SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.039 Deep Learning

Predicting Blood Sugar Spikes for Diabetic Patients

Group Number: Group 20
Group Members: Vanya Jalan (1006190), Elvern Neylmav Tanny (1006203)

Project Overview

This project focuses on predicting future blood glucose levels in diabetic patients using historical glucose, insulin, carbohydrate, and smartwatch activity data.

Dataset

The dataset used is from the BrisT1D Blood Glucose Prediction Competition on Kaggle.
It includes:

- Historical blood glucose (BG) readings, measured in **milligrams per deciliter (mg/dL)**
- Insulin dosage records, measured in **units (U)**
- Carbohydrate intake logs, measured in **grams (g)**
- Smartwatch-derived activity data:
    - Heart rate, measured in **beats per minute (bpm)**
    - Steps, **count** of steps taken
    - Calories, measured in **kilocalories (kcal)**

The goal is to leverage this multi-modal data to predict future blood glucose levels, specifically, the value one hour ahead (bg+1:00).

Data Cleaning & Processing

The dataset required significant preprocessing due to missing values, varying time series lengths, and sensor noise.

The following steps were applied:

1) To organize and process data more effectively, feature columns were grouped by prefix:

```
glucose_cols = [col for col in df.columns if col.startswith("bg-")]
insulin_cols = [col for col in df.columns if col.startswith("insulin-")]
carbs_cols   = [col for col in df.columns if col.startswith("carbs-")]
hr_cols      = [col for col in df.columns if col.startswith("hr-")]
steps_cols   = [col for col in df.columns if col.startswith("steps-")]
cals_cols    = [col for col in df.columns if col.startswith("cals-")]
```

2) Columns related to activity types (non-numeric) were dropped.

```
df.drop(columns=activity_cols, inplace=True)
```

3) We handled missing values for various feature groups as tabulated below.

Blood Glucose	Row-wise linear interpolation. `df[bg_cols] = df[bg_cols].apply( lambda row: row.interpolate(method='linear', limit_direction='both'), axis=1)`
Insulin & Carbs	Missing values replaced with 0 (assumed no intake/dosage). `df[insulin_cols + carbs_cols] = df[insulin_cols + carbs_cols].fillna(0)`
Heart Rate	Group-wise linear interpolation per participant (p_num). Missing values are replaced with the participant-specific mean. `df[hr_cols] = df[hr_cols].apply( lambda row: row.interpolate(method='linear', limit_direction='both'), axis=1)` `df[hr_cols] = df[hr_cols].fillna(df[hr_cols].mean(axis=1))`
Steps & Calories	Missing values replaced with 0, indicating no activity. `df[steps_cols + cals_cols] = df[steps_cols + cals_cols].fillna(0)`
Other Remaining NaNs	Forward-fill (ffill) followed by backward-fill (bfill) to cleanup leftover NaNs. `df = df.fillna(method='ffill').fillna(method='bfill')`

Data Augmentation

To increase generalization and robustness, Gaussian noise was added to all numeric features (excluding identifiers and the target) -

```
noise_factor = 0.01
```

```
df_aug[col] += np.random.normal(0, noise_factor * std_val, size=len(df))
```

The original and augmented data were then concatenated, effectively doubling the training set size.

Data Selection

Based on evidence from the article by Dave et al. 2020, features derived from recent CGM data such as the previous 30 minutes to 2 hours are significantly more predictive of near-future glycemic events compared to features from longer historical windows (e.g. 6 hours before). Hence, in our data preprocessing pipeline, we perform a data selection step to select only the most recent 24 time steps (approximately 2 hours of data) across glucose, insulin, carbohydrate intake, heart rate, step count, and calories. This approach ensures that the model is trained on the most relevant and predictive features for accurate forecasting of blood sugar spikes.

Feature Scaling

All numeric columns except the target (bg+1:00) were scaled using StandardScaler:

Features: Standard scaling (zero mean, unit variance)

```
feature_scaler = StandardScaler()
df[numeric_columns] = feature_scaler.fit_transform(df[numeric_columns])
```

Target: Scaled separately to preserve transformation logic for inverse prediction later

```
target_scaler = StandardScaler()
if target_col in df.columns:
        df[[target_col]] = target_scaler.fit_transform(df[[target_col]])
```

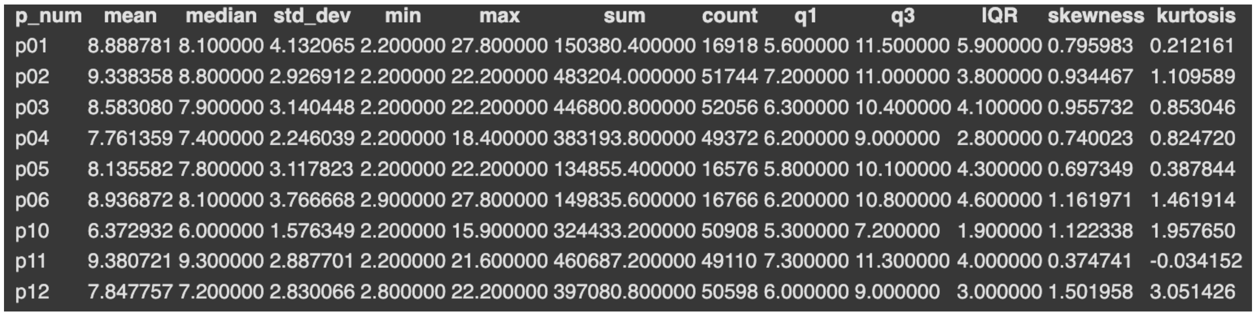
This step ensures faster convergence and better model performance during training.

Descriptive Statistics & Exploratory Analysis

Descriptive statistics were calculated per participant (p_num) for the target column bg+1:00.

Metrics include:

- Central tendency: mean, median
- Dispersion: std, min, max, IQR
- Distribution shape: skewness, kurtosis
- Summary count and sum



p_num	mean	median	std_dev	min	max	sum	count	q1	q3	IQR	skewness	kurtosis
p01	8.888781	8.100000	4.132065	2.200000	27.800000	150380.400000	16918	5.600000	11.500000	5.900000	0.795983	0.212161
p02	9.338358	8.800000	2.926912	2.200000	22.200000	483204.000000	51744	7.200000	11.000000	3.800000	0.934467	1.109589
p03	8.583080	7.900000	3.140448	2.200000	22.200000	446800.800000	52056	6.300000	10.400000	4.100000	0.955732	0.853046
p04	7.761359	7.400000	2.246039	2.200000	18.400000	383193.800000	49372	6.200000	9.000000	2.800000	0.740023	0.824720
p05	8.135582	7.800000	3.117823	2.200000	22.200000	134855.400000	16576	5.800000	10.100000	4.300000	0.697349	0.387844
p06	8.936872	8.100000	3.766668	2.900000	27.800000	149835.600000	16766	6.200000	10.800000	4.600000	1.161971	1.461914
p10	6.372932	6.000000	1.576349	2.200000	15.900000	324433.200000	50908	5.300000	7.200000	1.900000	1.122338	1.957650
p11	9.380721	9.300000	2.887701	2.200000	21.600000	460687.200000	49110	7.300000	11.300000	4.000000	0.374741	-0.034152
p12	7.847757	7.200000	2.830066	2.800000	22.200000	397080.800000	50598	6.000000	9.000000	3.000000	1.501958	3.051426

Figure 1: Participant Statistics (Recreate by running all cells until the cell heading "Visual Analysis" in data_analysis.ipynb)

This allowed for insights into variability between participants and helped inform potential outlier detection and stratified evaluation.

To further analyze patterns in the target variable (bg+1:00), we conducted visual analyses focused on participant-specific trends and distributions. Histograms and boxplots helped highlight variability and outliers. A skewness-kurtosis scatterplot was used to compare distribution shapes across participants. Additional bar plots summarized standard deviation, IQR, and mean values. Three key visuals: histogram, boxplot, and skewness-kurtosis plot are included below to illustrate the most relevant findings. (Due to the large number of participants, the histogram plot is shown only for participant 12 as a representative example.)

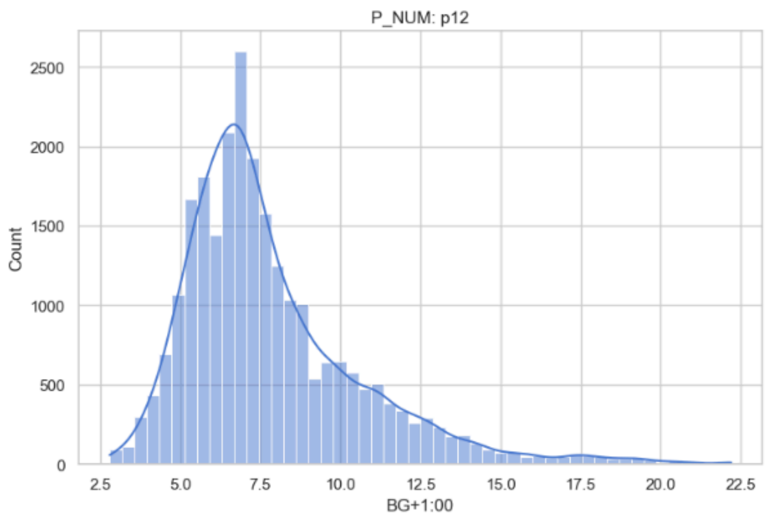


Figure 2: Histogram for Participant 12 (Recreate by running all cells until the cell heading "Histogram: BG+1:00/P_NUM" in data_analysis.ipynb)

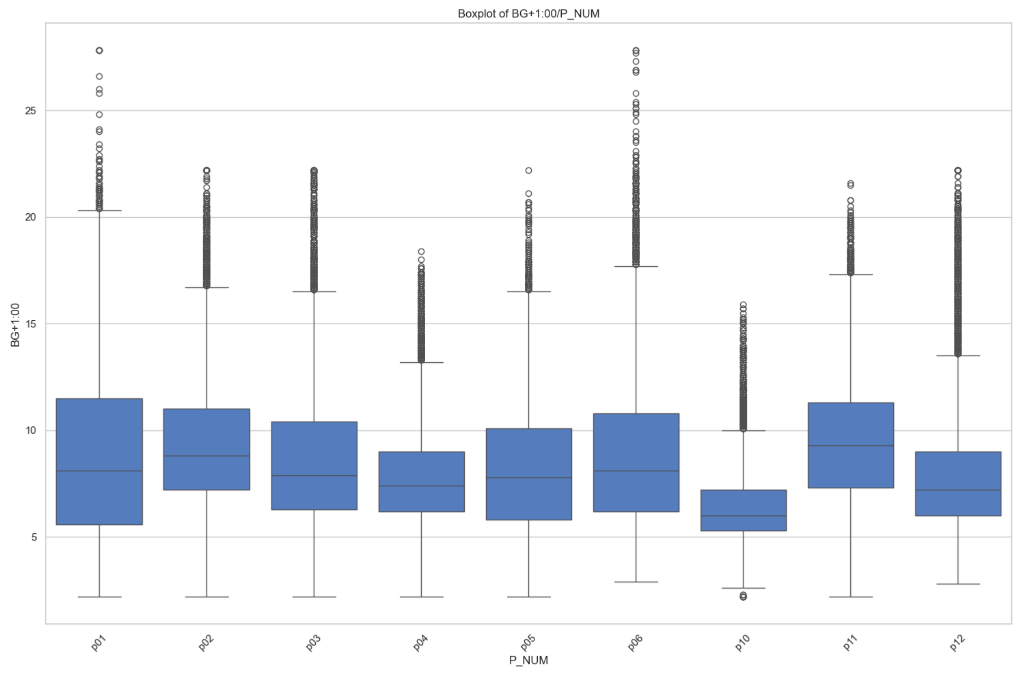


Figure 3: Boxplot for Participant Data (BG+1:00) (Recreate by running all cells until the cell heading "Boxplot: BG+1:00/P_NUM" in data_analysis.ipynb)

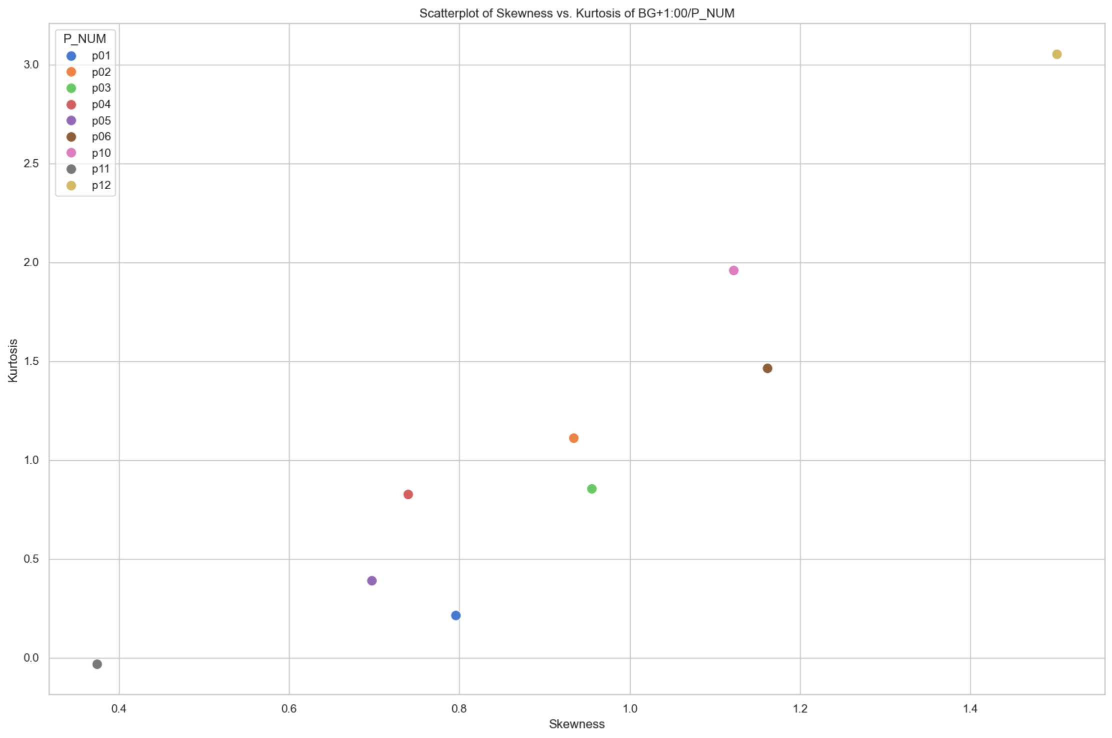
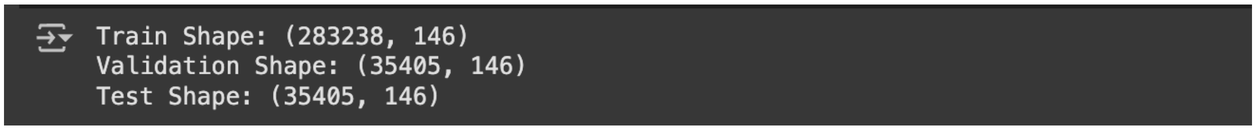


Figure 4: Scatterplot of Skewness vs Kurtosis (BG+1:00) (Recreate by running all cells until the cell heading "Scatterplot: Skewness vs Kurtosis of BG+1:00/P_NUM" in data_analysis.ipynb)

Since the processed dataset originates from a Kaggle competition, the official test set does not include ground truth labels, making it impossible to directly evaluate model accuracy on it. To address this, we split the provided training data into separate training (80%), validation (10%), and test sets (10%), as this portion includes the target values (bg+1:00). This allows us to compute various loss metrics and monitor both training and validation performance during model development.



```
Train Shape: (283238, 146)
Validation Shape: (35405, 146)
Test Shape: (35405, 146)
```

Figure 5: Shapes of Dataset split into Train, Validation and Test Sets (Recreate by running all cells until the cell heading "Data Splitting" in LSTM.ipynb/TCN.ipynb/Transformer.ipynb)

To enable efficient loading and batching of the processed data during model training and evaluation, we create a custom PyTorch dataset class. This class ensures the model receives data in the expected format, 24 time steps with 6 features each, consistent with our selection of the most recent 2 hours of physiological and lifestyle data (glucose, insulin, carbohydrates, heart rate, steps, and calories). We add an `is_train` flag to enable flexibility for both labeled training/validation sets and unlabeled test sets during inference.

To efficiently feed data into the model during training and evaluation, we use PyTorch DataLoaders with a batch size of 256. The training loader is configured with `shuffle=True` to ensure that each epoch sees a different order of samples, which helps prevent overfitting and improves generalization. For the validation and test sets, `shuffle=False` is used to maintain consistency in evaluation. This setup enables streamlined and optimized mini-batch processing during model development.

Additionally, we submit our model predictions to the Kaggle competition platform to obtain an official score. In this competition, a lower score indicates better performance, with the current top score on the leaderboard being 2.3615.

Model

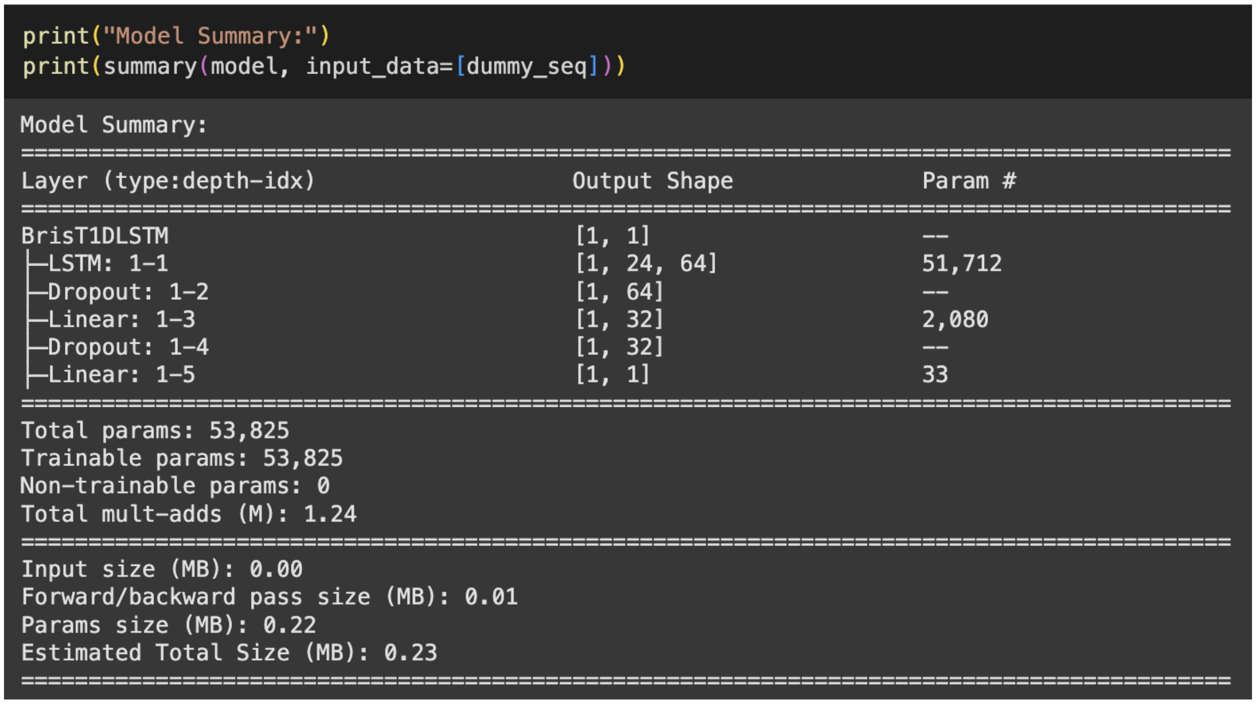
We experimented with three different models to choose the one with highest accuracy and best generalisation on the processed data as described below.

Long Short-Term Memory (LSTM)

The first model we experimented with was an LSTM network (Hochreiter et al., 1997), a popular type of recurrent neural network suited for sequential data like time series.

<u>Architecture</u>

- LSTM Layer: 2 layers with a hidden size of 64, applied with dropout (0.25) to prevent overfitting. It processes the sequence and captures temporal dependencies.
- Fully Connected Layers: The final hidden state of the LSTM is passed through a dropout layer, then a dense layer reducing it to 32 units, followed by ReLU activation and another dropout.
- Output Layer: A final linear layer maps the representation to a single output value, representing the predicted blood glucose level one hour in the future (bg+1:00).



```
print("Model Summary:")
print(summary(model, input_data=[dummy_seq]))
```

```
Model Summary:
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
BrisT1DLSTM                              [1, 1]                    --
├─LSTM: 1-1                              [1, 24, 64]               51,712
├─Dropout: 1-2                           [1, 64]                   --
├─Linear: 1-3                            [1, 32]                   2,080
├─Dropout: 1-4                           [1, 32]                   --
├─Linear: 1-5                            [1, 1]                    33
==========================================================================================
Total params: 53,825
Trainable params: 53,825
Non-trainable params: 0
Total mult-adds (M): 1.24
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.22
Estimated Total Size (MB): 0.23
==========================================================================================
```

Figure 6: LSTM Model Architecture Summary (Recreate by running all cells until the cell heading "Model Summary" in LSTM.ipynb)

Computational Graph

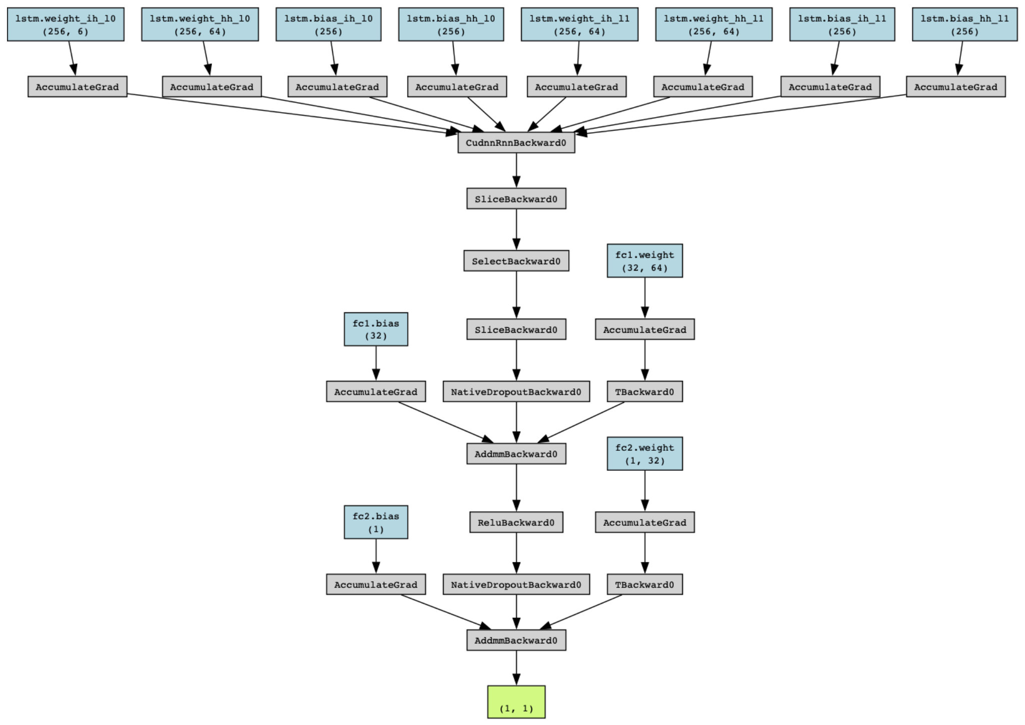
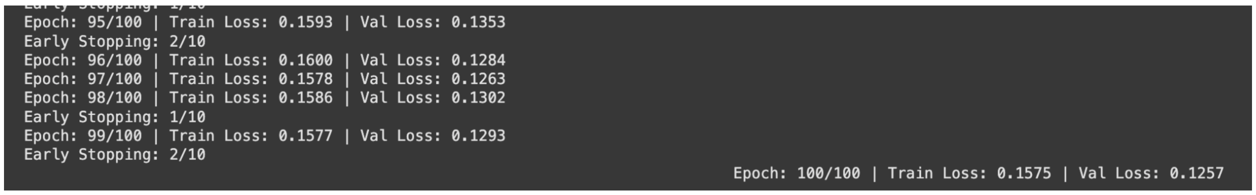


Figure 7: LSTM Computational Graph (Recreate by running all cells until the cellheading “Model Computational Graph” in LSTM.ipynb)

Training Step

The model was trained for up to 100 epochs using the Mean Squared Error (MSE) loss function and the Adam optimizer with a learning rate of 1e-3. To prevent overfitting and ensure training stability, early stopping with a patience of 10 epochs was implemented. Some basic hyperparameter tuning was also performed like adjusting factors such as the number of LSTM layers, hidden size, dropout rate, and batch size to improve model performance.
During each epoch, the model was trained on the training split and evaluated on the validation set. Losses were computed and stored separately for both sets to monitor performance trends. If the validation loss improved, the model was saved as a checkpoint. Otherwise, the early stopping counter was incremented. This process allowed the model to generalize better without overfitting to the training data. By the final epoch, the LSTM achieved a training loss of 0.1575

and a validation loss of 0.1257, showing strong performance on the internal validation split.



```
Epoch: 95/100 | Train Loss: 0.1593 | Val Loss: 0.1353
Early Stopping: 2/10
Epoch: 96/100 | Train Loss: 0.1600 | Val Loss: 0.1284
Epoch: 97/100 | Train Loss: 0.1578 | Val Loss: 0.1263
Epoch: 98/100 | Train Loss: 0.1586 | Val Loss: 0.1302
Early Stopping: 1/10
Epoch: 99/100 | Train Loss: 0.1577 | Val Loss: 0.1293
Early Stopping: 2/10
                                                                   Epoch: 100/100 | Train Loss: 0.1575 | Val Loss: 0.1257
```

Figure 8: LSTM Training (Recreate by running all cells until the cell heading "Training" in LSTM.ipynb)

Below attached is the training curve which illustrates a steady decline in both training and validation loss over the course of 100 epochs, indicating that the model is learning effectively without overfitting. The validation loss consistently remains lower than the training loss, which is suggesting good generalization to unseen data.

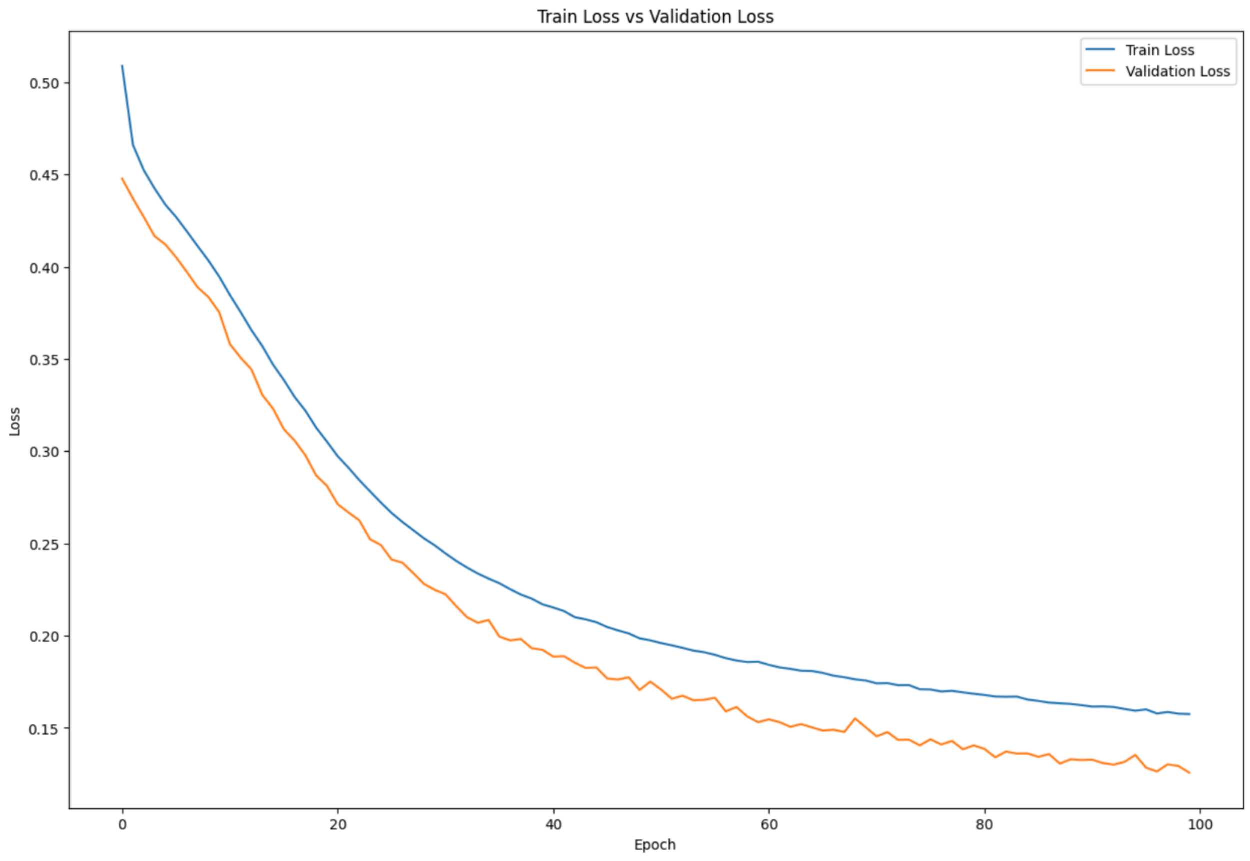


Figure 9: LSTM Loss Curves (Recreate by running all cells until the cell heading "Training" in LSTM.ipynb)

<u>Results</u>

After training, the model was evaluated on the internal test set using a scatter plot comparing actual vs predicted glucose values. The predictions were first inverse-transformed to their original scale using the saved scaler before visualization. As shown in the plot, the points largely align with the diagonal red reference line, indicating a strong correlation between predicted and actual values. The spread remains tight along this line, especially in the mid-glucose range, suggesting the model performs reliably in typical conditions. Slight deviations at higher glucose levels are expected due to data sparsity or noise, but overall, the model demonstrates high predictive accuracy and generalization on unseen data.

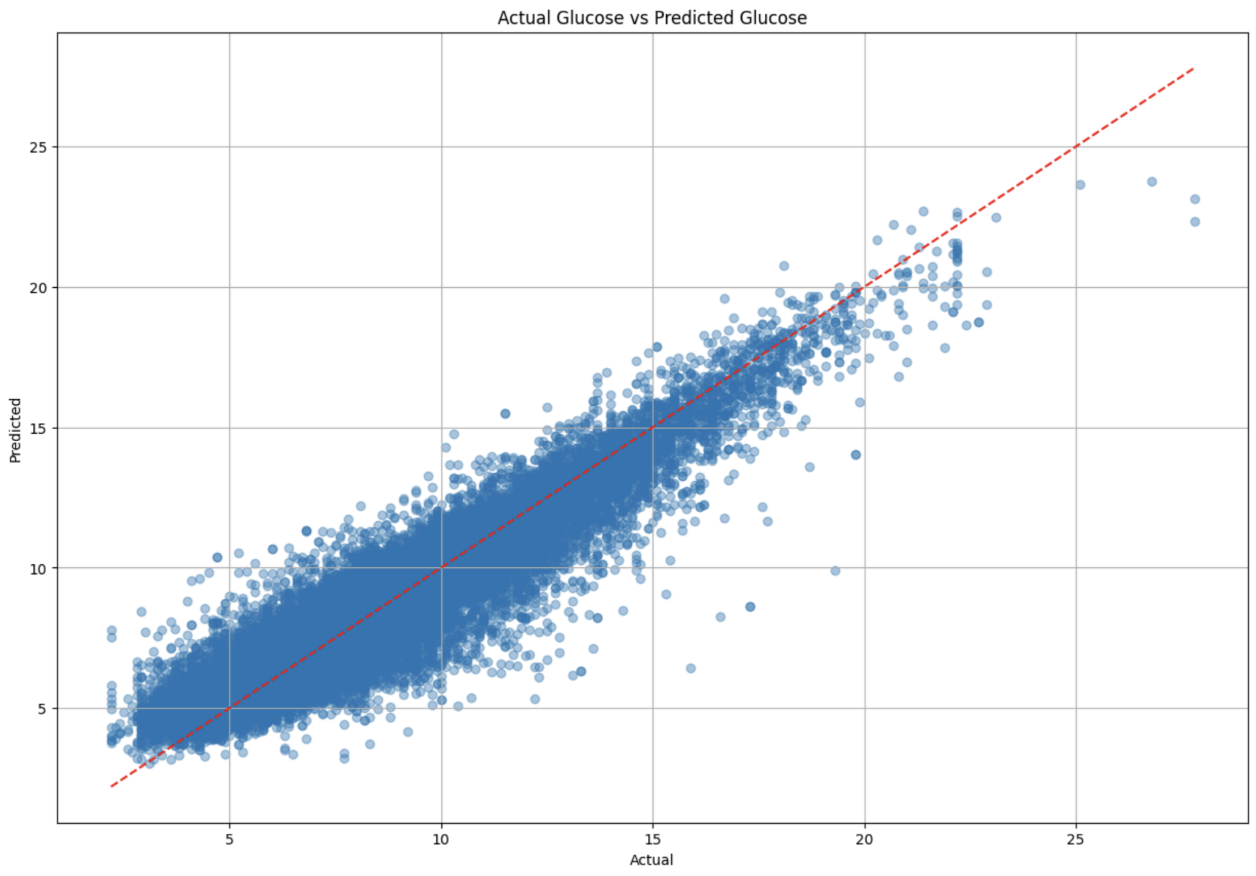
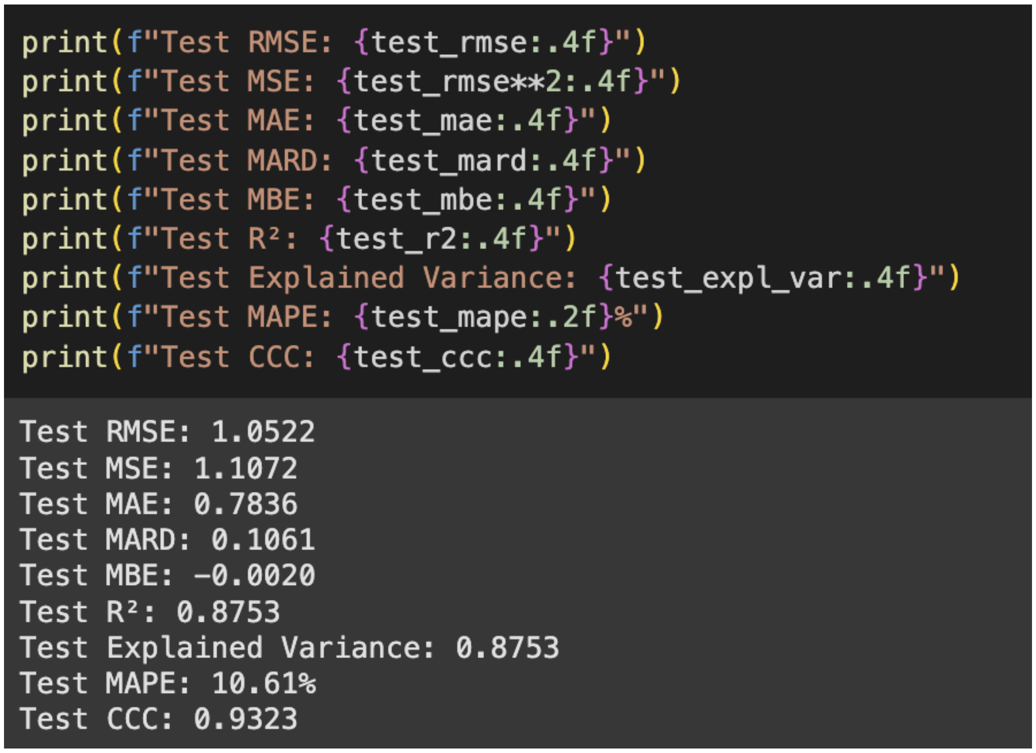


Figure 10: LSTM Results (Recreate by running all cells until the cell heading "Evaluation" in LSTM.ipynb)

The performance was further assessed using some regression metrics as well on the internal test set. The Root Mean Squared Error (RMSE) was 1.0522, and the Mean Absolute Error (MAE) was 0.7836, both indicating low average prediction error. The Mean Absolute Relative Deviation (MARD) of 0.1061 and Mean Absolute Percentage Error (MAPE) of 10.61% reflect a good fit relative to the scale of the data. The model also achieved a high R^2 score of 0.8753 and an Explained Variance of 0.8753, suggesting that it captures a substantial proportion of the variability in blood glucose levels. Notably, the Concordance Correlation Coefficient (CCC) was 0.9323, indicating excellent agreement between actual and predicted values. Together, these metrics confirm that the LSTM model demonstrates strong predictive accuracy and generalization.

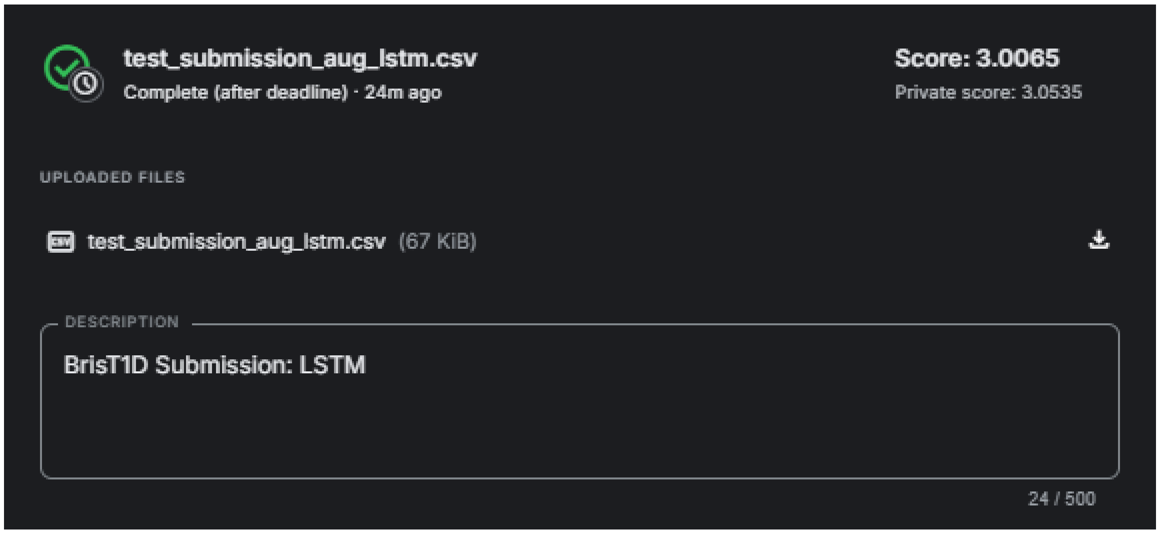


```python
print(f"Test RMSE: {test_rmse:.4f}")
print(f"Test MSE: {test_rmse**2:.4f}")
print(f"Test MAE: {test_mae:.4f}")
print(f"Test MARD: {test_mard:.4f}")
print(f"Test MBE: {test_mbe:.4f}")
print(f"Test R²: {test_r2:.4f}")
print(f"Test Explained Variance: {test_expl_var:.4f}")
print(f"Test MAPE: {test_mape:.2f}%")
print(f"Test CCC: {test_ccc:.4f}")
```

```
Test RMSE: 1.0522
Test MSE: 1.1072
Test MAE: 0.7836
Test MARD: 0.1061
Test MBE: -0.0020
Test R²: 0.8753
Test Explained Variance: 0.8753
Test MAPE: 10.61%
Test CCC: 0.9323
```

Figure 11: LSTM Performance (Recreate by running all cells until the cell heading "Evaluation" in LSTM.ipynb)

Predictions were also submitted to the Kaggle competition where the LSTM model achieved a Kaggle score of 3.0065 as attached below.



test_submission_aug_lstm.csv
Complete (after deadline) · 24m ago

Score: 3.0065
Private score: 3.0535

UPLOADED FILES

test_submission_aug_lstm.csv (67 KiB)

DESCRIPTION

BrisT1D Submission: LSTM

24 / 500

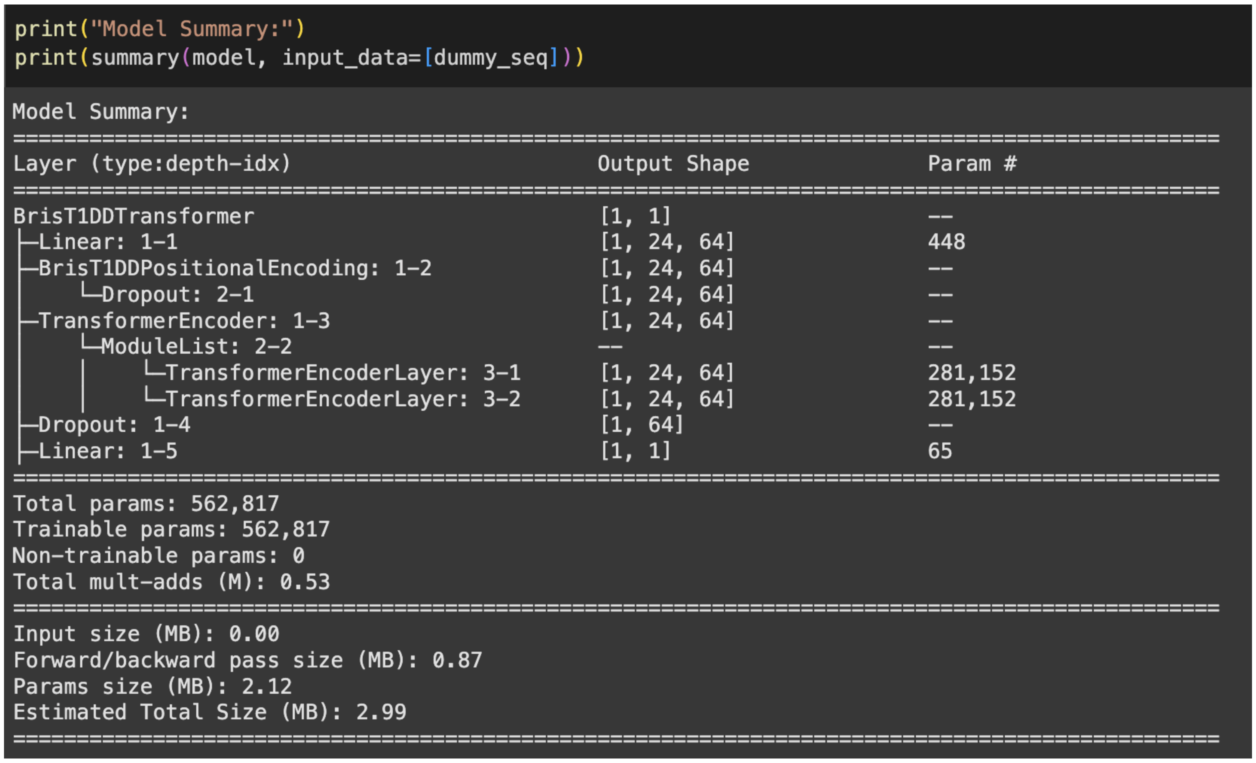
Figure 12: Kaggle Result for LSTM (Recreate by running all cells until the cell heading "Submission" in LSTM.ipynb)

Transformer

The second model we implemented was a Transformer-based network (Vaswani et al., 2017), adapted from the architecture originally developed for natural language processing. We tested it to model long-range dependencies in our time-series forecasting task.

Architecture

- Input Projection Layer: The input sequence, consisting of 24 time steps and 6 physiological features, is first passed through a linear layer that projects each feature vector into a 64-dimensional space. This transformation allows the model to operate in a higher-dimensional embedding space more suitable for capturing complex interactions.
- Positional Encoding: Since Transformers do not have an inherent sense of order, positional encoding is added to the input embeddings to retain information about the position of each time step. This is implemented using sine and cosine functions and helps the model distinguish between temporal locations.
- Transformer Encoder: The model uses 2 Transformer encoder layers, each equipped with multi-head self-attention (8 heads) to learn temporal dependencies across all time steps simultaneously. Each encoder layer includes a feed-forward sub-layer, layer normalization, and dropout (0.25), enabling the model to generalize well while maintaining stable gradients.
- Sequence Aggregation: The outputs of the encoder layers are averaged across the time dimension using global average pooling. This operation condenses the learned representation of the entire sequence into a single vector, summarizing the temporal patterns across the 24 time steps.
- Output Layer: After a final dropout layer, the aggregated sequence representation is passed through a fully connected layer that maps the 64-dimensional vector to a single output value. This output corresponds to the predicted blood glucose level one hour into the future (bg+1:00).



```
print("Model Summary:")
print(summary(model, input_data=[dummy_seq]))
```

```
Model Summary:
==========================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
BrisT1DDTransformer                                [1, 1]                    --
├─Linear: 1-1                                      [1, 24, 64]               448
├─BrisT1DDPositionalEncoding: 1-2                  [1, 24, 64]               --
│    └─Dropout: 2-1                                [1, 24, 64]               --
├─TransformerEncoder: 1-3                          [1, 24, 64]               --
│    └─ModuleList: 2-2                             --                        --
│    │    └─TransformerEncoderLayer: 3-1           [1, 24, 64]               281,152
│    │    └─TransformerEncoderLayer: 3-2           [1, 24, 64]               281,152
├─Dropout: 1-4                                     [1, 64]                   --
├─Linear: 1-5                                      [1, 1]                    65
==========================================================================================
Total params: 562,817
Trainable params: 562,817
Non-trainable params: 0
Total mult-adds (M): 0.53
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.87
Params size (MB): 2.12
Estimated Total Size (MB): 2.99
==========================================================================================
```

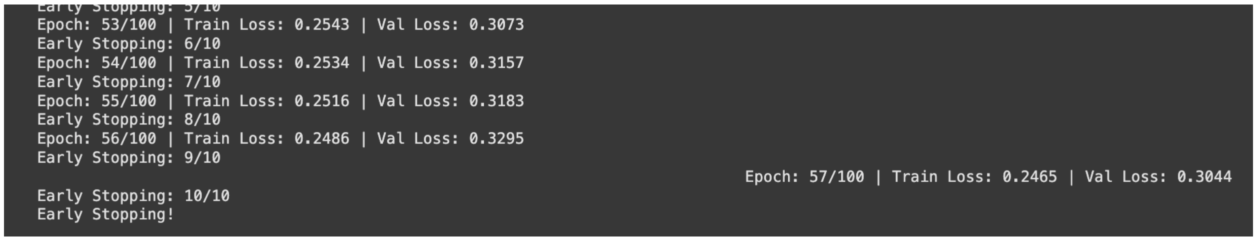
Figure 13: Transformer Model Architecture Summary (Recreate by running all cells until the cell heading "Model Summary" in Transformer.ipynb)

Computational Graph

The computational graph for the transformer is too lengthy to be attached in this report. PFA here.
[*Figure 14: Transformer Computational Graph (Recreate by running all cells until the cellheading "Model Computational Graph" in Transformer.ipynb)*]

Training Step

The Transformer model was trained for up to 100 epochs using the Mean Squared Error (MSE) loss function and optimized with the Adam optimizer at a learning rate of 1e-3. To prevent overfitting and promote generalization, early stopping with a patience of 10 epochs was implemented, i.e. we halt training if validation loss failed to improve for 10 consecutive epochs. During each epoch, the model was trained on the training set and evaluated on the validation set, with loss values computed and tracked separately to monitor learning progress. The model's weights were saved whenever a new best validation loss was observed. This approach ensured training stability while avoiding overfitting. By epoch 57, the model reached a training loss of 0.2465 and a validation loss of 0.3044. However, due to the early stopping mechanism being triggered, training was stopped further, preserving the best-performing model based on validation loss.



```
Early Stopping: 5/10
Epoch: 53/100 | Train Loss: 0.2543 | Val Loss: 0.3073
Early Stopping: 6/10
Epoch: 54/100 | Train Loss: 0.2534 | Val Loss: 0.3157
Early Stopping: 7/10
Epoch: 55/100 | Train Loss: 0.2516 | Val Loss: 0.3183
Early Stopping: 8/10
Epoch: 56/100 | Train Loss: 0.2486 | Val Loss: 0.3295
Early Stopping: 9/10
                                                       Epoch: 57/100 | Train Loss: 0.2465 | Val Loss: 0.3044
Early Stopping: 10/10
Early Stopping!
```

Figure 15: Transformer Training (Recreate by running all cells until the cell heading "Training" in Transformer.ipynb)

The training curve below illustrates a steady decline in both training and validation loss over the course of training, indicating that the model is learning effectively. Although some fluctuations are observed in the validation loss, which is expected due to the stochastic nature of the optimization process.

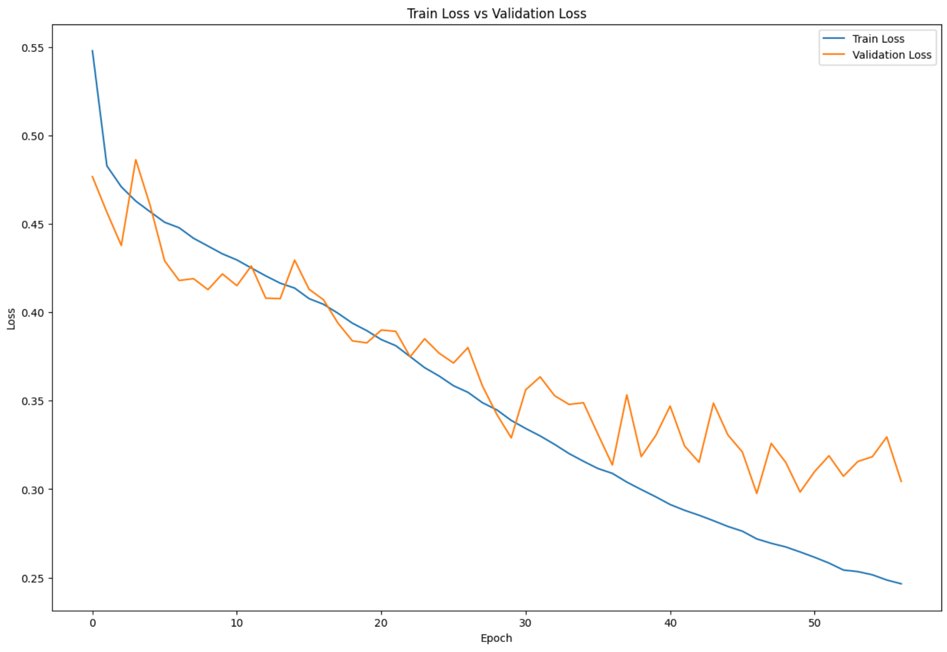


Figure 16: Transformer Loss Curves (Recreate by running all cells until the cell heading "Training" in Transformer.ipynb)

Results

After training, the Transformer model was evaluated on the internal test set using a scatter plot comparing the actual and predicted glucose values. Similar to the LSTM model above, the predictions were inverse-transformed back to their original glucose scale before visualization. The plot shows that the predicted values closely follow the red diagonal reference line, indicating a strong positive correlation between actual and predicted glucose levels. The concentration of points along this line especially in the mid-range values demonstrates that the model performs reliably under normal glucose conditions. While there is some spread at higher glucose levels, this is likely due to fewer samples or greater variability in those regions. Overall, the Transformer model shows robust predictive accuracy and generalization to unseen data.

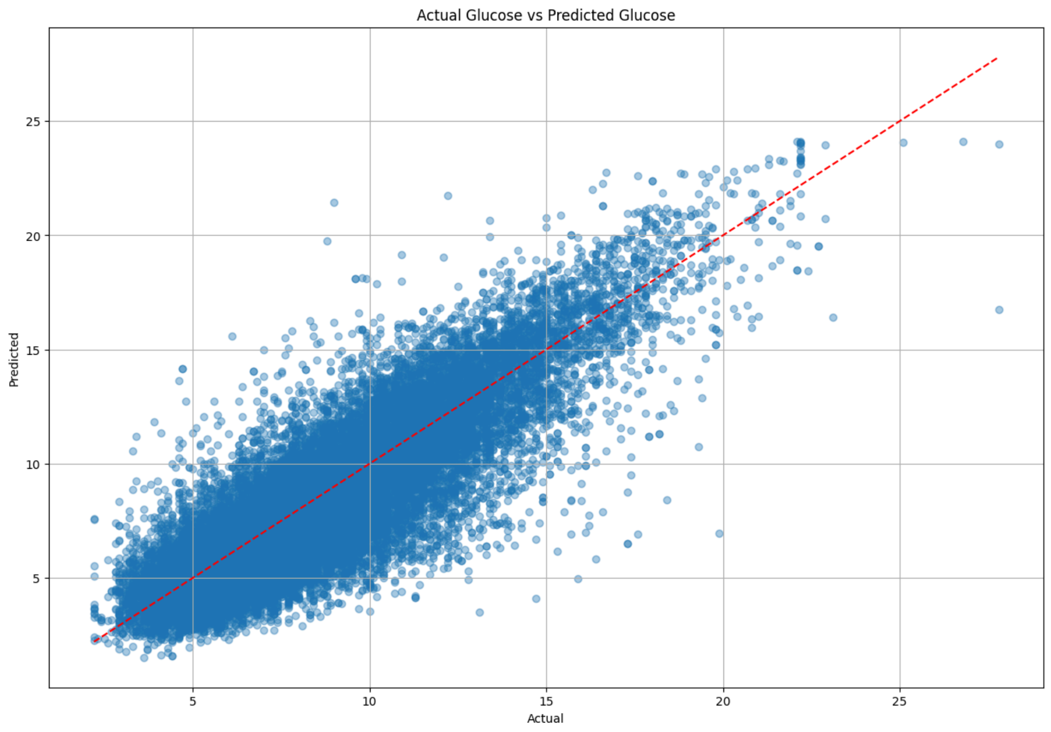
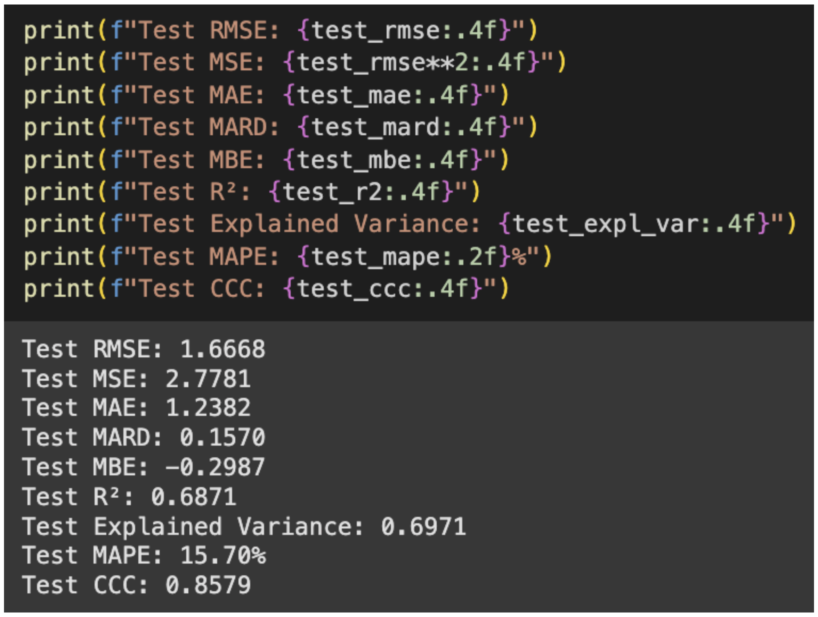


Figure 17: Transformer Results (Recreate by running all cells until the cell heading "Evaluation" in Transformer.ipynb)

The performance of the Transformer model was further assessed using a range of regression metrics on the internal test set. The Root Mean Squared Error (RMSE) was 1.6668, and the Mean Absolute Error (MAE) was 1.2382, indicating a moderate level of prediction error. The Mean Absolute Relative Deviation (MARD) of 0.1570 and Mean Absolute Percentage Error (MAPE) of 15.70% suggest reasonable alignment with the scale of glucose values, though slightly less precise than the LSTM baseline. The model achieved an R^2 score of 0.6871 and an Explained Variance of 0.6971, indicating that it captures a substantial portion of the variability in glucose levels. Importantly, the Concordance Correlation Coefficient (CCC) was 0.8579, reflecting strong agreement between actual and predicted values. While not outperforming the LSTM in all metrics, the Transformer still demonstrates solid predictive accuracy and generalization across unseen test data.

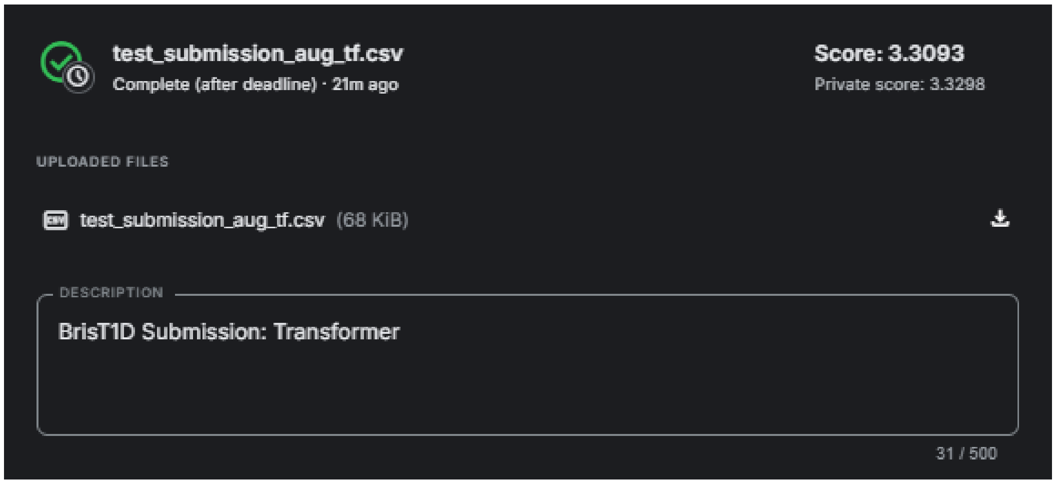


```
print(f"Test RMSE: {test_rmse:.4f}")
print(f"Test MSE: {test_rmse**2:.4f}")
print(f"Test MAE: {test_mae:.4f}")
print(f"Test MARD: {test_mard:.4f}")
print(f"Test MBE: {test_mbe:.4f}")
print(f"Test R²: {test_r2:.4f}")
print(f"Test Explained Variance: {test_expl_var:.4f}")
print(f"Test MAPE: {test_mape:.2f}%")
print(f"Test CCC: {test_ccc:.4f}")
```

```
Test RMSE: 1.6668
Test MSE: 2.7781
Test MAE: 1.2382
Test MARD: 0.1570
Test MBE: -0.2987
Test R²: 0.6871
Test Explained Variance: 0.6971
Test MAPE: 15.70%
Test CCC: 0.8579
```

Figure 18: Transformer Performance (Recreate by running all cells until the cell heading "Evaluation" in Transformer.ipynb)

Predictions were also submitted to the Kaggle competition where the Transformer model achieved a Kaggle score of 3.3093 as attached below.



test_submission_aug_tf.csv
Complete (after deadline) · 21m ago

Score: 3.3093
Private score: 3.3298

UPLOADED FILES

test_submission_aug_tf.csv (68 KiB)

DESCRIPTION

BrisT1D Submission: Transformer

31 / 500

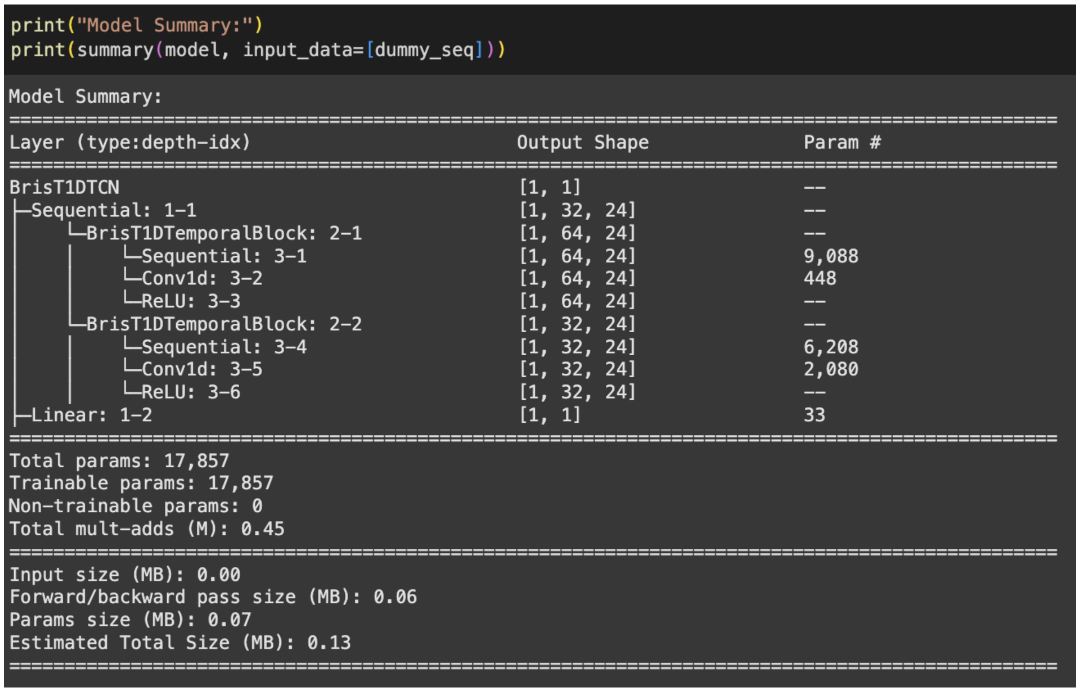
Figure 19: Kaggle Result for Transformer (Recreate by running all cells until the cell heading "Submission" in Transformer.ipynb)

Temporal Convolutional Networks (TCN)

The final model we experimented with was a TCN model (Lea et al., 2016), a popular type of neural network architecture well-suited for sequential data like time series.

Architecture

- Input Layer: The input sequence, consisting of 24 time steps and 6 physiological features, is passed directly into a stack of Temporal Convolutional Network (TCN) blocks. Unlike RNNs, TCNs use 1D dilated causal convolutions to capture sequential dependencies, making them suitable for time-series prediction tasks.
- Temporal Blocks: The model comprises two BrisT1DTemporalBlock modules. Each block contains two convolutional layers with a kernel size of 2, causal padding, and dropout (0.25) applied after each convolution. The first block uses 64 filters and no dilation, while the second block reduces the dimensionality to 32 filters and applies a dilation factor of 2 to increase the receptive field, allowing the model to capture longer-term dependencies.
- Residual Connections and Downsampling: Each temporal block includes a residual connection that projects the input to match the output shape when necessary using a 1x1 convolution. This helps preserve gradient flow and stabilizes training. ReLU activations follow each block to introduce non-linearity.
- Sequence Aggregation: Unlike Transformers that use attention mechanisms, the TCN implicitly aggregates information across time through stacked convolutions and increasing dilation rates. The output retains the temporal structure with reduced dimensionality ([batch_size, 32, 24]).
- Output Layer: The temporal output is passed through a final linear layer that maps the 32-dimensional output at each time step to a single scalar prediction. The output is reduced to shape [batch_size, 1], corresponding to the predicted blood glucose level one hour into the future (bg+1:00).



```
print("Model Summary:")
print(summary(model, input_data=[dummy_seq]))
```

```
Model Summary:
==============================================================================================
Layer (type:depth-idx)                        Output Shape              Param #
==============================================================================================
BrisT1DTCN                                    [1, 1]                    --
├─Sequential: 1-1                             [1, 32, 24]               --
│    └─BrisT1DTemporalBlock: 2-1              [1, 64, 24]               --
│    │    └─Sequential: 3-1                   [1, 64, 24]               9,088
│    │    └─Conv1d: 3-2                       [1, 64, 24]               448
│    │    └─ReLU: 3-3                         [1, 64, 24]               --
│    └─BrisT1DTemporalBlock: 2-2              [1, 32, 24]               --
│    │    └─Sequential: 3-4                   [1, 32, 24]               6,208
│    │    └─Conv1d: 3-5                       [1, 32, 24]               2,080
│    │    └─ReLU: 3-6                         [1, 32, 24]               --
├─Linear: 1-2                                 [1, 1]                    33
==============================================================================================
Total params: 17,857
Trainable params: 17,857
Non-trainable params: 0
Total mult-adds (M): 0.45
==============================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 0.07
Estimated Total Size (MB): 0.13
==============================================================================================
```

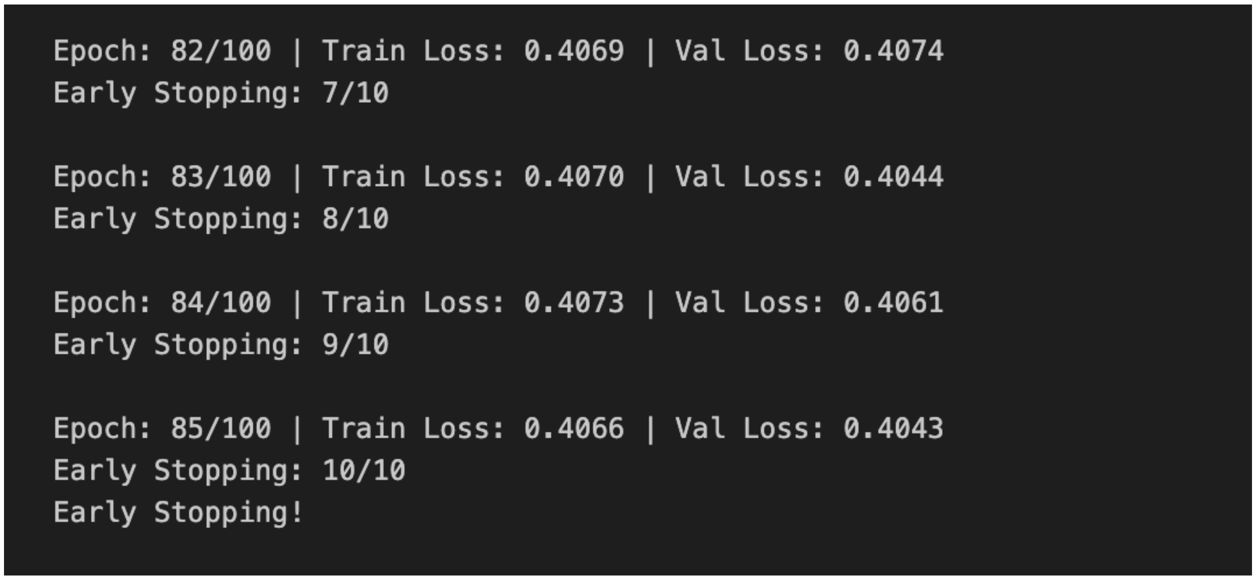
Figure 20: TCN Model Architecture Summary (Recreate by running all cells until the cell heading "Model Summary" in TCN.ipynb)

Computational Graph

The computational graph for the TCN is too lengthy to be attached in this report. PFA here.
[*Figure 21: TCN Computational Graph (Recreate by running all cells until the cell heading "Model Computational Graph" in TCN.ipynb)]*

Training Step

The TCN model was trained for up to 100 epochs using the Mean Squared Error (MSE) loss function and the Adam optimizer with a learning rate of 1e-3. To prevent overfitting and ensure robust learning, early stopping with a patience of 10 epochs was applied, meaning training was halted if the validation loss did not improve for 10 consecutive epochs. During each epoch, the model was trained on the training dataset and then evaluated on the validation set. Both training and validation losses were recorded to monitor learning progression. Some basic hyperparameter tuning was also performed such as adjusting the number of temporal blocks, kernel size, dropout rate, and hidden dimensions to improve performance. Whenever a new lowest validation loss was observed, the model's parameters were saved. This strategy helped maintain training stability and preserved the most generalizable model. By epoch 85, the TCN reached a training loss of 0.4066 and a validation loss of 0.4043. However, since validation performance had plateaued, early stopping was triggered, and training was stopped to retain the best-performing model checkpoint.



```
Epoch: 82/100 | Train Loss: 0.4069 | Val Loss: 0.4074
Early Stopping: 7/10

Epoch: 83/100 | Train Loss: 0.4070 | Val Loss: 0.4044
Early Stopping: 8/10

Epoch: 84/100 | Train Loss: 0.4073 | Val Loss: 0.4061
Early Stopping: 9/10

Epoch: 85/100 | Train Loss: 0.4066 | Val Loss: 0.4043
Early Stopping: 10/10
Early Stopping!
```

Figure 22: TCN Training (Recreate by running all cells until the cell heading "Training" in TCN.ipynb)

The training curve below illustrates a consistent and smooth decline in both training and validation loss throughout the course of training, suggesting that the TCN model is learning effectively and steadily improving its performance. While minor fluctuations are present in the validation loss, these are expected due to the stochastic nature of optimization and data

variation. Notably, the gap between training and validation loss remains minimal across epochs, indicating excellent generalization and stability.
Compared to the Transformer and LSTM models, this curve demonstrates a more stable convergence pattern with reduced overfitting, reinforcing the effectiveness of the TCN architecture for modeling temporal dependencies in blood glucose prediction.

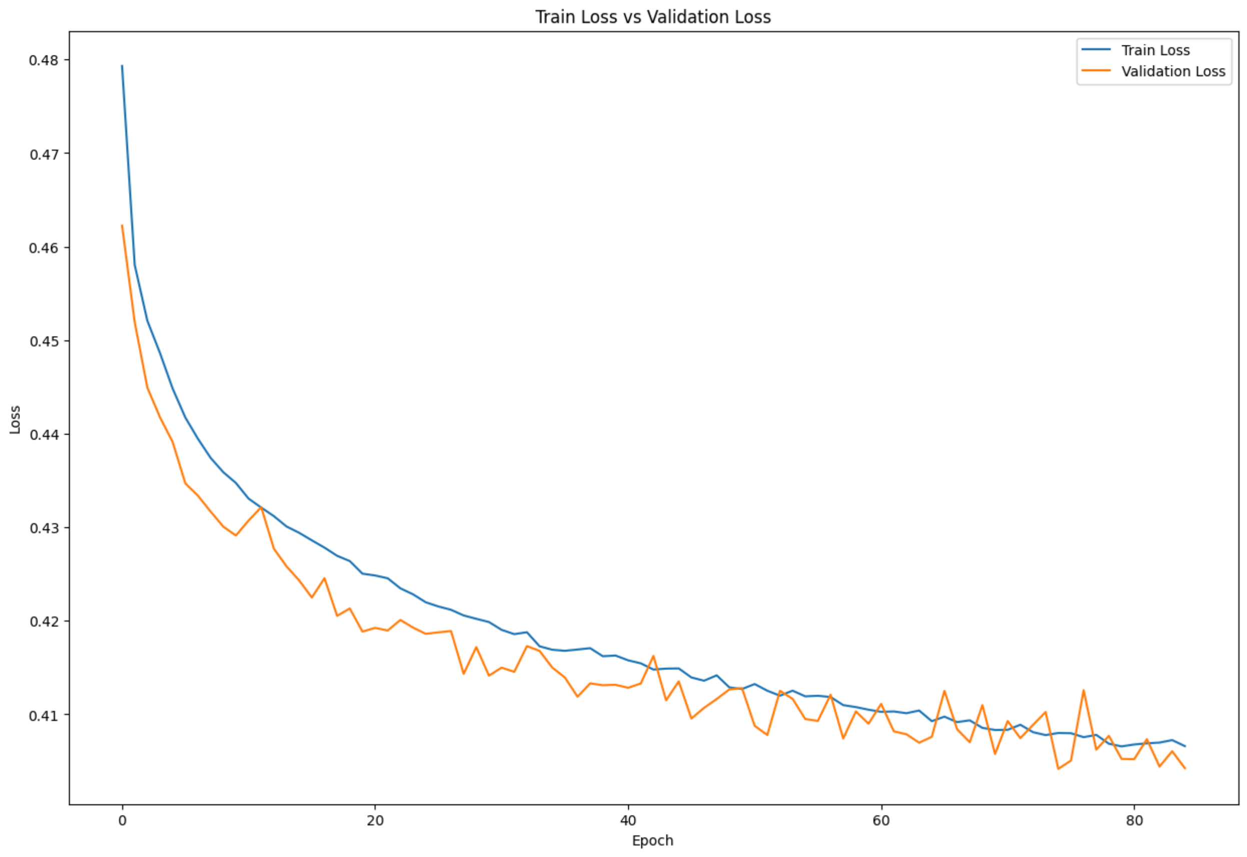


Figure 23: TCN Loss Curves (Recreate by running all cells until the cell heading "Training" in TCN.ipynb)

<u>Results</u>

After training, the TCN model was evaluated on the internal test set using a scatter plot comparing actual and predicted glucose values. As with previous models, the predictions were inverse-transformed to the original glucose scale. The scatter plot reveals that the TCN's predictions generally align well with the diagonal reference line, indicating a strong correlation with actual values. The points are densely clustered along this line in the low to mid-range glucose levels, reflecting accurate performance in typical cases. Although some underestimation is observed at higher glucose levels, the model still demonstrates solid predictive ability and generalization on unseen data.

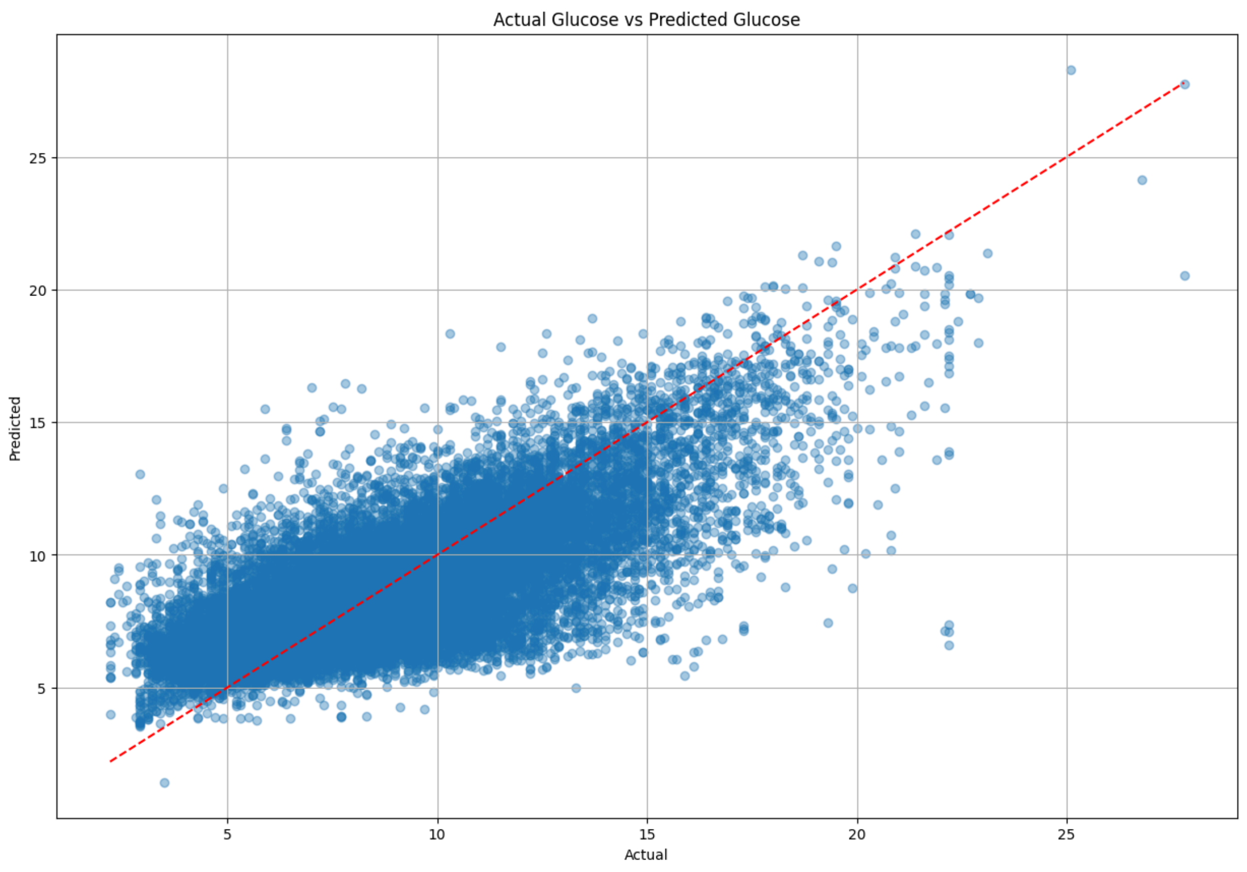
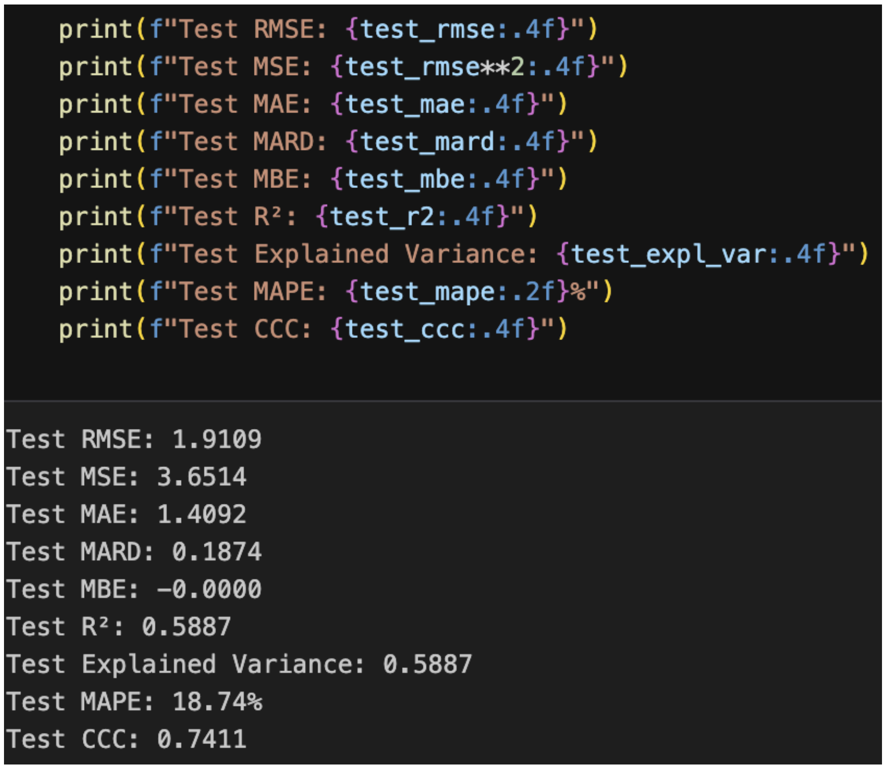


Figure 24: TCN Results (Recreate by running all cells until the cell heading “Evaluation” in TCN.ipynb)

The performance of the TCN model was evaluated using several regression metrics on the internal test set. The Root Mean Squared Error (RMSE) was 1.9109, and the Mean Absolute Error (MAE) was 1.4092. The Mean Absolute Relative Deviation (MARD) of 0.1874 and Mean Absolute Percentage Error (MAPE) of 18.74% indicate a relatively looser fit to the scale of the target values. The model achieved an R^2 score of 0.5877 and an Explained Variance of 0.5877, suggesting that it captures a moderate portion of the variability in glucose readings. The Concordance Correlation Coefficient (CCC) stood at 0.7411, showing fair agreement between actual and predicted values.

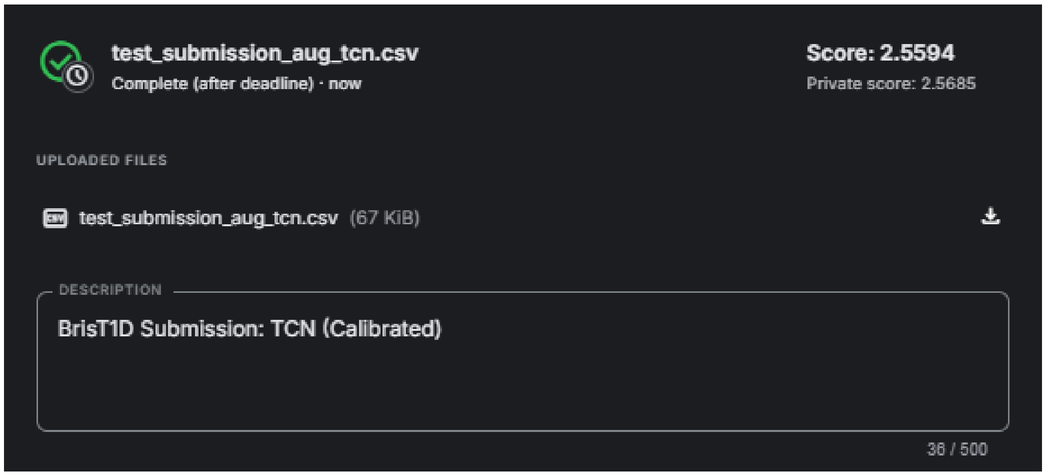


```
    print(f"Test RMSE: {test_rmse:.4f}")
    print(f"Test MSE: {test_rmse**2:.4f}")
    print(f"Test MAE: {test_mae:.4f}")
    print(f"Test MARD: {test_mard:.4f}")
    print(f"Test MBE: {test_mbe:.4f}")
    print(f"Test R²: {test_r2:.4f}")
    print(f"Test Explained Variance: {test_expl_var:.4f}")
    print(f"Test MAPE: {test_mape:.2f}%")
    print(f"Test CCC: {test_ccc:.4f}")
```

```
Test RMSE: 1.9109
Test MSE: 3.6514
Test MAE: 1.4092
Test MARD: 0.1874
Test MBE: -0.0000
Test R²: 0.5887
Test Explained Variance: 0.5887
Test MAPE: 18.74%
Test CCC: 0.7411
```

Figure 25: TCN Performance (Recreate by running all cells until the cell heading "Evaluation" in TCN.ipynb)

Predictions were also submitted to the Kaggle competition where the TCN model achieved a Kaggle score of 2.5594 as attached below, the best out of all 3 models we experimented with.



test_submission_aug_tcn.csv
Complete (after deadline) · now

Score: 2.5594
Private score: 2.5685

UPLOADED FILES

test_submission_aug_tcn.csv (67 KiB)

DESCRIPTION

BrisT1D Submission: TCN (Calibrated)

36 / 500

Figure 26: Kaggle Result for TCN (Recreate by running all cells until the cell heading "Submission" in TCN.ipynb)

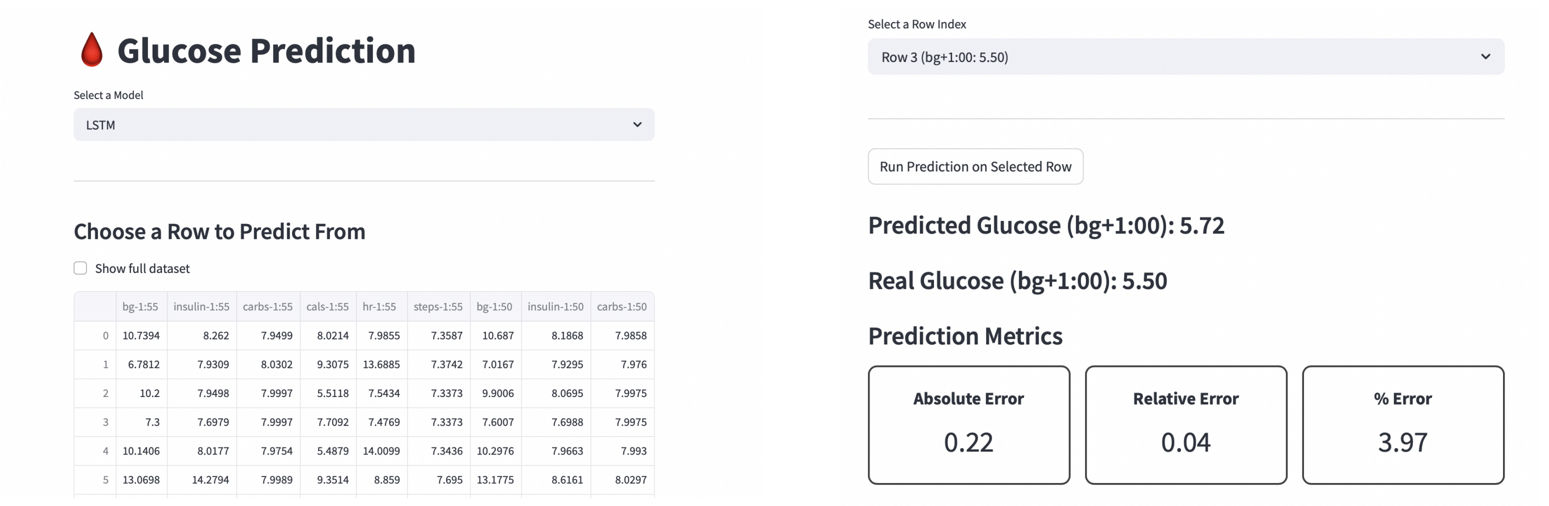
Results & Visualisation of Model Performance

To evaluate the performance across different models, we tested three neural architectures LSTM (Long Short-Term Memory), Transformer, and TCN (Temporal Convolutional Network) on our processed dataset. Each model was trained using the most recent 24 time steps (roughly two hours of data) to predict blood glucose levels one hour ahead. Their performance was assessed using both internal test metrics and external validation on the Kaggle leaderboard, where a lower score signifies better predictive accuracy. While all models showed reasonable performance and generalization, the TCN model emerged as the most effective, achieving a final Kaggle score of 2.5594 remarkably close to the competition's top score of 2.3615, demonstrating its strength in capturing temporal dependencies in time-series glucose data. Additionally, the results and loss curves have been discussed earlier in each model sub-section above.

Streamlit Interface

We developed an interactive web interface using Streamlit to visualize and evaluate our glucose prediction models. The interface allows users to select from three different models: LSTM, Transformer, and TCN, and run predictions on any row from the preprocessed dataset.

In the code, the selected row's features are reshaped into a 3D tensor (1, 24, 6) to match the input format expected by the models. The model then predicts the scaled value of the future glucose level (bg+1:00), which is converted back to its original scale using a saved target scaler. The interface compares this prediction with the ground truth and computes the error metrics: absolute error, relative error, and percentage error on the original scale, which are then displayed. This setup enables transparent performance inspection and easy comparison between models. The Streamlit dashboard is hosted here.



Glucose Prediction

Select a Model: LSTM

Choose a Row to Predict From

Show full dataset

	bg-1:55	insulin-1:55	carbs-1:55	cals-1:55	hr-1:55	steps-1:55	bg-1:50	insulin-1:50	carbs-1:50
0	10.7394	8.262	7.9499	8.0214	7.9855	7.3587	10.687	8.1868	7.9858
1	6.7812	7.9309	8.0302	9.3075	13.6885	7.3742	7.0167	7.9295	7.976
2	10.2	7.9498	7.9997	5.5118	7.5434	7.3373	9.9006	8.0695	7.9975
3	7.3	7.6979	7.9997	7.7092	7.4769	7.3373	7.6007	7.6988	7.9975
4	10.1406	8.0177	7.9754	5.4879	14.0099	7.3436	10.2976	7.9663	7.993
5	13.0698	14.2794	7.9989	9.3514	8.859	7.695	13.1775	8.6161	8.0297

Select a Row Index: Row 3 (bg+1:00: 5.50)

Run Prediction on Selected Row

Predicted Glucose (bg+1:00): 5.72

Real Glucose (bg+1:00): 5.50

Prediction Metrics

Absolute Error	Relative Error	% Error
0.22	0.04	3.97

Figure 27: Screenshots of the Streamlit Interface (Recreate by visiting the link where the app is hosted, as provided above)

Comparison with Other State-of-the-Art Models

While our deep learning-based models (LSTM, Transformer, and TCN) demonstrated strong performance in terms of generalization and learning glucose dynamics from temporal patterns, the top-performing solution on the Kaggle leaderboard used a more traditional machine learning approach, a single LightGBM model. This solution achieved a final leaderboard score of 2.3615, outperforming our best model (TCN) which scored 2.5594. Interestingly, this result highlights that with robust feature engineering and cross-validation, even simpler models can achieve state-of-the-art performance in time-series forecasting tasks. This comparison underlines the importance of not just model complexity but also data preprocessing and feature engineering, which remain crucial in achieving competitive results in real-world applications.

In addition to the models evaluated in this report, several other state-of-the-art approaches have been applied to blood glucose prediction in recent literature. These include deep learning architectures such as DeepAR (Salinas et al., 2017), N-BEATS (Oreshkin et al., 2019), and the Temporal Fusion Transformer (Lim et al., 2019), which offer strong forecasting performance, especially for multivariate or long-range predictions. Transformer variants like Informer (Zhou et al., 2020) and Autoformer (Wu et al., 2021) are also gaining traction for their ability to handle extended time sequences efficiently. While these models may require more data and computational resources, they offer promising directions for future improvements in glucose forecasting systems.

Conclusion & Future Improvements

In this project, we implemented and compared LSTM, Transformer, and TCN models for short-term blood glucose prediction using the BrisT1D dataset. While the TCN model achieved the best performance among the deep learning models, traditional models like LightGBM still outperformed it on the leaderboard, emphasizing the importance of feature engineering and data handling. Nonetheless, the deep learning models demonstrated strong generalization and provided a solid foundation for future enhancements.

Some improvements we could consider -

- Hybrid modeling: Combine deep learning with engineered features from models like LightGBM for better performance.
- Ensemble methods: Experiment with ensembling multiple models to reduce variance and boost robustness.
- Advanced architectures: Try state-of-the-art models like Temporal Fusion Transformer, Autoformer, or N-BEATS.
- Better preprocessing and feature engineering: Improve missing data handling, apply noise filtering techniques, and engineer additional contextual features (e.g., sleep, stress, glucose rate of change, time-of-day effects) to enrich the input representation and boost model performance.
- Longer context: Experiment with input windows beyond 2 hours to test effects on long-range prediction.

- Patient-specific tuning: Fine-tune models on individual users for more personalized and adaptive predictions.
- Real-time deployment: Explore model performance in streaming or real-time environments for practical use in CGM devices.

Code Setup & Instructions

Follow the steps below to set up and run the code to reproduce the model, and train it:

1. **Clone the repository**
   ```
   git clone
   https://github.com/verneylmavt/st-blood-sugar-spike-prediction.gi
   t
   ```

2. **Navigate to the project directory**

   ```
   cd st-blood-sugar-spike-prediction
   ```

3. **Install the required dependencies**

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

   Note: If your machine supports CUDA for GPU acceleration, refer to the official PyTorch installation guide: https://pytorch.org/get-started/locally/.
   This is necessary because `requirements.txt` does not accommodate GPU-specific PyTorch installations.

4. **Data Setup**
   Open `data_analysis.ipynb`. The dataset will be automatically downloaded using the following command executed within the notebook:

   ```
   subprocess.run(["kaggle", "competitions", "download", "-c",
   "brist1d", "-p", "./.data"])
   ```

   - However, this requires the kaggle library, which is already included in `requirements.txt` and Kaggle API key.
   - If you don't have the API key set up, manually download the data from: https://www.kaggle.com/competitions/brist1d/data.
   - For this project, only `train.csv` and `test.csv` are required. Once downloaded, place both files in the `.data/` directory at the root of the project: st-blood-sugar-spike-prediction/.data/

    - You do **not** need to extract the ZIP files manually, this is handled automatically in the notebook.

5. **Running the Notebooks**
   After setup, you can run the following notebooks in sequence as needed:

    - `data_analysis.ipynb`
    - `LSTM.ipynb / TCN.ipynb / Transformer.ipynb`

Model Reproducibility

All the models we experimented with, LSTM, Transformer, and TCN have been saved as pre-trained checkpoints to ensure reproducibility. The corresponding files are *checkpoint-aug-lstm.pt* for the LSTM, *checkpoint-aug-tf.pt* for the Transformer, and *checkpoint-aug-tcn.pt* for the TCN. These models can be loaded directly from these files without retraining, which is the approach we followed while integrating them into our Streamlit interface for evaluation and visualization.

Group Members & Contributions

Vanya Jalan (1006190)	Worked on code, report and slides
Elvern Neylmav Tanny (1006203)	Worked on code, report and slides

References

1) Dave, D., DeSalvo, D. J., Haridas, B., McKay, S., Shenoy, A., Koh, C. J., Lawley, M., & Erraguntla, M. (2020). Feature-Based machine learning model for Real-Time Hypoglycemia Prediction. *Journal of Diabetes Science and Technology*, *15*(4), 842–855. https://doi.org/10.1177/1932296820922622

2) Hochreiter, S., PhD, Schmidhuber, J., Ronald Williams, Fakultät für Informatik, Technische Universität München, & IDSIA. (1997). Long Short-Term memory. In Massachusetts Institute of Technology, *Neural Computation* (Vols. 9–9, pp. 1735–1780). https://deeplearning.cs.cmu.edu/F23/document/readings/LSTM.pdf

3) Lea, C., Flynn, M. D., Vidal, R., Reiter, A., & Hager, G. D. (2016). *Temporal convolutional networks for action segmentation and detection*. arXiv. https://doi.org/10.48550/arXiv.1611.05267

4) Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2019). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1912.09363

5) Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1905.10437

6) Salinas, D., Flunkert, V., & Gasthaus, J. (2017). DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1704.04110

7) Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. arXiv. https://doi.org/10.48550/arXiv.1706.03762

8) Wu, H., Xu, J., Wang, J., & Long, M. (2021). Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2106.13008

9) Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2020). Informer: Beyond efficient transformer for long sequence Time-Series forecasting. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2012.07436