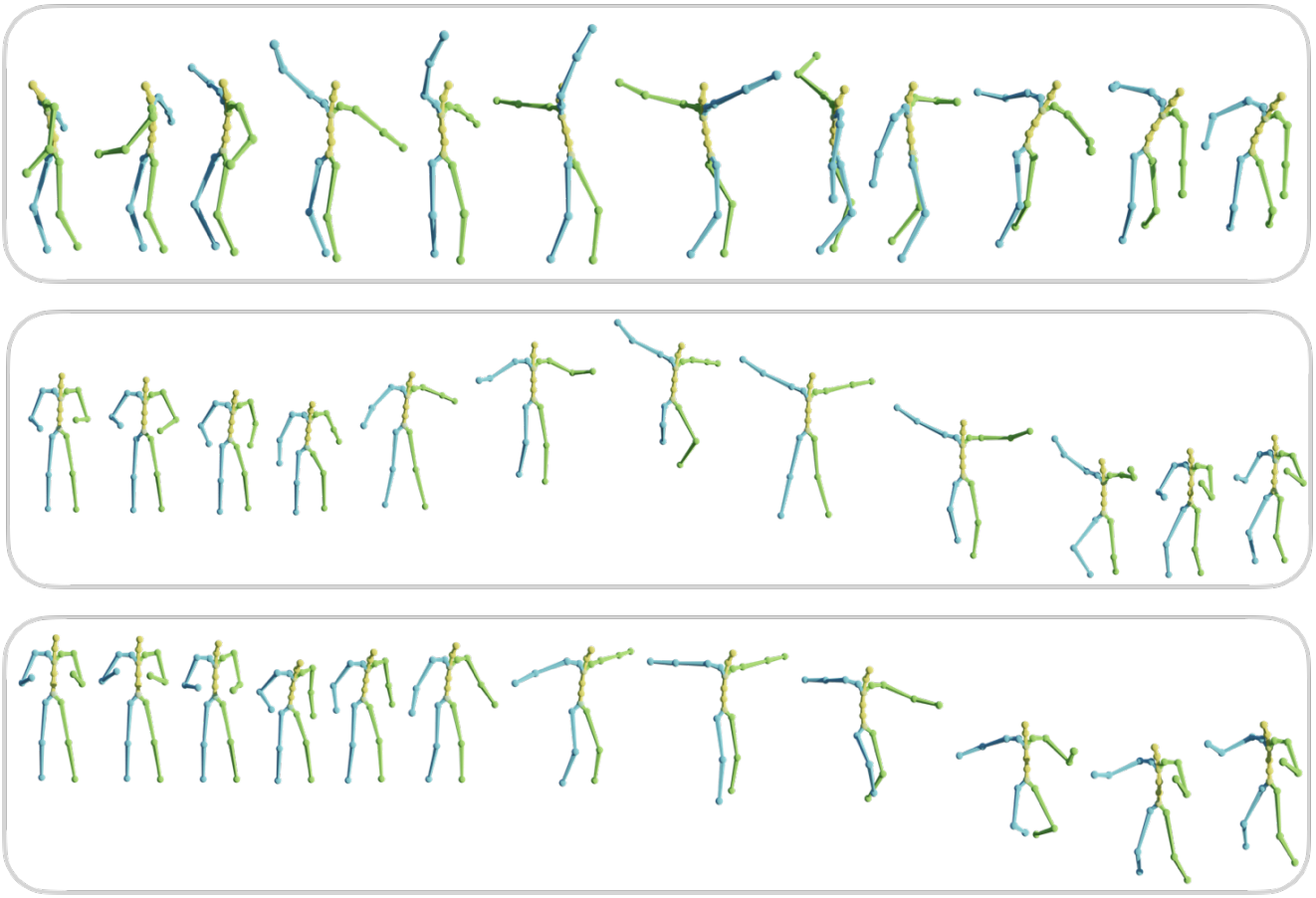

Figure 7. Qualitative results. The top motion depicts a synthesised wild dance, and the next two show synthesised jumps, illustrating the diversity in semantically related motions. All motions are unconditionally generated. See more results in the supplementary video.

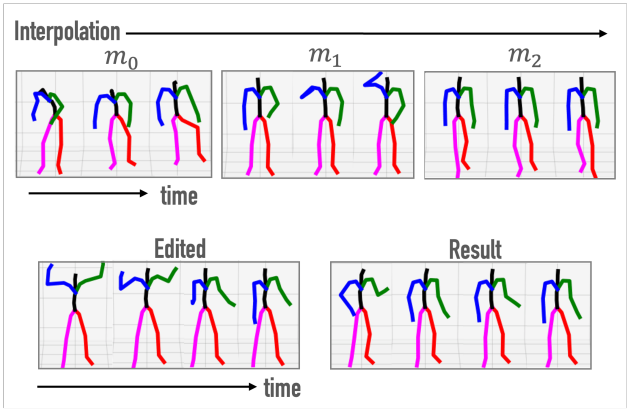

Figure 8. Qualitative comparison of our applications with state-of-the-art ACTOR [18]. Top: latent space interpolation between a sitting motion and a walking one. Bottom: spatial editing. Our applications (see main paper) outperform ACTOR, highlighting the effectiveness of our approach.

4. Experiments – Additional Details

4.1. Datasets

Mixamo – training and evaluation We construct our 3D motion dataset using the Mixamo [4] 3D animation collection, which contains approximately 2500 extremely diverse motions that are not constrained by any set of categories. These motions are applied to 70 characters. Examples of the motions in the dataset are elementary actions (jumping, walking), dance moves (samba, hip-hop), martial arts (boxing, capoeira), acrobatics (back/front flips, acrobatic jumps), and non-standard motions (running on a wall, flying).

We generate our data samples by first extracting the relevant edges from each motion (*e.g.*, we drop the fingers). Then we crop each motion to partially overlapping sequences of frames, hence increasing the amount of data.

HumanAct12 – evaluation HumanAct12 [10] is not as diverse as Mixamo and offers approximately 1200 motion

Name	Hierarchy level	channels × joints × frames
Generator - Motion Synth. Net.	0	256 × 1 × 4
	1	128 × 2 × 8
	2	64 × 7 × 16
	3	64 × 12 × 32
	4	32 × 20 × 64
Discriminator	0	32 × 20 × 64
	1	64 × 12 × 32
	2	64 × 7 × 16
	3	128 × 2 × 8
	4	256 × 1 × 4

Table 2. Architecture: Dimensions of all hierarchy levels.

Name	Neural building blocks
Generator - Motion Synth. Net.	Skeletal Conv. Scaler (upsample) Skeletal Conv. (in-place) Skeletal Convl (in-place)
Discriminator	Skeletal Conv. (in-place) Skeletal Conv. (in-place) Skeletal Conv. Scaler (downsample) Add Residual

Table 3. Architecture: Building blocks in hierarchical levels. Skeletal operators are based on [2].

clips, organized into 12 action categories and 34 subcategories. Due to its small number of motions, we use HumanAct12 for quantitative comparison only.

UESTC – training and evaluation We use this dataset for the conditional setting (Sec. 4.5) only. It contains 25K sequences that span 40 categories of actions, with a primary focus on exercises and some featuring circular motions.

4.2. Hyper-parameters and Training Details

In this section, we describe the details of the network architectures. Tab. 2 describes the architecture of our generator and discriminator networks. The sizes of the kernels are configurable by hyper-parameters, and in the table we specify which hyper-parameters we have used for our best model. Note that the number of joints varies according to the topology of the skeleton on which the network is trained. The values in Tab. 2 belong to the skeleton used by the model presented in this work. The structure of each hierarchical level in our generator and discriminator is described in Tab. 3. A hierarchy level in the motion synthesis network contains input/output skips, and a hierarchy level in the discriminator contains a residual skip, both based on Karras *et al*. [14].

In our experiments, we use $\lambda_{fcon} = 1$, $\lambda_{touch} = 0.01$, batch size 16, learning rate 0.002 for both generator and discriminator, mixing 0.9, and train for 80,000 iterations. We use pytorch version 1.5.0, and CUDA version 10.1 on a GeForce GTX 1080 Ti GPU.

4.3. Quantitative Metrics

Our metrics build upon the latent features of an action recognition model. However, training such a model on Mixamo is challenging, as there are no action labels in it.

Our approach to this challenge is interdisciplinary. Mixamo has textual labels, and using the Sentence-BERT [19] NLP model, we attain latent features representing the textual characteristics of each motion. Then we use K-means to cluster the embedding, and use each cluster as a pseudo-action label. With action labels at hand, we train an action recognition model – STGCN [23]. The features extracted from this trained model are then used for metrics calculation. We randomly sample 2000 motions for calculating metric scores on the Mixamo dataset. We draw 1000 motions for scores on the HumanAct12 dataset since it is a lot smaller.

Following is a brief description of each metric used for the quantitative results.

FID Frèchet inception distance is the distance between the feature distribution of generated motions and that of the real motions, namely the difference in mean and variance. Despite its simplicity, FID is an important metric widely used to evaluate the overall quality of generated motions [10, 18]. FID is borrowed from the image domain, where the inception network is used for features. To adjust this metric to the motion domain, we replace the inception with an action recognition network. A lower value implies better FID results.

KID Kernel Inception Distance (KID), proposed by Binkowski *et al*. [6], compares skewness as well as the values compared in FID, namely mean and variance. KID is known to work better for small and medium size datasets. Lower values are better.

Precision and Recall These measures are adopted from the discriminative domain to the generative domain [20]. Precision measures the probability that a randomly generated motion falls within the support of the distribution of real images, and is closely related to fidelity. Recall measures the probability that a real motion falls within the support of the distribution of generated images, and is closely related to diversity. Higher precision and recall values imply better results.

Loss \ Error	Reconst. (L2)	Reconst. (L1)	Global root position (mm)	Local position (mm)	Global position (mm)
all	<u>.293</u>	<u>.316</u>	59.0	**20.4**	<u>78.2</u>
w/o $\mathcal{L}^I_{fcon}$	**.271**	**.293**	<u>57.0</u>	<u>21.2</u>	**76.0**
w/o $\mathcal{L}^I_{pos}$	.300	.321	**47.0**	47.7	93.7
w/o $\mathcal{L}^I_{root}$	.319	.328	472.2	27.3	493.5

Table 4. Quantitative results for the encoder losses, on the Mixamo encoder test set. Best scores are emphasized in **bold**, second best are <u>underlined</u>.

Diversity This metric measures the variance of generated motions [10, 18]. In the context of action recognition models, it measures the variance across all action categories, and therefore it suits an unconstrained generator. The diversity value is considered good if it is close to the diversity of the ground truth. In all our experiments, the diversity of the generated data was lower than the ground truth, so for clarity we mark it with an upwards pointing arrow, implying that in our case, higher is better.

4.4. Additional Ablation

In Tab. 4 we conduct an ablation study of the encoder losses. The best scores are mostly obtained when *not* using the foot contact loss, and the second best ones are mostly obtained when using all losses. This is expected, as the foot contact loss biases the results towards more accurate foot contact on the account of other body parts' accuracy. However, qualitatively, the human eye prefers coherent foot contact, and in our supplementary video, the pleasing foot contact results can be noticed. The phenomenon of foot contact loss degrading the quantitative results, but upgrading the qualitative ones, has been also reported in a concurrent work, MDM [21].

As detailed in the main paper, the losses of the encoder are a reconstruction loss $\mathcal{L}^I_{rec}$, a foot contact loss $\mathcal{L}^I_{fcon}$, a root loss $\mathcal{L}^I_{root}$, and a position loss $\mathcal{L}^I_{pos}$. We measure the performance of the encoder using several metrics. Recall $\mathcal{M}_{tst}$ denotes the encoder test set, m denotes an unseen motion, I denotes our trained encoder and G denotes our trained generator.

Reconstruction L2 Error This is the most important metric, as it makes sure the encoder is fulfilling its goal, *i.e.* project a motion data structure into the latent space such that the motion data structure generated from the projected value is as similar as possible to the original one. This metric is identical to the reconstruction loss, $\mathcal{L}^I_{rec}$, and is measured with

$$E^I_{recL2} = \mathop{\mathbb{E}}_{m\sim\mathcal{M}_{tst}} \left[\|m - G(I(m))\|_2^2 \right]. \quad (11)$$

Repr. \ Metric	FID ↓	KID ↓	Diversity ↑	Precision ↑	Recall ↑
Velocity	11.3	.118	**15.8**	**.470**	**.696**
Location	**10.7**	**.113**	15.1	.468	.695

Table 5. Quantitative results for the root position representation, on the Mixamo dataset. Best scores are emphasized in **bold**.

Reconstruction L1 Error Same as the previous metric, but this time with L1. The error is measured by

$$E^I_{recL1} = \mathop{\mathbb{E}}_{m\sim\mathcal{M}_{tst}} \left[\|m - G(I(m))\|_1 \right]. \quad (12)$$

Position Error In addition to the reconstruction error that mainly measures rotation angle error, we measure the error of the joint position itself. Since the global root position has a large error component, we split the error into the global root position error only, the local position error relative to the root, and both accumulated together. These errors are measured by

$$E^I_{rt} = \mathop{\mathbb{E}}_{m\sim\mathcal{M}_{tst}} \left[\|FK(m)_{rt} - FK(G(I(m)))_{rt}\|_2^2 \right], \quad (13)$$

$$E^I_{nrt} = \mathop{\mathbb{E}}_{m\sim\mathcal{M}_{tst}} \left[\|FK(m)_{nrt} - FK(G(I(m))_{nrt})\|_2^2 \right], \quad (14)$$

$$E^I_{pos} = \mathop{\mathbb{E}}_{m\sim\mathcal{M}_{tst}} \left[\|FK(m) - FK(G(I(m)))\|_2^2 \right], \quad (15)$$

where FK is a forward kinematic operator yielding joint locations, $(\cdot)_{rt}$ is the root component of the position array, and $(\cdot)_{nrt}$ is the position array excluding its root component.

In Tab. 5 we run an ablation study of the root position representation. Predicting a root position that faithfully reflects the dataset and yields natural motions is challenging, and many existing works either avoid predicting global position or predict it inaccurately, yielding a floating or jittery appearance. We study two possible representations for the root; a 3D location, or its velocity. The quantitative results of the two representations are comparable, and yet, the qualitative results have been in favor of the velocity representation, hence our choice.

Overfitting Analysis Using 3D convolutions may raise a concern about model overfitting. However, our 3D kernel is sparse as it focuses on adjacent joints only, reducing such risk. To validate that the model does not overfit, we construct a 4:1 training-to-validation split and re-train our model only on the training set. Then we measure the Chamfer distance between the generated samples and the

	Ours w/ split	Ours w/o split	Train.	Val.
Chamfer. to Train.	0.404	0.414	0.402	0.389
Chamfer. to Val.	0.466	0.461	0.519	0.492

Table 6. Overfitting analysis. Chamfer distance to training and validation sets. The results refute overfit.

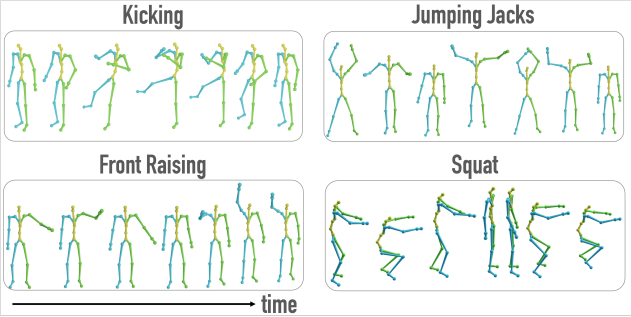

Figure 9. Our results on the UESTC dataset, after fine-tuning for the action-to-motion task.

two splits, using the distance between the two splits themselves as baselines. When measured within one set, we exclude the point itself, i.e., Chamfer(e, S\{e}). The results in Table 6 show no significant difference in the distance between the model trained with split or without split (i.e. the entire dataset) and a consistent pattern in the baselines, indicating that our model has learned the distribution of the training set, rather than simply memorizing it.

4.5. Additional Quantitative Results

Method	**FID$_{\text{train}}$** ↓	**FID$_{\text{test}}$** ↓	**Accuracy**↑	**Diversity**→	**Multimod.**→
Real	2.92	2.79	0.99	33.34	14.16
ACTOR [18]	20.49	23.43	0.911	31.96	14.52
INR [7]	9.55	15.00	0.941	31.59	14.68
MDM [21]	9.69	13.08	0.96	33.10	14.06
MoDi (ours)	10.30	14.40	0.90	33.61	13.94

Table 7. Evaluation of the action-to-motion network on the UESTC [13] dataset. Our model is designed for a completely different task, which is unconstrained synthesis. While it outperforms other works by a large margin for the unconstrained task (see main paper), it is also comparable to models that were specifically trained for the action-to-motion task.

In this section, we show that MoDi facilitates multiple short fine-tunings for various conditions, based on a one-time unconditional training. Conditional and unconditional synthesis are different problems and in a sense, conditional training is easier as the labels assist in shaping the latent space. Figure 9 and Tab. 7 show results obtained by fine-tuning an unconditional model for the action-to-motion task. Action-to-motion is the task of generating motion given an input action class, represented by a scalar. To assess our model's performance, we employ a set of metrics proposed by Guo *et al*. [10], that includes Frèchet Inception Distance (FID), action recognition accuracy, diversity, and multi-modality. This combination of metrics provides a comprehensive measure of the authenticity and variety of the generated motions. The same unconditional model can be fine-tuned for other conditions (*e.g*., text, music).

4.6. Additional Qualitative Results

In Fig. 7 we show additional qualitative results. The reader is encouraged to watch the supplementary video in order to get a full impression of our results.

4.7. Comparative Discussion of Unconditional Works

Unconditional motion synthesis is rarely studied. It is an under-explored and quite important problem.

Holden *et al*. [11] are one of the first to generate human motion using deep learning. They introduce an unconditional autoencoder, which, unlike our model, is not skeleton-aware. In their work, they train a separate feed-forward network for each editing task. On the other hand, MoDi's disentangled latent space facilitates editing in the latent space with no need to train an additional network, or uses a single encoder for a variety of applications.

Yan *et al*. [22] (CSGN) introduce a conditional framework but also mention an unconditional baseline. However, as they do not provide any code, it is not possible to perform a meaningful comparison to their approach. Their objective is to generate long coherent actions, by sampling a sequence of latent vectors from a Gaussian process and gradually increasing the spatial and temporal resolutions of the graph. The generation process is conditioned on a given prefix or a few disjoint time ranges, and the rest of the time steps are filled in. They achieve this by first transforming the conditioning sequences into latent vectors using an inverse mapping network, then sampling the missing time steps in the space of the latent vectors, and finally decoding them back to motion space.

Degardin *et al*. [8] (KineticGAN) base their conditional framework on an unconditional one, on which no results are reported. Although conditional training is easier in a sense (explained in Sec. 4.5), our unconditional results appear to be better in quality compared to their conditional ones, shown in their paper and video. Their work combines GANs and GCNs to generate human body kinetics. Similar to our approach, they also employ a mapping network from StyleGAN [14]. However, their model does not utilize critical aspects of StyleGAN, such as multi-level style injection, which we show significantly improves the quality of the synthesized motions.

Tevet *et al*. [21] (MDM) supports an unconstrained variation, and although they use state-of-the-art diffusion models, our work outperforms them in the unconditional setting.

References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4432–4441, 2019. 3

[2] Kfir Aberman, Peizhuo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. Skeleton-aware networks for deep motion retargeting. *ACM Transactions on Graphics (TOG)*, 39(4):62–1, 2020. 1, 2, 5, 7

[3] Kfir Aberman, Yijia Weng, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. Unpaired motion style transfer from video to animation. *ACM Transactions on Graphics (TOG)*, 39(4):64–1, 2020. 2

[4] Adobe Systems Inc. Mixamo, 2021. Accessed: 2021-12-25. 6

[5] Sourav Biswas, Kangxue Yin, Maria Shugrina, Sanja Fidler, and Sameh Khamis. Hierarchical neural implicit pose network for animation and motion retargeting. *arXiv preprint arXiv:2112.00958*, 2021. 2

[6] Mikołaj Bińkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *International Conference on Learning Representations*, 2018. 7

[7] Pablo Cervantes, Yusuke Sekikawa, Ikuro Sato, and Koichi Shinoda. Implicit neural representations for variable length human motion generation. *arXiv preprint arXiv:2203.13694*, 2022. 9

[8] Bruno Degardin, João Neves, Vasco Lopes, João Brito, Ehsan Yaghoubi, and Hugo Proença. Generative adversarial graph convolutional networks for human action synthesis. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1150–1159, Los Alamitos, CA, USA, 2022. IEEE Computer Society. 2, 9

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 2

[10] Chuan Guo, Xinxin Zuo, Sen Wang, Shihao Zou, Qingyao Sun, Annan Deng, Minglun Gong, and Li Cheng. Action2motion: Conditioned generation of 3d human motions. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2021–2029, 2020. 6, 7, 8, 9

[11] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016. 9

[12] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017. 2

[13] Yanli Ji, Feixiang Xu, Yang Yang, Fumin Shen, Heng Tao Shen, and Wei-Shi Zheng. A large-scale rgb-d database for arbitrary-view human action recognition. In *Proceedings of the 26th ACM international Conference on Multimedia*, pages 1510–1518, 2018. 9

[14] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020. 2, 3, 7, 9

[15] Peizhuo Li, Kfir Aberman, Zihan Zhang, Rana Hanocka, and Olga Sorkine-Hornung. Ganimator: Neural motion synthesis from a single sequence. *ACM Transactions on Graphics (TOG)*, 41(4):138, 2022. 2

[16] Shubh Maheshwari, Debtanu Gupta, and Ravi Kiran Sarvadevabhatla. Mugl: Large scale multi person conditional action generation with locomotion. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 257–265, Los Alamitos, CA, USA, 2022. IEEE Computer Society. 2

[17] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine learning (ICML)*, 2018. 3

[18] Mathis Petrovich, Michael J Black, and Gül Varol. Action-conditioned 3d human motion synthesis with transformer vae. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10985–10995. IEEE Computer Society, 2021. 5, 6, 7, 8, 9

[19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019. 7

[20] Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. *Advances in Neural Information Processing Systems*, 31, 2018. 7

[21] Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Amit H Bermano, and Daniel Cohen-Or. Human motion diffusion model. *arXiv preprint arXiv:2209.14916*, 2022. 8, 9, 10

[22] Sijie Yan, Zhizhong Li, Yuanjun Xiong, Huahan Yan, and Dahua Lin. Convolutional sequence generation for skeleton-based action synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4394–4402. IEEE Computer Society, 2019. 2, 9

[23] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-second AAAI conference on artificial intelligence*, 2018. 7

[24] Ping Yu, Yang Zhao, Chunyuan Li, Junsong Yuan, and Changyou Chen. Structure-aware human-action generation. In *European Conference on Computer Vision*, pages 18–34. Springer, 2020. 2