Laying The Foundations for Real-Time Auto-Segmentation of Time Series Gaze and Movement Data

by

Thomas Riley Lakusta Dawson

A final report submitted in partial fulfillment of the requirements for the degree of

Master of Business/Master of Engineering

in

Software Engineering & Intelligent Systems

Electrical and Computer Engineering
University of Alberta

© Thomas Riley Lakusta Dawson, 2024

Abstract

Motion capture technology is a powerful tool for revealing insight into human behavior and decision-making, but the segmentation of time series-based movement data into subphases for precise analysis can be an arduous and time-consuming process. Bespoke segmentation rulesets require iteration, and interdisciplinary expertise, and may only be applicable post-hoc.

The main objective of this project was to explore the feasibility of leveraging machine learning techniques to auto-segment gaze and movement data, with the ultimate goal of the development of a system capable of real-time segmentation. The project consisted of two primary phases: data preparation and model design. An existing multi-site corpus of labeled gaze and movement data was curated and processed into a condensed set for training. The standard Multi-Layer Perceptron (MLP), and the Long Short-Term Memory (LSTM), architectures were selected for performance comparison. Alongside architecture choice, context window length and the inclusion of previous sample phase predictions were explored as additional training conditions.

While both architectures were able to achieve better-than-chance accuracy, every LSTM architecture outperformed its MLP analog as predicted. With a cross-fold peak average accuracy of 93%, the best-performing configuration was an LSTM model with a single frame context window and no previous sample predictions.

The project findings confirm that machine learning techniques offer significant promise for automated movement segmentation. Single frame context window lengths and previous frame prediction inclusion offering no tangible benefit suggest that alternative machine learning strategies, such as decision trees, are worth exploring in future research.

Table of Contents

Abstract i
Table of Contents ii
List of Figures iii
List of Tables iv
1.0 Introduction 1
2.0 Methodology 2
2.1 Data Set 2
2.1.1 Collection 2
2.1.2 Pre-processing 4
2.2 Model Design 5
2.2.1 MLP Architecture 6
2.2.2 LSTM Architecture 7
3.0 Results 9
4.0 Discussion 12
5.0 Conclusions 14
6.0 References 17
Appendix A: Pasta Box Task Apparatus and Sequence 20
Appendix B: Included GaMA Features 21
Appendix B: Software Tools 24
Appendix D: Test/Train Performance 25

List of Figures

Figure 1: Simplified MLP Architecture 6

Figure 2: LSTM Cell Architecture 8

Figure 3: LSTM-1-NP Best, Median, and Worst Performance 10

Figure 4: LSTM-1-NP Phase Labeling Accuracy 11

List of Tables

Table 1: True Phase Proportions across Trials 3

Table 2: Data Set Composition Details 4

Table 3: Cross-Fold Split Summaries 6

Table 4: Model Training Summary Results 9

Table 5: Predicted Phase Proportions 9

1.0 Introduction

Hardware and software solutions for motion capture are immensely valuable for understanding human behavior and biomechanics, with applications in medicine, sports performance, and human factors engineering (Nunes et al., 2015). While the gold standard for motion capture remains infrared-based, multi-camera marker tracking (Adesida et al., 2019), markerless motion capture via standard consumer cameras has shown exceptional promise and has begun to see broader industry adoption (Cronin et al., 2023).

As the barriers to acquiring motion data have continued to abate, the tooling for deriving meaningful insight from it has remained largely bespoke. Even while constraining analysis to a niche domain, the development of extensible tooling is immensely time-consuming and requires interdisciplinary domain expertise (Nunes et al., 2015).

Segmentation of the motion data into composite movement phases (i.e. reaching, kicking, etc.) is critical for effective analysis, but this in itself is incredibly challenging. One can rely on simple, coarse signals (such as an "object interaction has begun," "a set period has expired," etc.) that are quick to deploy but lack expressivity, or they can use highly refined rulesets that are resource-intensive to define, but robust to diverse movements and participant task completion strategies. One flexible approach for creating a segmentation ruleset consists of defining a basket of conditions that must be met for a new segment to begin. For example, the onset of reach can be defined as beginning when either the velocity of the hand exceeds 5% of its maximum velocity or the distance between it and its starting position exceeds a threshold value. These rules can then be batch-applied to the complete data set to unlock the next stage of analysis. The difficulty of defining an effective ruleset is compounded by the immense variability inherent between target populations. A ruleset defined for an able-bodied individual may fail when used for an individual who uses a prosthetic limb, for example.

A more intuitive and fluid approach to building segmentation pipelines would be to have a domain expert label the onset of different movements and then have an autonomous agent learn what signals are associated with those transitions. Beyond simplifying and speeding up the segmentation definition process, this approach would bring a system closer to providing real, or near-real-time, insight. While there have been great strides in using deep learning for text

prediction (Vaswani et al., 2017) and image understanding and generation (Rombach et al., 2022), it remains to be seen if such tools will readily apply and prove useful for time series motion data analysis.

The motivation for this project is to explore the feasibility of applying topical deep-supervised learning architectures to segment gaze and motion capture data. The project consists of two phases, data pre-processing and model development. The test/training data for this project will be adapted from an existing composite data set of labeled gaze and movement data. The source data consists of individual trials of participants performing a simple object relocation task.

2.0 Methodology

2.1 Data Set

2.1.1 Collection

Motion capture data was collected across four sites following the Pasta Box Task Protocol (Valevicius et al., 2018). The task consists of a participant, outfitted with infrared marker cluster plates (used for rigid body definition) maneuvering a pasta box around a test apparatus. The participant begins standing with their hand resting on a table with two shelves of different heights. A complete trial consists of 3 composite movements: reaching for the pasta box, grasping it, transporting it to a new location, releasing it, and returning the hand to its starting position. There are three target locations, the box starting position, the first shelf target, and the second shelf target. See Appendix A: Pasta Box Task Apparatus and Sequence for a visualization of the task apparatus and movement sequence.

Post-collection, motion capture data was then exported from its raw format to Matlab files for ingestion and processing using the Gaze and Movement Analysis (GaMA) research software platform (Williams et al., 2019). Processing in GaMA consisted of constructing a three-dimensional recreation of each trial, complete with time series entity features such as distances between objects (i.e. a participant's hand and the pasta box), velocities, accelerations, and binary "Logicals" that maintain state and interaction status between entities. These

secondary features serve as the primary resource for defining the composite movement segments of the task. The relative distribution of movement phases averaged across all trials is summarized in Table 1: True Phase Proportions across Trials below.

Table 1: True Phase Proportions across Trials		
Phase	Average (%)	Standard Deviation (%)
Home	43.35	3.78
Reach	14.99	1.77
Grasp	5.62	1.07
Transport	28.22	2.27
Release	7.81	1.85

Three of the data sets were collected using Optitrack motion capture systems and one was recorded using a Vicon motion capture system. See Table 2: Data Set Composition Details for a breakdown of each site's total collected trials, their recording frequency, and a sample of the number of data fields created per trial. While all source data sets were collected using the same protocol, some details remain disparate in their creation, as evidenced by the variable number of data fields. These differences are primarily due to variations in protocol adherence, the need for further filtering/cleaning of raw data, process improvements and refinement between collection periods (i.e. dynamic support for best gaze vector selection), and inclusion of superfluous fields (i.e. including unused, unlabelled data during export from the capture software).

Table 2: Data Set Composition Details				
Data Set	Total Trials	Motion Capture System	Frame Rate (Hz)	Sample Number of Data Fields
A	400	Optitrack Motive	120	1031
B	379	Optitrack Motive	120	1114
C	399	Vicon Nexus	200	1086
D	200	Optitrack Motive	120	1167

The original, GaMA processed data sets included the following data types: Timestamps (sec), raw marker positions (3D point, mm), Rigid Bodies (3D point, mm & 3D rotation, degrees), 3D Objects (3D vertex points, mm), Pupil Coordinates (2D Point, pixel coordinates), Gaze Vectors (3D origin and fixation points, mm), Distances (mm), Velocity (mm^2) and Acceleration (mm^3) derivatives, Boolean Logicals, and Angles (degrees).

2.1.2 Pre-processing

A subset of Distances, Derivatives, and Angles, were selected as the candidate features for this analysis, as they are the primary features informing the segmentation ruleset. A full listing of the selected features and their definitions is provided in Appendix B: Included GaMA Features. Unused features were dropped from the complete GaMA projects during conversion from the Matlab file format to Comma Separated Value (CSV) files. Seven trials from data set B were additionally dropped at this stage due to incompleteness. The 200 Hz data set was downsampled using the closest time-step pairing to match the 120 Hz framerate of the other three data sets.

All individual CSV files were then collated into Python Numpy arrays, stored in a single Numpy Python file, and the mean and standard deviation for each target feature were calculated. The data was then normalized and standardized using the feature descriptive statistics. Finally, a separate data set of 500 msec windows (flattened 60 frame tensors) was generated from the standardized master data set. This number was chosen to inform whether multi-frame samples improved classification accuracy while still maintaining the potential for near-real-time

processing. The 500 msec data set uses a five-frame offset to balance between providing a comparable number of training samples to the longer context models while ensuring the full data set can be loaded in memory for efficient processing. A top-level listing of which software and version was used for each step of data processing, as well as model training and evaluation, is available in Appendix B: Software Tools.

2.2 Model Design

While there have been impressive advances in machine learning across a plethora of applications, time series and spatial data analysis has remained a persistent challenge and active area of research (Jin et al., 2023). Various specialized architectures have been proposed to achieve greater performance than possible with standard feed-forward multi-layer neural networks. For example, convolutional neural networks have proven effective for image recognition and classification (Rombach et al., 2022).

While there is no end to possible architectures to explore, two simple implementations were selected for benchmarking performance: a standard Multi-Layer Perceptron (MLP) and a Long Short-Term Memory (LSTM) based network. The number of output neurons for each layer was kept consistent between both architectures to facilitate comparison between models of approximately equivalent complexity. Cross entropy was used as the loss function for training, and accuracy for both architectures was defined as the percentage of correctly labeled frames per data set.

As a final experimental condition, the inclusion of the previous frame phase prediction for each sample was tested to aid in discerning whether or not model performance could be improved by leveraging the presumed time series nature of phase classification, resulting in eight total model/data configurations. Every sample for these models included an appended phase prediction of its immediately preceding sample. Due to the five frame sliding window, the previous sample prediction for the 60 frame context window models corresponds to the fifth last frame in the subsequent sample. A zeroed-out initial prediction was provided for the first prediction of each epoch.

Five-fold cross-validation was used to gauge the performance of each architecture and context window pairing. Each validation fold consists of a non-intersecting set of 20% of each participant's trials, equating to 256 trials per fold. Due to the number of trials per participant and

trial length varying, each fold contains a slightly different number of inputs. Summary information for each test/train split is shown in Table 3: Cross-Fold Split Summaries.

Every model/data configuration was run for five epochs in total. This number of epoch runs was chosen to provide insight into how susceptible each configuration is to overfitting without requiring excessive training time.

Table 3: Cross-Fold Split Summaries

Context Window Size (frames)	Fold	Number of Train Samples	Number of Test Samples	Test/Train Split
60	1	351,181	84,162	0.193
	2	351,002	84,341	0.194
	3	351,354	83,989	0.193
	4	350,770	84,573	0.194
	5	350,953	84,390	0.194
1	1	1,755,964	420,865	0.193
	2	1,755,069	421,760	0.194
	3	1,756,825	420,004	0.193
	4	1,753,908	422,921	0.194
	5	1,754,821	422,008	0.194

2.2.1 MLP Architecture

The Multi-Layer Perceptron is the fundamental architecture underlying the field of supervised deep learning (Sarker, 2021). It consists of a fixed number of layers (one input layer, one output layer, and n-many hidden layers), each with a pre-defined number of neurons. Each neuron accepts some input and multiplies it by a weight. The outputs of all neurons on a layer, including a bias neuron, are then summed together and passed through a

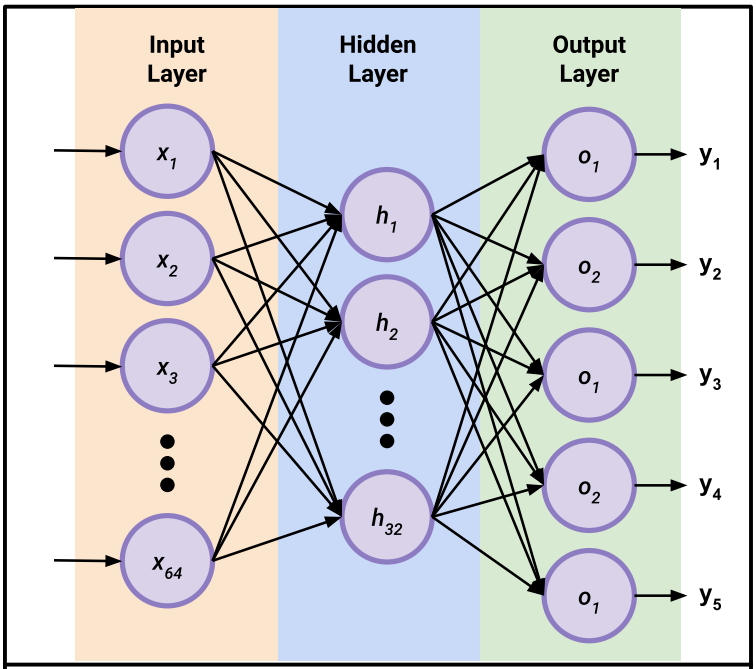


Figure 1: Simplified MLP Architecture. The fully connected model consists of a 64 node input layer, a 32 node hidden layer, and a 5 node output layer. The ReLU function is used for node activation, and the log softmax function is used for output layer's phase prediction output. Adapted from Sarker, 2021.

non-linear activation function, before then passing through to the next layer of neurons. The final output layer's predicted values are then compared to the actual labels for the given input and assessed using a defined loss function. The prediction error is then propagated back through the network to update each neuron's weights, thus training the network.

The MLP architecture used for this project, displayed in Figure 1: Simplified MLP Architecture, under all test conditions, consists of an input layer with 64 neurons, a hidden layer with 32 neurons, and an output layer with five neurons. The input layer and hidden layers use the rectified linear unit (ReLU) function as their activation function (Krizhevsky et al., 2012; see Eqn. 1). The final output layer uses the log softmax function for final segment classification (see Eqn. 2).

$$\mathrm{relu}(x) = \max(x, 0)$$

Eqn. 1:Rectified Linear Unit Function. Adapted from Heek et al., 2023.

$$\mathrm{log_softmax}(x)_i = \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right)$$

Eqn. 2: Log Softmax Function. Adapted from Heek et al., 2023.

2.2.2 LSTM Architecture

For many years, the standard structure deployed when handling time series data has been Long Short-Term Memory cells (Kader et al., 2022). LSTM-based networks are a specialized form of Recurrent Neural Network (RNN) (Hochreiter & Schmidhuber, 1997). In traditional RNNs, a hidden state propagates throughout training by passing the output of a node back into the same node as an additional input. One weakness of this implementation is that the impact of errors vanishes as training continues, also known as the vanishing gradient problem (Rueckert & Fichtinger, 2019). LSTMs address this weakness by including a "forget gate" to modulate how much incoming information is accepted into the cell.

Each LSTM cell has three gates: the input gate, the forget gate, and the output gate (see Figure 2: LSTM Cell Architecture). The input to each cell is concatenated with a previous hidden state, and this combined input then undergoes a pointwise multiplication with the previous cell's persistent state. This process is the forget gate, which acts to determine what information persisted from the previous state should be forgotten. The concatenated and sigmoid

$$\mathrm{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

Eqn. 3: Sigmoid Function. Adapted from Heek et al., 2023.

function-activated result (see Eqn. 3: Sigmoid Function) is then added to the output of the input gate. The input gate is another pointwise multiplication result, derived from the previous hidden state, the previous cell state, and a candidate value generated by the hyperbolic tangent operation applied to the concatenation of the current input and previous hidden state. This summation of the forget and input gate outputs represents the new cell state. Finally, the output gate uses the current input, the previously hidden state, and the newly calculated cell state to generate the final output of the LSTM cell.

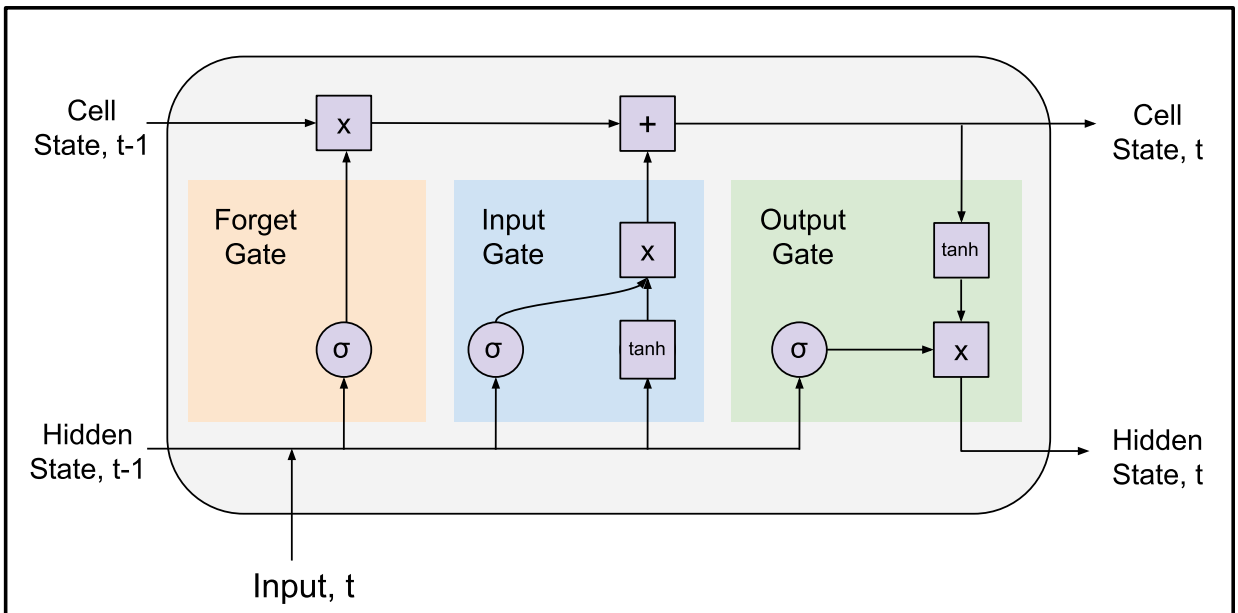


Figure 2: LSTM Cell Architecture. Each circular node represents a fully connected layer with the corresponding activation function (sigmoid or tanh). Each square block indicates an element-wise operation of the indicated function (multiplication, addition, or tanh) on the incoming vectors. Time-steps for states are indicated by "t". Adapted from Zhang et al. 2023.

Similar to the MLP architecture, the LSTM network architecture for this project, under all test conditions, consists of a layer with 64 LSTM cells, a hidden fully connected layer with 32 feed-forward neurons, and an output layer with five neurons. The hyperbolic tangent function, which constrains the output between -1 and 1, was used as the activation function for the output and memory update gates. As in the case of the MLP architecture, the ReLU activation function is used for the hidden feed-forward layer, and the log softmax function is used for the output layer.

3.0 Results

The test/train loss and accuracy for each model/data configuration were calculated at the end of each epoch, as well as before any training to capture baseline performance. The average of each model/data configuration's loss and accuracy was calculated across all five cross-validation folds. Visualizations of each model's test/train loss and accuracy across epochs are supplied in Appendix D: Test/Train Performance. The peak average test accuracy across folds for each model/data configuration is provided in Table 4: Model Training Summary Results, alongside the corresponding epoch and across-fold average training accuracy. The across-trial average phase proportions are provided in Table 5: Predicted Phase Proportions.

Table 4: Model Training Summary Results

Name	Model Architecture	Context Length (frames)	Last Frame Prediction	Peak Average Test Accuracy	Average Training Accuracy	Peak Test Accuracy Epoch
MLP-1-NP	MLP	1	Excluded	0.8679	0.9274	1
MLP-1-P			Included	0.4301	0.4659	1
MLP-60-NP		60	Excluded	0.8937	0.9196	1
MLP-60-P			Included	0.8929	0.9262	1
LSTM-1-NP	LSTM	1	Excluded	0.9300	0.9685	1
LSTM-1-P			Included	0.9183	0.9523	5
LSTM-60-NP		60	Excluded	0.9131	0.9561	5
LSTM-60-P			Included	0.9118	0.9545	4

Table 5: Predicted Phase Proportions

Phase	Average (%)	Standard Deviation (%)
Home	44.82	3.64
Reach	13.24	1.77
Grasp	5.66	1.65
Transport	26.81	2.51
Release	9.47	2.96

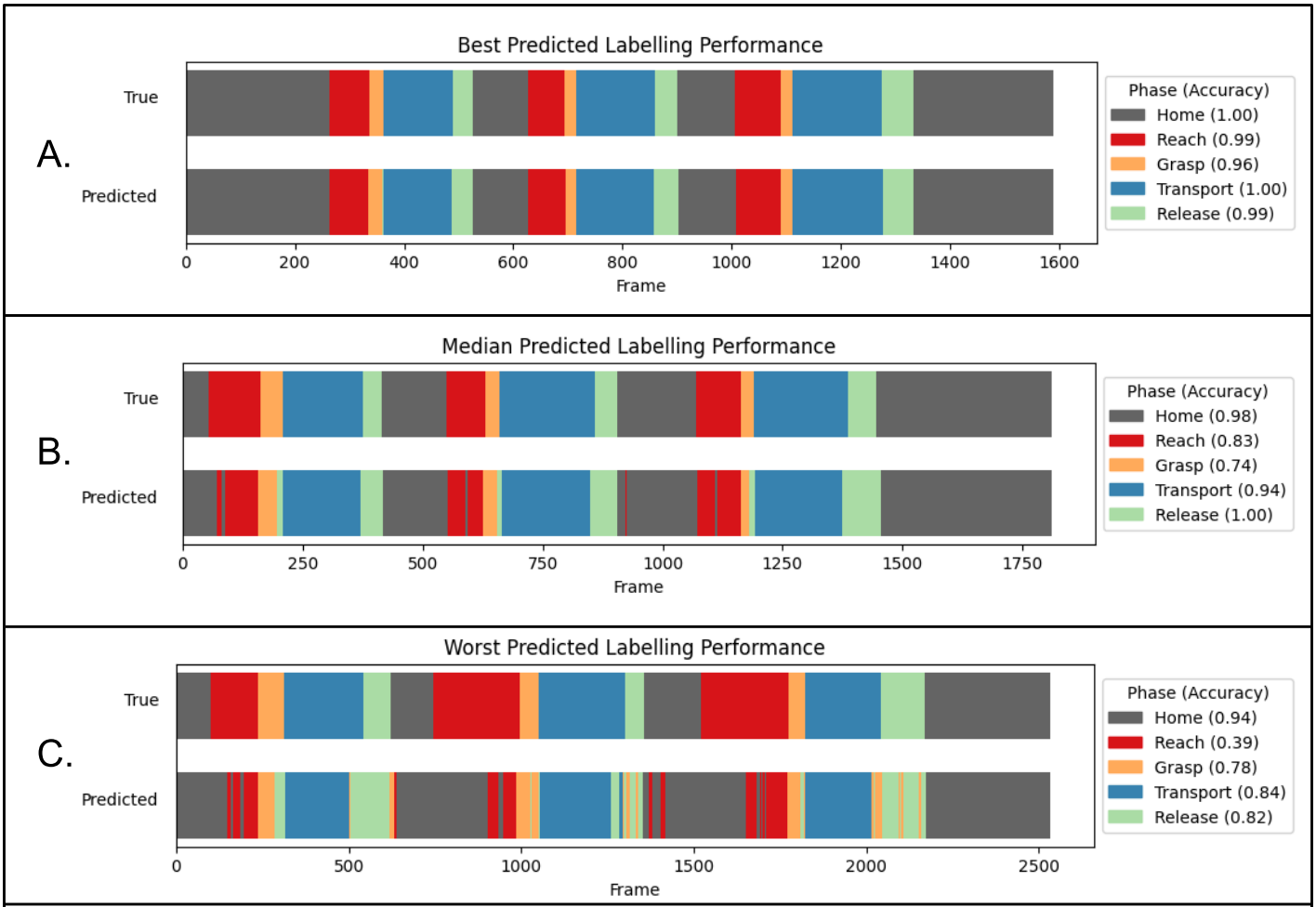


Figure 3: LSTM-1-NP Best, Median, and Worst Performance. The top row for each subplot visualizes the true phase label across a test trial, while the bottom row visualizes the LSTM-1-NP model's phase predictions. The x-axis for every plot shows each trial's sequential frame number. Individual movement phases are indicated by color.

The LSTM model architectures achieved higher accuracy than the MLP architectures, and the inclusion of previous sample phase predictions did not significantly improve performance. With a peak average test accuracy of 93%, the best-performing model overall was the LSTM architecture trained using the single frame context window data set, and excluding previous sample predictions (LSTM-1-NP). One model/data configuration, MLP-1-P, suffered from an immediate loss explosion and accordingly had significantly worse accuracy than all other models.

Select test trial performance for the LSTM-1-NP (fold 3) model is visualized in Figure 3: LSTM-1-NP Best, Median, and Worst Performance. The three demonstrative test trials chosen were: best, median, and worst performance, defined in terms of cumulative phase accuracy. The

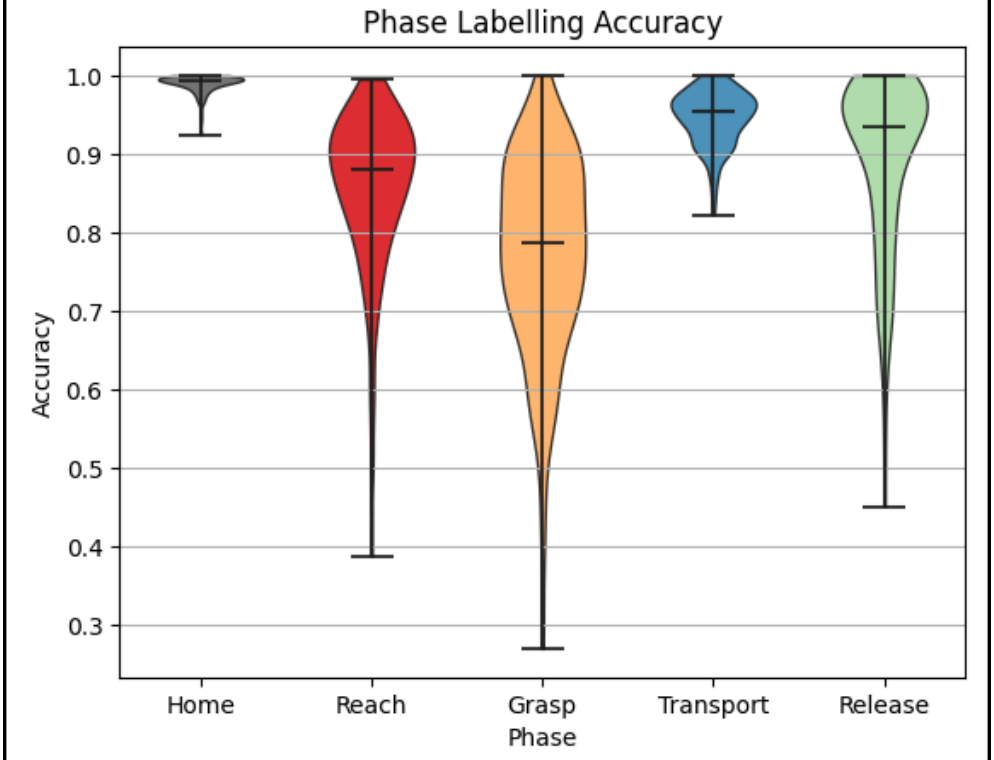


Figure 4: LSTM-1-NP Phase Labeling Accuracy. Accuracy range for each phase is indicated by the vertical and horizontal black lines. Accuracy distribution for each phase is visualized by shaded regions.

top row of each plot shows the true phase breakdown over the trial and the bottom row shows the corresponding labels as predicted by LSTM-1-NP, with the x-axis indicating the sequential ordering of frames throughout each trial. Each color bar corresponds to a movement segment: gray for “home,” red for “reach,” orange for “grasp,” blue for “transport,” and green for “release” phases. Each phase’s corresponding accuracy is listed in the legend beside each plot. Across all trials, the “home” phase was the most accurately predicted, followed by “release” and “transport” phases, which were comparatively similar. The “grasp” and “reach” phases had the worst accuracy overall.

The per-phase accuracy across all test trials for LSTM-1-NP (fold 3) was calculated and visualized in Figure 4: LSTM-1-NP Phase Labeling Accuracy. The median accuracy and range are indicated by the vertical and horizontal black lines in each violin plot. The distribution of the accuracies is shown by each violin’s shaded region. Similar to the individual trial survey, the “home” and “transport” phases proved to be the most accurate over all test trials. “Release” and

“reach” phase prediction accuracies have similar profiles, with the “release” phase accuracy distribution concentrating higher. The “grasp” phase again shows the worst performance across all trials, with the widest distribution and accuracy tailing sub 30%.

4.0 Discussion

The findings of this project confirm that supervised deep learning techniques can successfully be applied to the classification of motion data into composite movement phases and further exploration is warranted. Both architectures were able to achieve better-than-chance accuracy across all conditions. Initial analysis reveals that while some assumptions of the problem domain proved accurate, there are further avenues of research left to pursue.

As expected, the MLP architectures were more susceptible to overfitting than the LSTM architectures, with their training loss increasing and test accuracy decreasing after a single epoch in all configurations. In comparison, while training loss increased modestly between epochs for the LSTM models, test accuracy remained relatively stable or increased. The LSTM model’s architecture allows for it to capture temporal dependencies in a way that MLP architectures lack, potentially aiding in their ability to better model the sequential ordering of movement phases.

The most striking finding from the results was the similarity in accuracy between LSTM and MLP architectures. Although the LSTM architectures demonstrated superior overall accuracy and were less prone to overfitting and loss explosion, the results indicate MLP architectures may prove sufficient for future applications where reducing training time is a priority. Coupled with minimal accuracy improvements for the LSTM models over multiple epochs and the lack of benefit from including previous sample prediction or longer context windows, these results together suggest that treating movement phase classification as a time series challenge may not be necessary. Rather, the phase classification problem appears to be more Markov in nature than anticipated, relying primarily on information encoded in the current state over previous ones.

The range in performance of LSTM-1-NP on individual trials provides insight into what improvements may be necessary before rolling a model like this out to production, even without further refinement. The highest accuracy sample trial, Figure 3: LSTM-1-NP Best, Median, and Worst Performance - A, demonstrates near-perfect performance. The only errors present could be

mitigated by layering a simple algorithm on top of the model to ensure any phase transitions followed the ordinal phase sequence (i.e. ensure a transition to the "reach" phase only occurs after the "home" phase). The median accuracy trial, Figure 3: LSTM-1-NP Best, Median, and Worst Performance - B, would require the above measure, as well as the application of a filter to address short phase "blips." These processing techniques parallel mitigations that the manual segmentation policy already employs. The worst accuracy trial, Figure 3: LSTM-1-NP Best, Median, and Worst Performance - C, has characteristics that would be significantly more difficult to mitigate and would be unreliably addressed in a near-real-time scenario. Further investigation should be done to uncover what facets of this trial, and others like it, lead to poor prediction accuracy and how to mitigate or at minimum flag them.

Turning to the predicted phase accuracy across all test trials shown in Figure 4: LSTM-1-NP Phase Labeling Accuracy results largely mirror the trends identified in the select trial performance plots. While high accuracy for the "home" and "transport" phases is to be expected, the comparatively poor performance of the "reach" phase is not. "Reach" onset in the rules-based segmentation is triggered by the velocity of the right hand object exceeding 5% of the hand object's maximum velocity for a trial. Looking at the worst trial performance, the predicted onset begins much later for each "reach" than the true onset. It is possible that the worst performance trial is from a participant who deviated from their baseline performance, or substantially differed in behavior from other participants, though further analysis will be required for confirmation. The "grasp" phase also exhibited performance that would be unusable in a deployed setting. This finding is not surprising given the "grasp" phase is the shortest phase in a trial and accordingly provides the least amount of training data relative to other phases (see Table 1: True Phase Proportions across Trials).

One avenue of potentially improving overall performance would be to collapse the "grasp" and "release" phases into combined phases with "reach" and "transport," respectively. The combined phases would have comparatively strong signals for both onset and offset (hand object and pasta box object velocities, respectively). This would hamper the granularity of insight from analyzing this movement task specifically but may prove sufficient for other tasks.

Another potential improvement would be to treat each repeated phase as a separate segment class, i.e. training on "reach 1," "reach 2," and "reach 3," rather than encoding a single reach class. This approach better approximates the strategy employed by the manually defined

segmentation strategy–the specific dynamics of each phase are different enough that they may add more noise than useful signal when conflated together. For example, every "reach" phase ends with the hand object in a completely different orientation, as is the case for every phase the participant is completing. A potentially significant tradeoff of this approach is that by separately labeling each sub-phase, the amount of training data will be insufficient to allow for adequate learning to occur.

A final opportunity for boosting model performance will be to engineer novel features in the data set. The existing training features were already part of the GaMA data processing pipeline and were not created with machine learning applications in mind. Some potential features to improve learning performance include the number of frames/time since the last transition, the next possible phase class (i.e. grasp after reach, etc), and the present movement stage (i.e. "movement 1," "movement 2," or "movement 3").

Aside from procedural modifications and increases in the number of model layers/parameters, the observation that this problem domain may not be best solved as a time series problem suggests alternative machine learning-based approaches may lead to even better results. Despite the age of the approach, gradient-boosted decision trees, for example, have remained immensely powerful tools for various classification problems (Bentéjac et al., 2021). Re-running the data using XGboost (Chen et al., 2016) and comparing its performance is a natural follow-up experiment to this project.

5.0 Conclusions

The primary objective of this project was to explore the feasibility of leveraging deep machine learning techniques to classify motion capture data by movement phase. It made use of a large, multi-site data set labeled and analyzed using the GaMA research software platform. The LSTM architecture was selected as the prime model candidate due to its efficacy at processing time series data, with standard MLP architectures used for comparison. In addition to architecture choice, two additional test conditions were explored, context window size (60 frames vs. a single frame) and inclusion of previous sample phase predictions.

The LSTM architecture proved successful at classifying movement phases across all conditions, outperforming each corollary MLP architecture. Performance was significantly better than chance, with the best model achieving an average peak accuracy of 93%.

Several options for further research and possible improvements to boost model performance were presented in the discussion section above, but more work should be done to understand the limitations of the architectures tested in this project. All architectures used default parameters and activation functions, alternative context windows should be tested, and a comprehensive hyper-parameter search (including testing of alternative activation and loss functions) should at minimum be performed to optimize the best-performing model, LSTM-1-NP. All available data was also used for training, but given only marginal improvements in test accuracy were observed over repeated epochs it is possible that smaller data sets can yield equivalent results with shorter training time. This set could be minimized by either including fewer trials, and/or explicitly removing duplicate or superfluous sample frames. Smaller training sets would also inform whether or not a production roll-out of the model is feasible–there is no point in trialing this approach if the time required to manually label an adequately sized test set exceeds the time it takes to manually define and test explicit segmentation rulesets. Additionally, the splits used for cross-validation were balanced to ensure that ~80% of every participant's trials were included in each training fold. There is accordingly little confidence in the ability of a model to accurately generalize to novel participant data. Alternative training splits that purposefully leave out participants should be generated and trained on to address this limitation.

The promising results of this project reveal the significant potential for classifying gaze and motion capture data by movement phase, through the use of deep learning techniques–LSTM architectures in particular. Successful applications of this approach could greatly benefit research in fields where accurate movement analysis is critical, such as sports science, physical therapy, and human factors engineering. Compute resource and data collection requirements for model training must of course remain a consideration, and may preclude this approach in some domains. Future work should also investigate the model's performance with minimal data sets, and its ability to generalize across different types of movements and individuals to define its ultimate utility. This project has laid a firm foundation for these future investigations and will

optimistically aid in harnessing the power of machine learning to unlock real-time movement analysis.

6.0 References

Adesida, Y., Papi, E., & McGregor, A. (2019, April 2). *Exploring the Role of Wearable Technology in Sport Kinematics and Kinetics: A Systematic Review*. MDPI. Retrieved October 14, 2023, from https://www.mdpi.com/1424-8220/19/7/1597

Barto, A. (1998). Finite Markov Decision Processes. In R. S. Sutton (Ed.), *Reinforcement Learning: An Introduction* (p. 47). MIT Press. http://incompleteideas.net/book/RLbook2020.pdf

Bentéjac, C., Csörgő, A., & Martínez-Muñoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, *54*, 1937-1967.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785

Cronin, N., Walker, J., Tucker, C., Nicholson, G., Cooke, M., Merlino, S., & Bissas, A. (2023). Feasibility of OpenPose markerless motion analysis in a real athletics competition. *Frontiers in Sports and Active Living*, *5*. https://doi.org/10.3389/fspor.2023.1298003

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., & van Zee, M. (2023). *Flax: A neural network library and ecosystem for JAX*. Github. http://github.com/google/flax

Hochreiter, S., & Schmidhuber, J. (1997, November 15). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735-1780. https://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Term-Memory?redirectedFrom=fulltext

Jin, M., Wen, Q., Liang, Y., Zhang, C., Xue, S., Wang, X., Zhang, J., Wang, Y., Chen, H., Li, X., Pan, S., Tseng, V., Zheng, Y., Chen, L., & Xiong, H. (2023, Pre-print). *Large Models for Time Series and Spatio-Temporal Data: A Survey and Outlook*. arXiv. Retrieved January 14, 2024, from https://arxiv.org/pdf/2310.10196.pdf

Kader, N., Yusof, U., Khalid, M., & Husain, N. (2022). A Review of Long Short-Term Memory Approach for Time Series Analysis and Forecasting. *Lecture Notes in Networks and Systems*, *573*. https://doi.org/10.1007/978-3-031-20429-6_2

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, *25*, 1097-1105. https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

Nunes, J., Moreira, P., & Tavares, J. (2015). Human Motion Analysis and Simulation Tools: A Survey. In F. Miranda & C. Abreu (Eds.), *Handbook of Research on Computational Simulation and Modeling in Engineering* (pp. 359-389). Engineering Science Reference. 10.4018/978-1-4666-8823-0

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-Resolution Image Synthesis with Latent Diffusion Models. *Conference on Computer Vision and Pattern Recognition*, *2022*. https://arxiv.org/abs/2112.10752

Rueckert, D., & Fichtinger, G. (2019). Deep learning: RNNs and LSTM. In S. K. Zhou (Ed.), *Handbook of Medical Image Computing and Computer Assisted Intervention* (pp. 503-519). Elsevier Science. https://www.sciencedirect.com/science/article/pii/B9780128161760000260

Sarker, I. (2021, August 18). Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Computer Science*, *2*(6), 420. 10.1007/s42979-021-00815-1

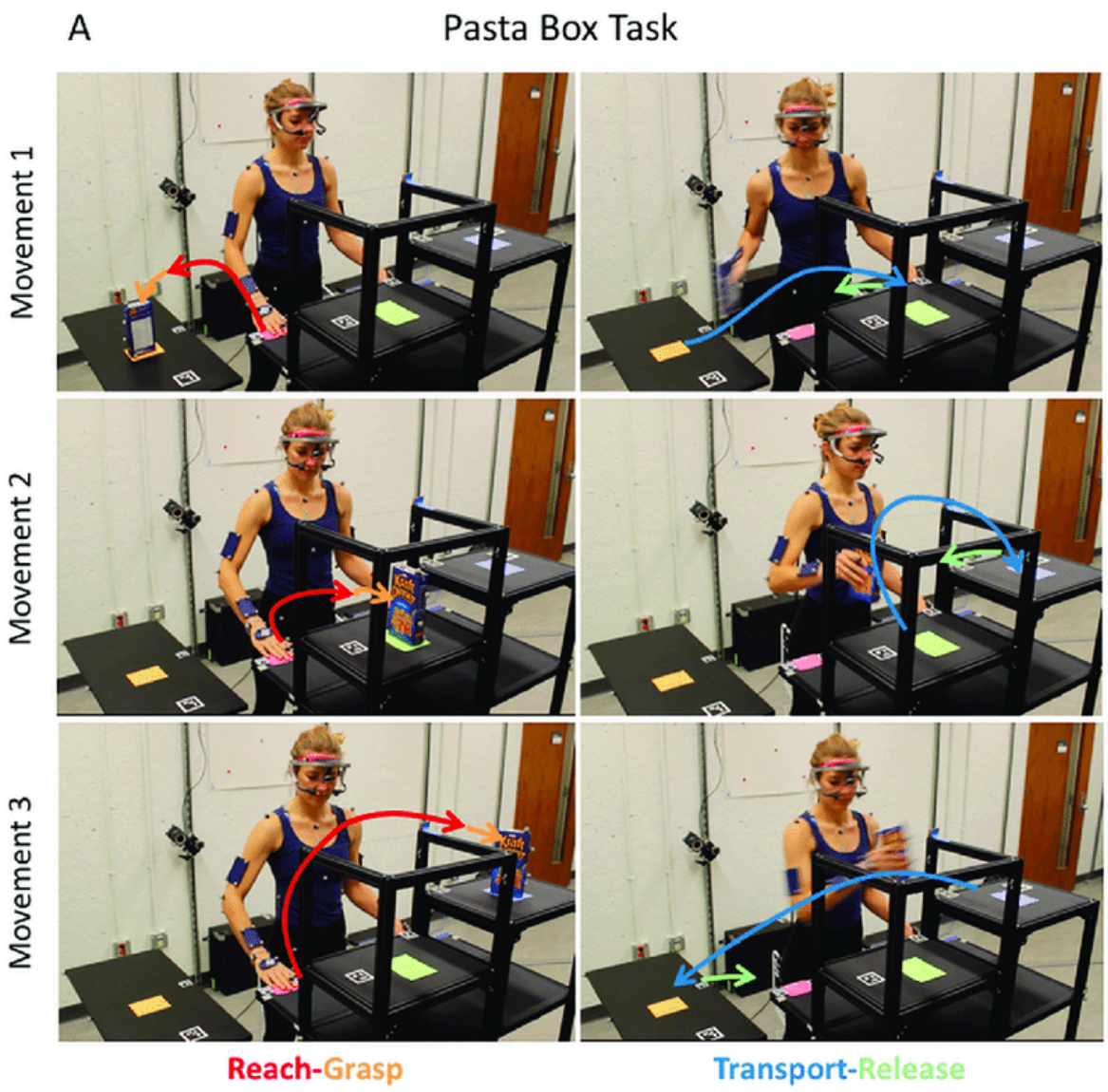
Valevicius, A. M., Boser, Q. A., Lavoie, E. B., Murgatroyd, G. S., Pilarski, P. M., Chapman, C. S., Vette, A. H., & Hebert, J. S. (2018). Characterization of normative hand movements during two functional upper limb tasks. *PloS one*, *13*(6). https://doi.org/10.1371/journal.pone.0199549

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems*, *30*. https://papers.nips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Williams, H., Chapman, C., Pilarski, P., Vette, A., & Hebert, J. (2019, December 30). Gaze and Movement Assessment (GaMA): Inter-site validation of a visuomotor upper limb functional protocol. *PLoS One*, *14*(12). 10.1371/journal.pone.0219333

Zhang, A., Lipton, Z., Mu, L., & Smola, A. (2023). *Dive into Deep Learning*. Cambridge University Press. https://D2L.ai

Appendix A: Pasta Box Task Apparatus and Sequence



Pasta Box Task Apparatus and procedure demonstration,
courtesy of Valevicius et al., 2018

Appendix B: Included GaMA Features

Field Name	Field Type	Description
D:GripAperture:Magnitude	Distance	Distance between the right index fingertip marker and the right thumb tip marker.
D:HandDistancefromHome:Magnitude	Distance	Distance between the right hand and “Home” target object centroids.
D:HandDistancefromPastaBox:Magnitude	Distance	Distance between the right hand and the Pasta Box object centroids.
D:PastaDistancefromStartAOI:Magnitude	Distance	Distance between the Pasta Box and “Home” target object centroids
D:PastaDistancefromShelf1AOI:Magnitude	Distance	Distance between the Pasta Box and the first drop-off location object centroids.
D:PastaDistancefromShelf2AOI:Magnitude	Distance	Distance between the Pasta Box and second drop-off location object centroids.
D:HandDistance_fromReleaseOnset1:Magnitude	Distance	Distance between the right hand and first drop-off location object centroids.
D:HandDistance_fromReleaseOnset2:Magnitude	Distance	Distance between the right hand and second drop-off location object centroids.
D:HandDistance_fromReleaseOnset3:Magnitude	Distance	Distance between the right hand and third drop-off location object centroids.
D:HandDistance_fromTransportOnset1:Magnitude	Distance	Distance between the right hand and first transport start location object centroids.
D:HandDistance_fromTransportOnset2:Magnitude	Distance	Distance between the right hand and second transport start location object centroids.
D:HandDistance_fromTransportOnset3:Magnitude	Distance	Distance between the right hand and third transport start location object centroids.
D:BestGazeVector_to_HandObj_Magnitude:Magnitude	Distance	Shortest distance between the highest accuracy gaze vector to the right hand object centroid.
D:BestGazeVector_to_NeutralObj_Magnitude:Magnitude	Distance	The shortest distance between the highest accuracy gaze vector to the neutral marker.

D:BestGazeVector_to_PastaBoxObj_Magnitude:Magnitude	Distance	The shortest distance between the highest accuracy gaze vector to the pasta box object centroid.
D:BestGazeVector_to_HomeAOIObj_Magnitude:Magnitude	Distance	The shortest distance between the highest accuracy gaze vector to the home target object centroid.
D:BestGazeVector_to_Shelf1TargetAOI_Magnitude:Magnitude	Distance	The shortest distance between the highest accuracy gaze vector to the first shelf target object centroid.
D:BestGazeVector_to_Shelf2TargetAOI_Magnitude:Magnitude	Distance	The shortest distance between the highest accuracy gaze vector to the second shelf target object centroid.
D:BestGazeVector_to_PastaStartAOI_Magnitude:Magnitude	Distance	The shortest distance between the highest accuracy gaze vector to the initial pickup target object centroid.
DV:PastaVelocity:Magnitude	Derivative	Velocity of the pasta box object.
DV:HandVelocity:Magnitude	Derivative	Velocity of the right hand object.
DV:HandAcceleration:Magnitude	Derivative	Acceleration of the right hand object.
DV:AngularVelocity__TrunkFE:Magnitude	Derivative	Angular velocity of the trunk rigid body about the frontal axis.
DV:AngularVelocity__TrunkBend:Magnitude	Derivative	Angular velocity of the trunk rigid body about the sagittal axis.
DV:AngularVelocity__TrunkRot:Magnitude	Derivative	Angular velocity of the trunk rigid body about the vertical axis.
DV:AngularVelocity__ShoulderFE:Magnitude	Derivative	Angular velocity of the right shoulder flexion.
DV:AngularVelocity__ShoulderAbAd:Magnitude	Derivative	Angular velocity of the right shoulder abduction.
DV:AngularVelocity__ShoulderRot:Magnitude	Derivative	Angular velocity of the right shoulder rotation.
DV:AngularVelocity__ElbowFE:Magnitude	Derivative	Angular velocity of the right elbow flexion.
DV:AngularVelocity__WristFE:Magnitude	Derivative	Angular velocity of the right wrist flexion.
DV:AngularVelocity__WristDev:Magnitude	Derivative	Angular velocity of the right wrist deviation.
DV:AngularVelocity__ProSup:Magnitude	Derivative	Angular velocity of the right wrist pronation.

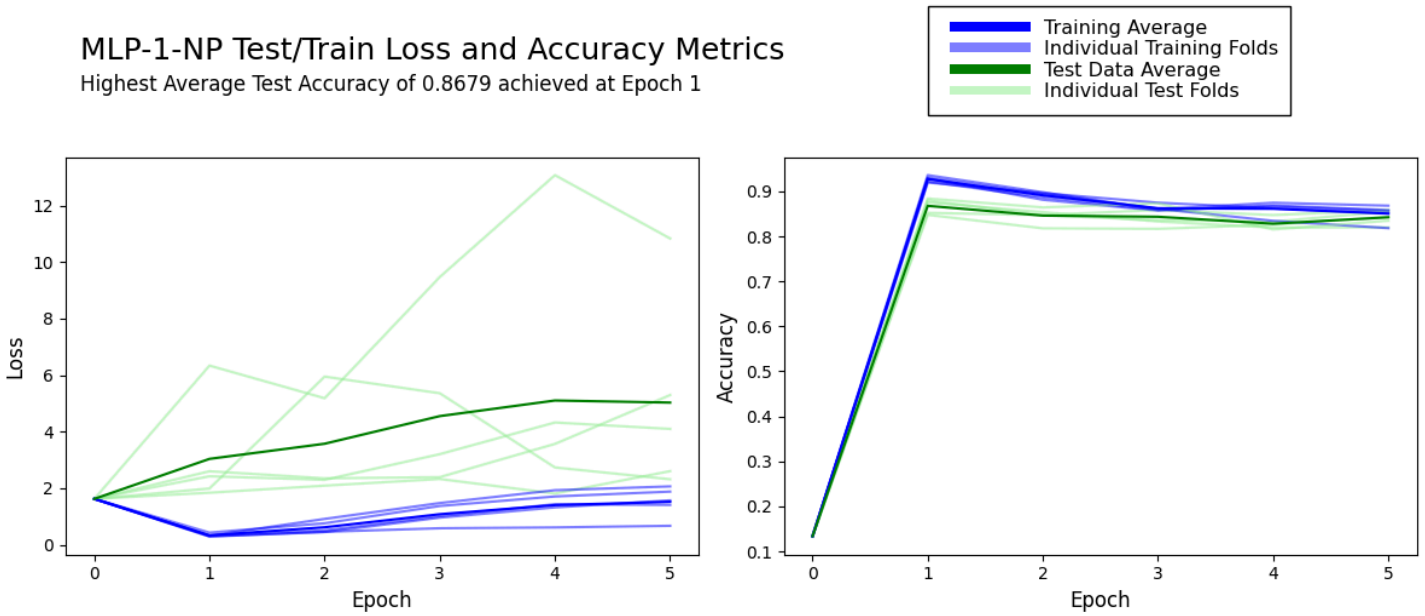
A:Cluster_Elbow__FlexExt:Rotation1D	Angle	Angle of the right elbow flexion.
A:Cluster_Elbow__Y:Rotation1D	Angle	The rotation angle of the right elbow in the Y axis, the corollary of A:Cluster_Elbow__Z:Rotation1D
A:Cluster_Elbow__Z:Rotation1D	Angle	The rotation angle of the right elbow in the Z axis, the corollary of A:Cluster_Elbow__Y:Rotation1D
A:Cluster_Wrist__FlexExt:Rotation1D	Angle	Angle of the right wrist flexion.
A:Cluster_Wrist__UlnRadDev:Rotation1D	Angle	Angle of the right wrist radial deviation.
A:Cluster_Wrist__Z:Rotation1D	Angle	Angle of the right wrist rotation about the Z axis.
A:Cluster_Frm__ProSup:Rotation1D	Angle	Angle of the right forearm pronation.

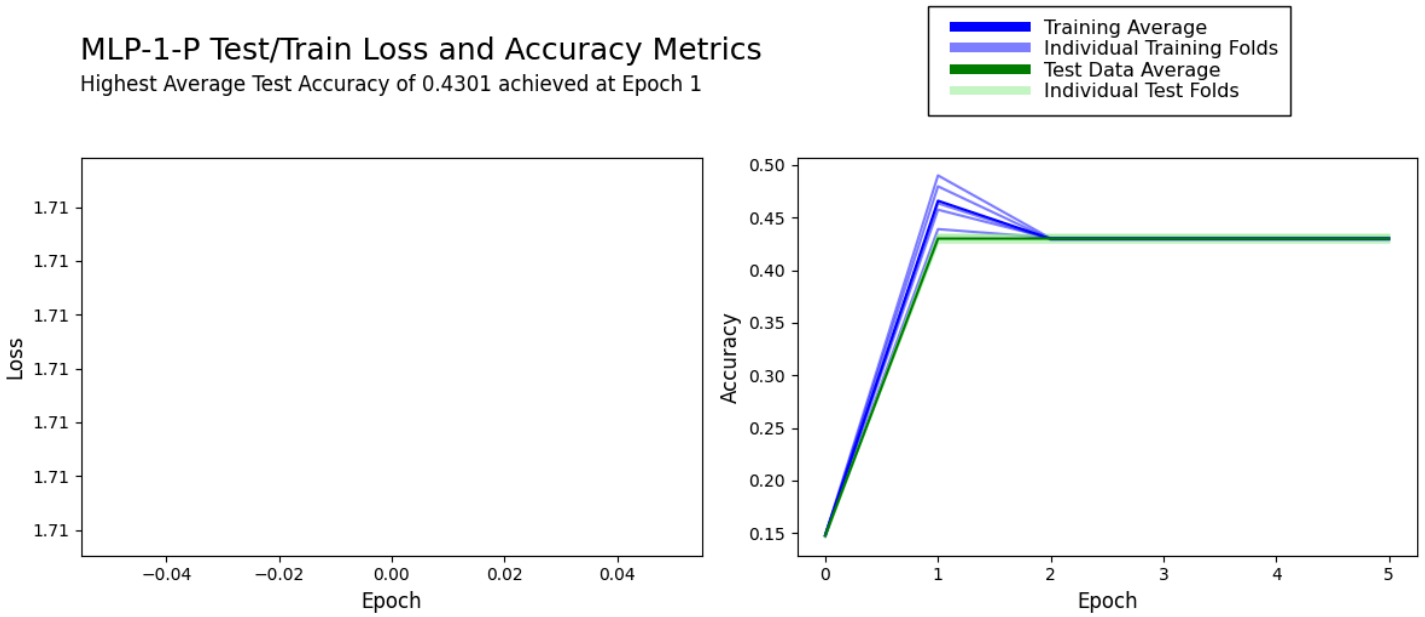
Appendix B: Software Tools

Note: Dependencies of each software library are not included in this listing.

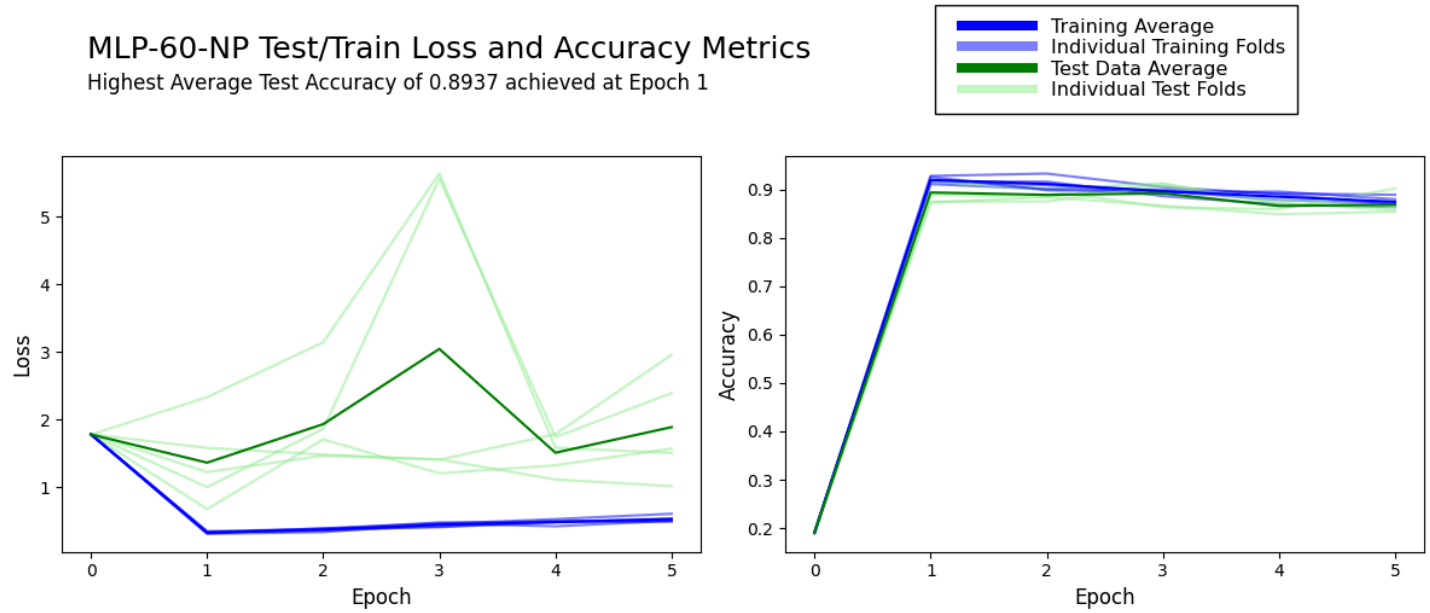
Software Name	Version(s) / Release	Usage/Description
Matlab	2020b(1), 2021a(2)	Engineering-focused programming language and environment. Used for 1 - The GaMA Software platform 2 - Initial pre-processing of data into terse CSVs
Python	3.10.12	Secondary data pre-processing: ● Normalization ● Standardization ● Sequence splitting ● One-hot encoding of labels) ● Model training and evaluation
Google Colab Environment	2024-02-21	Primary Python development environment
Jax	0.4.23	Python number library modeled after Numpy, optimized for machine learning applications. Used for data manipulation and training
Flax	0.8.0	Neural Net API built in Jax, used for model definition and training.
Optax	0.1.9	Used for optimizing the training learning rate for each model.
Matplotlib	3.7.1	Graphical plotting library for Python, used for result plots and visualizations

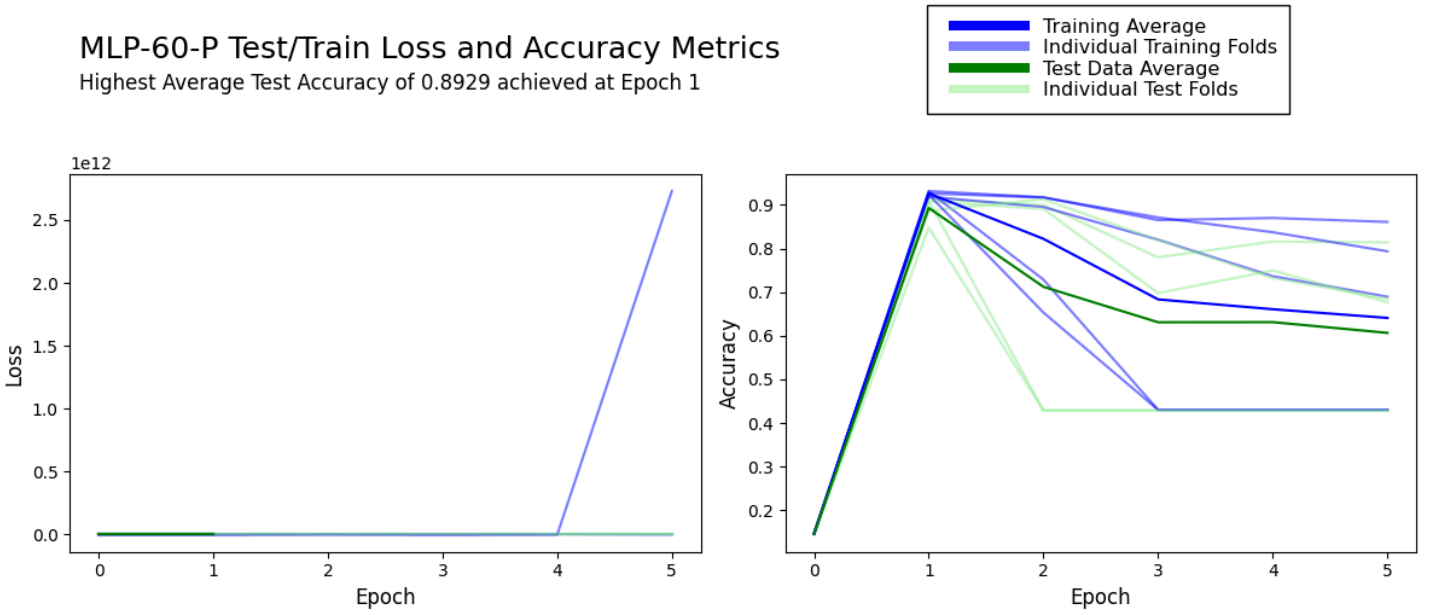
Appendix D: Test/Train Performance

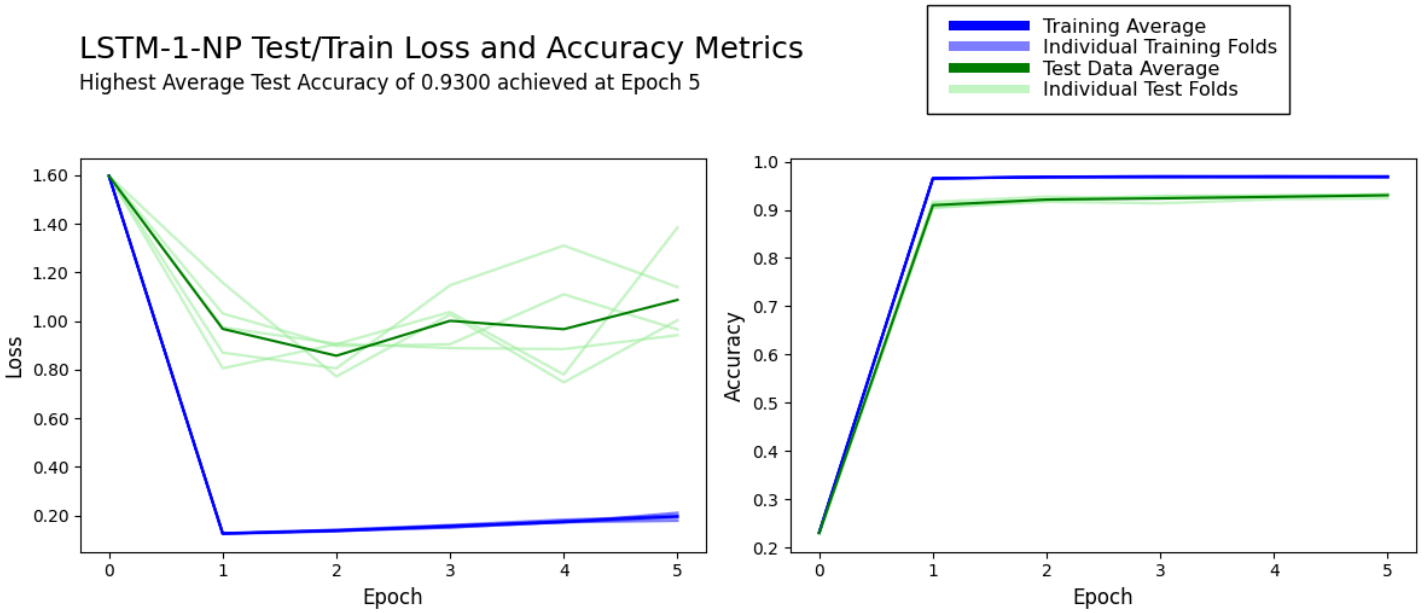


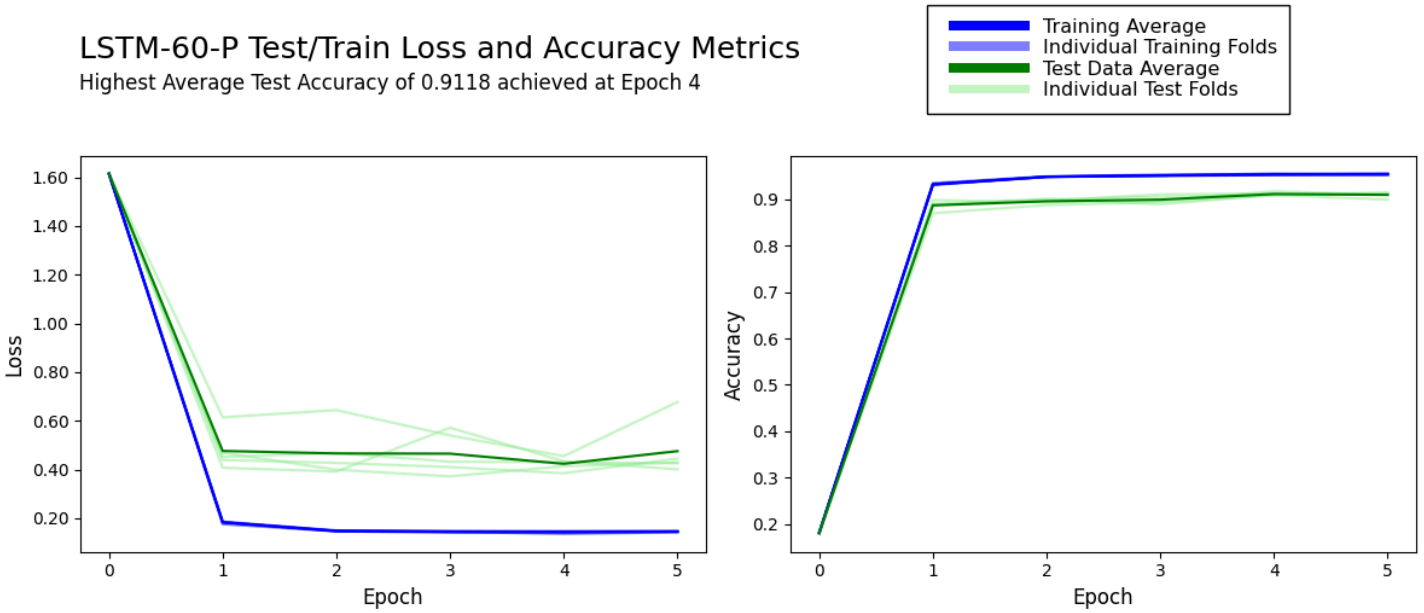


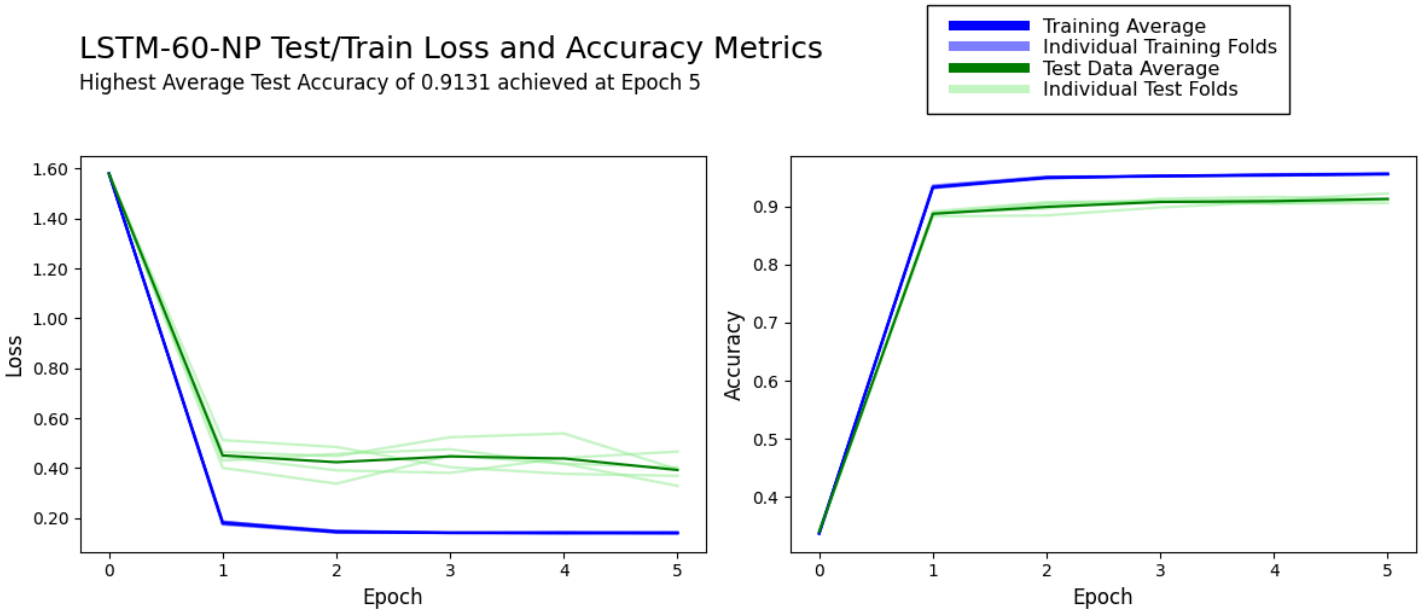
Note: Test/Train loss for MLP-1-P immediately exploded to NaN values within the first training epoch. The average pre-training test and training loss was 1.71.

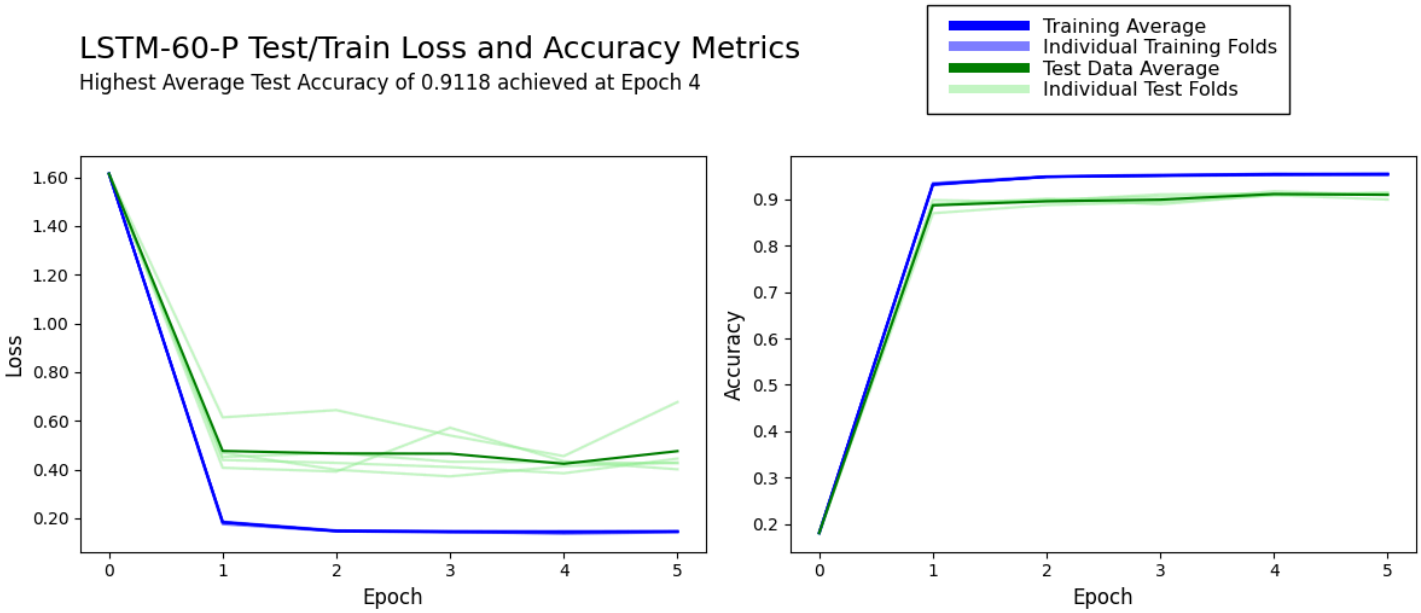

MLP-60-NP Test/Train Loss and Accuracy Metrics
Highest Average Test Accuracy of 0.8937 achieved at Epoch 1
Training Average
Individual Training Folds
Test Data Average
Individual Test Folds
Loss
Accuracy
Epoch
Epoch


MLP-60-P Test/Train Loss and Accuracy Metrics
Highest Average Test Accuracy of 0.8929 achieved at Epoch 1
Training Average
Individual Training Folds
Test Data Average
Individual Test Folds
1e12
Loss
Accuracy
Epoch
Epoch


LSTM-1-NP Test/Train Loss and Accuracy Metrics
Highest Average Test Accuracy of 0.9300 achieved at Epoch 5
Training Average
Individual Training Folds
Test Data Average
Individual Test Folds
Loss
Accuracy
Epoch
Epoch


LSTM-60-P Test/Train Loss and Accuracy Metrics
Highest Average Test Accuracy of 0.9118 achieved at Epoch 4
Training Average
Individual Training Folds
Test Data Average
Individual Test Folds
Loss
Accuracy
Epoch
Epoch


LSTM-60-NP Test/Train Loss and Accuracy Metrics
Highest Average Test Accuracy of 0.9131 achieved at Epoch 5
Training Average
Individual Training Folds
Test Data Average
Individual Test Folds
Loss
Accuracy
Epoch
Epoch


LSTM-60-P Test/Train Loss and Accuracy Metrics
Highest Average Test Accuracy of 0.9118 achieved at Epoch 4
Training Average
Individual Training Folds
Test Data Average
Individual Test Folds
Loss
Accuracy
Epoch
Epoch