Report Title: Comparative Analysis of Positional, Euler, and Quaternion Representations for Auto-Conditioned LSTM-based Character Motion Synthesis and Investigation of Seed Length Impact

Authors: Chrysis Andreou, Mohamad Fatfat

Abstract: This project investigates character motion prediction using an Auto-Conditioned Long Short-Term Memory (acLSTM) model, focusing on a comparative analysis of three distinct motion representations: Positional, Euler Angles, and Quaternions. The primary objective was to implement and evaluate these representations for salsa dance motion synthesis and to explore a research question concerning the impact of initial seed sequence length on prediction accuracy. The methodology involved preprocessing BVH motion capture data into each representation, adapting and training acLSTM models with appropriate loss functions, and conducting qualitative and quantitative evaluations. Key findings from Part A highlight that the Positional representation served as a stable baseline. Euler Angles offered dynamic motion but were prone to visual instabilities (flips), even with low quantitative error. Quaternions initially suffered from motion freezing during synthesis, a challenge that was successfully addressed by increasing the `Condition_num` parameter during training (notably to 45), which, combined with a 300-frame seed, enabled continuous long-term motion generation. The Part B research investigation revealed that the optimal seed length for minimizing short-term quantitative prediction error varies across representations and does not always align with the seed length required for optimal long-term qualitative performance. Main contributions include the successful adaptation and analysis of Euler and Quaternion representations for the acLSTM framework and the systematic resolution of the quaternion motion stillness problem, underscoring the importance of training regime design for autoregressive models.

1. Introduction

1.1 Objective

The primary objective of this project is to experiment with character motion prediction using different character representations and to extend these experiments by formulating and answering a specific research question. This involves implementing, training, and evaluating models for various representations to gain insights into their influence on motion prediction performance.

1.2 Background

The baseline method for this project is the Auto-Conditioned Recurrent Neural Network (acRNN), specifically an Auto-Conditioned LSTM (acLSTM). This approach aims to synthesize highly complex human motions in real-time. Traditional autoregressive techniques for motion synthesis often suffer from error accumulation, causing the generated motion to freeze or diverge after a short period, particularly with complex motions like dances or martial arts. The acRNN architecture addresses this challenge by explicitly accommodating for autoregressive noise accumulation during its training regime, enabling the generation of long, continuous sequences of complex human motion.

1.3 Representations Studied

This project investigates three distinct character motion representations:

- **Positional Representation:** Describes joint locations in 3D space, typically relative to a root joint (e.g., the hip).
- **Euler Angle Representation:** Defines joint orientation using three angles of rotation (pitch, yaw, roll) around the X, Y, and Z axes. This method is intuitive but can suffer from a problem known as **gimbal lock**, where at

certain configurations, two rotation axes align, leading to a loss of one degree of rotational freedom and potential animation artifacts like flipping.

- **Quaternion Representation:** Utilizes four-dimensional numbers (w, x, y, z) to represent rotations. This offers advantages like numerical stability and, crucially, the **avoidance of gimbal lock**, making them more robust for complex 3D animations.

1.4 Project Scope

The project encompasses several key activities:

1. **Implementation/Adaptation:** Adapting existing methodologies or implementing new components for handling the three specified motion representations within the acLSTM framework. This includes developing data preprocessing pipelines to convert BVH files into these representations and corresponding decoding functions to convert them back for visualization.
2. **Model Training:** Training acLSTM models for each representation using the "salsa" dataset.
3. **Evaluation:** Assessing the performance of the trained models both qualitatively (visual inspection of generated motion) and quantitatively (using appropriate error metrics).
4. **Research Investigation (Part B):** Formulating and investigating a research question related to character motion prediction, extending the core experiments.

1.5 Structure of the Report

This report is organized as follows:

- **Part A: Core Experiments - Methodology and Setup:** Details the experimental procedure, data preprocessing for each representation, model training specifics, and evaluation setup.
- **Part A: Results and Discussion:** Presents and discusses the training performance, qualitative and quantitative synthesis results for the core experiments.
- **Part B: Research Question - Impact of Seed Sequence Length:** Outlines the research question, methodology, presents the results, and discusses the findings.
- **Further Discussion & Challenges:** Discusses any significant challenges encountered or interesting observations made.
- **Conclusion:** Summarizes the main achievements, key findings, and contributions of the project.
- **References:** Lists the cited works.

2. Part A: Core Experiments - Methodology and Setup

2.1. Experiment Overview

The core experiments in Part A of this project followed a structured 3-step procedure for each motion representation studied:

1. **Data Preprocessing:** This initial step involved converting raw motion data from BVH (Biovision Hierarchy) files into the specific target representation (Positional, Euler Angles, or Quaternions). The processed data was then stored in a format suitable for training neural networks, typically NumPy arrays. This also included developing functions to decode the representations back into BVH format for visualization and evaluation.
2. **Model Training:** The processed data for each representation was used to train a sequence prediction model. The base model architecture employed was an LSTM-based sequence model, specifically leveraging the Auto-Conditioned LSTM (acLSTM) concept. The acLSTM approach is designed to mitigate error accumulation in long-term motion synthesis by periodically feeding the network's own output back as input during the training process.

3. **Evaluation:** Once trained, the models were evaluated both qualitatively and quantitatively. Qualitative evaluation involved visually inspecting the synthesized motion for realism, coherence, and style preservation. Quantitative evaluation involved measuring the performance error of the synthesized motion against ground truth data using appropriate metrics for each representation.

The overall workflow is depicted in Figure 1 of the project instructions, emphasizing the cyclical nature of preprocessing, training, and evaluation, with the ultimate goal of generating new motion sequences.

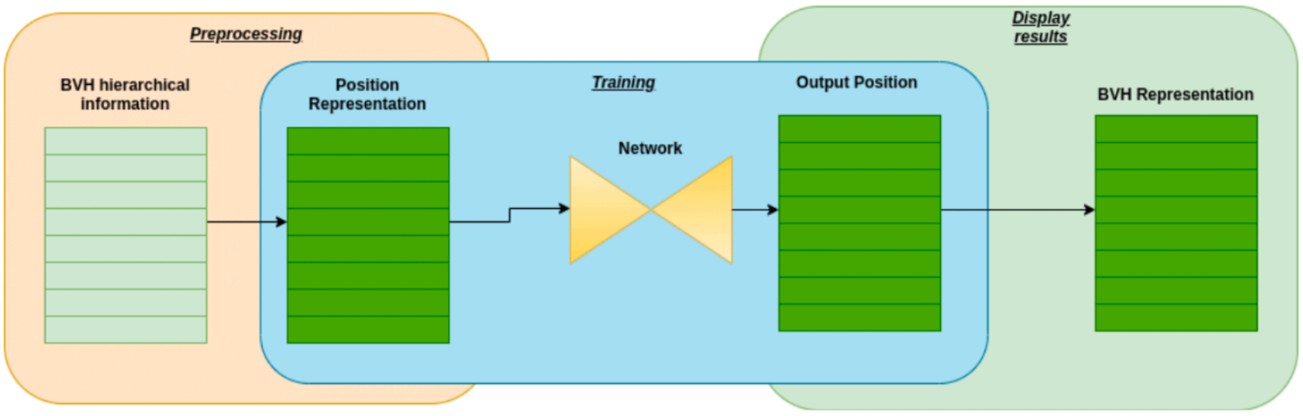


Figure 1: This figure displays the 3 separate steps of the procedure. First, the preprocessing step converts BVH to the desired motion representation, the training procedure learns motion sequence data and the last step is the evaluation of the model. To display the generated motion we need to convert motion representation to BVH.

2.2. Data Preprocessing

Data preprocessing is a critical step in preparing character motion data for machine learning models. It involves transforming raw motion capture data (typically in BVH format) into a numerical representation suitable for the neural network, and also defining a reverse transformation to convert network outputs back to BVH for visualization and analysis.

2.2.1. General Preprocessing Steps

The general workflow for processing BVH data, as outlined in Algorithm 1 of the project instructions, involves the following key stages:

1. **Hierarchy Parsing:** Understanding the skeletal structure (bones, hierarchy, joint offsets) from the BVH file. This is typically done once using scripts like `read_bvh_hierarchy.py` to obtain skeleton information (e.g., `skeleton`, `non_end_bones`).
2. **Motion Data Extraction:** Reading the `MOTION` section of each BVH file, which contains frame-by-frame joint rotation and root translation data. The `read_bvh.py` script provides core functionalities for parsing these motion lines into numerical arrays.
3. **Representation Conversion:** For each frame, converting the raw motion data (hip translation and joint rotations) into the desired target representation (Positional, Euler, or Quaternion).
4. **Data Storage:** Storing the processed frame-wise data, usually as NumPy arrays (`.npy` files), for efficient loading during model training.

A crucial aspect of this pipeline is the corresponding decoding function, which reverses the representation conversion to allow synthesized motion to be written back to BVH files for qualitative evaluation.

2.2.2. Positional Representation

The positional representation describes the character's pose by the 3D Cartesian coordinates (x, y, z) of each joint. As detailed in project instructions (Section 2.1, "Positional Representation"), the process involves:

1. **Forward Kinematics:** Calculating the global 3D position of each joint based on the skeletal hierarchy and the Euler angle rotations provided in the BVH file.
2. **Hip-Relative Coordinates:** Translating the global positions of all joints to be relative to the hip joint's global position for each frame. The hip joint itself retains its global position, representing the character's overall displacement.

The script `generate_training_pos_data.py` was largely provided and implements this conversion. It takes BVH files as input, performs the transformation to hip-relative joint positions, and saves the output as NumPy arrays. It also includes a corresponding function to decode these positional arrays back into BVH format. The `read_bvh.py` script (specifically `get_one_frame_training_format_data` and `get_training_format_data` functions) handles the core logic of computing these joint positions and making them hip-relative, including a scaling factor (0.01) applied to the coordinates. This scaling factor is used to convert the typically large joint position values (often in centimeters) into a smaller, more numerically stable range (e.g., meters), which improves gradient behavior and optimizer efficiency during neural network training. Additionally, scaling ensures that positional features are on a scale comparable to other input features, such as joint rotations in radians, thereby preventing any single feature type from disproportionately influencing the model. Finally, adopting this convention maintains compatibility with baseline implementations and facilitates fair comparison across different model variants.

2.2.3. Euler Angle Representation (Our Contribution)

For the Euler angle representation, the character's pose is defined by the hip's 3D position and the Euler angle rotations (typically ZXY order for joints, ZYX for hip) for each joint. The project instructions (Section 2.1, "Euler Angle Representation") specify that only the hip location is maintained as a position, while other joints are represented by their rotations. Euler angles are intuitive but are susceptible to **gimbal lock**, a phenomenon where the alignment of two rotational axes at specific orientations (e.g., a 90-degree pitch) causes a loss of one degree of rotational freedom. This can lead to abrupt, uncontrollable flips or an inability to rotate around a specific world axis, which can manifest as visual instabilities in synthesized motion.

Our implementation choices in `generate_training_euler_data.py` for preprocessing Euler angle data are as follows:

- **Hip Position Normalization:** The raw hip position from the BVH file undergoes normalization. Specifically:
  - The X and Z components are made relative to their values in the first frame of the sequence (i.e., `current_hip_x - first_frame_hip_x`).
  - All three components (X, Y, Z) are then scaled by a `HIP_POS_SCALE_FACTOR` (0.01).
  - The Y component, while scaled, remains an absolute global height (not relative to the first frame's Y).
  - *Reasoning:* Making X and Z hip positions relative to the start of the sequence helps the model learn translational patterns independent of the absolute starting location in the world. Keeping Y absolute (though scaled) preserves global height information which can be important for certain motions. Scaling all components helps normalize the input range for the neural network, potentially improving training stability and convergence speed.
- **Angle Conversion:** Euler angles for all joints (including the hip) are read from the BVH files (which are in degrees) and converted to radians, as neural networks typically work better with radian inputs for trigonometric calculations in loss functions or internal operations.

The `generate_training_euler_data.py` script implements these steps, reading BVH files, applying the hip normalization and angle conversion, and saving the results as `.npy` files. The corresponding decode function in this script reverses the process: it converts Euler angles from radians back to degrees and un-scales the hip position components (dividing by `HIP_POS_SCALE_FACTOR`). Note that the decode function for verification in this script does not add back the initial X/Z hip offsets, as that absolute positioning is typically handled during the synthesis phase when a specific starting pose is given. This design choice enables better side-by-side visualization of synthesized and ground truth motions in BVH viewers.

2.2.4. Quaternion Representation (Our Contribution)

The Quaternion representation aims to describe joint rotations using four-dimensional numbers (w, x, y, z), which helps avoid issues like gimbal lock and can offer more stable interpolation than Euler angles. Unlike Euler angles, which apply rotations sequentially around predefined axes, quaternions represent a rotation as a single operation around an arbitrary axis in 3D space. This fundamental difference means quaternions do not suffer from the loss of rotational freedom associated with gimbal lock, leading to smoother and more predictable rotational behavior. The input features for the network consist of the hip's 3D position and quaternion rotations for all joints.

Our implementation choices in `generate_training_quad_data.py` for preprocessing quaternion data include:

- **Hip Position Normalization:** Similar to the Euler angle representation, the hip position is normalized:
  - X and Z components are made relative to their values in the first frame and then scaled by `HIP_POS_SCALE_FACTOR` (0.01).
  - The Y component is scaled by `HIP_POS_SCALE_FACTOR` but remains absolute.
- **Euler to Quaternion Conversion:** The Euler angles for each joint (hip and others) are read from the BVH file (in degrees). These are first converted to radians. Then, for each joint, the Euler angles (in radians, order ZYX for hip, ZXY for other joints as per typical BVH structure) are converted to rotation matrices, and subsequently, these rotation matrices are converted into quaternions (in wxyz format). This conversion leverages functions from `rotation_conversions.py` (specifically, `euler_angles_to_matrix` and `matrix_to_quaternion`).
- **Metadata Saving:** A `metadata_quad.json` file is saved alongside the processed `.npy` files. This file stores important information such as the total number of channels in the processed data, the `hip_pos_scale_factor` used, the original frame time from the BVH files, the source BVH folder path, and the quaternion order (wxyz). This metadata is useful for ensuring consistency during the decoding/synthesis phase, allowing the synthesis script to correctly interpret the data and reconstruct BVH files with the correct scaling and frame rate.

The `generate_training_quad_data.py` script orchestrates this process. It reads BVH files, applies hip normalization, converts rotations to quaternions using the `rotation_conversions.py` module, and saves the `.npy` data and `metadata_quad.json` . The corresponding decode function in this script converts quaternions back to Euler angles (via rotation matrices, using `quaternion_to_matrix` and `matrix_to_euler_angles` from `rotation_conversions.py`) and un-scales the hip positions for BVH visualization. It also uses the `frame_time_original` from metadata to write the reconstructed BVH with the correct timing.

2.3. Model Training

Once the data was preprocessed into the respective representations, the next phase involved training the Auto-Conditioned LSTM (acLSTM) models.

2.3.1. General Training Procedure

The general training procedure followed Algorithm 2 from the project instructions (Section 2.3). This involved iterating through the training data, feeding sequences to the acLSTM model, calculating the loss between the model's predictions and the ground truth, and updating the model weights via backpropagation using an optimizer (Adam).

A core aspect of the training was the Auto-Conditioned LSTM (acLSTM) approach, as detailed by Zhou et al. (2017, Section 3, Figures 1 & 10). Traditional training of autoregressive models often feeds ground truth data at each time step. However, during inference, the model relies on its own previous outputs. This mismatch, known as **exposure bias**, can lead to an accumulation of errors where small inaccuracies compound, potentially causing the generated motion to freeze or diverge significantly.

The acLSTM architecture addresses this by incorporating an **auto-conditioning** training strategy. During training, instead of always feeding ground truth frames, the model is periodically fed its own output for a certain number of steps (`Condition_num`) before ground truth is reintroduced (`Groundtruth_num`). This process forces the model to learn to recover from its own errors and makes it more accustomed to its own generated data distribution. This technique helps the model learn to generate longer, more stable sequences, which is crucial for robust autoregressive synthesis.

As per the teaching assistant instructions, the intention was to train all models for 50,000 iterations for a fair comparison. The actual pre-trained model checkpoints used for the synthesis results (Part A) and the Part B analysis were as follows:

- **Positional Model:** `0049000.weight` (trained for 49,000 iterations).
- **Euler Model:** `0051000.weight` (trained for 51,000 iterations).
- **Quaternion Model (Condition_num=5):** `0028000.weight` (trained for 28,000 iterations).
- **Quaternion Model (Condition_num=30):** `0031000.weight` (trained for 31,000 iterations).
- **Quaternion Model (Condition_num=45):** `0049000.weight` (trained for 49,000 iterations). In some cases, training was interrupted early because further iterations did not improve the model's tendency to generate static (i.e., motionless or frozen) sequences, where the synthesized figure would remain immobile rather than producing dynamic motion.

2.3.2. Loss Functions

The choice of loss function is critical and depends on the data representation being learned. The project instructions (Section 2.2) specified the following loss functions:

- **Positional Representation:** Mean Squared Error (MSE) was used to compare the predicted 3D joint positions with the ground truth positions.
  - Formula: `MSE = (1/N) * Σ(xi - yi)²`
- **Euler Angle Representation:** A combined loss was used:
  - MSE for the 3D hip position (comparing predicted vs. ground truth hip coordinates).
  - Angle Distance (AD) for the joint rotations (comparing predicted vs. ground truth Euler angles). Angle Distance is typically calculated as `Σ(1 - cos(predicted_angle - true_angle))` , encouraging predicted angles to be close to true angles, handling periodicity correctly.
- **Quaternion Representation:** A combined loss was used:
  - MSE for the 3D hip position.
  - Quaternion Loss (QL) for the joint rotations. This is often calculated as `2 * arccos(|q_pred · q_true|)` , where `·` is the dot product. This measures the angular difference between the two quaternions.

2.3.3. Positional Model Training

The training script `pytorch_train_pos_aclstm.py` was largely provided as part of the baseline. This script implements the acLSTM architecture and training loop for the positional representation.

- **Input/Output Channels:** For the positional data, each frame consists of 3D coordinates for each joint. Given 57 joints, the input and output channel size for the LSTM was `57 joints * 3 coordinates/joint = 171` channels.

- The script handles the specific preprocessing for positional data during training, where the hip X and Z positions are converted to frame-to-frame differences, while the Y position and all other joint positions (relative to the hip) are used directly (after initial scaling and hip-relativization during data generation).

2.3.4. Euler Model Training (Our Contribution)

For the Euler angle representation, we adapted the baseline training script to create `pytorch_train_euler_aclstm.py`. Our key modifications included:

- **Adjusted Input/Output Channels:** The input/output channel dimensions were modified to suit the Euler representation. As implemented for this project, the raw BVH channel structure was used directly, following normalization for the hip position and conversion of angles to radians. For the salsa dataset, this approach resulted in 132 channels. This specific channel count is derived from the 44-joint skeleton structure characteristic of the dataset's BVH files: 3 channels are allocated for the hip's 3D position, and the remaining 43 rotating joints are each described by 3 Euler angles (amounting to 43 joints × 3 angles/joint = 129 channels for rotations). Consequently, the total channel count is 3 (hip position) + 129 (joint rotations) = 132 channels. The `pytorch_train_euler_aclstm.py` script dynamically determines this channel count by inspecting the loaded `.npy` data.
- **Implementation of Combined Loss Function:** The `calculate_loss` method within the `acLSTM` class was modified to compute the combined loss: MSE for the hip position (first 3 channels of the frame data) and Angle Distance for all subsequent Euler angle channels. The angles were already in radians from preprocessing, suitable for `torch.cos`.
- **Adaptation of BVH Saving Function:** The logic for saving intermediate BVH files during training (`train_one_iteration`) was adapted to correctly decode the Euler angle representation. This involved taking the network's output (which includes scaled hip position differences and absolute radian rotations), reconstructing absolute scaled hip positions by accumulating the differences, then un-scaling the hip positions and converting all radian rotations back to degrees before writing to BVH using a reference hierarchy.
- *Reasoning for modifications:* These changes were essential to ensure that the loss calculation was appropriate for the distinct data types (Cartesian positions for the hip, angular values for rotations) and that the intermediate training outputs could be correctly visualized as BVH animations to monitor training progress.

2.3.5. Quaternion Model Training (Our Contribution & Key Experiment)

Training the Quaternion model involved creating `pytorch_train_quad_aclstm.py`. Similar to the Euler model, this required several adaptations:

- **Adjusted Input/Output Channels:** The input/output channel dimensions were set based on the quaternion representation: 3 channels for the hip position, and 4 quaternion components (wxyz) for each rotating joint. With 43 rotating joints, this results in 3 (hip) + (43 * 4) (rotations) = 175 channels. The script `pytorch_train_quad_aclstm.py` loads this channel count directly from the `metadata_quad.json` file (where `num_channels` is specified as 175).
- **Implementation of Combined Loss Function:** The `calculate_loss` method was updated to compute MSE for the hip position and Quaternion Loss for the rotations. The quaternion loss involved normalizing the predicted and ground truth quaternions, calculating their dot product, and then applying the `2 * arccos(|dot_product|)` formula.
- **Adaptation of BVH Saving Function:** The BVH saving function (`convert_lstm_output_to_bvh_frames`, called by `train_one_iteration`) was significantly modified to bridge the gap between the LSTM model's internal output and the requirements of the BVH animation format. Specifically, it takes the LSTM output—comprising scaled hip position differences (for X and Z), a scaled absolute Y, and absolute quaternions for each joint—and reconstructs the absolute, scaled hip positions by accumulating the differences over time.

These positions are then un-scaled to their original magnitudes using the scaling factor from the metadata. For joint rotations, each quaternion is converted to a rotation matrix and then to Euler angles (using `rotation_conversions.py`), following the conventions required by BVH (e.g., "ZYX" for the hip). This ensures the synthesized motion is spatially accurate and visually correct. Additionally, the function uses the `ORIGINAL_FRAME_TIME` from the metadata to set the correct playback speed in the output BVH file. Intuitively, these adaptations ensure that the model's learning-optimized outputs are faithfully and reliably translated into a standardized, human-interpretable animation format for visualization and evaluation.

- **Focus on the `Condition_num` Experiment:** A key challenge encountered with the quaternion representation was motion stillness. It was observed that initial trainings with default auto-conditioning parameters (e.g., `Condition_num=5`, `Groundtruth_num=5` in `pytorch_train_quad_aclstm.py`) resulted in synthesized motion that would freeze after about 20 frames, despite appearing dynamic during the BVH generation within the training loop itself. The hip displacement values were observed to decay to near-zero during autoregressive synthesis. The hypothesis was that this "hand-holding" during training, with short autoregressive periods, did not adequately prepare the model for extended self-guided generation required in the synthesis phase. The solution involved a systematic adjustment of the auto-conditioning parameters in `pytorch_train_quad_aclstm.py`. Specifically, `Condition_num` was increased significantly (first to 30, then to 45) while `Groundtruth_num` was kept constant at 5. This change forced the model to practice generating longer sequences based on its own outputs during training, making the training regime more aligned with the demands of long-term synthesis. By reducing the discrepancy between training and inference conditions, this adjustment helped mitigate the exposure bias discussed earlier, making the model more robust to its own accumulated errors during prolonged generation. This modification proved crucial for overcoming the stillness problem and was a key problem-solving contribution of this project, enabling the generation of continuous, dynamic motion with the quaternion representation when combined with an appropriate seed sequence length during synthesis.

2.4. Evaluation Setup

The evaluation of the trained models aimed to assess their performance both qualitatively and quantitatively, as outlined in project instructions (Section 2.4).

- **Qualitative Evaluation:** This involved visually inspecting the synthesized motion for naturalness, coherence, and adherence to the learned style (salsa dance). This was primarily done using an online BVH viewer (specifically, the viewer at `theorangeduck.com/media/uploads/BVHView/bvhview.html` was noted for its utility in side-by-side comparisons). For settings, the best is to choose 'nearest' (between nearest, cubic, linear) and a lower speed of 0.1.
- **Quantitative Evaluation:** This focused on measuring the prediction error of the synthesized motion against ground truth motion data from the test set. Specific metrics appropriate to each representation (e.g., MSE for positions, combined MSE and Angle Distance for Euler, combined MSE and Quaternion Loss for Quaternions) were used. Typically, this involved comparing a short sequence of generated frames (e.g., the first 20 frames) immediately following a seed sequence against the corresponding ground truth continuation.

The synthesis process, which forms the basis of evaluation, was carried out using dedicated Python scripts for each representation:

- `synthesize_pos_motion.py` for the Positional representation.
- `synthesize_euler_motion.py` for the Euler Angle representation.
- `synthesize_quad_motion.py` for the Quaternion representation.

These scripts generally perform the following steps:

1. Load a pre-trained model checkpoint.
2. Load a seed sequence of motion frames from the preprocessed dataset (e.g., `train_data_pos/salsa/*.npy`). The length of this seed sequence (`initial_seq_len`) was a parameter

that could be varied (e.g., 20-100 frames for initial qualitative checks, and systematically varied as 100, 200, 300 for Part B analysis).

3. Feed the seed sequence to the model to initialize its internal state.
4. Autoregressively generate a longer sequence of motion frames (e.g., 400 frames for qualitative viewing).
5. Convert the generated sequence (and corresponding ground truth, if available) back to BVH format for visual inspection.
6. For quantitative evaluation, calculate the error metrics by comparing a portion of the generated sequence against the corresponding ground truth continuation from the dataset.

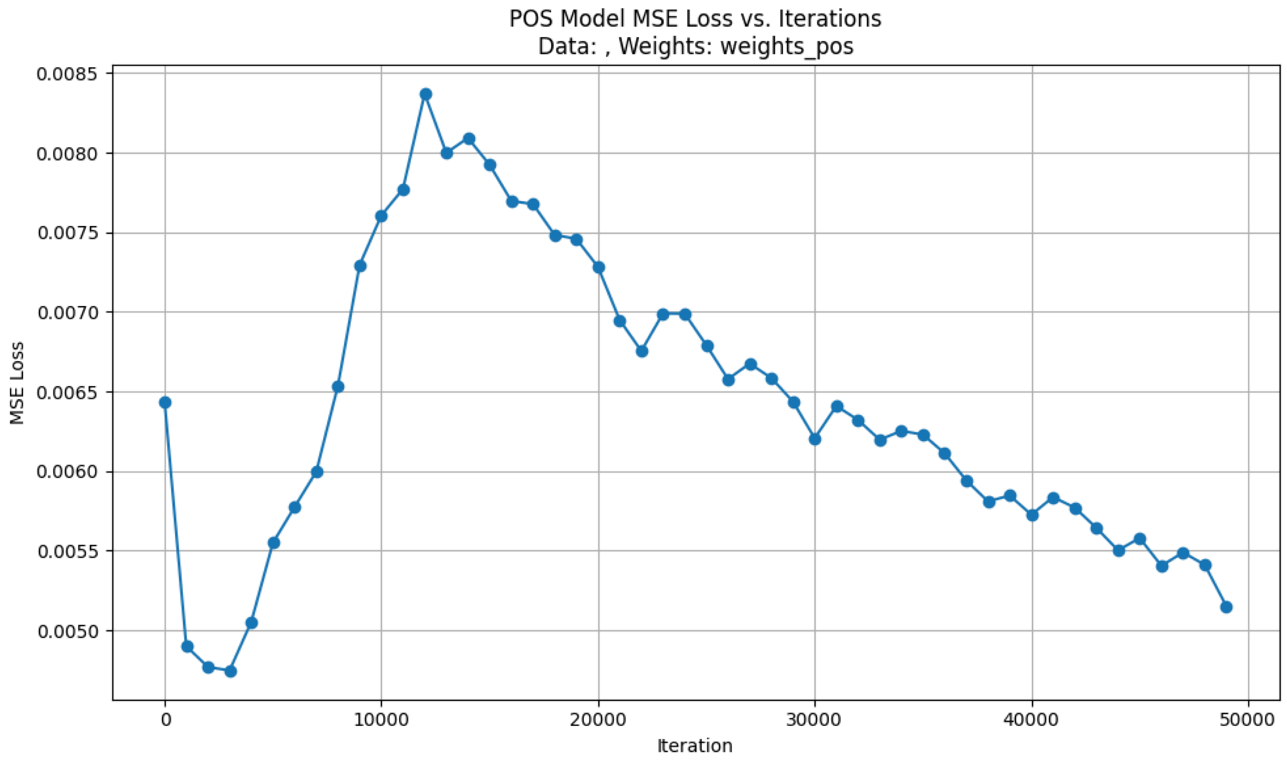
3. Part A: Results and Discussion

This section presents the results obtained from the core experiments (Part A), covering training performance, qualitative synthesis evaluation, and quantitative synthesis evaluation, followed by a discussion of these findings.

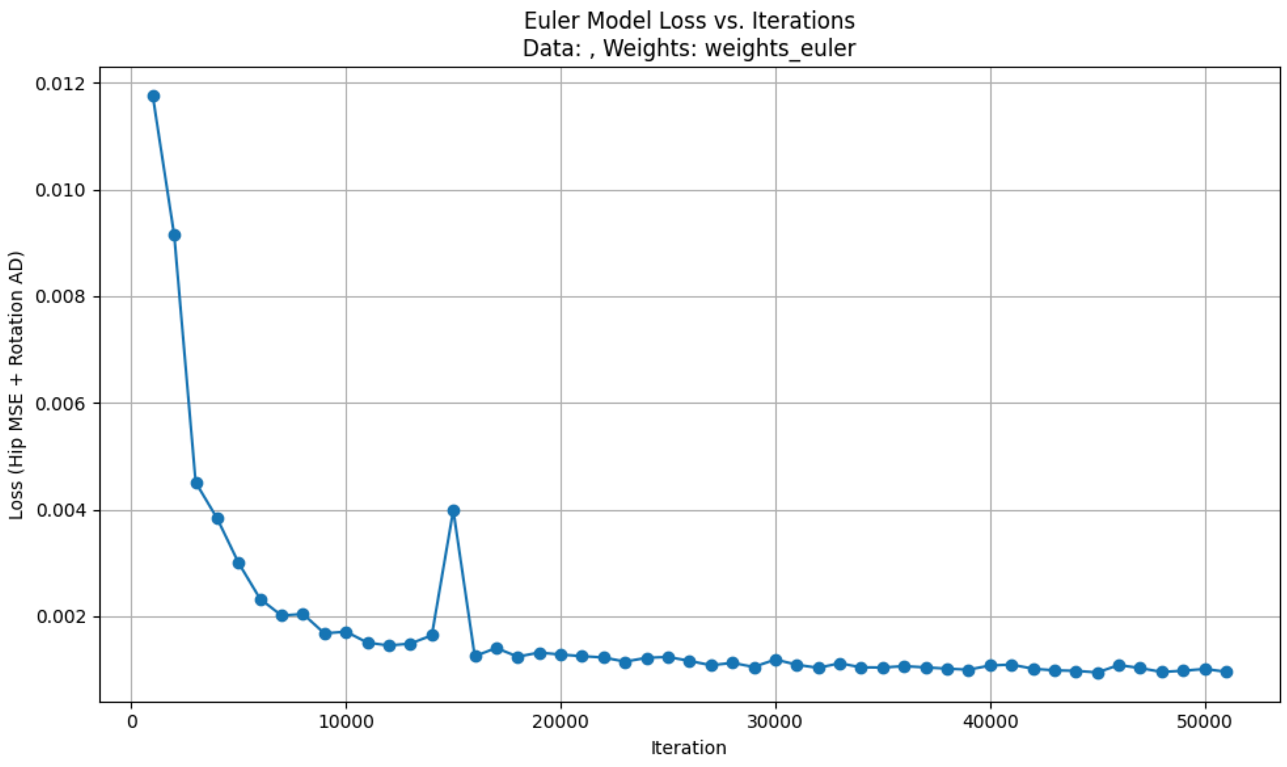
3.1. Training Performance - Loss Trajectories

To assess how well each model learned the motion data, their respective loss trajectories were recorded during training. The scripts `plot_loss.py` (for Positional and Quaternion representations) and `plot_euler_loss.py` (for Euler representation) were used to generate visualizations of these trajectories by loading model weights at different training iterations and calculating the loss on a consistent validation data segment. This allows for a comparable view of learning progress across different models and iterations.

- **Positional Representation:** Figure 2 shows the Mean Squared Error (MSE) loss of the positional model as a function of training iterations. The plot demonstrates the learning trajectory of the Auto-Conditioned LSTM (acLSTM) model trained on the positional representation of the salsa dance dataset. Initially, the MSE loss decreases rapidly, indicating that the model is quickly learning to predict joint positions from the training data. After this initial drop, the loss exhibits some fluctuations but generally trends downward as training progresses, reflecting continued improvement and fine-tuning of the model. The overall decrease in MSE loss over 50,000 iterations suggests that the model is effectively minimizing prediction error and learning a stable mapping from input sequences to future joint positions. This stable and consistent reduction in loss highlights the suitability of the positional representation as a robust baseline for character motion prediction.



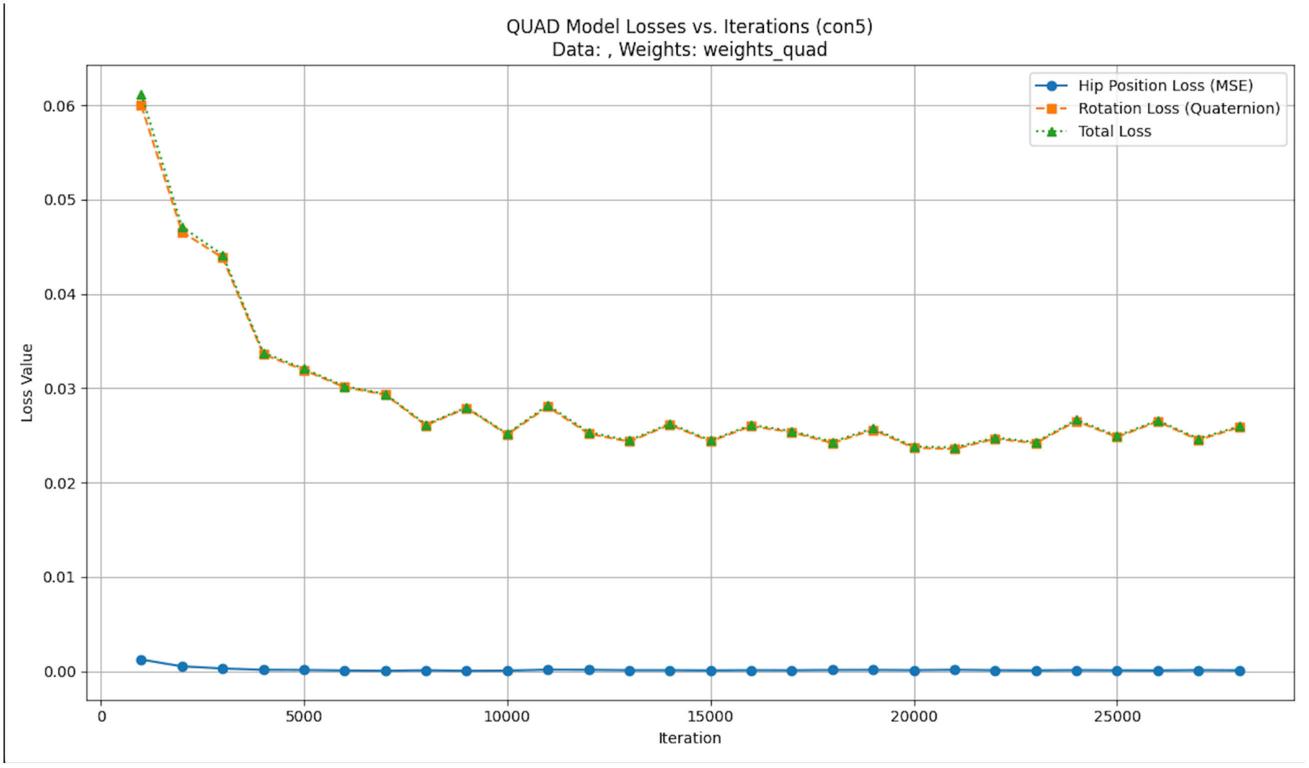
- **Euler Angle Representation:** Figure 3 shows the combined loss (Hip MSE + Rotation Angle Distance) for the Euler angle representation as a function of training iterations. The loss decreases sharply in the early stages, dropping from 0.012 at iteration 1000 to below 0.002 by iteration 10,000, indicating rapid initial learning. After this, the loss continues to decline more gradually, converging to a stable minimum around 0.001 by iteration 50,000. The brief spike in loss near iteration 15,000 likely reflects a temporary instability during training, but the model quickly recovers and resumes its downward trend. Overall, the curve demonstrates effective learning and convergence of the acLSTM model for the Euler angle representation, despite occasional fluctuations.



- **Quaternion Representation (Con5, Con30, Con45):** Figures 4, 5, and 6 illustrate the training loss trajectories for the Quaternion representation models, corresponding to `Condition_num` values of 5, 30, and 45, respectively. Each plot displays the Hip Position Loss (MSE), Rotation Loss (Quaternion), and the Total Loss against the number of training iterations. These figures allow for a comparative analysis of how different auto-conditioning lengths affect the learning process for quaternion-based motion synthesis.

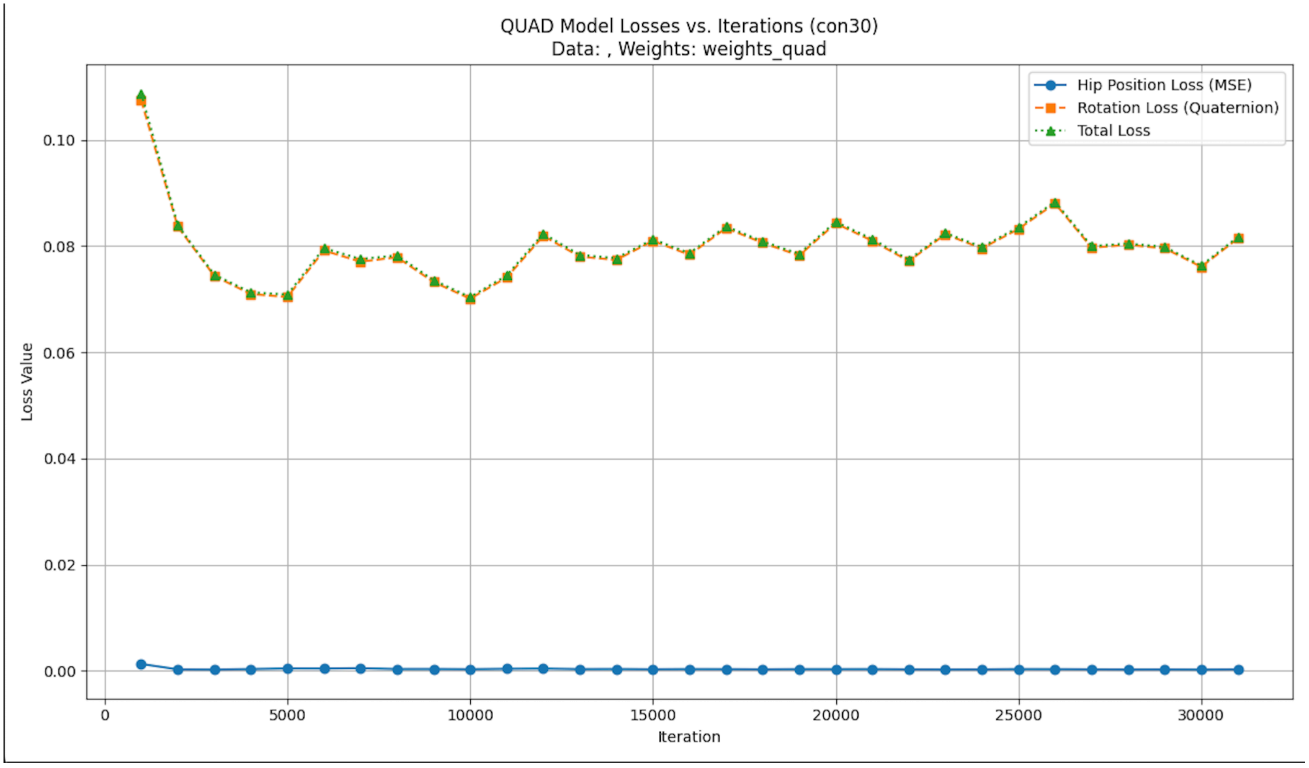
  A common observation across these plots is that the Hip Position Loss (MSE) component starts relatively low and remains significantly lower than the Rotation Loss throughout training. This is often normal because the hip position data, being normalized 3D Cartesian coordinates, might result in numerically smaller MSE values compared to the quaternion loss, which measures angular differences across many joints. Additionally, predicting the hip's trajectory can be a simpler task for the model than capturing the complex, high-dimensional rotations of all body joints, leading to faster and more effective initial learning for the hip component.

Figure 4: Quaternion Model Loss (Con5)



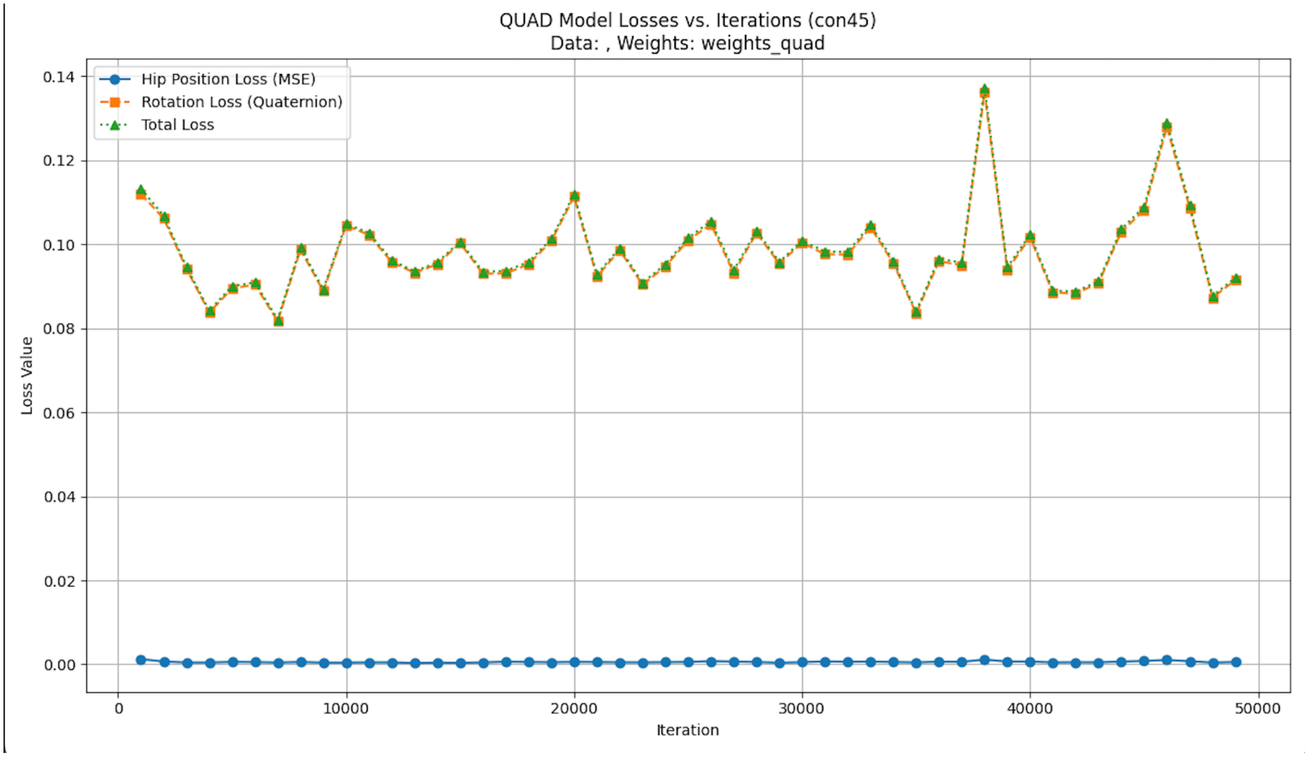
This plot shows the training loss for the quaternion model with `Condition_num=5`. The Total Loss and Rotation Loss start relatively high and decrease over the 28,000 iterations to around 0.025, though with some fluctuations. The Hip Position Loss remains consistently low throughout training, significantly lower than the rotation component.

Figure 5: Quaternion Model Loss (Con30)



This plot displays the training loss for the quaternion model with `Condition_num=30`, trained for 31,000 iterations. Both Total Loss and Rotation Loss show a notable decrease, appearing to stabilize at a higher level (0.08) compared to the Con5 model. The Hip Position Loss, as with other models, stays very low.

Figure 6: Quaternion Model Loss (Con45)



This plot illustrates the training loss for the quaternion model with `Condition_num=45`, trained for 49,000 iterations. The Total Loss and Rotation Loss decrease and then exhibit more pronounced fluctuations while generally remaining at a low level (around 0.09). These fluctuations likely reflect the increased difficulty of the training task: with a higher `Condition_num`, the model must predict further into the future based on its own potentially imperfect outputs, forcing it to learn more robustly and capture longer-term dependencies. This more challenging training regime, despite the visible variations in the loss curve, is what contributed to the best qualitative long-term motion generation. The Hip Position Loss remains minimal.

3.2. Synthesis - Qualitative Evaluation

Qualitative evaluation involved visually inspecting the synthesized motion for naturalness, coherence, and adherence to the learned style (salsa dance). This was primarily done using an online BVH viewer. The following qualitative observations were made.

- **Positional Representation:** The model trained with positional representation performed very well qualitatively. When synthesized with an initial seed sequence length (`SYNTH_INITIAL_SEQ_LEN`) of 100 frames and using the original training `condition_num=5` (as per the training script `pytorch_train_pos_aclstm.py`), the character was observed to dance proficiently for the entire 400-frame synthesized sequence. The motion was generally smooth and maintained the salsa dance style.
- **Euler Angle Representation:** The Euler angle representation presented more challenges. While the synthesized character generally performed dance-like motions, it was prone to occasional unnatural flips or twists. The training `condition_num` was kept at the default of 5. The impact of `SYNTH_INITIAL_SEQ_LEN` was investigated:
  - With `SYNTH_INITIAL_SEQ_LEN=100` frames (baseline), the motion was sustained for about 200 frames before becoming still, though occasional flips were present during the active phase.
  - With `SYNTH_INITIAL_SEQ_LEN=200` frames, the model successfully generated motion for the entire 400-frame duration. Flips were still present but the overall motion was more continuous.
  - With `SYNTH_INITIAL_SEQ_LEN=300` frames, the motion continued for 400 frames but exhibited excessive flipping, and a general less dance-like style.
  - Based on these observations, a seed length of 200 frames was deemed optimal for the Euler representation, balancing motion continuity with the occurrence of flips.

- **Quaternion Representation (Con5, Con30, Con45 experiments):** The quaternion representation initially suffered significantly from motion stillness.
    - **Con5 (`Condition_num=5`):** The model trained with `Condition_num=5` (the default auto-conditioning setting) produced motion that became static after only about 20 frames during synthesis. This was consistent even when varying `SYNTH_INITIAL_SEQ_LEN` .
    - **Con30 (`Condition_num=30`):** Increasing `Condition_num` to 30 showed some improvement, with motion lasting for approximately 60 frames before freezing.
    - **Con45 (`Condition_num=45`):** The model trained with `Condition_num=45` demonstrated the most significant improvement. The impact of `SYNTH_INITIAL_SEQ_LEN` for this Con45 model was further analyzed:
        - With `SYNTH_INITIAL_SEQ_LEN=100` frames, the motion was good for about 150 frames before stillness.
        - With `SYNTH_INITIAL_SEQ_LEN=200` frames, the dance-like motion extended to around 300 frames before stopping.
        - With `SYNTH_INITIAL_SEQ_LEN=300` frames, the Con45 model was able to generate relatively good, continuous salsa dance motion for the entire 400-frame synthesis duration.
    - The conclusion was that the combination of training with `Condition_num=45` and using a `SYNTH_INITIAL_SEQ_LEN=300` during synthesis provided the best qualitative results for the quaternion representation, successfully addressing the initial stillness problem.

3.3. Synthesis - Quantitative Evaluation

Quantitative evaluation aimed to measure the error between the synthesized motion and the ground truth continuation. The metrics were typically calculated for the first 20 generated frames immediately following a seed sequence.

- **Positional Representation:** The synthesis for the positional model (using `0049000.weight`) was performed with `SYNTH_INITIAL_SEQ_LEN=100` (as per its configuration). The quantitative evaluation shows Mean Squared Error (MSE) values for 10 different batches. For instance, Batch 0 had an MSE of `0.01716142` , and Batch 1 had an MSE of `0.04938352` . An average or representative MSE would summarize this, for example, the average of these 10 batches is approximately `0.0464` .
- **Euler Angle Representation:** For the Euler model (using `0051000.weight`), the synthesis providing the quantitative results was performed with `SYNTH_INITIAL_SEQ_LEN=200` (as this was qualitatively best). The output shows Hip MSE and Rotation Angle Distance for 10 batches. For example:
    - Batch 0: Hip MSE `0.0004` , Rot Ang Dist `0.0067` (Total Error Contribution: ~0.0071)
    - Batch 1: Hip MSE `0.0002` , Rot Ang Dist `0.0001` (Total Error Contribution: ~0.0003) An average could be reported, for instance, averaging the sum of Hip MSE and Rot Ang Dist for these 10 batches gives an approximate summed error of `0.0022` .
- **Quaternion Representation (Con5, Con30, Con45):** The quantitative results for quaternion models are reported as Total Loss (Hip MSE + Quaternion Loss). The `SYNTH_INITIAL_SEQ_LEN` used for these specific quantitative results was 300 frames.

    - **Con5 (`0028000.weight`):** From the synthesis results, Batch 0 Total Loss: `0.0824` (Hip MSE: 0.0008, Rot Quat Loss: 0.0815). Average Total Loss over 10 batches: ~ `0.1039` .
    - **Con30 (`0031000.weight`):** From the synthesis results, Batch 0 Total Loss: `0.0455` (Hip MSE: 0.0002, Rot Quat Loss: 0.0453). Average Total Loss over 10 batches: ~ `0.0414` .
    - **Con45 (`0049000.weight`):** From the synthesis results, Batch 0 Total Loss: `0.1341` (Hip MSE: 0.0005, Rot Quat Loss: 0.1335). Average Total Loss over 10 batches: ~ `0.1056` .

    It is important to note that while these quantitative results were obtained with a `SYNTH_INITIAL_SEQ_LEN=300` for the quaternion models, the qualitative discussion in Section 3.2

highlighted how different seed lengths affected the visual quality and duration of motion, particularly for the Con45 model where 300 frames was the best qualitatively.

3.4. Discussion of Part A Results

The core experiments in Part A provided valuable insights into the performance of Positional, Euler Angle, and Quaternion representations for character motion synthesis using an acLSTM model.

Overall Performance Comparison:

- **Positional Representation:** Demonstrated strong performance both qualitatively and quantitatively. It produced stable, long-duration motion that adhered well to the salsa style, and had reasonably low MSE values in quantitative tests. This representation, being largely provided, served as a solid baseline.
- **Euler Angle Representation:** Showed promise in generating dynamic motion but was hampered by issues of instability, manifesting as occasional flips or twists in the character. Qualitatively, a seed length of 200 frames provided the best balance for continuous motion over 400 frames, though flips persisted. Quantitatively, with this seed length, the summed error (Hip MSE + Rotational Angle Distance) was very low, suggesting good short-term prediction accuracy despite the visual artifacts in longer sequences.
- **Quaternion Representation:** Initially presented the most significant challenge due to the motion stillness problem. However, after systematic experimentation with the `Condition_num` training parameter, particularly the Con45 model, it achieved good qualitative performance, generating continuous 400-frame salsa motion when seeded with 300 frames. Quantitatively, the Con30 model showed the lowest average total loss among the quaternion variants tested under the specific synthesis conditions (300-frame seed), although the Con45 model was qualitatively superior for long-term motion.

Challenges and Solutions:

- **Euler Angles - Instability (Flips):** The primary challenge with Euler angles was the occurrence of flips. While the model could learn and generate dance-like sequences, the inherent properties of Euler angles likely contributed to these instabilities. Euler angles define rotations sequentially around X, Y, and Z axes. At certain rotation configurations (e.g., when pitch is +/- 90 degrees), two of the three rotational axes can align, leading to a condition known as **gimbal lock**. In this state, one degree of rotational freedom is lost, meaning it becomes impossible to rotate around one of the global axes independently. This can manifest as sudden, large, and unintuitive changes in orientation, or "flips," in the animation even for small changes in the input angles. The choice of seed sequence length helped in mitigating how quickly the motion might degrade or stop, but did not eliminate the flips, as gimbal lock is an intrinsic issue with the Euler angle representation itself.
- **Quaternions - Motion Stillness & `Condition_num`:** The most critical issue was the tendency of quaternion models (initially trained with `Condition_num=5`) to freeze after a short period. This was hypothesized to be due to the training regime not adequately preparing the model for long-term autoregressive synthesis. The solution, increasing `Condition_num` to 30 and then to 45, was a key finding. This forced the model to learn longer-range dependencies during training by making it rely on its own outputs for more extended periods. The Con45 model, when paired with a substantial seed sequence (300 frames), demonstrated that this approach effectively overcame the stillness problem. This highlights a critical aspect of training autoregressive models for sequential data: the training conditions must closely mirror the inference conditions to achieve robust long-term generation.

Observed Trade-offs:

- **Positional:** Relatively easy to train and produced stable, good-quality motion, but might be less expressive for highly complex or subtle rotational movements compared to orientation-based representations if not carefully handled.

- **Euler:** Capable of dynamic motion and achieved low quantitative short-term error. However, this came at the cost of visual stability, with flips being a persistent issue. Fine-tuning the seed length was necessary to maximize continuous motion duration.
- **Quaternions:** Theoretically superior for representing rotations (avoiding gimbal lock). However, they proved more challenging to train effectively for long-term synthesis. Achieving good results required significant tuning of the training process itself (the `Condition_num` parameter) and careful selection of synthesis parameters (`SYNTH_INITIAL_SEQ_LEN`). While the Con45 with a 300-frame seed produced good long motion, its quantitative short-term error (for the first 20 frames) was higher than the Euler model or the Con30 quaternion model. This suggests a possible trade-off between short-term predictive accuracy and long-term generative stability/quality for this specific setup.

The experiments underscore that there is no single "best" representation for all criteria. The choice depends on the specific requirements regarding motion stability, complexity, ease of training, and the acceptable level of artifacts.

4. Part B: Research Question - Impact of Seed Sequence Length

This part of the project aimed to investigate a specific research question extending from the core experiments.

4.1. Research Question Formulation

The research question guiding Part B was formulated as follows:

How does the length of the initial seed sequence (controlled by the `initial_seq_len` parameter during synthesis) affect the quantitative prediction error of the synthesized motion for:

1. The three different motion representations studied (Positional, Euler Angles)?
2. The different training configurations of the Quaternion model (specifically, models trained with `Condition_num=5` , `Condition_num=30` , and `Condition_num=45`)?

This investigation sought to understand if there is an optimal seed length for minimizing short-term prediction error and how this relationship varies across different model types and training strategies.

4.2. Methodology

The investigation into the impact of seed sequence length was conducted systematically using the `analysis.py` script. This script automated the process of running synthesis experiments with varying parameters and collecting quantitative error data.

The core methodology involved the following steps for each pre-trained model (Positional, Euler, Quaternion-Con5, Quaternion-Con30, and Quaternion-Con45):

1. **Varying Seed Lengths:** For each model, motion synthesis was performed multiple times using three different initial seed sequence lengths (`initial_seq_len`): 100 frames, 200 frames, and 300 frames. These values were defined in the `INITIAL_SEQ_LENS` variable within `analysis.py` .
2. **Short Sequence Synthesis:** In each run, the models were tasked to synthesize a relatively short sequence of motion (20 frames, as defined by `QUANTITATIVE_COMPARISON_LEN` in `analysis.py`) immediately following the provided seed.
3. **Error Metric Calculation:** The synthesized 20-frame sequence was quantitatively compared against the corresponding 20 frames of ground truth motion from the dataset. The specific error metrics used were consistent with those in Part A:
    - **Positional:** Mean Squared Error (MSE).

- **Euler:** Summed Loss (Hip MSE + Rotational Angle Distance).
- **Quaternion:** Total Loss (Hip MSE + Quaternion Loss). The `analysis.py` script parsed these metrics from the standard output of the respective synthesis scripts.

4. **Averaging Results:** To ensure robustness of the findings, each configuration (model type + seed length) was run multiple times (`NUM_RUNS_PER_CONFIG = 3` in `analysis.py`), and the error metrics from these runs were averaged to produce a mean error for that specific configuration. The standard deviation was also calculated to understand the variability of the error.

This automated procedure allowed for a consistent comparison of how different seed lengths influenced the short-term prediction accuracy across all the developed models.

4.3. Results

The results of the Part B investigation, detailing the average quantitative error for each model configuration across different initial seed sequence lengths (100, 200, and 300 frames), are presented below. These values are sourced from the output of the `analysis.py` script. The error reported is for the first 20 frames of synthesized motion beyond the seed.

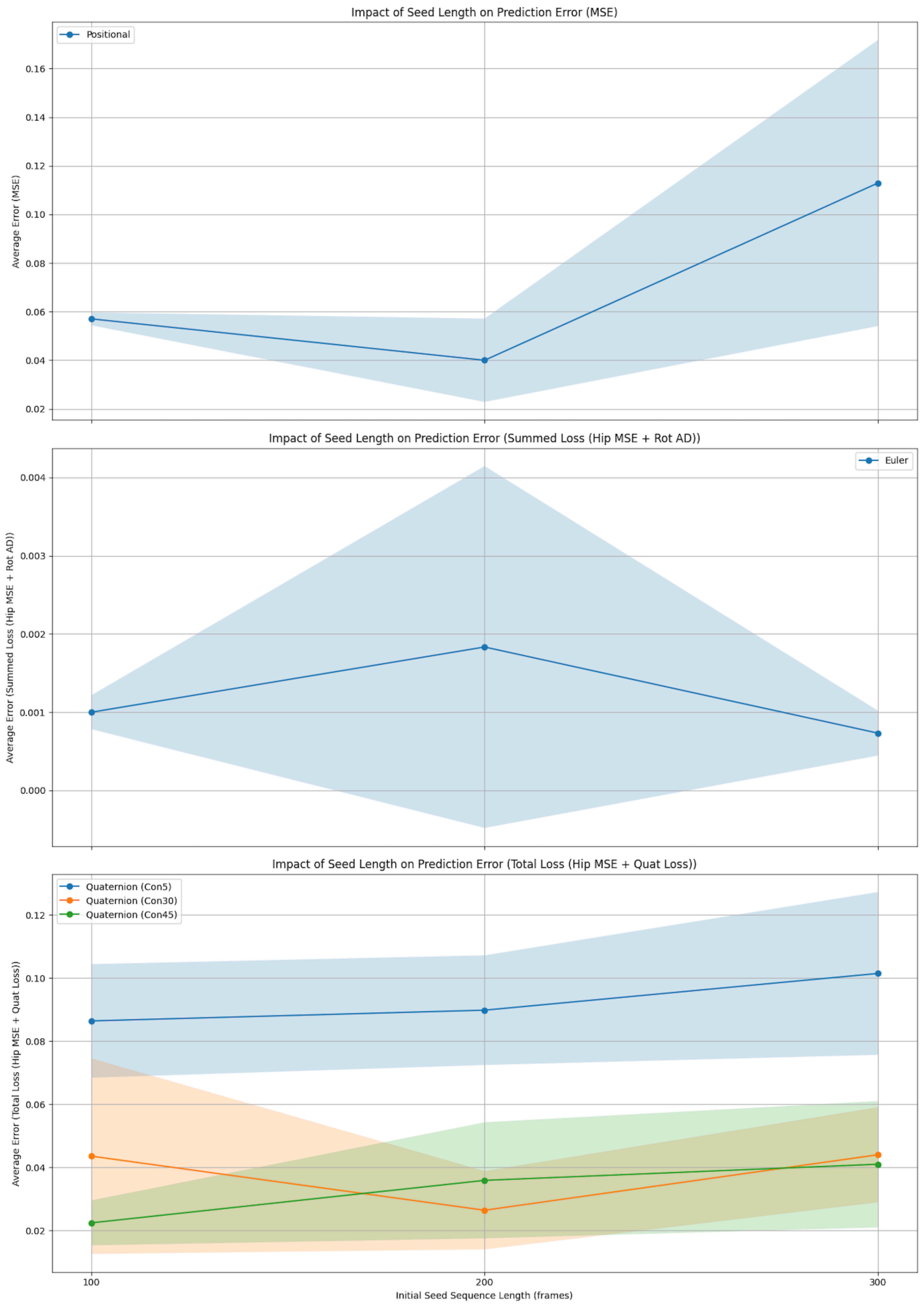
Summary of Average Errors (+/- Standard Deviation):

- **Positional Model (MSE):**
  - Seed 100 frames: `0.0570 +/- 0.0026`
  - Seed 200 frames: `0.0400 +/- 0.0171`
  - Seed 300 frames: `0.1129 +/- 0.0588`
- **Euler Model (Summed Loss: Hip MSE + Rot AD):**
  - Seed 100 frames: `0.0010 +/- 0.0002`
  - Seed 200 frames: `0.0018 +/- 0.0023`
  - Seed 300 frames: `0.0007 +/- 0.0003`
- **Quaternion Model - Con5 (Total Loss: Hip MSE + Quat Loss):**
  - Seed 100 frames: `0.0864 +/- 0.0180`
  - Seed 200 frames: `0.0898 +/- 0.0174`
  - Seed 300 frames: `0.1014 +/- 0.0258`
- **Quaternion Model - Con30 (Total Loss: Hip MSE + Quat Loss):**
  - Seed 100 frames: `0.0435 +/- 0.0310`
  - Seed 200 frames: `0.0264 +/- 0.0124`
  - Seed 300 frames: `0.0440 +/- 0.0150`
- **Quaternion Model - Con45 (Total Loss: Hip MSE + Quat Loss):**
  - Seed 100 frames: `0.0224 +/- 0.0071`
  - Seed 200 frames: `0.0359 +/- 0.0184`
  - Seed 300 frames: `0.0410 +/- 0.0200`

Figure 7 visually summarises the quantitative impact of initial seed sequence length (100, 200, or 300 frames) on prediction error, as detailed in Section 4.3. The figure is composed of three subplots:

1. **Top Subplot (Positional Model, MSE):** Shows that the Mean Squared Error for the Positional model is minimized with a 200-frame seed.
2. **Middle Subplot (Euler Model, Summed Loss):** Illustrates that the Summed Loss (Hip MSE + Rotational Angle Distance) for the Euler model is lowest with a 300-frame seed, while also being low for a 100-frame seed.
3. **Bottom Subplot (Quaternion Models, Total Loss):** Displays the Total Loss (Hip MSE + Quaternion Loss) for the three Quaternion configurations:

- **Con5:** Error generally increases with seed length.
- **Con30:** Error is minimized at a 200-frame seed.
- **Con45:** Error is lowest with a 100-frame seed. In each subplot, the x-axis represents the initial seed sequence length, the y-axis represents the respective average error metric, and the shaded areas indicate the standard deviation of the error.



4.4. Discussion of Part B Findings

The quantitative results presented in Section 4.3 and visualized in Figure 7 reveal several interesting trends regarding the impact of initial seed sequence length on short-term prediction error.

General Trends and Optimal Seed Lengths:

- **Not Always Better with Longer Seeds:** Contrary to a simple intuition that more initial context (longer seed) should always lead to lower prediction error, this was not universally observed. For some models, error increased with the longest seed length (300 frames).
- **Positional Model:** The lowest MSE (`0.0400`) was achieved with a 200-frame seed. Both shorter (100 frames: `0.0570`) and longer (300 frames: `0.1129`) seeds resulted in higher error. This suggests an optimal seed length around 200 frames for the best short-term quantitative performance for this model. The standard deviation was also notably higher for the 300-frame seed, indicating less stable error.
- **Euler Model:** The Euler model exhibited very low errors overall. The lowest summed loss (`0.0007`) was observed with the longest seed (300 frames). A 100-frame seed also performed well (`0.0010`), while the 200-frame seed surprisingly showed slightly higher average error (`0.0018`) and significantly higher standard deviation, suggesting some instability or variability at this particular seed length for short-term prediction. This quantitative finding for the 200-frame seed contrasts somewhat with the qualitative assessment where 200 frames was preferred for longer, more continuous motion (though still with flips).
- **Quaternion Models:**
  - **Con5:** This model, which performed poorly qualitatively, showed a trend of increasing error with longer seed lengths (100 frames: `0.0864` , 200 frames: `0.0898` , 300 frames: `0.1014`). This suggests that for a poorly trained model (low `Condition_num`), more context might not help and could even be detrimental if the model cannot effectively utilize it.
  - **Con30:** This model showed its lowest error (`0.0264`) with a 200-frame seed. Both 100-frame (`0.0435`) and 300-frame (`0.0440`) seeds resulted in higher errors. The standard deviation was highest for the 100-frame seed.
  - **Con45:** This model, which was best qualitatively with a 300-frame seed for long-term motion, showed its lowest quantitative short-term error with a 100-frame seed (`0.0224`). The error increased for 200-frame (`0.0359`) and 300-frame (`0.0410`) seeds. This is an interesting divergence: the seed length optimal for short-term quantitative prediction (100 frames) was not the same as that for long-term qualitative performance (300 frames).

Alignment with Qualitative Observations:

- **Euler Model:** As mentioned, the quantitative results for Euler (300-frame seed being best for low error) didn't perfectly align with qualitative findings (200-frame seed best for overall motion quality over 400 frames). This highlights that short-term prediction error doesn't always directly correlate with long-term generation quality or the absence of specific artifacts like flips.
- **Quaternion Con45 Model:** The most notable divergence was with the Con45 model. Qualitatively, it required a 300-frame seed to generate continuous 400-frame motion. However, for minimizing quantitative error over the first 20 generated frames, a 100-frame seed was optimal. This suggests that while a longer seed provides richer dynamic context crucial for sustained, stable long-term generation (as discussed regarding the training adaptations), it might lead to a noticeably higher deviation (in this case, the error was approximately 83% greater) from the ground truth in the immediate frames following the seed. One possible interpretation is that a 100-frame seed offers sufficient information for the model to make accurate short-term predictions without overfitting to the nuances of a very long sequence. Conversely, a 300-frame seed, while beneficial for establishing a robust internal state for long-term stability (helping the model "understand" the broader motion context and avoid freezing), might introduce more complex dynamics or subtle patterns from the extended history. The model, while capable of leveraging this for sustained motion, might exhibit higher error in the immediate next few frames as it "settles in" or transitions from this very specific long-term

context, compared to the more straightforward prediction task following a shorter seed. This highlights a potential trade-off: a shorter seed may be better for immediate, precise replication, while a longer seed might be necessary for the model to build the internal momentum and understanding required for coherent, extended generation, even if short-term precision decreases as a consequence.

Error Stability (Standard Deviation): The standard deviations reported alongside the mean errors give insight into the consistency of performance.

- For the Positional model, the 300-frame seed had a much larger standard deviation (`0.0588`) compared to 100-frame (`0.0026`) or 200-frame (`0.0171`) seeds, indicating less predictable performance at that longer seed length.
- For the Euler model, the 200-frame seed had a notably high standard deviation (`0.0023`) compared to 100-frame (`0.0002`) and 300-frame (`0.0003`) seeds, suggesting its error was more variable for that specific seed length.
- For Quaternion models, Con30 with a 100-frame seed had a relatively high standard deviation (`0.0310`). Other configurations had more moderate standard deviations.

In summary, the Part B investigation revealed that the optimal seed length for minimizing short-term quantitative error is model-dependent and does not always align with the seed length that produces the best long-term qualitative results. Longer seeds are not universally better, and some models benefit from shorter or intermediate seed lengths for immediate prediction accuracy. The Quaternion Con45 model, in particular, demonstrated a clear trade-off between the needs of short-term accuracy and long-term generative stability.

5. Further Discussion & Challenges

This section reflects on broader observations and challenges encountered during the project, emphasizing insights gained beyond the primary experimental results.

One area of exploration involved attempts to fine-tune model behavior through modified loss functions, such as applying differential weighting to components of the loss for Euler and Quaternion representations. The initial hypothesis was that this might mitigate issues like rotational instabilities (e.g., flipping with Euler angles) or motion freezing (with Quaternions). However, these attempts did not yield straightforward improvements and, in some cases, introduced new complexities. Ultimately, the more impactful solution for the Quaternion motion stillness, for instance, was found by adjusting the training methodology itself—specifically, the auto-conditioning parameters—rather than through intricate loss re-weighting.

Another point of reflection concerns the role of data normalization. While standard normalization techniques, such as scaling hip positions, were applied to stabilize training, initial theories about normalization causing adverse effects like motion stillness with quaternions were reconsidered. The evidence pointed more strongly towards the training regime (particularly the interplay between `Condition_num` and `Groundtruth_num` in the acLSTM) and exposure bias as dominant factors in achieving long-term generative stability. Problems like motion freezing are often rooted in how well the model learns to cope with its own generated outputs over extended periods, a challenge that was more effectively addressed by modifying the training strategy to better simulate inference conditions.

The project underscores that achieving robust, long-term motion synthesis is a nuanced task. While the theoretical advantages of certain representations are clear (e.g., quaternions avoiding gimbal lock), practical success hinges on a holistic approach. This includes careful data preprocessing, appropriate network architecture, and, critically, a training regime that adequately prepares the model for the demands of extended autoregressive generation. The experience gained suggests that robust solutions to generative failures often lie in refining the core training process to align better with the inference objectives, which can be more effective than isolated adjustments to loss functions or standard normalization practices.

6. Conclusion

This project successfully explored character motion prediction using an Auto-Conditioned LSTM (acLSTM) model across three different motion representations: Positional, Euler Angles, and Quaternions, using the "salsa" dance dataset. The main achievements include the implementation and adaptation of preprocessing pipelines, training procedures, and evaluation methods for each of these representations.

A key finding from the comparative analysis (Part A) was the distinct trade-offs offered by each representation. The Positional model provided a stable and qualitatively strong baseline. The Euler Angle model, while achieving very low short-term prediction errors, suffered from visual instabilities such as flips, though its motion was generally dynamic. The Quaternion model initially faced significant challenges with motion stillness during synthesis.

The primary contributions of this project are twofold. Firstly, the successful adaptation and comparative analysis of the Euler and Quaternion representations within the acLSTM framework, including the development of specific data processing and loss function implementations. Secondly, and critically, the investigation and effective resolution of the motion stillness problem encountered with the Quaternion representation. By systematically adjusting the `Condition_num` parameter in the acLSTM training regime (specifically, increasing it to 45) and identifying an optimal seed length (300 frames) for synthesis, continuous and dynamic long-term motion generation was achieved for quaternions. This demonstrated the crucial impact of aligning training conditions with inference demands for autoregressive models.

The Part B investigation into the effect of initial seed sequence length on quantitative prediction error revealed that optimal seed length is model-dependent and does not always correlate with what produces the best long-term qualitative motion. For instance, the Quaternion Con45 model, which required a 300-frame seed for best qualitative long motion, exhibited lower short-term prediction error with a 100-frame seed.

Potential future work could explore several avenues. Investigating more advanced loss functions or weighting schemes, particularly for Euler angles to mitigate flips, could be beneficial. Further exploration of network architectures beyond the standard LSTM, or hybrid representations, might yield improved performance. Additionally, applying these models to different motion styles or datasets would test the generalizability of the findings. Finally, exploring methods to more directly correlate short-term quantitative metrics with long-term perceptual quality of motion remains an open area of research.

7. References

[1] Zimo Li et al. *Auto-Conditioned Recurrent Networks for Extended Complex Human Motion Synthesis*. 2018. arXiv: 1707.05363.

[2] Dario Pavllo, David Grangier, and Michael Auli. *QuaterNet: A Quaternion-based Recurrent Model for Human Motion*. 2018. arXiv: 1805.06485.