# Neural Net-based Gyroscope Predictor for VR Headsets


**Joshua Shabanov**

**Omer Reuven**

**GitHub Repository:** https://github.com/EEJoshua/Gyro_Predictor_Neural_Net

## Abstract

In this project, we inspect several deep learning-based approaches for predicting head movements in Virtual Reality (VR) and Augmented Reality (AR) environments using gyroscope from IMU (inertial measurement unit) data.
We develop and evaluate three deep learning models (LSTM, CNN, and Transformer) alongside classical Recursive Least Squares (RLS) baselines. Each approach predicts 4 short horizons: 1, 2,3 and 4 samples ahead of the last acquired IMU sample, at both 52 Hz and 104 Hz sampling rates. This allows us to overcome most of the processing latency and improve user experience.
We compare models on prediction accuracy, memory consumption, and inference time to determine which setup is best suited for real-time VR/AR applications under the tight constraints of existing headset hardware.


# 1. Introduction

### 1.1 Project Goal

The goal of this project is to develop and evaluate multiple deep learning models and classical baselines for short-horizon IMU data prediction. We aim to identify the best performing architecture for predicting future head motion in VR/AR headsets, based on trade-offs between prediction accuracy, latency, and computational efficiency, while operating under the constraints of current VR/AR goggles.

### 1.2 Motivation

Latency in VR/AR headsets arises from both the processing time for the next frame and sensor delays. Delays between head movement and visual updates can cause motion sickness and degrade user experience. Predicting head movement a few milliseconds into the future enables rendering systems to prepare frames earlier, reducing perceived latency. Short-horizon forecasting (tens of milliseconds) offers a practical approach, enabling smooth tracking without large error accumulation.

### 1.3 Previous Work

Previous research on VR motion prediction has explored:

- Classical filters such as Kalman and Recursive Least Squares (RLS) for short-term motion estimation.
- Recurrent neural networks (RNN, LSTM, GRU) for time-series prediction.
- CNN based temporal models for low-latency forecasting.

While classical methods are computationally efficient, they often struggle with complex non-linear motion. Deep learning approaches have shown improved accuracy, especially when trained on large datasets of real IMU recordings. However, many studies focus on single-horizon prediction rather than multi-horizon forecasting and emphasize error reduction without considering latency or computational cost.

Our work extends this by comparing CNN, LSTM, Transformer, and RLS methods on predefined datasets at both 52 Hz and 104 Hz, evaluating accuracy and inference time jointly.

## 2. Method

### 2.1 Algorithm Overview

We evaluate multiple architectures for IMU based motion prediction, including:

- LSTM (Small and Large) for temporal sequence modeling of motion patterns.
- CNN for low-latency feature extraction from short windows.
- Transformer for attention based temporal modeling of IMU sequences.
- RLS baseline as a classical filter-based prediction method for fast, low-resource devices.

Each model takes an input sequence of recent IMU readings (accelerometer and gyroscope) and outputs predicted readings for horizons H = {1, 2, 3,4} samples ahead. At 52 Hz, these correspond to approximately 19 ms, 38 ms, and 57 ms 76ms. At 104 Hz, they correspond to approximately 9.6 ms, 19.2 ms, and 28.8 ms 38.4ms.

### 2.2 Architecture

The pipeline consists of:

1. Data preprocessing and normalization (data/io.py, data/normalize.py).
2. Continuous (no jumps) data segmentation (data/segment.py).
3. Dataset creation and splitting to train\test.
4. Model training and evaluation (train/trainer.py, train/metrics.py) with architectures from models/.
5. Visualization and diagnostics (viz/plots.py, viz/summaries.py).

### 2.3 Hyperparameter

| Param name | Default | Explanation |
|---|---|---|
| data_root | Path = Path(.) | project root for data (expects ./epsilon_52Hz and ./epsilon_104Hz). |
| artifacts | Path = Path(./artifacts) | output dir for models, metrics, plots, caches (splits), etc.. |
| use_acc | bool = True | include accelerometer (ax, ay, az) alongside gyro inputs. |
| skip_uncalibrated | bool = True | strictly skip files without valid calibration (never use raw). |
| jump_factor | float = 1.5 | split where dt > jump_factor × nominal dt (handles BT time jumps). |
| min_segment_len | int = 32 | minimum contiguous segment length (in samples) kept after splitting. |
| hz_list | tuple = (52,104) | sampling rates (Hz) to process; must match dataset folders. |
| horizons | tuple = (1,2,3,4) | prediction horizons k (steps ahead) to train/evaluate. |
| history_map | dict = {1:8, 2:10, 3:10, 4:12} | mapping k → history length H (past samples fed to the model). |
| split_ratios | tuple = (0.70, 0.15, 0.15) | train/val/test split fractions applied per-Hz to segments. |
| seed | int = 42 | global RNG seed (splits, init, augmentation). |
| epochs | int = 50 | max training epochs (early stopping may stop earlier). |
| batch_size | int = 2048 | DataLoader batch size (adjust to your VRAM). |
| lr | float = 1e-3 | optimizer learning rate (Adam/AdamW). |
| weight_decay | float = 1e-4 | L2 weight decay (AdamW) strength. |
| early_patience | int = 5 | stop after this many epochs with no val-loss improvement. |
| retrain | bool = False | if True, ignore cached weights/metrics and retrain from scratch. |
| num_workers | int = 8 | |
| persistent_workers | bool = True | only takes effect if num_workers > 0. |
| prefetch_factor | int = 4 | only used if num_workers > 0. |
| pin_memory | bool = True | CUDA input pipeline. |
| seed | int = 42 | for worker seeding. |

**model architecture is fixed and is the result of trial and error:**

Transformer

| | | | |
|---|---|---|---|
| **Linear** | **1x12x128** | **in=6, out=128** | |
| **Pos Encoding** | **1x12x128** | | |
| **Dropout** | **1x12x256** | | |
| **LayerNorm** | **1x12x128** | | |
| **Linear** | **1x12x256** | | |
| **Dropout** | **1x12x128** | | **X2** |
| **Linear** | **1x12x256** | | |
| **Dropout** | **1x12x128** | | |
| **LayerNorm** | **1x12x128** | | |
| **Linear** | **1x12x256** | | |
| **LayerNorm** | **1x12x128** | | |
| **Linear** | **1x3** | **in=128, out=3** | |

LSTM

| | | |
|---|---|---|
| **LSTM** | **1x12x128** | **hidden=128, layers=2** |
| **Linear** | **1x3** | |

CNN

| **Layer** | **Output Shape** | **Activation** | |
|---|---|---|---|
| Conv1d | 1x64x12 | GELU | |
| GELU | 1x64x12 | GELU | **X3** |
| BatchNorm 1d | 1x64x12 | GELU | |
| AdaptiveAvg Pool1d | 1x64x1 | | |
| Flatten | 1x64 | | |
| Linear | 1x3 | | |

## 3. Experiments and Results

### 3.1 Dataset

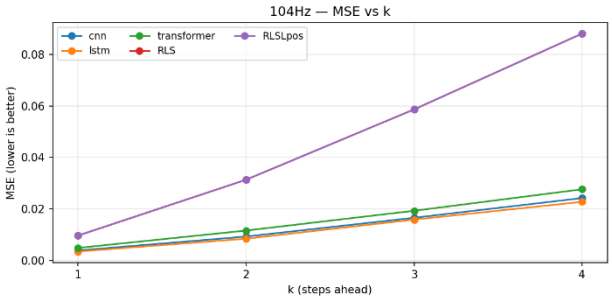
The dataset consists of recorded VR headset IMU data, sampled at both 52 Hz and 104 Hz, containing accelerometer and gyroscope readings during real head movements.

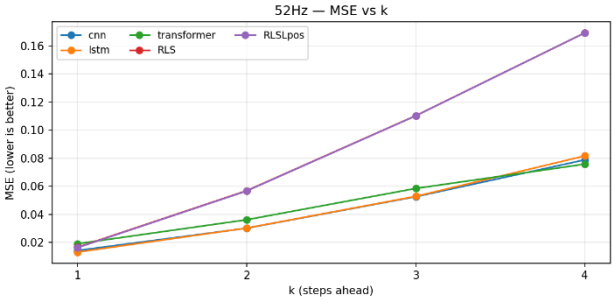
### 3.2 Workflow

1. Preprocess and normalize IMU recordings.
2. Segment data into sliding windows.
3. Train each model (CNN, LSTM, Transformer, RLS) on identical horizons.
4. Evaluate accuracy (MSE) and latency (average inference time).
5. Select Pareto optimal models balancing accuracy and speed.

### 3.3 Results

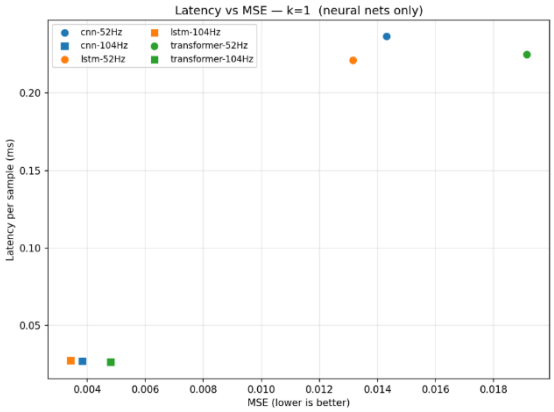
First, we wanted to estimate how the L2 loss compares and from the graphs bellow it appears LSTM performance is best on all horizons very close to the CNN network.

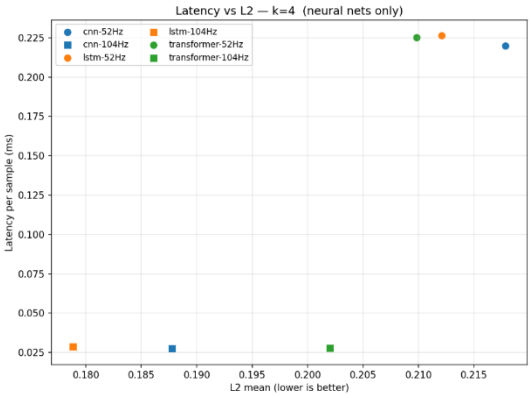



Using higher frequency did not improve performance significantly.

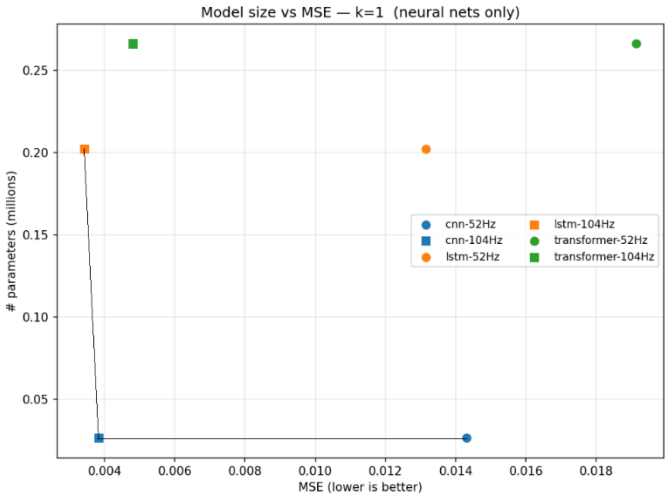
The following graphs present l2 loss vs latency so the pareto front of viable tradeoffs it appears that LSTM on 104HZ performs well in both parameters and therefore appears best in all horizons only two shown here but more available on out git page.



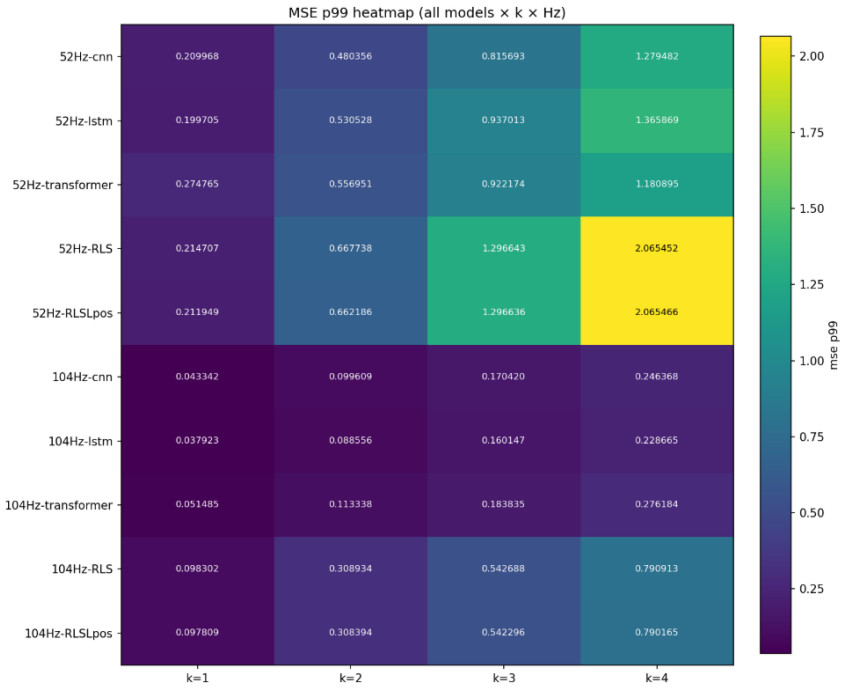



Comparing RLS that runs on a CPU to Deep model that runs on a GPU does not compare effectively. In order to compare them we need to run our deep models in a quantized form on hardware that can support that. Hence the comparison is only among the models of the same type deep and classic.

The following graph shows the pareto front of l2 loss and parameters count (e.g the weight of the model). Weight is a key parameter on an embedded system. We can see here that CNN on 52Hz performs well on both loss and weight and is x8 times lighter than LSTM. However, if weight limits allow LSTM. is more desirable.




Another consideration we checked is edge cases since prediction that has high l2 loss means the frame will be completely off causing unpleasant jitter to the user, and is highly undesirable the following graph shows heat map of the p99 of each model



As seen in the heatmap graph above classic RLS models have high degree of uncertainty in edge cases that must be solved algorithmically.  Both CNN and LSTM with 104Hz offer great improvement in that aspect.

**4. Alternative Solutions**

While this project focuses on deep learning and RLS baselines, other classical approaches such as Kalman filters or polynomial fitting remain viable for low-resource devices. Hybrid systems could use neural networks to handle non-linear motion while classical filters refine the final prediction.

Combinations of multiple models were also attempted but weren’t competitive since they weren't able to significantly improve results for short-term predictions and had more latency

**5. Conclusion and Future Work**

Our findings show that LSTM in 104Hz achieved the best trade-off between accuracy and inference time for short-horizon VR motion prediction at the cost of Weight in memory. CNN was not far behind but is significantly lighter and may be better suited for an embedded system.

Both models also offer great improvement over classic RLS which is used today. Showing that deep learning model are viable option in AR/VR system for short term prediction.

Future work will explore:

- Extending horizons beyond 3 samples for applications requiring larger pre-render buffers.
- Incorporating uncertainty estimation to guide latency adaptive rendering.
- Deploying models to real-time VR/AR hardware to measure perceptual improvements.

**Ethics Statement**

**1. Introduction**
**Student names:** Omer Reuven, Joshua Shabanov
**Project Title:** Neural Net-based Gyroscope Predictor for VR Headsets

**Describe your project:**
This project develops a machine learning system that predicts head movement in virtual reality (VR) environments using IMU (Inertial Measurement Unit) sensor data. The aim is to forecast motion 1–3 steps ahead to reduce display latency, improve visual stability, and enhance user comfort. By predicting motion before it occurs, the system enables smoother rendering and more immersive VR experiences.

---

**2. LLM Responses**

**a. List 3 types of stakeholders that will be affected by the project:**

1. VR headset end-users (gamers, professionals, educators).
2. VR hardware and software developers.
3. VR platform and content providers.

**b. What will an explanation that is given to each stakeholder look like?**

- **VR headset end-users:**
  "Our system uses sensors inside your VR headset to predict where you'll look or move next, reducing the lag you might feel between moving your head and seeing the scene update. This means a smoother, more comfortable, and more immersive VR experience."
- **VR hardware/software developers:**
  "This project implements a low-latency multi-horizon IMU prediction model that can forecast head motion up to 60 ms ahead. The predictions can be integrated into your rendering pipeline to enable frame preloading and reduced motion-to-photon latency, improving system responsiveness."
- **VR platform/content providers:**
  "Our predictive motion system can be integrated into your VR platform to ensure smoother frame delivery for your users. By lowering perceived latency, it enhances the quality of interactive VR content, making it more engaging and less prone to motion sickness."

**c. Who is responsible for giving the explanation to each stakeholder?**

- **VR headset end-users:** Marketing and customer support teams, trained to communicate technical benefits in simple, relatable terms.
- **VR hardware/software developers:** The engineering and R&D teams, providing technical documentation, performance benchmarks, and API usage guides.

- **VR platform/content providers:** Business development and technical partnership managers, ensuring integration guidelines are clear and aligned with platform requirements.

---

**3. Reflection (Independent Creative Thinking)**

**a. What do you think needs to be added/changed in the Generative AI responses, to make the explanation more ethical?**
The explanation does not include how we keep the privacy of the users, and their data would be safe with them. The ai answers mostly refer to performance and business aspects. Privacy concerns often arise among users when using wearables and it is better to address.