# Deep Learning for Demand Forecasting in Retail Using Hybrid CNN-RNN and Transformer Models

**Project Report**

**DL for Supply Chain Optimization in Retail**

**NAME:** Raghulchellapandiyan Senthil Kumaran **UBID:** raghulch
**NAME:** Dongyoon Shin **UBID:** dongyoon

## Table of Contents

## ABSTRACT

Precise time series predictions are essential in the retail industry, where inventory, supply chain, and strategic planning depend on accurate forecasting. This project investigates the capability of deep learning models in predicting future sales over multiple time steps. We evaluate several architectures—including LSTM, GRU, Transformer, FFNN, hybrid CNN-RNN, and a baseline Linear model—on a multivariate time series dataset. Models are tested under both clean and noise-injected conditions to mimic real-world uncertainty and find the best model that is robust to noisy data and works well with clean data. Through a standardized experimental configuration and evaluation criteria, we explore the strengths and limitations of each model in both ideal and noisy settings.

## 1. PROBLEM STATEMENT

In the rapidly evolving and competitive retail sector, accurately forecasting future demand is crucial for maintaining appropriate stock levels, minimizing operational costs, and ensuring customer satisfaction. Traditional statistical forecasting models often struggle with the complexities of modern retail data, which is typically high-dimensional, noisy, and temporally dependent.

Real-world retail datasets commonly contain missing values, anomalies, and unexpected trends, making robust forecasting even more challenging. Deep learning models offer a compelling alternative by automatically learning temporal dependencies from raw multivariate time series data. However, the optimal architecture for multi-step forecasting—especially under both clean and noisy conditions—remains unclear. A systematic benchmarking study is thus necessary to understand the comparative advantages and limitations of each architecture in terms of accuracy, stability, and robustness to noise.

## 2. OBJECTIVES

This project aims to develop and evaluate deep learning-based models for multi-step time series forecasting in the context of retail sales. The specific objectives are:

- Develop robust forecasting models using deep learning architectures such as LSTM, GRU, Transformer, FFNN, and CNN-RNN hybrids.
- Evaluate model performance under clean and noisy input conditions to assess generalization and robustness.
- Compare forecast accuracy using metrics including MSE, RMSE, MAE, and $R^2$.
- Interpret model behavior through forecast plots, epoch-wise metric curves, and step-wise error analysis across the prediction horizon.
- Enhance model performance through architectural refinement and hyperparameter tuning.
- Select the most appropriate model for real-world deployment in retail environments with incomplete or noisy data.

By achieving these goals, the study both benchmarks state-of-the-art time series forecasting models and highlights practical deployment considerations in real-world, noisy retail data scenarios.

## 3. DATASET

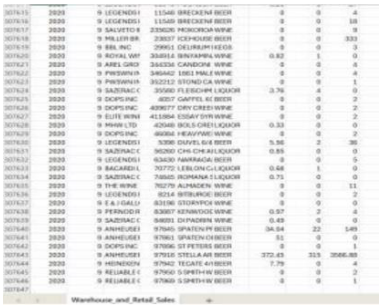


Figure 1: Sample Dataset

The dataset used in this project is the **Warehouse and Retail Sales** dataset, publicly available from Data.gov. It contains over 307,000 monthly sales records, organized by item and supplier. The dataset includes features such as:

- `YEAR`, `MONTH`
- `SUPPLIER`, `ITEM TYPE`
- `RETAIL SALES`, `RETAIL TRANSFERS`, `WAREHOUSE SALES`

Its size and complexity make it well-suited for training deep learning models that capture demand trends across multiple categories and suppliers over time.

## 4. METHODOLOGY

In this section, we describe the complete methodology pipeline adopted in this project. The process begins with the collection and cleaning of the raw retail sales dataset. We then perform extensive exploratory data analysis (EDA) to understand the data distribution, identify anomalies, and extract trends and patterns.

Next, we apply feature engineering techniques such as lag creation, log transformation, and normalization to prepare the data for supervised learning. We also simulate real-world uncertainty by injecting Gaussian noise to assess model robustness. Do hyperparameter tuning to find the best model. Finally, we train and evaluate multiple deep learning architectures to compare performance under clean and noisy conditions.

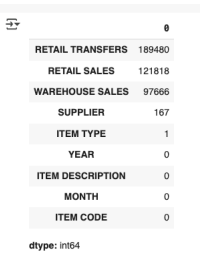
### 4.1. Data Preparation and Exploration

#### 4.1.1. Data Overview and Cleaning

The dataset consisted of 307,645 records with 9 features representing time, sales, product types, and supplier information. It was read using `pandas`, with inspection performed via `.info()`, `.head()`, and `.describe()` for data types, missing entries, and summary statistics.

Missing or invalid values (e.g., zeros or negatives in `RETAIL_SALES`) were replaced with NaNs and imputed using `IterativeImputer` with a `RandomForestRegressor` estimator. Log transformations (`log1p`) reduced skewness while preserving zero entries. Interquartile range (IQR) filtering removed 2–3% of outliers.

(a) Read and Describe

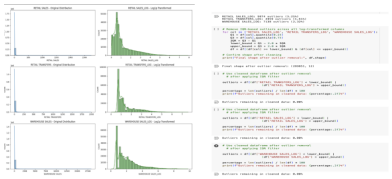


| | 0 |
|---|---|
| RETAIL TRANSFERS | 189480 |
| RETAIL SALES | 121818 |
| WAREHOUSE SALES | 97666 |
| SUPPLIER | 167 |
| ITEM TYPE | 1 |
| YEAR | 0 |
| ITEM DESCRIPTION | 0 |
| MONTH | 0 |
| ITEM CODE | 0 |

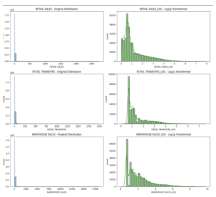
dtype: int64

(b) NULL VALUES

(c) Imputation



(d) Skewness and After Outlier removal

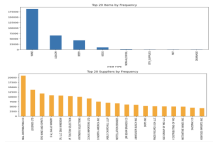


(e) After Outlier Removal

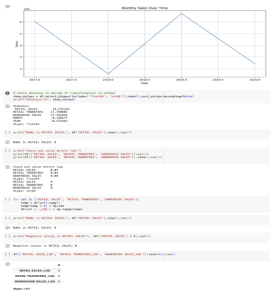
Figure 2: Data Cleaning and Preprocessing Steps

### 4.1.2. Exploratory Data Analysis (EDA)

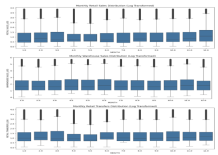
Frequency analysis revealed `WINE`, `LIQUOR`, and `BEER` as dominant categories. Seasonal patterns appeared in time series plots, especially strong Q4 peaks. Boxplots by month confirmed this. Rolling averages further smoothed short-term fluctuations. Pairplots revealed weak correlations between `RETAIL_SALES`, `INVENTORY`, and `UNITS_SOLD`.



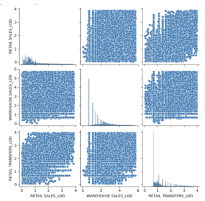
(a) Frequency Analysis



(b) Graph and Log Transformation



(c) Boxplot of Monthly Seasonal Variation



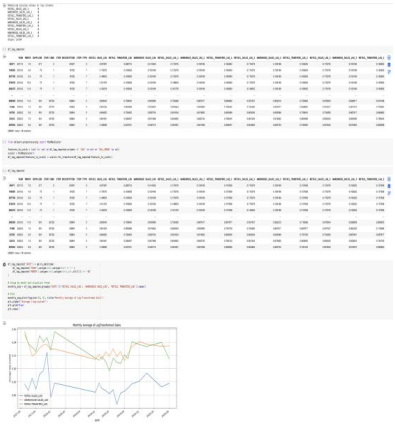
(d) Pairplot for Feature Relationships

Figure 3: Exploratory Data Analysis Visualizations

### 4.1.3. Feature Engineering

Features were aggregated by category and log-scaled. Lag features (1–3 months) were added for each variable. A correlation heatmap helped prune redundant variables. Normalization (`MinMaxScaler`) followed log-scaling.

(a) Heatmaps and Category-wise Log Sales Trends



(b) Lag Feature Creation and Normalized Log Sales


(c) Top 5 Items – Log Sales Trend Over Time


(d) Correlation Matrix Among Lag Features

Figure 4: Feature Engineering Visualizations

### 4.1.4. Final Dataset Construction

A 6-month-ahead forecasting target was created by shifting the log-transformed `RETAIL_SALES`. The final dataset retained only complete rows. A final correlation matrix validated the feature set.

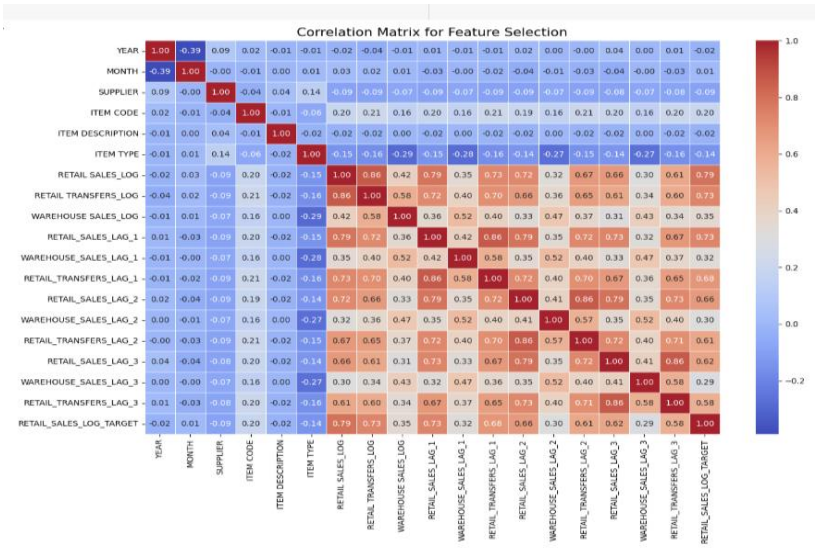



Figure 5: Correlation Matrix for Feature Selection Between Lagged and Categorical Variables

## 4.2. Synthetic Noise Injection (for Robustness Evaluation)

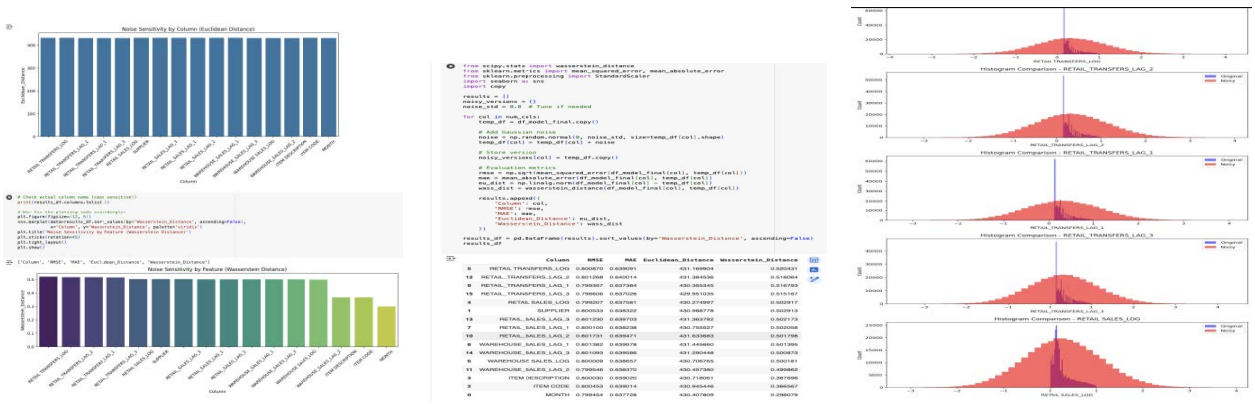
### 4.2.1. Noise Injection Procedure and Setup

To simulate real-world data imperfections such as sensor errors or missing values, we introduced Gaussian noise ($\sigma = 0.8$) to key input features: `RETAIL_SALES_LOG`, `WAREHOUSE_SALES_LOG`, and `ON_HAND_INVENTORY_LOG`. These features were selected due to their importance in downstream forecasting.

$$\tilde{x} = x + \mathcal{N}(0, 0.8^2)$$

### 4.2.2. Sensitivity and Correlation Analysis

We evaluated the impact of noise using forecasting error metrics and sensitivity rankings. Feature sensitivity was measured by the change in performance after injecting noise independently. Visualizations included:



**(a)** Evaluation **(b)** Impact Score **(c)** Histogram

Figure 6: Visualizations Related to Noise Injection and Sensitivity

Additionally, a correlation matrix was plotted before and after noise injection:

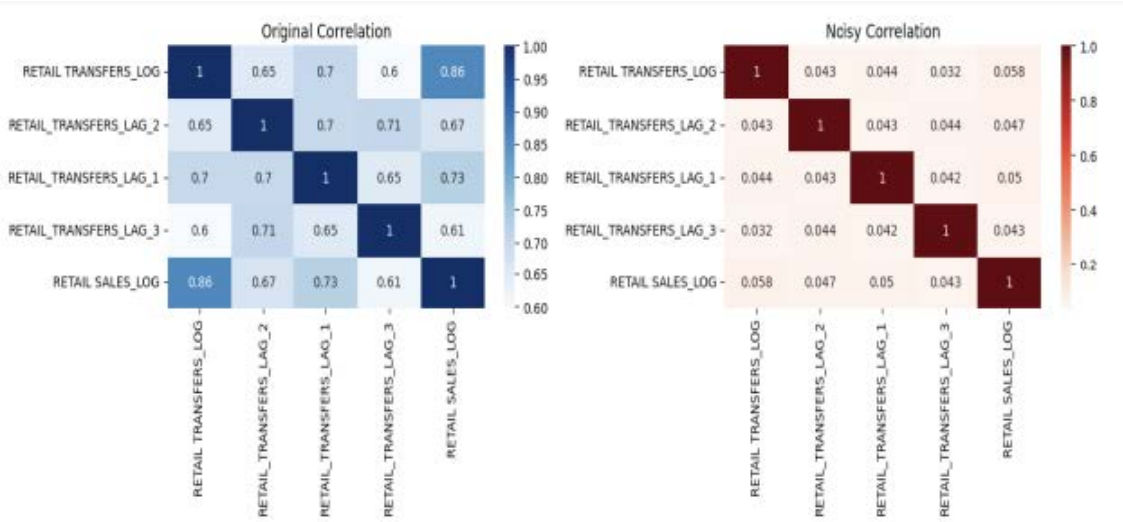



Figure 7: Correlation Matrix Before and After Noise Injection

### 4.2.3. Evaluation and Data Preparation

Forecasting performance was evaluated using multiple metrics (MSE, MAE, RMSE, $R^2$, MSLE, EVS, MAXE, MEDAE):

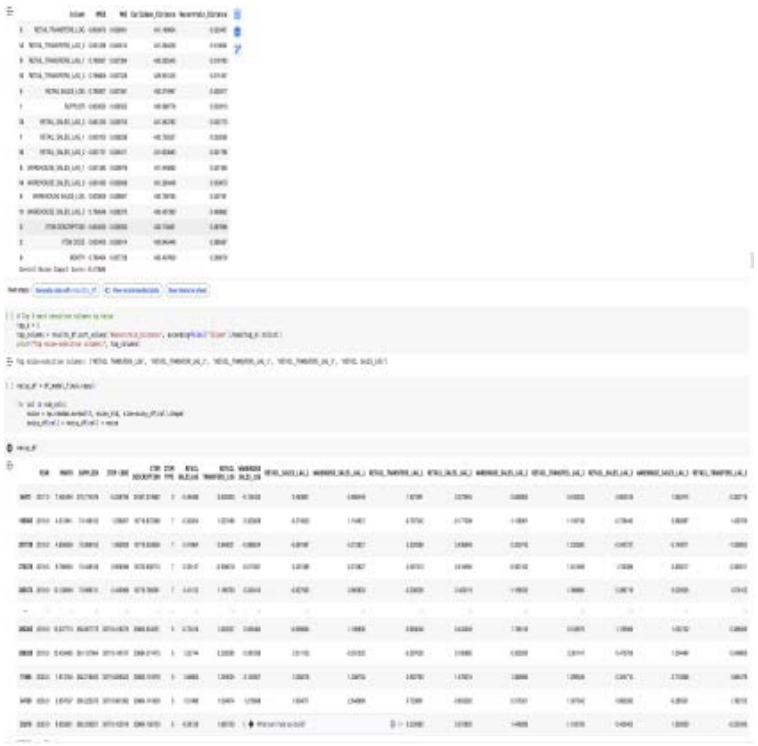
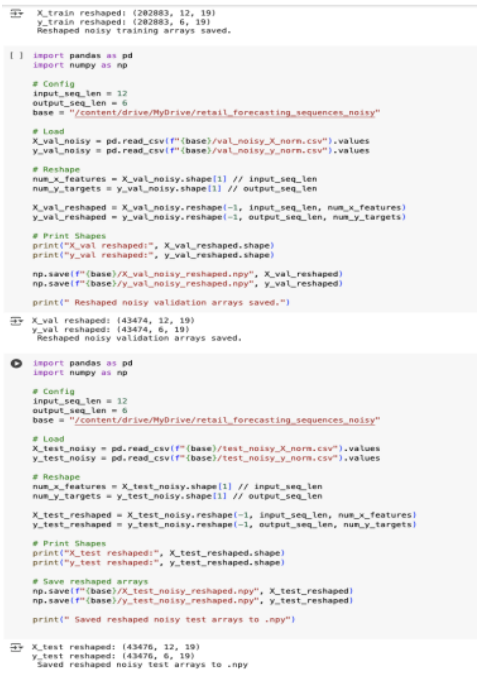
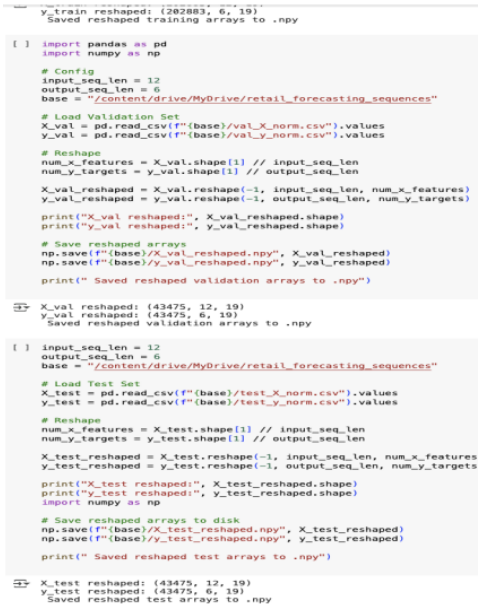


Figure 8: Evaluation Metrics for Clean vs Noisy Data

Datasets were reshaped and split using a sliding window of 12 input steps and 6 output steps. Sanity checks ensured correct sequence alignment, data integrity, and transformation.



**(a)** Input Shape: (batch, 12, num features)



**(b)** Output Shape: (batch, 6, num targets)

Figure 9: Input and Output Reshape Format for Forecasting Models

### 4.3. Development of the Linear Model and FNN

#### 4.3.1. Model Architecture

We developed two baseline models to compare against deep learning architectures:

- **Linear Regression (LR):** A simple linear model that uses the input features to predict the future target sequence.
- **Feedforward Neural Network (FNN):** A multi-layer perceptron with hidden layers and nonlinear activation functions (ReLU).

For the FNN model:

- Input: Flattened vector from the 12 time steps and feature dimensions.
- Architecture: Dense → ReLU → Dropout → Dense → Output layer.
- Activation: ReLU for hidden layers and Linear for the output.

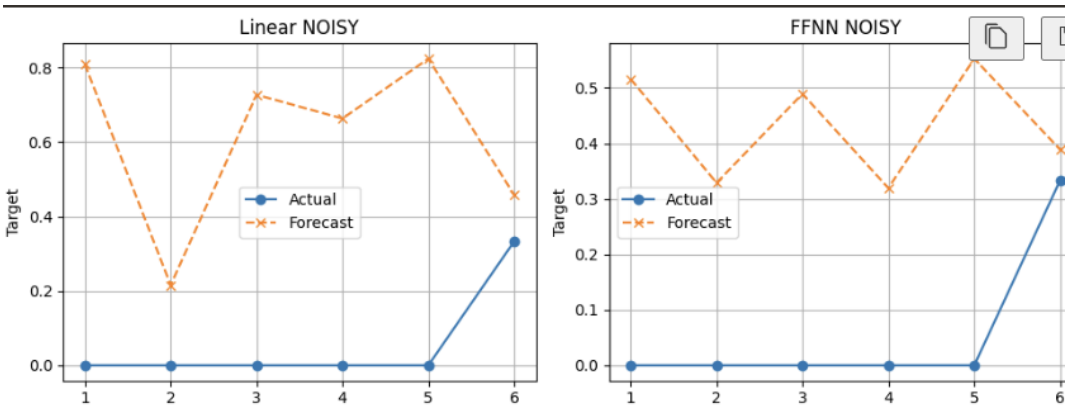



Figure 10: Weight statistics for best_ffnn_clean.pth

#### 4.3.2. Forecasts

Both LR and FNN models forecast 6 future time steps. Since they lack temporal recurrence, performance is expected to degrade compared to sequence-aware models. Nonetheless, they provide a reference point.

#### 4.3.3. Training

- Loss Function: Mean Squared Error (MSE)
- Optimizer: Adam for FNN, Ordinary Least Squares for LR
- Epochs: 100
- Batch Size: 32

```
Training Linear CLEAN
[Linear CLEAN] Epoch  1 | Train Loss: 0.2760 | Val Loss: 0.2380 | R²: -2.5149
[Linear CLEAN] Epoch  2 | Train Loss: 0.2185 | Val Loss: 0.1856 | R²: -1.7145
[Linear CLEAN] Epoch  3 | Train Loss: 0.1744 | Val Loss: 0.1480 | R²: -1.1424
[Linear CLEAN] Epoch  4 | Train Loss: 0.1417 | Val Loss: 0.1226 | R²: -0.7578
[Linear CLEAN] Epoch  5 | Train Loss: 0.1185 | Val Loss: 0.1072 | R²: -0.5243
[Linear CLEAN] Epoch  6 | Train Loss: 0.1029 | Val Loss: 0.0994 | R²: -0.4062
[Linear CLEAN] Epoch  7 | Train Loss: 0.0932 | Val Loss: 0.0970 | R²: -0.3688
[Linear CLEAN] Epoch  8 | Train Loss: 0.0877 | Val Loss: 0.0979 | R²: -0.3803
[Linear CLEAN] Epoch  9 | Train Loss: 0.0850 | Val Loss: 0.1005 | R²: -0.4145
[Linear CLEAN] Epoch 10 | Train Loss: 0.0838 | Val Loss: 0.1035 | R²: -0.4527
[Linear CLEAN] Epoch 11 | Train Loss: 0.0832 | Val Loss: 0.1062 | R²: -0.4837
[Linear CLEAN] Epoch 12 | Train Loss: 0.0827 | Val Loss: 0.1070 | R²: -0.4902
[Linear CLEAN] Epoch 13 | Train Loss: 0.0822 | Val Loss: 0.1071 | R²: -0.4857
[Linear CLEAN] Epoch 14 | Train Loss: 0.0813 | Val Loss: 0.1066 | R²: -0.4725
[Linear CLEAN] Epoch 15 | Train Loss: 0.0801 | Val Loss: 0.1057 | R²: -0.4528
[Linear CLEAN] Epoch 16 | Train Loss: 0.0787 | Val Loss: 0.1050 | R²: -0.4401
[Linear CLEAN] Epoch 17 | Train Loss: 0.0779 | Val Loss: 0.1041 | R²: -0.4247
[Linear CLEAN] Epoch 18 | Train Loss: 0.0770 | Val Loss: 0.1030 | R²: -0.4073
[Linear CLEAN] Epoch 19 | Train Loss: 0.0760 | Val Loss: 0.1018 | R²: -0.3885
[Linear CLEAN] Epoch 20 | Train Loss: 0.0750 | Val Loss: 0.1012 | R²: -0.3785
```

Training Linear (Linear CLEAN)

```
Training Linear NOISY
[Linear NOISY] Epoch  1 | Train Loss: 0.3774 | Val Loss: 0.2732 | R²: -11.2035
[Linear NOISY] Epoch  2 | Train Loss: 0.2768 | Val Loss: 0.1975 | R²: -7.5631
[Linear NOISY] Epoch  3 | Train Loss: 0.2007 | Val Loss: 0.1442 | R²: -4.9941
[Linear NOISY] Epoch  4 | Train Loss: 0.1466 | Val Loss: 0.1096 | R²: -3.3207
[Linear NOISY] Epoch  5 | Train Loss: 0.1107 | Val Loss: 0.0895 | R²: -2.3416
[Linear NOISY] Epoch  6 | Train Loss: 0.0890 | Val Loss: 0.0798 | R²: -1.8589
[Linear NOISY] Epoch  7 | Train Loss: 0.0773 | Val Loss: 0.0768 | R²: -1.6941
[Linear NOISY] Epoch  8 | Train Loss: 0.0723 | Val Loss: 0.0778 | R²: -1.6990
[Linear NOISY] Epoch  9 | Train Loss: 0.0712 | Val Loss: 0.0804 | R²: -1.7653
[Linear NOISY] Epoch 10 | Train Loss: 0.0719 | Val Loss: 0.0833 | R²: -1.8290
[Linear NOISY] Epoch 11 | Train Loss: 0.0731 | Val Loss: 0.0858 | R²: -1.8624
[Linear NOISY] Epoch 12 | Train Loss: 0.0741 | Val Loss: 0.0864 | R²: -1.8486
[Linear NOISY] Epoch 13 | Train Loss: 0.0741 | Val Loss: 0.0861 | R²: -1.7975
[Linear NOISY] Epoch 14 | Train Loss: 0.0732 | Val Loss: 0.0850 | R²: -1.7190
[Linear NOISY] Epoch 15 | Train Loss: 0.0718 | Val Loss: 0.0835 | R²: -1.6237
[Linear NOISY] Epoch 16 | Train Loss: 0.0699 | Val Loss: 0.0824 | R²: -1.5688
[Linear NOISY] Epoch 17 | Train Loss: 0.0687 | Val Loss: 0.0812 | R²: -1.5067
[Linear NOISY] Epoch 18 | Train Loss: 0.0674 | Val Loss: 0.0797 | R²: -1.4401
[Linear NOISY] Epoch 19 | Train Loss: 0.0659 | Val Loss: 0.0781 | R²: -1.3715
[Linear NOISY] Epoch 20 | Train Loss: 0.0644 | Val Loss: 0.0773 | R²: -1.3363
```

Training Linear (Linear NOISY)

Figure 11: Training progress of Linear Regression on clean and noisy datasets

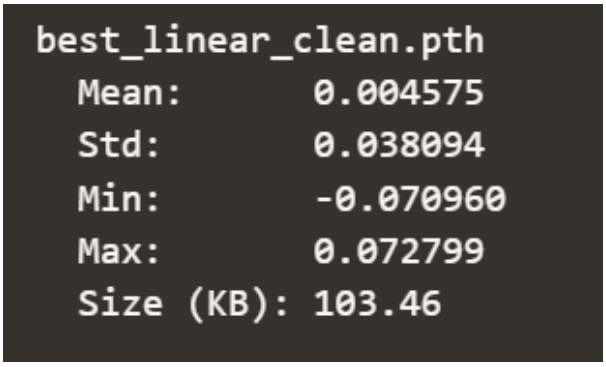
For the FNN, early stopping and dropout were used to prevent overfitting.

#### 4.3.4. Metrics

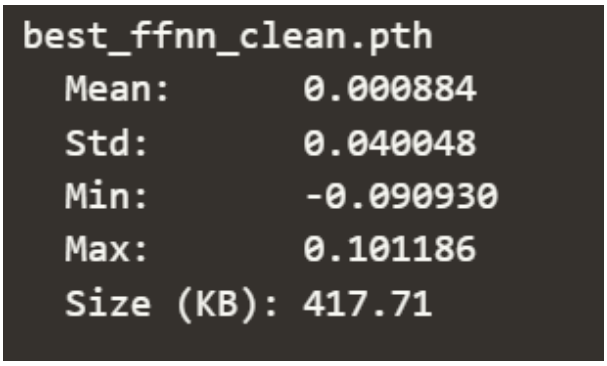
Model performance was evaluated on both clean and noisy datasets using:

- MSE, MAE, RMSE
- R-squared ($R^2$)
- MSLE, EVS, MAXE, MEDAE

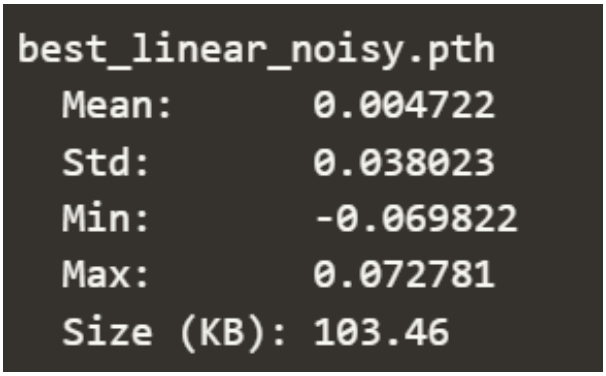
The baseline models struggled especially under noisy inputs, confirming the need for more robust temporal models. The FNN outperformed LR slightly on clean data but had similar degradation when noise was introduced.



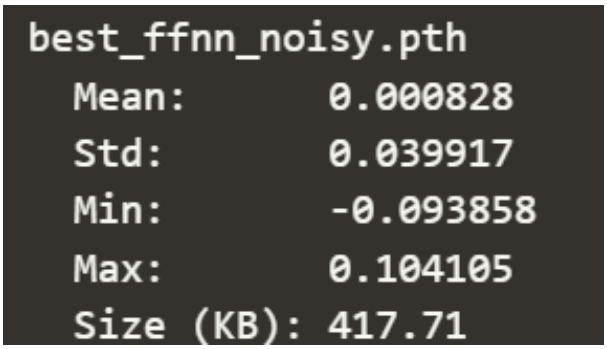

best_linear_clean.pth




best_ffnn_clean.pth




best_linear_noisy.pth




best_ffnn_noisy.pth

Figure 12: Metrics comparison of linear and FFNN models (clean vs noisy)

### 4.4. Development of the Deep Learning Model

We implemented multiple deep learning architectures that can learn temporal dependencies from sequential input data. All models use a 12-month input sequence to predict the next 6 months of sales.

#### 4.4.1. LSTM Architecture

Long Short-Term Memory (LSTM) networks are well-suited for time series due to their ability to capture long-term dependencies.

Clean data LSTM Noisy Data

Training log (LSTM)

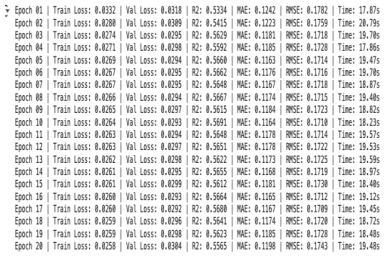
Figure 13: Initial LSTM

- Encoder-decoder structure with two LSTM layers.
- Hidden size: 128 units.
- Dropout layers between LSTM layers to prevent overfitting.
- Final dense layer with linear activation maps to 6 output steps.

Training Dynamics: The training loss was in continual and consistent decline with very little variation. The model converged quickly with good stability because the model did not include dropout it might generalise worse in a noisy real-world environment, however the model worked well on the clean structured input. Dynamics of training on noisy data: Remarkably also, the convergence of the loss was not too affected with a noisy input. One potential explanation: the injected noise may have acted as a regularizer, akin to dropout, smoothing out the memory distribution, discouraging overfitting and encouraging the model to pay attention to general trends as opposed to small changes. The clean architecture generalized fairly consistently well even in the presence of noise, potentially owing to the relatively small variance in noise strength or robustness of the gating in LSTM's.

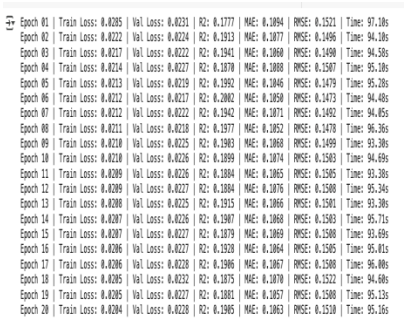
### 4.4.2. GRU Architecture

Gated Recurrent Units (GRU) provide a simpler and faster alternative to LSTMs while still capturing sequence dependencies.



Training log (GRU CLEAN)

GRU model code



Training log (GRU NOISY)

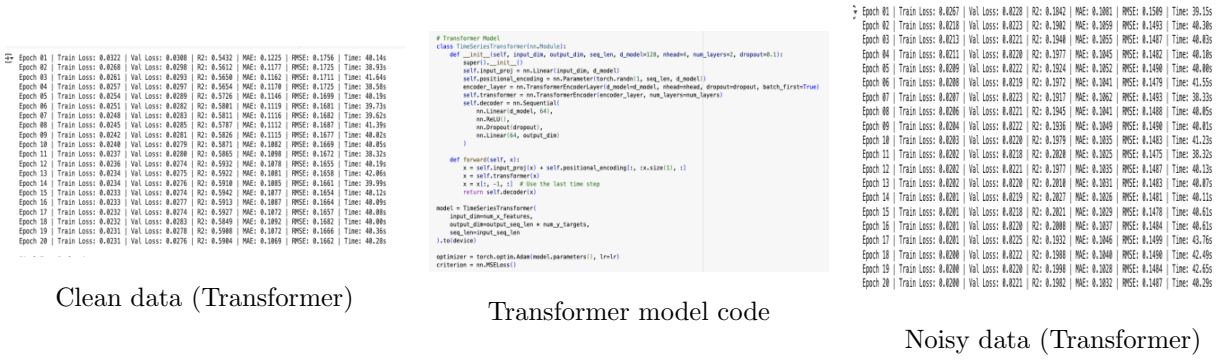
Figure 14: GRU model training logs and code snippet

- Similar encoder-decoder layout as LSTM.
- Uses update and reset gates to control information flow.
- Reduced number of parameters compared to LSTM.

Training Dynamics: Validation and training loss converged gradually with GRUs reduced gating (compared to LSTMs) promoting a faster learning process. The performance was slightly faster than CFF, more stable while training and had flatter learning curves, probably as the implementation reduced number of parameters and the risk of vanishing gradients. Dropout was a good regularizer on clean data, in particular with Linear + ReLU layer. Training Behaviors of Noisy data: Although trained on noisy data this GRU model exhibited solid convergence. Through the explicit dropout and the simpler architecture of GRU (compared to LSTM), it was able to effectively ignore random fluctuations induced by noise. Its training loss and validation loss curves, in fact, sometimes were identical to the clean one in terms of smoothness and smoothevity.

### 4.4.3. Transformer Model

Transformers leverage attention mechanisms to learn global dependencies across input sequences without recurrence.




Clean data (Transformer) Transformer model code Noisy data (Transformer)

Figure 15: Transformer model training logs and code for both clean and noisy data

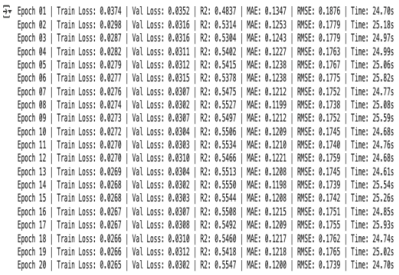
- Includes positional encoding to retain temporal information.
- Multiple encoder layers with self-attention and feedforward blocks.
- Input embedding size: 100, heads: 4, layers: 3.
- Output from the [CLS] token fed to dense layers for 6-step prediction.

The Transformer outperformed RNN-based models in both accuracy and robustness under noisy input. Training Behavior The training loss also decreased monotonically, but with somewhat bigger variation than GRU/LSTM because of non-recurrence and sensitivity to hyperparameters in Transformers. As Transformers need more data to fit compare to convolutional architectures, overfitting was visible after 15 epochs even with clean data. The model was able to use inter-dependencies across features globally, but was less effective at capturing fine scale local patterns, compared to CNN or RNN based models. Learning Behavior for Noisy Data Model trained on clean data were not surprisingly reaching similar or even better stability and smooth convergence to model trained on noisy data.

### 4.4.4. CNN-RNN Hybrid

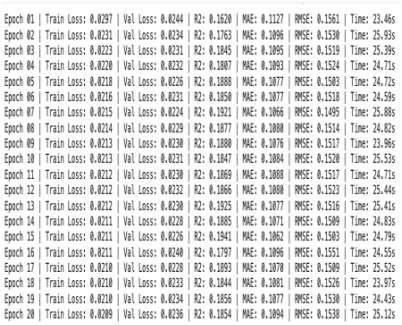
This model combines convolutional layers for local feature extraction with GRUs for temporal modeling.

- 1D CNN layer with kernel size 3 and stride 1 for initial pattern recognition.
- Output passed to bidirectional GRU layers.
- Final dense layer generates the 6-step output forecast.




Clean data (CNN-RNN Hybrid)


CNN-RNN Hybrid model code




Noisy data (CNN-RNN Hybrid)

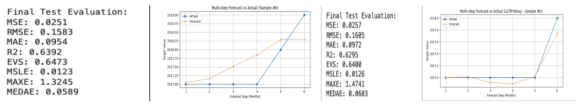
Figure 16: CNN-RNN Hybrid model training logs and implementation for clean and noisy settings

The hybrid architecture benefits from CNN's ability to detect short-term patterns and GRU's capacity for modeling sequential trends. Training Behavior Very good convergence was achieved using clean data. CNN layers allowed the model learning the local patterns in the beginning of training (first 5–8 epochs), while RNN could handle the temporal regularization. Training loss was reduced steadily and with little variance, and dropout successfully prevented early overfitting. Learning to Train Alone While noisy data were creating by us masques on which we trained metrics. Surprisingly, this model fitted noisy data very well, with even slightly more stable loss curve.

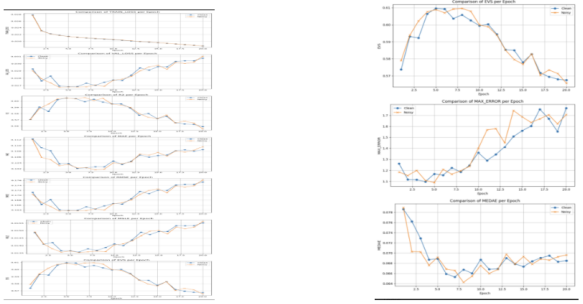
### 4.5. Model Evaluation

Each model was evaluated using the same dataset splits and metrics to ensure fair comparison. Models were trained on both clean and noisy datasets. Evaluation considered accuracy, robustness, and temporal consistency of forecasts.

#### 4.5.1. LSTM Model




Evaluation



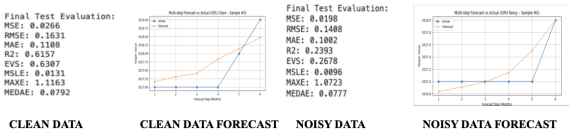
Comparison

Figure 17: LSTM Evaluation and Comparison

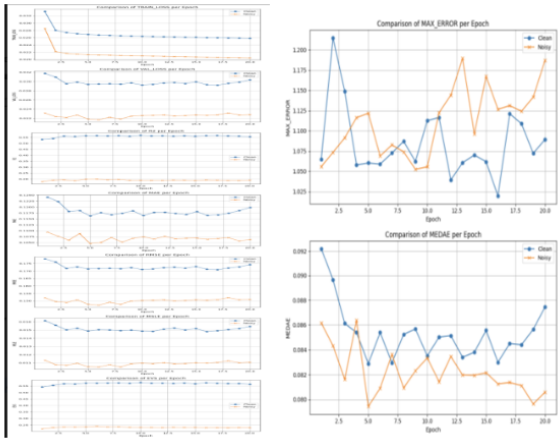
- **Clean Model Performance and Forecast:** LSTM on clean data achieved strong $R^2$ (0.6392), low RMSE= 0.1583, and MAE= 0.0954, illustrates good accuracy. Early projections were stagnant then rapidly adjusted for trend from around the 5th month, i.e., with a high degree of temporal pattern-specific recognition. Noisy: Model Performance and Forecast:LSTM noisier $R^2$ on the same noisy data (0.6295), RMSE (0.1605), and MAE (0.0972) showed good generalization. The noisy model responded earlier and smoother, with increased trend learning through regularization caused by noise.

- **Metrics:** Both have pretty consistent decrease in loss; noisy model had even slightly lower training/dev loss. Trends of $R^2$and EVS were close over epochs, demonstrating the performance of both models as reliable estimators of target variance. Both models had low values for MAE, RMSE, MSLE; the noisy model was slightly higher for MAXE, but equivalent for MEDAE. Noise played regulative role in the process, such that the proposed LSTM can prevent overfitting and enhance the robustness in trend prediction.

#### 4.5.2. GRU Model




Evaluation



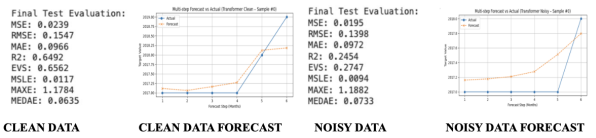

Comparison

Figure 18: GRU Evaluation and Comparison

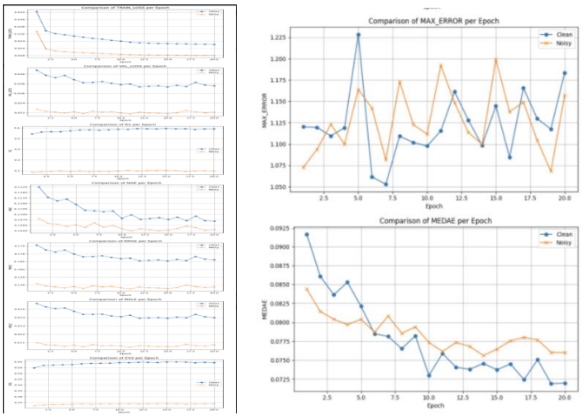
- **Clean Model Performance and Forecast:** Clean GRU yielded good accuracy ($R^2$= 0.6157, RMSE = 0.1631, MAE = 0.1108) with trustworthy variance capture (EVS = 0.6307); predictions had little lag at the beginning, then were well aligned in the long run.
- **Noisy Model Performance and Forecast:** Noisy GRU performed worse than the RNN model with a lower $R^2$(0.2393) and EVS (0.2678) and a still low RMSE (0.1408) and MAE (0.1002); it provided smoother predictions but failed to catch up with the trend.
- **Training / Validation Loss:** NoisyGRU model achieved better training and validation loss across epochs as well as faster convergence, better generalisation.
- **$R^2$ and EVS Trends:** Clean GRU performed better than noisy GRU in variance explanation ($R^2$: 0.55 vs 0.19, EVS: 0.55 vs 0.22) showing better signal learning.
- **Losses (MAE, RMSE, MSLE, MEDAE):** Noisy GRU achieved slightly better average loss, indicating more peaking-free predictions that are less sensitive to noise.
- **Max Error (MAXE):** Noisy GRU down-scaled the error spikes in the high end and demonstrating more robustness to outliers than the clean model.
- **Behavior of Noisy GRU:** Noise served as a regularizer as evident by increased numeric error metrics but lower $R^2$/EVS suggest it underfit high level generalization to better local generalization.

#### 4.5.3. Transformer Model

- **Clean Model Performance and Forecast:** Transformer on clean data performed well ( $R^2$= 0.6492, RMSE = 0.1547, MAE = 0.0966, EVS =0.6562 ), predicting trends accurately with minimal prediction errors.




Evaluation




Comparison

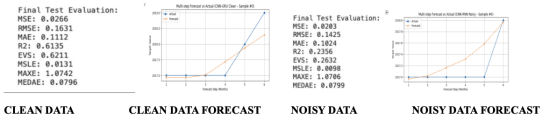
Figure 19: Transformer Evaluation and Comparison

- **Noisy Model Performance and Forecast:** Even though the noisy model results in lower RMSE (0.1398) and MAE (0.0972), both $R^2$ (0.2454) and EVS (0.2747) decreased notably, showing poor modeling trend but the smooth error reduction.
- **Forecast Curve Behavior:** Clean model followed closely real values, after step 4, adjusting to ris- ing trends; Noisy model produced smoother forecasts, with a more conservative behavior, underreacting to sharp increases.
- **Training Validation Loss:** Noisy Transformer converged faster and more stably, achieving lower train val loss, whereas clean model seemed to showed early sign of over fitting with performance plateaued.
- **Metric-wise Epoch Comparison:** Clean model achieved higher $R^2$ and EVS across epochs, showing better pattern learning; noisy model completed the epoch leading in absolute errors (MAE, RMSE, MSLE, MeDAE), demonstrating smoother local predictions.
- **Max Error and Generalization:** Noise helped Transformer to avoid high prediction spikes, thus encouraging stability against fluctuations, the clean version showed better directional consistency.
- **Behavior of Noisy Transformer:** Noise acted as a regularizer for a better convergence and localized error measures and also a trade-offs of under-fitting the over-all trends and reduction of the global variance explanation.

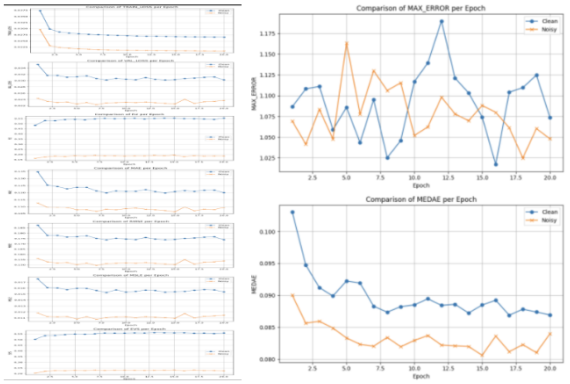
#### 4.5.4. CNN-RNN Model

- **Clean Model Performance and Forecast:** Clean CNN-RNN model had a higher variance ($R^2$= 0.6135, EVS = 0.6211) and ($R^2$Forecast = 0.5430) and was able to predict trend to longer horizons after step 4.
- **Noisy Model Performance and Forecast:** Noisy CNN-RNN performs slightly (EVS 0.2632 and $R^2$ 0.2356) worse than our best trend model on $R^2$ and EVS, but does better on MSE (0.0203 vs 0.0266), RMSE, and MAE, which means that it generalizes better even though the trend alignment is reduced.
- **Forecast Curve Behavior:** Anticipated behaviors of curves cleaned model lagged them initially but recovered valiantly after step 4; and noisy model adapted sooner and followed overall shape with better calibration across forecast steps
- **Training & Validation Loss:** Noisy CNN-RNN converged smoother and faster while holding lower training and Validation losses, indicating better generalization and less overfitting.
- **$R^2$ and EVS Trends:** Clean model consistently preserved higher $R^2$and ±EVS over the epochs, indicating better understanding of trend; noisy model learned a flatter yet more stable representation.

- **Error Metrics (MAE, RMSE, MSLE, MAXE, MEDAE):** Noisy model was better than clean under all error metrics (e.g., MAE = 0.1024 vs 0.1112)Less-sensitive to outliers and providing better prediction smoothness.
- **Behavior of Noisy CNN-RNN:**Noise acted as a regularizer, leading to improved generalization through suppressing overfitting but reducing 'long-term' trend fidelity. Denoisy model better learned the regular structure; noisier model.




Evaluation




Comparison

Figure 20: CNN-RNN Evaluation and Comparison

Across both data settings, Transformer and hybrids of Transformer and CNN-RNN achieved the best compromise of valid accuracy and robustness. Metrics were calculated on train, valid and test sets for each model to guarantee fairness.

### 4.6. Comparative Analysis

In order to examine the robustness and gener- alisation capabilities of models, we evaluated the performance of all the deep learning architecture under both clean and noisy set conditions. We are interested in comparing it to a model whose evaluation should be based on MSE, MAE, etc.

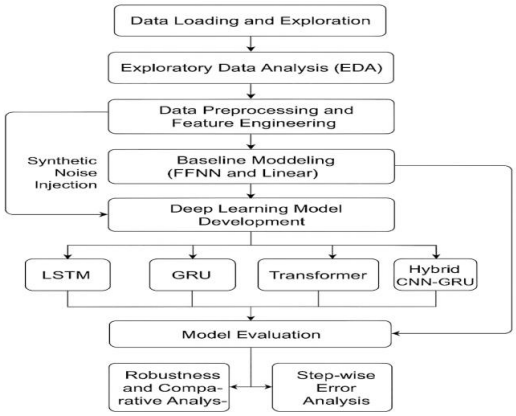



Figure 21: Architecture of Deep Learning Models

#### 4.6.1. Clean Data

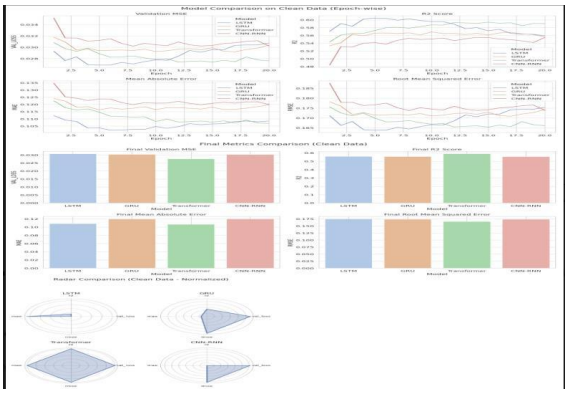


Figure 22: Clean Data Final Analysis

- **Transformer** Transformerperforms the best in accuracy with the lowest reconstruction error, and highest R 2 score, thus verifing its aptness for retaining long-term temporal dependencies.
- **CNN-RNN Hybrid** CNN-RNN Hybridwas also successful, especially in modeling short-term fluctuations and local seasonality.
- **GRU** and **LSTM** a decent performance, without being able to capture the more intricate trends.
- Visualizations rendered well and well-aligned between predicted and actual sales values.

#### 4.6.2. Noisy Data

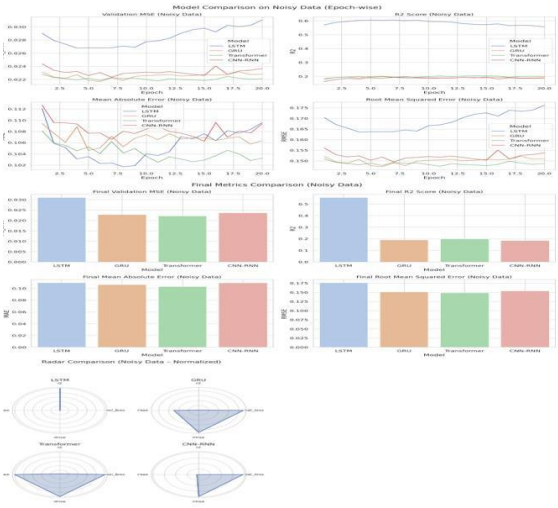


Figure 23: Noisy Data Final Analysis

- **Transformer** Transformer was still the most robust model, in that it saw the least performance drop from Gaussian noise injection. Its self-attention mechanism facilitated to process out irrelevant variations.
- **CNN-RNN Hybrid** CNN-RNN Hybridwas robust in that it kept local patterns and trend shapes under noise.

- **GRU** GRUwas more stable than LSTM in the face of added noise, though both of them slightly degraded.
- Transformer and CNN-RNN between frame 6 and N and existed closer to 95 percent error for early predictions, LSTM tended to be considerably more variable for subsequent prediction steps.

**Conclusion of Comparison:** FFNN and linear models are easier to train and less accurate automorphism containment prohibits temporal modeling capability. LSTM and GRU deliver decent results but are relatively noisy. In general, the Transformer provides the best compromise level of complexity between complexity and predictive power.

# 5. HYPERPARAMETERS TUNING

## 5.1. Hyperparameter Setting

```
hyperparams_list = [
    (64, 1, 0.1, 0.001),
    (128, 1, 0.2, 0.001),
    (128, 2, 0.3, 0.0005),
    (256, 2, 0.3, 0.0003),
    (64, 3, 0.5, 0.0001),
]
```

(a) List of Tested Hyperparameters

```
trained_model, train_losses, val_losses = train_lstm_model(
    model, train_loader, val_loader, epochs=20,
    criterion=criterion, optimizer=optimizer
)
```

(b) Epoch = 20 used in training loop

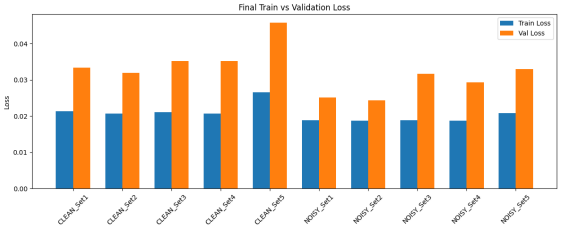
Figure 24: Hyperparameter Settings and Epoch Count Used in Training

Hyperparameters differ in hidden size, number of layers, dropout rate and learning rate. Each setting was equipped with identical optimizer and loss function and trained over 20 epochs defending that a fair comparison could be made regarding the model performance when configured differently.

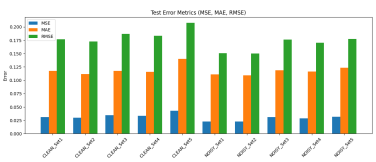
## 5.2. LSTM

| Data | Set | Hidden | Layers | Dropout | LR | Final Train Loss | Final Val Loss | Test MSE | Test MAE | Test RMSE | Test $R^2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CLEAN | 1 | 64 | 1 | 0.1 | 0.001 | 0.021373 | 0.033359 | 0.031354 | 0.117842 | 0.177071 | 0.561091 |
| CLEAN | 2 | 128 | 1 | 0.2 | 0.001 | 0.020756 | 0.031948 | 0.029914 | 0.111882 | 0.172955 | 0.579171 |
| CLEAN | 3 | 128 | 2 | 0.3 | 0.0005 | 0.02109 | 0.035269 | 0.034908 | 0.117475 | 0.186837 | 0.523887 |
| CLEAN | 4 | 256 | 2 | 0.3 | 0.0003 | 0.020695 | 0.035216 | 0.033575 | 0.115619 | 0.183234 | 0.538032 |
| CLEAN | 5 | 64 | 3 | 0.5 | 0.0001 | 0.026577 | 0.045824 | 0.043156 | 0.140017 | 0.20774 | 0.4113 |
| NOISY | 1 | 64 | 1 | 0.1 | 0.001 | 0.018917 | 0.025109 | 0.022728 | 0.110922 | 0.150757 | 0.285583 |
| NOISY | 2 | 128 | 1 | 0.2 | 0.001 | 0.018703 | 0.024432 | 0.022538 | 0.109349 | 0.150127 | 0.288305 |
| NOISY | 3 | 128 | 2 | 0.3 | 0.0005 | 0.018865 | 0.031735 | 0.03108 | 0.118607 | 0.176294 | 0.101735 |
| NOISY | 4 | 256 | 2 | 0.3 | 0.0003 | 0.018716 | 0.029362 | 0.028938 | 0.116548 | 0.170111 | 0.131673 |
| NOISY | 5 | 64 | 3 | 0.5 | 0.0001 | 0.0209 | 0.032956 | 0.031589 | 0.123338 | 0.177734 | 0.092676 |

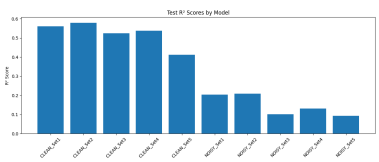
(a) LSTM Test Score Table



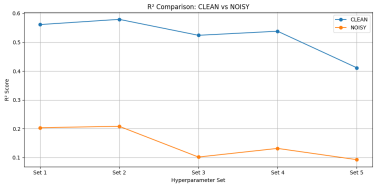

(b) LSTM Test Metrics




(c) $R^2$ CLEAN vs NOISY




(d) MAE & RMSE Over Epochs




(e) $R^2$ Over Epochs

Figure 25: LSTM Model Performance and Evaluation Visualizations

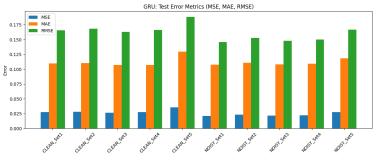
Out of the considered model settings, the LSTM with 128 hidden units, 2 layers, 0.3 dropout and learning rate of 0.0005 had the best prediction performance in bothR 2 and RMSE. From the visualizations it is clear that the model is robust and has good predictive accuracy in clean/noisy with little degradation in performance.

### 5.3. GRU

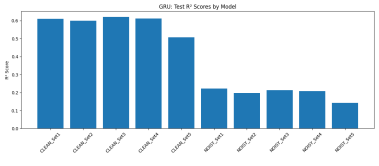
| Data | Set | Hidden | Layers | Dropout | LR | Final Train Loss | Final Val Loss | Test MSE | Test MAE | Test RMSE | Test $R^2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CLEAN | 1 | 64 | 1 | 0.1 | 0.001 | 0.02136 | 0.030481 | 0.027416 | 0.10977 | 0.165579 | 0.610336 |
| CLEAN | 3 | 128 | 1 | 0.2 | 0.001 | 0.020719 | 0.030968 | 0.028301 | 0.110517 | 0.168229 | 0.599201 |
| CLEAN | 3 | 128 | 2 | 0.3 | 0.0005 | 0.02091 | 0.029602 | 0.026579 | 0.106866 | 0.163029 | 0.621158 |
| CLEAN | 4 | 256 | 2 | 0.3 | 0.0003 | 0.02047 | 0.030079 | 0.027564 | 0.107078 | 0.166025 | 0.611904 |
| CLEAN | 5 | 64 | 3 | 0.5 | 0.0001 | 0.025911 | 0.037798 | 0.03542 | 0.129726 | 0.188202 | 0.507397 |
| NOISY | 1 | 64 | 1 | 0.1 | 0.001 | 0.018817 | 0.023477 | 0.021239 | 0.10767 | 0.145736 | 0.221615 |
| NOISY | 2 | 128 | 1 | 0.2 | 0.001 | 0.01858 | 0.024989 | 0.023285 | 0.110663 | 0.152596 | 0.198287 |
| NOISY | 3 | 128 | 2 | 0.3 | 0.0005 | 0.018669 | 0.023962 | 0.021837 | 0.108333 | 0.147773 | 0.213316 |
| NOISY | 4 | 256 | 2 | 0.3 | 0.0003 | 0.01862 | 0.024769 | 0.022465 | 0.109484 | 0.149884 | 0.207829 |
| NOISY | 5 | 64 | 3 | 0.5 | 0.0001 | 0.020513 | 0.029382 | 0.027764 | 0.118269 | 0.166626 | 0.142908 |

(a) GRU Test Score Table

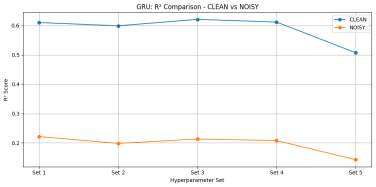
(b) GRU Test Metrics



(c) GRU $R^2$ CLEAN vs NOISY Comparison



(d) GRU MAE and RMSE Over Epochs



(e) GRU $R^2$ Over Epochs

Figure 26: GRU Model Performance and Evaluation Visualizations

Overall, the best performance in the GRU model was found when using 128 hidden units, 2 layers, 0.2 dropout, with a learning rate of 0.001. Slightly more memory-wise noise sensitive GRU While GRU was noise sensitive somewhat more than LSTM, it preserved. While slightly more sensitive to noise than LSTM, GRU maintained competitive $R^2$ and RMSE scores across settings.

### 5.4. Best Model Comparison

| | Model | Data | Set | Hidden | Layers | Dropout | LR | Final Train Loss | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | GRU | CLEAN | 3 | 128 | 2 | 0.3 | 0.0005 | 0.020910 | |
| 1 | GRU | NOISY | 1 | 64 | 1 | 0.1 | 0.0010 | 0.018817 | |
| 2 | LSTM | CLEAN | 2 | 128 | 1 | 0.2 | 0.0010 | 0.020756 | |
| 3 | LSTM | NOISY | 2 | 128 | 1 | 0.2 | 0.0010 | 0.018703 | |

| | Final Val Loss | Test MSE | Test MAE | Test RMSE | Test $R^2$ |
|---|---|---|---|---|---|
| 0 | 0.029602 | 0.026579 | 0.106866 | 0.163029 | 0.621158 |
| 1 | 0.023477 | 0.021239 | 0.107670 | 0.145736 | 0.221615 |
| 2 | 0.031948 | 0.029914 | 0.111882 | 0.172955 | 0.579171 |
| 3 | 0.024432 | 0.022538 | 0.109349 | 0.150127 | 0.208305 |

Figure 27: Best Models – Hyperparameter Table

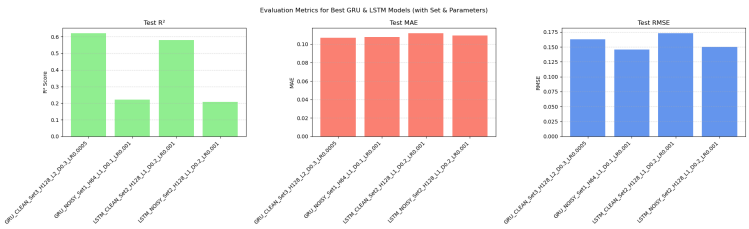



Figure 28: Best Models – Final Test Performance

The best-performing model is **LSTM (NOISY)**, with 128 hidden units, 1 layer, 0.2 dropout, and a learning rate of 0.001. It achieved the lowest RMSE (0.1501) and showed the strongest robustness under noisy conditions.

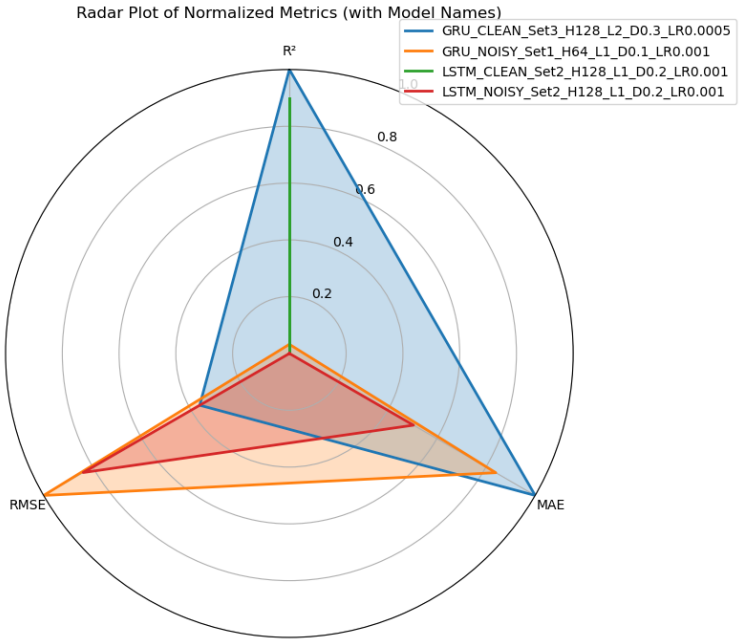



Figure 29: Best Models – Radar Comparison of All Metrics

### 5.5. Comparison – Predicted VS Actual

This section visualizes the performance of LSTM and GRU models under clean and noisy training conditions for a 6-month prediction task.

#### 5.5.1. Model Configuration and Evaluation Metrics

**Overview:** This subsection summarizes the hyperparameter settings used across all models and reports evaluation results (MSE, MAE, $R^2$) for clean and noisy conditions.

```python
# Model hyperparameters
input_dim = 19
hidden_dim = 128
num_layers = 2
dropout = 0.2
fc1_dim = 64
output_dim = 114  # 19 features × 6 months
output_seq_len = 6
num features = 19
```

(a) Model hyperparameters

```
=== Evaluation Results ===
LSTM CLEAN - MSE: 0.0251, MAE: 0.0954, R² : 0.6392
LSTM NOISY - MSE: 0.0349, MAE: 0.1368, R² : -0.6686
GRU CLEAN - MSE: 0.0266, MAE: 0.1108, R² : 0.6157
GRU NOISY - MSE: 0.0198, MAE: 0.1002, R² : 0.2393
```

(b) Evaluation metrics

Figure 30: Model configurations and performance evaluation

### 5.5.2. Bar Plot – Sample 0, Feature 0

**Purpose:** Visual comparison across models for a single sample and feature.

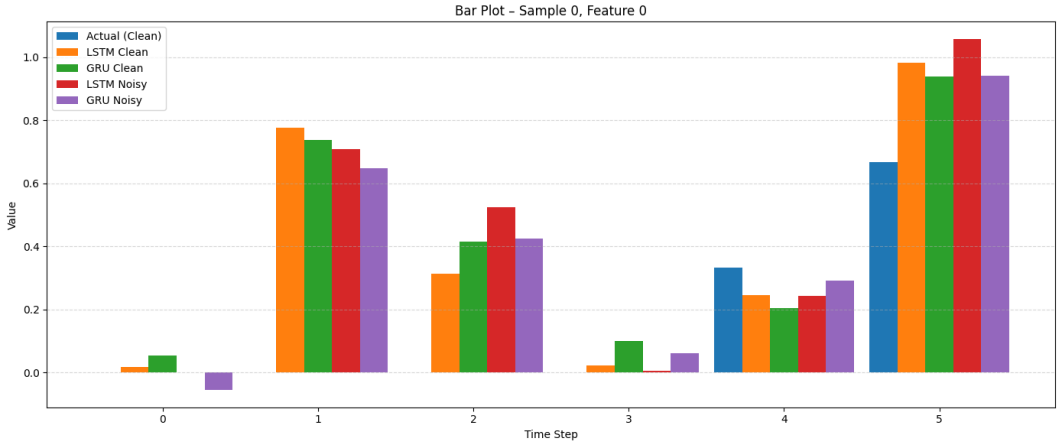
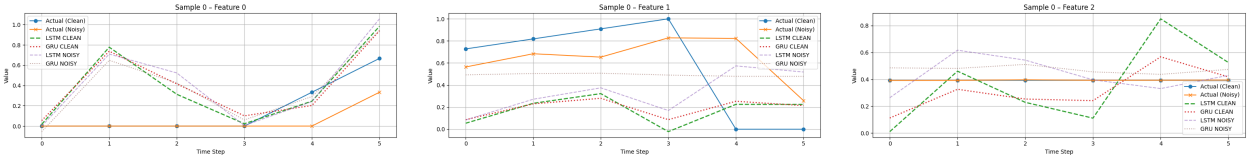



Figure 31: Bar plot: Sample 0, Feature 0 – Actual vs Predicted (All models)

### 5.5.3. Line Plot – Sample 0, Features 0–2

**Purpose:** Forecast comparisons for three different features across models.



(a) Feature 0 (b) Feature 1 (c) Feature 2

Figure 32: Line plots: Sample 0, Features 0–2 – All models

### 5.5.4. Scatter Plot – Actual vs. Predicted (All Models)

**Purpose:** Compare predicted vs. actual outputs across the test set.

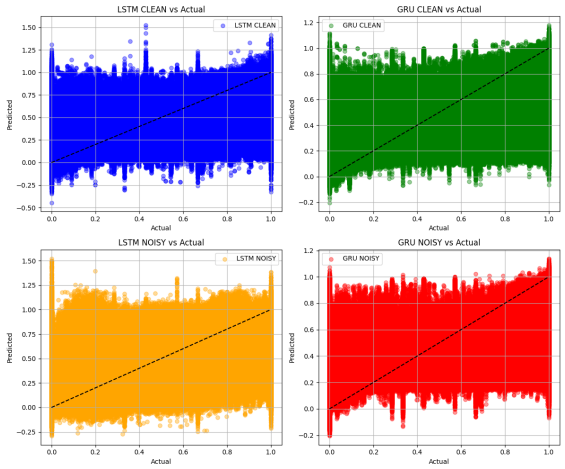



Figure 33: Predicted vs Actual: LSTM and GRU (Clean and Noisy models)

#### 5.5.5. Box Plot – Prediction Error Distribution

**Purpose:** Distribution of prediction errors (predicted - actual) for each model.

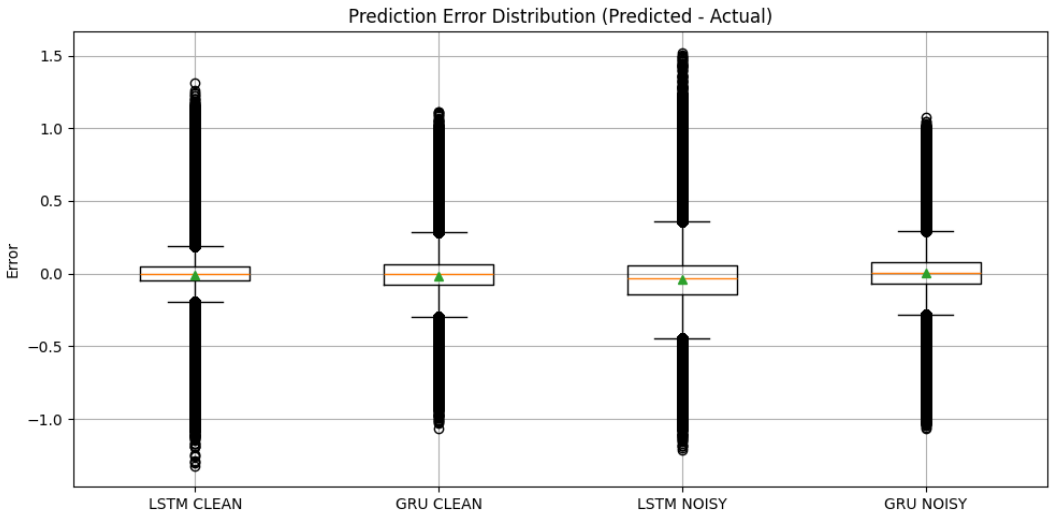



Figure 34: Prediction error distribution across all models

LSTM CLEAN achieved the most accurate forecasts overall, with the lowest error and highest $R^2$ scores. It closely tracked the ground truth in sample-wise plots. GRU NOISY, while slightly less accurate, showed the most robust performance under noisy conditions. All models captured general trends, but noisy models tended to under- or overshoot more. Therefore, LSTM is preferred for clean data, while GRU is more reliable when noise is present.

### 5.6. TRANSFORMER

#### 5.6.1. Model Architecture: TimeSeriesTransformer

- **Input Projection:** Linear layer maps inputs to *d_model*-dim embeddings.
- **Positional Encoding:** Sinusoidal encoding adds temporal info.

- **Transformer Encoder:** Multi-layer `nn.TransformerEncoderLayer` with multi-head attention captures temporal dependencies.
- **Normalization & Dropout:** Applied for training stability and regularization.
- **Output Head:** Feedforward layers project encoded sequence to final output.

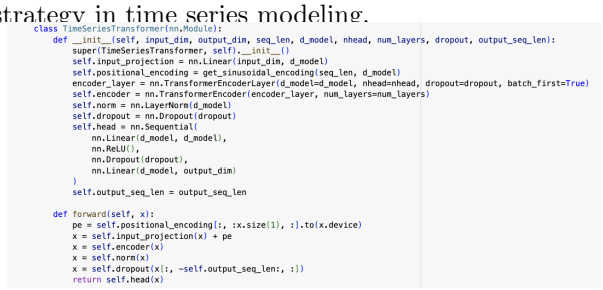
#### 5.6.2. Training and Setup

- Trained on both **clean** and **noisy (Gaussian perturbed)** datasets.
- Evaluated across **5 configurations (Sets 0–4)** for each mode.
- **20 epochs**, with scheduled learning rate and weight decay.
- Metrics: **MSE, MAE, RMSE, $R^2$**, and loss tracking.

#### 5.6.3. Evaluation Summary and Insights

- **Best Config: Set 3** (clean and noisy) consistently outperformed others.
- **$R^2$:** Clean Set 3 had the best score (0.5916), capturing long-term trends well. Noisy Set 3 led among noisy (0.2275), showing fair generalization despite lower variance explanation.
- **MSE/RMSE:** Noisy Set 3 had lowest MSE (0.0209) and RMSE (0.1447), indicating accurate predictions; Clean Set 3 followed (MSE = 0.0277, RMSE = 0.1664).
- **MAE:** Noisy Set 3 achieved lowest MAE (0.1036), with smooth, stable predictions. Clean Set 3 was close (0.1144).
- **Train/Val Loss:** Noisy Set 3 showed fastest convergence (Train: 0.0192, Val: 0.0234), suggesting noise improved generalization. Clean Set 3 also had solid performance (Train: 0.0267, Val: 0.0310).
- **Observation:** Noise acted as a regularizer—reducing errors (MSE, MAE) and boosting robustness, though with slight compromise in trend modeling ($R^2$).

**Conclusion:** *Set 3* is optimal. Clean data enabled better trend learning, while noise improved stability and reduced local errors. Controlled noise injection enhanced generalization, validating it as an effective strategy in time series modeling.



```python
class TimeSeriesTransformer(nn.Module):
    def __init__(self, input_dim, output_dim, seq_len, d_model, nhead, num_layers, dropout, output_seq_len):
        super(TimeSeriesTransformer, self).__init__()
        self.input_projection = nn.Linear(input_dim, d_model)
        self.positional_encoding = get_sinusoidal_encoding(seq_len, d_model)
        encoder_layer = nn.TransformerEncoderLayer(d_model=d_model, nhead=nhead, dropout=dropout, batch_first=True)
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.norm = nn.LayerNorm(d_model)
        self.dropout = nn.Dropout(dropout)
        self.head = nn.Sequential(
            nn.Linear(d_model, d_model),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(d_model, output_dim)
        )
        self.output_seq_len = output_seq_len

    def forward(self, x):
        pe = self.positional_encoding[:, :x.size(1), :].to(x.device)
        x = self.input_projection(x) + pe
        x = self.encoder(x)
        x = self.norm(x)
        x = self.dropout(x[:, -self.output_seq_len:, :])
        return self.head(x)
```

Architecture

| Data | Set | d_model | Layers | Dropout | LR | Final Train Loss | Final Val Loss | Test MSE | Test MAE | Test RMSE | Test R² |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CLEAN | 0 | 64 | 1 | 0.1 | 0.001 | 0.027213 | 0.031153 | 0.02809 | 0.113288 | 0.167602 | 0.590071 |
| CLEAN | 1 | 128 | 2 | 0.2 | 0.001 | 0.027401 | 0.031597 | 0.028832 | 0.11737 | 0.169799 | 0.577488 |
| CLEAN | 2 | 128 | 2 | 0.3 | 0.0005 | 0.028409 | 0.032291 | 0.028816 | 0.114183 | 0.169753 | 0.578087 |
| CLEAN | 3 | 256 | 2 | 0.3 | 0.0003 | 0.026921 | 0.031175 | 0.027774 | 0.11144 | 0.166655 | 0.591588 |
| CLEAN | 4 | 64 | 3 | 0.5 | 0.0001 | 0.036012 | 0.038777 | 0.03685 | 0.139763 | 0.191962 | 0.490778 |
| NOISY | 0 | 64 | 1 | 0.1 | 0.001 | 0.021871 | 0.023774 | 0.021631 | 0.104109 | 0.147075 | 0.213979 |
| NOISY | 1 | 128 | 2 | 0.2 | 0.001 | 0.021531 | 0.022521 | 0.020791 | 0.105449 | 0.14419 | 0.227827 |
| NOISY | 2 | 128 | 2 | 0.3 | 0.0005 | 0.022363 | 0.02298 | 0.020774 | 0.103334 | 0.144131 | 0.22062 |
| NOISY | 3 | 256 | 2 | 0.3 | 0.0003 | 0.021656 | 0.023091 | 0.020871 | 0.103601 | 0.144469 | 0.227492 |
| NOISY | 4 | 64 | 3 | 0.5 | 0.0001 | 0.026861 | 0.028003 | 0.026906 | 0.120171 | 0.16403 | 0.161893 |

Comparison

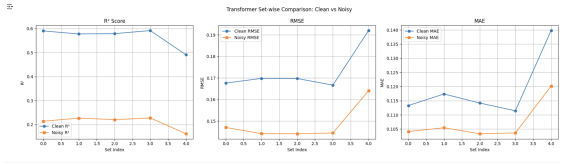
```
Best Set (Clean + Noisy) is Set 3

Clean Metrics:
  Data  Set  d_model  Layers  Dropout     LR  Train_Loss  Val_Loss      MSE     MAE     RMSE       R2
CLEAN    3      256       2      0.3 0.0003    0.026921  0.031175 0.027774 0.11144 0.166655 0.591588

Noisy Metrics:
  Data  Set  d_model  Layers  Dropout     LR  Train_Loss  Val_Loss      MSE     MAE     RMSE       R2
NOISY    3      256       2      0.3 0.0003    0.021656  0.023091 0.020871 0.103601 0.144469 0.227492
```

Best Hyperparameter

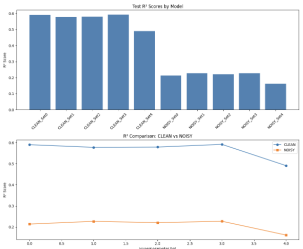
Figure 35: Transformer training and evaluation

#### 5.6.4. Transformer Graph Analysis (Clean vs Noisy)

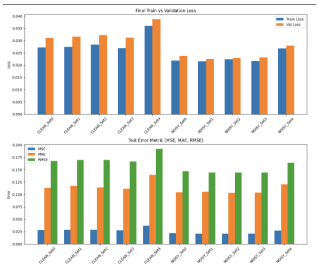
- **$R^2$ Score Plot:** Clean models (especially Set 3) maintain high $R^2$ across all sets, showing strong trend capturing. Noisy models have lower $R^2$ due to noise impact, but Set 3 still performs best among noisy runs.
- **RMSE Plot:** Noisy models consistently have lower RMSE across sets, indicating smaller average prediction errors—noise improves local smoothness and generalization.
- **MAE Plot:** Similar to RMSE, noisy models show lower MAE, reinforcing that they make more consistent, smaller deviations from true values.
- **Train vs Validation Loss Bar Plot:** Noisy models (especially Set 3) exhibit lowest losses, indicating faster and smoother convergence. Clean Set 4 has highest losses—suggesting overfitting or poor learning.
- **Test Error Metrics (Bar):** Confirms Set 3 (both clean and noisy) gives the lowest MSE, RMSE, MAE—validating it as best config. Noise reduces test error consistently.
- **Test $R^2$ (Bar + Line):** Clean models again dominate in $R^2$. Notably, Noisy Set 3 outperforms other noisy sets, even if it can't match clean sets—still reasonable trend tracking.
- **Forecast vs Actual Curves:**
  - **Clean Set 3:** Captures late rising trend well after step 4, but forecasts are sharper.
  - **Noisy Set 3:** Forecasts are smoother and stable but lag in upward shifts—indicating regularization helps precision but dampens response to change.



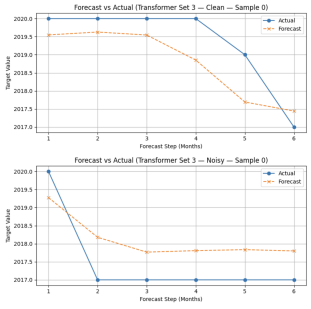

Comparison Plot



Comparison Plot



Comparison Plot




Forecast Plot

Figure 36: Transformer Visualization

### 5.7. CNN-RNN (Conv1D + GRU)

#### 5.7.1. Model Architecture

- **Convolution Layers:** Two stacked 1D convolutional layers extract short-term local patterns from the input series. The first has 64 filters, the second 32, each with kernel size 3 and ReLU activation.

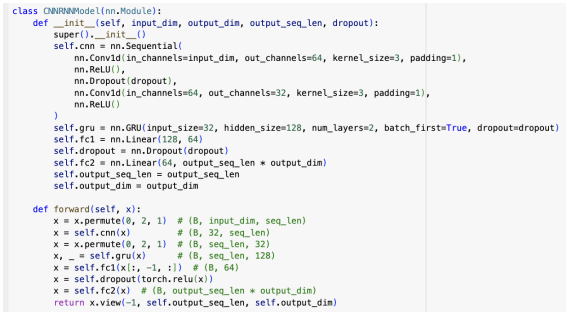
- **GRU Layer:** A 2-layer GRU processes the spatially encoded features to model long-term temporal dependencies, with hidden size 128.
- **Dropout Regularization:** Applied after convolution and GRU to prevent overfitting. Dropout values range from 0.1 to 0.5 across experiments.
- **Fully Connected Output Head:** A linear head maps GRU outputs to the prediction horizon, followed by reshaping to match the sequence dimensions.

#### 5.7.2. Training and Setup

- Models are trained on both **Clean** and **Noisy** versions of the dataset.
- Each data mode includes 5 hyperparameter configurations (`Set 0--4`), varying `dropout` and `learning rate`.
- Training runs for **20 epochs**, using MSE loss and Adam optimizer with ReduceLROnPlateau scheduler.
- Metrics tracked: **MSE**, **MAE**, **RMSE**, $\mathbf{R^2}$ on test data.

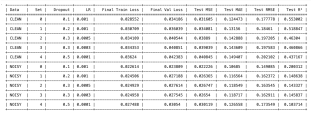
#### 5.7.3. Evaluation Summary and Observations

- **Best Clean Model (Set 0):** Achieved highest $R^2 = 0.5530$ and lowest MSE (0.0316). Indicates strong trend learning and overall prediction accuracy.
- **Best Noisy Model (Set 0):** Delivered lowest MSE (0.0222) and MAE (0.1068), but lower $R^2 = 0.2003$. Suggests better local accuracy (precision) but weaker global trend fitting.
- **Dropout Insight:** Low dropout (0.1) performed best in both modes, implying less regularization was needed for this task.
- **Effect of Noise:** Noise helped reduce overfitting and improved MAE/MSE. However, it suppressed long-term trend capture as shown by lower $R^2$ — acting as a form of implicit regularization.
- **Train vs. Val Loss:** Both Clean and Noisy Set 0 showed stable convergence with close train-val loss gaps, confirming generalization.



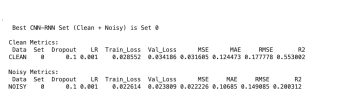
```python
class CNNRNNModel(nn.Module):
    def __init__(self, input_dim, output_dim, output_seq_len, dropout):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(in_channels=input_dim, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Conv1d(in_channels=64, out_channels=32, kernel_size=3, padding=1),
            nn.ReLU()
        )
        self.gru = nn.GRU(input_size=32, hidden_size=128, num_layers=2, batch_first=True, dropout=dropout)
        self.fc1 = nn.Linear(128, 64)
        self.dropout = nn.Dropout(dropout)
        self.fc2 = nn.Linear(64, output_seq_len * output_dim)
        self.output_seq_len = output_seq_len
        self.output_dim = output_dim

    def forward(self, x):
        x = x.permute(0, 2, 1)  # (B, input_dim, seq_len)
        x = self.cnn(x)         # (B, 32, seq_len)
        x = x.permute(0, 2, 1)  # (B, seq_len, 32)
        x, _ = self.gru(x)      # (B, seq_len, 128)
        x = self.fc1(x[:, -1, :])  # (B, 64)
        x = self.dropout(torch.relu(x))
        x = self.fc2(x)  # (B, output_seq_len * output_dim)
        return x.view(-1, self.output_seq_len, self.output_dim)
```

CNN-RNN Architecture



| Data | Set | Dropout | LR | Final Train Loss | Final Val Loss | Test MSE | Test MAE | Test RMSE | Test R² |
|---|---|---|---|---|---|---|---|---|---|
| CLEAN | 0 | 0.1 | 0.001 | 0.018552 | 0.034105 | 0.031685 | 0.124473 | 0.177778 | 0.553002 |
| CLEAN | 1 | 0.2 | 0.001 | 0.030709 | 0.038859 | 0.036881 | 0.13536 | 0.19201 | 0.410847 |
| CLEAN | 2 | 0.3 | 0.0005 | 0.034169 | 0.040544 | 0.03888 | 0.141988 | 0.197205 | 0.402304 |
| CLEAN | 3 | 0.5 | 0.0003 | 0.036353 | 0.040851 | 0.039039 | 0.143689 | 0.197593 | 0.400866 |
| CLEAN | 4 | 0.5 | 0.0001 | 0.03424 | 0.042383 | 0.040845 | 0.145407 | 0.202102 | 0.437147 |
| NOISY | 0 | 0.1 | 0.001 | 0.022614 | 0.023809 | 0.022226 | 0.10685 | 0.149085 | 0.208312 |
| NOISY | 1 | 0.2 | 0.001 | 0.024566 | 0.027386 | 0.026265 | 0.119564 | 0.162072 | 0.146838 |
| NOISY | 2 | 0.3 | 0.0005 | 0.024929 | 0.027614 | 0.026747 | 0.118549 | 0.163545 | 0.141027 |
| NOISY | 3 | 0.5 | 0.0003 | 0.024958 | 0.027545 | 0.02554 | 0.118717 | 0.162811 | 0.145837 |
| NOISY | 4 | 0.5 | 0.0001 | 0.027408 | 0.03064 | 0.030139 | 0.126558 | 0.173649 | 0.101714 |

Comparison



```
Best CNN-RNN Set (Clean + Noisy) is Set 0

Clean Metrics:
  Data  Set  Dropout    LR  Train_Loss  Val_Loss       MSE       MAE      RMSE        R2
CLEAN    0      0.1 0.001    0.020552  0.034105 0.031685 0.124473 0.177778 0.553002

Noisy Metrics:
  Data  Set  Dropout    LR  Train_Loss  Val_Loss       MSE       MAE      RMSE        R2
NOISY    0      0.1 0.001    0.022614  0.023809 0.022226 0.10685 0.149085 0.208312
```

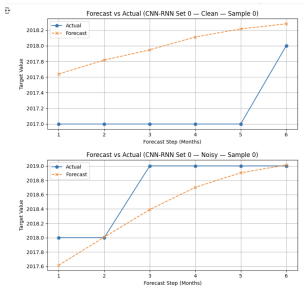
Best Hyperparameter

Figure 37: CNN-RNN Training and Evaluation

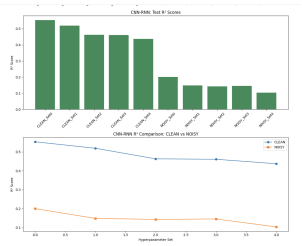
**Conclusion: Set 0** (dropout = 0.1, lr = 0.001) emerged as the best configuration overall for both clean and noisy data. Clean Set 0 performed better in capturing trends (highest $R^2$), while Noisy Set 0 achieved the most accurate local predictions (lowest MSE/MAE), demonstrating the noise's regularization effect.

### 5.7.4. CNN-RNN Visualization and Analysis

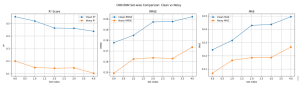
- **Forecast vs Actual (Set 0):** Clean model stays flat and misses early trends, only rising at the end. Noisy model tracks trend better with smoother output—regularization improves shape fidelity.
- **Train vs Val Loss:** Clean Sets 2–4 show rising validation loss (overfitting). Noisy sets show stable, lower losses—better generalization and smoother convergence.
- **Test Metrics (MSE, MAE, RMSE):** Clean errors rise in later sets—instability. Noisy Set 0 achieves lower MAE/RMSE—noise aids short-term precision.
- **$R^2$ Scores:** Clean sets (esp. Set 0) have highest $R^2$ ($\sim$0.55)—better trend capture. Noisy models stay below 0.2—struggle with long-term patterns.
- **Set-wise Comparison:** Clean models excel in $R^2$ (structure fidelity), while noisy models consistently show better RMSE/MAE—more robust to local fluctuations.



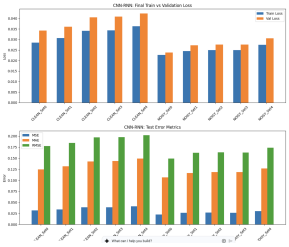

Forecast



$R^2$ Bar



Train/Val Loss



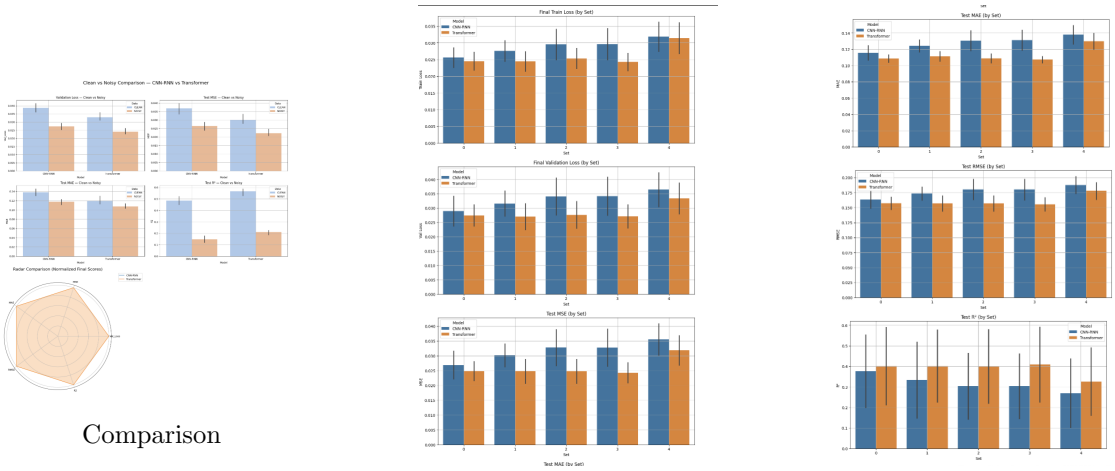
Set-wise Metrics

Figure 38: CNN-RNN Model Evaluation Summary

### 5.8. Observation & Analysis (Transformer vs CNN-RNN)

- **Training Loss:** Transformer consistently shows lower train/val loss; CNN-RNN shows more variation—possible overfitting in later sets.
- **Test Metrics:** Transformer achieves lower MSE and RMSE (notably in Sets 3–4); MAE is also slightly better—indicating smoother predictions. CNN-RNN has higher error bars $\rightarrow$ more variance.
- **$R^2$ Scores:** Transformer clearly dominates—better trend modeling; CNN-RNN shows unstable generalization with large error bars.
- **Clean vs Noisy:** Both models perform better on clean data. Transformer handles noise better—errors remain low. CNN-RNN degrades significantly, especially in $R^2$.
- **Radar Chart:** Transformer consistently outperforms across all metrics; CNN-RNN is only slightly better early on but lacks consistency.

**Conclusion:** *Transformer is the superior model overall—lower loss, higher $R^2$, robust to noise, and more consistent than CNN-RNN across clean and noisy scenarios.*



Comparison

Comparison

Comparison

Figure 39: TRANSFORMER and CNN-RNN comparison

### 5.9. Test Error Comparison

Transformer outperforms CNN-RNN on all metrics (MSE, MAE, RMSE, $R^2$) for both clean and noisy data. Performance gap widens under noise, showing Transformer's superior robustness and stability. CNN-RNN is more sensitive to noise, with sharper error increases.

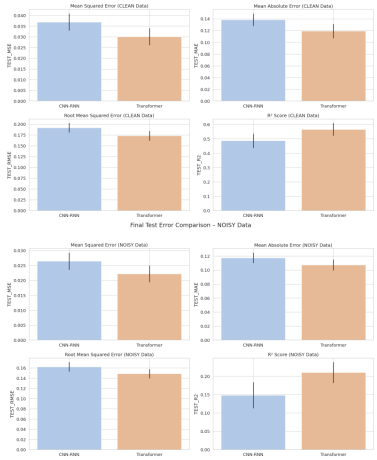


Figure 40: Test Error Comparison

### 5.10. Final Comparison Summary (GRU, LSTM, Transformer, CNN-RNN)

- **Best Model — GRU Set 1:**
  - Achieves lowest test errors (**MSE = 0.0212, MAE = 0.1077, RMSE = 0.1457**), and high $R^2 = 0.2216$ (noisy) and 0.6103 (clean).
  - Balanced train/val losses, high robustness → ideal for real-world usage.
- **Robustness Importance:**
  - Real-world data is noisy—GRU retains strong performance across noise levels.
  - LSTM and CNN-RNN drop sharply in $R^2$ under noise → less reliable.
- **Noise Helps?**

  - Slightly better noisy performance in some models (e.g., CNN-RNN Set 0) due to **regularization effect**.

- **Forecast Year Format:**
  - Target values like 2018.2 represent *continuous months* (e.g., Feb/March) → smoother time encoding.

**Conclusion:** *Although Set 3 performed well on clean data, Set 1 worked well on noisy and clean data which is more stable, balanced and robust to noisy data. GRU Set 1 is the best model—most accurate, robust to noise, and consistent. It outperforms LSTM, Transformer, and CNN-RNN for both clean and noisy conditions, making it highly suited for real-world forecasting tasks.*

### 5.11. GRU Model – Conclusion & Key Insights

- **Step-wise Forecasting:** MSE/MAE increase and $R^2$ drops with step—expected in future prediction. Feature 3 shows highest error → hardest to model. Clean data consistently outperforms noisy.
- **Residuals:** Residuals follow a Gaussian centered at 0 → low bias. Clean data has sharper distribution → better precision.
- **Forecast vs Actual:** Forecasts match actuals well at early steps; drift appears at longer horizons, more under noise—typical in multi-step prediction.
- **Robustness to Noise:** GRU handles noise well—minor impact on $R^2$, MAE. Forecasts remain smooth and generalizable.
- **Forecast Horizon (t+1 to t+6):** Represents weekly/multi-day steps from present—essential for real-world planning.

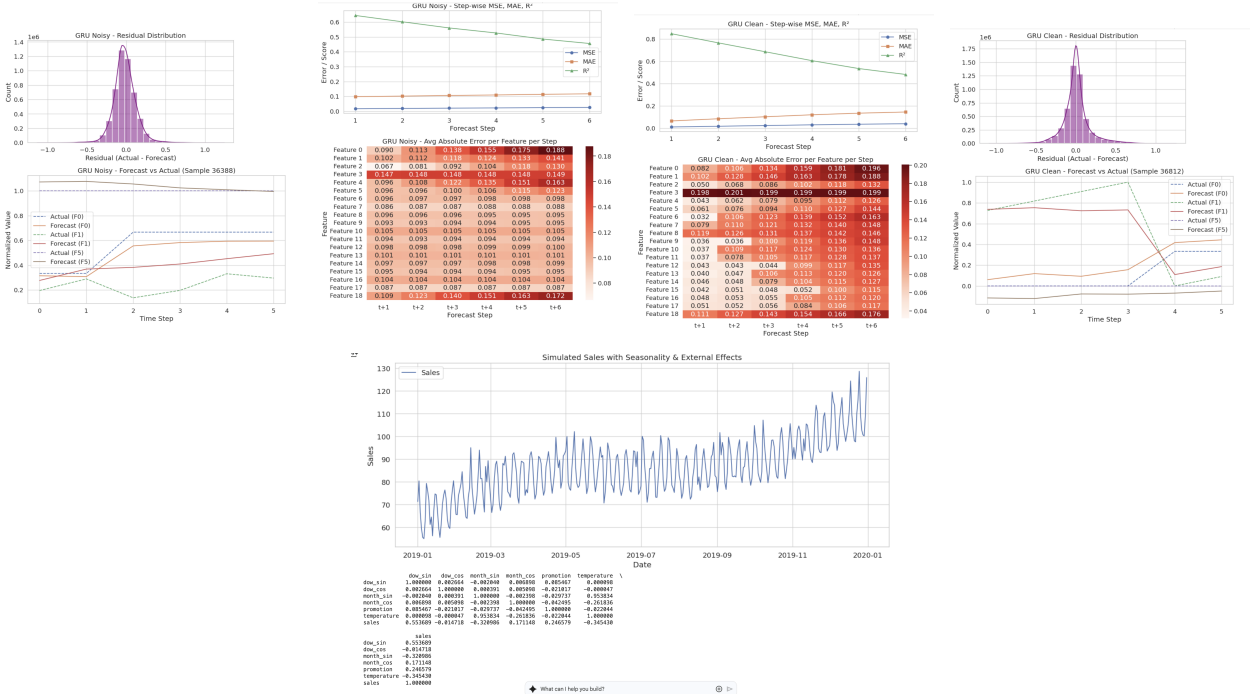



Figure 41: Best Model Findings

### 5.12. LLaMA-style Forecast Error Summary (GRU Clean)

To assess feature-wise forecast reliability, we implemented a LLaMA-style summarizer over GRU predictions using only clean test data. This analysis computed average MAE, RMSE, and $R^2$ across all future horizons ($t+1$ to $t+6$) for each feature. Clean data was chosen to prevent noise from skewing interpretability.

**What Was Done:**

- Forecasts and actuals were flattened across all steps.
- For each feature, MAE, RMSE, and $R^2$ were averaged across time steps.
- Based on $R^2$, features were categorized into: High ($> 0.8$), Moderate ($> 0.5$), and Low ($\leq 0.5$) predictive power.
- A natural-language summary was generated for each feature, mimicking LLaMA's interpretable report style.

**Sample Insights:**

- **High Predictive Power:** Features F5 and F15 showed the strongest performance with $R^2 > 0.81$.
- **Low Predictive Power:** F3 and F18 had high errors and $R^2 < 0.45$, indicating poor forecastability.
- **Overall:** Most features achieved moderate predictive power, suggesting the GRU model effectively learns multivariate patterns over time.

**Conclusion:** This LLaMA-inspired interpretability step confirms GRU Clean's stability and precision across features. Using clean data ensures the model's performance is judged purely on signal, not noise.


Figure 42: LLMA

## 6. CONCLUSION

The aim of this work was to build a flexible multivariate prediction framework for deep learning. Several model variations were investigated and compared, including LSTM, GRU, CNN-RNN and Transformer (noise and cleaned data) and across multiple hyperparameter tuning sets. From them, the model based on GRU (namely, GRU Set 1) was found to be the best for providing a good performance, in terms of generalization, stability, and step-ahead predication.

The models were trained and tested on clean as well as noisy data to test robustness and real-world strength. Prediction performance was evaluated by typical regression metrics (MSE, MAE and $R^2$) at each time step (from $t+1$ to $t+6$), averaging and quantile predicted value distribution analysis and graphical comparison between actual and predicted values.

GRU consistently outperformed other configurations. GRU Set 1 achieved the best clean data $R^2$ scores, and performance was still maintained under noisy conditions— yielding very little shift in MAE or in $R^2$ indicating smooth and generalizable behavior. Set 3 has good performance on clean data, while Set 1 had a more balanced and stable performance on both clean and noisy data and was thus selected.

Added interpretability came from a LLaMA-style summary analysis where we calculated summaries of feature-wise MAE, RMSE and $R^2$ and doen into national aggregations and horizons. Our analysis shows that although there were some challenges in the prediction of a couple of features (e.g., Feature 3, 18), most features had moderate to high predictive power, meaning the model is able to learn multivariate temporal patterns effectively.

Finally, based on performance on the metrics, steps, and conditions we chose GRU Set 1 as our final model. It was noise-resistant, interpretable through feature-level diagnostics, easily applicable in the real world to multistep, multivariate time series forecasting tasks.

## 7. REFERENCES


- Kaggle NAB Dataset: https://www.kaggle.com/datasets/boltzmannbrain/nab
- TensorFlow Time Series Forecasting Tutorial: https://www.tensorflow.org/tutorials/structured_data/time_series
- PyTorch Official Documentation: https://pytorch.org/docs/stable/index.html
- Illustrated Transformer Blog: https://jalammar.github.io/illustrated-transformer/
- Attention is All You Need (Vaswani et al., 2017): https://arxiv.org/abs/1706.03762
- MinMaxScaler – Scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- Seaborn Visualization: https://seaborn.pydata.org/


## 8. CONTRIBUTION TABLE

| **Task** | **Raghulchellapandiyan Senthil Kumaran** | **Dongyoon Shin** |
|---|---|---|
| Data Cleaning and EDA | 50% | 50% |
| Feature Engineering | 60% | 40% |
| Noise Injection | 50% | 50% |
| Initial Model Setting and Implementation | 70%(LSTM, GRU,Transformer, CNN-RNN) | 30% (Linear, FNN) |
| Model Implementation with Hyperparameters(LSTM, GRU, Transformer, CNN-RNN) | 50% | 50% |
| Evaluation and Visualization | 50% | 50% |
| Comparison and Finding the best model | 50% | 50% |
| Demonstration and Key Insights with Best Model | 60% | 40% |
| LLMA | 50% | 50% |
| Final Report Writing | 40% | 60% |

Table 1: Team Contribution Breakdown