Neural Networks and Deep Learning

Assignment 4: Advanced Topics in Deep Learning

University of Tehran

Faculty of Electrical and Computer Engineering

Submitted By:

Mohammad Taha Majlesi
Student ID: 810101504

September 26, 2025

Contents

List of Figures

List of Tables

1 Question 1: Image Captioning with ResNet50 + LSTM-GRU

Note: The provided report focuses exclusively on Question 2. This section serves as a placeholder for the content of Question 1 as described in the assignment prompt.

1.1 Introduction to Image Captioning

Image captioning is a fascinating task that sits at the intersection of Computer Vision and Natural Language Processing (NLP). The goal is to create a model that can take an image as input and produce a human-like, descriptive sentence as output. This process mimics the human ability to perceive a scene and describe it in words. The dominant architecture for this task is the Encoder-Decoder framework, which we will explore in this problem.

1.2 Dataset and Preprocessing

The foundation of any deep learning model is high-quality data. For this task, the Flickr8k dataset is used. Preprocessing involves two main streams:

1. **Image Preprocessing:** Preparing the visual data for the model. This typically involves resizing all images to a uniform size required by the CNN encoder and normalizing the pixel values to stabilize the training process.

2. **Text Preprocessing:** Preparing the textual captions. This includes cleaning the text (lowercasing, removing punctuation), creating a vocabulary of all unique words, and converting each word into a unique integer token. Special tokens like ‘¡start¿‘, ‘¡end¿‘, and ‘¡pad¿‘ are added to manage the sequence generation process.

1.3 Model Implementation

The model consists of two key components:

- **The Encoder:** A pre-trained Convolutional Neural Network (CNN), such as ResNet50, is used to ”understand” the image. We remove its final classification layer and use the output of the preceding layer as a rich feature vector—a numerical summary of the image’s content.

- **The Decoder:** A Recurrent Neural Network (RNN), such as an LSTM or GRU, takes the image feature vector from the encoder as its initial state. It then generates the caption word by word, using the previously generated word as input for the next step, until an ‘¡end¿‘ token is produced.

1.4 Training and Evaluation

Training involves feeding the model image-caption pairs and optimizing its weights to minimize a loss function (like cross-entropy). For evaluation, we use metrics like BLEU score, which compares the machine-generated caption to several human-written reference captions to measure its quality and fluency.

2 Question 2: Time Series Prediction for Clinical Events

2.1 The Importance of Clinical Time Series Analysis

In modern medicine, accurately forecasting a patient's health trajectory is a primary goal. The ability to model and predict future events based on historical clinical data—such as vital signs, lab results, and medication history—is transformative. It allows healthcare providers to shift from a reactive to a proactive approach, enabling early interventions that can prevent adverse events, optimize resource allocation, and ultimately improve patient outcomes.

One of the most significant challenges in this domain is the inherent complexity of clinical time series data. A patient's physiological state can be influenced by thousands of interacting factors, resulting in highly noisy, multi-dimensional, and often incomplete data streams. The referenced academic paper introduces a model leveraging a Long Short-Term Memory (LSTM) architecture, which is exceptionally well-suited for this challenge. LSTMs can learn from two distinct information sources: the immediate history of recent medical events and a compressed, long-term memory stored in the model's "hidden state," which captures dependencies over extended periods.

2.2 Methodology and Foundational Models

2.2.1 Overview of Common Time Series Models

The study of time series has a rich history, leading to a variety of modeling techniques. These can be broadly categorized into classical statistical methods and modern deep learning approaches.

Classical Statistical Models (Markovian Approaches) For many years, models based on the **Markov property** were standard for time series prediction. This property posits that the future state of a system depends only on its present state, not on the sequence of events that preceded it. The joint probability of a sequence is simplified as:

$$p(y_1, y_2, \ldots, y_T) = p(y_1) \prod_{t=2}^{T} p(y_t | y_{t-1})$$

A primary limitation of standard Markov models is their assumption that all states are fully observable. In reality, especially in medicine, the underlying health state is not directly visible. This leads to the use of **Hidden Markov Models (HMMs)**, which assume that the observed data (e.g., vital signs) are probabilistic manifestations of an unobserved, or hidden, state. HMMs have proven effective, but they typically

assume a discrete state space, which is often an oversimplification for continuous biological processes.

Modern Deep Learning Models (Neural Networks) With the rise of deep learning, Recurrent Neural Networks (RNNs) and their advanced variants, **Long Short-Term Memory (LSTM)** and **Gated Recurrent Unit (GRU)**, have become state-of-the-art for sequence modeling. Unlike Markov models, which have a limited memory, LSTMs are explicitly designed to learn long-term dependencies in data. They achieve this through a system of "gates" (input, forget, and output gates) that regulate the flow of information into and out of a cell's memory, mitigating the vanishing gradient problem that plagues simple RNNs.

The referenced paper explores four foundational models, which we will briefly review.

1. **Current Markov State:** The simplest model, where the prediction for the next time step, $\hat{Y}_{t+1}$, is a direct function of the current state, y_t. The relationship is typically learned via a linear transformation followed by a non-linear activation function like sigmoid.
$$\hat{Y}_{t+1} = \sigma(W \cdot y_t + b)$$

2. **Binary History (LR-Binary):** An extension of the Markov model that considers all past events, not just the most recent one. The input y_t is a multi-hot encoded vector, where each '1' signifies the occurrence of a specific event at some point in the past.

3. **Count History (LR-Count):** Similar to the binary history model, but instead of binary indicators, the input vector contains counts of how many times each event has occurred in the past.

4. **Current LSTM State:** A departure from Markovian assumptions. This model uses the LSTM's hidden state, which is a compressed representation of the entire past sequence, to make its prediction. This allows it to capture more complex and long-range temporal patterns.

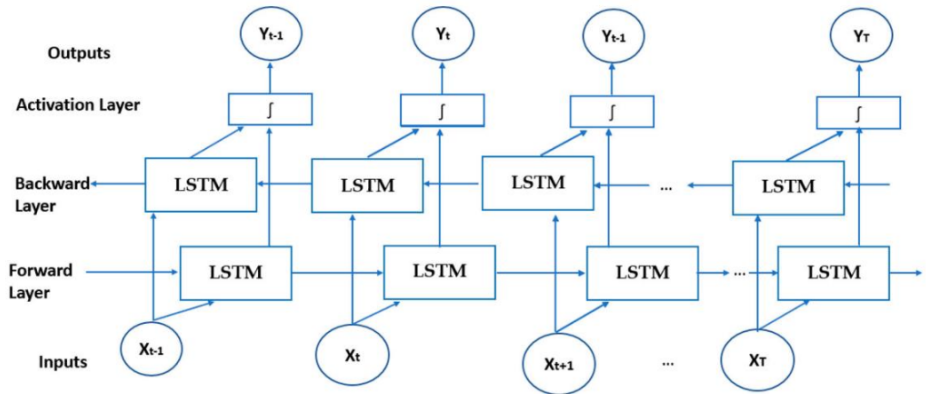

Figure 1: A conceptual diagram of a Bidirectional LSTM model, which processes sequence data in both forward and backward directions to capture context from both past and future time steps. This figure is based on a diagram from the reference paper.

2.3 Data Preparation and Exploratory Analysis

2.3.1 Feature Selection and Engineering

The timely diagnosis of sepsis, a life-threatening condition, is critical. Our objective is to build a model capable of predicting a patient's heart rate—a key vital sign—using time series data from both septic and non-septic patients.

Initial Data Overview The dataset contains 44 distinct features, including 8 vital signs, 26 laboratory measurements, and 16 demographic or administrative data points. The target variable indicates the presence of sepsis. Our dataset is split into 39,136 training samples, 3,244 validation samples, and 689 test samples.

	Unnamed: 0	Hour	HR	O2Sat	Temp	SBP	MAP	DBP	Resp	EtCO2	...	Fibrinogen	Platelets	Age	Gender	Unit1	Unit2	HospAdmTime	ICULOS	SepsisLabel	Patient_ID
0	100	100	100.0	100.0	37.43	129.0	87.0	68.0	21.0	33.03	...	579.25	106.04	27.92	1	0.0	1.0	-0.03	101	0	9
1	101	101	98.0	100.0	37.33	132.0	85.0	66.0	20.0	33.03	...	582.88	106.36	27.92	1	0.0	1.0	-0.03	102	0	9
2	102	102	103.0	99.0	37.41	121.0	75.0	58.0	26.0	33.03	...	586.50	106.68	27.92	1	0.0	1.0	-0.03	103	0	9
3	103	103	95.0	100.0	37.49	122.0	81.0	64.0	33.0	33.03	...	590.12	107.00	27.92	1	0.0	1.0	-0.03	104	0	9
4	104	104	94.0	100.0	37.57	129.0	81.0	63.0	23.0	33.03	...	593.75	110.09	27.92	1	0.0	1.0	-0.03	105	0	9

5 rows × 44 columns

Figure 2: A snapshot of the training data, showing the first 5 rows and a subset of the available features. Each row represents a single time-stamped observation for a patient.

Correlation-Based Feature Reduction Many machine learning algorithms perform better when input features are not highly correlated (a phenomenon known as multi-collinearity). To reduce redundancy and simplify our model, we analyze the correlation matrix of the features. We select features that have low correlation with each other, aiming to create a set of more independent predictors. After this process, we arrive at a reduced set of 20 core features for our models.

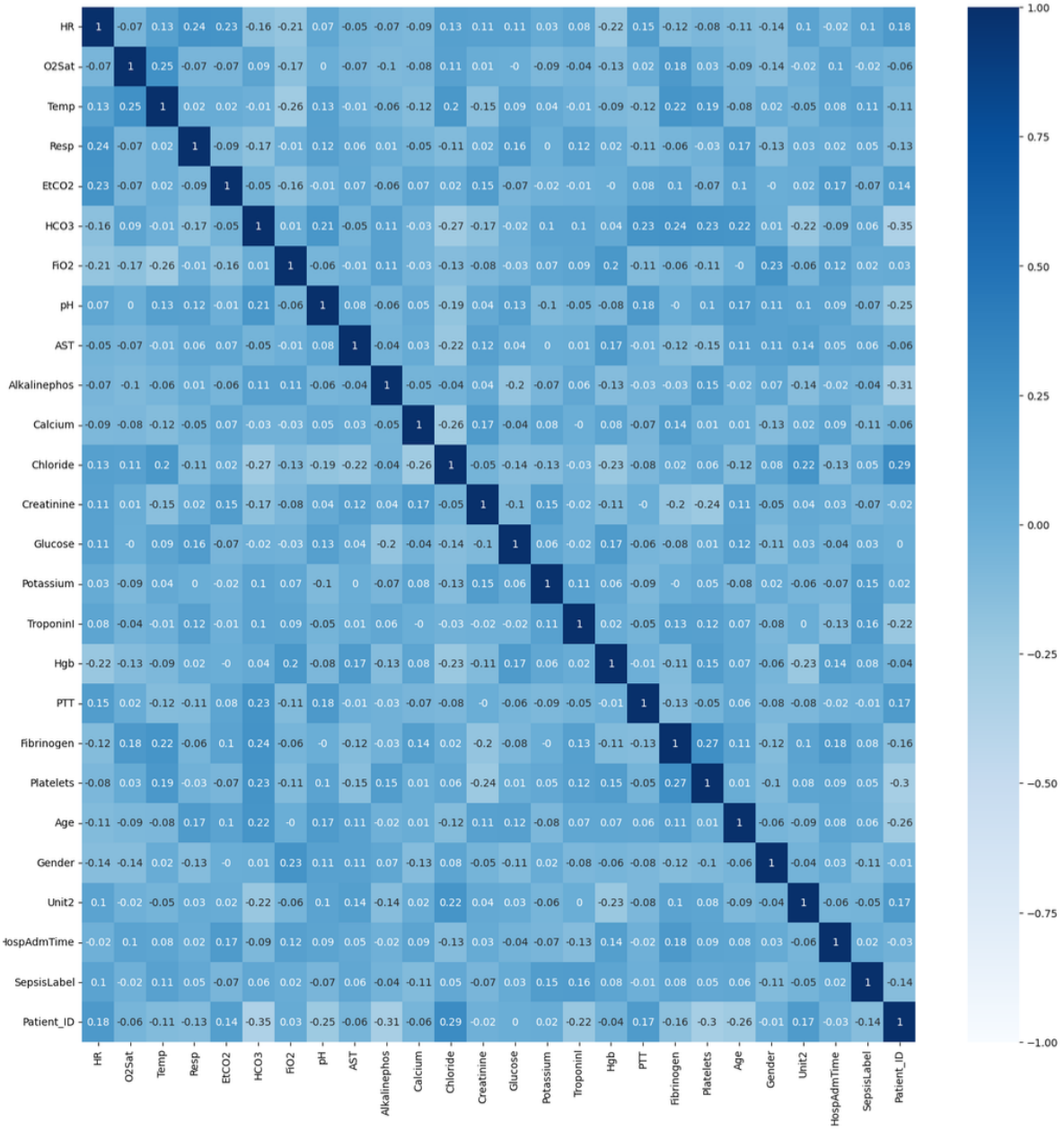

Figure 3: Correlation matrix heatmap highlighting pairs of features with low correlation. The color scale indicates the strength and direction of the correlation, with darker colors representing stronger relationships.

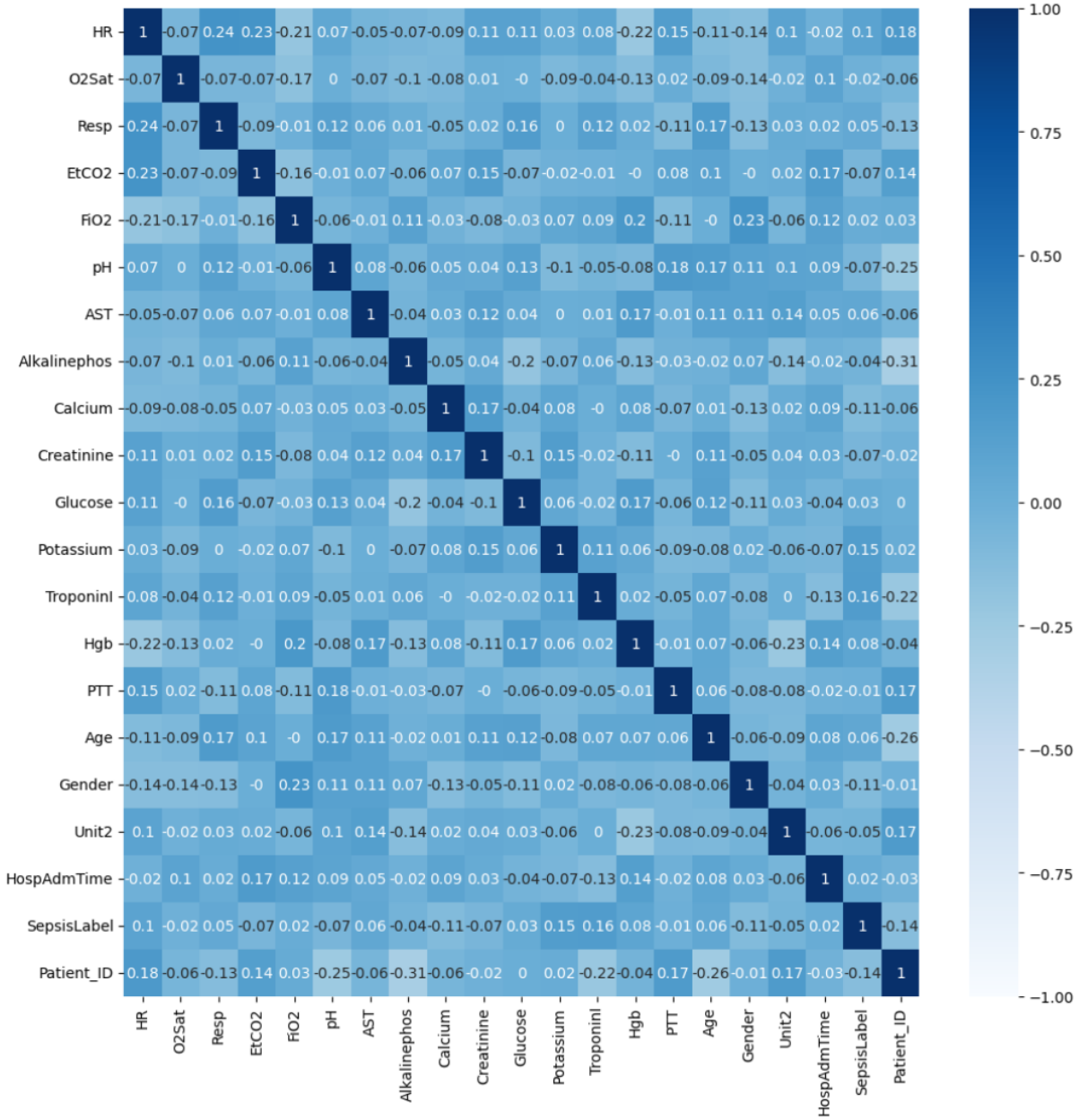

Figure 4: The final correlation matrix for the 20 features selected for model training. This visualization confirms that we have successfully reduced the multicollinearity in our input data.

2.3.2 Understanding Stationarity in Time Series

A core concept in time series analysis is **stationarity**. A time series is stationary if its statistical properties—specifically its mean, variance, and autocorrelation—are all constant over time. Many statistical models, including the SARIMAX model we will use, assume stationarity. Non-stationary data, which often exhibits trends or seasonality, can lead to unreliable and spurious results.

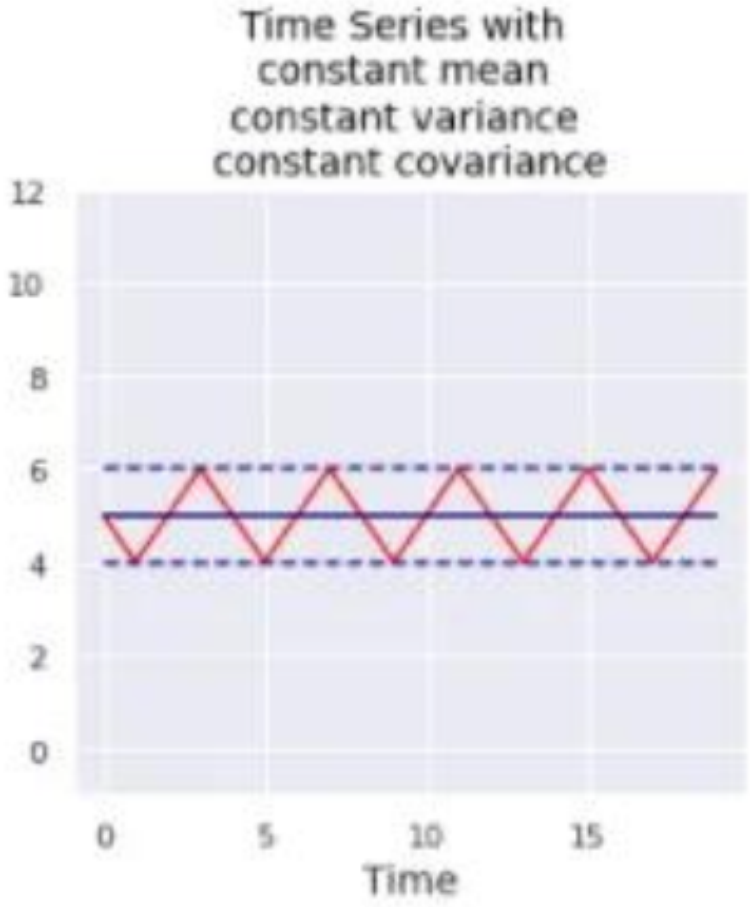

Figure 5: An example of a stationary time series. Note how the mean and variance remain consistent across the time axis.

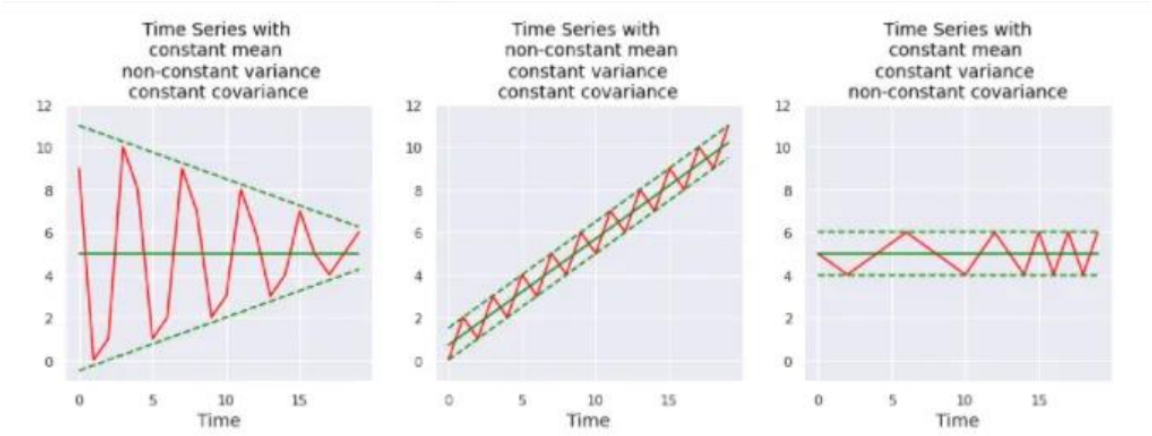

Figure 6: Examples of non-stationary time series, exhibiting clear trends and changing variance over time.

Achieving Stationarity through Differencing To handle non-stationary data, we can apply transformations. A common and effective technique is **differencing**, where we transform the series from values to the differences between consecutive values. This often helps to stabilize the mean and remove trends. We use the Augmented Dickey-Fuller (ADF) test, a statistical hypothesis test, to check for stationarity. The null hypothesis of the ADF test is that the series is non-stationary. If the test's p-value is below a significance threshold (e.g., 0.05), we can reject the null hypothesis and conclude that the series is stationary.

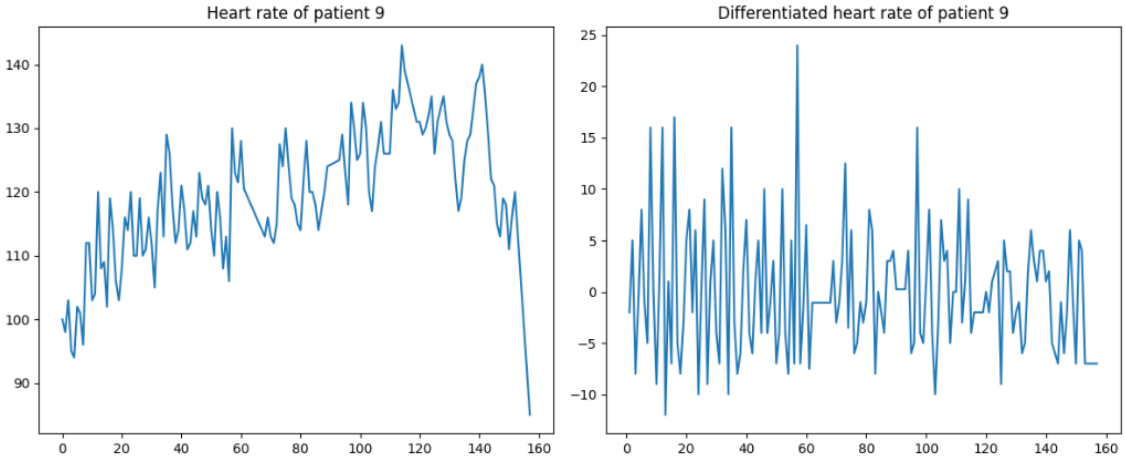

Figure 7: On the left, the original heart rate time series for a patient, showing non-stationary behavior. On the right, the same series after applying first-order differencing. The transformed series appears much more stationary, with a stable mean around zero.

2.3.3 SARIMAX Modeling for Parameter Estimation

To determine appropriate input and output window sizes for our deep learning models, we first build a classical statistical model: **SARIMAX** (Seasonal AutoRegressive Integrated Moving Average with eXogenous variables). This model helps us understand the underlying temporal structure of the data. To configure it, we need to determine its key parameters (p, d, q).

Using ACF and PACF Plots We use the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots of the stationary time series to estimate the 'p' (autoregressive order) and 'q' (moving average order) parameters.

- **ACF Plot:** Shows the correlation of the series with its own lagged values.
- **PACF Plot:** Shows the partial correlation of the series with its lagged values, controlling for the values of the intermediate lags.

A sharp cutoff in the PACF plot suggests the AR order ('p'), while a sharp cutoff in the ACF plot suggests the MA order ('q').

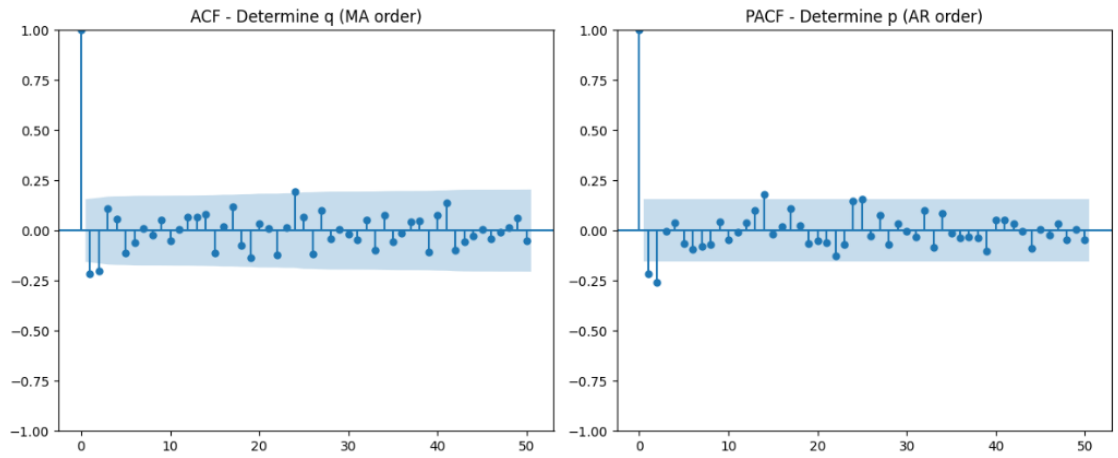

Figure 8: ACF (left) and PACF (right) plots for the differenced (stationary) heart rate data. The significant spikes outside the blue shaded confidence interval help us estimate the 'p' and 'q' parameters for the SARIMAX model.

Based on the analysis, we experiment with different SARIMAX configurations.

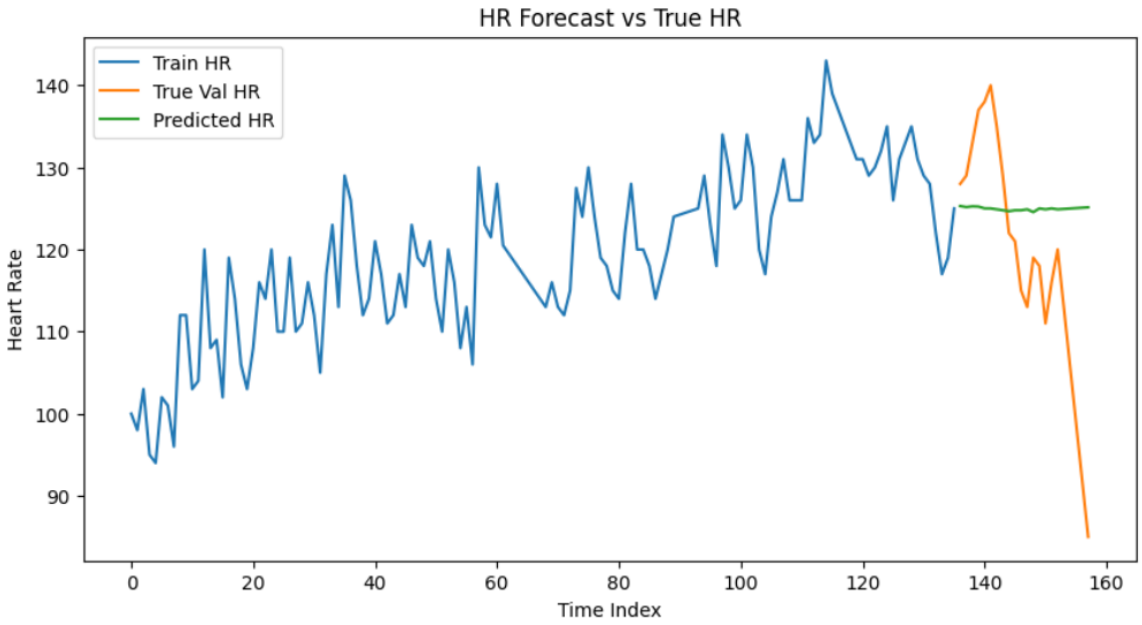

Figure 9: SARIMAX model prediction (orange line) versus the true heart rate values (blue line) on the validation set, using parameters (p,d,q) = (2,1,2). The prediction seems to lag behind the true values.

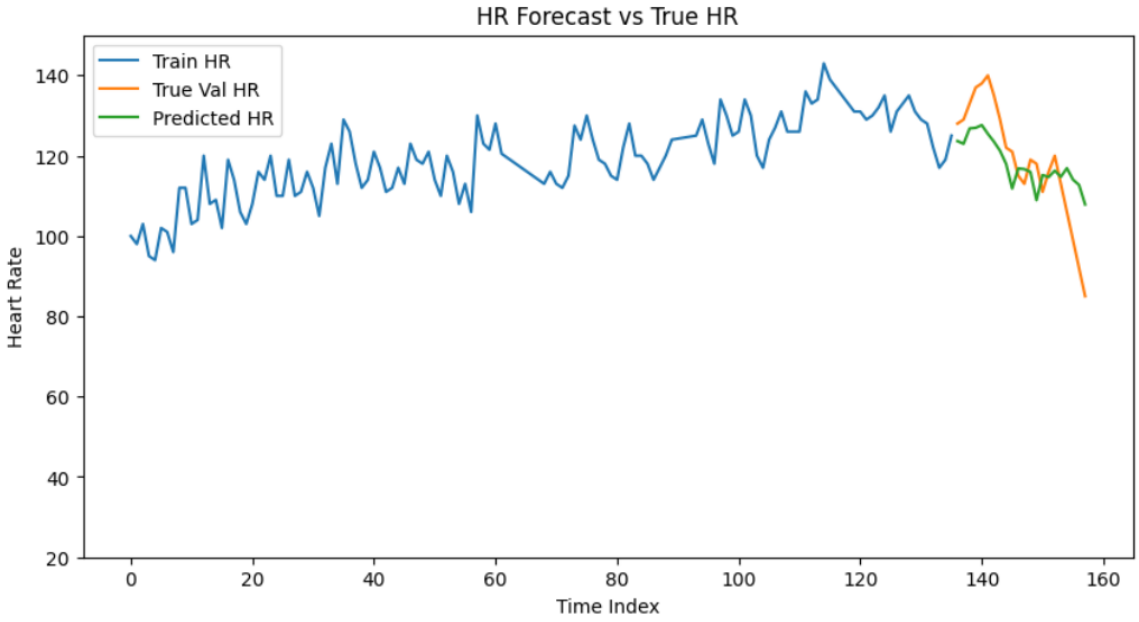

Figure 10: A more complex SARIMAX model prediction with parameters (p,d,q) = (14,1,24). This configuration, informed by deeper lags in the ACF/PACF plots, appears to capture the data's structure more effectively.

2.4 Deep Learning Model Implementation and Training

Having established a baseline with SARIMAX, we now turn to our deep learning models. We will build five different architectures: a simple Dense network, GRU, LSTM, Bidirectional GRU, and Bidirectional LSTM.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MarkovPredictor                          [1, 1]                    --
├─Flatten: 1-1                           [1, 40]                   --
├─Linear: 1-2                            [1, 256]                  10,496
├─BatchNorm1d: 1-3                       [1, 256]                  512
├─Dropout: 1-4                           [1, 256]                  --
├─Linear: 1-5                            [1, 1]                    257
├─Sigmoid: 1-6                           [1, 1]                    --
==========================================================================================
Total params: 11,265
Trainable params: 11,265
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.01
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.05
Estimated Total Size (MB): 0.05
==========================================================================================
```

Figure 11: Model summary for the fully connected (Dense) network. This is our simplest deep learning baseline.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
GRUPredictor                             [1, 1]                    --
├─Flatten: 1-1                           [1, 40]                   --
├─GRU: 1-2                               [1, 512]                  850,944
├─Dropout: 1-3                           [1, 512]                  --
├─Linear: 1-4                            [1, 128]                  65,664
├─Dropout: 1-5                           [1, 128]                  --
├─Linear: 1-6                            [1, 1]                    129
==========================================================================================
Total params: 916,737
Trainable params: 916,737
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 435.75
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 3.67
Estimated Total Size (MB): 3.67
==========================================================================================
```

Figure 12: Model summary for the unidirectional GRU network. This model can capture temporal dependencies.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTMPredictor                            [1, 1]                    --
├─LSTM: 1-1                              [1, 2, 512]               1,093,632
├─Linear: 1-2                            [1, 128]                  65,664
├─BatchNorm1d: 1-3                       [1, 128]                  256
├─Dropout: 1-4                           [1, 128]                  --
├─Linear: 1-5                            [1, 1]                    129
├─Sigmoid: 1-6                           [1, 1]                    --
==========================================================================================
Total params: 1,159,681
Trainable params: 1,159,681
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 2.25
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 4.64
Estimated Total Size (MB): 4.65
==========================================================================================
```

Figure 13: Model summary for the unidirectional LSTM network, a slightly more complex recurrent architecture than GRU.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
GRUPredictor                             [1, 1]                    --
├─GRU: 1-1                               [1, 2, 512]               820,224
├─Linear: 1-2                            [1, 128]                  65,664
├─BatchNorm1d: 1-3                       [1, 128]                  256
├─Dropout: 1-4                           [1, 128]                  --
├─Linear: 1-5                            [1, 1]                    129
├─Sigmoid: 1-6                           [1, 1]                    --
==========================================================================================
Total params: 886,273
Trainable params: 886,273
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 1.71
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 3.55
Estimated Total Size (MB): 3.56
==========================================================================================
```

Figure 14: Model summary for the Bidirectional GRU network, which processes sequences in both chronological and reverse-chronological order.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTMPredictor                            [1, 1]                    --
├─LSTM: 1-1                              [1, 2, 1024]              2,187,264
├─Linear: 1-2                            [1, 128]                  131,200
├─BatchNorm1d: 1-3                       [1, 128]                  256
├─Dropout: 1-4                           [1, 128]                  --
├─Linear: 1-5                            [1, 1]                    129
├─Sigmoid: 1-6                           [1, 1]                    --
==========================================================================================
Total params: 2,318,849
Trainable params: 2,318,849
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 4.51
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 9.28
Estimated Total Size (MB): 9.29
==========================================================================================
```

Figure 15: Model summary for the Bidirectional LSTM network, the most complex architecture in our experiment.

Training Process The models are trained to minimize the Mean Squared Error (MSE) loss. We monitor both training and validation loss to prevent overfitting. The training curves below show how the model error decreases over epochs.

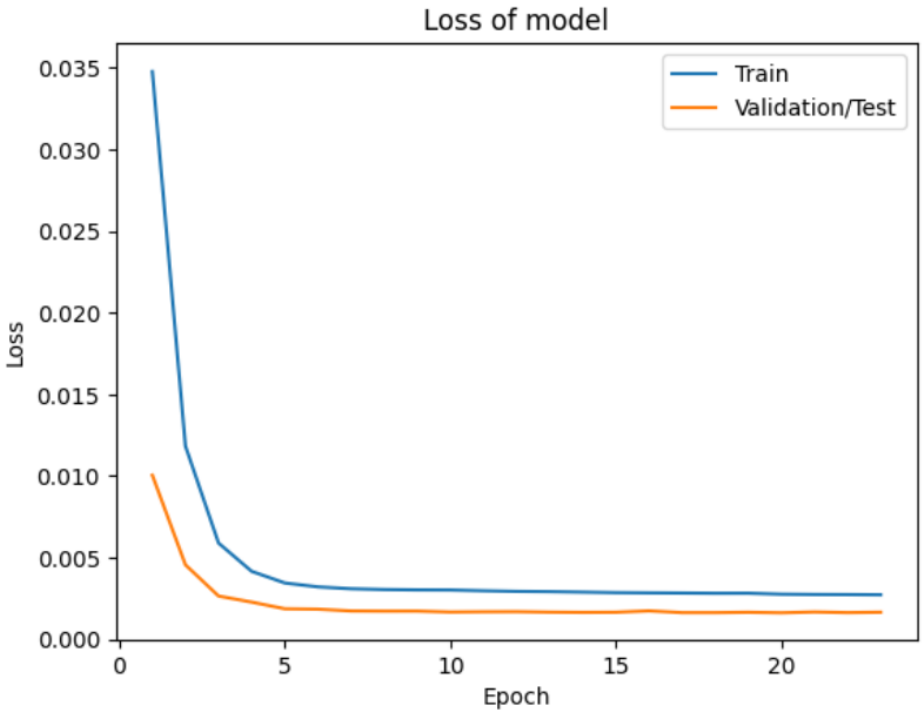

Figure 16: Loss curve for the fully connected network. The model converges quickly, with both training and validation loss stabilizing at a low value.

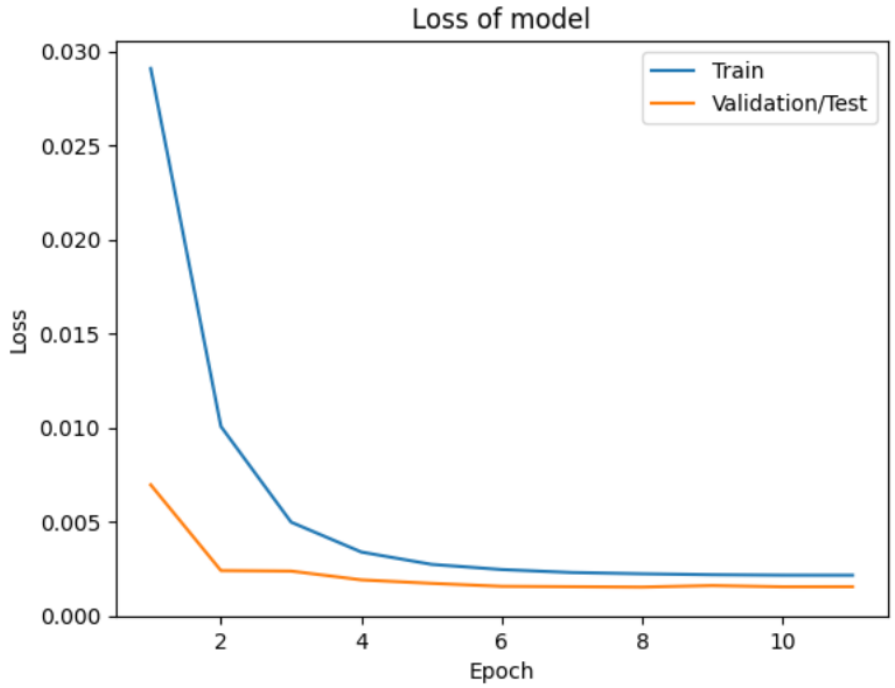

Figure 17: Loss curve for the GRU model. Similar to the dense model, it shows good convergence without signs of significant overfitting.

2.5 Results and Discussion

2.5.1 Quantitative Evaluation on Validation and Test Sets

We evaluate the performance of all models using standard regression metrics. The tables below summarize the results on the validation data, for a specific patient (Patient 9), and on the final held-out test set.

Table 1: Summary of evaluation metrics for all deep learning models on the validation dataset.

Metric	Dense	GRU	LSTM	Bi-GRU	Bi-LSTM
Explained Variance	0.776	0.790	0.791	**0.792**	0.791
Mean Squared Error	0.0016	0.0015	0.0015	**0.0015**	0.0015

Table 2: Detailed evaluation metrics for all models on the specific time series of Patient 9.

Model	Explained Variance	Mean Squared Error
SARIMA (p=2, q=2)	0.0007	239.01
SARIMA (p=14, q=25)	0.5033	102.21
Dense (Markov)	0.8122	38.36
GRU	0.8332	34.57
LSTM	0.8494	35.50
Bidirectional GRU	**0.8648**	**27.62**
Bidirectional LSTM	0.8529	32.17

Table 3: Final evaluation metrics for all deep learning models on the held-out test dataset.

Metric	Dense	GRU	LSTM	Bi-GRU	Bi-LSTM
Explained Variance	0.659	**0.665**	0.664	0.664	0.655
R2 Score	0.658	**0.664**	0.664	0.664	0.655

The results consistently show that the deep learning models significantly outperform the classical SARIMAX models. Among the neural network architectures, the recurrent models (GRU and LSTM) perform better than the simple dense network, and the bidirectional models, particularly the Bidirectional GRU, show the strongest performance on specific patient predictions. This suggests that considering both past and future context is beneficial for this task.

2.5.2 Qualitative Evaluation: Visualizing Predictions

Visual inspection of the predictions provides valuable insights into model behavior.

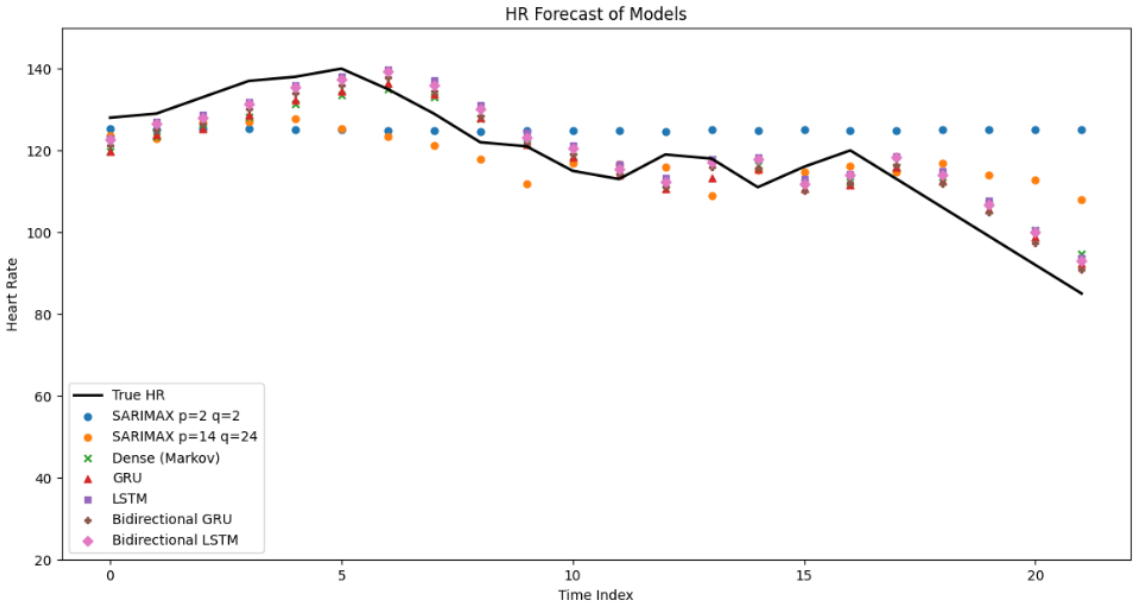

Figure 18: Heart rate predictions from all models plotted against the true values for a segment of the test set. The deep learning models track the true signal much more closely than the baseline models.

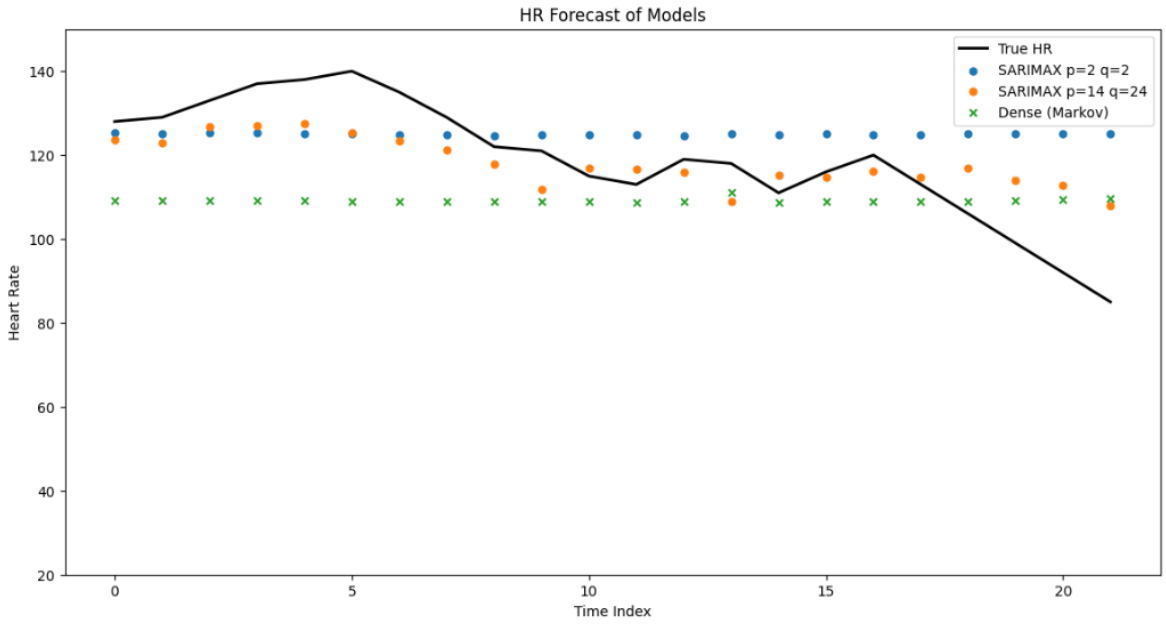

Figure 19: A closer look at the predictions for Patient 9. The superiority of the deep models over the SARIMAX (p=14, q=24) model is evident.

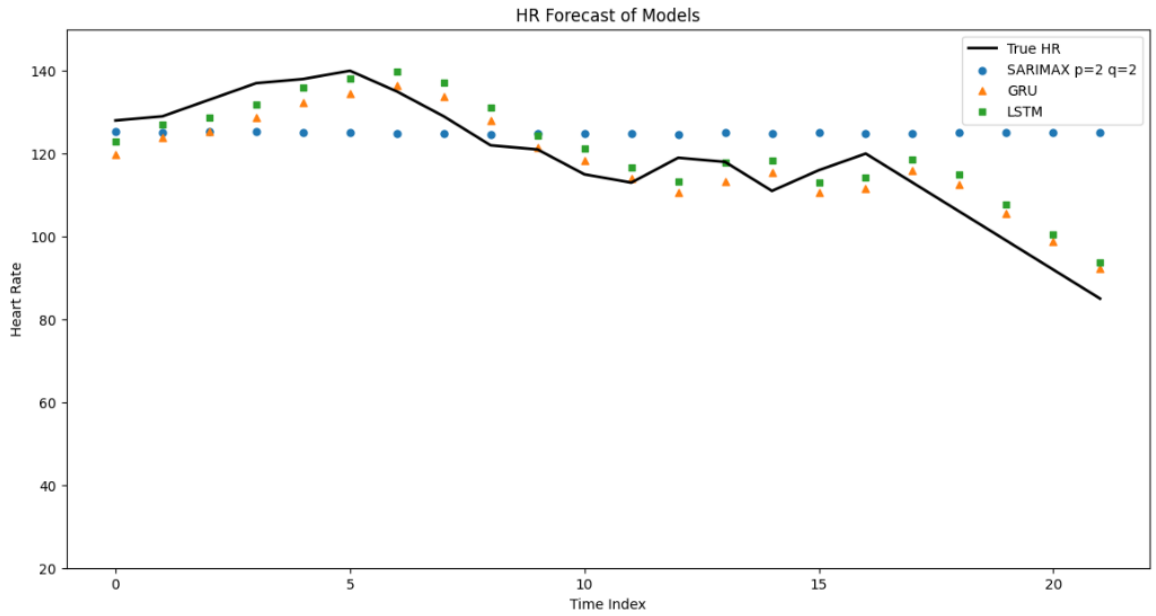

Figure 20: Comparing the unidirectional models (GRU and LSTM) for Patient 9. Both models capture the general trend well.

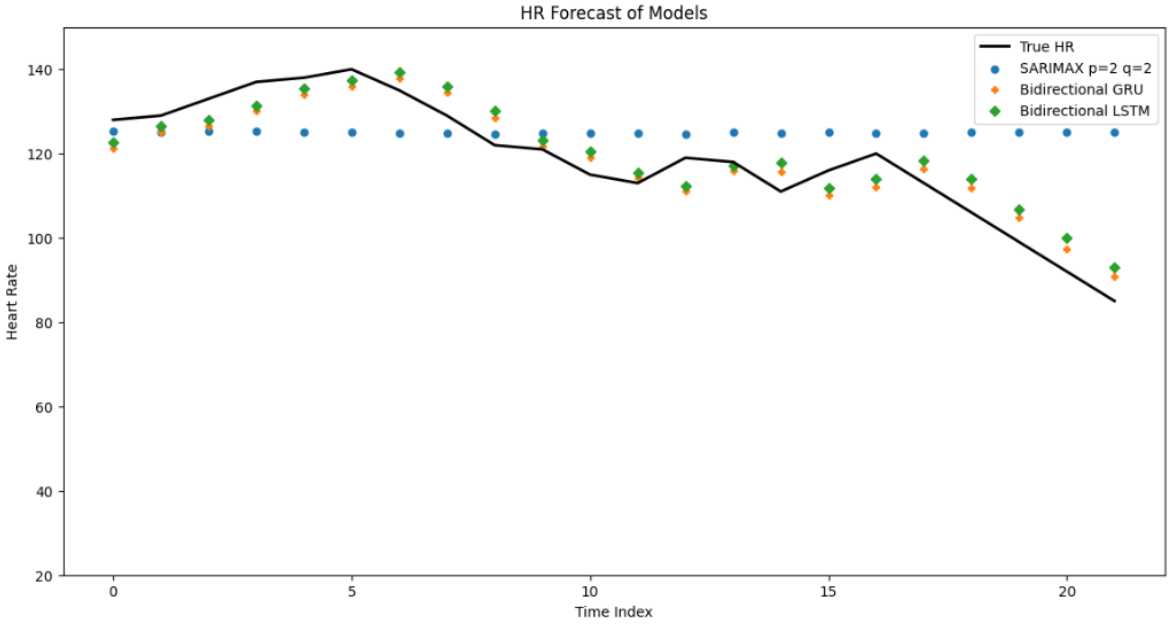

Figure 21: Comparing the bidirectional models for Patient 9. The predictions are very close to the true values, demonstrating their high accuracy.

2.6 Advanced Modeling: Maximum Log-Likelihood Estimation

As a final exploration, we move beyond simply predicting a single value (point estimate) and instead try to predict the entire probability distribution of the target variable. We build an LSTM model that outputs the parameters of a Gaussian distribution—the mean (μ) and the variance (σ^2)—at each time step.

The Loss Function Instead of MSE, we use the **Negative Log-Likelihood (NLL)** as our loss function. The goal of Maximum Log-Likelihood Estimation (MLE) is to find the model parameters that maximize the probability of the observed data. Minimizing the NLL is equivalent to maximizing this probability. For a Gaussian distribution, the NLL loss is:

$$\text{Loss} = \frac{1}{2}\log(\sigma^2) + \frac{(x-\mu)^2}{2\sigma^2}$$

This loss function penalizes the model not only for being wrong (the MSE term) but also for being uncertain (the $\log(\sigma^2)$ term).

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLELSTMPredictor                         [1, 1]                    --
├─LSTM: 1-1                              [1, 2, 512]               1,093,632
├─Linear: 1-2                            [1, 128]                  65,664
├─BatchNorm1d: 1-3                       [1, 128]                  256
├─ReLU: 1-4                              [1, 128]                  --
├─Dropout: 1-5                           [1, 128]                  --
├─Linear: 1-6                            [1, 2]                    258
├─ReLU: 1-7                              [1, 1]                    --
==========================================================================================
Total params: 1,159,810
Trainable params: 1,159,810
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 2.25
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 4.64
Estimated Total Size (MB): 4.65
==========================================================================================
```

Figure 22: Summary of the LSTM architecture designed for MLE. Note that the final layer outputs two values, corresponding to the predicted mean and variance.

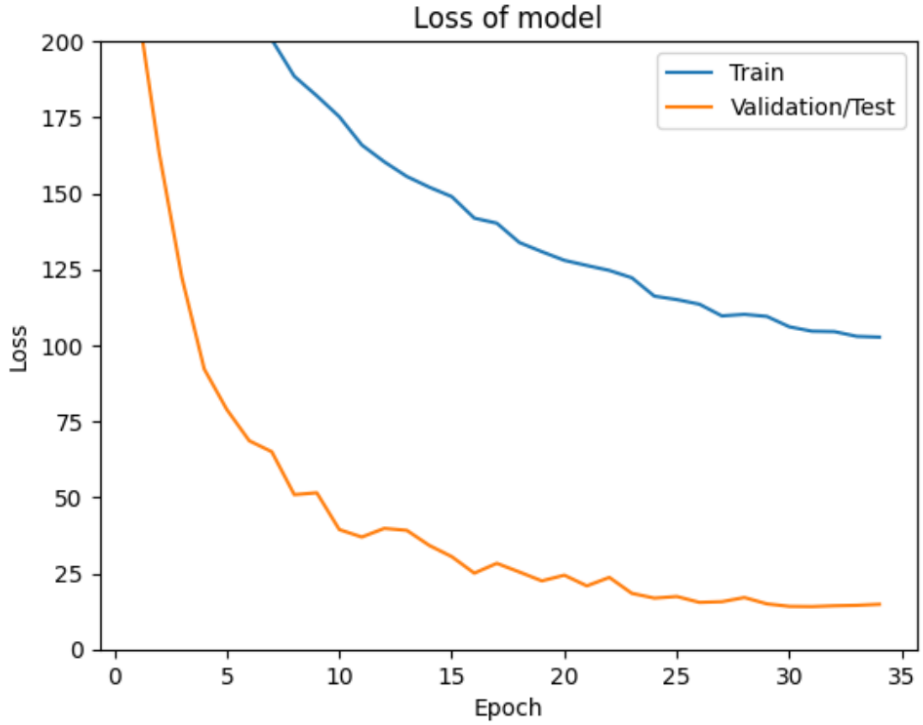

Figure 23: The NLL loss curve during training for the MLE model. The model converges to a stable loss, indicating that it has learned to predict the distribution parameters effectively.

2.7 Final Conclusion

This comprehensive assignment demonstrated the process of building, training, and evaluating models for clinical time series prediction. Our key findings are as follows:

1. **Data Preprocessing is Crucial:** Techniques like feature selection, stationarity testing, and normalization are essential for building robust models.

2. **Deep Learning Surpasses Classical Methods:** All deep learning architectures significantly outperformed the SARIMAX statistical model, highlighting their ability to capture complex, non-linear patterns in the data.

3. **Recurrent Architectures are Superior:** Recurrent models like GRU and LSTM were more effective than a simple dense network, confirming their suitability for sequential data. Bidirectional models showed a slight edge, suggesting that broader temporal context is valuable.

4. **Probabilistic Forecasting is Possible:** The Maximum Log-Likelihood Estimation approach shows that we can move beyond point predictions to estimate the full probability distribution of future outcomes, providing a richer, more informative forecast that includes a measure of uncertainty.

Throughout this project, we gained practical experience in handling real-world time series data and implementing a range of models from classical statistics to advanced deep learning, providing a solid foundation for further work in this critical domain.