50.039 Theory and Practice of Deep Learning Project

Resale Flat Price Prediction

Group 14
1007036 Celest Teng Roh Yee
1006152 Goh Yun Ni Jamie
1006088 Vannara Lim

Introduction

In this project, we aim to develop a deep learning model to predict the resale price of HDB flats in Singapore. This is framed as a **supervised regression task** where the goal is to **estimate a numerical output (resale price)** given a set of input features related to the flat and its location. Understanding and predicting HDB resale prices is valuable for both buyers and sellers in the real estate market, as it enables data-driven decisions and can offer transparency in price estimation.

Dataset

The dataset used for this project is "Resale flat prices based on registration date from Jan-2017 onwards" curated by the Singapore government retrieved on 6th March 2025. It contains historical records of HDB flat resale transactions, including the following features:

- Month
- Town
- Flat Type
- Block
- Street Name
- Storey Range
- Floor Area (sqm)
- Flat Model
- Lease Commence Date / Remaining Lease
- Resale Price (target variable)

Preprocessing

For all experimentations the group applied several key concepts to handle data:

Components	**Description**
Data Loading and Preprocessing	• Loads a CSV file containing resale flat prices.

	• Converts the `month` column to a datetime format and removes invalid dates. • Encodes categorical columns (e.g., `town`, `flat_type`) using `LabelEncoder`. • Normalizes continuous columns (e.g., `floor_area_sqm`, `resale_price`) using `MinMaxScaler`.
Feature Engineering	• Encode `month` into numerical format • Normalisation of input columns • Implements enhanced month encoding with cyclical features (sin/cos) for better temporal representation.
Data Splitting	• Created a function to filter dataset by criteria (e.g. `town`, `flat_type`, `storey_range`) and handle insufficient data.
Visualisation	• Plotted trends such as average resale price by year, price over time by flat type, and price vs remaining lease duration to better understand the data.

Repository

Overview

There are 2 files in this project, both represent separate experimentation to answer the identified machine learning problem of this project.

Running Notebook

1. Create your python environment of choice. In this case, we shall use `conda` environment management

```
# Create conda environment
conda create -n hdb-price-prediction python=3.10

# Activate the environment
conda activate hdb-price-prediction
```

2. Install dependencies

```
pip install -r requirements.txt
```

3. Clone the public GitHub repository

```
git clone https://github.com/vann-lim/50.39DeepLearningProj
```

4. We have already included the dataset inside this repository, you do not need to access it from the web

5. You can now open the notebook file named `grp14.ipynb`

Experimentation Process

Model Exploration and Selection

We began by experimenting with several neural network architectures suitable for time series and tabular data, including **Transformers**, **RNNs, GRUs**, and **LSTMs**. GRUs consistently underperformed in both training and validation, failing to capture temporal patterns effectively in our dataset. LSTM and Transformer-based models, on the other hand, produced comparable results.

Ultimately, we chose to proceed with **LSTM as opposed to transformers** due to its relative simplicity, less number of trainable parameters, and lower computational requirements—making it a pragmatic choice for our problem.

Architecture and Parameter Tuning

After selecting LSTM as our base architecture, we fine-tuned the model's hyperparameters to improve performance. The final architecture is composed of the following components:

- **LSTM Layer**: A multi-layer LSTM configured with `batch_first=True`, where inputs follow the format `[batch_size, sequence_length, input_size]`.
- **Dropout Layer**: A dropout layer set to 0.2, reduced from an earlier 0.3, to prevent overfitting while maintaining model capacity.
- **Linear Layer**: A fully connected layer that maps the hidden state of the LSTM to the output size.
- **Activation Function**: A LeakyReLU activation with a negative slope of 0.01 to allow minor gradient flow for negative values and improve learning stability.

```
class LSTM_pt(nn.Module):
```

```
        def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(LSTM_pt, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.output_size = output_size

        # LSTM layer
        self.lstm = nn.LSTM(input_size, hidden_size,
num_layers=self.num_layers, batch_first=True)

        # Dropout layer
        self.dropout = nn.Dropout(p=0.2)  # Reduced dropout probability from
0.3 to 0.2

        # Fully connected layer
        self.linear = nn.Linear(hidden_size, output_size)

        # Activation function
        self.activation = nn.LeakyReLU(negative_slope=0.01)

        def forward(self, inputs, hidden_state, cell_state):
        # LSTM forward pass
        lstm_out, (h_n, c_n) = self.lstm(inputs, (hidden_state, cell_state))

        # Apply dropout
        dropped_out = self.dropout(lstm_out)

        # Fully connected layer
        output = self.linear(dropped_out)

        # Apply activation function
        output = self.activation(output)

        return h_n, c_n, output
```

Tuning efforts focused on optimizing dropout rate, hidden state size, number of LSTM layers, and the alpha value used in the LeakyReLU activation function, along with optimizer settings.

We observed that increasing the hidden size and number of layers initially led to improved performance, as the model gained more capacity to learn complex patterns. However, beyond a certain point, these additions resulted in overfitting—where the model performed well on the

training data but poorly on validation data. We eventually settled on a configuration that balanced complexity and generalization.

The alpha value in the LeakyReLU activation also had a noticeable effect. For our architecture, smaller alpha values helped improve learning stability and reduce loss in the early stages. However, making alpha too small (close to zero) began to degrade model performance, likely due to restricted gradient flow during backpropagation. We retained an alpha of 0.01, which provided a good balance.

Finally, the dropout rate was reduced from 0.3 to 0.2, which resulted in slightly better validation performance. This suggests that the higher dropout rate may have been too aggressive, unnecessarily limiting the model's learning ability.

Revisiting Data Splitting Strategy

Initially, our model was trained using a single aggregated dataset that combined all available resale transactions, regardless of flat type, town, or storey range. While this approach was straightforward and easy to implement, it introduced several limitations that hindered the model's performance.

Firstly, aggregating diverse flat types and locations into a single training stream diluted the temporal patterns specific to each group. Since different towns or flat types may exhibit unique trends in pricing over time, lumping them together made it difficult for the LSTM to learn meaningful sequential dependencies. This reduced the effectiveness of using a time series model in the first place, as the temporal dynamics were overshadowed by the heterogeneity of the data.

Secondly, this approach introduced high variance and noise, as the model had to learn a single mapping from sequences of very different flat categories to prices. This made the learning process less stable and increased the chances of the model generalizing poorly, especially for underrepresented categories in the data.

In essence, this strategy undermined the core strength of an LSTM, which is designed to capture sequential and temporal dependencies. Without a more structured input segmentation, the LSTM was unable to learn consistent patterns over time, leading to suboptimal results.

Upon consultation with Prof. Matt, we revised our data splitting strategy to better align with the temporal modeling capabilities of our chosen LSTM architecture. Initially, training a single model across the entire dataset resulted in the LSTM being exposed to a highly diverse set of inputs—varying in flat types, towns, and storey ranges. This diversity introduced substantial noise and diluted the temporal signals, effectively reducing the model's reliance on sequence patterns and rendering the LSTM's strength underutilized.

To address this, we restructured our approach by dividing the dataset into multiple homogeneous subsets, each defined by a unique combination of three key categorical features: Flat Type, Town, and Storey Range. A separate LSTM model was trained for each of these subsets, allowing it to specialize in learning the temporal trends specific to that housing segment.

This targeted modeling approach significantly enhanced the ability of each LSTM to capture meaningful time-based patterns, improving the quality and stability of predictions. Moreover, by localizing the data, we reduced variance within each model's input space, resulting in more robust learning and minimized overfitting.

Most importantly, this adjustment reinstated the relevance of using a sequence-based model. The LSTM could now effectively learn time-dependent changes in resale prices without being overwhelmed by conflicting signals from unrelated segments. At inference time, our system selects the appropriate pre-trained LSTM model based on the input's categorical attributes—a model pool strategy that allows for fine-grained, context-aware predictions.

Final Model

The final model architecture adopts a "pool-based LSTM selection" mechanism. Instead of relying on a single generalized model, we maintain a pool of LSTM models, each pre-trained on a unique subset of the dataset defined by a specific combination of three features: Flat Type, Town, and Storey Range.

During inference, when a new data point is given, we extract these three feature values and dynamically select the LSTM model that was trained on that corresponding subset. This ensures that the prediction is made by a model that has been optimized for similar housing characteristics, allowing for more localized learning, context-specific understanding, and ultimately improved accuracy. This approach significantly enhances the model's ability to generalize across distinct subgroups without the need for a one-size-fits-all model.

For evaluation, we divided the data into training, validation, and test sets, ensuring that each model's test set only included unseen samples from its specific combination of features. This was crucial to prevent data leakage and to accurately assess how well each model generalized beyond its training data.

Output Visualisation

To evaluate the performance of our LSTM models, we plotted histograms for each subgroup using two error metrics:

- **Percentage Error**: (Predicted − Actual) / Actual × 100%
- **Absolute Error**: |Predicted − Actual|

What we visualised:

- **Percentage Error Histograms** helped reveal whether models overpredicted or underpredicted.
- **Absolute Error Histograms** showed the typical size and spread of errors in dollar terms.

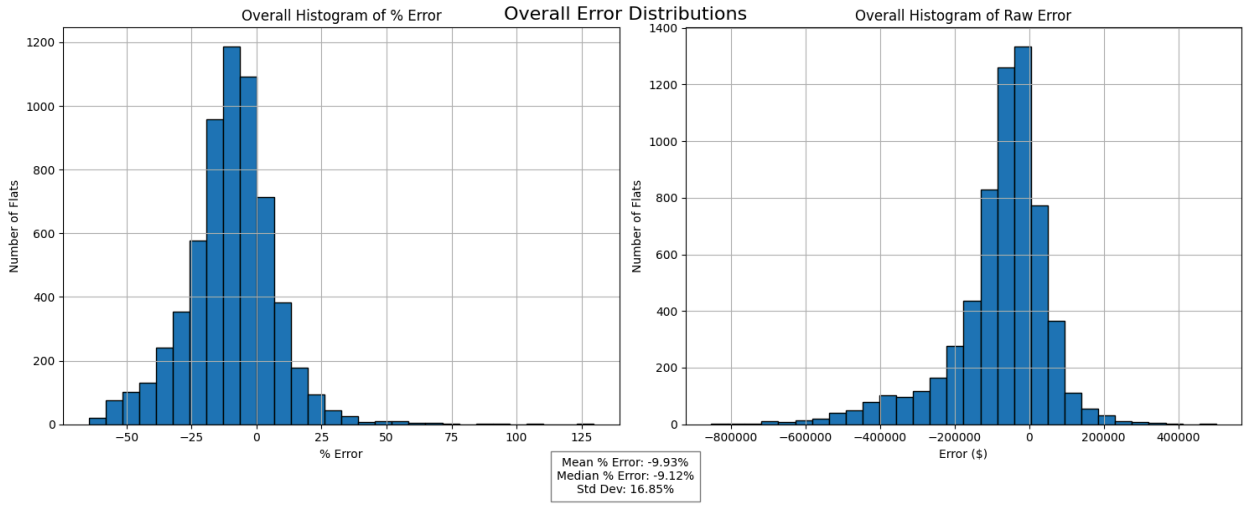
What we observed:

- Most subgroups showed percentage errors clustered around **0%**, indicating accurate predictions.
- Some subgroups had **wider error spreads**, suggesting less consistent performance—likely due to fewer training samples or higher variance in those segments.
- **Flat or heavy-tailed distributions** often appeared in segments with volatile prices or imbalanced data.

Understanding skewness:

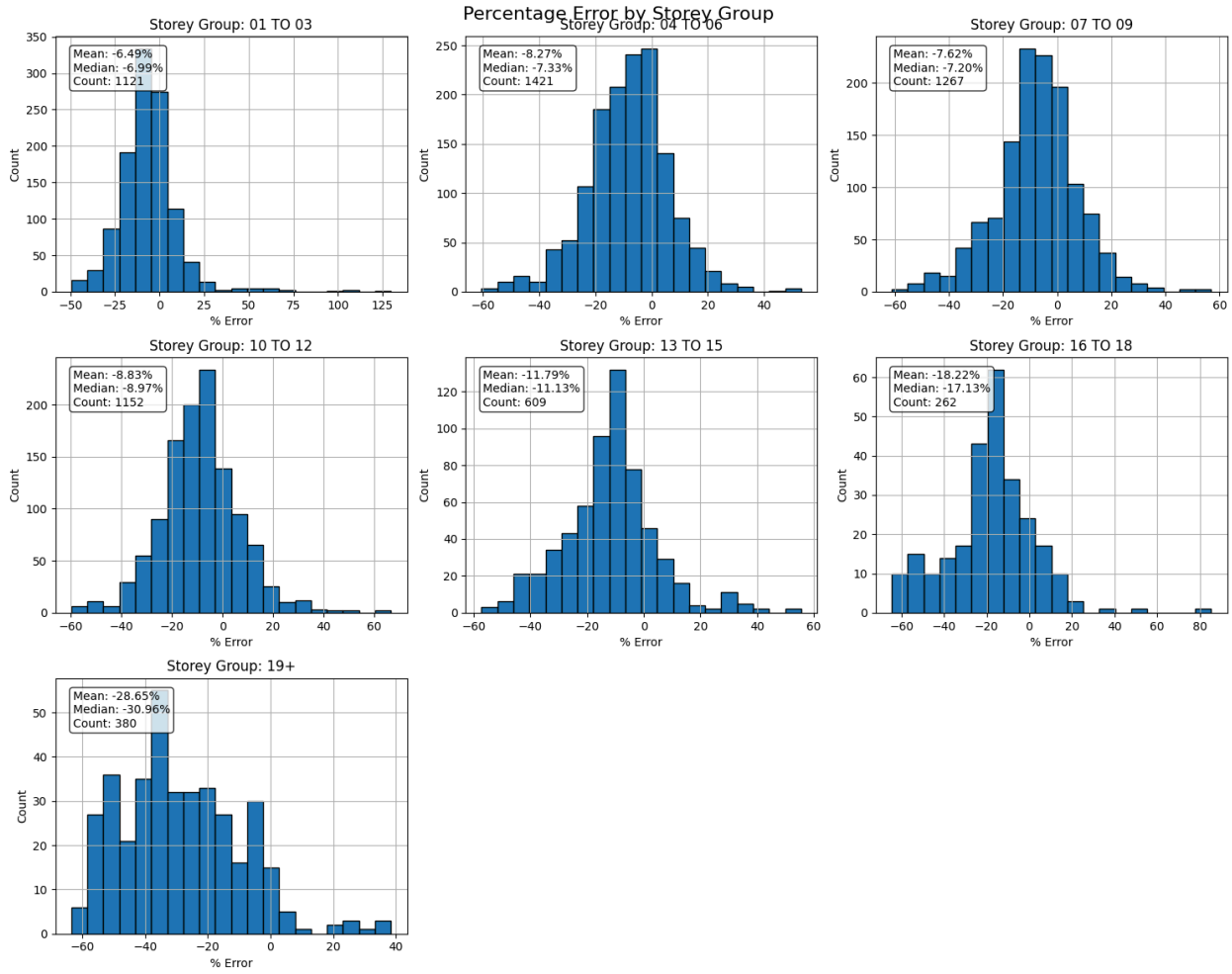
- A **left-skewed** percentage error (peak on the negative side) meant consistent **underprediction**.
- A **right-skewed** histogram pointed to **overprediction**.
- **Symmetric distributions centered around 0%** reflected balanced and ideal model behavior.

The graph below visualises from running the final model, here are the overall histograms of errors. For both percentage and raw errors, we saw that the majority number of flats has close to 0 errors. This is a good indicator of performance.

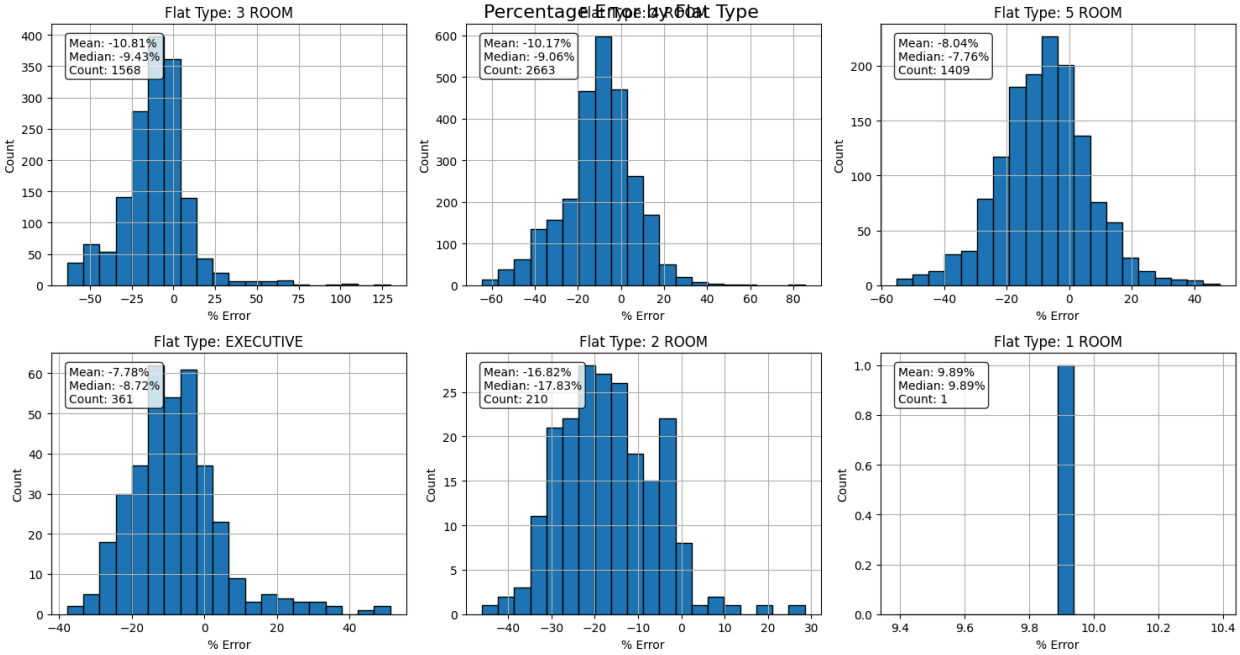


From here, we dived deep to explore the percentage errors in all 3 of the permuting parameters.

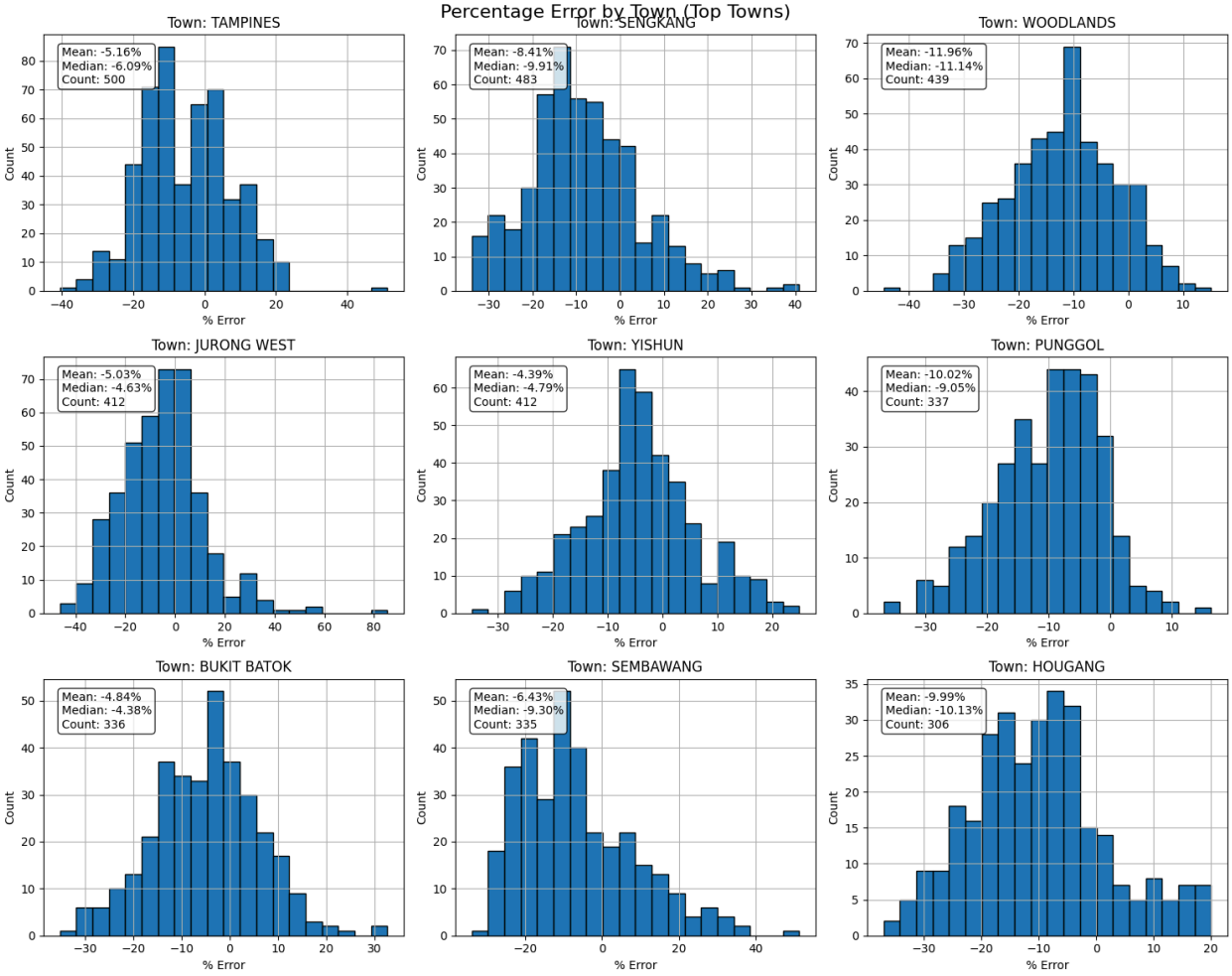
Storey Group



Flat Type


Percentage Error by Flat Type
Flat Type: 3 ROOM
Mean: -10.81%
Median: -9.43%
Count: 1568
Flat Type: 4 ROOM
Mean: -10.17%
Median: -9.06%
Count: 2663
Flat Type: 5 ROOM
Mean: -8.04%
Median: -7.76%
Count: 1409
Flat Type: EXECUTIVE
Mean: -7.78%
Median: -8.72%
Count: 361
Flat Type: 2 ROOM
Mean: -16.82%
Median: -17.83%
Count: 210
Flat Type: 1 ROOM
Mean: 9.89%
Median: 9.89%
Count: 1
Count
% Error

Town


Percentage Error by Town (Top Towns)
Town: TAMPINES
Mean: -5.16%
Median: -6.09%
Count: 500
Town: SENGKANG
Mean: -8.41%
Median: -9.91%
Count: 483
Town: WOODLANDS
Mean: -11.96%
Median: -11.14%
Count: 439
Town: JURONG WEST
Mean: -5.03%
Median: -4.63%
Count: 412
Town: YISHUN
Mean: -4.39%
Median: -4.79%
Count: 412
Town: PUNGGOL
Mean: -10.02%
Median: -9.05%
Count: 337
Town: BUKIT BATOK
Mean: -4.84%
Median: -4.38%
Count: 336
Town: SEMBAWANG
Mean: -6.43%
Median: -9.30%
Count: 335
Town: HOUGANG
Mean: -9.99%
Median: -10.13%
Count: 306
Count
% Error

Reflections

Model Selection: We experimented with a variety of model architectures, including LSTM, GRU, and Transformer-based models. Surprisingly, LSTM demonstrated competitive performance even when trained on the entire dataset, despite signs of overfitting. This suggested that it was able to capture temporal patterns effectively within the available data (like long term price appreciation, preference for location, etc). In contrast, GRU underperformed in both training and validation, and Transformer models, while promising, offered no significant advantage over LSTM in this context. Given these observations and in favor of computational efficiency, we ultimately selected LSTM as the core architecture.

Data Selection: Preprocessing the HDB resale dataset posed several challenges due to its mix of categorical, numerical, and temporal features. Time-based features required special handling to preserve chronological integrity for training and validation splits, especially for time series models. Features like “Remaining Lease” and “Storey Range” also needed custom parsing and conversion to numerical formats. To avoid data leakage (i.e. the introduction of information from outside the training data into the model building process, leading to overly optimistic performance), we ensured that the training and validation sets were strictly split based on time (trained data). Additionally, we chose to drop certain features—such as “Block”—after observing that their inclusion did not improve model performance.

Parameters Tuning: Hyperparameter tuning was essential to achieving meaningful performance but proved to be resource-intensive. We explored several different parameters in training our models, comparing the performance for each set of parameters before arriving at an optimal set. Such parameters include learning rate, batch size, sequence length, number of hidden units, and dropout rate. Given the complexity of the models and training time, we had to strategically limit the search space and number of trials.