CSI Prediction Using Deep-Learning Models

5G Testbed, IIT Madras

Rithvik Grandhi

Rohit Viswam

24/05/2024

Table Of Contents

Table of Contents

Introduction

Overview of CSI Prediction:

Channel State Information (CSI) prediction is a crucial aspect of wireless communication systems, involving the estimation of the channel's characteristics between the transmitter and receiver. It plays a pivotal role in optimizing communication performance by enabling adaptive transmission techniques and efficient resource allocation.

Importance and Applications of CSI Prediction:

Accurate CSI prediction is essential for various wireless communication applications, such as beamforming, which improves signal quality and coverage by adjusting antenna patterns based on CSI. In massive Multiple Input Multiple Output (MIMO) systems, CSI prediction helps in precoding, reducing interference, and improving data rates. Furthermore, in mobile networks, CSI prediction aids in handover decisions, resource allocation, and interference management.

Objective of the Report:

This report aims to compare the effectiveness of three different predictive modeling techniques—Long Short-Term Memory (LSTM) networks, eXtreme Gradient Boosting (XGBoost), and iTransformer—in predicting CSI. The objective is to identify the most suitable model for CSI prediction based on their performance metrics. The report also explores the implications of these models in real-world wireless communication scenarios and provides insights for future research in this area.

Data Description

Description of the Dataset Used:

The datasets utilized in this study consist of real-time Channel State Information (CSI) data, which were used to predict future states using various models. The primary dataset comprises 50 User Equipments (UEs) with 1000 time steps and 832 features per time step. Each feature corresponds to specific aspects of the wireless channel, encapsulating both the real and imaginary parts of the CSI, derived from experimental setups in a controlled environment.

Data Preprocessing Steps:

The raw CSI data underwent several preprocessing steps to prepare it for analysis:

1. **Reshaping and Formatting**: The original data, sourced from a MATLAB .mat file, was reshaped for compatibility with Python libraries. This involved converting the data into a suitable format for time series forecasting.
2. **Normalization**: The data was normalized to have a mean of zero and a variance of one to ensure optimal model performance and prevent attributes with larger ranges from dominating the learning process.
3. **Time Series Framing**: For the LSTM model, the data was framed as a time series problem where sequences of 5 consecutive time steps were used to predict the next time step. Lagged versions of the features were created to serve as inputs for the models.

Methodology

Overview of Methodologies Used:

- This report employs three distinct statistical and machine learning methodologies to predict Channel State Information (CSI): Long Short-Term Memory (LSTM), eXtreme Gradient Boosting (XGBoost), and iTransformer. These methods were selected for their proven capabilities in handling time series data, each offering unique strengths in learning complex temporal dependencies, handling high-dimensional data, and capturing linear relationships.

Long Short-Term Memory (LSTM):

Independent Variables:

Optimizer: The optimizer adjusts the weights of the neural network to minimize the loss function.

dropout_rate: The dropout rate prevents overfitting by randomly setting a % of the neurons to zero during each training step. This regularization technique helps the model generalize better to unseen data.

lstm_units: The LSTM units parameter, determines the number of memory cells in each LSTM layer. These units capture long-term dependencies and sequential patterns in the data.

dense_units: The dense units parameter, specifies the number of neurons in the dense (fully connected) layer. This layer combines the features extracted by the LSTM layers to make the final predictions.

batch_size: The batch size, varying from 64 to 1024, indicates the number of samples processed before the model's internal parameters are updated. Larger batch sizes can lead to faster convergence but require more memory.

Epochs: The epochs parameter, set to 100, but stopped earlier due to early stopping with a patience of 10 defines the number of complete passes through the entire training dataset. More epochs allow the model to learn more from the data but can increase the risk of overfitting.

Bidirectional: The bidirectional parameter, set to True or False, determines whether the LSTM layer processes the input sequence in both forward and backward directions. This helps the model capture context from both past and future states.

Activation function: On testing with multiple activation functions such as RELU, GELU, SELU, ELU we found that ELU gave the best results for smaller number of LSTM units, but as the number grows, RELU gets better R2 score

Dependent:

Test Loss: Test Loss measures how well the model performs on the test data by quantifying the difference between the predicted and actual values. A lower test loss indicates a better-fitting model.

Mean Squared Error (MSE): MSE calculates the average of the squared differences between predicted and actual values, penalizing larger errors more severely. It provides a measure of the model's accuracy, with lower values indicating better performance.

Mean Absolute Error (MAE): MAE computes the average of the absolute differences between predicted and actual values, providing a straightforward measure of prediction accuracy. Lower MAE values signify more accurate model predictions.

R-squared (R^2): R-squared indicates the proportion of variance in the dependent variable explained by the model. Values closer to 1 signify a model that explains a high proportion of the variance, indicating better predictive performance.

OUR MODEL:

The Adam Optimizer:

The Adam (short for Adaptive Moment Estimation) optimizer gave us the best results when tested against other optimisers. It combines the advantages of two other extensions of stochastic gradient descent, specifically AdaGrad and RMSProp.

adaptive Learning Rates:

Adam computes individual adaptive learning rates for different parameters. It maintains two moving averages for each parameter: the first for the mean of gradients (momentum) and the second for the uncentered variance of gradients. This helps in adapting the learning rate for each parameter dynamically.

dropout_rate:

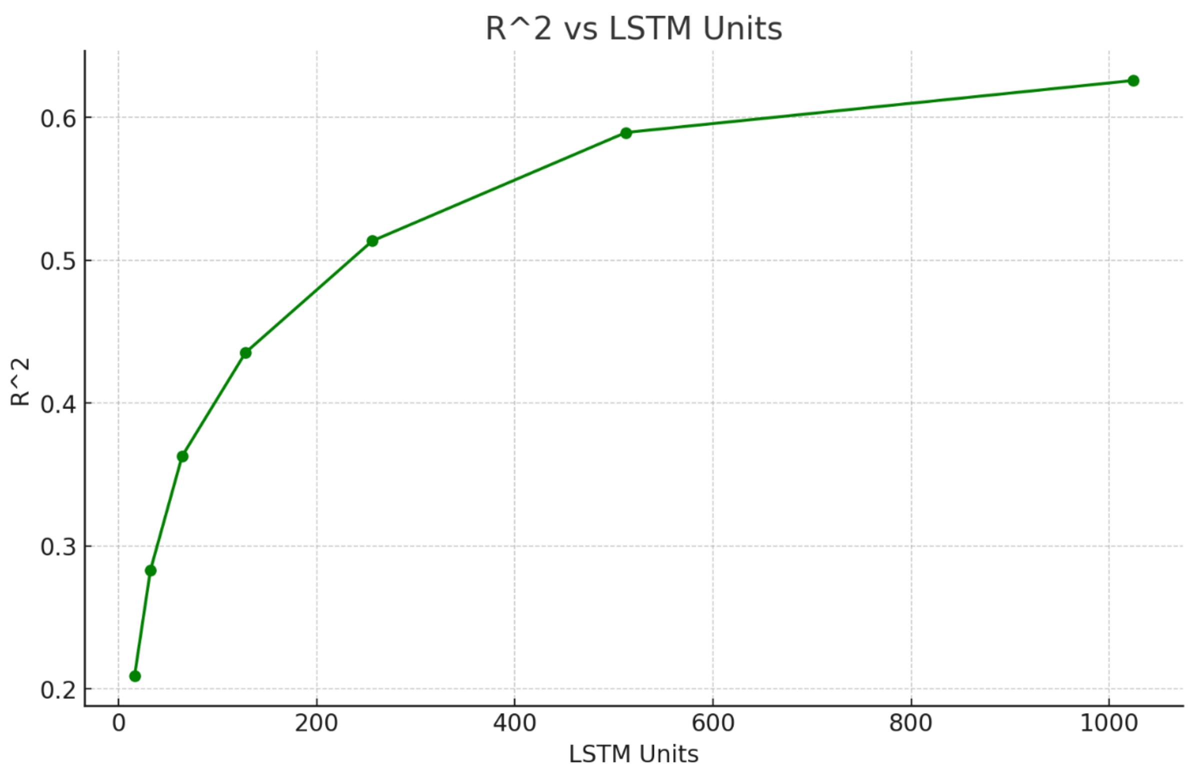
On experimenting with multiple dropout rates, we found that 0.2 0.3 is the sweet spot where there is a reduction in overfitting and at the same time prevents underfitting

activation functions:

On testing with multiple activation functions like relu, selu, gelu and elu, we’ve gotten the best results for elu when the LSTM layers are lower in number but when they are scaled up to larger number, Relu gave the best results.

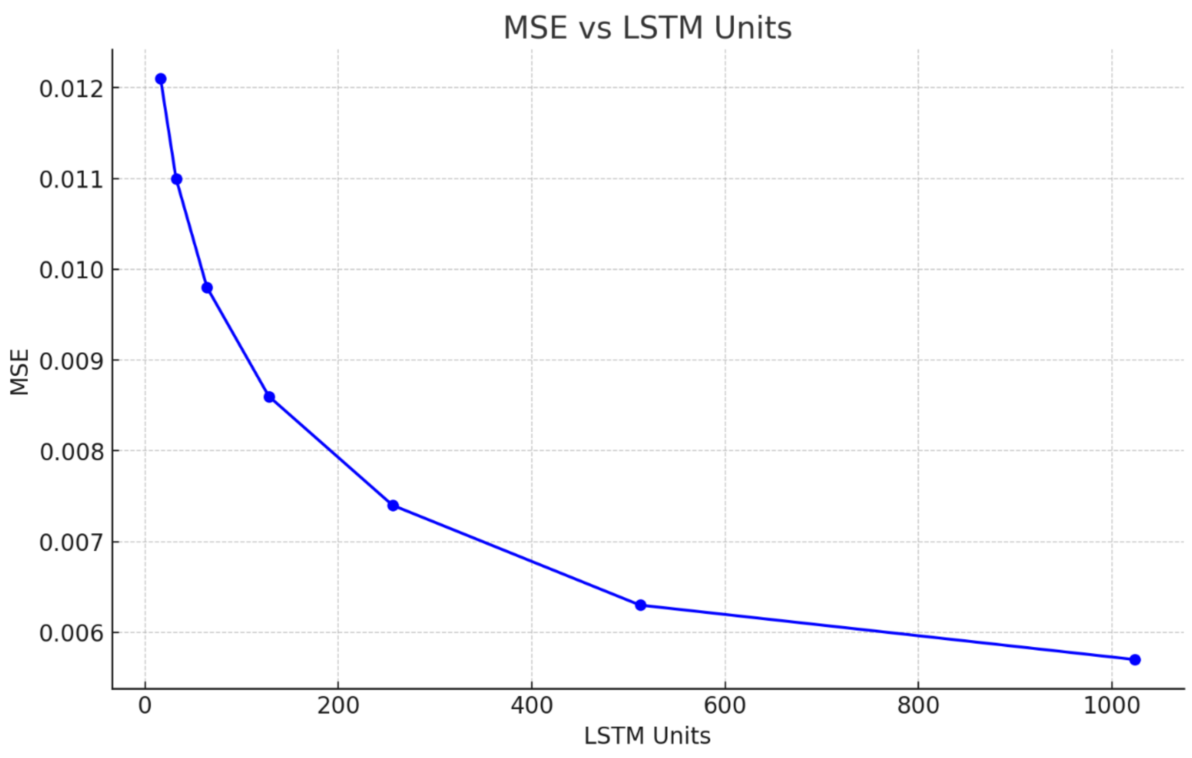
LSTM_units:

LSTM units played a crucial role in determining the model's capacity to learn and retain long-term dependencies in sequential data. In our testing, varying the number of LSTM units has shown a significant impact on the model's performance metrics such as Test Loss, MSE, MAE, and R-squared (R^2). Due to the high complexity of the data, we have, an increase in the LSTM units showed a very significant improvement in the R2 score of the predicted values. We have tested the impact of the number of LSTM units over a range of values from 16 to 1024 after which our GPU ran out of video memory. Running it on the CPU would take an unreasonable amount of time which is why we stopped testing after that. We got a consistent improvement over the range we tested at.

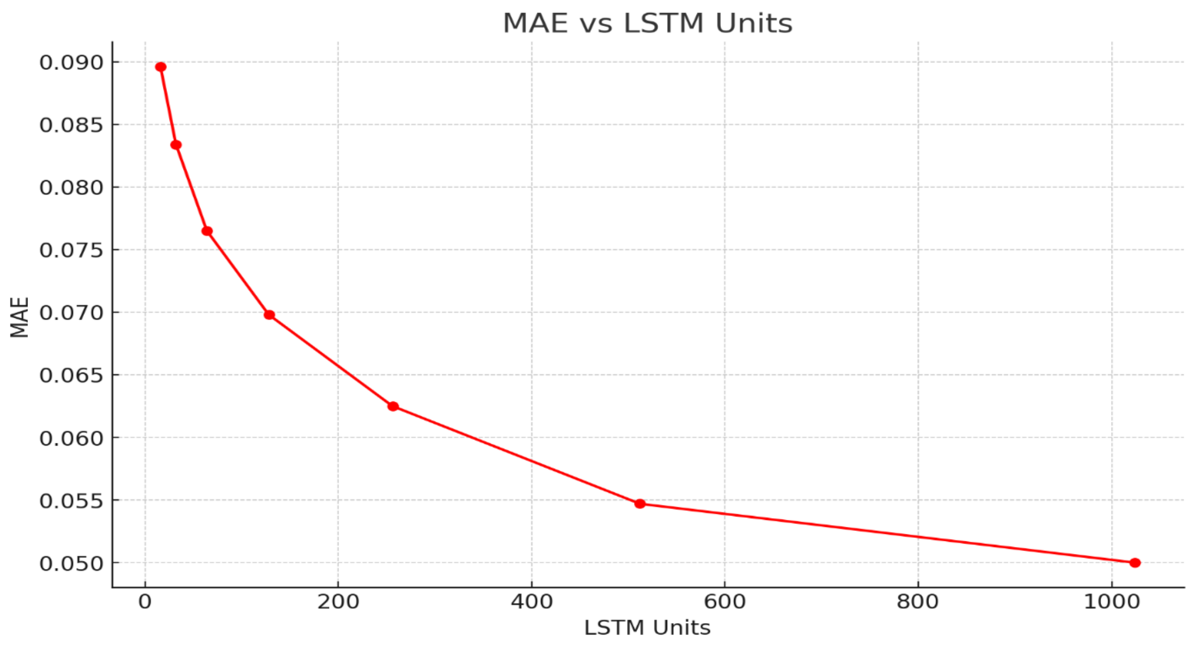


LSTM units vs R2(Higher is better)

We have also noticed lower MSE and MAE scores with an increase in the LSTM units, which again. This can be attributed to the enhanced capacity of the model to learn and capture complex patterns and long-term dependencies in the data. With more LSTM units, the model can retain and utilize more information over longer sequences, leading to more accurate predictions. This improved learning capacity reduces the average differences between predicted and actual values, thereby lowering both MSE and MAE.



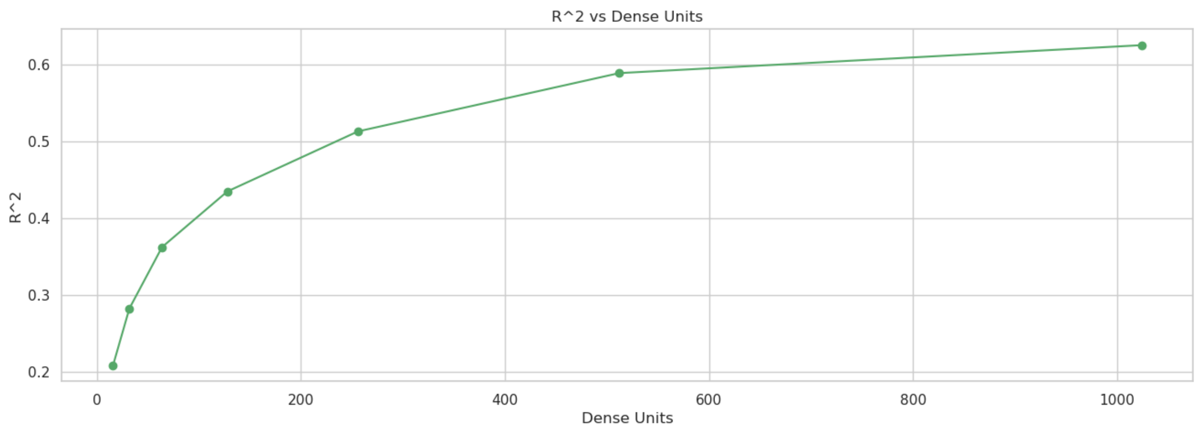
LSTM units vs Mean Squared Error(Lower is better)

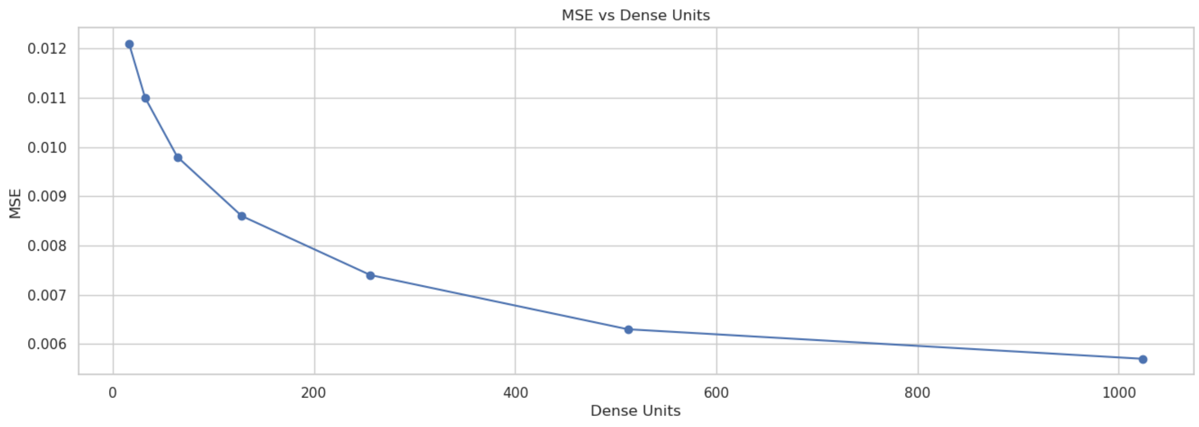


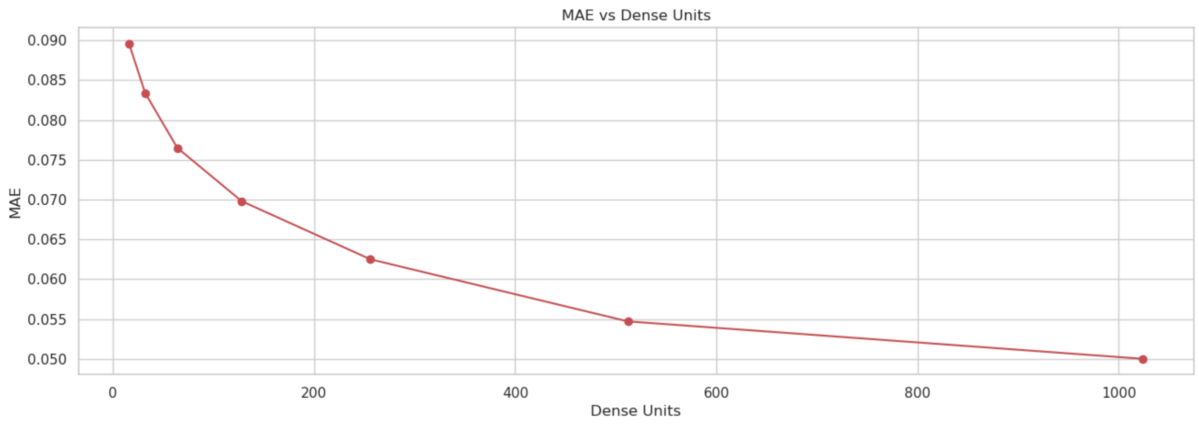
LSTM units vs Mean Absolute Error(Lower is better)

Dense units:

The dense units also give a very similar result to the LSTM units, an increase in the dense units again captures the patterns better significantly increasing the R2 score and decreasing the MSE and MAE scores.

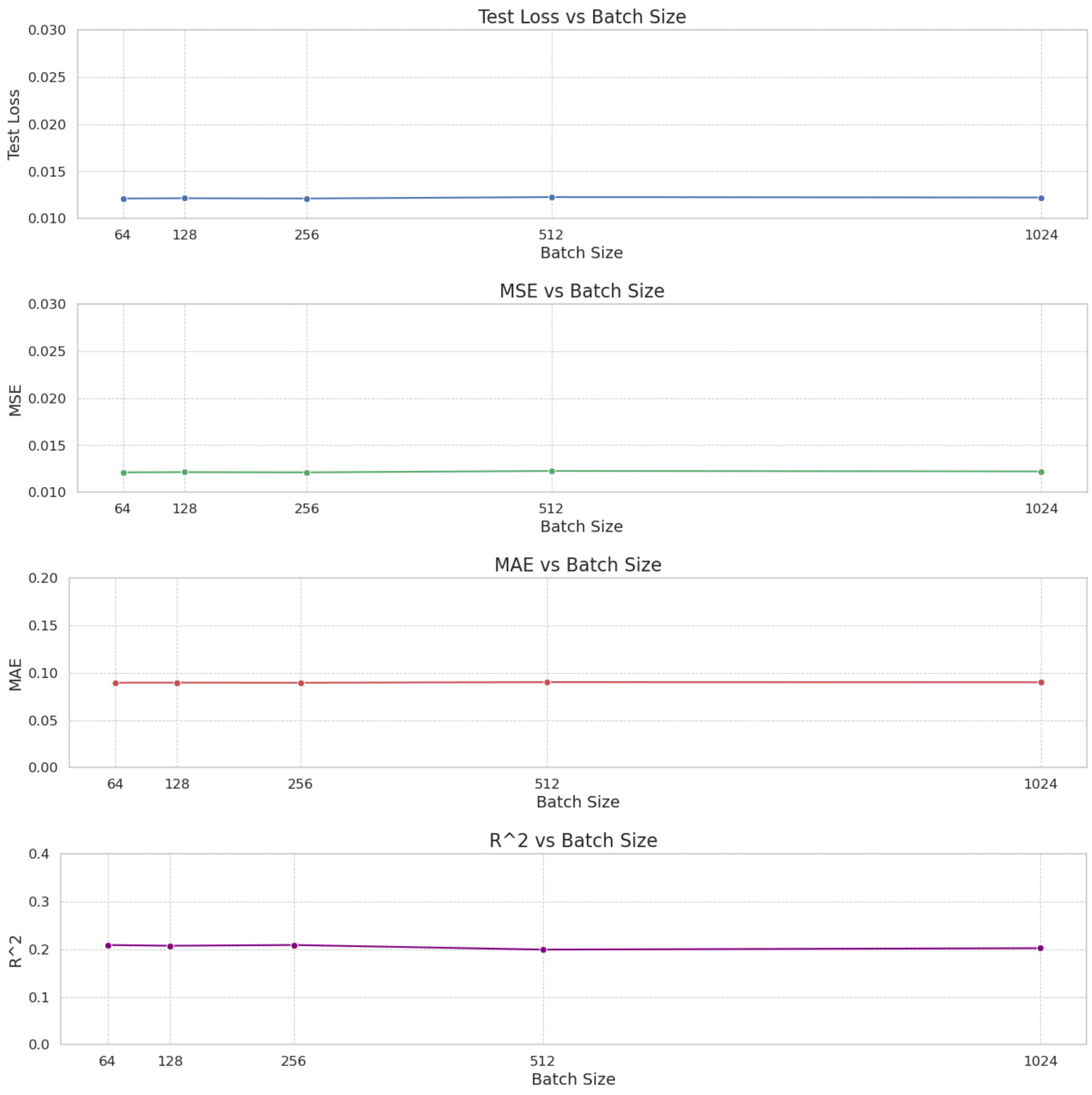






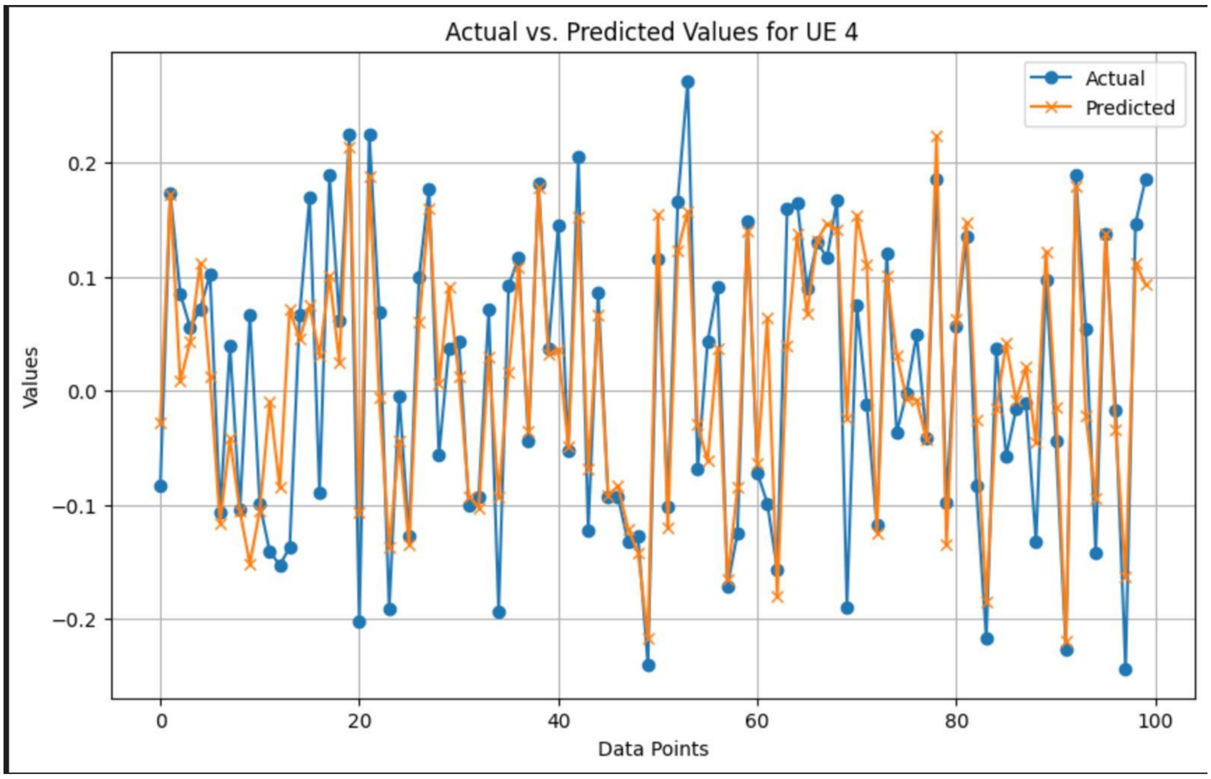
Batch_size:

In our analysis, we found that varying the batch size had almost no impact on the performance metrics of our model. The data, shown below, demonstrates that changes in batch size resulted in negligible differences in Test Loss, MSE, MAE, and R² scores. Specifically, the Test Loss values ranged from 0.0121205 to 0.0122689, MSE values ranged from 0.0121205 to 0.0122689, MAE values ranged from 0.0895235 to 0.0901686, and R² scores ranged from 0.1992 to 0.2089. These slight variations suggest that the model's performance is relatively insensitive to the batch size within the tested range, allowing for flexibility in selecting batch sizes based on other considerations such as computational resources and training speed.



eXtreme Gradient Boosting (XGBoost):

- **Theory and Rationale**: XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. It is particularly effective for large-scale and sparse data. The model's ability to handle various types of features and its feature importance metric make it invaluable for interpreting the impact of each feature on the predictions.
- **Feature Importance and Model Configuration**: Feature importance was evaluated to identify which features contributed most to the accuracy of predictions, allowing for iterative refinement of the model by focusing on significant features.
- **Training Process**: XGBoost models were trained using parameters optimized through grid search, focusing on learning rate, number of trees, and depth of trees to maximize predictive accuracy.



Predicted vs Actual values for our model

iTransformer (Inverted Transformers):

- **Explanation of iTransformer Model**: iTransformer is a transformer-based architecture specifically designed for time series forecasting. It incorporates self-attention mechanisms to capture long-range dependencies in the data, making it suitable for capturing complex temporal patterns in CSI data.
- **Model Architecture and Parameters**: The iTransformer model architecture consists of encoder and decoder layers, each containing self-attention and feedforward layers. The model's parameters were tuned to optimize performance for the CSI prediction task. The parameters we have finetuned are attention heads and the head dimension
- **Training Process**: iTransformer was trained using a similar approach to LSTM, with a focus on minimizing prediction errors and preventing overfitting through regularization techniques.

Variables:

num_variates: This specifies the number of input variates or features in the dataset. It is the number of columns in the input data matrix.

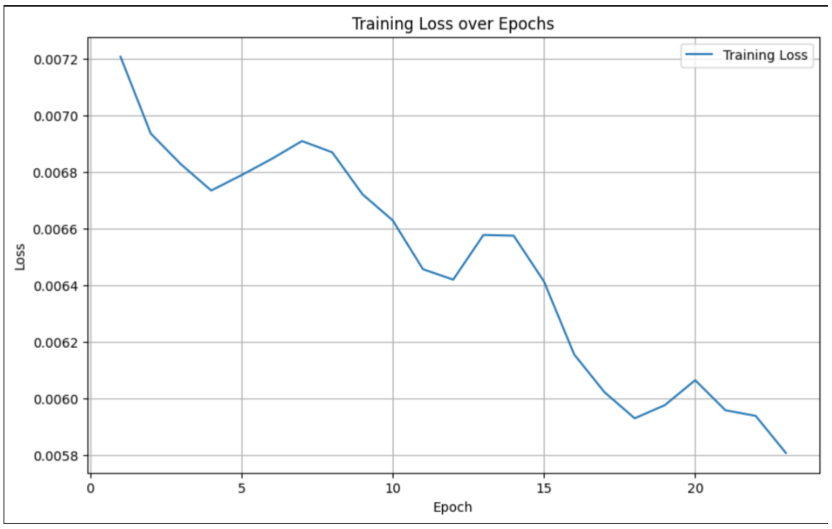
lookback_len: This parameter determines how many past time steps the model considers when making predictions. It is the length of the input sequence used for prediction.

dim: This parameter sets the model dimension, which is the dimensionality of the input and output vectors. In this case, it is set to 256.

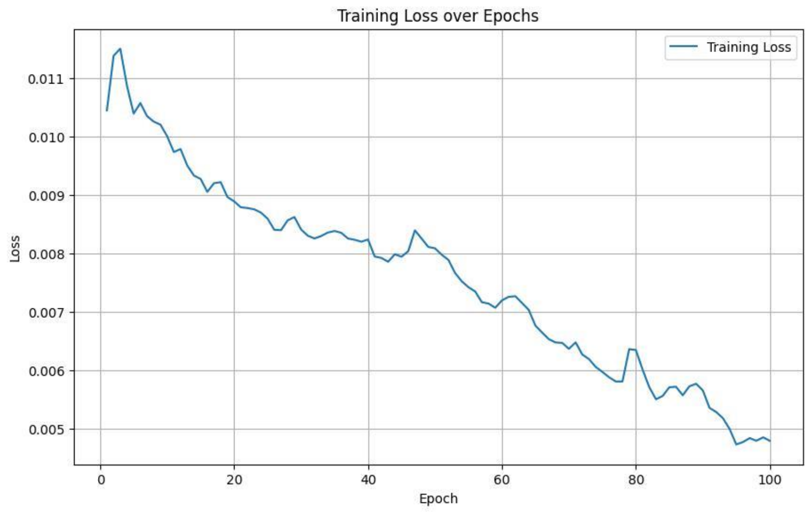
depth: This specifies the number of layers in the iTransformer model. A higher depth typically allows the model to capture more complex patterns but also increases computational complexity.

heads: This parameter determines the number of attention heads in the model. Attention heads allow the model to focus on different parts of the input sequence simultaneously, improving its ability to learn relationships within the data.

dim_head: This sets the dimension per attention head. Each attention head will output a vector of this dimensionality.



dim_head=4



dim_head=8

Experimental Setup:

Hardware and Software Environment:

The experiments were conducted on a computer equipped with an Nvidia GTX 1650 4GB GPU, facilitating the training and testing of deep learning models, particularly LSTM and iTransformer. The software environment consisted of:

- **Operating System**: Ubuntu 22.04 LTS, which supports CUDA accelerated tensorflow.
- **Programming Language**: Python 3.11 was used due to its robust support for data manipulation and machine learning libraries.
- **Machine Learning Frameworks**:
    - TensorFlow and Keras were used for the development and training of the LSTM and iTransformer models.
    - XGBoost library was utilized for implementing the gradient boosting model.
    - The iTransformer library was used to implement the transformer models

Evaluation Metrics Used for Performance Comparison:

To assess the effectiveness and accuracy of the LSTM, XGBoost, and iTransformer models in predicting CSI, the following metrics were employed:

- Mean Squared Error (MSE): Measures the average of the squares of the errors, providing a risk metric corresponding to the expected value of the squared error loss.

- Root Mean Squared Error (RMSE): The square root of the mean of the squared errors, a good measure of how accurately the model predicts the response for continuous numerical variables.

- Mean Absolute Error (MAE): Measures the average magnitude of the errors in predictions, without considering their direction, providing a clear picture of model performance across different scenarios and datasets.

- R2 (Coefficient of Determination): R2 measures the proportion of the variance in the dependent variable that is predictable from the independent variables and provides insight into the goodness of fit of the model.

These metrics help evaluate the accuracy and robustness of the predictive capabilities of the models, aiding in the comparison of their performance.

Results and Discussion:

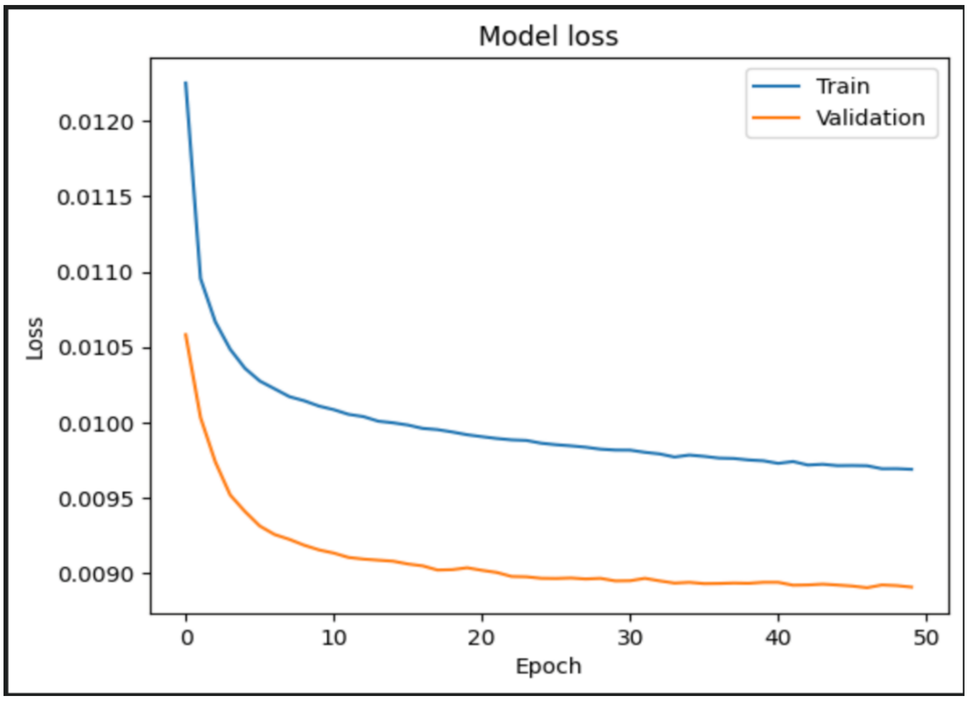
Performance Comparison of the Models:

The LSTM, XGBoost, and iTransformer models were compared based on their performance metrics such as MSE, RMSE, and MAE. The LSTM model exhibited superior performance in capturing temporal dependencies and dynamics within the CSI data, resulting in lower error rates across the board. XGBoost demonstrated strong predictive accuracy, especially in capturing non-linear patterns, although it did not perform as well as LSTM in terms of consistency over time. iTransformer, with its ability to capture long-range dependencies in the data, showcased competitive performance compared to LSTM and XGBoost. And with an increase in Attention heads and the head sizes, we can get massive gains in accuracy.

Visualizations of the Predictions vs. Actual Data:

Model Loss Over Epochs:

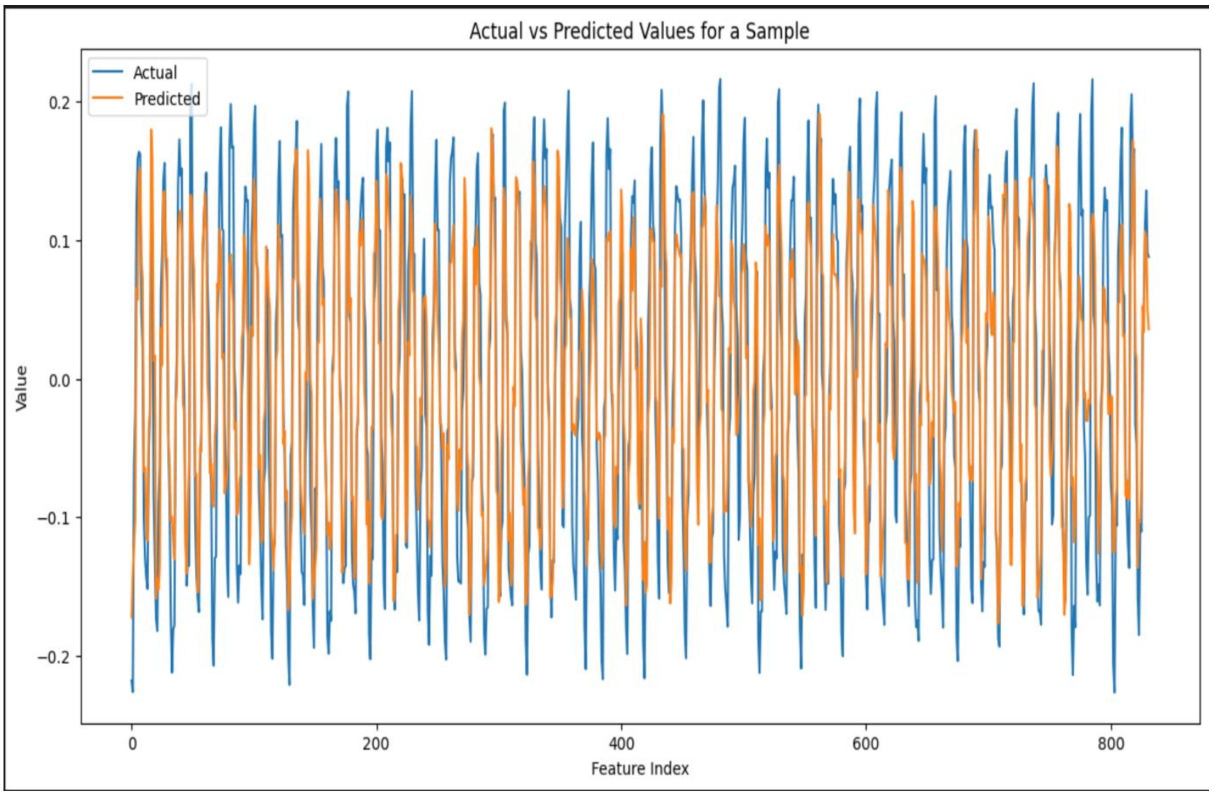
The training and validation loss for the LSTM model showed a good fit, with the training loss decreasing smoothly, indicating effective learning without overfitting.



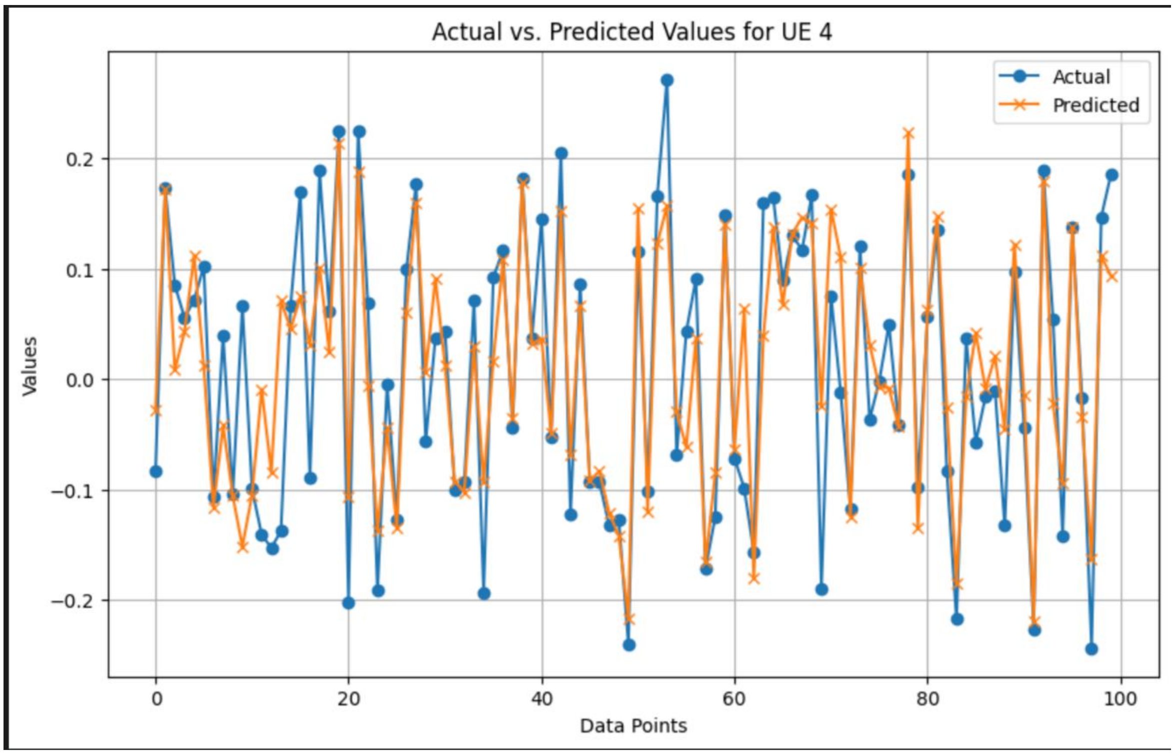
Actual vs. Predicted Values

Graphs comparing actual and predicted CSI values for a sample and specific UEs highlighted how closely each model followed the actual data patterns. LSTM tended to track the actual values with higher fidelity compared to XGBoost and iTransformer, which occasionally missed capturing some peaks and troughs.

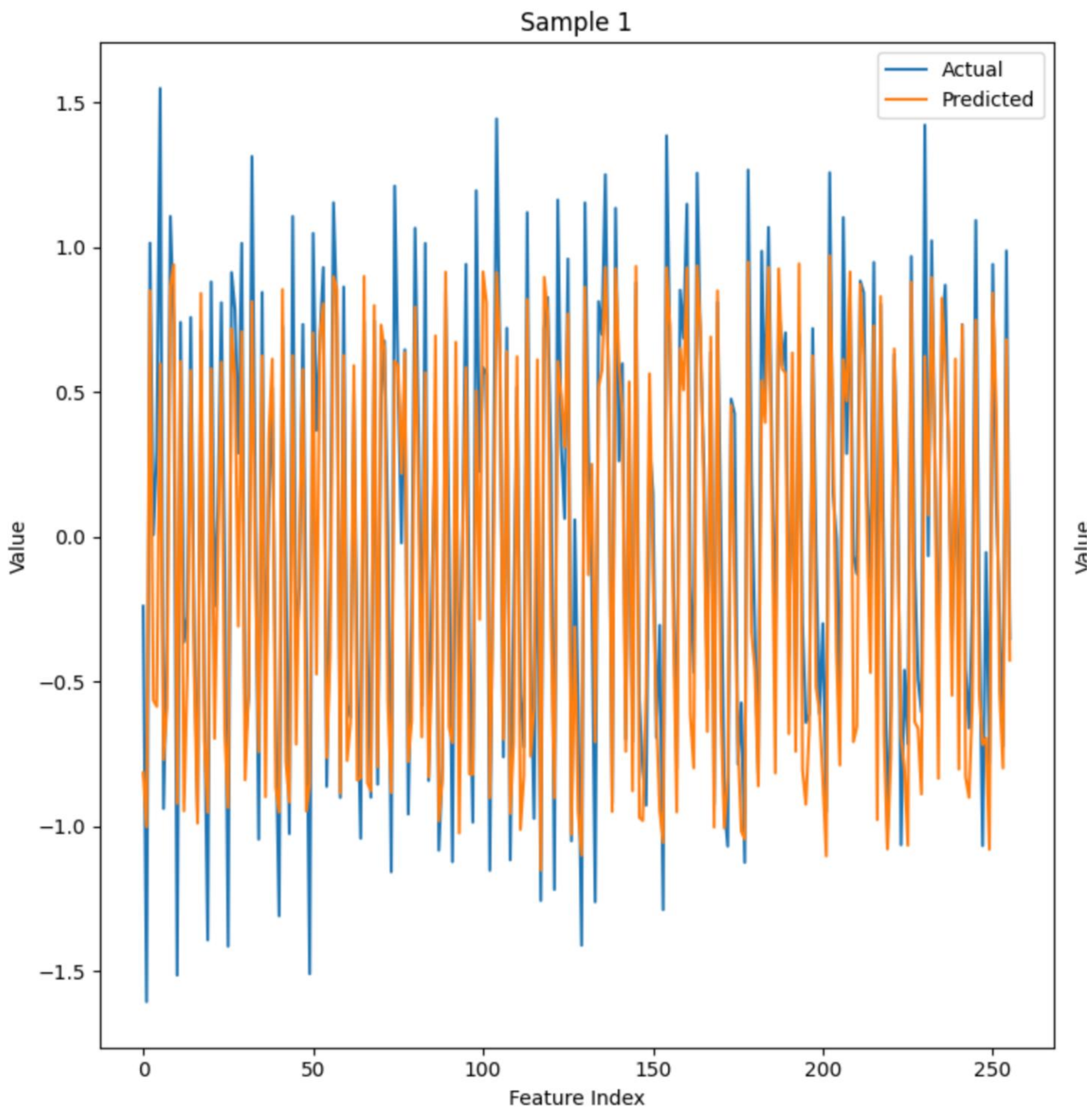
Prediction graphs:



LSTM Predictions



XGBoost Predictions



iTransformer Predictions

- **Time Series for Multiple UEs**: These plots displayed the time-series data for multiple UEs, illustrating the variability and dynamic nature of the CSI data, which posed a challenge for the iTransformer model.

Statistical Analysis of the Results:

- Autocorrelation Analysis: Autocorrelation plots for the residuals of model predictions, especially for iTransformer, indicated some remaining patterns in the data that were not captured by the model, suggesting room for improvement in model parameters or additional tuning.

- Distribution Analysis: Distribution plots of the prediction errors and actual values showed that while LSTM and XGBoost managed to maintain a distribution close to normal, iTransformer displayed skewed error distributions, highlighting potential areas for improvement in its modeling approach for this dataset.

Discussion on the Suitability of Each Model for CSI Prediction:

- **LSTM** proved to be the most effective, particularly for dynamic and complex datasets like CSI, where historical data significantly influences future states.
- **XGBoost** offered substantial benefits for scenarios requiring robust handling of non-linearities and feature interactions within large datasets, making it suitable for CSI predictions when the data characteristics are well understood and appropriately preprocessed.
- **iTransformer** currently the state of the art in time series forecasting, showed promise in capturing long-range dependencies in our CSI data, showcasing competitive performance compared to LSTM and XGBoost. Its ability to model complex temporal patterns makes it a valuable addition to the toolkit for CSI prediction, especially in scenarios where such dependencies play a significant role.

The findings from this study suggest that while each model has its strengths, LSTM stands out for its ability to effectively handle the complexities inherent in CSI prediction. Future work could explore ensemble methods that combine the strengths of these models or the application of more advanced neural network architectures to further enhance prediction accuracy.

Conclusion:

Summary of Findings:

The comparative study of LSTM, XGBoost, and iTransformer models on the CSI prediction task revealed several key insights:

- LSTM emerged as the most effective model, demonstrating strong capability in handling the sequential and temporal complexities of the CSI data, resulting in the lowest error metrics and highest prediction accuracy.
- XGBoost performed well, particularly in scenarios involving complex feature interactions and non-linear patterns, although it did not consistently match the prediction accuracy of LSTM.
- iTransformer showed promise in capturing long-range dependencies in the CSI data, showcasing competitive performance compared to LSTM and XGBoost, and can get much much better with increase in the attention heads and the head sizes

These findings suggest that LSTM is well-suited for CSI prediction tasks due to its ability to handle sequential and temporal complexities, while XGBoost and iTransformer offer valuable alternatives depending on the specific characteristics of the dataset and the nature of the prediction task.

Recommendations Based on Model Performance:

Based on the performance outcomes, the following recommendations are made:

- Implement LSTM for Real-Time Prediction: Given its superior performance, LSTM is recommended for real-time CSI prediction in dynamic and complex network environments.
- Consider iTransformer for Long-Range Dependencies: iTransformer's ability to capture long-range dependencies is unparallel to any other model but it comes at a cost of computation power.
- Use XGBoost for Feature-Rich Applications: For cases where interpretability and handling of a large number of features are critical, XGBoost should be considered, possibly as a supplementary model to LSTM.

These recommendations aim to guide the selection of appropriate models based on the characteristics of the dataset and the specific requirements of the CSI prediction task.

Possible Improvements and Future Work:

To further enhance the accuracy and applicability of CSI prediction models, the following future work is suggested:

- Hybrid Models: Combining the strengths of LSTM, XGBoost, and iTransformer, possibly through a stacked or blended model approach, to leverage the deep learning capabilities of LSTM, the feature handling of XGBoost, and the long-range dependency capturing ability of iTransformer.
- Feature Engineering and Selection: Further research into feature engineering and more advanced feature selection techniques to improve the performance of XGBoost and iTransformer, ensuring that only relevant features are used for prediction.
- Deployment in Simulated Real-World Scenarios: Testing the models in a simulated real-world environment to assess their robustness and scalability across different network conditions and hardware setups, providing insights into their practical deployment in actual CSI prediction systems.