National College of Ireland

Configuration Manual for Leveraging Self-Attention in Transformer Models for Improved Retail Demand Forecasting

MSc Research Project
MSc in Data Analytics

Rohith Sardena
Student ID: x23331313

School of Computing
National College of Ireland

Supervisor: John Kelly

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rohith Sardena
Student ID:	X23331313
Programme:	MSc in Data Analytics **Year:** 2024 - 2025
Module:	MSc Research Practicum
Lecturer:	John Kelly
Submission Due Date:	11/08/2025
Project Title:	Leveraging Self-Attention in Transformer Models for Improved Retail Demand Forecasting
Word Count:	856 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rohith Sardena

Date: 11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Leveraging Self-Attention in Transformer Models for Improved Retail Demand Forecasting

Rohith Sardena
Student ID: X23331313

1. Introduction

This manual provides step-by-step instructions to set up the environment, dependencies, and configurations necessary to replicate and execute the research for Leveraging Self-Attention in Transformer Models for Improved Retail Demand Forecasting. The models implemented in this study include Linear Regression, Decision Tree, Random Forest, LSTM, GRU, and a Transformer-based model. The research works with historical retail inventory and demand data (including features like weather, promotions, region, seasonality), using Python and PyTorch for modeling and evaluation.

2. System Hardware Requirements

Ensure your system meets the following minimum requirements:

- **Operating System:** Ubuntu 18.04/20.04 or Windows 10/11 (Linux-based preferred)
- **Processor:** Intel Core i5 / AMD Ryzen 5 or higher (Quad-Core recommended)
- **RAM:** Minimum 16 GB (32 GB recommended for large-scale datasets)
- **GPU:** NVIDIA GPU with CUDA support (optional but strongly recommended for deep learning training)
- **Disk Space:** At least 20 GB free for dataset, dependencies, and results storage

3. Software Requirements:

- **Python:** Version 3.8 or newer
- **Core libraries:** pandas, numpy, seaborn, matplotlib, scikit-learn
- **Deep Learning:** PyTorch (torch and torchvision)
- **Utilities:**
  - torch.utils.data.Dataset & DataLoader
  - LabelEncoder, StandardScaler, MeanAbsoluteError, MSELoss
- **Development environments:** Jupyter Notebook, VS Code, or PyCharm
- **Optional:** Conda or virtualenv for environment isolation

4. Dataset Details

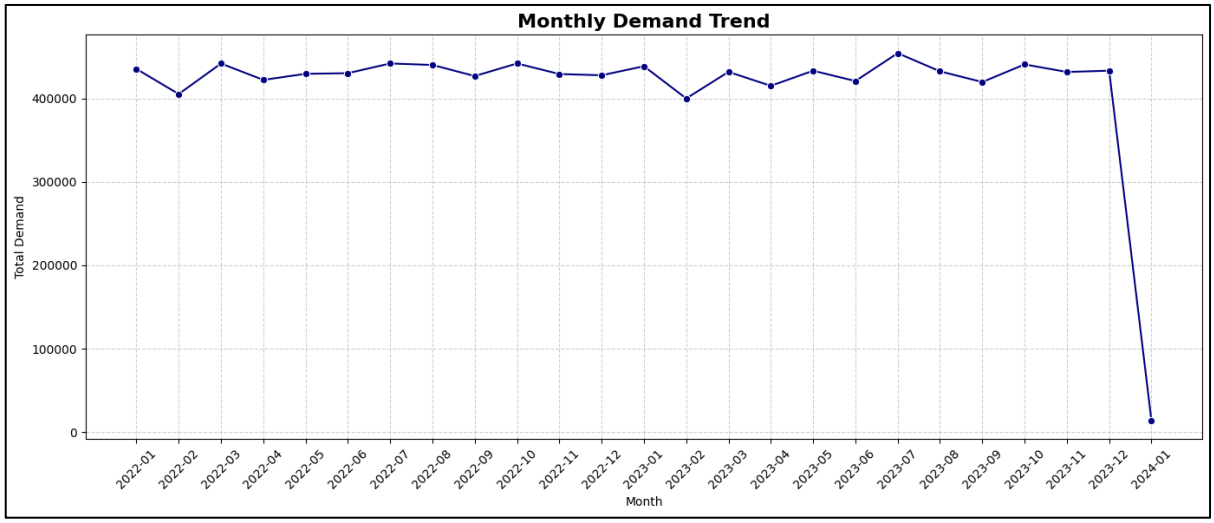
Kaggle Source: Retail Store Inventory Forecasting Dataset by Anirudh Chauhan on Kaggle

4.1 Dataset Description

This retail inventory forecasting dataset provides synthetic but realistic daily records (~73,000+ rows) across multiple stores and products, including features such as sales, inventory, pricing, promotions, holidays, weather, and more—aimed to simulate real-world retail behavior.

Feature of the Dataset:

Feature Name	Data Type	Description
`Date`	Object (Date)	Date of the record (likely needs to be converted to datetime format).
`Store ID`	Object	Unique identifier for each retail store.
`Product ID`	Object	Unique identifier for each product.
`Category`	Object	Category of the product (e.g., Electronics, Groceries).
`Region`	Object	Geographical region of the store location.
`Inventory Level`	Integer	Number of product units available in inventory at the store.
`Units Sold`	Integer	Number of units sold on the given date.
`Units Ordered`	Integer	Number of units ordered to replenish inventory.
`Demand Forecast`	Float	Predicted demand for the product on the given date.
`Price`	Float	Selling price of the product.
`Discount`	Integer	Discount applied on the product (likely in percentage or monetary value).
`Weather Condition`	Object	Weather condition on the given date (e.g., Sunny, Rainy).
`Holiday/Promotion`	Integer	Binary flag indicating a holiday or promotion event (1 = Yes, 0 = No).
`Competitor Pricing`	Float	Price of similar products from competitors.
`Seasonality`	Object	Season tag (e.g., Summer, Winter), capturing seasonal effects.



5. Model Configuration

5.1 Baseline Models

Default algorithms and hyperparameters:

```
# Feature Engineering to extract features
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['Day'] = data['Date'].dt.day
data['DayOfWeek'] = data['Date'].dt.dayofweek
data.drop(columns=['Date'], inplace=True)
```

```
# import the LabelEncoder and StandardScaler Module for Data Converting and Scaling of the data
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
# Label encode categorical columns
from sklearn.preprocessing import LabelEncoder
for col in data.select_dtypes(include='object').columns:
    data[col] = LabelEncoder().fit_transform(data[col].astype(str))
```

```
# Fill any missing values
data.fillna(method='ffill', inplace=True)
```

```
# Define features and target
X = data.drop(columns=['Demand'])
y = data['Demand']
```

- **Linear Regression**
  - No hidden layers, standard solver
  - Optimized with Least Squares
- **Decision Tree**
  - Default depth unless tuned
- **Random Forest**
  - 100 trees; bootstrap-enabled; random_state=42

```python
# Define baseline models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "Decision Tree": DecisionTreeRegressor(random_state=42)
}
```

```python
# model evaluation metrics libraries
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    median_absolute_error,
)
```

```python
# Train and evaluate
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    median_ae = median_absolute_error(y_test, y_pred)

    results[name] = {
        'MAE': mae,
        'MSE': mse,
        'RMSE': rmse,
        'MedianAE': median_ae,
```

Each model trained using features and target (Demand) with scaling via StandardScaler.

```
Model Evaluation Summary:
                            MAE         MSE       RMSE  MedianAE
Linear Regression      7.471644   74.798376   8.648605   7.45321
Random Forest          7.553444   77.784750   8.819566   7.36540
Decision Tree         10.091956  152.546690  12.350979   8.90000
```

5.2 Deep Learning Models (LSTM / GRU)

Configured using a sliding window time-series setup:

- **Window Size:** 30 days
- **Forecast Horizon:** 1 day ahead

LSTM Configuration:

- Layer type: LSTM
- Units: 64 hidden units
- Layers: 2
- Dropout: 0.2

- Optimizer: Adam
- Loss: MSE
- Epochs: 10

```python
# Define LSTM Model
class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim=64, num_layers=2, dropout=0.2):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers=num_layers,
                            batch_first=True, dropout=dropout)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        out, _ = self.lstm(x)
        return self.fc(out[:, -1, :]).squeeze()

# Initialize model and train
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
input_dim = train_data.shape[1] - 1

lstm_model = LSTMModel(input_dim=input_dim).to(device)
optimizer = optim.Adam(lstm_model.parameters(), lr=0.001)
criterion = nn.MSELoss()

EPOCHS = 10
for epoch in range(EPOCHS):
    lstm_model.train()
    train_loss = 0
    for x_batch, y_batch in train_loader:
        x_batch, y_batch = x_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        output = lstm_model(x_batch)
        loss = criterion(output, y_batch)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    print(f"LSTM Epoch {epoch+1}/{EPOCHS}, Loss: {train_loss:.4f}")
```

```python
# Evaluate LSTM
lstm_model.eval()
y_true_lstm, y_pred_lstm = [], []

with torch.no_grad():
    for x_batch, y_batch in test_loader:
        x_batch = x_batch.to(device)
        preds = lstm_model(x_batch).cpu().numpy()
        y_pred_lstm.extend(preds)
        y_true_lstm.extend(y_batch.numpy())

lstm_mae = mean_absolute_error(y_true_lstm, y_pred_lstm)
lstm_rmse = np.sqrt(mean_squared_error(y_true_lstm, y_pred_lstm))
lstm_mse = mean_squared_error(y_true_lstm, y_pred_lstm)
lstm_median_ae = median_absolute_error(y_true_lstm, y_pred_lstm)

results["LSTM"] = {
    "MAE": lstm_mae,
    "MSE": lstm_mse,
    "RMSE": lstm_rmse,
    "MedianAE": lstm_median_ae
}
```

GRU Configuration:
Same setup as LSTM using GRU recurrent cell

```python
# implementation of GRU Model
class GRUModel(nn.Module):
    def __init__(self, input_dim, hidden_dim=64, num_layers=2, dropout=0.2):
        super(GRUModel, self).__init__()
        self.gru = nn.GRU(input_dim, hidden_dim, num_layers=num_layers,
                          batch_first=True, dropout=dropout)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        out, _ = self.gru(x)
        return self.fc(out[:, -1, :]).squeeze()

# Initialize GRU model
gru_model = GRUModel(input_dim=train_data.shape[1] - 1).to(device)
optimizer = torch.optim.Adam(gru_model.parameters(), lr=0.001)
criterion = nn.MSELoss()

# Training loop
EPOCHS = 10
for epoch in range(EPOCHS):
    gru_model.train()
    train_loss = 0
    for x_batch, y_batch in train_loader:
        x_batch, y_batch = x_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        preds = gru_model(x_batch)
        loss = criterion(preds, y_batch)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    print(f"GRU Epoch {epoch+1}/{EPOCHS}, Loss: {train_loss:.4f}")
```

```python
# evaluate the GRU model
gru_model.eval()
y_true_gru, y_pred_gru = [], []

with torch.no_grad():
    for x_batch, y_batch in test_loader:
        x_batch = x_batch.to(device)
        preds = gru_model(x_batch).cpu().numpy()
        y_pred_gru.extend(preds)
        y_true_gru.extend(y_batch.numpy())

gru_mae = mean_absolute_error(y_true_gru, y_pred_gru)
gru_rmse = np.sqrt(mean_squared_error(y_true_gru, y_pred_gru))
gru_mse = mean_squared_error(y_true_gru, y_pred_gru)
gru_median_ae = median_absolute_error(y_true_gru, y_pred_gru)

results["GRU"] = {
    "MAE": gru_mae,
    "MSE": gru_mse,
    "RMSE": gru_rmse,
    "MedianAE": gru_median_ae
}
```

5.3 Transformer Model

Architecture Settings:

- Input projection: Linear layer from input_dim → d_model

- d_model = 64;
- Number of heads (nhead): 4;
- Encoder layers: 2;
- Dropout: 0.1

Implementation of the Transformer Model

```
# Define window and prediction horizon
WINDOW_SIZE = 30   # Number of past days to use for forecasting
HORIZON = 1        # Predict 1 step ahead

# PyTorch dataset class for time-series windows
class RetailDemandDataset(Dataset):
    def __init__(self, data, target_col, window_size, horizon):
        self.window_size = window_size
        self.horizon = horizon
        self.target_col = target_col
        self.features = data.drop(columns=[target_col]).values
        self.target = data[target_col].values

    def __len__(self):
        return len(self.features) - self.window_size - self.horizon

    def __getitem__(self, idx):
        x = self.features[idx:idx + self.window_size]
        y = self.target[idx + self.window_size + self.horizon - 1]
        return torch.tensor(x, dtype=torch.float32), torch.tensor(y, dtype=torch.float32)
```

```
# Define Transformer Model Architecture
class TimeSeriesTransformer(nn.Module):
    def __init__(self, input_dim, d_model=64, nhead=4, num_layers=2, dropout=0.1):
        super(TimeSeriesTransformer, self).__init__()
        self.input_proj = nn.Linear(input_dim, d_model)  # Project input features to model dimension
        encoder_layer = nn.TransformerEncoderLayer(d_model=d_model, nhead=nhead, dropout=dropout)
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.regressor = nn.Linear(d_model, 1)  # Final output layer for regression

    def forward(self, src):
        # Input: [batch_size, sequence_length, input_dim]
        src = self.input_proj(src)              # Shape: [batch, seq_len, d_model]
        src = src.permute(1, 0, 2)              # Convert to [seq_len, batch, d_model] for Transformer
        transformer_out = self.transformer(src) # Pass through transformer
        output = transformer_out[-1, :, :]      # Use last time step output
        return self.regressor(output).squeeze() # Return predicted demand
```

Training Setup:

- Window-based input (batch × seq_len × features), permuted to (seq_len × batch × d_model)
- Regression head: final linear layer to scalar output
- Loss: MSELoss; Optimizer: Adam; Epochs: 10

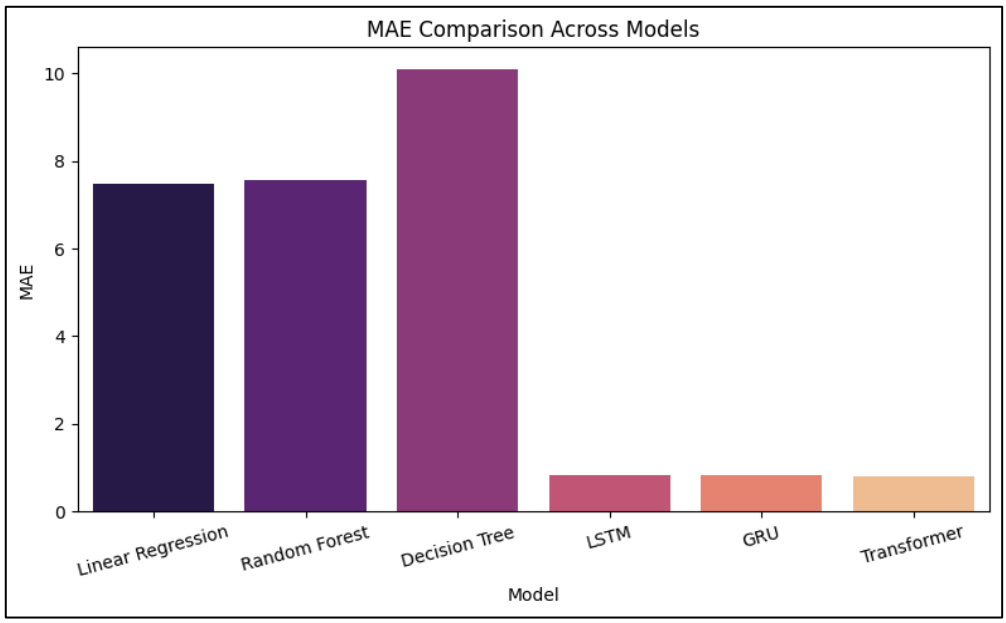
```python
# Train the Transformer Model
# Use GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Initialize model, loss, optimizer
transformer_model = TimeSeriesTransformer(input_dim=train_data.shape[1] - 1).to(device)
optimizer = torch.optim.Adam(transformer_model.parameters(), lr=0.001)
criterion = nn.MSELoss()

# Training loop
EPOCHS = 10
for epoch in range(EPOCHS):
    transformer_model.train()
    epoch_loss = 0
    for x_batch, y_batch in train_loader:
        x_batch, y_batch = x_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        preds = transformer_model(x_batch)
        loss = criterion(preds, y_batch)
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()

    print(f"Epoch {epoch+1}/{EPOCHS}, Loss: {epoch_loss:.4f}")
```

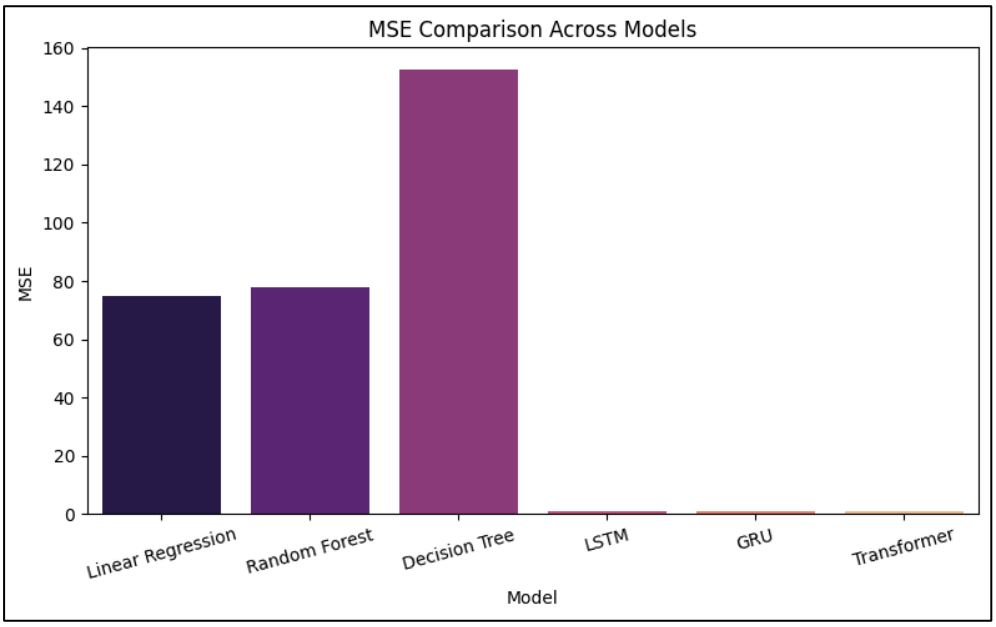
```python
# Evaluate Model Performance
transformer_model.eval()
true_vals = []
pred_vals = []

# Disable gradient calculation for evaluation
with torch.no_grad():
    for x_batch, y_batch in test_loader:
        x_batch = x_batch.to(device)
        output = transformer_model(x_batch).cpu().numpy()
        pred_vals.extend(output)
        true_vals.extend(y_batch.numpy())

# Convert to numpy arrays (in case they are still lists or nested)
true_vals = np.array(true_vals).squeeze()
pred_vals = np.array(pred_vals).squeeze()

# Calculate metrics
transformer_mae = mean_absolute_error(true_vals, pred_vals)
transformer_mse = mean_squared_error(true_vals, pred_vals)
transformer_rmse = np.sqrt(transformer_mse)
transformer_median_ae = median_absolute_error(true_vals, pred_vals)

# Print results
print(f"Transformer MAE: {transformer_mae:.4f}")
print(f"Transformer MSE: {transformer_mse:.4f}")
print(f"Transformer RMSE: {transformer_rmse:.4f}")
print(f"Transformer Median AE: {transformer_median_ae:.4f}")
```

```
Transformer MAE: 0.8106
Transformer MSE: 0.9868
Transformer RMSE: 0.9934
Transformer Median AE: 0.7419
```

Model Evaluation Summary:

	MAE	MSE	RMSE	MedianAE
Linear Regression	7.471644	74.798376	8.648605	7.453210
Random Forest	7.553444	77.784750	8.819566	7.365400
Decision Tree	10.091956	152.546690	12.350979	8.900000
LSTM	0.833684	1.030370	1.015071	0.759184
GRU	0.832551	1.019778	1.009841	0.770514
Transformer	0.810610	0.986801	0.993379	0.741942





6. Conclusion

This configuration manual outlines every step required to replicate the research workflow—from data ingestion to model training and comparison. Feel free to adapt architectures or hyperparameters as needed for further experimentation. For troubleshooting, review error messages carefully or consult the official documentation for libraries being used.

References

Python Official Documentation
https://www.python.org/doc/

Pandas: Data Analysis Library
https://pandas.pydata.org/

NumPy: Fundamental Package for Numerical Computation
https://numpy.org/

Matplotlib: Python Plotting Library
https://matplotlib.org/

Seaborn: Statistical Data Visualization
https://seaborn.pydata.org/

Scikit-learn: Machine Learning in Python
https://scikit-learn.org/stable/

PyTorch: Deep Learning Framework
https://pytorch.org/

Retail Store Inventory Forecasting Dataset
Anirudh Chauhan, Kaggle
https://www.kaggle.com/datasets/anirudhchauhan/retail-store-inventory-forecasting-dataset

Transformer Model Architecture (Vaswani et al., 2017)
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017).
Attention Is All You Need.