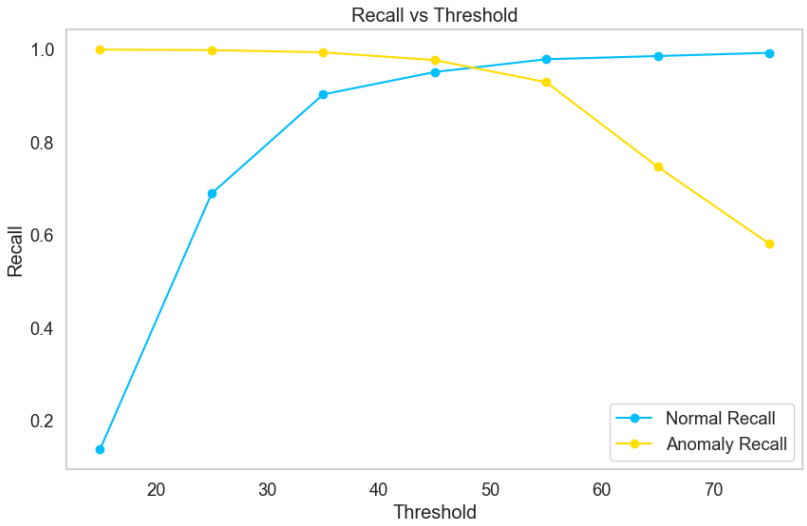
QUESTION 1 (PART - A)

```python
# Set thresholds
thresholds = range(15, 76, 10)

# Calculate predictions and losses for normal and anomaly datasets
normal_losses = predict(model, test_normal_dataset)
anomaly_losses = predict(model, test_anomaly_dataset)

# Initialize results storage
results = []

# Iterate through thresholds to compute recall values
for threshold in thresholds:
    normal_recall = sum(l <= threshold for l in normal_losses) / len(normal_losses)
    anomaly_recall = sum(l > threshold for l in anomaly_losses) / len(anomaly_losses)
    results.append({"Threshold": threshold, "Normal Recall": normal_recall, "Anomaly Recall": anomaly_recall})

# Convert results to a DataFrame for visualization
results_df = pd.DataFrame(results)

# Display or save the results
print(results_df)
```

	Threshold	Normal Recall	Anomaly Recall
0	15	0.137931	1.000000
1	25	0.689655	0.999039
2	35	0.903448	0.994234
3	45	0.951724	0.977415
4	55	0.979310	0.929841
5	65	0.986207	0.747237
6	75	0.993103	0.581451



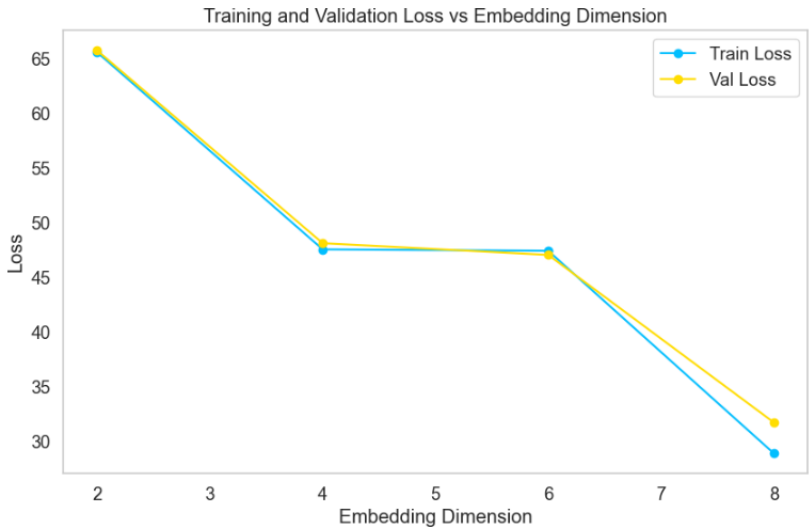
QUESTION 1 (PART - B)

As the threshold increases, the recall for normal data steadily improves, while the recall for anomalies declines. This trend occurs because a higher threshold allows more instances to be classified as normal, reducing the number of false negatives for the normal class and increasing its recall. However, this leniency causes some anomalies to be misclassified as normal, thereby increasing false negatives for the anomaly class and lowering its recall. At lower thresholds, the model is stricter, resulting in higher anomaly recall but lower normal recall, as some normal instances are misclassified as anomalies. This behavior illustrates a trade-off: a low threshold is ideal for maximizing anomaly recall, crucial in applications where detecting all anomalies (e.g., heart disease diagnosis) is critical, even at the cost of more false alarms for normal cases. Conversely, a higher threshold reduces false alarms for normal data but risks missing critical anomalies. Selecting the optimal threshold depends on the specific priorities of the application.

QUESTION 2 (PART - A)

```
# Vary embedding dimensions and collect results
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
embedding_dims = range(2, 9, 2)
results = []

for embedding_dim in embedding_dims:
    model = RecurrentAutoencoder(seq_len, n_features, embedding_dim).to(device)
    train_loss, val_loss = train_model(model, train_dataset, val_dataset, n_epochs=25)
    results.append({"Embedding Dim": embedding_dim, "Train Loss": train_loss, "Val Loss": val_loss})

# Convert results to DataFrame and display
results_df = pd.DataFrame(results)
print(results_df)

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(results_df["Embedding Dim"], results_df["Train Loss"], label="Train Loss", marker='o')
plt.plot(results_df["Embedding Dim"], results_df["Val Loss"], label="Val Loss", marker='o')
plt.xlabel("Embedding Dimension")
plt.ylabel("Loss")
plt.title("Training and Validation Loss vs Embedding Dimension")
plt.legend()
plt.grid()
plt.show()
```

```
   Embedding Dim  Train Loss   Val Loss
0              2   65.584744  65.746523
1              4   47.555376  48.120243
2              6   47.424777  47.047580
3              8   28.899111  31.726997
```



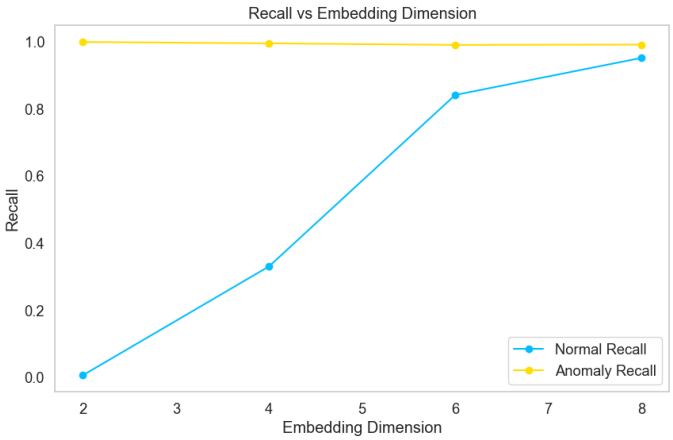
QUESTION 2 (PART - B)

As the embedding dimension increases, both the training and validation losses decrease. This trend suggests that higher embedding dimensions improve the model's ability to learn and reconstruct the data. With an embedding dimension of 2, the model's capacity is limited, leading to higher losses as it struggles to capture the complex patterns in the data. As the embedding dimension increases to 4 and 6, the model gains more representational capacity, resulting in a significant reduction in loss. At an embedding dimension of 8, the training loss reaches its lowest value, and the validation loss is also relatively low, indicating improved generalization. However, the validation loss at this point is slightly higher than the training loss, suggesting a potential risk of overfitting if the embedding dimension is increased further. The results highlight the importance of balancing embedding dimension to optimize model performance while avoiding overfitting.

QUESTION 2 (PART – C)

```python
# Define a function to compute recall
def compute_recall(model, test_normal_dataset, test_anomaly_dataset, threshold):
    normal_losses = predict(model, test_normal_dataset)
    anomaly_losses = predict(model, test_anomaly_dataset)

    normal_recall = sum(l <= threshold for l in normal_losses) / len(normal_losses)
    anomaly_recall = sum(l > threshold for l in anomaly_losses) / len(anomaly_losses)

    return normal_recall, anomaly_recall

# Initialize results storage
threshold = 45
recall_results = []

# Compute recall for each embedding dimension
for embedding_dim in embedding_dims:
    model = RecurrentAutoencoder(seq_len, n_features, embedding_dim).to(device)
    train_model(model, train_dataset, val_dataset, n_epochs=25)  # Train the model for 25 epochs

    normal_recall, anomaly_recall = compute_recall(
        model, test_normal_dataset, test_anomaly_dataset, threshold
    )
    recall_results.append({
        "Embedding Dim": embedding_dim,
        "Normal Recall": normal_recall,
        "Anomaly Recall": anomaly_recall
    })

# Convert results to DataFrame and display
recall_results_df = pd.DataFrame(recall_results)
print(recall_results_df)

# Plot recall values
plt.figure(figsize=(10, 6))
plt.plot(recall_results_df["Embedding Dim"], recall_results_df["Normal Recall"], label="Normal Recall", marker='o')
plt.plot(recall_results_df["Embedding Dim"], recall_results_df["Anomaly Recall"], label="Anomaly Recall", marker='o')
plt.xlabel("Embedding Dimension")
plt.ylabel("Recall")
plt.title("Recall vs Embedding Dimension")
plt.legend()
plt.grid()
plt.show()
```

```
   Embedding Dim  Normal Recall  Anomaly Recall
0              2       0.006897        0.999039
1              4       0.331034        0.995195
2              6       0.841379        0.990389
3              8       0.951724        0.991350
```



QUESTION 2 (PART – D)

As the embedding dimension increases, the recall for normal time-series improves significantly, while the recall for anomalies remains consistently high. With a small embedding dimension of 2, the model struggles to correctly classify normal time-series, resulting in a very low recall of 0.0069. This is because the limited representational capacity restricts the model's ability to learn and reconstruct the patterns of normal data effectively. However, as the embedding dimension increases to 4, 6, and 8, the normal recall improves substantially, reaching 0.9517 at an embedding dimension of 8. This indicates that a larger embedding dimension enables the model to better capture the nuances of normal data. On the other hand, the anomaly recall remains high across all embedding dimensions, starting at 0.9990 for an embedding dimension of 2 and slightly decreasing to 0.9913 at 8. This suggests that the model is inherently effective at detecting anomalies due to their distinct patterns, even with limited representational capacity. Overall, increasing the embedding dimension enhances the classification of normal time-series, while the detection of anomalies remains robust across the range of embedding dimensions.