Ensemble Recurrent Neural Network(RNN) Deployment on Raspberry Pi for High-rate Dynamic Systems

Metrid Dorothy Akinyi Okumu

FAST FOUNDATIONS OF ADAPTIVE SYSTEMS AND TECHNOLOGIES

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

High-Rate SYSTEMS

Systems experiencing high-rate dynamics

- Accelerations higher than 100 g_n (g_n =9.81 m/s^2) in less than 1ms.

Characterized by

- Large uncertainties in external loading.
- High levels of non-stationarity and heavy disturbance.
- Generations of unmodeled dynamics from changes in mechanical configuration.

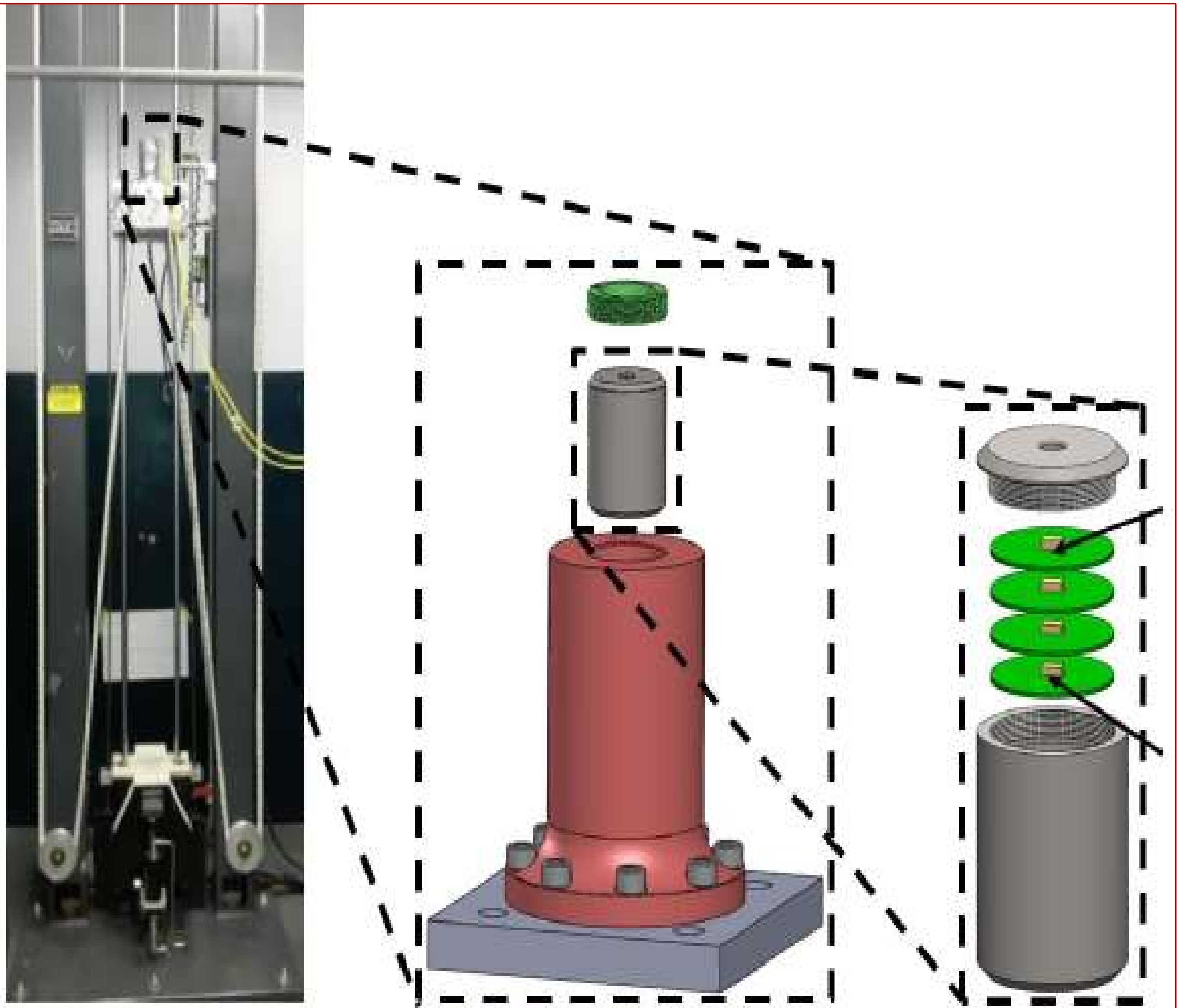
Challenges

- Unknown or uncertain dynamics.
- Real-time modeling requirements.
- Less than 100 μs computation time per decision step.

DATASET

High-rate laboratory dataset

- The data used in this algorithm is obtained from high-rate dynamic experiments conducted using a drop tower system.
- The dataset consists of acceleration and time measurements, capturing the response of a test specimen subjected to sudden impact.



RESEARCH OBJECTIVE

PROBLEM STATEMENT

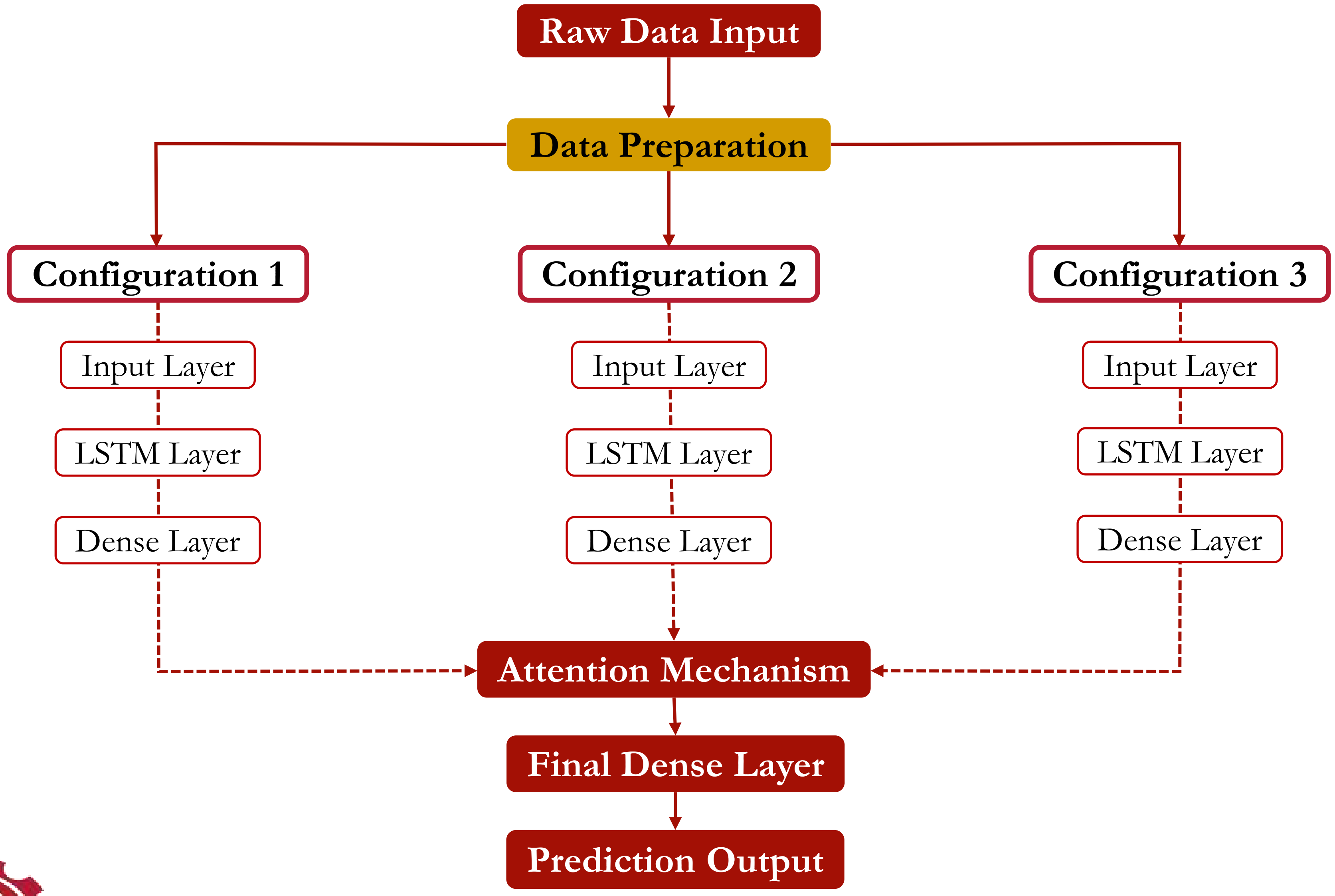
- We are developing and deploying a real-time, lightweight ensemble RNN model on a Raspberry Pi to forecast high-rate dynamic responses

WHAT ARE WE SOLVING

We are addressing the challenge of:

- Establishing a data pipeline
- Achieving sub-millisecond inference on a resource-constrained device-raspberry pi
- Capturing meaningful patterns in high-frequency data
- Understand and test the limits of the raspberry pi

RNN Architecture Workflow



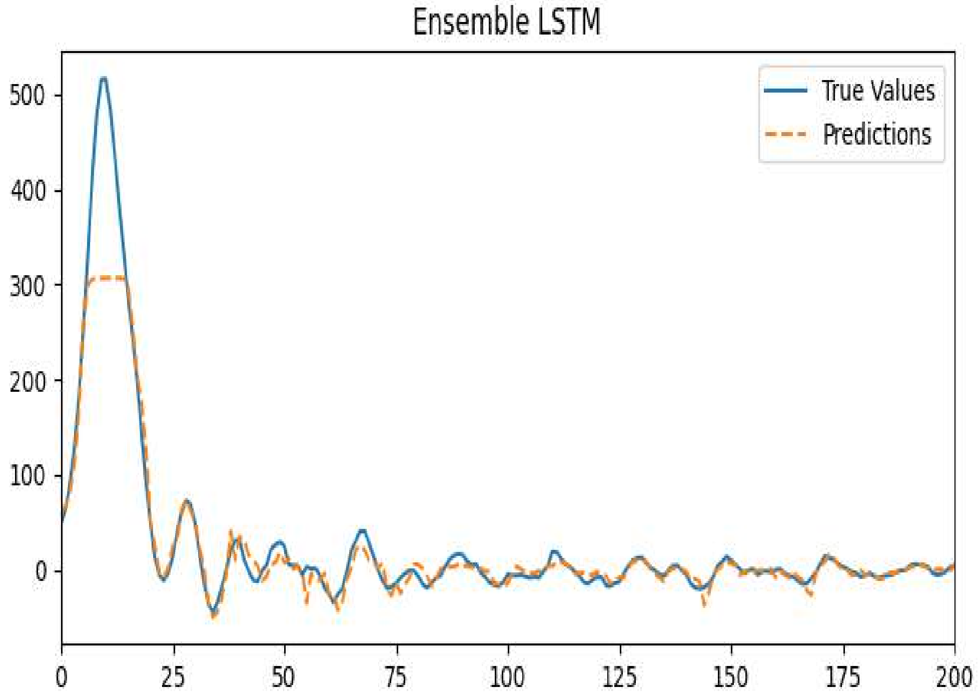
Prior work: Benchmark

Reference Benchmark:

- We benchmark our implementation against the work by Barzegar et al. (2022), which introduced an ensemble of RNNs with LSTM cells for high-rate structural health monitoring (HRSHM).
- Their system achieved:
    - 25 μs per timestep (inference time)
    - High accuracy on experimental drop tower data
    - Robust performance using multi-rate sampling and attention
- This benchmark serves as our performance target for real-time inference on edge devices like the Raspberry Pi.
- Optimal goal is < 100us , however < 1ms is acceptable.

INITIAL RESULTS

- Deployed the model on raspberry pi
- The total execution time as well as time per timestep was out of the threshold



Predicted vs Actual results from initial test

PERFORMANCE RESULTS

Metric	Local Machine	Raspberry pi
Prediction runtime	2.62 s	5.37s
Mean Absolute Error(MAE)	1.752	2.10
Mean Squared Error(MSE)	83.91	144.63
Root Mean Squared Error(RMSE)	9.16	12.02
R-squared (R^2)	0.94	0.90

RESULTS : TENSORFLOW-LITE (TFLITE) ON RASPBERRY PI

Tensor flow-lite ?

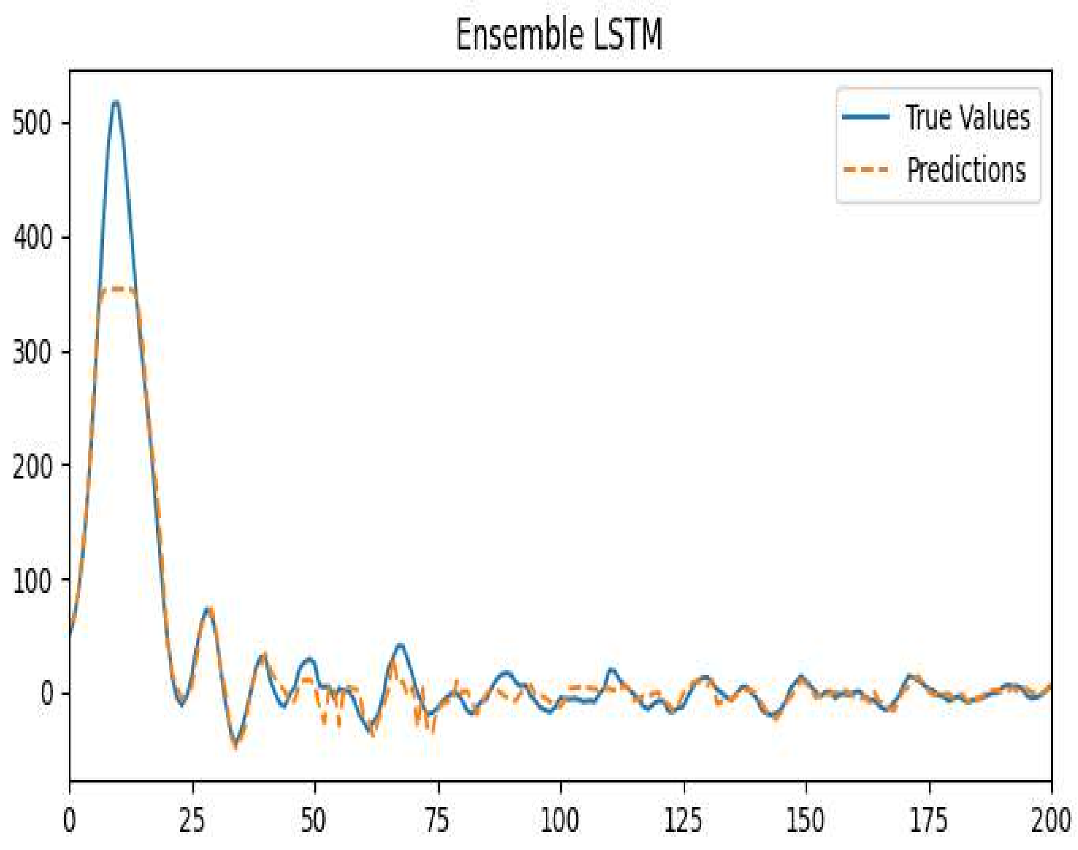
- A lightweight version of TensorFlow optimized for edge devices
- Designed for fast inference
- Reduce overall computational time

Workflow

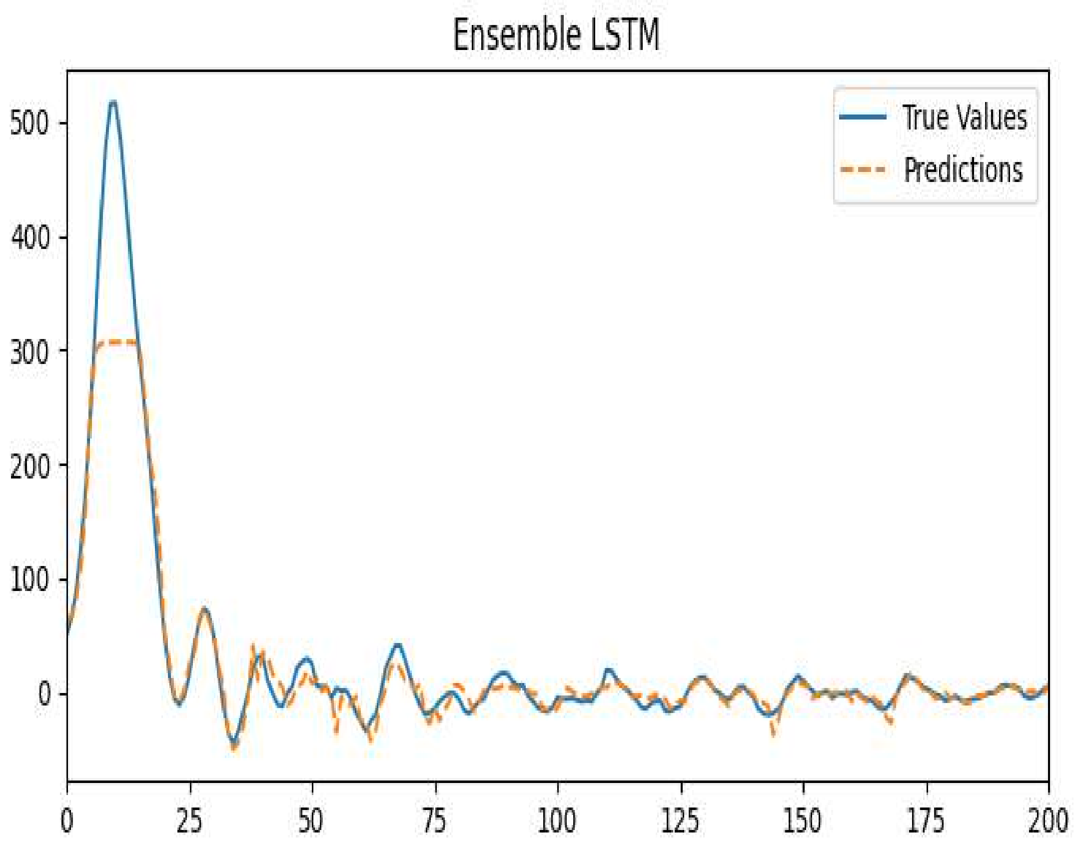
- Train the model on my laptop
- Converted model .h5 to .tflite
- Deploy on raspberry pi 4

Computational Time	
Tflite time per timestep	3.68 ms
Tflite total runtime	1.125s

VISUAL COMPARISON BETWEEN TFLITE AND TENSORFLOW

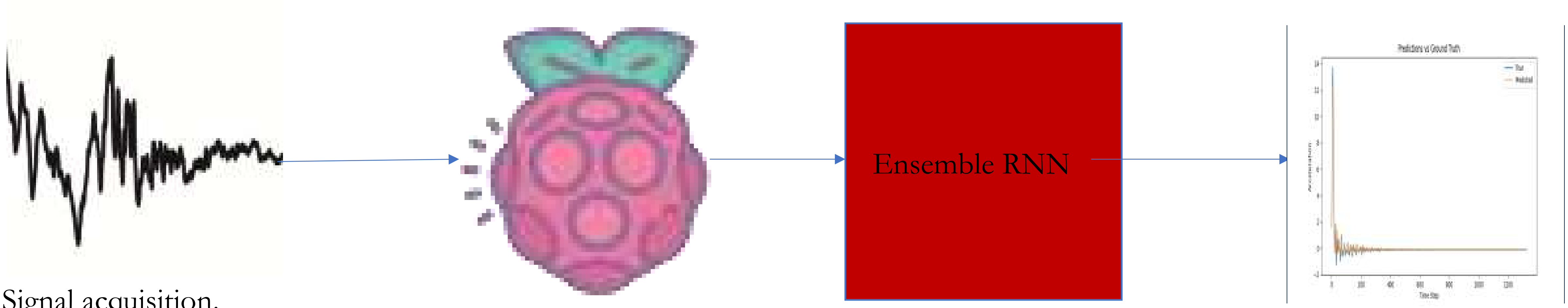


Predicted vs Actual results from tflite test



Predicted vs Actual results from initial test

SYSTSEM ARCHITECTURE : END-TO-END PIPELINE



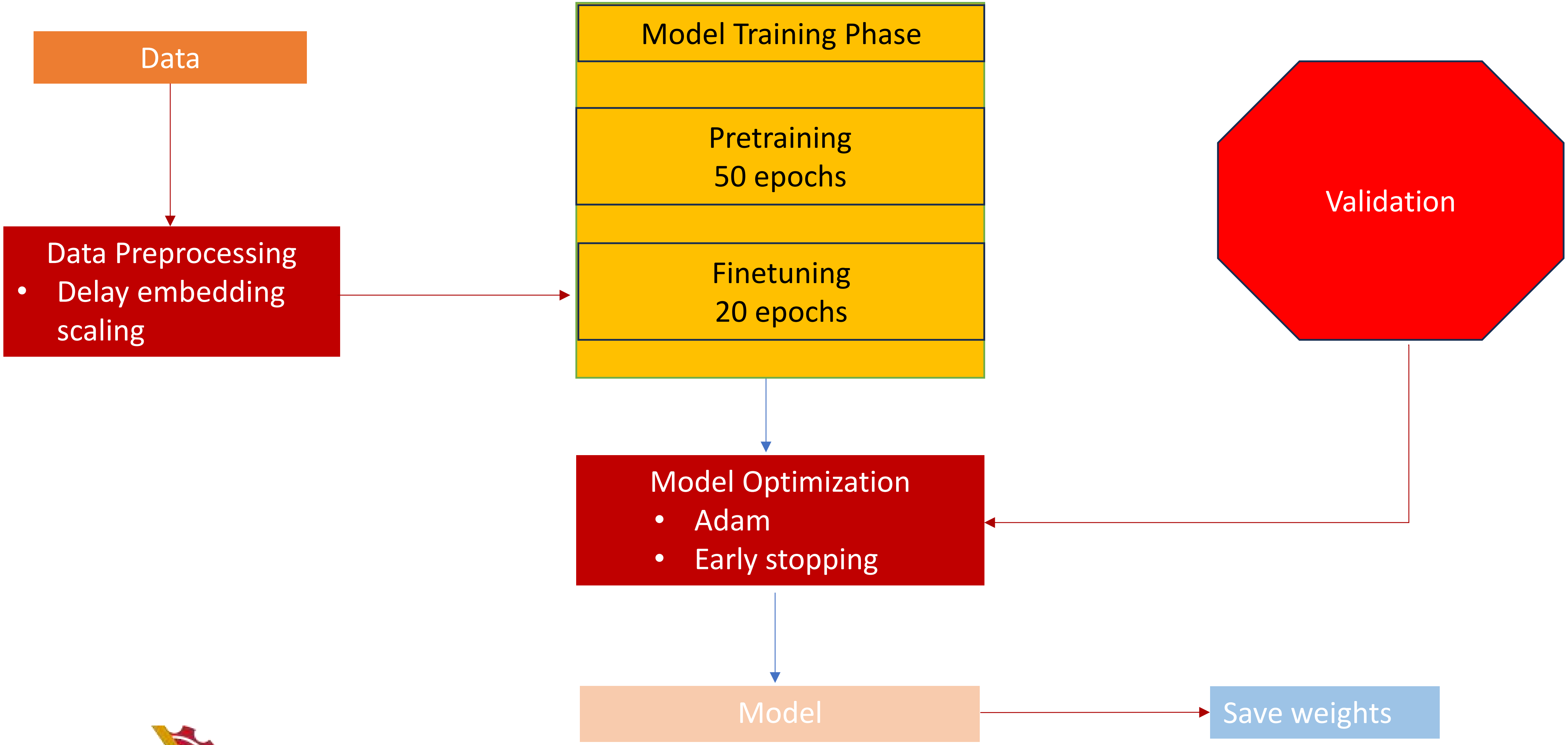
Signal acquisition.
Signal-ADC module- I2C -Pi.

Pre-processing

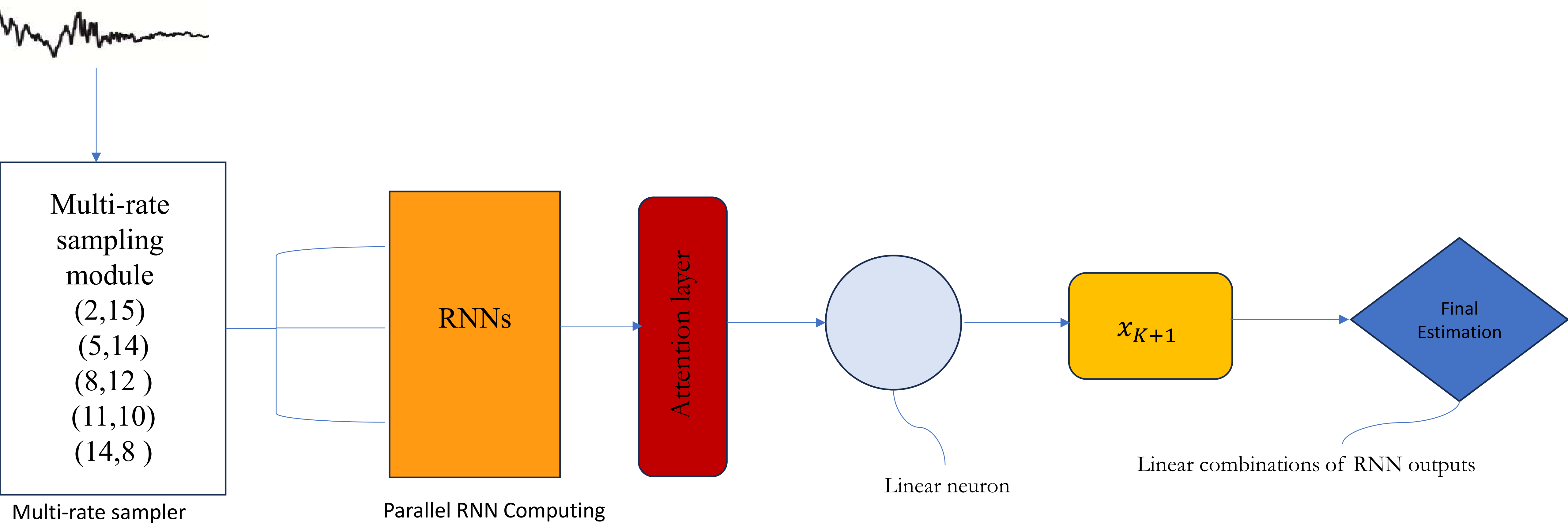
Inference

Output:Prediction

Training Pipeline

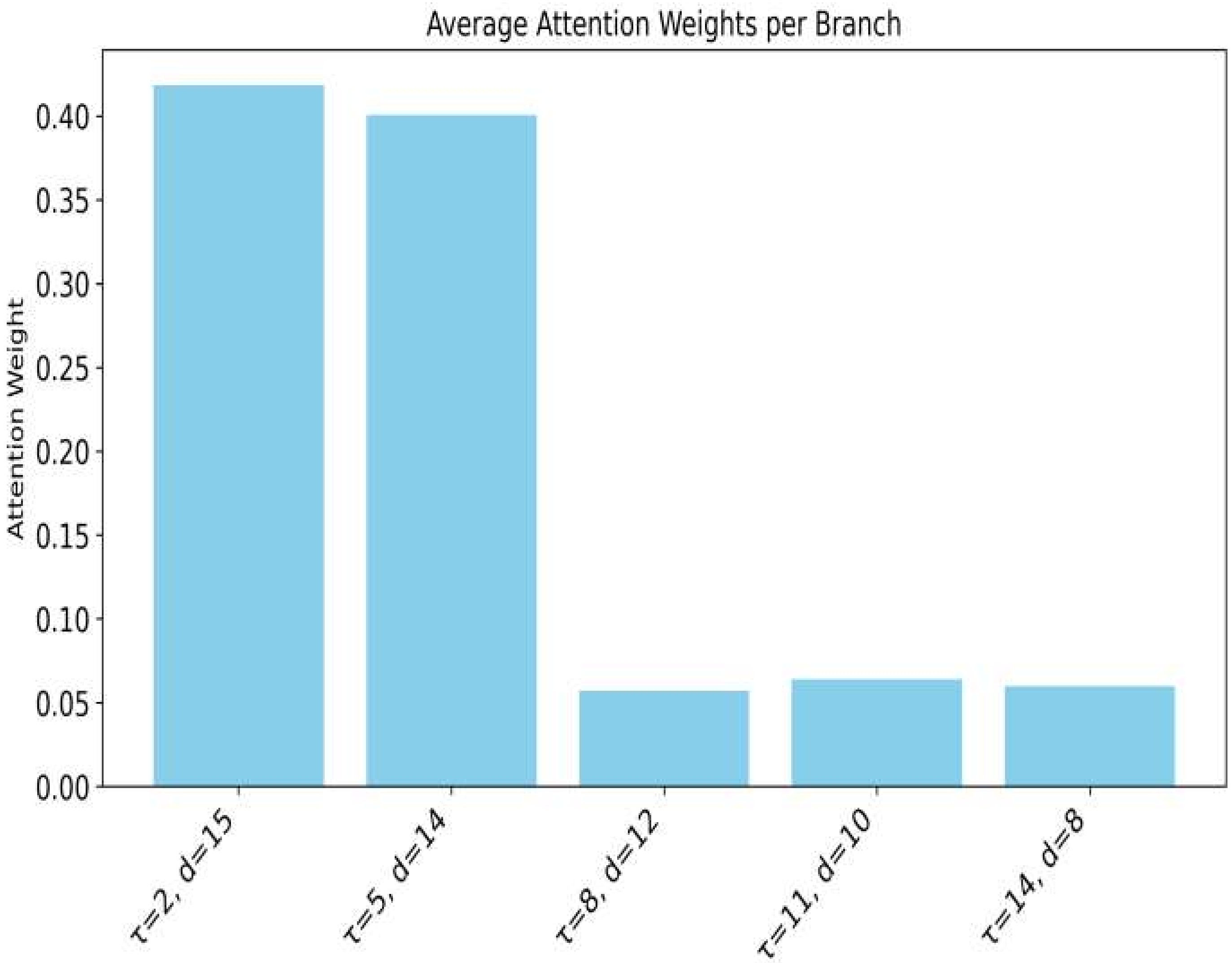


RNN Architecture



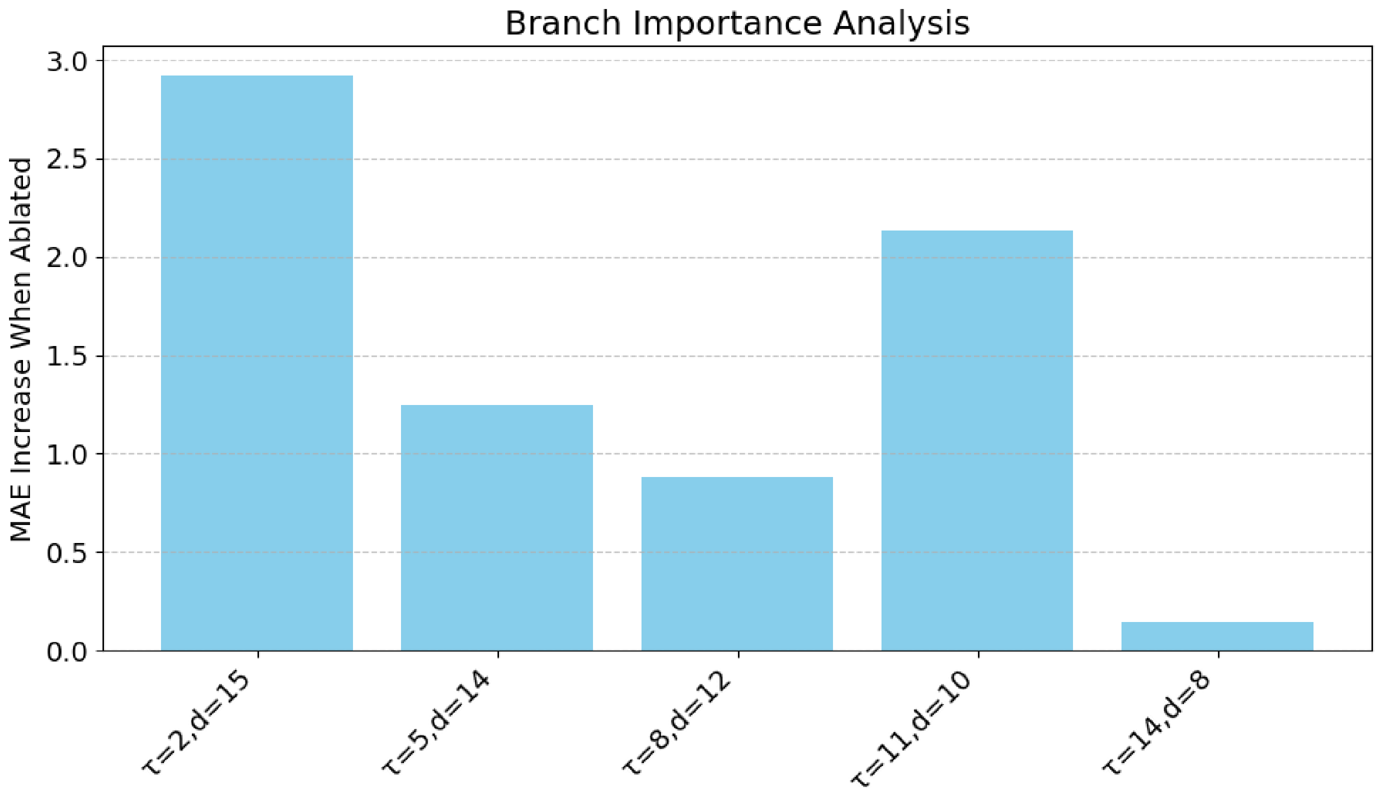
Understanding Attention Mechanism

- How much attention the model assigns to each input branch.
- High weights (e.g., τ=2, d=15 and τ=5, d=14) indicate that the model found these branches most informative for predicting the output.
- Lower weights (e.g., τ=8, d=12; τ=11, d=10; τ=14, d=8) suggest these inputs contributed less to the model's prediction on average.



UNDERSTANDING EACH INPUT BRANCH

- Understand the impact of each input branch in the algorithm
- Removing τ=2, d=15 causes the largest performance drop (↑MAE).
- τ=11, d=10 and τ=5, d=14 also have strong contributions.
- Branches with low attention τ=14, d=8 have minimal impact.



Results of branch analysis

LSTM FROM SCRATCH

Objective

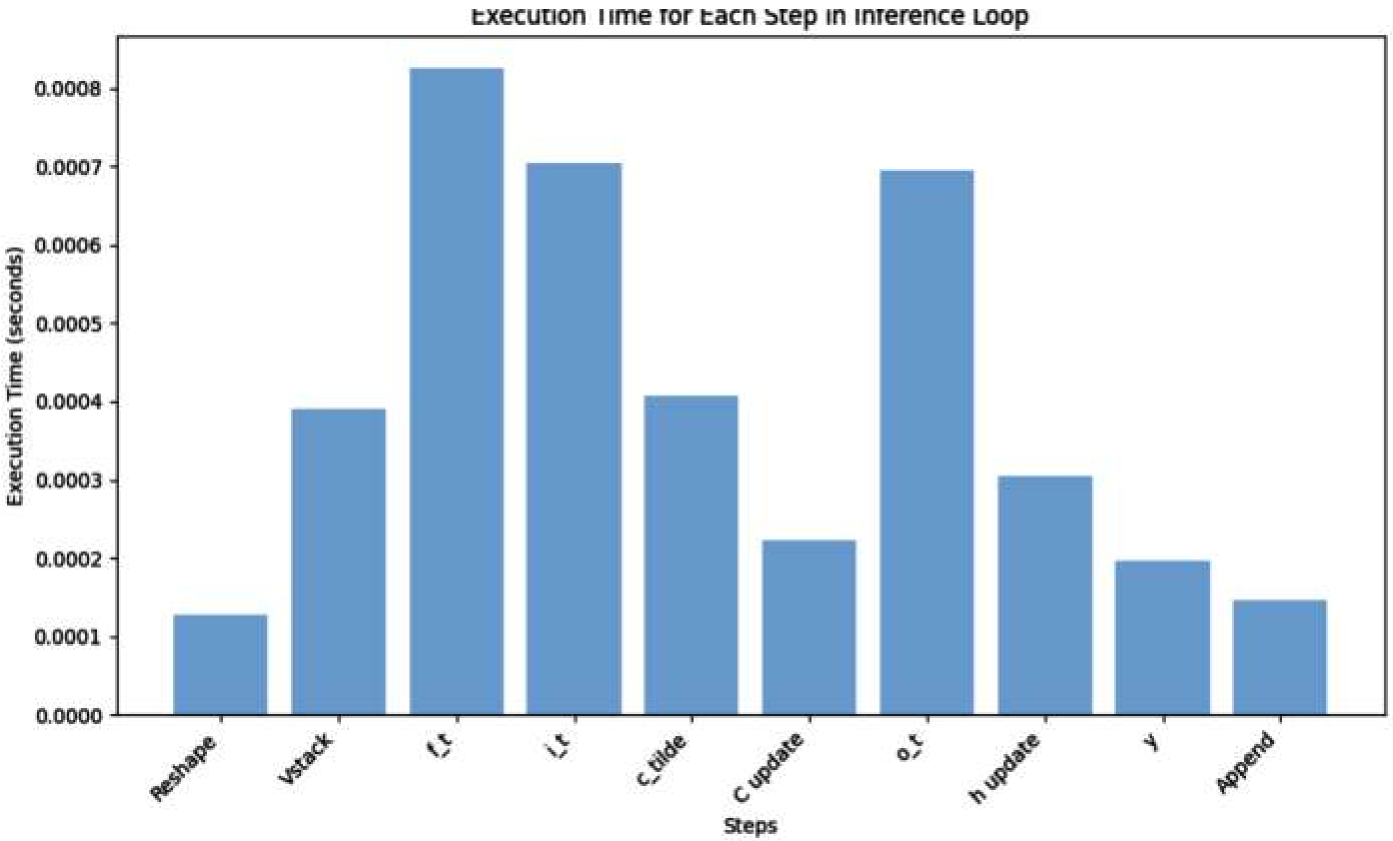
- Implemented a linear algebra-based model.
- Demonstrate a custom implementation of an LSTM using only NumPy.

Why This Matters:

•Understanding low-level details is crucial for debugging, optimizing, and extending LSTMs.

Key Questions:

•How does the LSTM work under the hood?
•What are the challenges of implementing it from scratch?
•How does this implementation compare to frameworks like TensorFlow ?



Results of different steps in the inference loop

Rank-Reduction Technique

Rank reduction Technique

- Involves approximating large LSTM weight matrices with lower-rank versions, using Singular Value Decomposition (SVD).
- It compresses the model by eliminating redundant or less significant weight components.
- Cuts down multiply-accumulate operations, reducing inference time significantly
- Maintains comparable accuracy.
- Reduced per-sample inference time.

How it works

- Standard LSTM layers contain large weight matrices:

$$W \in R^{(n_{in}+n_{hidden})\times 4n_{hidden}}$$

- Apply **SVD** to decompose W into U Σ V^T :

$$W = U\Sigma V^\top$$

- Keep only top-*r* singular values and vectors (low-rank approximation):

$$W \approx U_r \Sigma_r V_r^\top,$$

LSTM Branch	Original shape	Rank reduced	Parameter reduction
Branch 1	(31, 120)	27	10.0%
Branch 2	(29, 112)	26	7.9%
Branch 3	(25, 96)	22	9.3%
Branch 4	(21, 80)	18	11.1%
Branch 5	(17, 64)	15	9.0%

Result Overview : Rank Reduction technique

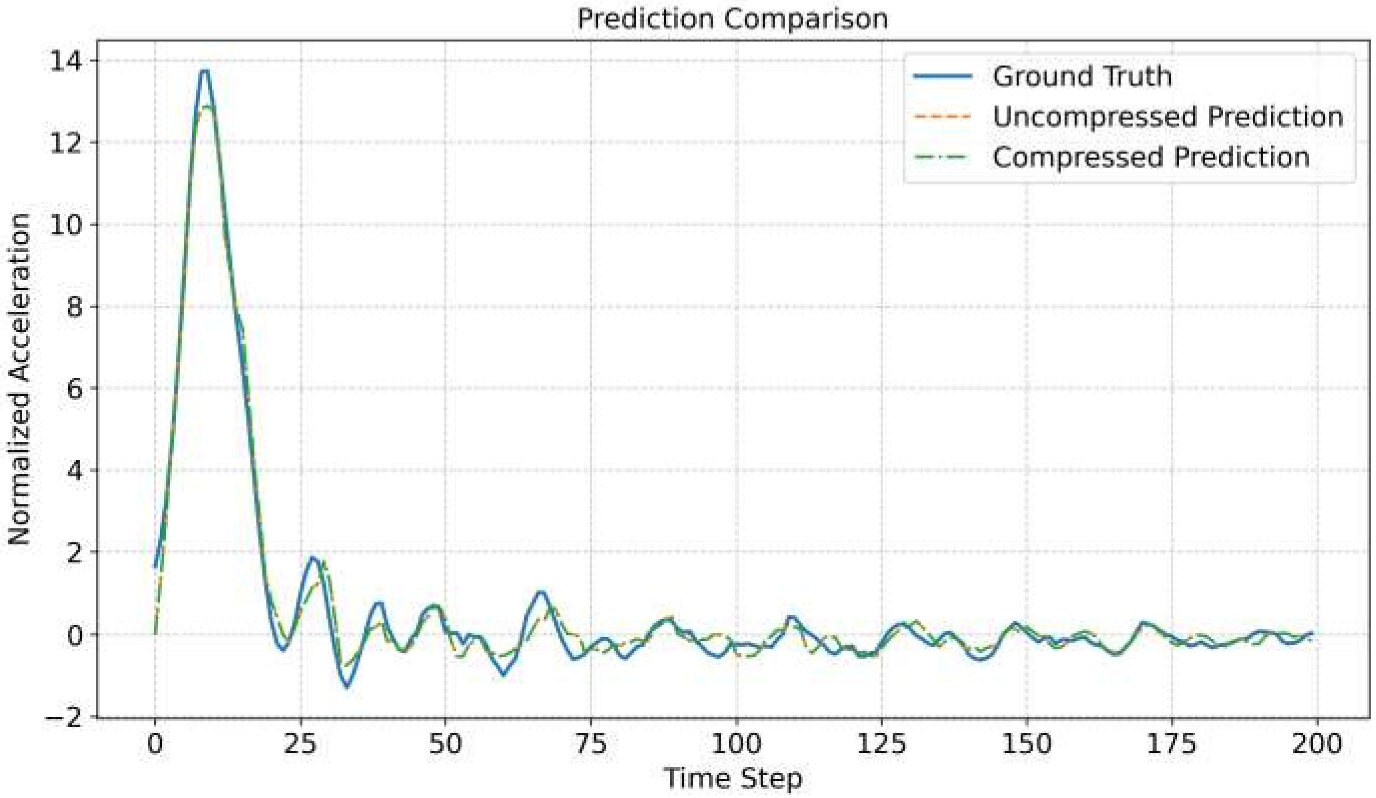
Observations:

- Faster Inference
- Lower Latency & Runtime
- Improved Accuracy : compression did not compromise prediction quality
- Smaller Memory Footprint

Performance Results from Rank Reduction

Metric	Uncompressed	Compressed (Rank-Reduced)	Change
MAE	0.0735	0.0728	-0.95%
R^2	0.9842	0.9851	+0.09%
Memory footprint	47.4KB	43KB	-4.4KB
Average Latency	135.966 ms	104.285 ms	-23.3%
Speed-up		1.30× faster	✓
Time per timestep	10.47ms	8.05ms	↓ 23.1%
Total Runtime	239 sec	184.5sec	↓ 23.1%

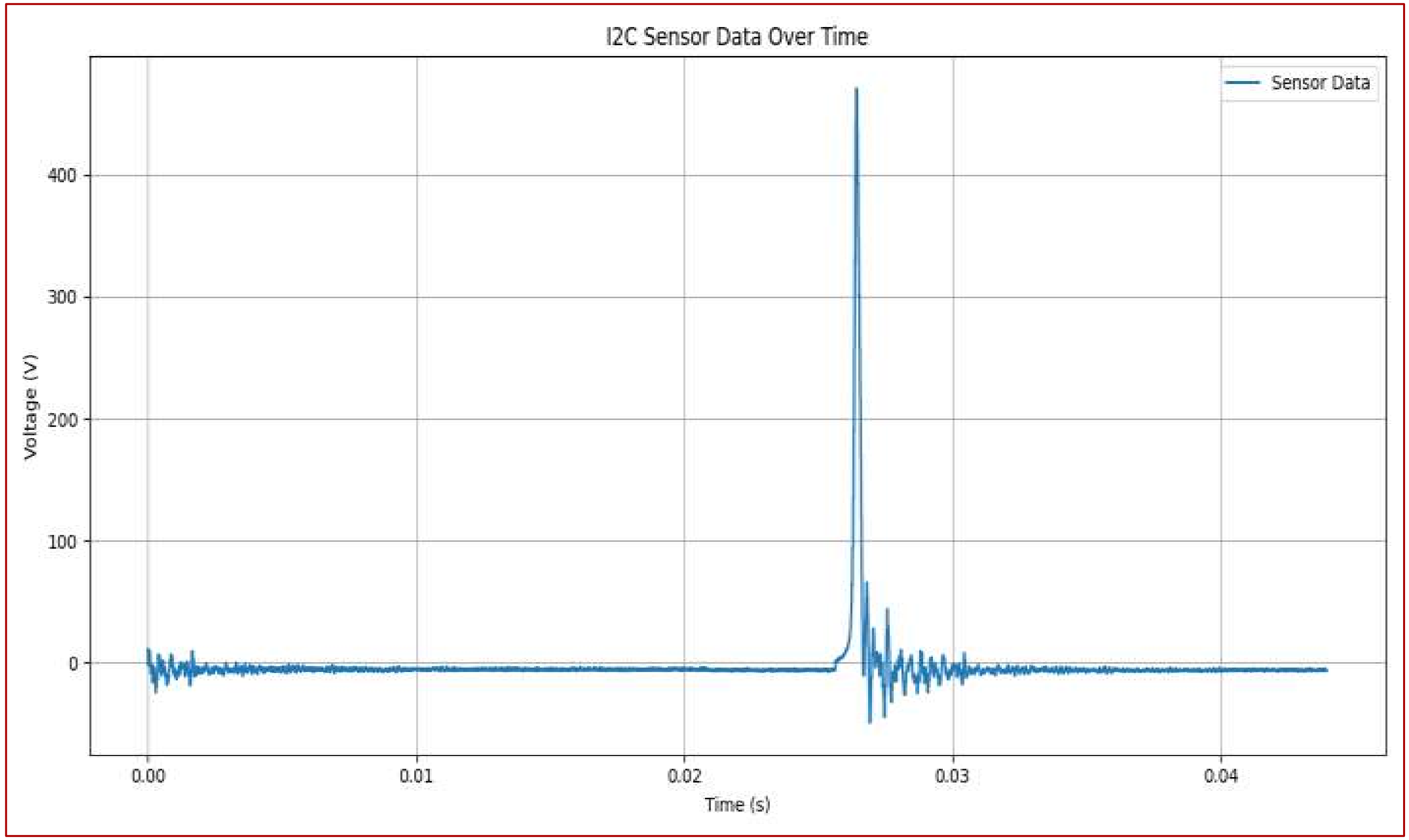
RESULT OVERVIEW : RANK REDUCTION TECHNIQUE



LAB DEMONSTRATION CHALLENGES

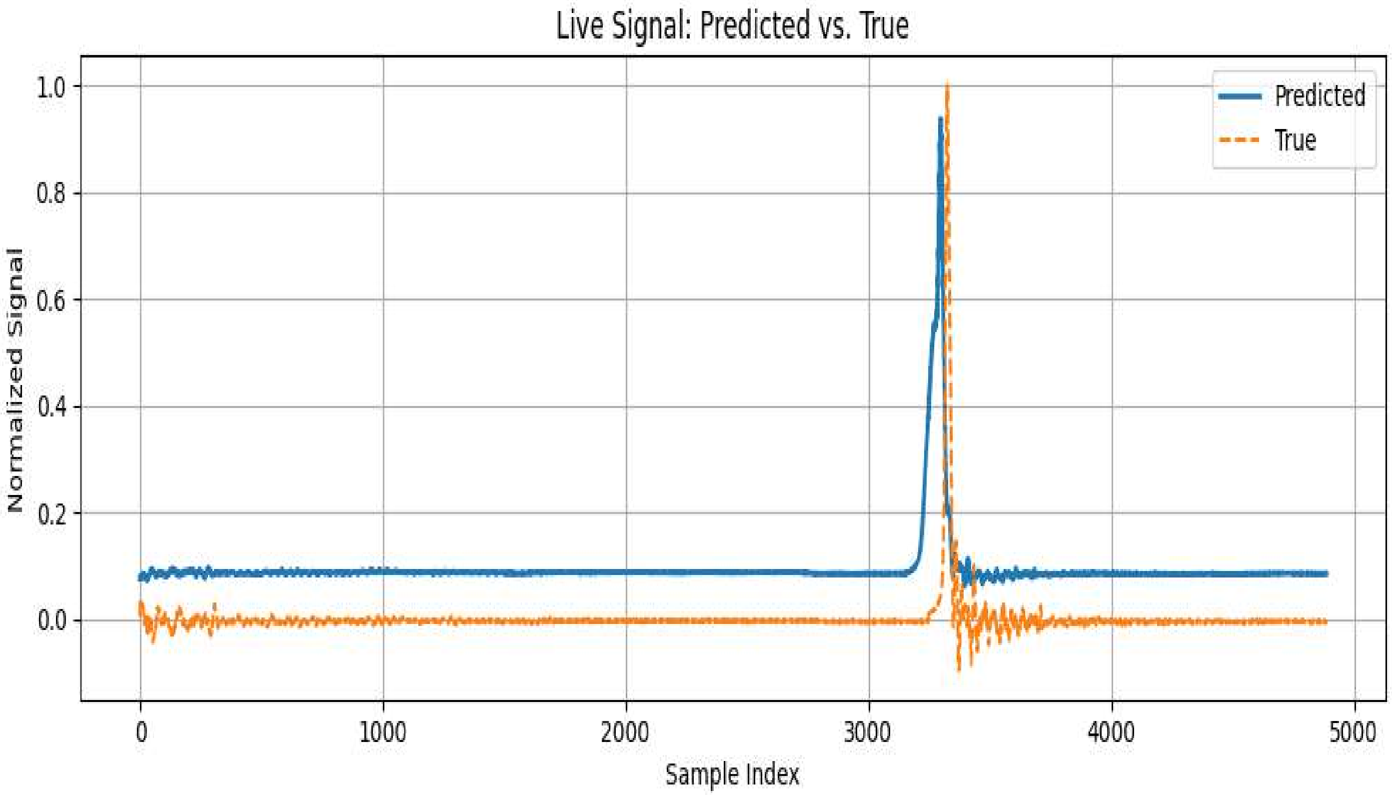
1) Setup & Data Flow
•Took time to connect ADC and Raspberry Pi correctly
•I2C signal showed multiple impacts.
•Needed to check if the right signal was reaching the model

2) Model Speed
•First test took over 30 minutes to finish
•Too slow for real-time use on Raspberry Pi



Lab Demonstration Challenges

3) Prediction Issues
•Output didn't match expected signal
•Delay between actual and predicted signal
•Hard to align model input and true values



CONCLUSION AND FUTURE WORK

- The primary goal was to successfully receive high-rate sensor data on a Raspberry Pi and run inference
- We established a full pipeline: signal acquisition → ADC → I2C → preprocessing → model inference
- Initial challenges included signal noise, alignment, and slow model runtime
- Explored different techniques tuning delays and using rank reduction to enhance quality of prediction

FUTURE WORK

- Future work involves a thorough parametric study to systematically investigate how changes in key model parameters— delay values, rank reduction ratio, and LSTM unit size—affect prediction accuracy, computational efficiency, and inference latency.

QUESTIONS?

ACKNOWLEDGEMENTS

- Air Force Research Laboratory Munitions Directorate.
- Air Force Office of Scientific Research (AFOSR).
- Defense Established Programs to Simulate Competitive Research (DEPSCoR).