PoseGPT: Quantization-based 3D Human Motion Generation and Forecasting
Supplementary Material

Thomas Lucas*, Fabien Baradel*, Philippe Weinzaepfel, and Grégory Rogez

NAVER LABS Europe
https://europe.naverlabs.com/research/computer-vision/posegpt

In this supplementary material, we first discuss the attached video which shows samples generated from PoseGPT (Section 1). We then present additional details about PoseGPT (Section 2).

1 Discussion on the attached video

Samples generated without observed motion. First we show some samples generated from scratch for four different human actions, namely 'walk', 'turn', 'jump' and 'dance'. The initial human poses are different for all samples and the human motions are diverse; this indicates that PoseGPT is able to generate realistic, diverse and discriminative human motions. We also observe that even though the human action 'dance' is not well represented in the BABEL dataset, PoseGPT is still able to generate diverse samples.

Samples conditioned on an initial human pose. We then show samples obtained while conditioning our model on a initial pose for the human actions 'run' and 'turn'. There is high diversity in the future human motions generated by PoseGPT, yet all samples are realistic futures given the initial human pose, which demonstrates the flexibility of the model.

Samples conditioned on an observed human motion. Finally we generate samples while conditioning the model on 10 frames, and visualize future motions for the classes 'turn', 'throw' and 'stretch'. We observe that increasing the duration of the observations allows us to decrease the uncertainty on the predicted futures, even though they remain diverse and do not mode-drop to a single modality.

2 Additional details

Auto-regressive prediction head. In Section 3.1 of the main paper, we propose to model correlations between the K codebook indices produced by product quantization by using a prediction head that is auto-regressive over the K codebooks. At train time, this prediction head takes as input (i) the logits produced by the standard head

*Equal contribution.

of the network and (ii) the K indices embedded using the same embedding as used for the inputs. Then index k is predicted from a concatenation of ($\texttt{logits}_{1\ldots k-1}$, $\texttt{embed(Input)}_{k\ldots K}$). Note that this induces almost no overhead: $K-1$ extra linear layers, used in parallel, with K typically in $\{2,4\}$. This stands in contrast with the naive solution which would be to concatenate the products along the time dimension: it would have increased the cost of the whole network by a factor K^2 due to the quadratic cost of self-attention mechanisms. At sample time, the K indices predicted at a time step t are sampled sequentially and used as input for the next prediction; thus, the $k-th$ token is predicted conditionally on tokens $0\ldots k-1$ as is the case at train time. Importantly, this only requires running the *prediction heads* sequentially rather than in parallel as is done at train time. On the other hand, the naive solution would have increased the sampling time by a factor K.

Input embedding. Self-attention based architectures are invariant to permutations of their input, which avoids inductive biases in the architecture. However this can be detrimental when modeling sequential data, as positional information is lost; we follow the standard remedy of learning 1-D positional encodings, added to the embedded input data, to account for the temporal dimension. We also learn embeddings for each action token a and for the sequence length T. We experiment with two ways of adding this conditioning information to the input data: the first one is by adding an extra token to the input sequence, which is possible because the a and T are constant across time. The second is to add the information at every time step. For this we again test two strategies: embedding all informations separately and simply summing them, or concatenating the embeddings and learning an extra layer on top of that. More precisely, each embedding –input, positional, action and length – is a $D_{\text{emb}}-$dimensional vector; they are concatenated together, linearly projected again to a $D_{\text{emb}}-$ dimensional space and fed to the transformer model. We find in the experimental section (Section 4.2) that the last strategy is the best one.

Architectural details. All three components - E, D, and G are based on transformers. The encoder E is a stack of 3 blocks where each block is composed of 4 transformer layers. We perform temporal downsampling by a factor of 2 after the first block by default, and after each following block when downsampling by a factor greater than two (i.e., when $T/T_d \in \{4,8\}$). The input embedding dimension of each block is 512. each transformer layer is composed of a self-attention module which has 5 attention heads – each of dimension 32 – and a feedforwad layer with 512 hidden units. We use a dropout of 0.1 both in the self-attention and in feed-forward modules. The decoder D mirrors the decoder, with the same hyper-parameters and blocks called in reversed order. The auto-regressive model G is a transformer which has 8 layers; the self attention blocks use 4 attentions heads, each of dimension 256. The input embedding dimension is 256, and we use a dropout of 0.2 for this network.

Training details. We implement PoseGPT in Python using the PyTorch framework [3] and we train our network from scratch using the Adam optimizer [2] with a learning rate of 5.10^{-5} and default parameters. Both the auto-encoder in the first training stage

and the auto-regressive network in the second stage are trained for 2 million iterations. The whole training procedure takes 3 days on one single Nvidia V100 GPU. L_2 reconstruction losses are applied to body model parameters as well as directly on vertices for a randomly sampled subset of time steps for efficiency/memory reason; we found that using 10% to 20% of the frames is sufficient.

Action classifiers. For HumanAct12 we use the classifier provided by [1] which is a GRU followed by a a fully-connected layer. The classifier takes as input 3D joints of the human skeleton centered around the spine. For both BABEL and GRAB, we train the classifier ourselves using the same architecture as described above.

References

1. Guo, C., Zuo, X., Wang, S., Zou, S., Sun, Q., Deng, A., Minglun, Cheng, L.: Action2motion: Conditioned generation of 3d human motions. In: ACMMM (2020)
2. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
3. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)