A Appendix

A.1 Training details

The optimization is done with an Adam optimizer with a learning rate of 1e-3 for the MLP-based models and 1e-4 for the Transformer-based models. We put some effort in tuning the learning rate for both models and settle on such simple settings to preserve comparability. We train for 10000 epochs, but find that strong performance is often already reached earlier. Unless otherwise stated we report performance for the model after 10000 epochs. During training, we find that frequent sampling of representations is important for motion generation performance. Thus for each training iteration, we sample a sequence representation and an action representation per-sequence (as opposed to sampling a single action representation per-batch). We set the weight for the Kullback-Leibler divergence to 1e-5, following the findings in [31].

Since the sequence and action representations are persistent parameters, we need to explicitly initialize them. We set a unit log-variance for all MLP-based experiments and a log-variance to -10 for all Transformer-based experiments according to our findings in Appendix B.1.

The full representation can be composed either through addition or concatenation of action-wise and sequence-wise representations. For *additive composition* we set the representation size of both action representation to $\alpha \in \mathbb{R}^{256}$ and sequence-wise representations to $\beta \in \mathbb{R}^{256}$ and the resulting sequence representation is given by $\alpha + \beta$. For *concatenation* we set both action representation to $\alpha \in \mathbb{R}^{128}$ and sequence-wise representations to $\beta \in \mathbb{R}^{128}$. We investigate the effect of both approaches in Appendix B.4 and find that the MLP-based decoder is best trained with composition through *concatenation* and the Transformer-based decoder is best trained with *additive composition*.

The optimization problems in Eqs. 6 - 8 require joint optimization with sequence-wise, action-wise and dataset-wise parameters. We employ an alternating updating scheme where first all sequence-wise and action-wise parameters are updated with respect to a fixed model and then the model parameters are updated with respect to all other parameters.

Because our work produces persistent sequence representations, our decoder is trained with temporal embeddings τ corresponding to an absolute position within a groundtruth sample. In other words, during training τ_0 always corresponds to the element at $t = 0$ of a real sample. This is different from the previous work [31], which samples fixed-length subsequences during each training iteration. In their case, τ_0 corresponds to the first element of the subsequence that was input to the encoder. As a consequence of training with random subsequences, the sequence representations of [31] are entangled with the starting point of the sequence and randomly sampling a new sequence may produce a representation at any starting point including the middle of an ongoing action. In our work, on the other hand, τ_0 always corresponds to the beginning of an action.

Algorithm 1: Greedy interval fitting

Input: set of sequence representations $C = \{c^i | i \in \mathcal{M}^z\}$; corresponding sequence lengths $T = \{T^i | i \in \mathcal{M}^z\}$; minimum interval size d_{min}; minimum population p_{min}; overlap d_{overlap}

Output: set of length intervals l_t

$t_{\text{left}} \leftarrow \min(T)$
while $t_{right} < \max(T)$ **do**
  $t_{\text{right}} \leftarrow t_{\text{left}} + d_{\text{min}} - d_{\text{overlap}}$
  $p = |\{T^i \in T | t_{\text{left}} \leq T^i \leq t_{\text{right}}\}|$
  while $p < p_{\text{min}}$ **do**
    $t_{\text{right}} \leftarrow t_{\text{right}} + 1$
    $p = |\{T^i \in T | t_{\text{left}} \leq T^i \leq t_{\text{right}}\}|$
  end
  $t_{\text{left}} \leftarrow t_{\text{right}}$
  if $p \geq p_{\text{min}}$ **then**
    $l_t \leftarrow l_t \cup \{(t_{\text{left}}, t_{\text{right}})\}$
  end
end
`// last interval may not have the minimum population`
while $p < p_{\text{min}}$ *&* $t_{left} > 0$ **do**
  $t_{\text{right}} \leftarrow t_{\text{right}} + 1$
  $p = |\{T^i \in T | t_{\text{left}} \leq T^i \leq t_{\text{right}}\}|$
end
$l_t \leftarrow l_t \cup \{(t_{\text{left}}, t_{\text{right}})\}$

Finally, the loss proposed by [31] is memory-intensive due to the computation of a human mesh, which makes training on UESTC time-intensive. Our experiments suggest that replacing the vertices loss with a joint loss akin to [10] leads to similar performance and we utilize this loss for our experiments on UESTC.

In the kinematic tree of the skeleton representation, only the root joint can't be represented as a rotation. We find that this can make it difficult to balance the root loss with the other joints, since the magnitude of the root joint is unconstrained. For some actions with significant root motion (running, jumping) this may affect performance. This may be mitigated by carefully weighing the root reconstruction loss against other losses or even using a distinct model just for the root joint. However, for the sake of comparability to the previous work, we don't address this issue in our proposed method further.

A.2 Generative Modeling

In the following we describe the details of Algorithm 1 as described in Section 3.3. Identifying a suitable set of length intervals that cover all sequence lengths in the set of sequences of an action class is a combinatorial optimization problem akin to the Knapsack problem. The objective is to maximize the number of intervals to allow fine-grained modeling of the sequence-length while also guaranteeing

a minimum number of training samples within each interval and a minimum amount of overlap between intervals.

The algorithm is action-conditional and so it is only applied to sequence representations within an action class. The set of all representations with the same action label is denoted $\mathcal{M}^z$. The population of a set (cardinality) is denoted as $|\cdot|$.

The ability of the GMM to fit to complex data is determined by the number of components (mixtures) used and a larger number of components increases the chance of Gaussian component collapse, in which one of the components is identical to the distribution of a single sample. If this happens, the sampling procedure just recreates training samples. This type of collapse would inflate our performance under the realism metric, but is not desirable. To avoid this, we first detect cases of collapsing by computing the distance between the mean of each Gaussian component and the representations of all training samples. If we find a distance below a threshold (10% of the average magnitude of all representation), we consider the GMM contain collapsed components. Since the EM algorithm is sensitive to the initial conditions we repeat the fitting with different initial conditions until we find a non-collapsed distribution. Typically we start with 15 components and if we can't find such a distribution after 100 attempts, we reduce the number of components and repeat the fitting procedure.

A.3 Tools

Our model is implemented with pytorch [28]. For conversion between different rotation representations we use pytorch3D [32] and for fitting GMMs to our representations we use scikit-learn [30]. Also we use the python implementation of smplify-x [29] during loss computation and for visualization.

A.4 Runtime

We measure the inference time for the generation of a single 60 time-step sequence with our small MLP-based model ($7.53^{\pm 0.22}$ ms), large MLP-based model ($7.66^{\pm 0.21}$ ms) used on UESTC, and the Transformer-based model ($16.65^{\pm 0.18}$ ms) and find that the MLP-based models are generally faster than Transformer-based models even with similar parameters counts.

Furthermore, we measure the time of a single training iteration with a batch size of 32, a temporal mini-batch of 5 and the reconstruction loss proposed by the Transformer baseline [31]. A single iteration takes $115.53^{\pm 0.19}$ ms on our small MLP-based model, $116.98^{\pm 0.19}$ ms on our large MLP-based model and $138.49^{\pm 0.78}$ ms on our Transformer-based model.

These time were measured on a single Nvidia Titan Xp under the same conditions. During actual training the batch size and temporal mini-batch size may differ and in particular training with larger datasets involves more sequence-wise parameters. We find that the training with our proposed settings of our MLP-based models takes 24 hours on HumanAct, 29 hours on NTU13 with a single Nvidia A100 and 40 hours on UESTC with a node of 5 Nvidia A100. As

for our Transformer-based models, it takes 15 hours on HumanAct and 31 hours on NTU13 with a node of 4 Nvidia A100.

Note that during training our method needs to update each INR frequently. This can be scaled to large datasets by distributing the INRs across multiple GPUs. While we were able to train our light-weight MLP model on a large dataset such as UESTC, training a Transformer based model with the available resources would have been excessively slow. So, due to resource constraints, we didn't perform full-scale experiments on UESTC with a Transformer-based model. Given sufficient resources, training of a Transformer-based implicit model should be straightforward.

A.5 Negative Societal Impacts

The goal of this work is to generate life-like motions for animation or downstream tasks such as data augmentation. Such life-like motions can contribute to the development of *deep-fakes*, which could be used with malicious intend to impersonate or deceive. With advances in monocular pose estimation, procuring motion data of people (and thus potential training data) without their knowledge or consent is becoming easier. The authors strongly encourage research into systems that can detect augmented/fake and/or verify real media.

A.6 Personal Data of Human Subjects

The HumanAct12, NTU RGBD and UESTC datasets are all publicly available, but no public statement about the consent of the human subjects is provided. However, this work does not use personally identifiable information such as subject labels or appearance.

A.7 Evaluation Metrics

A difference between the evaluation of [10] and [31] is the frequency with which samples of a given action class are generated. In [10] motions of all action classes are generated equally often, irrespective of the frequency in the dataset. This results in an inflated FID score, so we follow the protocol by [31], which generates motions with the frequencies found in the dataset.

The **Diversity** is computed by sampling two subset $\mathcal{S}_{d1} = \{f_1, ..., f_S\}$ and $\mathcal{S}_{d2} = \{\hat{f}_1, ..., \hat{f}_S\}$ of equal size S_d from the features f_i of all generated motions. We follow [10] and use $\mathcal{S}_d = 200$.

$$\text{Diversity} = \frac{1}{S_d} \sum_{i=1}^{S_d} \left\| f_i - \hat{f}_i \right\|_2 \tag{12}$$

Similarly, the **Multimodality** is computed by sampling two subset $\mathcal{S}_{l1} = \{f_{z,1}, ..., v_{f,S_l}\}$ and $\mathcal{S}_{l2} = \{\hat{f}_{z,1}, ..., \hat{f}_{z,S_l}\}$ of equal size S_l from the features $v_{z,i}$ of the

generated motions of action class z averaged over all action classes. Again, we follow [10] and use $\mathcal{S}_l = 20$.

$$\text{Multimodality} = \frac{1}{|\mathcal{M}_z| \cdot S_l} \sum_{z=1}^{\mathcal{M}_z} \sum_{i=1}^{S_l} \left\| f_{z,i} - \hat{f}_{z,i} \right\| \tag{13}$$

B Additional Experiments

B.1 Initialization

We find that the initialization of the variational implicit representations can effect the performance of our models. All representations are initialized with a zero mean and we scale the diagonal variance matrix with a scalar. In particular, our Transformer-based model fails to fit the training data accurately unless the variational representations are initialized with relatively small variance. On the other hand, our MLP-based model has reasonable performance independent of the initialization of the variance, but greater variances clearly improve performance.

		HumanAct12			
Method	Log-variance	FID$_{train}$ ↓	Accuracy ↑	Diversity →	Multimod. →
MLP	1	$0.114^{\pm.001}$	$0.970^{\pm.001}$	$6.786^{\pm.057}$	$2.507^{\pm.034}$
MLP	0	$0.163^{\pm.002}$	$0.955^{\pm.001}$	$6.868^{\pm.063}$	$2.706^{\pm.034}$
MLP	-10	$0.277^{\pm.004}$	$0.881^{\pm.002}$	$6.794^{\pm.059}$	$3.340^{\pm.036}$
Transformer	1	$4.355^{\pm.022}$	$0.536^{\pm.002}$	$6.195^{\pm.053}$	$3.619^{\pm.049}$
Transformer	0	$1.812^{\pm.016}$	$0.709^{\pm.003}$	$6.559^{\pm.054}$	$3.540^{\pm.037}$
Transformer	-10	$0.088^{\pm.003}$	$0.973^{\pm.001}$	$6.881^{\pm.048}$	$2.569^{\pm.040}$

Table 4. Ablation study for different initializations of the variational implicit neural representations. (± indicates 95% confidence interval, → closer to real is better

B.2 Number of GMM components

By using a conditional Gaussian Mixture Model (GMM) as a generative model, we can sample from a representation space that may be structured according to semantic factors such as action-class and sequence-length. In the ablation study in Table 5, we investigate how the complexity of the GMM (determined by the number of components) affects the quality of our generated motions. We find that our method is stable with respect to the number of components and even with a single component the method reaches the performance of the

baseline Action2Motion[10]. By increasing the number of components of the GMM we can improve the realism, while maintaining diversity. In particular, with 15 components the model can reach SOTA performance while maintaining a healthy Mean Maximum Similarity, which, as discussed in Section 3.4, indicates that the model generates motions distinct from the training set.

	HumanAct12				
#	FID ↓	Accuracy ↑	Diversity →	Multimod. →	MMS
1	$0.351^{\pm.004}$	$0.915^{\pm.002}$	$6.761^{\pm.048}$	$2.985^{\pm.040}$	$1.155^{\pm.005}$
3	$0.209^{\pm.002}$	$0.943^{\pm.001}$	$6.755^{\pm.043}$	$2.753^{\pm.044}$	$1.015^{\pm.004}$
5	$0.172^{\pm.003}$	$0.965^{\pm.001}$	$6.792^{\pm.041}$	$2.625^{\pm.031}$	$0.924^{\pm.005}$
10	$0.125^{\pm.002}$	$0.971^{\pm.001}$	$6.792^{\pm.043}$	$2.698^{\pm.033}$	$0.902^{\pm.003}$
15	$0.114^{\pm.001}$	$0.970^{\pm.001}$	$6.786^{\pm.057}$	$2.507^{\pm.034}$	$0.884^{\pm.004}$

Table 5. Ablation study of the number of components of each GMM for the MLP model on HumanAct.

B.3 Temporal Mini-Batch

During training we sample fixed-length, temporal mini-batches. These can be sampled with two strategies; (R) randomly or as (C) consecutive subsequences. Since our MLP-based model predicts each time-step independently, consecutive sampling isn't meaningful for it. The Transformer-based model predicts multiple time-steps simultaneously, so here consecutive sampling is a reasonable choice. We investigate the effect of different mini-batch sizes and sampling strategies. Note, that all models were evaluated with a fixed evaluation length of 60 time-steps.

B.4 Representation Composition

Our representations consist of sequence-wise representations and action-wise representations. These are composed to become a single sequence representation by either addition or concatenation. To ensure a fixed latent dimension of 256, we set the sequence-wise and action-wise representations to have 256 dimensions for additive composition and 128 dimensions when composing through concatenation. In Table 7 we investigate which composition performs best for which model and find that the MLP-based model performs best with composition through concatenation and the Transformer-based model performs best with additive composition.

		HumanAct12			
Method	Batch size	FID_{train} ↓	Accuracy ↑	Diversity →	Multimod. →
MLP	R5	$0.114^{\pm.001}$	$0.970^{\pm.001}$	$6.786^{\pm.057}$	$2.507^{\pm.034}$
MLP	R10	$0.141^{\pm.003}$	$0.965^{\pm.065}$	$6.856^{\pm.065}$	$2.634^{\pm.040}$
Transformer	R5	$0.214^{\pm.004}$	$0.938^{\pm.001}$	$6.755^{\pm.061}$	$2.877^{\pm.031}$
Transformer	C5	$0.498^{\pm.009}$	$0.877^{\pm.002}$	$6.674^{\pm.055}$	$3.208^{\pm.041}$
Transformer	C60	$0.088^{\pm.003}$	$0.973^{\pm.001}$	$6.881^{\pm.048}$	$2.569^{\pm.040}$

Table 6. Ablation study for size of the temporal mini-batch. The temporal mini-batch is either constructed from random frames (R) or consecutive frames (C). (± indicates 95% confidence interval, → closer to real is better)

		HumanAct12			
Method	Composition	FID_{train} ↓	Accuracy ↑	Diversity →	Multimod. →
MLP	concat	$0.114^{\pm.001}$	$0.970^{\pm.001}$	$6.786^{\pm.057}$	$2.507^{\pm.034}$
MLP	add	$0.123^{\pm.003}$	$0.960^{\pm.001}$	$6.798^{\pm.058}$	$2.642^{\pm.042}$
Transformer	concat	$0.090^{\pm.002}$	$0.959^{\pm.001}$	$6.861^{\pm.045}$	$2.737^{\pm.026}$
Transformer	add	$0.088^{\pm.003}$	$0.973^{\pm.001}$	$6.881^{\pm.048}$	$2.569^{\pm.040}$

Table 7. Ablation study for different representation compositions. The sequence and action representations can be either added or concatenated. (± indicates 95% confidence interval, → closer to real is better)

C Additional qualitative results

We present additional qualitative results in Fig. 4 as well as videos [5]. In Fig. 4 we show motions generated by our MLP-based model with different target sequence-lengths. Generating short sequences in particular can be difficult, since the generated sequence should include the a full action, beginning to end. Our method is able to reliable generate such short complete actions due to our sequence-length conditional representation space.

The videos are organized into skeleton videos for the HumanAct12 dataset and full 3D renderings for the UESTC dataset. For the HumanAct12 dataset we provide a direct side-by-side comparison for Action2Motion[10], ACTOR[31] and the proposed INR approach with a Transformer and MLP decoder. For the UESTC dataset we provide a direct side-by-side comparison between ACTOR and the proposed INR approach with an MLP decoder. To highlight the ability of each model to handle variable-length sequences, we generate sequences with

[5] Videos are included in the supplementary materials. Upon acceptance, they will be released.

Fig. 4. Examples of motions for the action classes *boxing*, *lift dumbbell* and *warm up* generated with our MLP-based model with different target lengths. The generated motions are complete (finish within the target sequence length) and are diverse.

sequences lengths of 40, 60 and 120 frames. The sequences for the baselines were generated by the models provided by the authors without further fine-tuning.