Long Term Motion Prediction Using Keyposes
Exploring Transformers and VQVAE

Dhruti Shah, Sena Kiciroglu, and Pascal Fua

Computer Vision Lab
Ecole Polytechnique Federale de Lausanne (EPFL)
{dhruti.shah,sena.kiciroglu,pascal.fua}@epfl.ch

Long term human motion prediction is essential in safety-critical applications such as human-robot interaction and autonomous driving. To achieve long term forecasting, we predict a few keyposes and approximate intermediate ones by linearly interpolating the keyposes. In previous work, prediction of these keyposes has been implemented using Recurrent Neural Networks (RNNs). Transformers have recently been shown to achieve high accuracy in Natural Language Processing tasks. In this work, we use transformers for keypose prediction to gain higher accuracy. Furthermore, in previous work, keyposes have been extracted from the data by simple clustering methods. We also explore the Vector Quantised-Variational AutoEncoder (VQ-VAE) to obtain keyposes which are more representative of the underlying pose data.

1 Introduction

Human motion prediction is a key component of many vision-based applications, such as automated driving, surveillance, accident prevention, and human-robot interaction. Its goal is to forecast the future 3D articulated motion of a person given their previous 3D poses. [3] showed that predicting the pose in every future frame is unnecessary because, given two poses separated by a short time interval, one can easily guess the intermediate ones using simple linear interpolation. It is therefore enough to predict a set of keyposes from which all others can be re- covered by interpolation. Such keyposes are by definition sparser than the full set of future poses and therefore easier to forecast. [3] use a RNN-based motion prediction network that takes earlier keyposes as input and yields a probability distribution over the clusters for each future keypose along with transition times. They extract the keyposes for every training motion individually and collect in one set which is then clustered into K clusters via k-means.

In this work, we replace the RNN-based keypose prediction network by a transformer (Section 2). Transformers were introduced in [10] initially for the task for sequence prediction in the context of Natural Language Processing (NLP). Since then, they have been used in various other applications like speech prediction [5], time series forecasting [4], pose prediction [9].

Moreover, we also explore Vector Quantised-Variational AutoEncoder (VQ-VAE) architecture to obtain better keyposes from the training data (Section 3). [3] use

a simple k-means algorithm to cluster the poses in the training data and obtain keyposes. In this work we try to leverage the power of VQ-VAEs to yield keyposes which are better representative of the training data.

2 Transformers for keypose prediction

2.1 Background

The paper 'Attention Is All You Need' [10] describes transformers and what is called a sequence-to-sequence architecture. Sequence-to-Sequence (or Seq2Seq) is a neural net that transforms a given sequence of elements, into another sequence. In the case of NLP, this sequence would be a sequence of words, and in our case this would be a sequence of keyposes. Given as input a sequence of keyposes, we would like to predict the following sequence of keyposes.

Transformers were initially proposed for the task of translation, where the sequence of words from one language is transformed into a sequence of different words in another language. The overall architechture of the transformer architecture is shown in Fig. 1. The Encoder is on the left and the Decoder is on the right. Both Encoder and Decoder are composed of modules that can be stacked on top of each other multiple times, which is described by Nx in the figure. We see that the modules consist mainly of Multi-Head Attention and Feed Forward layers. The inputs and outputs (target sentences) are first embedded into an n-dimensional space since we cannot use strings directly.

One slight but important part of the model is the positional encoding of the different words. Since we have no recurrent networks that can remember how sequences are fed into a model, we need to somehow give every word/part in our sequence a relative position since a sequence depends on the order of its elements. These positions are added to the embedded representation (n-dimensional vector) of each word.

The Transformer applies a mask to the input in the first multi-head attention module to avoid seeing potential 'future' sequence elements. This is specific to the Transformer architecture because we do not have RNNs where we can input our sequence sequentially.

2.2 Modelling the transformer

The architecture of the transformer we saw in the previous section was proposed for the task of translation. We however need a model which is used for sequence prediction. Our task is analogous to the task of language modelling using transformers. In language modelling, a part of a sentence is given as input to the transformer, and the transformer predicts the rest of the sentence. In our case, the keyposes of motion are analogous to words of a sentence. We would like to input a set of keyposes, and predict the next set of keyposes that the motion would follow. Hence,

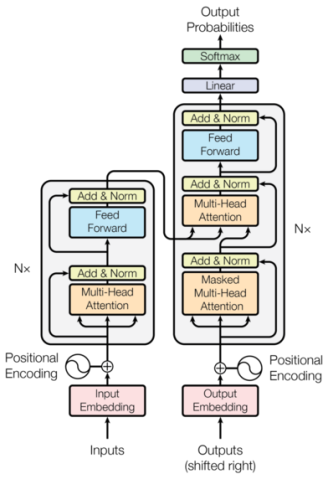


Figure 1: The transformer architecture as proposed in [10].

we do not need both the encoder and decoder. We just use the encoder part of the architecture shown in Fig. 1.

We start with the model of the transformer used for sequence-to-sequence language modelling given in [8]. The transformer model takes as input a sequence of keyposes ($kp_1, kp_2, ...kp_{t_{obs}}$), along with a mask. The mask ensures that the transformer cannot look at future keyposes during prediction. It outputs a right-shifted sequence of keyposes ($kp_2, kp_3, ...kp_{t_{obs}+1}$).

2.2.1 Positional encoding module

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottom of the encoder stack. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. We experiment with two different types of positional encoding modules: 1. where we assume that the input keyposes are equidistant 2. where we inject some information about the time difference between the keyposes into the positional encoding module. We show the results of thes in the experiments section.

2.2.2 Mask

Along with the input sequence, a square attention mask is required because the self-attention layers in nn.TransformerEncoder are only allowed to attend the earlier positions in the sequence. For our task, any tokens on the future positions should

be masked. Hence, we use a mask as shown in Fig. 2. This shows that to predict the n^{th} keypose in the sequence, the transformer will only be allowed to look at the past keyposes i.e. from 1 to n, and will not be able to look at the future keyposes.

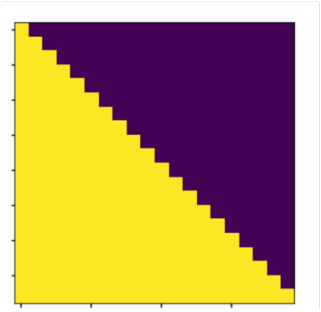


Figure 2: The square attention mask input the transformer, which masks the future positions.

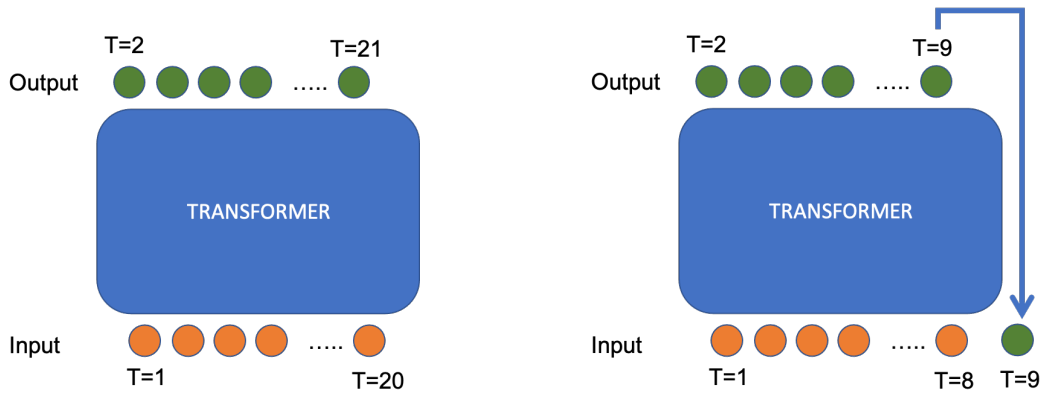
2.2.3 Training, Validation and Testing

The transformer takes a input a sequence of keyposes and outputs a right-shifted sequence of keyposes. By default, internally the transformer architecture implements *teacher-forcing*, ie to predict the next keypose, it uses the ground-truth of the previous keyposes and not the predicted keyposes at the previous time-steps. During training, we use this teacher-forcing with a certain probability. We explore different probabilities of teacher-forcing in the experiments section.

We show in Fig. 3 how the transformer training and testing works. Assume we have a sequence of keyposes of length 20. We would like to input 8 keyposes to the transformer and predict the next 12 keyposes. During training, with teacher forcing, the transformer takes as input the full sequence ie kp_1 to kp_{20} and predicts $kp_2 to kp_{21}$. To calculate the loss, we penalize the keyposes kp_{12} to kp_{20}. During validation and testing, we input the sequence kp_1 to kp_8, we get as output kp_2 to kp_9. We take kp_9, append it to the input sequence, and once again input this to the transformer, to get kp_{10}. We do this iteratively till we predict upto kp_{20}.

2.3 Experiments

We implement our code in pytorch. Our final transformer model uses a transformer encoder consisting of 5 nn.TransformerEncoderLayer, embedding size 200 and 2 multi-attention heads. It takes an input a sequence of probability distributions over the keyposes for each timestep. During training, we do teacher forcing with a probability of 0.5. We use an sgd optimizer to train the transformer. We use data consisting of 1500 keyposes. We report the top-1 OMAC (Only Motion Action Classifier) and FAC (Full Action Classifier) accuracies.



(a) Transformer training.

(b) Transformer validation and testing.

Figure 3: Methodology for transformer training, validation and testing.

All of the above we obtained through a series of ablations on different parts of the model and input data. In the following subsections, we show and discuss these.

2.3.1 Transformer model

In Fig. 4 we see the comparison between two popular transformer-based models used for language modelling. We see that the GPT-2 [7] is built using transformer decoder blocks. BERT [1], on the other hand, uses transformer encoder blocks.

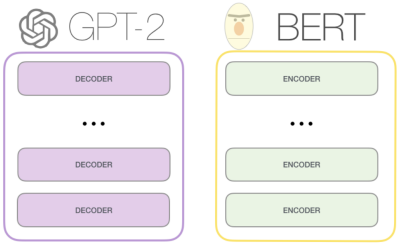


Figure 4: The GPT-2 is built using transformer decoder blocks. BERT, on the other hand, uses transformer encoder blocks. [2]

We started with a encoder-based model (like BERT) and compared out results with a decoder-based model (like GPT-2), keeping the rest of the parameters of the model constant. Table 1 shows that the encoder-based model performs superior to the decoder-based model. This difference mainly arises because of the difference between encoder and decoder stacks.

2.3.2 Input keypose number, teacher forcing

Unlike in an RNN-based network, for a transformer there are no explicit hidden states which we can access which hold the current model state. Hence, the input se-

Model	Top-1 OMAC	Top-1 FAC
Encoder-based	0.307	0.392
Decoder-based	0.289	0.332

Table 1: Comparison of performance of encoder-based and decoder-based models.

quence number for a transformer matters. We experimented with different number of input keyposes for the sequence.

As mentioned before, during training, we input the entire sequence to the transformer and it outputs a right-shifted sequence. By default, at each timestep the transformer takes in the ground-truth value of the previous keypose. This is known as teacher-forcing. However, if teacher forcing happens all the time during training, then this leads to a distribution shift and leads to lower validation and testing accuracies. Hence We implement teacher forcing with a certain probability. We explore different values of teacher forcing probability along with the number of input keyposes. The results are shown in Table 2. We observe that the model performs best for input keypose number 10 and with a teacher-forcing probability of 0.5.

Input keyposes	Output keyposes	Teacher-forcing prob	Top-1 OMAC	Top-1 FAC
8	12	0	0.2010	0.2896
10	12	0	0.2292	0.3083
12	12	0	0.2094	0.3052
16	12	0	0.2073	0.2969
20	12	0	0.2167	0.2875
10	12	0.25	0.2188	0.3177
10	**12**	**0.5**	**0.2292**	**0.3219**
10	12	0.75	0.2146	0.2823
10	12	1	0.2115	0.2812

Table 2: Comparison of performance of model due to varying input keypose number and teacher-forcing probabilities.

2.3.3 Positional encoding module

As mentioned previously, the transformer has a positional encoding module which injects information about the relative or absolute position of the tokens in the sequence. These positions of tokens can be one of two ways:

1. Equidistant : We assume that the keyposes in our input sequence have constant distance between consecutive keyposes.

2. Time-information : For the input sequence, we have access to time difference between the keyposes. We can use this to inject more information into the positional encoding module.

We observe that both the positional encoding modules described above perform almost similar with no significant gains observed by incorporating time-based information into the positional encoder.

2.3.4 Number of keyposes (cluster centers)

The number of keyposes in our case is analogous to the dictionary size in the language modelling task. We ran our code with different number of distinct keyposes in the input data and the results are shown in Table 3. We see that the accuracy increases on increasing the number of keyposes upto a certain extent, and then decreases again.

Number of keyposes	Top-1 OMAC	Top-1 FAC
150	0.2302	0.3313
1000	0.260	0.338
1500	**0.307**	**0.392**
2000	0.293	0.376
3000	0.283	0.355

Table 3: Comparison of performance of model due to varying number of keyposes.

2.3.5 Multimodality

The transformer model gives as output a probability distribution over the keyposes. For the unimodal case, we choose the highest probability keypose and the chosen one. To show that our model works not just in the best case, but when predicting multiple diverse sequences, we sample the keypose from the output distribution instead of taking the highest probability one. We then report the average accuracy of the diverse predicted sequences. With predicting 30 diverse trajectories for each input sequence, we get the top-1 OMAC and FAC accuarcies as 0.287/0.370.

3 VQ-VAE for keypose estimation from training data

In the original work [3], the keyposes are extracted for every training motion individually and collected in one set which is then clustered into K clusters via k-means. Each keypose is given a label determined by the cluster it is assigned to. In this work, we explore a means to obtain keyposes which better capture the data. To do this, we utilize the concept Vector Quantised- Variational AutoEncoder (VQ-VAE) as proposed in [6].

For each pose in our data, our goal is to assign a keypose which is closest to the pose. These keyposes transform the poses from the continuous to discrete domain. The model we porpose learns the following: it maps each pose to its corresponding keypose (cluster label). This is where the VQ-VAE architecture helps us.

3.1 Background

[6] introduce a new family of generative models succesfully combining the variational autoencoder (VAE) framework with discrete latent representations through a novel parameterisation of the posterior distribution of (discrete) latents given an observation, termed VQ-VAE. Since VQ-VAE can make effective use of the latent space, it can successfully model important features that usually span many dimensions in data space.

Fig. 5 shows various top level components in the architecture along with dimensions at each step. Encoder takes in images $x : (n, h, w, c)$ and give outputs $z_e : (n, h, w, d)$. Vector Quantization layer takes z_e and selects embeddings from a dictionary based on distance and outputs z_q. Decoder consumes z_q and outputs x' trying to recreate input x.

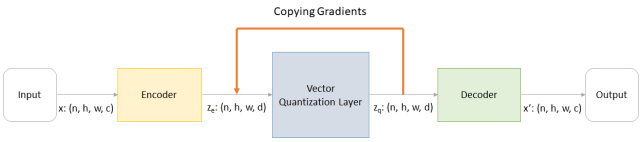


Figure 5: The VQ-VAE architecture. [11]

Fig. 6 shows the working of VQ layer. We use this block in our keypose model.

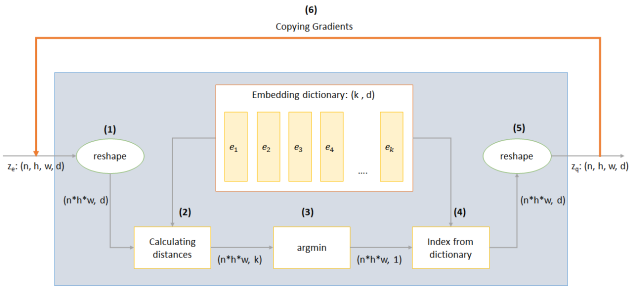


Figure 6: The Vector Quantization Layer. [11]

3.2 Model

We use the VQ block shown in previous section, along with an encoder and decoder block. The encoder block takes in as input the sequence of poses $p_1, p_2, ...p_n$. The encoder maps takes the first and last pose of the input sequence and encodes them to the embedding dimension. These encoded beginning and end of the sequence are then input to the VQ block. This block maps the encoded poses to the nearest encoded keypose. These are then input to the decoder block which converts from

embedding dimension to pose dimension. Then we interpolate between the reconstructed beginning and end pose to get the reconstructed output sequence. The overall model is shown in Fig.7.

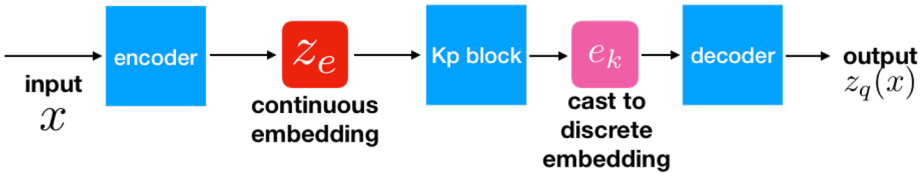


Figure 7: The overall model used to obtain latent keyposes.

To train the encoder, decoder and the KP block, we use 3 different losses which make up our total loss:

1. **Reconstruction loss**: which optimizes the decoder and encoder.

$$\text{Reconstruction loss} = -\log(p(x|z_q(x)))$$

2. **Codebook loss**: due to the fact that gradients bypass the embedding, we use a dictionary learning algorithm which uses an l2 error to move the embedding vectors e_i towards the encoder output.

$$\text{Codebook loss} = ||sg(z_e(x)) - e||^2$$

   where sg refers to the stop gradient operator meaning no gradient flows through whatever it's applied on.

3. **Commitment loss**: since the volume of the embedding space is dimensionless, it can grow arbitrarily if the embeddings e_i do not train as fast as the encoder parameters, and thus we add a commitment loss to make sure that the encoder commits to an embedding.

$$\text{Commitment loss} = \beta||z_e(x) - sg(e)||^2$$

   where β is a hyperparameter that controls how much we want to weigh commitment loss compared to other components.

3.3 Experiments

We train our model with $\beta = 0.25$. We observe the loss profile as shown in Fig. 8. Here *loss* refers to the total loss of the model, *loss pos* refers to the reconstruction loss and *loss vq* refers to codebook loss + commitment loss. We observe that all three losses are decreasing, however the VQ losses dominate over the reconstruction loss.

We also show in Fig. 9 a few visualizations of the keyposes chosen by the model. In each figure, the ground-truth pose is in black, and the keypose chosen for that

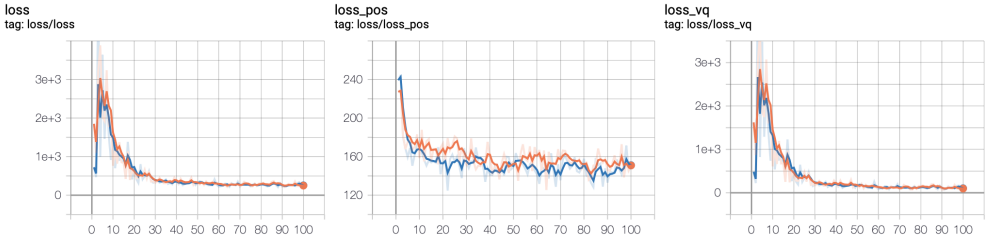


Figure 8: Loss profile during training the VQ-based model.

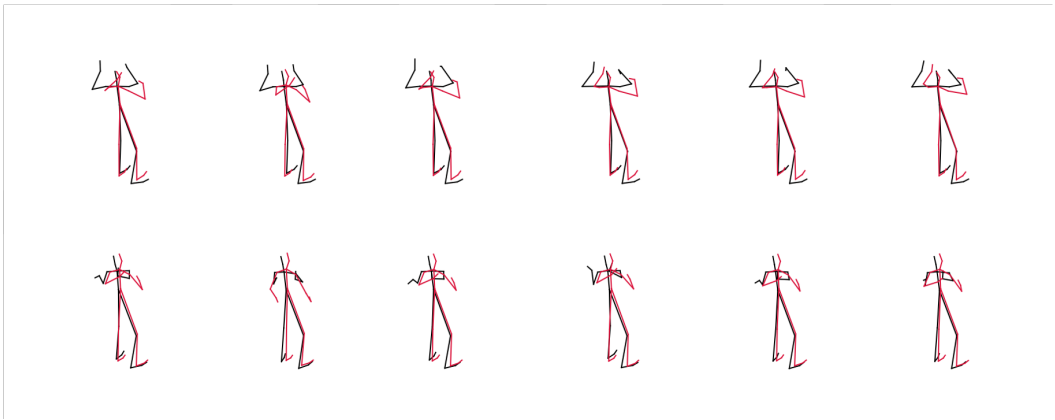


Figure 9: Vizualization of keypose chosen for each pose in a sequence. Ground-truth pose is in black, and the keypose chosen for that pose, by our model, is shown in red.

pose is shown in red. Each row corresponds to a sequence of movement, and we shown three different sequences.

Finally, the goal of the model was to choose appropriate cluster centres for the poses, which are represented by keyposes. In Fig. 10 we show a few keyposes which form the cluster centers.

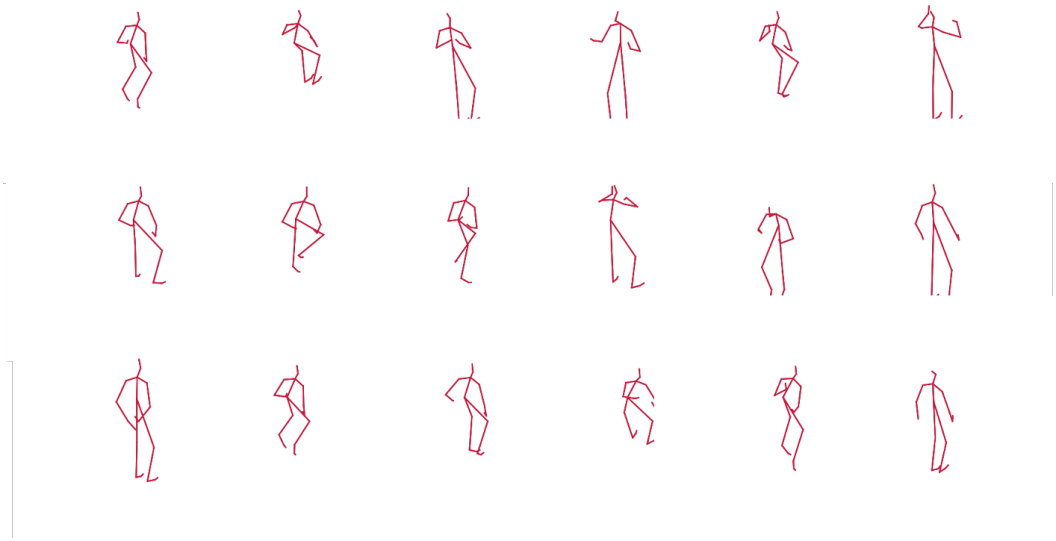


Figure 10: Vizualization of the keyposes corresponding to the cluster centres.

4 Future work

When we use transformers for keypose prediction, we use the keyposes generated by k-means clustering. Combining the two parts, we can use the trained VQ-based model to predict keyposes and use these for keypose prediction using transformers. Further, there exists scope for improvement in the VQ-based model architecture.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[2] *Illustrated GPT2*. URL: https://jalammar.github.io/illustrated-gpt2/.

[3] S. Kiciroglu, W. Wang, M. Salzmann, and P. Fua. "Long Term Motion Prediction Using Keyposes". In: *arXiv preprint arXiv:2012.04731* (2020).

[4] B. Lim, S. O. Arik, N. Loeff, and T. Pfister. "Temporal fusion transformers for interpretable multi-horizon time series forecasting". In: *arXiv preprint arXiv:1912.09363* (2019).

[5] L. Lu, C. Liu, J. Li, and Y. Gong. "Exploring transformers for large-scale speech recognition". In: *arXiv preprint arXiv:2005.09684* (2020).

[6] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu. "Neural discrete representation learning". In: *arXiv preprint arXiv:1711.00937* (2017).

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), page 9.

[8] *SEQUENCE-TO-SEQUENCE MODELING WITH NN.TRANSFORMER AND TORCHTEXT*. URL: https://pytorch.org/tutorials/beginner/transformer_tutorial.html.

[9] L. Stoffl, M. Vidal, and A. Mathis. "End-to-end trainable multi-instance pose estimation with transformers". In: *arXiv preprint arXiv:2103.12115* (2021).

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017).

[11] *vqvaeblog*. URL: https://blog.usejournal.com/understanding-vector-quantized-variational-autoencoders-vq-vae-323d710a888a.