Table 2: **Dataset details.**

	FineGym		Diving48		FisV	
	Long	Short	Long	Short	Long	Short
Training trajectories	404k	665k	787	27k	231k	276k
Validation trajectories	50k	83k	98	3k	29k	35k
Test trajectories	50k	83k	98	3k	29k	35k
Total trajectories	505k	832k	984	33647	289k	345k
Total videos	13k		18k		500	

A Limitations and Societal Impact

Limitations Our approach assumes all the information about a trajectory can be represented by a single latent-space embedding. While this holds true in our experiments, it is unclear how well it can scale to more complex data (like pixel-space videos) or longer trajectories (of the order of minutes). See Subramani et al. [48] for a related discussion. Another limitation we noticed from our model is that, while it is in principle capable of generating diverse trajectories from given segments, it tends to predict "average" trajectories when asked to predict far into the future (or past). Baselines have a similar behavior. A generative approach to the decoding, which would naturally fit into our framework, would probably help generating diverse trajectories far into the future; we leave it to future work. Finally, we mention in Section 3.1 that, by construction, contextual information can easily be added to the model, but we did not try it. We leave this exploration to future work.

Negative societal impact Our approach is data-driven, and thus it will replicate the biases seen in the data. Trajectory data is not as sensitive as image data on its own. However, trajectory-modeling methods can be used in sensible applications such as tracking of people for surveillance purposes.

B Dataset Details

For each one of the considered datasets, we process all the frames individually with OpenPose [10] to extract human skeleton keypoints. OpenPose returns the keypoints of all the detected humans. We then post-process the extracted keypoints, in order to group the ones belonging to the same person over time. We use heuristics that include spatial closeness between two consecutive frames and similar size of the skeleton. Once the correspondences have been obtained, and we get a series of trajectories, we filter out those that are either too small, too short, or too noisy. The minimum length (in seconds) of the trajectories is set to be the maximum length of the combination segments being sampled, so that, from the point of view of the segments, the trajectories are not limited in time.

In the short-segment experiments, the segment length is up to 30 frames, from which we randomly (read: non-uniformly) sample a third of them (this is, up to 10), for slightly over a second long segments. In the long-segment experiments, the segment length is up to 90 frames, from which we randomly sample a third of them (this is, up to 30), for slightly under four seconds long segments. Note that our model is not limited to predicting trajectories within this time-span, and it can extrapolate to larger time values.

We randomly divide the obtained trajectories into training, validation and test splits in a 80/10/10 proportion. We report the size of each dataset in Table 2. Because Diving48 is a small dataset, we fine-tune the FineGym-trained models for it, both in our models and in baselines. The reason there are very few long trajectories for Diving48 is that OpenPose extracts noisy keypoints due to out-of-distribution positions, which get filtered-out in our post-processing. The amount of videos reported for FineGym and Diving48 corresponds to the number of events; there is more than one event in each original video, but we pre-process them into separate event-level videos before extracting the keypoint trajectories.

C Implementation Details

Architecture We implement the encoder network Θ using a Transformer Encoder [54] with two layers and two heads, and hidden size 512. The latent space dimensionality N is also 512. The input

to the Transformer are the spatial positions at every sampled time-step, as well as a [CLS] token to represent the whole trajectory. We use the output of the [CLS] token to represent the distributions Q. We append a temporal embedding to each input, corresponding to the time each point was sampled. We use Fourier encodings [51] to encode the continuous time value to its vector representation. We found using MLPs had similar results but slightly worse generalization to extrapolation to out-of-distribution time-steps.

The decoder network Φ is implemented using a ResNet with four blocks and hidden size 512. The temporal indices are also encoded using Fourier encodings, and they are concatenated to the z value to be decoded, following prior work [53]. We use the same architecture for the decoder network in the baselines. The capacity of the baselines is the same as our method's (~10M parameters).

Inputs In order to avoid learning shortcuts by Θ, the combination segment has the same temporal span as the combination of the past and future segments, but the specific sampled points are different (different times), so that the model cannot simply identify that the values are exactly the same and use that information to bring representations together. For every trajectory in the dataset, we randomly sample a start and end point for the segments to be used.

Losses We classify segment pairs as hard positives, which we deem very important, soft positives, soft negatives, and hard negatives, and sample hard positives and negatives more often. For example, the pair consisting of the past **P** and its reconstruction **PP** is deemed important enough to be a hard positive, and samples from other random elements in the batch are considered soft negatives. We found in initial experiments that the selection of which pairs are hard or soft is only marginally important, as long as the overall intuition described in Section 2.2 is followed. We report the specific choices in Fig. 7. Note that in the conditional setting, where we are minimizing and maximizing probability values, optimizing a binary cross-entropy loss instead of a triplet loss may seem more adequate. However, we found in initial experiments that the triplet loss performed better, so we kept it.

We found using l^1-norm for the distance function δ to perform similarly to the l^2-norm, so we kept using l^2-norm but l^1-norm is a viable alternative. This shows that our model is general and not dependant on the specific δ.

In the triplet loss in Eq. (1), we set the margin $\alpha = 1$.

When sampling trajectories to be decoded, we sample 3 trajectories for every distribution Q. These will be either positives or negatives among themselves, depending on the case (see Fig. 7).

Optimization The model parameters are optimized using AdamW [32] with weight decay of 0.05. For the learning rate, we use a cosine annealing strategy, with ranges $[1e-6, 1e-4]$, a period of 4k steps, and a warmup of 1k steps. Additionally, we clip the gradients by norm for values larger than 0.01. We did not run a hyperparameter search on the previous parameters, as we found that the initial ones worked well, except for the gradient clipping, which we found necessary to stabilize the training of the Transformer. Finally, we trained the model using mixed precision.

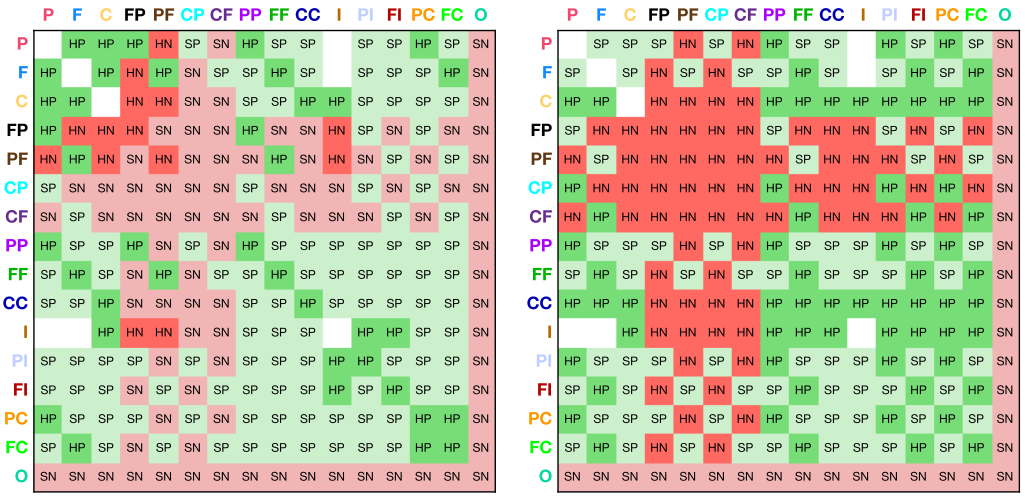
We use PyTorch [37], and the box embeddings library [12]. The models take two days to train on four RTX208-Ti GPUs.

Code, models, and data are provided.

C.1 All pair-to-pair negative/positive relationships

We follow the notation in Fig. 3, and show pairwise negatives and positives in Fig. 7. Note that some segments have a negative or positive value defined with respect to themselves: this corresponds to the relationship between different samples. For example, given a past **P**, multiple futures **FP** can be defined, with a specific relationship between them (in this case, they will be hard negatives). Additionally, we add other options that are not shown in Fig. 3. We list all of them here:

- **P** Past.
- **F** Future.
- **C** Combination.
- **FP** Future given past. Representation that has been obtained from encoding a segment that has been decoded from the past **P** representation at times t obtained from the future **F** segment.



(a) **Symmetric approach** (b) **Conditional approach**

Figure 7: **Positive and negative pairs**. HP, SP, SN, HN stand for "hard positive", "soft positive", "soft negative" and "hard negative", respectively. Note that the two matrices are the same if we consider hard and soft to be equal. The symmetric approach results in symmetric negatives and positives, while the conditional one results in an asymmetric one, because $P(A|B) \neq P(B|A)$ in general. However, as expected, it is symmetric if we consider hard and soft negatives to be equal. In that case, the two approaches result in the same matrices. Best seen in color.

PF Past given future.
CP Combination given past.
CF Combination given future.
PP Past given past. As in the previous cases, this corresponds to decoding the past **P** representation into the times in the past segment, and then re-encoding the obtained segment.
FF Future given future.
CC Combination given combination.
I Intersection. Distribution that results from intersecting the past **P** and future **F** representations in the latent space.
PI Past given intersection.
FI Future given intersection.
PC Past given combination.
FC Future given combination.
O Other (segment from a different trajectory in the batch).

In practice, not all positive and negative pairs are equally important. We differentiate between soft and hard positives and negatives, where hard ones are deemed more relevant and we give them a more important role in the optimization, by sampling them more often during training. The motivation behind distinguishing between hard and soft pairs is that we want to mostly rely on strong signals, and not add too much noise to the training. However, the distinction between hard and soft pairs is not as crucial and fundamental to our framework as the distinction between positives and negatives. Different criteria to select hard and soft pairs could be used, and the rules we used to differentiate between hard and soft positives and negatives are as follows. First, let us define the representations of segments that come directly from the data as "first encodings", to distinguish them from the re-encoded ones.

For the symmetric approach, where positives are defined as those segments that can belong to the same trajectory, *hard* positives are the positive pairs where *either* of these conditions is met:

- The pair consists of two first encodings. For example, (**P**, **F**).
- Pairs that encode *exactly* the same segment, and thus the distribution should be exactly the same. For example, (**P**, **PP**).
- Additionally, we add four extra hard positives that do not meet either of the previous conditions, but are necessary so that some hard negatives can act as such. In a triplet loss, a (hard) negative

requires a (hard) positive to be contrasted to. Some hard negatives like (**PF**, **PF**) would not have a hard positive associated to them if we only followed the two previous conditions (**PF** would not be hard positive with any other segment), and for this reason we explicitly create these four hard positives, which are (**P**, **FP**), (**F**, **PF**), (**PP**, **FP**) and (**FF**, **PF**).

Hard negatives in the symmetric case are the negative pairs where *both* of these conditions are met:

- One element of the pair represents either the past or the future.
- The other element is either a first encoding (*e.g.* the pair (**PF**, **P**)), or it represents the same segment as the first element of the pair, but a different sample (*e.g.* the pair (**PF**, **PF**)).

For the conditional approach, the distinction between hard and soft positives is more clear: hard positives are those where $P(A|B) = 1$ (this is, the first case in the list, where given B we can be certain of A), for example for $P(\mathbf{P}|\mathbf{C}) = 1$. Other positives (second and third cases) are treated as soft. All negatives are treated as hard negatives except for the ones coming from different elements in the batch.

An ablations where we make all positives hard is included in Tab. 3.

D Distribution Families

D.1 Normal Distributions

Multivariate normal (also called Gaussian) distributions have the following probability density function (PDF):

$$p(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^N |\boldsymbol{\Sigma}|}} \exp\left[-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{z}-\boldsymbol{\mu})\right], \tag{3}$$

where $\mathbf{z}$ is a real N-dimensional column vector, μ is the mean, and $\boldsymbol{\Sigma}$ is the covariance matrix. In the main paper all variables z and x are vectors, so we do not use their bold versions $\mathbf{z}$ and $\mathbf{x}$, as they cannot possibly be confused with scalars.

We assume an uncorrelated multivariate normal distribution, so the previous equation is simplified to:

$$p(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left[-\frac{(z_i-\mu_i)^2}{2\sigma_i^2}\right], \tag{4}$$

where $\boldsymbol{\sigma}$ is a vector representing the individual-dimension standard deviations.

The product of two normal PDFs (*not* the product of two normal random variables) results in a scaled normal distribution when the N variables are uncorrelated. This is *not* general for other multivariate normal distributions. See Bomiley [7] for derivations in the two scenarios. The resulting PDF is the product of the individual dimension density functions, which follow the equation:

$$\begin{gathered} p(z)q(z) = \frac{S_{pq}}{\sqrt{2\pi\sigma_{pq}^2}} \exp\left[-\frac{(z-\mu_{pq})^2}{2\sigma_{pq}^2}\right], \\ \text{where} \quad \sigma_{pq} = \sqrt{\frac{\sigma_p^2\sigma_q^2}{\sigma_p^2+\sigma_q^2}} \quad \text{and} \quad \mu_{pq} = \frac{\mu_p\sigma_q^2+\mu_q\sigma_p^2}{\sigma_p^2+\sigma_q^2}. \end{gathered} \tag{5}$$

In the previous equation, $p(z)$ and $q(z)$ are univariate normal PDFs with mean and variance μ_p, σ_q and μ_p, σ_q, respectively, and the scaling factor S_{pq} is itself a normal PDF on both μ_p and μ_q with standard deviation $\sqrt{\sigma_p^2+\sigma_q^2}$:

$$S_{pq} = \frac{1}{\sqrt{2\pi(\sigma_p^2+\sigma_q^2)}} \exp\left[-\frac{(\mu_p-\mu_q)^2}{2(\sigma_p^2+\sigma_q^2)}\right]. \tag{6}$$

In order to compute the intersection of the two original normal PDFs, we simply ignore the scaling factor.

Kullback–Leibler divergence For two distributions P and Q of a continuous random variable, the Kullback–Leibler (KL) divergence is defined to be the integral:

$$D_{\mathrm{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(z) \log \frac{p(z)}{q(z)}\, dz, \tag{7}$$

where p and q denote the probability densities of P and Q. The KL divergence measures how well a probability distribution Q represented another distribution P. In the specific case of normal distributions, the previous equation is [18]:

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= \frac{1}{2}\left[\log\frac{|\boldsymbol{\Sigma}_q|}{|\boldsymbol{\Sigma}_p|} - N + \mathrm{Tr}\left[\boldsymbol{\Sigma}_q^{-1}\boldsymbol{\Sigma}_p\right] + \left(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right)^T \boldsymbol{\Sigma}_q^{-1}\left(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right)\right] \overset{\text{uncorrelated}}{=} \\ &= \sum_i^N \log\sigma_{q_i} - \log\sigma_{p_i} - \frac{1}{2} + \frac{{\sigma_{p_i}}^2 + \left(\mu_{q_i} - \mu_{p_i}\right)^2}{2{\sigma_{q_i}}^2}. \end{aligned} \tag{8}$$

Note that $D_{\mathrm{KL}}(P\|Q)$ is technically not a distance metric, just a divergence. Also, $D_{KL}(p\|q)$ is asymmetric, so in the symmetric approach we use the symmetrized version $D_{\mathrm{KL}}(P,Q) = (D_{\mathrm{KL}}(P\|Q) + D_{\mathrm{KL}}(Q\|P))/2$.

Proper distance metrics between normal distributions can be defined. For example, the p^{th} Wasserstein distance between two distributions P and Q is generally defined as:

$$W_p(P,Q) := \left(\inf_{\gamma\in\Gamma(P,Q)} \int_{Z\times Z} d(z_p, z_q)^p \,\mathrm{d}\gamma(z_p, z_q)\right)^{1/p} \tag{9}$$

where $\Gamma(P,Q)$ denotes the collection of all measures on $Z \times Z$ with marginals P and Q on the first and second factors, respectively. When the underlying distance metric d is the l^2-norm distance function, the Wasserstein distance between normal distributions has the closed-form expression [43]:

$$\begin{aligned} W_2^2(P,Q) &= \left\|\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right\|^2 + \mathrm{Tr}\left[\boldsymbol{\Sigma}_p + \boldsymbol{\Sigma}_q - 2\left(\boldsymbol{\Sigma}_p^{\frac{1}{2}}\boldsymbol{\Sigma}_q\Sigma_p^{\frac{1}{2}}\right)^{\frac{1}{2}}\right] \overset{\text{uncorrelated}}{=} \\ &= \sum_i^N (\mu_{q_i} - \mu_{p_i})^2 + {\sigma_{p_i}}^2 + {\sigma_{q_i}}^2 - 2\sigma_{p_i}\sigma_{q_i}. \end{aligned} \tag{10}$$

Defining a distribution metric using an optimal transport approach has two main advantages. First, it results in an actual metric, as opposed to just a divergence or non-metric distance function. And second, it makes it very explicit what the underlying distance function d is in the latent space. However, it is less clear than in the KL case that minimizing the W_2 distance between normal distributions will result in a large overlap between them, as opposed to just spatial proximity. Therefore, we use KL divergence instead.

D.2 Box Embeddings

Box embeddings are N-dimensional hyperrectangles that can represent relationships such as intersection and containment. Boxes are parameterized by their two extreme points $z^\wedge$ and $z^\vee$. They are designed to represent unary and joint probabilities of events (segments, in our case), where large boxes represent highly probable and general concepts. The original paper [55] defines boxes as step functions (rectangles), but posterior papers smooth the edges of the boxes so that all pairs of boxes have positive intersections. Specifically, Li et al. [27] use Gaussian convolutions, and Dasgupta&Boratko et al. [14] improve on the previous paper using min and max Gumbel distributions. This results in better gradients during optimization, while keeping the intuition and parameters the same. We use the latter, which is conveniently implemented in an open source library [12].

We sample from a box by assuming the edges are hard instead of soft, and assuming a uniform distribution in the range $[z^\wedge, z^\vee]$. The volume of a box is computed as:

$$\mathrm{Vol}(A) = \prod_i^N \max(z_i^\vee - z_i^\wedge, 0) \tag{11}$$

In practice, the $\max$ operation is replaced by soft versions. The intersection between two boxes A and B can be computed as $z_{\cap}^{\wedge} = \max(z_a^{\wedge}, z_b^{\wedge})$ and $z_{\cap}^{\vee} = \min(z_a^{\vee}, z_b^{\vee})$. Note that if the intersection is zero (e.g. if $z_a^{\vee} < z_b^{\wedge}$), this will result in $z_{\cap}^{\wedge} > z_{\cap}^{\vee}$. Gumbel boxes [14] naturally handle these cases and the volume of such "negative" boxes is close to zero.

The conditional probability of one box given another one can be defined as:

$$P(A|B) = \frac{\text{Vol}(A \cap B)}{\text{Vol}(B)}. \tag{12}$$

When using the conditional approach (Section 2.4), the values we maximize in Eq. (1) are the conditional probability values obtained in Eq. (12). For the symmetric approach, box embeddings offer a variety of possibilities. We list a few of them next (Sim stands for "similarity function"):

- **Symmetric conditional**. $\text{Sim} = (P(A|B) + P(B|A))/2$
- **Intersection over Union (IoU)**. $\text{Sim} = \text{IoU} \rightarrow D = 1 - \text{IoU} = \text{Vol}(A \cup B)/\text{Vol}(A \cap B)$. This distance is known as the Jaccard distance, and it is a proper metric.
- **Sørensen–Dice coefficient**. $\text{Sim} = \text{Vol}(A \cup B)/(\text{Vol}(A) + \text{Vol}(B))$
- **Symmetric difference**. $\text{Sim} = \text{Vol}(A \triangle B) = \text{Vol}(A \cup B) - \text{Vol}(A \cap B)$.

For the values defined as similarities, we obtain the distance functions (not necessariliy metrics) as $D = 1 - \text{Sim}$.

E Additional Results and Experimental Details

Prediction into the future Our model is better than baselines for every time into the future that we tested. In Figure 10 we show a graph of error for future prediction, with respect to the time elapsed from the end of the past (input) segment.

Representing multiple futures In addition to Fig. 6 in the main paper, in Fig .9, we show how the evaluation results change depending on how many samples M we use at inference time (the reported value is the best out of the M predictions). Sampling $M > 1$ clearly improves the results, which demonstrates that our model represents (and predicts) more than one mode in the distribution. This applies not only to future prediction, but also to past and interpolation prediction.

Error bars We also report the quantitative results with standard deviations. At test time there are two factors of randomness. First, the input segments can be sampled at different times. Second, there is a sampling process to obtain trajectories z given segment representations Q. We run the test with 10 different random seeds, using the same seeds for all the experiments (this is, given a seed, both our method and the baselines use the same sampled segments). In Table 1 we report the average of the obtained values across the 10 random seeds. In this section we repeat the same results, but add information about the standard deviation across the seeds, shown in parentheses. See Table 4. Again, we report the l^2 error (the lower the better) across keypoints, after normalizing each trajectory to be contained in a region of size 100×100. FU, FR, P and I stand for "future uniform", "future random", "past" and "interpolation", respectively. The low standard deviation values imply that the significance of the average values reported in Table 1 is strong. The larger standard deviation values in the long version of the Diving48 dataset reflect the small number of test samples.

Ablations We report additional ablations of our framework in Tab. 3, for the FineGym short dataset. Specifically, we show the following ablations:

- **No re-encoding**. Same ablation as the main paper one, included here for completeness.
- **No trajectory loss**. We train without Eq. 1, which leads to a significant increase in prediction error.
- **Gaussian symmetric**. We use Gaussian distributions instead of box embeddings, and train using the symmetric scenario with KL divergence. The conditional case trained with box embeddings ("TrajRep (ours)" in the table) obtains better results, but that the symmetric case with Gaussian distributions is also competitive, and clearly outperforms the baselines.
- **Modify margin** α. The choice of α is rather arbitrary. Because the range of the distance function (for the distances used in our experiments) is in $[0, \infty)$, α influences the norm of

Table 3: Ablations on the FineGym short dataset. Values represent mean squared error. See Appendix E for a discussion.

	F	P	I
TrajRep (ours)	6.20	6.36	4.88
w/o re-encoding	6.49	6.59	5.15
w/o trajectory loss (Eq. 1)	7.37	7.55	7.02
w/ Gaussian symmetric	6.64	6.65	5.05
w/ margin $\alpha = 0.1$	**6.18**	**6.32**	4.89
w/ margin $\alpha = 10$	6.25	6.40	**4.83**
w/ all hard positives	6.32	6.51	5.03
w/ uniform sampling	6.58	6.70	5.36
w/ ST-GCN	6.77	6.99	5.26

the distances, but not the relative distances between segments. We run two experiments with different values of α (0.1 and 10), on top of the default $\alpha = 1$ and show that the model performance is not very sensitive to this hyperparameter.

- **All hard positives**. We replace all soft positives by hard negatives, in order to assess how important is to distinguish them. We notice an increase in error with respect to the original model, showing that distinguishing between hard and soft negatives has some influence in the final model.
- **Uniform sampling**. We train using uniform time-steps, and test regularly, with irregular time-steps. This model does not generalize as well to irregular time-steps as the one trained directly on irregular time-steps.
- **ST-GCN**. We implement the encoder Θ using an ST-GCN network [58] instead of a Transformer. ST-GCN is the most established model to process temporal skeleton data. We use the implementation in Yan et. al. [59]. ST-GCN performs worse than the Transformer network (although the results are competitive), probably because temporal information cannot be added to an out-of-the-box ST-GCN architecture.

Stable optimization The optimization process is stable across the training. We show loss curves in Fig. 8 both for $\mathcal{L}_{\text{enc}}$ in Eq. 1 (trajectory loss) and for $\mathcal{L}_{\text{dec}}$ in Eq. 2 (reconstruction loss).

Fig. 5 details We find the directions in Fig. 5 by sampling segments from the test set, and for every segment, we modify their time indices in a progressive way, obtaining 10 modified segments for each original one. These modified time-steps are created either by multiplying their time indices by a constant (for Fig. 5a), or by adding a constant to their time indices (for Fig. 5b). We compute their representations, and for every set of segments we find the main direction of variation across the set by computing the first PCA component. The overall direction is found by averaging these directions across the different test-set samples. This overall direction (for example, the direction representing "speed change") is general for all trajectories, not trajectory-specific, and therefore we can apply it to any encoded segment.

Baseline details Regarding the baselines, the Trajecton++ models the GMM with a first step consisting of a Categorical random variable, followed by a normal distribution. In order to selec the $M = 10$ future predictions, we sample the top-10 options in the Categorical variable, and for each one we sample once from the Gaussian. Note that this is exactly how the Trajectron++ is trained, both in the original paper and in our implementation. For the VRNN baseline, at test time we sample the 10 samples in the first time-step, and then sample the mode of the latent distributions for the rest of the samples.

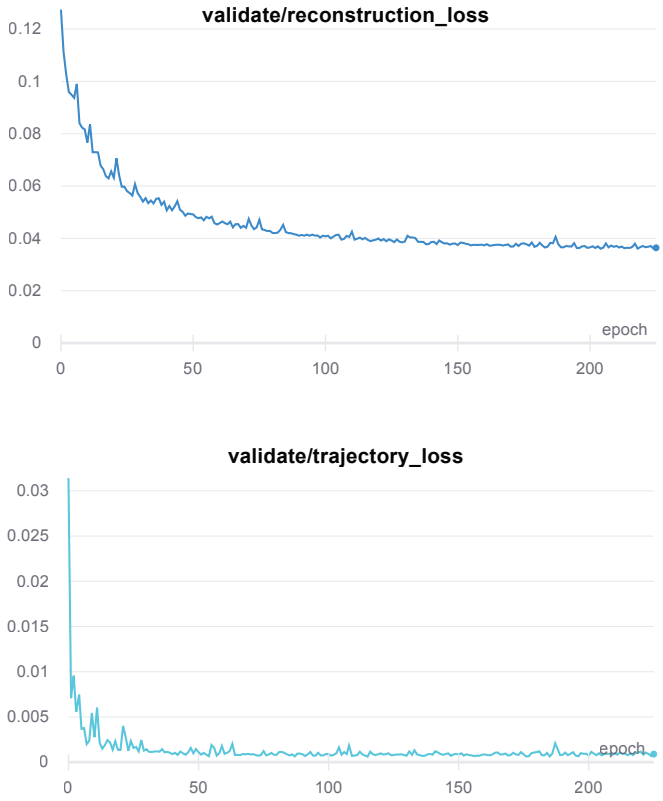


Figure 8: Loss curves during training, corresponding to the FineGym short experiment, trained with bounding boxes.

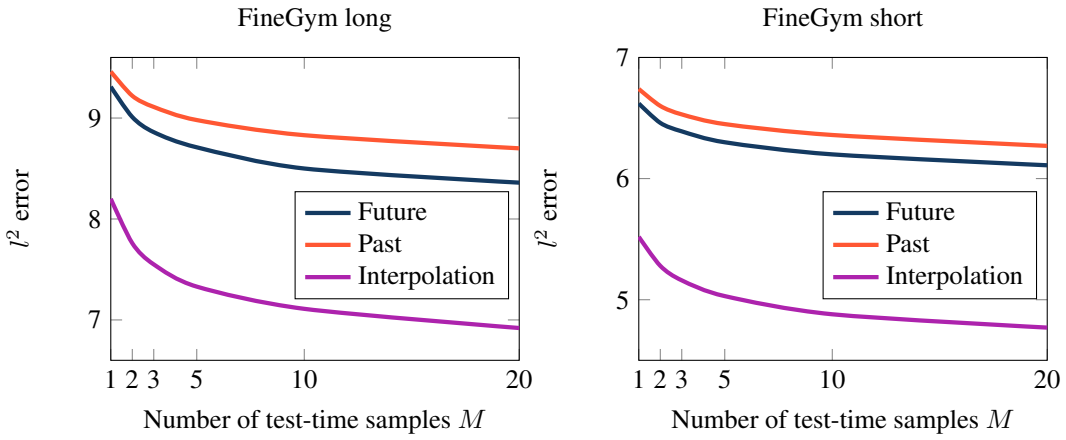


Figure 9: In our evaluation, we report the *Best-of-M* samples, where $M = 10$. In this figure, we show how the evaluation results change for different choices of M. These plots show how our model captures different modes in the distribution, as adding more samples results in better values, indicating that the samples result in different trajectories. This is qualitatively reinforced by Fig. 6. The figure on the left corresponds to the FineGym long dataset, and the figure on the right corresponds to the FineGym short dataset.

Table 4: Results from Table 1 extended with standard deviation information.

(a) **FineGym - Long sequences**

	F	P	I
VRNN [13]	15.85 (0.00)	15.93 (0.00)	16.10 (0.00)
Trajectron++ uni. [44]	9.54 (0.01)	9.98 (0.01)	9.73 (0.00)
Trajectron++ [44]	9.72 (0.00)	10.01 (0.00)	9.89 (0.00)
TrajRep (ours, ablation)	8.82 (0.01)	9.07 (0.00)	7.57 (0.00)
+ re-encoding (ours)	8.50 (0.01)	8.83 (0.00)	7.11 (0.00)

(b) **Diving48 - Long sequences**

	F	P	I
VRNN [13]	23.51 (0.13)	27.97 (0.17)	25.66 (0.06)
Trajectron++ uni. [44]	11.67 (0.22)	16.52 (0.55)	11.98 (0.10)
Trajectron++ [44]	11.59 (0.12)	16.23 (0.21)	12.68 (0.09)
TrajRep (ours, ablation)	10.00 (0.13)	11.74 (0.17)	10.06 (0.13)
+ re-encoding (ours)	9.81 (0.16)	12.00 (0.11)	9.58 (0.18)

(c) **FisV - Long sequences**

	F	P	I
VRNN [13]	14.95 (0.00)	15.03 (0.01)	15.08 (0.00)
Trajectron++ uni. [44]	11.42 (0.01)	11.85 (0.01)	11.68 (0.01)
Trajectron++ [44]	11.41 (0.00)	11.71 (0.00)	11.63 (0.00)
TrajRep (ours, ablation)	10.62 (0.01)	11.27 (0.01)	9.70 (0.00)
+ re-encoding (ours)	10.32 (0.01)	10.77 (0.00)	9.22 (0.00)

(d) **FineGym - Short sequences**

	F	P	I
VRNN [13]	12.77 (0.00)	13.20 (0.01)	13.40 (0.00)
Trajectron++ uni. [44]	7.80 (0.01)	8.28 (0.01)	7.48 (0.01)
Trajectron++ [44]	7.26 (0.01)	7.93 (0.01)	6.94 (0.00)
TrajRep (ours, ablation)	6.49 (0.00)	6.59 (0.01)	5.15 (0.00)
+ re-encoding (ours)	6.20 (0.01)	6.36 (0.01)	4.88 (0.00)

(e) **Diving48 - Short sequences**

	F	P	I
VRNN [13]	18.36 (0.05)	20.14 (0.07)	19.86 (0.02)
Trajectron++ uni. [44]	9.05 (0.02)	10.36 (0.05)	8.29 (0.03)
Trajectron++ [44]	8.74 (0.03)	11.35 (0.03)	8.31 (0.03)
TrajRep (ours, ablation)	6.94 (0.03)	6.99 (0.03)	5.00 (0.02)
+ re-encoding (ours)	6.76 (0.02)	6.85 (0.03)	5.04 (0.02)

(f) **FisV - Short sequences**

	F	P	I
VRNN [13]	13.26 (0.01)	13.44 (0.01)	13.45 (0.00)
Trajectron++ uni. [44]	9.23 (0.01)	9.68 (0.01)	8.86 (0.01)
Trajectron++ [44]	8.70 (0.01)	9.28 (0.01)	8.28 (0.00)
TrajRep (ours, ablation)	7.83 (0.01)	8.17 (0.01)	6.01 (0.01)
+ re-encoding (ours)	7.54 (0.01)	7.78 (0.01)	5.88 (0.01)

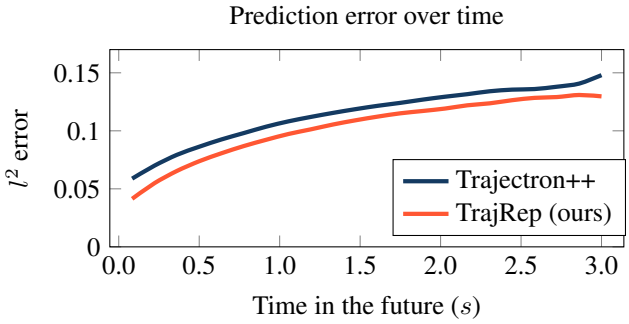


Figure 10: l^2 error with respect to the the time elapsed from the past (input) segment, where we evalute future prediction ($t = 0$ represents the end of the past segment). This result corresponds to the FineGym long dataset.