A Mixture of Experts Approach to 3D Human Motion Prediction

Edmund Shieh*, Joshua Lee Franco *, Kang Min Bae*, Tej Lalvani*
Georgia Institute of Technology
eshieh3@gatech.edu, jfranco33@gatech.edu, kbae36@gatech.edu, tlalvani3@gatech.edu

Abstract

This project addresses the challenge of human motion prediction, a critical area for applications such as autonomous vehicle movement detection. Previous works have emphasized the need for low inference times to provide real time performance for applications like these [1, 2, 3]. Our primary objective is to critically evaluate existing model architectures, identifying their advantages and opportunities for improvement by replicating the state-of-the-art (SOTA) Spatio-Temporal Transformer model as presented by Aksan et al. [4] as best as possible given computational constraints. These models have surpassed the limitations of RNN-based models and have demonstrated the ability to generate plausible motion sequences over both short and long term horizons through the use of spatio-temporal representations. We also propose a novel architecture to address challenges of real time inference speed by incorporating a Mixture of Experts (MoE) block within the Spatial-Temporal (ST) attention layer, inspired by Lepikhin et al. [5]. The particular variation that is used is Soft MoE [6], a fully-differentiable sparse Transformer that has shown promising ability to enable larger model capacity at lower inference cost.

1. Introduction

Human motion prediction is a critical component in various applications such as autonomous vehicles and interactive robotics, where understanding and anticipating pedestrian movements can drastically improve safety and interaction dynamics. Traditional methods primarily utilize Recurrent Neural Networks (RNNs) due to their ability to process sequential information. However, these models often struggle with long-term dependencies and high inference times, which are crucial for real-time applications [7, 8].

With the advancement of attention mechanisms, particularly the introduction of Transformers, new pathways have opened for handling sequential prediction tasks with improved accuracy and efficiency. The Spatial-Temporal (ST) Transformer model has emerged as a promising solution by effectively incorporating spatial and temporal information using a joint representation, which surpasses the limitations of RNN-based models in generating plausible motion sequences over varying horizons [4].

Despite the successes of ST Transformers, real-time performance remains a challenge due to the the complexity and auto-regressive inference pattern of the models. To address this, we propose a novel architecture that incorporates a MoE within the ST Transformer's attention layers. This approach aims to optimize the inference speed by dynamically selecting the most relevant model components during the prediction process, drawing inspiration from recent advancements in scalable neural network technologies [5]. Current trends are gravitating towards increasing model complexity and expanding dataset sizes. This escalation necessitates enhanced computational power. Integrating a Mixture of Experts (MoE) with Spatial-Temporal (ST) Transformers could provide a strategic advantage by scaling model complexity while maintaining swift inference times. This approach aligns with the increasing demands for more sophisticated models, as MoE allows the model to selectively activate only the relevant parts of the network for specific tasks. By combining MoE with ST Transformers, which are adept at processing time-related data, the models can efficiently handle complex applications like human motion prediction. If successful, this integration could significantly benefit sectors such as autonomous vehicles, interactive robotics, virtual reality, and gaming, where rapid and accurate predictions are crucial for both safety and enhancing user experiences.

For our experiments, we utilized the AMASS (Archive of Mocap as Surface Shapes) DIP dataset hosted by the Max Planck Institute for Intelligent Systems [9], which consolidates various human motion capture datasets into a single repository, significantly simplifying the preprocessing and usage of mocap data for research purposes. Comprising 8,593 sequences, AMASS encompasses over 9 million frames sampled at 60 Hz translating to approximately 42 hours of recorded motion. Each joint in the dataset can be

*Alphabetical order

represented using different formats; for our study, we chose the axis angle (aa) representation, where each joint's orientation is described by a 3-dimensional vector $E \in \mathbb{R}^3$ [7].

We employed the fairmotion library [10] for preprocessing the raw motion sequences into a structured format and used the dataset splits defined in the work [11] for training, validation, and testing of the motion prediction sequence modeling task. The task is approached as a sequence modeling problem, where the input consists of 120 consecutive poses, equivalent to two seconds of motion captured at a frequency of 60Hz. The output is defined as a sequence of 24 poses, representing 400 milliseconds of motion.

This report details our efforts to replicate the state-of-the-art ST Transformer model, assess its performance across diverse motion datasets, and introduce our novel MoE-enhanced ST Transformer model. Through rigorous testing and analysis, we aim to demonstrate the efficacy of our approach in reducing inference times while maintaining high accuracy in motion prediction.

2. Approach

Our approach was a three-step process. The initial step was to get the baseline performance from the existing models. Then compare the performance to the current state of the art. Finally, explore ways to enhance the current state of the art.

There were five models provided by the Facebook research team [10] in their motion prediction repository. seq2seq, basic seq2seq using encoder decoder; tied seq2seq, seq2seq with same LSTM used in both encoder and decoder; RNN, comprised of a single RNN; transfer encoder, which is a transformer-LSTM hybrid model; lastly, transformer. With these models, each team member trained a model using the default parameter to get our baseline.

After getting our baselines, we wanted to compare the performance between the existing models and the current state-of-the-art, ST Transformer [4]. In order to do this, we implemented it in PyTorch using a repository provided by the authors written for tensorflow as a foundation[12]. To get the optimal performance, we tuned several hyperparameters with various values to get the optimal model with the best settings. The hyperparameter that we tuned were batch size, optimizer, Dimensional joint embedding size, Hidden dimension, and number of layers.

Finally, taking inspiration from the promising results shown in [5], we swapped out the dense feed forward layers within the transformer block of the vanilla ST transformer and replaced them with a Soft MoE block [13] and then compared the inference performance between the two under different hyperparameters.

Joint Embedding

The sequence of poses $\mathbf{x}$ represents a tensor in $\mathbb{R}^{T \times S \times M}$, where T is the number of frames in the input sequence (120 frames), S is the number of joints (24 joints), and M is the dimension of the axis angle representation for each joint (3 dimensions). This sequence is processed through a linear layer, which replaces the traditional embedding lookup table. This transformation outputs embeddings $\mathbf{E} \in \mathbb{R}^{T \times S \times E}$. Here, the input dimension M is transformed to the model's working dimension E an adjustable hyperparameter to tune complexity of the effective joint embedding.

Positional Encoding

A Positional Encoding module is utilized to incorporate temporal information into the embeddings, enhancing the model's ability to understand sequence order. The positional encoding uses sinusoidal functions as proposed in section 3.5 of Vaswani et al. [14], which are calculated according to equations 2 and 1.

Spatial and Temporal Multi-head Attention

The core of the model lies in its dual attention mechanism as depicted in Figure 1 of [4]:

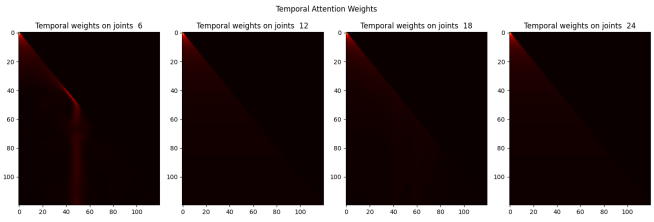


Figure 1: **Temporal Attention** map of temporal attention weights by timesteps in joints 6, 12, 18, 24

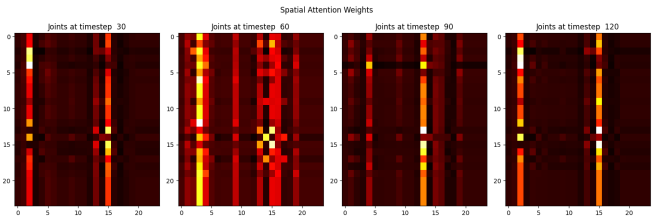


Figure 2: **Spatial Attention** map of spatial attention weights by joints 30, 60, 90, 120 timesteps

1. **Temporal Attention**:

   - The input x is reshaped and transposed to align for temporal attention, resulting in a new shape of $T \times (S \cdot E)$. This allows the multi-head attention layer to processes each joint independently

across the time dimension, ensuring that each pose contributes to the final prediction without peeking at future poses.

- A masked multi-head attention is applied to incorporate causal dependency, allowing each time step to only attend to previous and current steps.
- Shown in figure 1 is a map of self attention weight output of spatial multi head attention. It seems that the temporal attention patterns are similar across joints. In particular, most joints exhibit the temporal self-attention pattern in which the current timestep has notable attention weights with timesteps that just precede it (< 5 frames back). This is demonstrated by a bright diagonal line on the heat maps. Furthermore, the upper right triangle is completely dark indicating successful causal masking in which a given timestep does not attend to future timesteps. Counter to our SOTA implementation, Aksan et al. [4] SOTA model demonstrates different joints exhibiting different temporal attention patterns which are likely due to a more complex model being trained using higher number of heads in multi-head self attention and more layers allowing each head/layer to capture specific nuances for each joint.

2. **Spatial Attention**:

- The original input x is also reshaped to put spatial features at the forefront, resulting in a tensor shaped $S \times (T \cdot E)$. This allows a separate multi-head attention layer to operate across all joints within a frame jointly attending to information from different joint embedding subspaces, allowing the model to capture the complex interdependencies between joints.
- Shown in figure 2 is a map of self attention weight output of temporal multi head attention. This figure highlights that different joints exhibit different spatial attention patterns at different timesteps in line with the SOTA from [4]. We observe there are some important joints at each timestep that have high attention weights for all other joints. For example, joint 3 at timestep 60 has a bright column indicating it is attended to by most joints suggesting that this particular joint has high relevance to the model's prediction.

Both attention mechanisms are implemented using the `MultiheadAttention` PyTorch module which implement multi-head attention as described in [14]. The two attention outputs are then reshaped and added together. Note that the SOTA implements weight matrix sharing for key and value weights across joints whereas our implementation does not due no in-built support for this in `MultiheadAttention`.

Feedforward Layers (Vanilla ST Transformer)

For the vanilla ST Transformer, each attention output is then processed through two sequential linear transformations interspersed with a ReLU activation function. The sequence is as follows:

$$\text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Linear}.$$

This feedforward network is applied independently to each position, followed by dropout and residual connections to stabilize the learning process.

Soft MoE layer (MoE ST Transformer)

Sparse MoEs are shown to be able to scale model capacity in vision tasks without large increases in training or inference costs via processing inputs through a gating mechanism that selects which 'expert' networks to activate based on the input [15]. The layer can be represented as:

$$\mathbf{y} = \sum_{i=1}^{n} g_i(\mathbf{x}) \mathbf{E}_i(\mathbf{x}),$$

where $g_i(\mathbf{x})$ are the gating functions determining the weights for each expert's contribution, and $\mathbf{E}_i(\mathbf{x})$ represents the output from the i-th expert. The top k out of n experts are chosen to be active for any given input (where $k \ll n$). This reduces computational overhead significantly by activating only a small subset of the available experts, in contrast to dense layers that use all weights and biases for every input.

Soft MoE [6], an adaptation of the Sparse MoE, addresseses challenges like training instability and inability to scale the number of experts. Soft MoE performs an implicit soft assignment of input tokens to each expert. It utilizes slots which are essentially weighted averages of all input tokens, as determined by learned parameters. The slots are then processed by the experts, potentially lowering the computational cost even further while maintaining or enhancing model capacity and efficiency. This is achieved by computing weighted averages of all input tokens, where the weights are determined by the interaction between input tokens and a learned parameter matrix. This effectively results in each expert processing a subset of the data that it's specialized on as shown in 3, lowering computational cost.

The Soft MoE layer was implemented using this repository [13] for the corresponding paper [6]. It was used in place of the feed-forward layer for the MoE ST Transformer as shown in Figure 4.

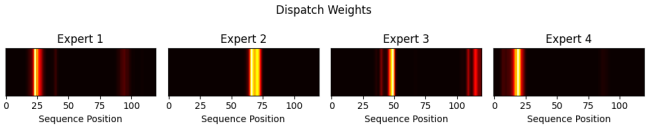


Figure 3: **MoE ST Transformer Dispatch Weights** Dispatch weights extracted from the MoE layer in a single forward pass of the trained MoE ST Transformer on an input sequence from the test split. The weights demonstrate a clear routing mechanism for the data to be processed by a specialized expert.

Normalization and Residual Connections

Normalization is applied after each sub-layer (attention, feedforward and MoE), using `LayerNorm` module in PyTorch to stabilize the training dynamics. Residual connections are also used extensively to facilitate gradient flow and mitigate the vanishing gradient problem.

Projection Layers

The output of the transformer is fed into two sequential projection layers to refine and reshape the neural network's output to the required dimensions for the task of next frame prediction.

1. `project_1` Layer:
    - The `project_1` layer is primarily responsible for reducing the dimensionality of the encoder's output back to the original axis angle representation dimension size.
2. `project_2` Layer:
    - Following the initial projection, the `project_2` layer further processes the data by mapping it to the final output dimensions required by the model which is a single frame, effectively collapsing the sequence length dimension of the tensor using learned weights from this linear layer.

Autoregressive Inference

The model operates by maintaining a rolling window of the last 120 frames, which corresponds to 2 seconds of data, given the frame rate. This window is used to predict the next frame in the sequence. Mathematically, the prediction for the next frame can be expressed as:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_{t-119:t}),$$

where $\mathbf{x}_{t-119:t}$ denotes the frames from $t-119$ to t, and f represents the predictive function of the model.

Once the next frame is predicted, the model updates its frame window by discarding the oldest frame and including the newly predicted frame. This shift allows the model to continuously predict new frames based on the most recent data, including its own predictions. The model repeats this process to predict a total of 24 frames, corresponding to 400 milliseconds. This sequence of predictions is generated autoregressively, with each new frame prediction incorporating the most recent frame into the window.

2.1. Challenges

During the implementation of the spatio-temporal transformer model, several challenges were encountered which required specific solutions to ensure the efficiency and stability of the training process. Below are the main points:

- **Numerical Stability:** We observed that the loss occasionally collapsed to zero, leading to NaN values during training. This was potentially due to exploding gradients. To address this issue:
    - We modified the teacher forcing ratio calculation by adding a small dummy value to prevent it from reaching zero, thus ensuring numerical stability.
- **Computational Constraints:** The training of our model was limited by the computational constraints imposed by Partnership for an Advanced Computing Environment (PACE) clusters, which includes a maximum of 512 CPU hours and 16 GPU hours per job, with a maximum walltime of 18 hours for CPU jobs and 16 hours for GPU jobs. Given this and high queue times for GPUs:
    - We developed a functionality within our training script to automatically save progress and resume training from the last saved checkpoint. This ensured continuity in model training despite time constraints.
    - Additionally, GPU memory emerged as a significant constraint, particularly for large models with a high number of parameters, such as the MoE ST transformer. To mitigate this, we employed a strategy of using float32 rather than double precision, despite knowing that it could result in worse performance. This trade-off was necessary to fit the model within the available memory constraints while still achieving reasonable training times. Additionally, we were unable to replicate the high parameter count of the SOTA using the same number of multi attention heads, layers and dimension sizes given these memory constraints and so a slimmer model was chosen instead.
- **Handling Large Datasets:** Due to the large dataset sizes and the model's complexity, the expected training times were significantly lengthy, risking frequent interruptions. To mitigate this:

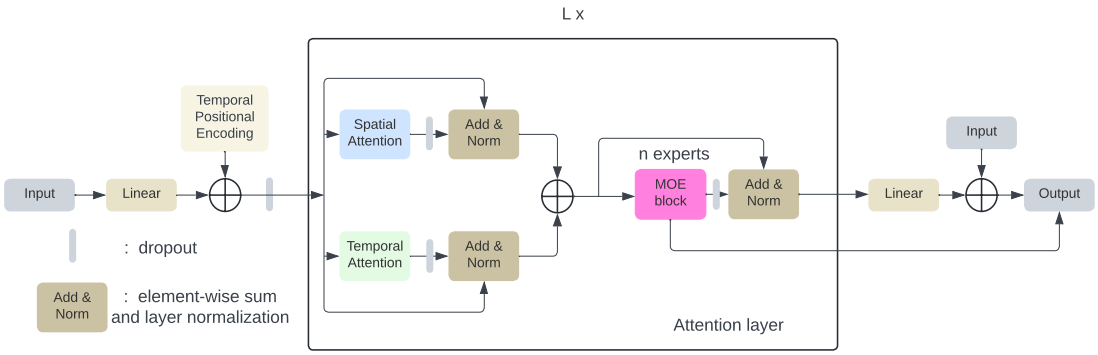


Figure 4: **Architecture Overview** Adapted version of ST transformer from Figure 2 in [4] with feed-forward layers in attention layers swapped with MoE block (highlighted in pink)

- We employed PyTorch's DataDistributedParallel module along with Hugging Face's Accelerate API to facilitate multi-node and multi-GPU training, significantly enhancing computational efficiency and reducing overall wall clock time.
- We also enhanced the existing code with a load function to continue from the last saved epoch. This addressed the issue of interruptions due to reaching the maximum wall time.

3. Experiments and Results

3.1. Success Criteria

We had two primary goals: firstly, to successfully implement the state-of-the-art ST Transformer, as described in [4], with appropriate hyperparameter tuning to surpass baseline models. Secondly, to implement a novel architecture, the MoE ST Transformer, using [13], to compare its performance against the current state of the art and assess any gains in inference efficiency. Both goals were achieved, with details provided below.

3.2. Experiment 1

3.2.1 Base Model Performance Analysis

Our team successfully trained the base models provided by [10] using the AMASS dataset [16] consisting of more than forty hours of motion captured data. We used the default hyperparameter values consisting of values shown in table 3 and the results are shown in table 2. We were surprised to find that the seq2seq model outperformed transformer and transformer encoder models in all criterias including MAE, training loss and validation loss. This may indicate that the default hyper-parameter values in 3 better suits the seq2seq model than the transformer models. But it may be worth noting that training time on A100 for transformers were averaging about 2 minutes per epoch whereas seq2seq training on two A100s took about 4 minutes per epoch as expected.

3.2.2 Hyper-Parameter tuning for ST Transformer

The hyper parameter tuning results can be seen in 1. Default values were 64 for joint embedding dimension; 1 for number of layers; 120 for source length; 64 for batch size; adam for optimizer; twenty epochs. Then for each hyper parameter, we adjusted the value while leaving all else as default values to measure the impact of change in given hyper parameter value. The best hyperparameter values can be seen on table 4. For some of the hyper parameters such as hidden dimension like batch size, source length, and hidden layers, we chose broad ranges as we wanted to get an idea if increasing or decreasing these values would lead to a significant increase in performance. For values like joint embedding dimension size and number of layers were tuned in small ranges to capture narrower performance comparisons. We also observed that hyperparameters such as joint dimension sizes and hidden dimensions require higher GPU ram as they increased. Which is why we did not increase these values above certain limit. After hyperparameter tuning we observed that the ST transformer outperformed most of the base models as shown on table2 as a whole as it had the lowest training loss and validation loss compared to seq2seq although they were similar in MAE values. Thus we ruled this experiment to be a success.

3.3. Experiment 2

3.3.1 ST Transformer vs MoE Inference Time Ablation

Table 7 presents the run times observed for each configuration during inference. The parameter varied for the ST Transformer was the hidden dimension of the feedforward layer that follows the spatio and temporal attention layers in the model. For the MoE ST Transformer which has these feedforward layers replaced by an MoE block, the param-

eter varied is the number of experts. All other parts of the architecture and computing resources remained the same.

We observe in figure 5 the ability for the MoE ST transformer to scale it's number of parameters without the same proportional increase in inference time compared to the Vanilla ST Transformer. A single Nvidia Quadro Pro RTX6000 GPU was used for this experiment. We use one slot per expert when scaling the MoE ST transformer as optimally recommended by Puigcerver et al. [6].

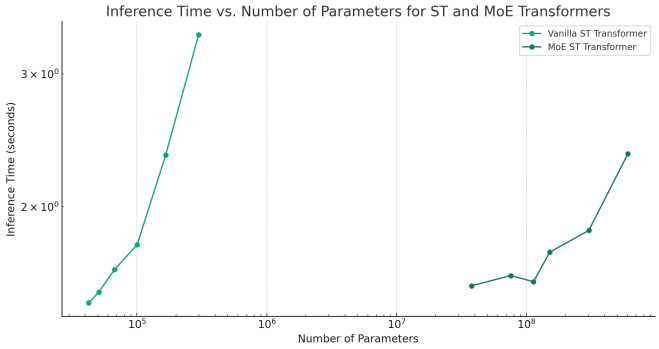


Figure 5: Graphical visualization of table 7 to demonstrate scalability of MoE and ability to handle large number of parameters

Traditional neural network models, such as the STtransformer, exhibit increased computational complexity and longer inference times as the hidden dimension size expands. This scaling effect is standard in neural networks where larger models require more computation per input, directly impacting performance metrics such as inference time.

The MoE model introduces a different architectural approach where only a subset of the model, termed "experts," is active at any given time. This design allows the model to scale in complexity, such as increasing the number of experts, without a corresponding linear increase in computational demand. As a result, inference times remain relatively stable, making the MoE model more scaleable and efficient for larger configurations.

Central to the SoftMoE architecture are two distinct types of weights, each serving a crucial role within the model: dispatch weights and combine weights. These weights are integral to the model's scalability and efficiency:

1. **Dispatch Weights:** Responsible for routing input sequences to the appropriate experts, dispatch weights are determined by a gating network that evaluates each part of the input sequence. The weights decide which expert should process which frames within a sequence, ensuring that experts are effectively specialized to handle phases of the input sequence as seen in figure 3.
2. **Combine Weights:** After the input has been processed by the designated experts, combine weights are used to aggregate the outputs from these experts. These weights determine the contribution of each expert's output to the final prediction, allowing for an effective synthesis of expert advice to achieve the most accurate overall output.

Through these, the model dynamically adapts computational load across different experts, thereby maintaining efficiency despite increases in model parameters or complexity.

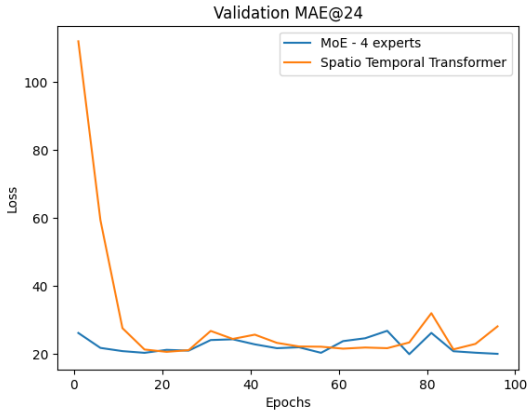


Figure 6: Validation MAE@24 of ST and MoE Transformers

We observe in figure 6 that the ST and MoE Transformers have similar performance on the validation set. This demonstrates how the MoE Transformer is able to scale without a reduction in performance.

3.4. ST Transformer vs MoE evaluation

We can see overall that MoE was able to perform well against the current state of the art ST Transformer. In table 6, both models outperformed the base models. This is to be expected, as the ST transformer incorporates both spatial and temporal attention as described above. This allows the model to better capture patterns in the motion data. The effectiveness of these models can be further seen in figures 12, 13, 14, and 15. In these figures, we see a ground truth motion compared to a predicted motion. From the perspective of joint distance, both models come close to predicting the reference motion given an input sequence preceding the motion. There is still room for improvement on this approach, though a promising area of improvement that MoE showed over the ST Transformer was in scalability of hyperparameters and its affect on inference timing discussed in 3.3.1.

References

[1] Alireza Shafaei and James J. Little. Real-time human motion capture with multiple depth cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Vancouver, Canada, 2016. The University of British Columbia.

[2] Xinyu Yi, Yuxiao Zhou, Marc Habermann, Vladislav Golyanik, Shaohua Pan, Christian Theobalt, and Feng Xu. Egolocate: Real-time motion capture, localization, and mapping with sparse body-mounted sensors. *arXiv*, abs/2305.01599, 2023. URL https://ar5iv.labs.arxiv.org/html/2305.01599.

[3] Avinash Ajit Nargund and Misha Sra. Spotr: Spatio-temporal pose transformers for human motion prediction, March 2023. URL https://arxiv.org/abs/2303.06277.

[4] Emre Aksan, Manuel Kaufmann, Peng Cao, and Otmar Hilliges. A spatio-temporal transformer for 3d human motion prediction. *arXiv preprint arXiv:2004.08692*, 2020. URL https://arxiv.org/abs/2004.08692.

[5] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. arXiv preprint arXiv:2006.16668, 2020. URL https://arxiv.org/abs/2006.16668.

[6] Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts, 2023. URL https://arxiv.org/abs/2308.00951.

[7] Julieta Martinez, Michael J. Black, and Javier Romero. On human motion prediction using recurrent neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4674–4683, 2017.

[8] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4346–4354. IEEE, 2015.

[9] Yinghao Huang, Manuel Kaufmann, Emre Aksan, Michael J. Black, Otmar Hilliges, and Gerard Pons-Moll. Deep inertial poser: Predicting full body pose from sparse inertial measurements. Max-Planck-Gesellschaft, 2020. URL https://dip.is.tue.mpg.de/.

[10] Deepak Gopinath and Jungdam Won. fairmotion - tools to load, process and visualize motion capture data. Github, 2020. URL https://github.com/facebookresearch/fairmotion.

[11] Emre Aksan, Manuel Kaufmann, and Otmar Hilliges. Structured prediction helps 3d human motion modelling. *arXiv preprint arXiv:1910.09070*, 2019. URL https://arxiv.org/pdf/1910.09070.pdf.

[12] eth-ait. Motion transformer. https://github.com/eth-ait/motion-transformer, 2024. Accessed: 2024-04-23.

[13] Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. Soft moe (mixture of experts). GitHub repository, 2023. URL https://github.com/lucidrains/soft-moe-pytorch.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, Dec 2017. URL https://arxiv.org/abs/1706.03762.

[15] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *arXiv preprint arXiv:2106.05974*, 2021. URL https://arxiv.org/abs/2106.05974.

[16] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. Amass: Archive of motion capture as surface shapes. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019. URL https://amass.is.tue.mpg.de.

[17] Dario Pavllo, David Grangier, and Michael Auli. Quaternet: A quaternion-based recurrent model for human motion. In *British Machine Vision Conference 2018 (BMVC 2018)*, page 299, Newcastle, UK, Sept 3–6 2018. Northumbria University.

A. Project Code Repository

The GitHub repository available at https://github.com/edshieh/motionprediction extends the open-source `fairmotion` library, which provides foundational scripts and baseline models for motion prediction. The repository introduces several significant enhancements to facilitate more advanced and flexible research in motion prediction technologies. Key developments include:

- Implementation of a PyTorch-based spatio-temporal (ST) transformer, which allows for advanced modeling of temporal dynamics and spatial relationships.
- Introduction of a mixture of experts within the ST transformer framework, enhancing the model's ability to handle diverse data scenarios by leveraging specialized sub-models.
- Enhanced training flexibility by allowing adjustments to hyperparameters such as the number of heads (`num_heads`), layers (`num_layers`), and experts (`num_experts`) in the ST transformer.
- Improved inspection capabilities for attention and mixture of experts weights, providing deeper insights into the model's decision-making process.
- Addition of experimentation scripts designed to test inference times, aiding in the evaluation of model efficiency under different computational constraints.

B. Model Specifics

Positional Encoding:

$$PE(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \quad (1)$$

$$PE(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \quad (2)$$

Where $PE(\text{pos}, 2i)$ and $PE(\text{pos}, 2i+1)$ represent the sine and cosine components of position encoding in a transformer model, respectively.

Datatype Precision: We experimented with both float32 and float64 data-types for training different architectures. Across the board, we found that while training with float64 had a larger memory overhead and a longer training time, the validation MAE was lower than an identical model trained with float32. As seen in figure 7, this makes a noticeable difference for different architectures. This is likely due to numerical stability of float64 given its higher precision, which ultimately leads to more precise gradients. In the interest of time, we trained all our models using float32. If performance was our goal, we would have opted to use float64.

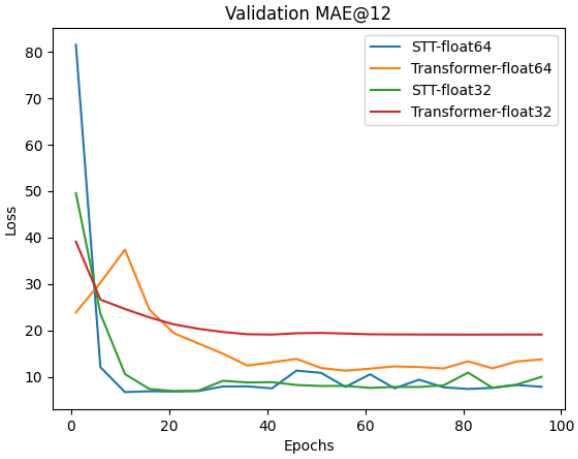


Figure 7: Comparison of float32 vs. float64 using validation MAE@12

C. Training Details

Learning Rate Equation: The learning rate schedule is adapted to the training progress and is calculated using the following equation [14]:

$$\text{learning rate} = D^{-0.5} \cdot \min\left(step^{-0.5}, step \cdot warmup^{-1.5}\right), \quad (3)$$

where D is the dimensionality of the model, $step$ denotes the current training step, and $warmup$ is the number of steps during the warm-up phase. This adaptive learning rate helps in stabilizing the training in the initial phases and gradually fine-tuning the model as training progresses. The default parameters were used from the noamopt option from the fairmotion library [10].

Loss Function for 3D Motion Prediction: The Mean Squared Error (MSE) loss was used as the loss criterion. Mathematically, the MSE loss for a prediction involving multiple joints across several time steps is defined as follows:

$$L = \frac{1}{N \cdot J \cdot 3} \sum_{t=1}^{N} \sum_{j=1}^{J} \sum_{k=1}^{3} \left(\theta_{\text{tgt},t,j,k} - \theta_{\text{pred},t,j,k}\right)^2, \quad (4)$$

where $N = 24$ represents the sequence v length, $J = 24$ denotes the number of joints, and k indexes the three components of the axis-angle representation of each joint. Here, $\theta_{\text{tgt},t,j,k}$ and $\theta_{\text{pred},t,j,k}$ are the target and predicted values, respectively.

Error Metric for Assessing 3D Motion Prediction: Mean angle error (MAE) was used as the error metric to

calculate the deviation between predicted and target pose. The mathematical formulation for the Euler angle error is as follows:

$$E = \sqrt{\sum_{i \in \mathcal{I}} \left(\text{Euler}(R_{\text{tgt},i}) - \text{Euler}(R_{\text{pred},i})\right)^2}, \quad (5)$$

where $R_{\text{tgt},i}$ and $R_{\text{pred},i}$ are the target and predicted rotation matrices for joint i, respectively, and $\text{Euler}(\cdot)$ converts a rotation matrix to its Euler angle representation. The index set $\mathcal{I}$ includes only those joints for which the standard deviation of the target Euler angles exceeds a small threshold, indicating significant motion and thus relevance for the prediction accuracy.

Dynamic Teacher Forcing Ratio: Pavllo et al. [17] suggests to use teacher forcing to expose the model to its own predictions to mitigate exposure bias, which occurs when there is a discrepancy between the training regime (where the model always sees the true previous output) and inference (where the model only has access to its own predictions). By gradually exposing the model to its own predictions during training, teacher forcing helps the model better handle the sequential dependencies and learn more robust representations. During the forward pass of the network, at each timestep, the decision to apply teacher forcing is probabilistic, governed by the current value of the teacher forcing ratio. Conceptually, this can be represented as:

$$\text{Input at } t = \begin{cases} \text{True Output at } t-1 & \text{with p=TFR} \\ \text{Model's Prediction at } t-1 & \text{otherwise} \end{cases}$$

The degree to which teacher forcing as recommended by is used is controlled by the *teacher forcing ratio*, which is a function of the training progress:

$$\text{Teacher Forcing Ratio} = \max\left(0, 1 - \frac{\text{current epoch}}{\text{total effective epochs}}\right) \quad (6)$$

This ratio determines the probability at each timestep of using the true previous output rather than the model's prediction. As training progresses and the model becomes more capable of generating accurate predictions, the ratio decreases, gradually reducing the model's reliance on the true output. This transition helps the model to learn to generate sequences independently, improving its robustness and ability to generalize from training to inference scenarios.

D. Hyperparameter Tuning

Hyperparameters and their values 20 epochs using Spatio Temporal Transformer											
J.dim	layers	Src Len	Batch Size	Optim	Hidden	T. Loss	V. Loss	MAE 6	MAE 12	MAE 18	MAE 24
16	1	120	64	adam	128	0.00185	0.00189	2.7055	7.4349	13.7244	20.9653
32	1	120	64	adam	128	0.00185	0.00189	2.7026	7.4326	13.7166	20.9568
48	1	120	64	adam	128	0.00185	0.00189	2.7022	7.4297	13.7132	20.9528
64	1	120	64	adam	128	0.00185	0.00189	2.7049	7.4341	13.7206	20.9629
80	1	120	64	adam	128	0.00185	0.00189	2.7021	7.4296	13.7131	20.9530
96	1	120	64	adam	128	0.00185	0.00189	2.7051	7.4339	13.7210	20.9614
112	1	120	64	adam	128	0.00185	0.00189	2.7025	7.4307	13.7147	20.9557
128	1	120	64	adam	128	0.00185	0.00189	2.7019	7.4275	13.7093	20.9491
64	2	120	64	adam	128	0.00185	0.00189	2.6861	7.4047	13.6753	20.9038
64	3	120	64	adam	128	0.00185	0.00189	2.6995	7.4244	13.7038	20.9436
64	4	120	64	adam	128	0.00185	0.00189	2.6991	7.4296	13.7015	20.9379
64	5	120	64	adam	128	0.00185	0.00189	2.7020	7.4275	13.7107	20.9509
64	1	120	32	adam	128	0.00367	0.00384	2.5957	7.2480	13.4984	20.7097
64	1	120	96	adam	128	0.00123	0.00125	2.7976	7.6433	14.0505	21.4340
64	1	120	64	sgd	128	0.00181	0.00191	2.8564	7.78822	14.24942	21.57944
64	1	120	64	noamopt	128	0.00059	0.00174	2.2324	6.9032	13.3162	20.6464
64	1	90	64	noamopt	128	0.00229	0.00211	3.0059	8.6208	15.5506	23.1432
64	1	60	64	noamopt	128	0.00348	0.00234	3.1276	9.1852	16.1807	23.5763
64	1	30	64	noamopt	128	0.00183	0.00190	2.3958	7.4460	13.9356	21.2564
64	1	120	64	adam	256	0.00185	0.00189	2.7048	7.4330	13.7194	20.9617
64	1	120	64	adam	512	0.00184	0.00188	2.748	7.5365	13.9419	20.3238

Table 1: Table for Joint Dimensions, Number of Layers, Sequence Length, Batch Size, Optimizer, Hidden Dimension, Training Loss, Validation Loss, and MAE for 20 Epochs using Spatio Temporal Transformer

E. Model Performance

Base model, MoE, and ST Transformer performance with default values and 95 epochs						
Model	Training Loss	Validation Loss	MAE 6	MAE 12	MAE 18	MAE 24
RNN	0.00593	0.00437	7.70777	15.92701	24.69410	33.78859
seq2seq	0.00238	0.00164	2.82729	7.51121	13.78625	21.23050
transformer	0.00427	0.00601	7.62462	19.089649	31.30014	43.94872
transformer encoder	0.00702	0.00686	10.04142	20.43901	31.18811	42.193925
ST transformer	0.00058	0.00050	2.78517	7.88353	14.46974	21.84928
MoE	0.00346	0.00188	2.63867	7.90675	14.87096	22.81373

Table 2: Base model, MoE, and ST Transformer performance with default values and 95 epochs

Default hyper-parameter values				
Batch Size	Optimizer	Joint Embedding Size	Hidden Dims	Num layers
64	sgd	56	1024	2

Table 3: Default hyper-parameter values used for base model training

Hyperparameters and their optimal values				
Batch Size	Optimizer	Joint Embedding Size	Hidden Dims	Num layers
32	noamopt	128	512	2

Table 4: Best hyper parameter values for Batch size, Optimizer, Joint Dimensional Embedding Size, Hidden Dimensions, and Number of layers

ST Transformer vs MoE with Double/Float64 precision and 30 epochs						
Model	Training Loss	Validation Loss	MAE 6	MAE 12	MAE 18	MAE 24
ST transformer	0.00408	0.00344	2.21965	6.80977	13.04387	20.19733
MoE	0.00623	0.00359	1.90865	6.48504	12.98277	20.49475

Table 5: ST Transformer vs MoE performance with using Double/Float64 precision on data types and and 30 epochs

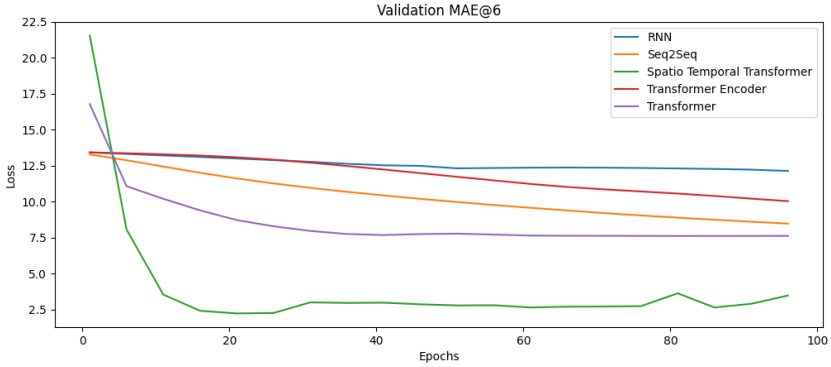


Figure 8: **V**alidation MAE@6

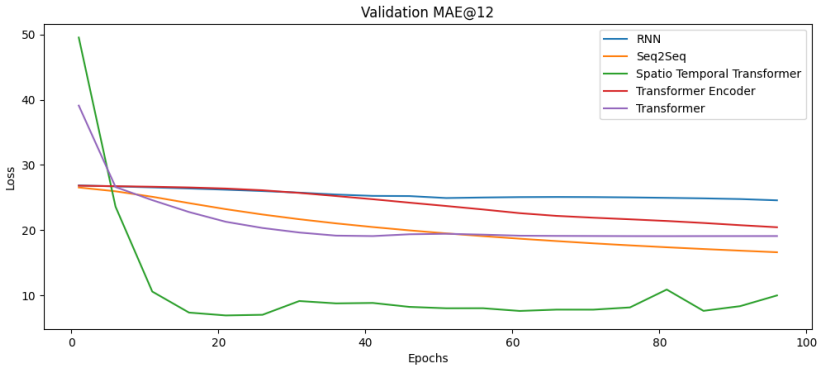


Figure 9: **V**alidation MAE@12

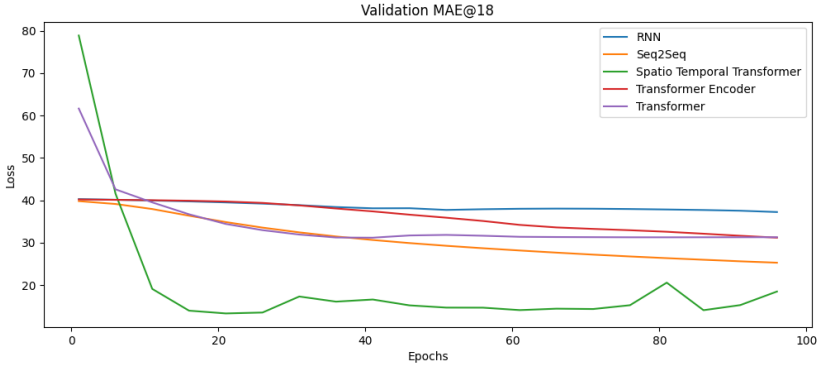


Figure 10: **V**alidation MAE@18

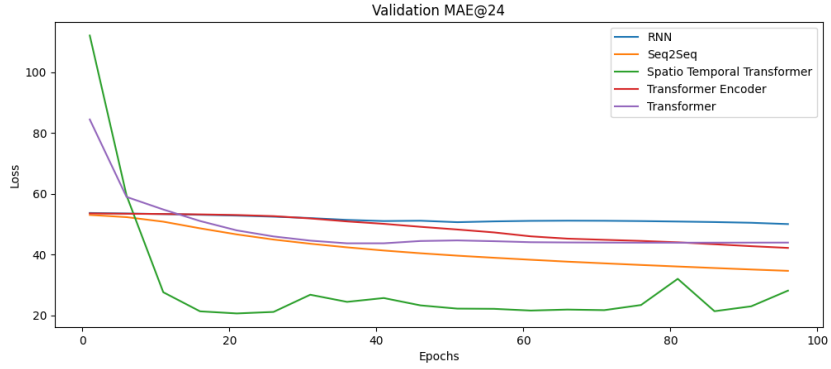
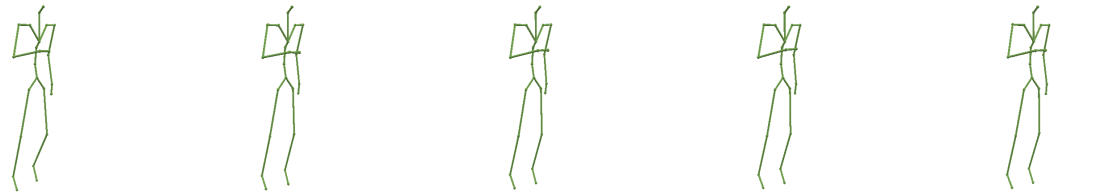


Figure 11: **V**alidation MAE@24

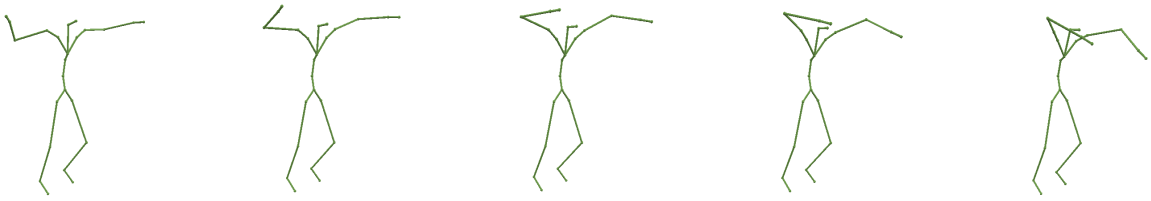


(a) Frame 124 (b) Frame 128 (c) Frame 132 (d) Frame 136 (e) Frame 140

Figure 12: Ground Truth Motion 1

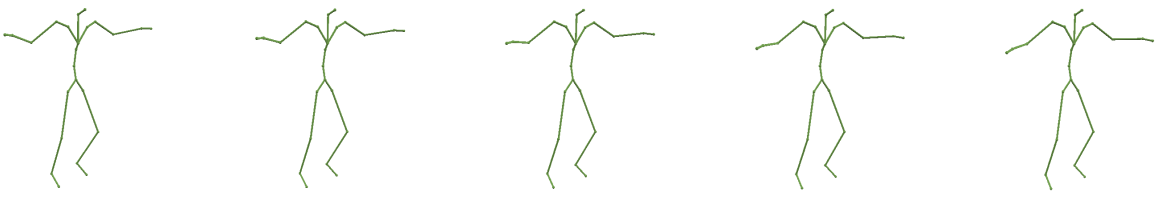
(a) Frame 124 (b) Frame 128 (c) Frame 132 (d) Frame 136 (e) Frame 140

Figure 13: ST Transformer Predicted Motion



(a) Frame 124 (b) Frame 128 (c) Frame 132 (d) Frame 136 (e) Frame 140

Figure 14: Ground Truth Motion 2



(a) Frame 124 (b) Frame 128 (c) Frame 132 (d) Frame 136 (e) Frame 140

Figure 15: MoE Predicted Motion

RNN, Seq2Seq, Transformer Encoder, Transformer, ST Transformer, and MoE performance after 100 epochs				
Architecture	MAE@6	MAE@12	MAE@18	MAE@24
RNN	11.5310	23.2960	35.2423	47.3383
Seq2Seq	7.7943	15.1377	23.0770	31.8828
Transformer Encoder	9.2491	18.8478	28.7915	39.0187
Transformer	6.8229	16.9269	27.7262	38.9774
STTransformer	2.0733	6.1653	11.7492	18.2334
MoE	1.9475	5.9395	11.3594	17.6420

Table 6: Test MAE for best models

F. Temporal Attention Visualizations

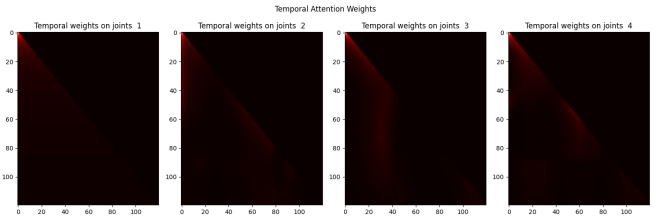


Figure 16: **Temporal Attention Map for joints 1, 2, 3, 4** map of temporal attention weights by timesteps in joints 1, 2, 3, 4

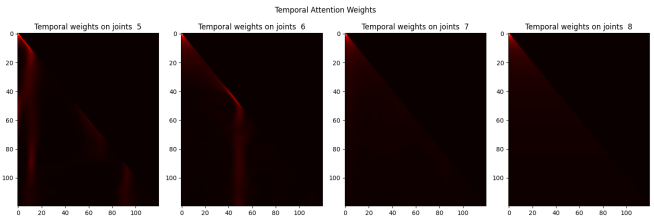


Figure 17: **Temporal Attention Map for joints 5, 6, 7, 8** map of temporal attention weights by timesteps in joints 5, 6, 7, 8

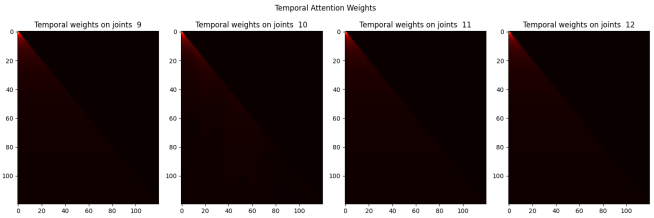


Figure 18: **Temporal Attention Map for joints 9, 10, 11, 12** map of temporal attention weights by timesteps in joints 9, 10, 11, 12

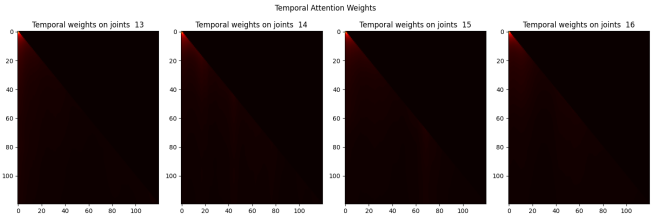


Figure 19: **Temporal Attention Map for joints 13, 14, 15, 16** map of temporal attention weights by timesteps in joints 13, 14, 15, 16

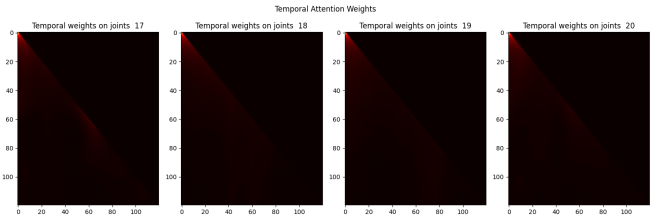


Figure 20: **Temporal Attention Map for joints 17, 18, 19, 20** map of temporal attention weights by timesteps in joints 17, 18, 19, 20

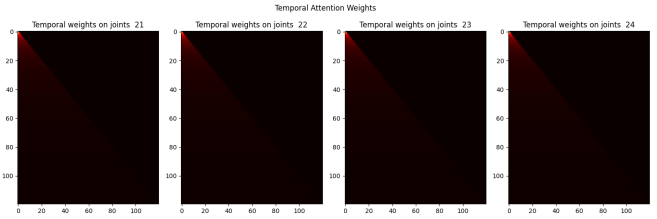
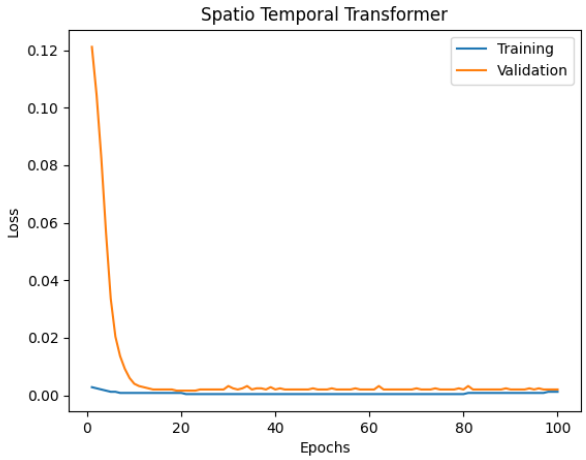


Figure 21: **Temporal Attention Map for joints 21, 22, 23, 24** map of temporal attention weights by timesteps in joints 21, 22, 23, 24

G. Loss Function



H. Inference Experiment Results

Table 7: Testing Run Times for Different Model Configurations

Model	Param	Total Params	Testing Time (s)
Vanilla	64	42.6K	1.49
Vanilla	128	50.8K	1.54
Vanilla	256	67.3K	1.65
Vanilla	512	100.3K	1.78
Vanilla	1024	166.4K	2.34
Vanilla	2048	298.5K	3.38
MoE	2	37.8M	1.57
MoE	4	75.6M	1.62
MoE	6	113.3M	1.59
MoE	8	151.1M	1.74
MoE	16	302.2M	1.86
MoE	32	604.3M	2.35

Table 8: Param refers to hidden dimension of feedforward layer for STtransformer and number of experts for MoE.

I. Work Division

Name	Contributed Aspects	Details
Edmund Shieh	Research and implementation of SOTA and novel architectures	Conducted literature review of SOTA papers to understand the model architecture of the SOTA and implement code in PyTorch. Further literature review into model architectures for speeding up inference times on large models and integrating these solutions into a new novel architecture in PyTorch.
Tej Lalvani	Analysis, parallelization, and visualization	Trained the SOTA with varying src lengths and datatypes, as well as the baseline RNN, Seq2Seq, and Transformer models. Incorporated parallelization into training code to speed up training when using two GPUs. Wrote code to create plots from log files and extract frames for motion visualizations.
Joshua Lee Franco	Experimentation and Analysis	Trained SOTA with varying number of layers and MoE with varying experts during hyperparameter tuning. Added logging and addition utilities to keep track of experiments and avoid loss of data during training. Added typing in training and test modules to help with code implementations.

Table 9: Contributions of team members.