SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

**50.039 Theory and Practice of Deep Learning**

Project Report: *Deep learning model for Time series data*

By Cohort 1 Group 9



| Name | Student ID |
|---|---|
| Oon Shao Ren | 1005935 |
| Tran Cong Nam Anh | 1005242 |
| Benetta Cheng Jia Wen | 1006232 |



19 April 2025

Professor Prof Matthieu De Mari & Prof Qun Song

**1. Dataset overview** **1**
**2. Dataset exploration and analysis** **1**
Statistical Summary 1
Feature Distribution and Outliers 2
Correlation Analysis 4
Temporal Patterns and Seasonality 4
Enhanced Analysis - Mean Pressure Spike 6
Insights and Implications for Modeling 7
**3. Data Cleaning, Dataset and Dataloader** **7**
Data Cleaning and Normalization 7
Custom Dataset Construction 8
DataLoader Setup 9
This setup allows for fast, memory-efficient training with support for GPU acceleration. 9
**4. Model Architectures** **9**
4.1 Seq2Seq Long Short-Term Memory 10
4.1.1 Architecture 10
4.1.2 Optimal Hyperparameters 12
4.1.3 Training 13
4.1.4 Loss Curve during Training 13
4.1.5 Model Evaluation 14
4.2 GRU 15
4.2.1 Architecture 15
4.2.2 Optimal Hyperparameters 16
4.2.3 Training 17
4.2.4 Loss Curve during Training 17
4.2.5 Model Evaluation 18
4.3 Transformer 19
4.3.1 Architecture 19
4.3.2 Optimal Hyperparameters 21
4.3.3 Training 22
4.3.4 Loss Curve during Training: 22
4.3.5 Model Evaluation 23
**5. Comparison among models** **23**
**6. Techniques to improve model training** **25**
Seq2Seq GRU 25
**7. Challenges and areas for improvements** **27**
References 28

# 1. Dataset overview

The dataset for this project is sourced from Kaggle and comprises daily climate observations from Delhi, India, spanning from January 1, 2013, to April 24, 2017. It contains two distinct CSV files:

- DailyDelhiClimateTrain.csv: Used to train the deep learning models.
- DailyDelhiClimateTest.csv: Used for validating and evaluating the model predictions.

Link to the dataset:
**https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data**

All project code and documentation are available at:
**https://github.com/LouisAnhTran/deep_learning_models_for_time_series_data**

Each file consists of daily observations featuring four primary numerical variables:

- Mean Temperature (meantemp): Averaged from multiple 3-hour intervals in a day, measured in degrees Celsius.
- Humidity: Grams of water vapor per cubic meter volume of air.
- Wind Speed: Daily average wind speed measured in kilometers per hour (km/h).
- Mean Air Pressure (meanpressure): Daily atmospheric pressure readings measured in milibars (hPa).

A separate date column links each data entry to a specific date, capturing the essential temporal dynamics necessary for effective time series forecasting.

# 2. Dataset exploration and analysis

In this section, we perform a detailed exploratory data analysis (EDA) to understand the intrinsic properties and patterns within the Daily Delhi Climate dataset, providing essential context and insights for our predictive modelling efforts.
The code facilitating this exploration can be found in `VizData_Notebook.ipynb`.

## Statistical Summary

We computed descriptive statistics to understand the data distributions:

- Mean Temperature ranged between approximately 6°C and 39°C.
- Humidity predominantly varied between approximately 19 g/$m^3$ and 100g/$m^3$.
- Wind Speed showed variability, generally staying below 20 km/h.

- Mean Air Pressure typically remained stable within 1000 - 1020 hPa, except for rare data anomalies.

## Feature Distribution and Outliers

To gain deeper insights into the characteristics of each variable, we visualized their distributions and detected outliers using box plots, histograms, and violin plots:

- Box Plots: Revealed potential outliers and the overall spread for each feature. Wind speed and mean pressure exhibited noticeable anomalies, including extreme values such as pressures falling below 100 hPa or exceeding 7000 hPa, far beyond the typical range of atmospheric pressure, indicating likely data entry or inconsistencies introduced during data collection or source compilation.

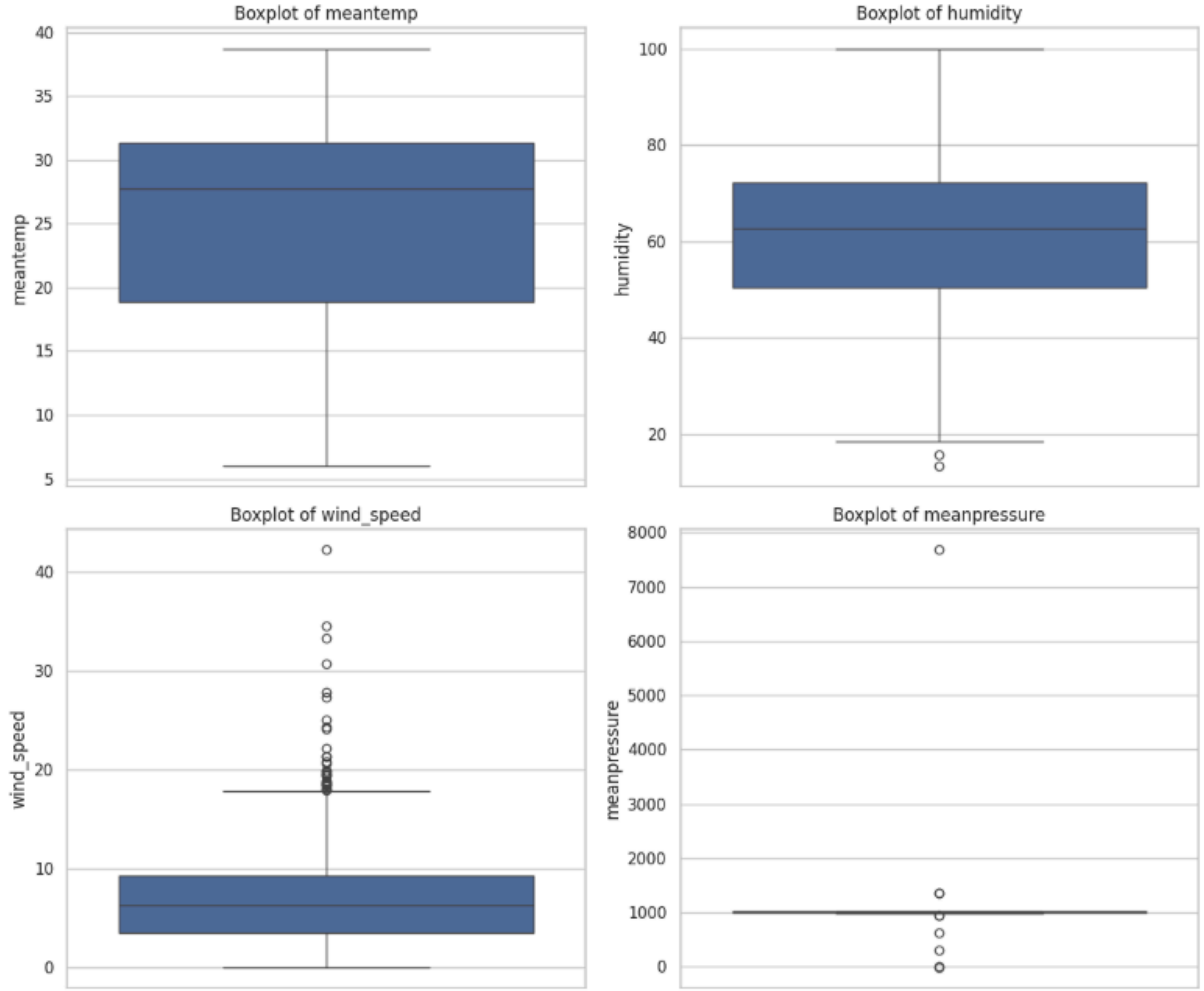



Figure 2.1: Box Plots

- Histograms: Showed the shape of each distribution. Wind speed was heavily right-skewed, with the majority of values clustered around 5 - 10 km/h, but with a long tail due to occasional high-speed outliers. Temperature and humidity had more symmetric distributions with clearly visible seasonal variation.


Figure 2.2: Histograms

- Violin Plots: Provided a combined view of the density and distribution spread, confirming multiple peaks in temperature and humidity likely corresponding to seasonal cycles. The flat, wide distributions in wind speed and pressure further reinforced their outlier-prone nature.

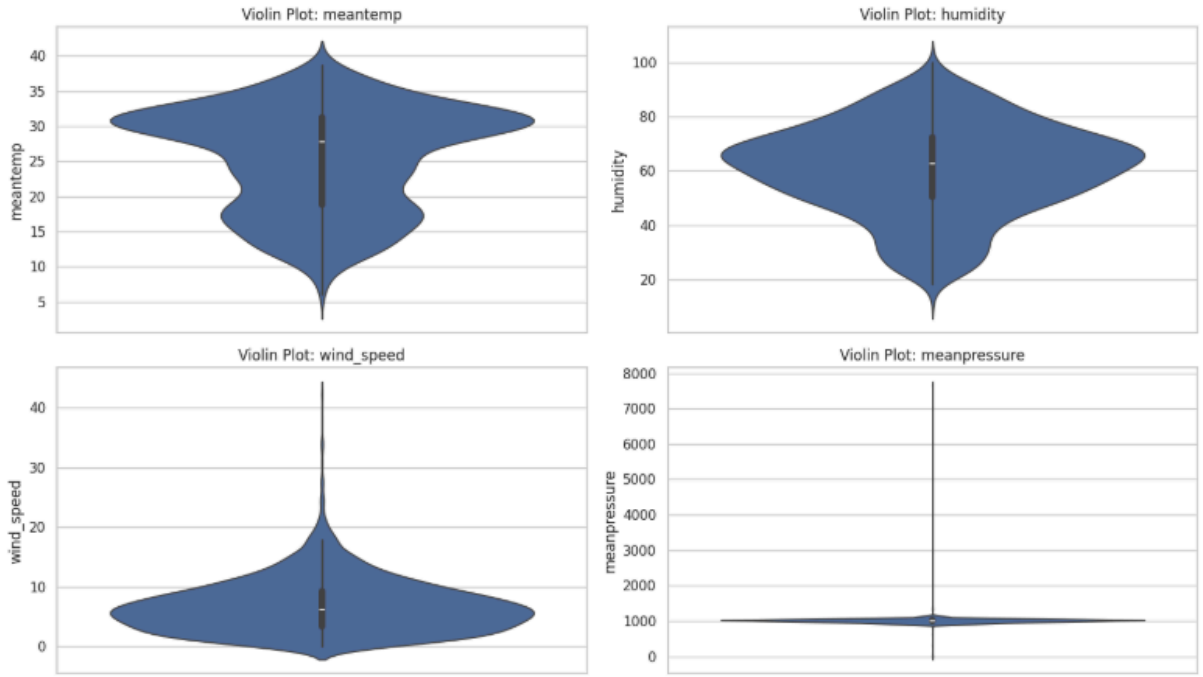



Figure 2.3: Violin Plots

## Correlation Analysis

A correlation heatmap analysis illustrated:

- A strong negative correlation (-0.57) between temperature and humidity, indicating inverse seasonal trends.
- Mild positive correlation (0.31) between temperature and wind speed
- Weak correlations of mean pressure with other variables, suggesting independence in its variability.

## Temporal Patterns and Seasonality

Time series visualizations provided critical insights into the seasonality and variability of the climate variables. To maximize clarity, we focused on the coloured year-over-year plots and a consolidated view of all weather variables.

- Mean Temperature and Humidity: Both variables exhibited strong annual cycles across the four-year span. Temperatures peaked consistently during mid-year summers, while humidity showed clear monsoon-driven spikes. These recurring patterns strongly support the use of models capable of capturing seasonal dependencies.

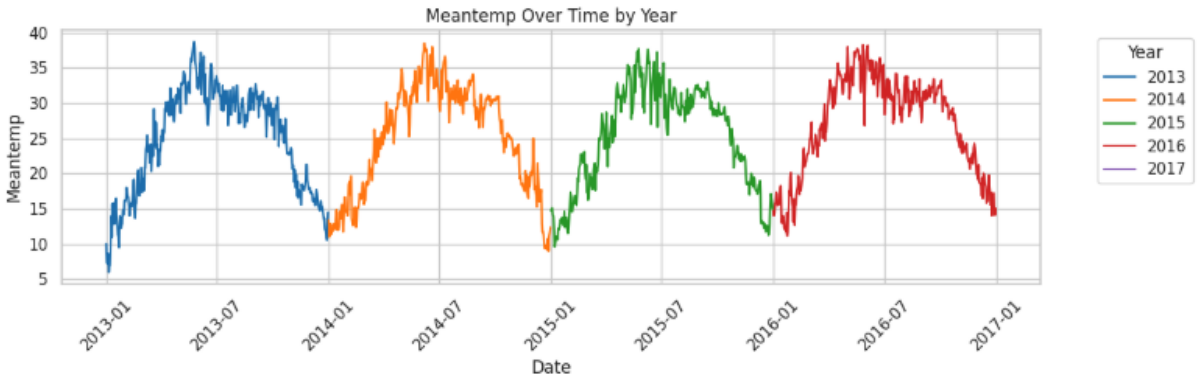



Figure 2.4: Time series visualization for meantemp

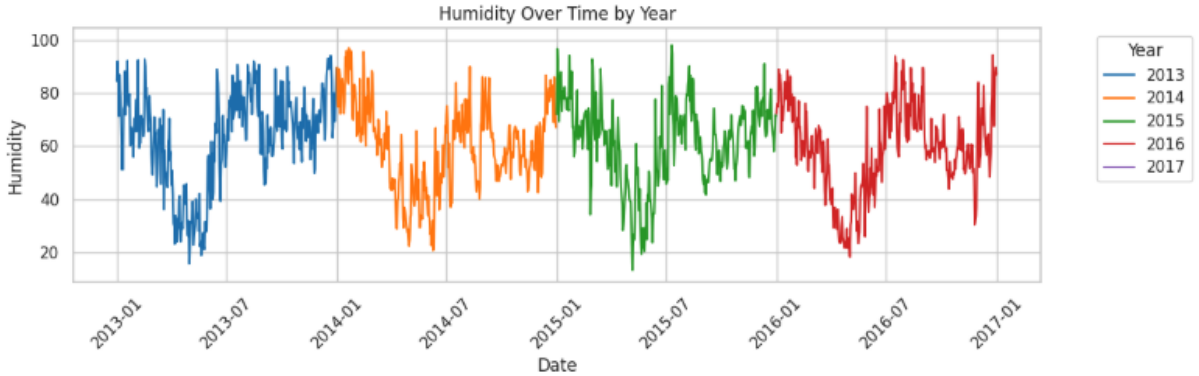



Figure 2.5: Time series visualization for humidity

- Wind Speed: Showed frequent short-term spikes without consistent seasonality. This irregularity indicates higher volatility, suggesting that models need to be flexible enough to capture both sudden bursts and low-activity phases.

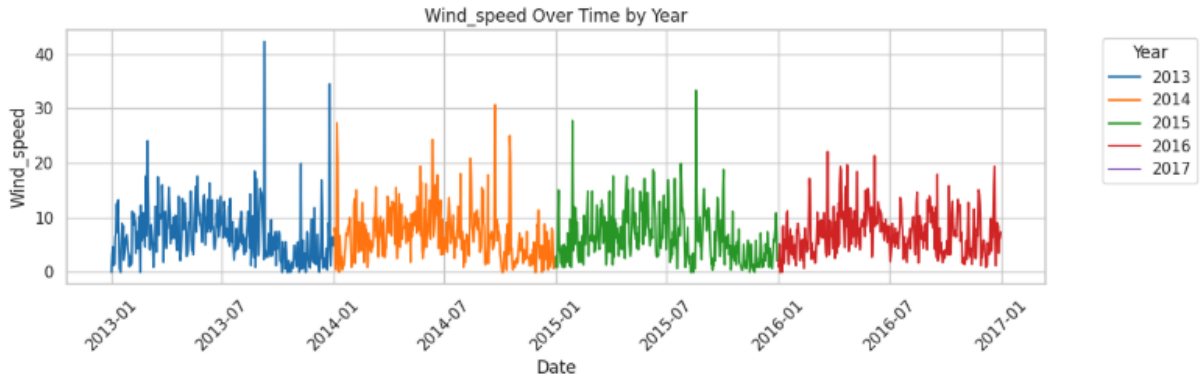



Figure 2.6: Time series visualization for windspeed

Mean Air Pressure: Appeared relatively stable when viewed by year, hovering around 1000 - 1020 hPa. However, its long-term view, particularly when plotted alongside other features, revealed rare but extreme deviations.

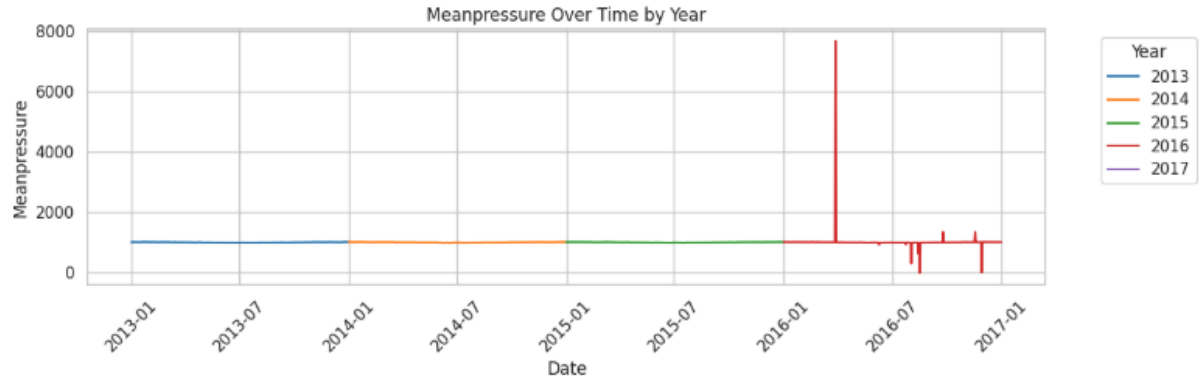



Figure 2.7: Time series visualization for meanpressure

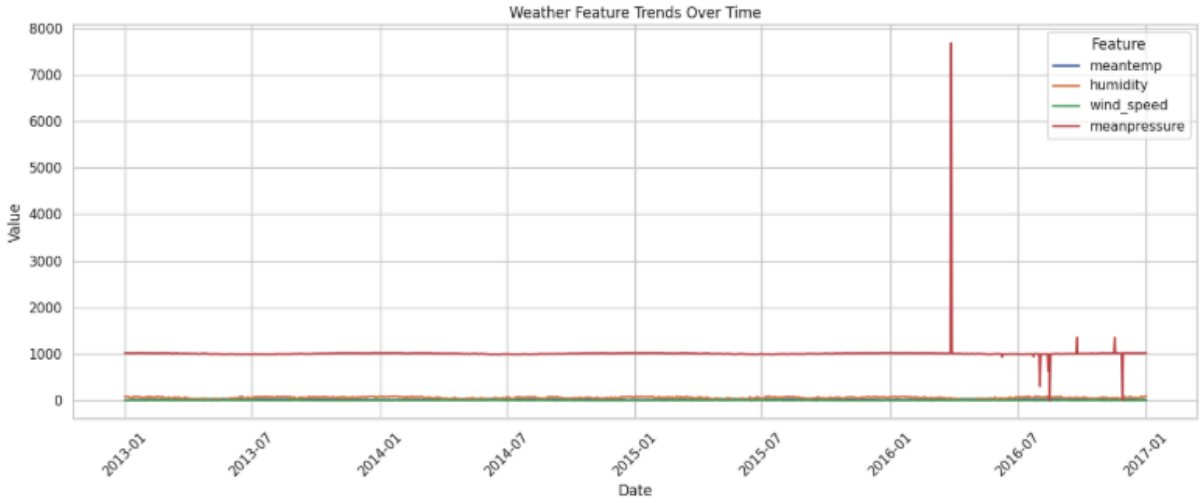



Figure 2.8: Time series visualization for all four features

## Enhanced Analysis - Mean Pressure Spike

To investigate a peculiar anomaly in mean air pressure, we isolated this feature and visualized it over targeted timeframes:

- Before the spike: Air pressure followed expected seasonal oscillations, showing no irregular behaviour.


Figure 2.9: Time series visualization for meanpressure before the spike

- During the spike: A sudden and drastic spike in late March 2016 exceeded 7000 hPa, far beyond physical plausibility, suggesting a clear data integrity issue.

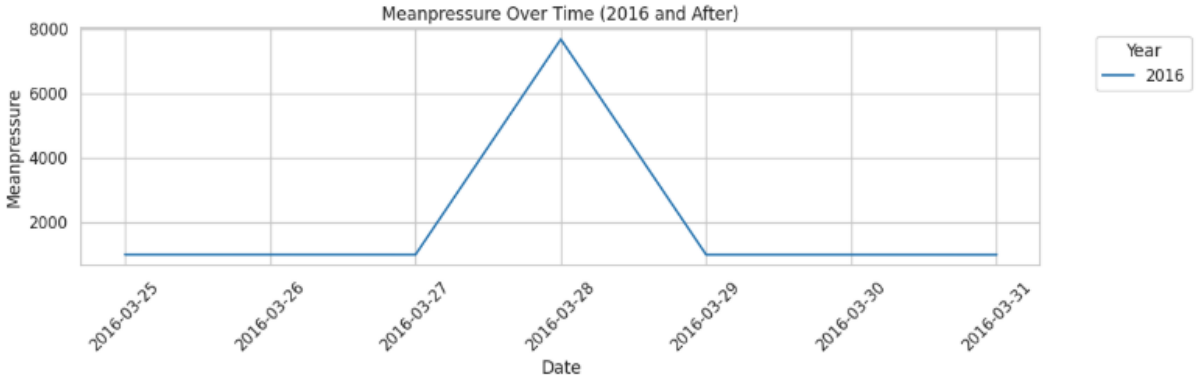



Figure 2.10: Time series visualization for meanpressure during the spike

- After the spike: Residual inconsistencies persisted, with abnormal pressure drops and minor spikes scattered across the remaining data.

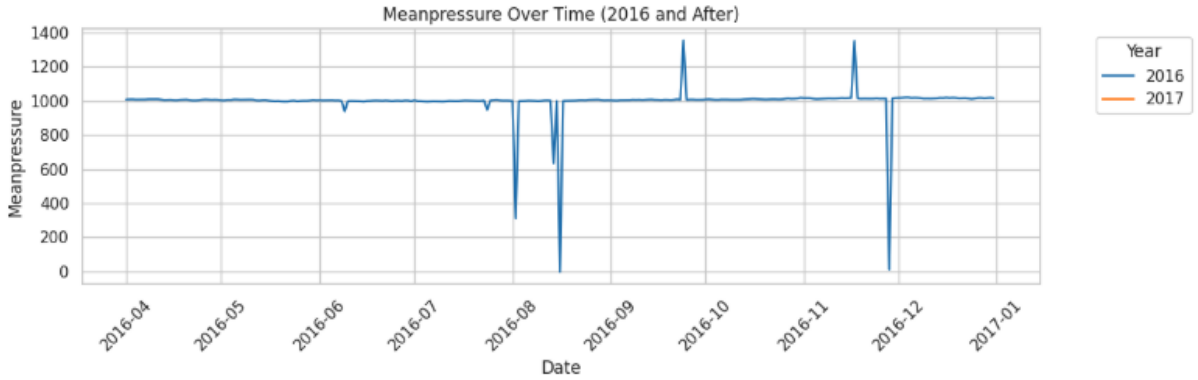



Figure 2.11: Time series visualization for meanpressure after the spike

This deep dive into the mean pressure anomaly emphasizes the importance of robust anomaly detection and potential pre-filtering when training forecasting models, as such extreme values could disproportionately influence model behaviour.

### Insights and Implications for Modeling

This exploration underscored several critical implications for our modelling approaches:

- Strong seasonal and temporal patterns validated our selection of sequential deep learning models like LSTM, GRU, and Transformers, known for capturing complex time dependencies.
- Weak inter-feature correlations further validated employing multivariate models, ensuring we leverage diverse predictive signals from all available variables.

These visual and analytical insights have guided our modelling strategy, emphasizing sequential architectures that effectively manage temporal dependencies and anomalies inherent to climate datasets.

# 3. Data Cleaning, Dataset and Dataloader

### Data Cleaning and Normalization

The dataset used for this project consists of daily climate measurements from Delhi, including features such as mean temperature, humidity, wind speed, and mean pressure. The data was split into a training and test set, each saved in separate CSV files: DailyDelhiClimateTrain.csv and DailyDelhiClimateTest.csv.

The load_dataset function handles the core preprocessing steps:

1. Date Column Removal: The date column is parsed but dropped, as our time series model relies only on the numerical features.
2. Normalization: Each feature is normalized using the mean and standard deviation calculated from the training set statistics to ensure a consistent scale across features.

```python
import pandas as pd

def load_dataset(train_path, test_path):
    # Load and drop date column
    train_df = pd.read_csv(train_path, parse_dates=['date']).drop(columns=['date'])
    test_df = pd.read_csv(test_path, parse_dates=['date']).drop(columns=['date'])

    # Normalize using training statistics
    stats = train_df.describe().transpose()

    def normalize(df):
        return (df - stats["mean"]) / stats["std"]

    train_norm = normalize(train_df)
    test_norm = normalize(test_df)

    return train_norm, test_norm, stats
```

This normalization ensures that the test data remains unseen and unbiased during the training phase, adhering to best practices in time series forecasting.

## Custom Dataset Construction

To handle sequence modeling, a custom PyTorch Dataset class was implemented. The CustomDataset class slices the time series data into overlapping input-output pairs:

- n_inputs: Number of past days used for input.
- n_outputs: Number of future days to predict.

```python
class CustomDataset(Dataset):
    def __init__(self, dataframe, n_inputs, n_outputs):
        self.dataframe = dataframe
        self.n_inputs = n_inputs
        self.n_outputs = n_outputs
        self.inputs = []
        self.outputs = []
        self.define_samples()

    def define_samples(self):
        data_array = self.dataframe.values
        for i in range(len(data_array) - self.n_inputs - self.n_outputs):
            input_seq = data_array[i:i+self.n_inputs, :]         # Sequence length of 30, incl all features
            output_seq = data_array[i+self.n_inputs:i+self.n_inputs+self.n_outputs, :]  # Sequence length of 7, incl all features
            self.inputs.append(input_seq)
            self.outputs.append(output_seq)

    def __len__(self):
        return len(self.inputs)

    def __getitem__(self, idx):
        x = torch.tensor(self.inputs[idx], dtype=torch.float32)  # Shape: (n_inputs, 4)
        y = torch.tensor(self.outputs[idx], dtype=torch.float32) # Shape: (n_outputs,)
        return x, y
```

This dataset design allows the model to learn from sliding windows of data, which is crucial for time-dependent prediction tasks.

### DataLoader Setup

PyTorch DataLoader objects are used to efficiently batch and shuffle the data during training. Shuffling is applied only to the training data to improve generalization, while the test data remains in sequence:

```
n_inputs = 30
n_outputs = 7
train_dataset = CustomDataset(train_data, n_inputs, n_outputs)
test_dataset = CustomDataset(test_data, n_inputs, n_outputs)
```

```
# Define batch size
batch_size = 128

# Random number generator (for reproducibility)
generator = torch.Generator()  # Defaults to CPU, which DataLoader expects

# Create DataLoader
train_dataloader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True,
    generator=generator
)

test_dataloader = DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False,  # Important: don't shuffle test data
    generator=generator
)
```

This setup allows for fast, memory-efficient training with support for GPU acceleration.

# 4. Model Architectures

To achieve optimal accuracy in predicting our time series dataset, we experimented with a diverse set of models trained using different architectures. The rationale behind this approach is that there are many available architectures for tackling time series problems, which encourages us to explore the most prominent options and select the models with the best performance. Due to time constraints and other limitations, we were not able to train all models from scratch. Instead, we selected a few models to train from scratch and applied transfer learning to fine-tune several state-of-the-art pretrained models. Once the model training was completed, we conducted a thorough evaluation of each model's performance and performed a comparative analysis to derive key insights. The table below summarizes all the different model architectures we experimented with.

| Models | Scope | Status |
|---|---|---|
| Seq2Seq auto-regressive Recurrent Neural Network using Long Short-Term Memory (LSTM) | Train from scratch | Completed |
| Seq2Seq non-auto-regressive Recurrent Neural Network using Long Short-Term Memory (LSTM) | Train from scratch | Completed |
| Seq2Seq auto-regressive Recurrent Neural Network using Gated Recurrent Unit (GRU) | Train from scratch | Completed |
| Seq2Seq non-auto-regressive Recurrent Neural Network using Gated Recurrent Unit (GRU) | Train from scratch | Completed |
| Non-recurrent Seq2Seq model using Transformer architecture | Train from scratch | Completed |
| Use/Fine-tune state-of-the-art pre-trained model as the baseline for comparison | Fine tune using Transfer learning | In progress |

For each model, cover parameters detailed training steps, results discussion with clear figures, loss curve during training, model performance on test set

# 4.1 Seq2Seq Long Short-Term Memory

## 4.1.1 Architecture

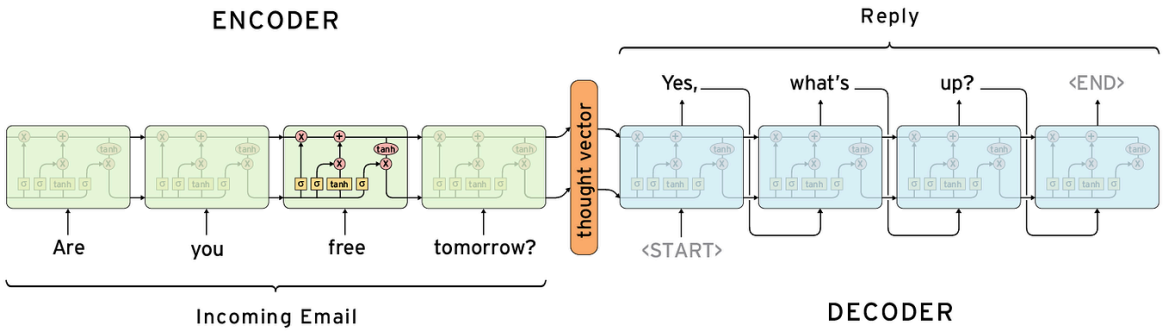



Figure 4.1.1: Overall architecture (Kumar, 2021)

Similar to the GRU model, in this section, we will explore the architecture and training implementation of the model trained using the Seq2Seq LSTM architecture. Our LSTM

Seq2Seq model takes in an input sequence comprising 4 features and attempts to predict an output sequence for all 4 features over a predefined period of time. Like the GRU model, the LSTM Seq2Seq architecture also consists of two components: the Encoder and the Decoder.

The EncoderRNN processes input weather data sequences through an LSTM, which is designed to capture patterns and dependencies in the data, and potentially learn the relationships among the four weather features. At the end of the encoding phase, only the final memory states—comprising the hidden states and cell states—are returned, encapsulating the learned representation of the entire sequence. This memory vector is then used by the decoder to generate the output sequence predictions.

The **DecoderRNN** uses the final hidden and cell states produced by the encoder as its initial states to begin generating the output sequence. We implemented two different approaches to generate the output sequences: **auto-regressive** and **non-auto-regressive** (using the teacher forcing technique).

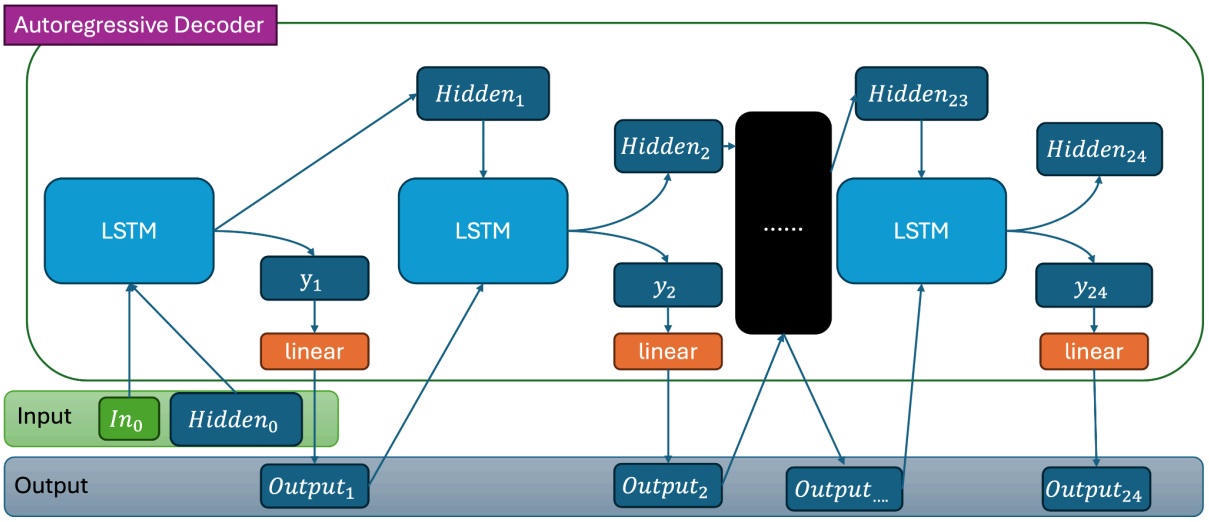



Figure 4.1.2 Autoregressive architecture (Source: HW3)

In the auto-regressive approach, at each time step, the decoder takes the output from the previous step as input for the current step, along with the hidden states produced by the encoder. This input is passed through an LSTM layer to generate the next output, which is then fed back into the LSTM at the following time step. This auto-regressive strategy is applied consistently during both the training and testing phases.

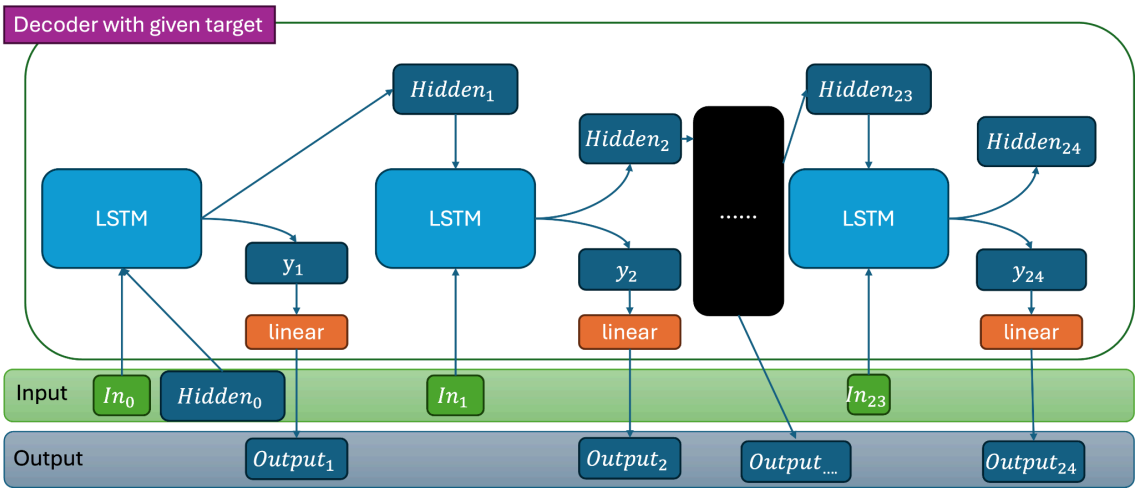



Figure 4.1.3 Non-autoregressive architecture using teach forcing (Source: HW3)

For the non-auto-regressive model, we employed a technique called **teacher forcing**, which is an effective way to enhance model training and accelerate convergence. In this approach, the ground truth outputs are used as inputs to the decoder at each time step, allowing the LSTM layer to more effectively learn the underlying patterns and dependencies in the weather time series data. It is important to note that teacher forcing is only used during training and is disabled during testing to allow the model to generate predictions autonomously.

### 4.1.2 Optimal Hyperparameters

| Variables | Hyperparameter Details | Values |
|---|---|---|
| Data dimensions | | |
| N_INPUTS | The number of input sequence length, meaning how many data points the model used as the input | 30 |
| N_OUTPUTS | The number of output sequence length, meaning how many data points in the future that model will generate | 7 |
| Model architecture hyperparameters | | |
| HIDDEN_SIZE | The dimensions of memory vector, which consist of hidden state and cell state | 100 |
| NUMBER_LAYERS | Number of layers in each LSTM layer in both encoder and decoder | 2 |

| | | |
|---|---|---|
| DROPOUT_RATE | The rate for dropout layer | 0.1 |
| Training hyperparameters | | |
| LEARNING_RATE | The learning rate used in gradient descent formula | 0.001 |
| NUM_EPOCHS | The number of epochs to be used during model training | 600 |

Figure 4.1.3: Optimal hyperparameters

### 4.1.3 Training

The detailed and step-by-step instructions for training this model can be found in this notebook:

For auto-regressive model: Notebook

For non auto-regressive model: Notebook

### 4.1.4 Loss Curve during Training


Figure 4.1.4: Loss curve training for auto-regressive Seq2seq LSTM model

From Figure XX, we can observe that the model's training loss decreased steadily over time and approached nearly zero after 300 epochs. This indicates that our chosen model architecture and hyperparameters contributed to effective model training.

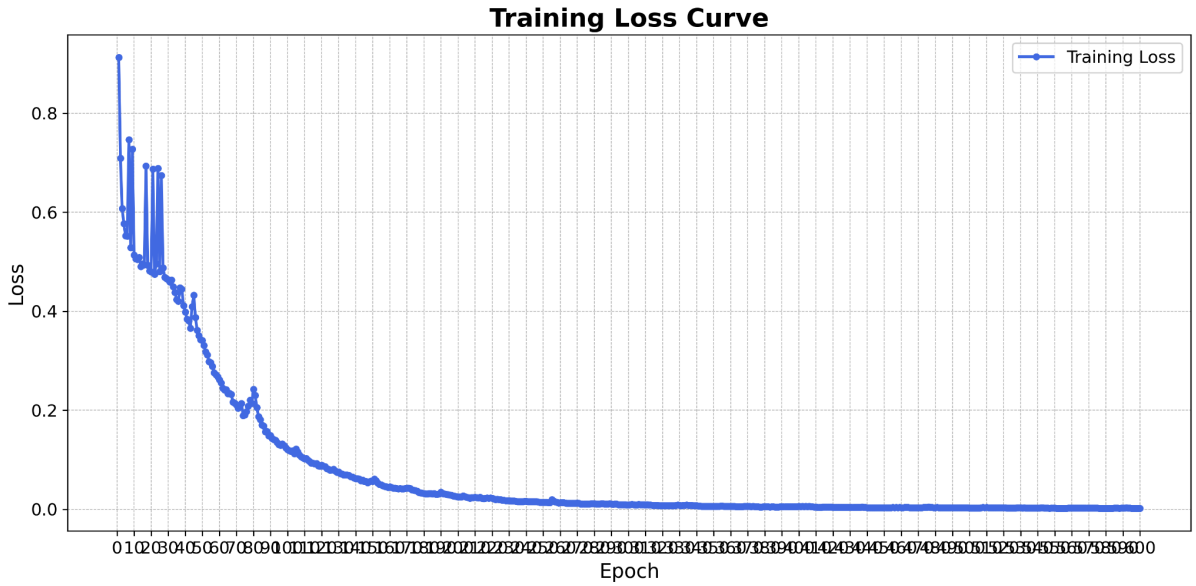



Figure 4.1.5: Loss curve training for non auto-regressive Seq2seq LSTM model

The loss curve for the model using a non-autoregressive architecture is smoother compared to that of the autoregressive model, and it appears to converge significantly faster. One of the main reasons for this improvement is the use of the teacher forcing technique, which has proven to be effective in enhancing model training and accelerating convergence.

## 4.1.5 Model Evaluation

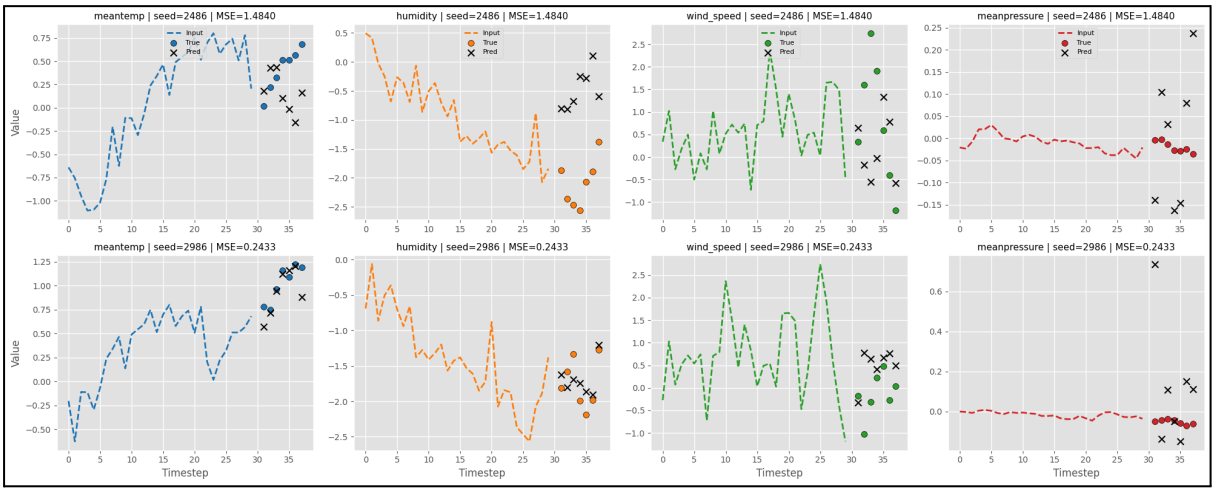



Figure 4.1.6: Model evaluation on auto-regressive model

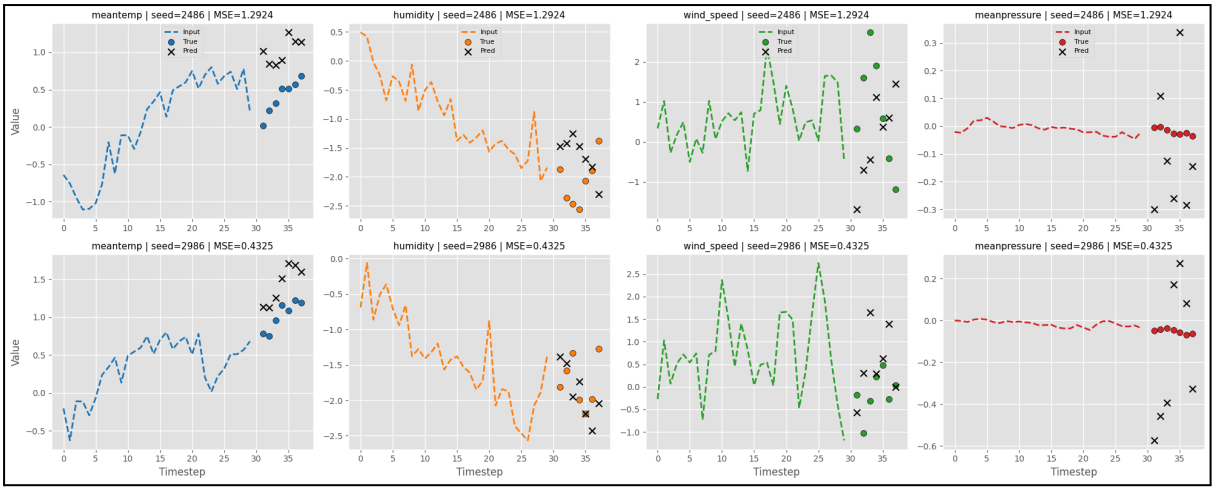



Figure 4.1.7: Model evaluation on non auto-regressive model

# 4.2 GRU

## 4.2.1 Architecture

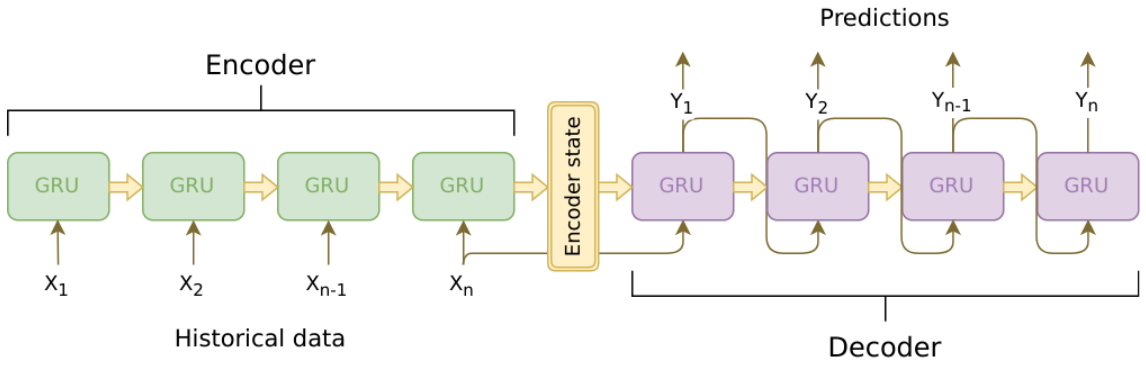



Figure 4.2.1: Overall Seq2Seq architecture with GRU

Similar to the LSTM model, in this section, we will explore the architecture and training implementation of the model trained using the Seq2Seq GRU architecture. Our GRU Seq2Seq model takes in an input sequence comprising 4 features and attempts to predict an output sequence for all 4 features over a predefined period of time. Like the LSTM Seq2Seq architecture, the GRU Seq2Seq model also consists of two components: the Encoder and the Decoder.

The EncoderGRU processes input weather data sequences through a GRU layer, which is designed to capture temporal patterns and dependencies within the time series data. It also models the relationships among the four weather features—mean temperature, humidity, wind speed, and mean pressure. At the end of the encoding phase, the encoder returns the

final hidden state, which serves as a condensed representation of the input sequence. This hidden state is then passed to the decoder to generate the output sequence predictions.

The DecoderGRU receives the encoder's final hidden state as its initial state and generates the output sequence step-by-step. Like before, we implemented two different approaches to generate the output sequences: auto-regressive and non-auto-regressive (using the teacher forcing technique).

In the auto-regressive approach, at each time step, the decoder takes the output from the previous step as input for the current step, along with the hidden states produced by the encoder. This input is passed through a GRU layer to generate the next output, which is then fed back into the GRU at the following time step. This auto-regressive strategy allows the model to condition its predictions on past outputs, and it is applied consistently during both the training and testing phases to simulate real-world forecasting conditions.

For the non-auto-regressive model, we employed **teacher forcing** as in the LSTM Seq2Seq, which is an effective strategy to enhance model training and accelerate convergence. In this approach, the ground truth outputs are used as inputs to the decoder at each time step during training, instead of relying on the model's own previous predictions. This allows the GRU layer to learn the underlying temporal and feature-wise dependencies in the weather time series data more effectively and with reduced exposure to compounding prediction errors. It is important to note that teacher forcing is only used during training, and is disabled during testing, where the model generates predictions autonomously from the encoder's final hidden state.

### 4.2.2 Optimal Hyperparameters

| Variables | Hyperparameter Details | Values |
|---|---|---|
| Data dimensions | | |
| N_INPUTS | The number of input sequence length, meaning how many data points the model used as the input | 30 |
| N_OUTPUTS | The number of output sequence length, meaning how many data points in the future that model will generate | 7 |
| Model architecture hyperparameters | | |

| | | |
|---|---|---|
| HIDDEN_SIZE | The dimensions of memory vector, which consist of hidden state and cell state | 64 |
| NUMBER_LAYERS | Number of layers in each LSTM layer in both encoder and decoder | 1 |
| DROPOUT_RATE | The rate for dropout layer | 0 |
| Training hyperparameters | | |
| LEARNING_RATE | The learning rate used in gradient descent formula | 0.001 |
| NUM_EPOCHS | The number of epochs to be used during model training | 1051 |

Figure 4.2.2: Optimal hyperparameters

### 4.2.3 Training

The detailed and step-by-step instructions for training this model can be found in this notebook:

For auto-regressive model: Notebook

For non auto-regressive model: Notebook

### 4.2.4 Loss Curve during Training

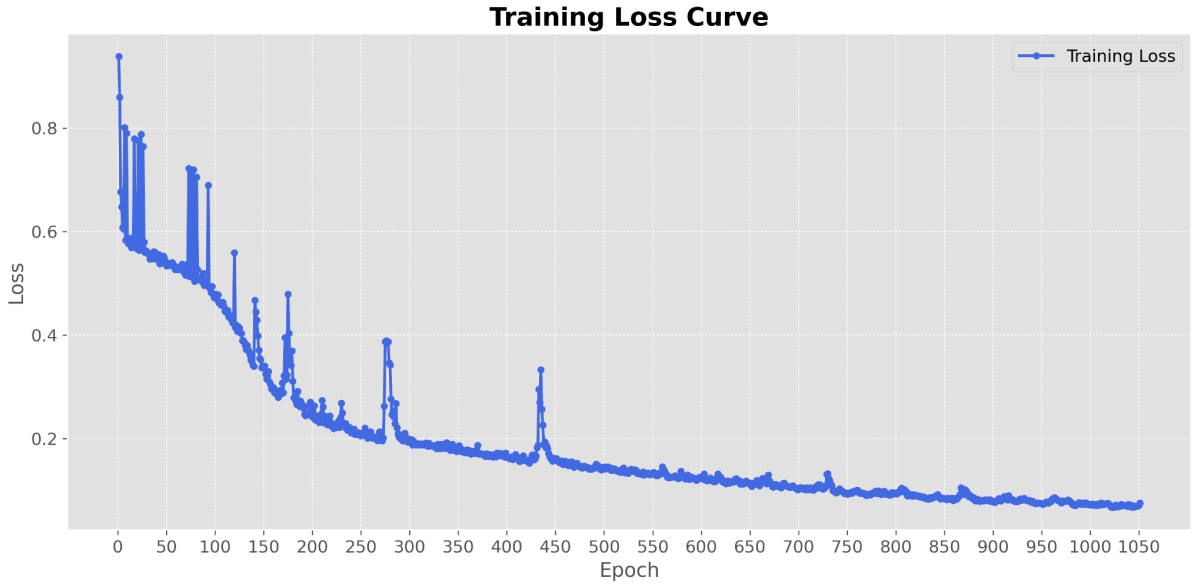



Figure 4.2.3: Loss curve training for auto-regressive Seq2seq GRU Model

From Figure 4.2.3, we observe that the training loss for the auto-regressive Seq2Seq GRU model decreases steadily over time, eventually approaching near zero after around 1050 epochs. However, the curve exhibits noticeable noise and occasional spikes throughout

training. This is expected for an auto-regressive model, where the decoder relies on its own previous predictions, leading to error accumulation and less stable training dynamics.

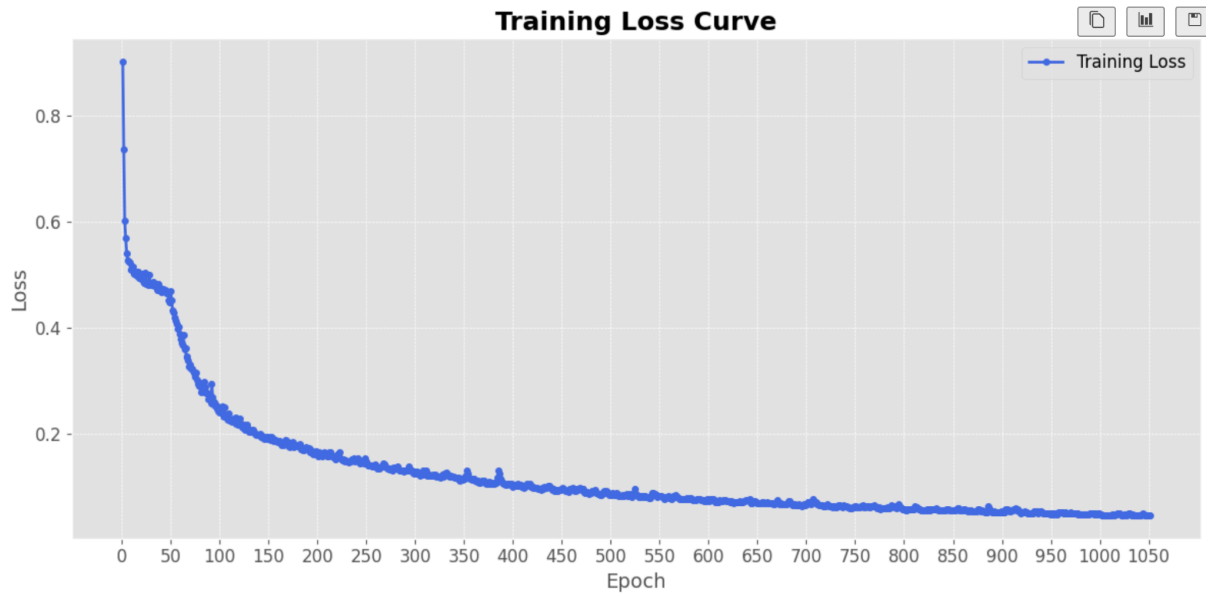



Figure 4.2.4: Loss curve training for non auto-regressive Seq2seq GRU  model

In contrast, the training loss curve for the non-auto-regressive Seq2Seq GRU model, shown in Figure 4.2.4, is significantly smoother and demonstrates a more consistent downward trend. By using teacher forcing during training—where ground truth outputs are provided as inputs at each step—the model converges faster and avoids the instability seen in the auto-regressive setting. As a result, training appears more stable and efficient.

## 4.2.5 Model Evaluation

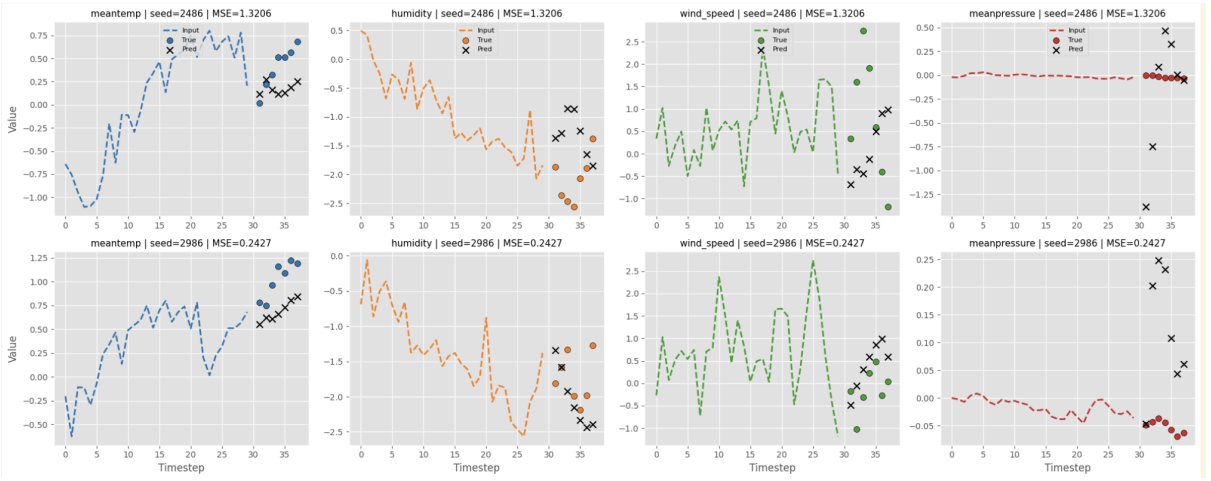



Figure 4.2.5 Model evaluation on auto-regressive model

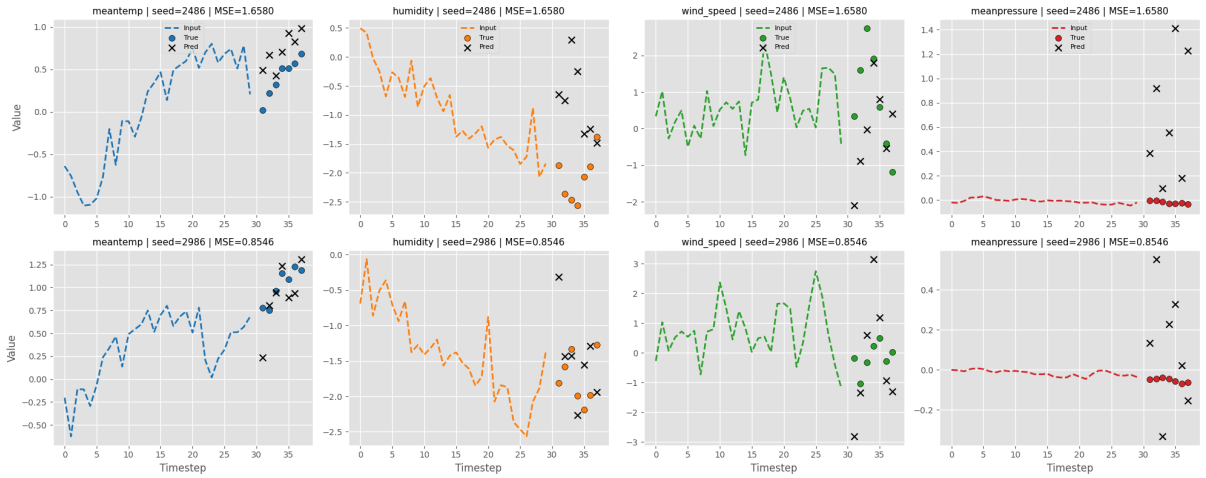



Figure 4.2.6: Model evaluation on non auto-regressive model

# 4.3 Transformer

## 4.3.1 Architecture

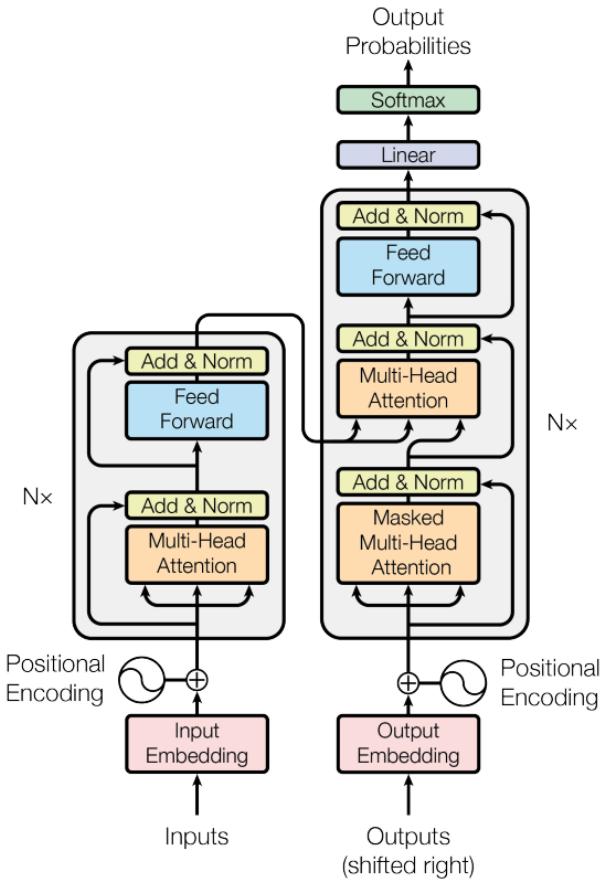



Figure 4.3.1: The Transformer - model architecture (Vaswani et al., 2023)

Similar to the LSTM and GRU models, in this section, we explore the architecture and training implementation of the model trained using the Seq2Seq Transformer architecture.

Our Transformer model accepts an input sequence of 4 weather features - mean temperature, humidity, wind speed, and mean pressure - and predicts future values of the same features over a predefined horizon. Like its RNN-based counterparts, the Transformer also follows an encoder-decoder structure, but replaces recurrence with self-attention mechanisms.

It is important to clarify that the model we implemented is not just an application of the attention mechanism, but a complete Sequence-to-Sequence (Seq2Seq) model based on the Transformer architecture. This terminology aligns with the original “Attention is All You Need” paper, which introduced the Transformer as an encoder-decoder framework for sequence modelling. Because our implementation adheres to this encoder-decoder design - with both a Transformer encoder and a Transformer decoder - it is appropriate to describe it as a Seq2Seq Transformer rather than just an attention-based model.

The Transformer architecture enables parallel computation across time steps and facilitates learning of long-range dependencies via self-attention. These capabilities make it especially suitable for modelling multivariate weather time series data, where complex temporal and cross-feature interactions often exist.

The Time Series Embedding layer first maps the 4-dimensional raw input vector into a higher-dimensional space (`d_model`). This transformation is performed using a simple linear layer and ensures that all input features are represented in a unified latent space suitable for the Transformer.

To address the lack of sequential awareness in the model,  Positional Encoding is added to the embeddings. These encodings use sinusoidal functions to inject timestep-related information, ensuring the model can differentiate between the order of events in the input sequence. The positional encodings are fixed and non-learnable, allowing the model to generalize to longer sequences.

The Transformer Encoder is composed of stacked PyTorch `nn.TransformerEncoderLayer` blocks. Each block includes:

- Multi-head self-attention: Allows the model to attend to multiple positions to the input simultaneously.
- Feedforward networks: A two-layer fully connected network applied independently to each time step.

- Add and Norm layers: Residual connections followed by layer normalization ensure stable and efficient learning.

The encoder processes the embedded and positionally encoded input sequence in parallel, capturing complex dependencies across time steps.

The Transformer Decoder is similarly built from stacked PyTorch `nn.TransformerDecoderLayer` blocks. Each block includes:

- Masked multi-head self-attention: Prevents a timestep from accessing future positions
- Cross-attention: Enabling the decoder to reference the encoded input sequence.
- Feedforward network and Add and Norm layers, as in the encoder.

A key component of the decoder is the causal attention mask. This mask is generated as an upper-triangular matrix where elements above the main diagonal are filled with `-inf`. When passed through the softmax function in the attention mechanism, these `-inf` values yield zero attention weights, ensuring the model only considers current and past time steps. This structure enforces autoregressive behaviour during training and is a standard approach in Transformer-based sequence generation.

During training, we use teacher forcing, feeding the decoder ground truth outputs to improve convergence. During inference, the model operates autoregressively, generating one step at a time using its own predictions.

Finally, the decoder output is passed through a linear layer to return to the original 4-dimensional feature space, producing the forecasted sequence.

### 4.3.2 Optimal Hyperparameters

| Variables | Hyperparameter Details | Values |
|---|---|---|
| Data dimensions | | |
| N_INPUTS | The number of input sequence length, meaning how many data points the model used as the input | 30 |
| N_OUTPUTS | The number of output sequence length, meaning how many data points in the future | 7 |

| | | |
|---|---|---|
| | that model will generate | |
| Model architecture hyperparameters | | |
| HIDDEN_SIZE | The dimensions of memory vector, which consist of hidden state and cell state | 64 |
| NHEAD | The number of attention heads | 4 |
| NUMBER_LAYERS | Number of encode and decoder layers | 2 |
| DROPOUT_RATE | The rate for dropout layer | 0.2 |
| Training hyperparameters | | |
| LEARNING_RATE | The learning rate used in gradient descent formula | 0.001 |
| NUM_EPOCHS | The number of epochs to be used during model training | 300 |

### 4.3.3 Training

The detailed and step-by-step instructions for training this model can be found in this Notebook.

### 4.3.4 Loss Curve during Training:

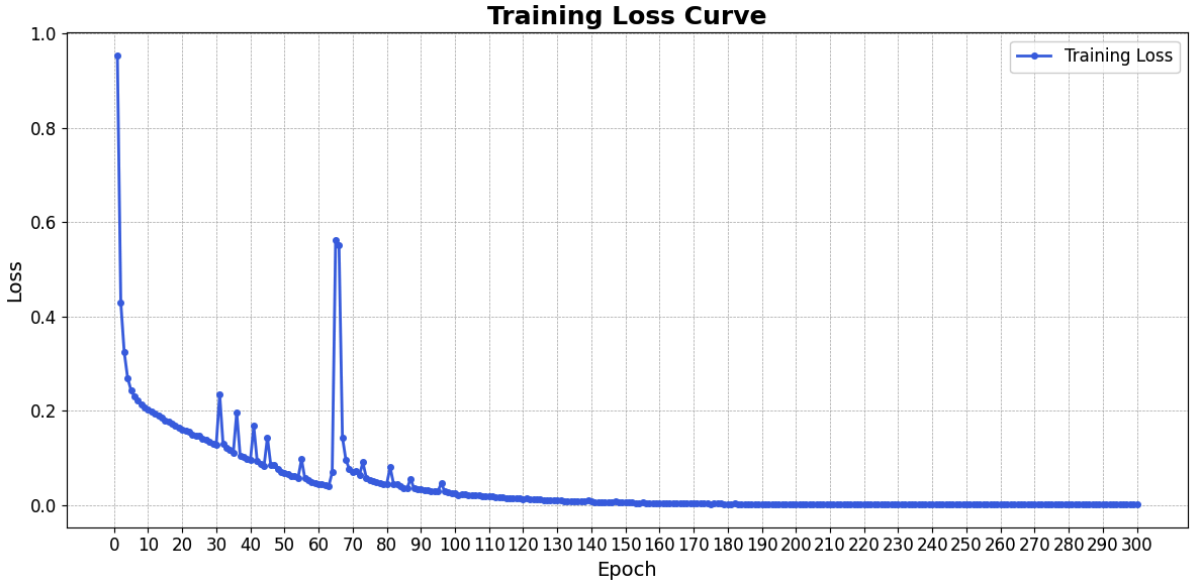



Figure 4.3.2: Training Loss Curve for Seq2Seq Transformer model

The training loss curve for the Transformer model shows a steep initial decline, indicating rapid learning in the early stages of training. The loss decreases consistently over the epochs, demonstrating successful optimization. Notably, there are several visible spikes in

the loss, particularly around epochs 60 and 70, which could be attributed to fluctuations in batch gradients, unstable learning rates, or noisy training samples. Despite these temporary disruptions, the model quickly recovers and continues to converge smoothly. From approximately epoch 100 onward, the loss flattens and approaches zero, suggesting that the model has stabilized and effectively learned the underlying patterns in the data. This overall trend confirms that the training process was effective, though the brief instability warrants consideration for potential improvements in regularization or optimizer configuration.

### 4.3.5 Model Evaluation

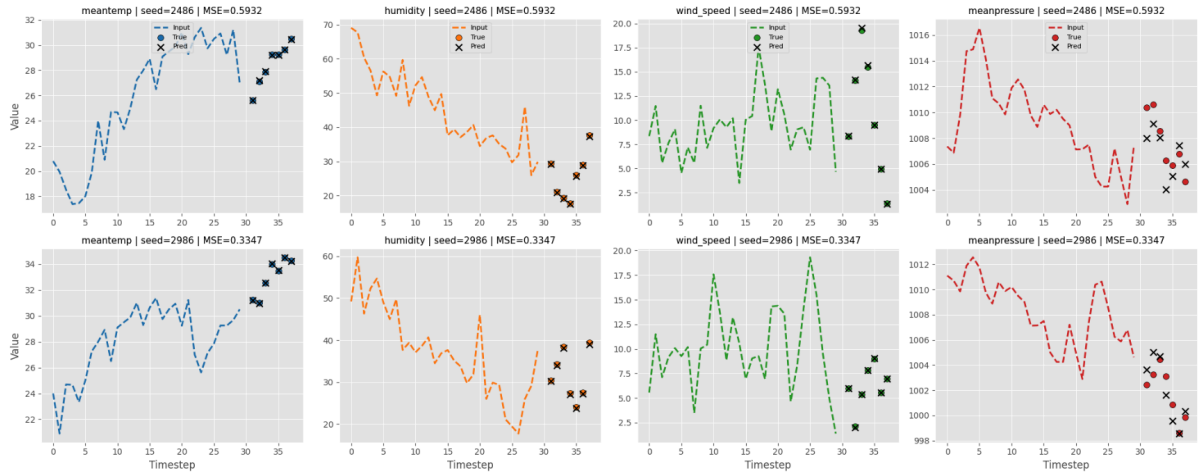



Figure 4.3.3: Model evaluation on Seq2Seq Transformer model

# 5. Comparison among models

In this section, we provide a comparative analysis of the performance of all the models trained in this project. To ensure a fair and consistent evaluation, we focus on the Mean Squared Error (MSE) computed on the **test set** for each model, averaged across the 2 runs on separate seeds. This metric allows us to objectively assess and rank the models based on their predictive accuracy.

| Model | MSE (on Test Set) |
|---|---|
| Seq2Seq LSTM (Auto-regressive) | *0.8637* |
| Seq2Seq LSTM (Non-Auto-regressive) | *0.8625* |
| Seq2Seq GRU (Auto-regressive) | *0.7817* |
| Seq2Seq GRU (Non-Auto-regressive) | *0.5784* |
| Transformer | *0.4640* |

**Observations:**

- **Non-Auto-regressive models consistently outperform Auto-regressive models** across both LSTM and GRU architectures. The use of teacher forcing during training enables the model to better learn the underlying dynamics without suffering from the compounding error typically seen in auto-regressive generation.
- Between the two RNN variants, **GRU-based models achieved lower MSE than LSTM-based models**. This highlights the efficiency of GRU cells in capturing temporal dependencies with fewer parameters, making them a better fit for our moderately-sized dataset.
- The **Seq2Seq GRU (Non-Auto-regressive) model significantly outperformed all other RNN-based models**, achieving an MSE of 0.5784, demonstrating the strong effectiveness of combining GRU with non-auto-regressive decoding.
- The **Transformer model achieved the best overall performance**, with the lowest test MSE of 0.4640. Its attention mechanisms likely allowed it to model long-range dependencies and intricate feature interactions more effectively than traditional recurrent networks.
- While LSTM models showed the highest MSE among the architectures tested, the performance gap between auto-regressive and non-auto-regressive LSTM was relatively small (0.8637 vs 0.8625), suggesting that for LSTM, teacher forcing had a less dramatic effect compared to GRU.

**Performance Ranking (from Best to Worst):**

1. Transformer (MSE = 0.4640)

2. Seq2Seq GRU (Non-Auto-regressive) (MSE = 0.5784)

3. Seq2Seq GRU (Auto-regressive) (MSE = 0.7817)

4. Seq2Seq LSTM (Non-Auto-regressive) (MSE = 0.8625)

5. Seq2Seq LSTM (Auto-regressive) (MSE = 0.8637)

The experimental results demonstrate the clear benefits of using **teacher forcing** and **Transformer-based architectures** for time series forecasting tasks. The Transformer model led by a significant margin, reinforcing its suitability for problems involving sequential dependencies. However, the Seq2Seq GRU (Non-Auto-regressive) model also achieved strong results, indicating that properly tuned RNN-based models remain competitive options, especially when computational efficiency is a concern. These findings suggest that for future

work, further improvements could be made by combining recurrent and attention mechanisms, or by fine-tuning larger Transformer-based models with specialized regularization techniques.

# 6. Techniques to improve model training

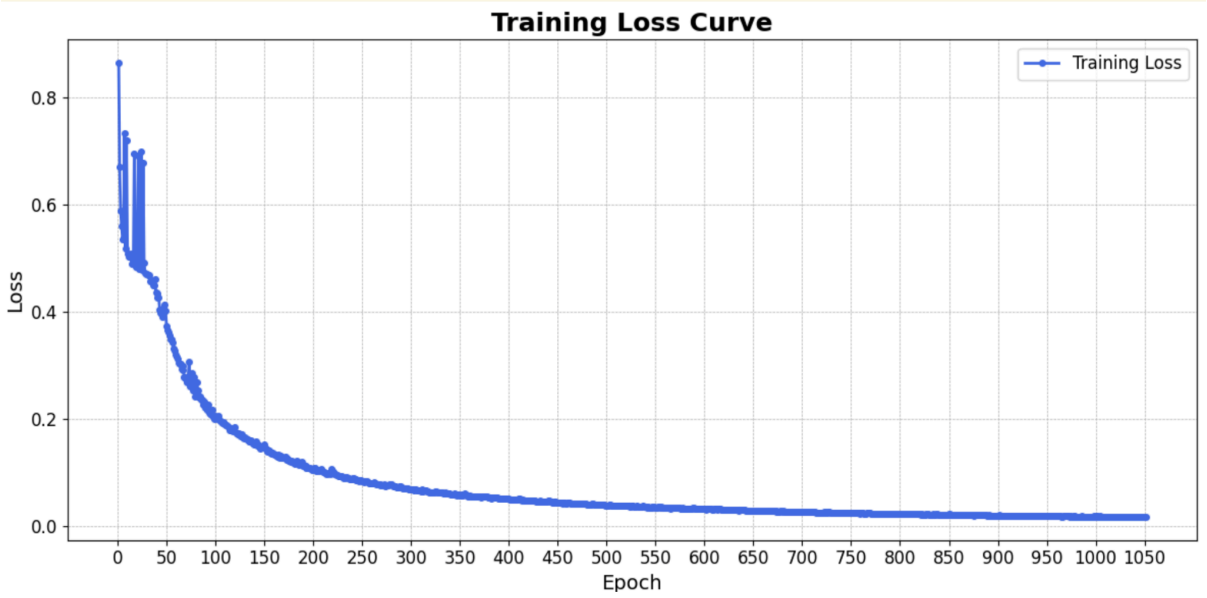
## Seq2Seq GRU

Note: Prior to the following changes we subsequently implemented which we will discuss, we have tried playing around with better initialization, namely:

1. Orthogonal Initialization for GRU weights + Zero or Slightly Positive Bias Initialization
2. Orthogonal Initialization skipping Update Gate Bias Boosting
3. Xavier for Weights + Zero Biases,

but did not find any notable success with said initializations, be it in terms of faster convergence or final lower test MAE.

What did work however, was changing certain hyperparameters, specifically:
**Increase num_layers to 2 with dropout 0.1**



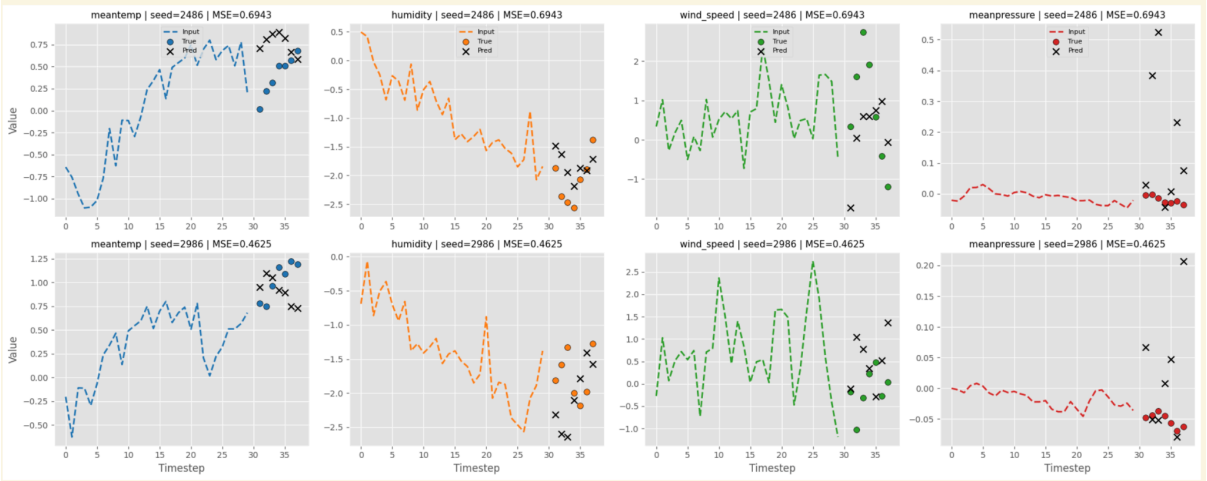

- Training loss drops faster and to a lower value.
- Curve is steeper early on and reaches near-zero around epoch 1000.
- Despite added dropout, which usually adds noise during training, the model still shows better convergence — suggesting improved generalization and learning capacity with the deeper architecture.

**Numerical Evaluation of Metrics**

| **Metric** | **1 Layer, No Dropout** | **2 Layers, Dropout 0.1** | **Improvement** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Final Loss (Epoch 1050) | 0.0416 | 0.0173 | 2.4x lower |
| Test MAE | 0.3725 | 0.2317 | ~37.8% lower |
| Training Time | 2m 6s | 4m 52s | 2.3x longer |

- The 2-layer model with dropout generalizes significantly better — achieving much lower test MAE, not just better training loss.
- The increased training time is expected, but worth the tradeoff given the improved performance.
- The addition of dropout didn't hinder convergence, which typically suggests overfitting was reduced.




| **Aspect** | **Before (1L, No Dropout)** | **After (2L, 0.1 Dropout)** |
|---|---|---|
| Generalization | Poorer, especially in volatility | Stronger, more stable |
| Prediction Spread | Wide and erratic in some plots | Tighter, less scattered |
| Visual Alignment | Often off-target | Closely follows ground truth |
| MSE Reduction | Higher errors across the board | ~50% reduction in MSE |

Increasing num_layers from 1 to 2 along with dropout is a highly effective change:

- Substantially improved generalization (lower MAE and MSE across all features).
- Faster convergence.
- Better resistance to overfitting.

# 7. Challenges and areas for improvements

In our initial implementation plan, we intended to develop a few RNN models from scratch using LSTM and GRU architectures, followed by developing a more advanced model using the Transformer architecture. Finally, we planned to compare those models we trained from scratch to a state-of-the-art pretrained model. However, we encountered configuration issues when using the state-of-the-art model, which prompted us to refine our implementation plan. So, we considered the Transformer-based model as the state-of-the-art, and it will serve as the baseline model against which we will compare all other models. Hence, one of the biggest areas for improvement is to use a pretrained state-of-the-art model in our project.

# 8. Contributions:

| Team member | Student ID | Contributions | Ackknowledgements | Signature |
|---|---|---|---|---|
| Tran Cong Nam Anh - Louis | 1005242 | ● Implemented data exploration and analysis to better understand the dataset's properties and patterns<br>● Designed and standardized the project folder structure<br>● Created and standardized notebook templates for training and evaluating all models across different architectures<br>● Implemented, trained, and evaluated a Seq2Seq RNN model using the LSTM architecture from scratch<br>● Created and finalized the README file<br>● Authored sections of the report related to assigned responsibilities | I hereby declare that every team member has made an equal and valuable contribution to the completion of the project. | Anh |
| Oon Shao Ren | 1005935 | ● Implemented, trained, and evaluated the Seq2Seq GRU models (auto-regressive and non-auto-regressive) from scratch.<br>● Improved GRU model performance by introducing a 2-layer architecture with dropout, significantly reducing test error.<br>● Analyzed training dynamics, loss curves, and led comparative evaluation across all models.<br>● Authored report sections on GRU implementation, training improvements, and model comparison.<br>● Contributed to troubleshooting, model refinement, and optimization | I hereby declare that every team member has made an equal and valuable contribution to the completion of the project. | Oon |

| | | | | |
|---|---|---|---|---|
| | | discussions. | | |
| Benetta Cheng Jia Wen | 1006232 | ● Implemented, trained, and evaluated the Seq2Seq Transformer architecture for time series forecasting from scratch.<br>● Conducted exploratory work on transfer learning using N-BEATS and Temporal Fusion Transformer (TFT), gaining insights despite their exclusion from the final deliverables.<br>● Authored report sections related to my model implementation. | I hereby declare that every team member has made an equal and valuable contribution to the completion of the project. | Cheng |

# 9.Project repository:

https://github.com/LouisAnhTran/deep_learning_models_for_time_series_data

# References

Kumar, A. (2021, December 10). DECODING MACHINE TRANSLATION - Amit Kumar - Medium. Medium. **https://medium.com/@amitbalharakr93/seq2seq-and-lstmseq2seq-and-lstm-af0e9d30e548**

PyTorch. (n.d.). PyTorch. **https://pytorch.org/**

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (2023). Attention Is All You Need. **https://arxiv.org/abs/1706.03762**