Hybrid Integrative Imputation Network for Multivariate Time Series Data

Viktoriia Kharchenko
April 2024

1 Introduction

In data-centric applications, handling missing values effectively is essential for building accurate and reliable machine learning (ML) models. Traditional imputation methods often struggle when faced with complex data dependencies, especially in multivariate time series where missing values inevitably occur both across variables and over time. This problem is particularly pronounced in the domain of electronic health records (EHR), which are essential for patient care management but are not primarily designed for research studies. The EHR records are visualized in Figure 1, which underscores the multidimensional nature of the data, including patient space, time space, and variable space.

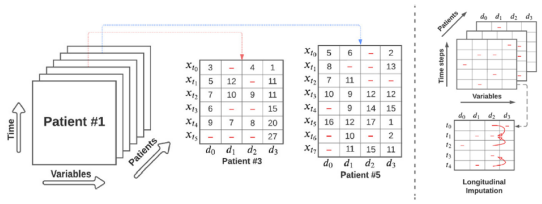

Figure 1: Structure of multivariate time series health data illustrating patient space, timespace, and variable space.

EHR data collection is inherently different from the structured records of clinical research trials. Patient visits and the measurements of clinical variables occur irregularly, influenced by the physician's recommendations and the patient's health status. This results in data with varying numbers of visits (time points) and measurements taken at inconsistent intervals, dictated by patient conditions, medical protocols, and administrative reasons. This means that the researchers do not have the consistent regular data inputs that ML algorithms typically require.

Moreover, the application of advanced machine learning methods in healthcare often encounters challenges due to the arbitrary selection of imputation methods, which are seldom validated or justified in the health science literature [4]. In scenarios where a complete case analysis is employed, samples with missing entries are often excluded to create a dataset free of missing values. This approach tends to disproportionately retain data from sicker patients who undergo more frequent testing, thereby introducing a selection bias that can significantly compromise the generalizability of predictive models. Such exclusions can also affect the performance of deep learning models, which are particularly reliant on large and diverse datasets to train effectively.

The primary goal of this project is to enhance the efficiency of imputation methods for multivariate time series data. Within the scope of this project, extensive analysis was conducted on three state-of-the-art model architectures: Bidirectional Recurrent Imputation for Time Series (BRITS) [1], Context-Aware Time Series Imputation (CATSI) [6], and Self-Attention Imputation for Time Series (SAITS) [2]. Building on the insights gained from these models, I developed my own network solution. In my implementation, I integrated the core strengths of these analyzed architectures, incorporating the recurrent imputation and the self-attention mechanism.

2 Background

2.1 Missing Value Types

In missing data imputation research understanding the underlying reasons for missingness can significantly enhance the accuracy of the chosen imputation method. The literature commonly identifies three distinct categories of missing data: Missing Completely at Random (MCAR), Missing at Random (MAR) and Missing Not at

Random (MNAR).

MCAR type occurs entirely at random without any systematic pattern or dependency on observed or unobserved variables, the data sample is likely still representative, and the missing data does not introduce bias. **MAR** means that the probability of missing values is related to the observed data but not directly to the missing values themselves. For **MNAR** type, the data is missing based on the missing column itself, or the missingness is related to events or factors not measured by the researcher, introducing a systematic bias.

In practice, however, the assumption of MCAR rarely holds, especially in medical datasets, where a missing laboratory test result is often a choice influenced by prior results or clinical judgment. Thus, imputing values that are MNAR is inherently more challenging and requires more sophisticated methods to avoid introducing biases and errors.

2.2 Deep Learning Methods for Time Series Imputation

Advancements in deep learning have significantly enhanced the field of time series data imputation. Modern methods often employ sophisticated recurrent neural network (RNN) architectures, such as Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRUs), adept at addressing the dependencies associated with time series data.

One of the first methods to capture both longitudinal (time) and cross-sectional (variable) dependencies is **BRITS**[1]. It utilizes bidirectional LSTM networks to address missing values in time series data effectively. By processing data in both forward and reverse temporal directions, it captures comprehensive temporal dependencies. The model also utilizes the temporal decay mechanism, which assigns weights that diminish over longer intervals, ensuring that more recent observations have a greater influence on the current imputation. In parallel to this temporal processing, the model employs feature regression to exploit correlations among different variables at the same time point. Finally, the model combines the historical imputations derived from past data trends with the imputations suggested by the feature correlations by calculating a combination weight for each feature.

Similarly to BRITS architecture, **CATSI** [6] method uses bidirectional and cross-sectional dynamics for missing value imputation. The model integrates two components to optimize the imputation process. In the recurrent component, it utilizes a GRU cell to capture temporal dynamics effectively, encoding this information into a context vector that encompasses both static and dynamic features of the dataset. Then the output goes through the LSTM cells in a bidirectional manner. Another major component employs a fully connected neural network for cross-sectional imputation. The model then produces the final imputation by combining the recurrent imputations and the cross-feature imputations using a fusion layer.

The latest in the series of state-of-the-art imputation methods is **SAITS** [2], which introduces a novel self-attention mechanism known as diagonally-masked self-attention (DMSA). This technique modifies the traditional self-attention by applying a diagonal mask to the attention weights, effectively setting self-weights to a large negative value preventing each time step from contributing to its own estimation. The SAITS architecture incorporates multiple self-attention layers grouped into blocks. The first block processes the input data combined with a mask indicating missing values, applying an embedding layer followed by self-attention and positional encoding to generate a preliminary imputed output. This output is then combined with the original data to replace missing values and serve as input for the subsequent block. A second self-attention block refines the imputation by reprocessing the combined data. It further transforms and integrates the outputs using a learned weighted combination, which dynamically adjusts the influence of earlier imputations based on attention weights derived from the data's temporal dependencies.

3 Methodology

3.1 Data Pre-Processing

To prepare the data for imputation, I denote the multivariate time series for a specific patient by $X \in \mathbb{R}^{T \times D}$, where T is the length of the time series, and D is the number of variables. Each row x_t represents the observations at the t-th time step, with s_t denoting the corresponding time stamp.

A masking matrix M of the same dimensions as X indicates missingness in the time series:

$$m_{d,t} = \begin{cases} 1 & \text{if } d\text{-th variable is observed at time } s_t \\ 0 & \text{otherwise} \end{cases}$$

To manage the irregularity caused by missing values, I utilize an observation gap matrix δ, representing the time gap from the last observed point:

$$\delta_{d,t} = \begin{cases} s_t - s_{t-1} + \delta_{d,t-1} & \text{if } m_{d,t-1} = 0 \\ s_t - s_{t-1} & \text{if } m_{d,t-1} = 1 \\ 0 & \text{if } t = 1 \end{cases}$$

For pre-completion, a trainable temporal decay module calculates the decay factor γ_t based on the observation gap δ_t:

$$\gamma_t = \exp\left(-\max(0, W_\gamma \delta_t + b_\gamma)\right)$$

This decay is employed to moderate the influence of the last observed values based on how recent they are. The more time has elapsed since an observation, the less it should theoretically contribute to the current imputation, hypothesizing that more recent data better reflects the current state.

The raw data for each variable is normalized using min-max normalization:

$$x_d = \frac{x_d - \min(x_d)}{\max(x_d) - \min(x_d)}$$

The pre-completed data $\tilde{x}_{d,t}$ is then calculated as a combination of the last observed value $x_{d,t'}$ and the empirical mean $\bar{x}_d$:

$$\tilde{x}_{d,t} = \gamma_{d,t} x_{d,t'} + (1 - \gamma_{d,t})\bar{x}_d$$

Finally, the pre-processed data combines the observed values with the pre-completed values:

$$x_{d,t} = m_{d,t} x_{d,t} + (1 - m_{d,t})\tilde{x}_{d,t}$$

The parameters W_γ and b_γ of the decay module are updated during training, allowing for adaptive learning of the optimal imputation strategy based on the observed data characteristics. This preprocessing strategy aligns with the techniques described in the CATSI paper [6].

3.2 Model Architecture

As mentioned earlier, the design of my model is inspired by integrating architectural elements from BRITS, CATSI, and SAITS. The core idea was to develop a hybrid model that combines their respective advantages in handling different aspects of time-series data imputation. According to benchmarking performed in [4], the CATSI model consistently ranked as one of the top performers across all experiments, showcasing its effectiveness in diverse imputation scenarios. However, the BRITS model, while also delivering strong performance overall, was particularly notable for its superior capability in handling datasets with extremely high missing value rates (up to 80%), which was evident across all three types of missing data scenarios: MCAR, MAR, and MNAR. It is important to note that the SAITS model was not included in the benchmarking study referenced in [4], as it was developed 2 years later. Nevertheless, comparisons provided in the original SAITS paper [2] reveal that it significantly outperforms the BRITS model. This outcome is largely attributed to SAITS's use of the self-attention mechanism, which enhances the model's ability to discern relationships within data.

When designing my model, I adopted a structure similar to the CATSI model. The model combines two specialized components: one for capturing temporal dependencies and another for cross-feature imputation. These components are integrated through a fusion layer to produce the final imputation. Unlike CATSI, which utilizes a context vector to encapsulate global temporal dynamics, my model does not employ a standalone context vector. This decision aligns my model closely with the BRITS architecture, which focuses on leveraging bidirectional LSTM to capture temporal dependencies. Moreover, instead of using a fully connected neural network, as in CATSI for capturing cross-feature correlations, my model implements a self-attention mechanism, the core idea of SAITS. This choice allows for a more nuanced understanding of the relationships between different variables by dynamically weighting the importance of each feature in relation to others across different time steps. The fusion layer in my model diverges from CATSI's approach by employing a three-layer fully connected neural network architecture, rather than a simpler linear combination.

Figure 2 graphically portrays the structure of the developed Hybrid Integrative Imputation Network (HIIN), illustrating its composite framework. The following subsections aim to give clarity to the methodology behind each of the HIIN's components in detail.

3.2.1 Recurrent Imputation Component

This part of the model is dedicated to capturing temporal dependencies within the time series data. The recurrent component of the model combines the functionalities of a GRU and LSTM cells.

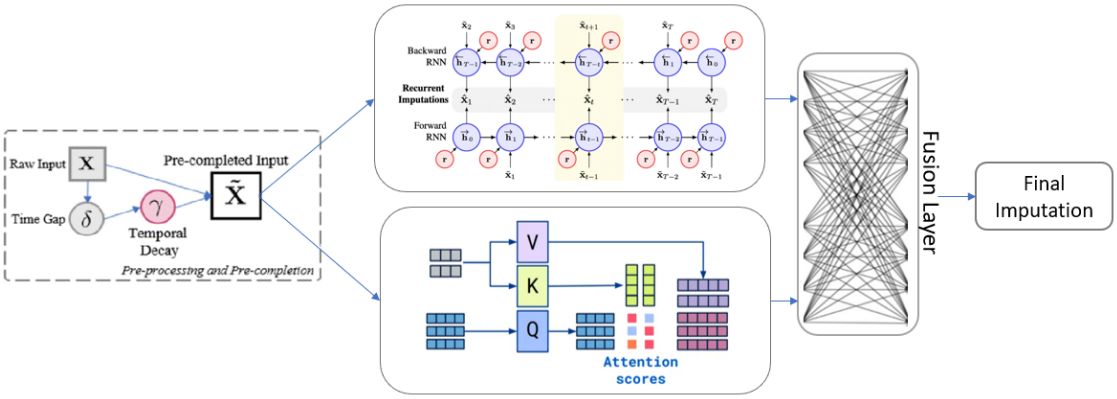

Figure 2: The framework of Hybrid Integrative Imputation Network

At each timestep, the GRU cell ingests the current input, which is a combination of preprocessed data reflective of the time gap since the last observation (capturing the decay of influence of previous data points), and generates a hidden state. This hidden state is a dynamic representation of the information up to that moment, summarizing the sequence's history and temporal context.

$$r = \text{GRU}(x \oplus \delta) \tag{1}$$

In Equation 1, r represents the output of the GRU cell, x is the current input, and δ denotes the time gap since the last observation. The symbol $\oplus$ indicates the concatenation of x and δ.

At each step, the GRU updates its hidden state by combining the previous state and the current input, modulated through a sophisticated gating mechanism, shown in Figure 3. The diagram illustrates the flow from one-time step to another, showing how the GRU updates its hidden state h_t based on the previous hidden state h_{t-1} and the current input x_t. The update gate z_t and reset gate r_t control how much past information is kept or discarded when forming the new state. The current memory content is then generated, which blends the new input with the past hidden state as dictated by the reset gate. Finally, the GRU cell produces the final memory state, which is a combination of the old state and the current memory content, controlled by the update gate.

The GRU's output then serves as the contextual input for the LSTM layers, which include a specified number of hidden layers, each with 58 units for the forward and backward directions and 28 units for context processing. This design

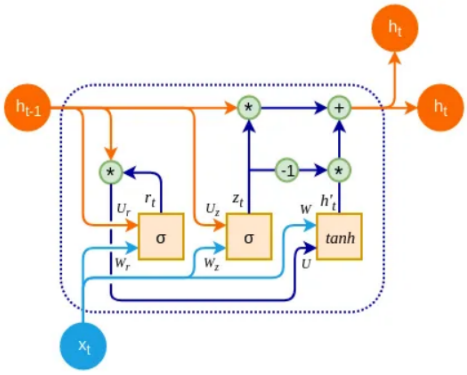

Figure 3: Structure of a Gated Recurrent Unit

allows the LSTM to capture the entire temporal spectrum of the sequence data. The initial hidden states (h_0) and cell states (c_0) for the LSTM are derived from the temporal vector r:

$$h_0 = W_{\text{hidden}} \cdot r + b_{\text{hidden}}$$
$$c_0 = \tanh(h_0)$$

The LSTM cells then are responsible for deep sequential processing of the input data in both forward and backward directions, creating a bidirectional representation of the sequence.

$$(h_{\text{f}}, c_{\text{f}}) = \text{LSTMCell}_{\text{f}}(x_t, (h_{\text{f}}, c_{\text{f}}))$$
$$(h_{\text{b}}, c_{\text{b}}) = \text{LSTMCell}_{\text{b}}(x_{T-t}, (h_{\text{b}}, c_{\text{b}}))$$

While GRU simplify the model architecture by using two gates and a single hidden state, LSTMs maintain a separate cell state alongside the hidden state, enabling more precise control over information flow. The structure of the LSTM cell is shown in Figure 4. This additional cell

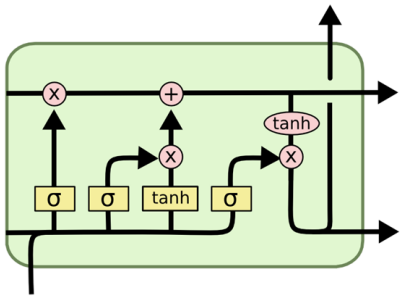

Figure 4: Structure of the LSTM cell

state in the LSTM helps to preserve information over longer periods without the risk of vanishing gradient issues that commonly affect standard RNNs. The distinct mechanisms of forget and input gates allow LSTM to decide more explicitly when to remember and when to update its memory, making it particularly useful for applications where long-term temporal dependencies are crucial.

3.2.2 Multi-head Self-Attention

The Multi-head Self-Attention component is used to capture complex interdependencies between different variables at various time steps. It extends the concept of attention across multiple 'heads', allowing the model to concurrently process information from different representational subspaces at different positions. The input sequence is divided into smaller parts or 'heads'. Each head processes the input independently, which enables the model to focus on different aspects of the sequence simultaneously.

Within each head, the attention mechanism computes three vectors for each element in the input sequence: $Queries(Q)$, $Keys(K)$, $Values(V)$. These vectors are derived from the input through distinct linear transformations specific to each head:

$$Q = W^Q x, \quad K = W^K x, \quad V = W^V x$$

where W^Q, W^K, and W^V are the weight matrices, and x is the input vector.

The attention function then computes a set of output values as weighted sums of the values, where the weight assigned to each value is determined by the softmax function applied to the dot products of the query with all keys:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Here, d_k is the dimension of the key vectors. This scaling factor helps in stabilizing the gradients during training.

In my implementation, the self-attention mechanism is configured with 2 heads, while the head dimension (the size of each head) is set to 2.

The outputs from all heads are then concatenated and passed through a final linear transformation to combine the information learned by each head:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2)W^O$$

where W^O is the weight matrix for the output linear transformation, integrating the information processed by each head.

3.2.3 Fusion Layer

The fusion layer is strategically designed to integrate the information processed by the recurrent imputation components and the multihead self-attention mechanism. The fusion mechanism is implemented using a neural network that takes inputs from both the recurrent and attention components. The inputs from these components are first concatenated to form a combined feature vector. Then the combined input undergoes a series of transformations facilitated by fully connected layers with the ReLU activation function. The process can be represented by the following equations:

$$\begin{aligned} x_{\text{combined}} &= \text{Concat}(x_{\text{RNN}}, x_{\text{Attention}}) \\ x_{\text{intermediate}} &= \text{ReLU}(W_1 x_{\text{combined}} + b_1) \\ x_{\text{fused}} &= \text{ReLU}(W_2 x_{\text{intermediate}} + b_2) \end{aligned}$$

Here, W_1, b_1, W_2, and b_2 are the weights and biases of the first and second fully connected layers, respectively.

Following the neural network processing, a gating mechanism is applied to dynamically weigh the influence of the combined features:

$$\beta = \sigma(W_g x_{\text{fused}} + b_g)$$

Where σ denotes the sigmoid activation function, and W_g and b_g are the weight and bias for the gate layer.

The output of the fusion layer is then used to compute the final imputation values:

$$\text{Final Imp} = \beta \times x_{\text{Attention}} + (1 - \beta) \times x_{\text{RNN}}$$

Here, β represents the gating output, and $x_{Attention}$ and x_{RNN} are the feature imputations from the attention and recurrent outputs, respectively.

3.3 Loss Function and Training

In my work, I adopt the Mean Absolute Error (MAE) as the loss function, which is defined as:

$$\mathcal{L}_{\text{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|Y_i - Y_i'| \tag{2}$$

where Y represents the true data, Y' - imputed data generated by the model, and N denotes the total number of observed entries.

However, as described in BRITS [1], optimizing the loss function of the final imputation alone leads to slow convergence. In practice, I found that training with only the fusion loss component $MAE(Y, Y')$ corroborated the findings from BRITS about the model converging slowly. Therefore, instead of focusing solely on the final imputation output, I accumulate the losses from the recurrent imputations $\hat{X}$ and the cross-feature imputations Z alongside the final fused imputations Y'. This leads to an aggregated loss function:

$$\mathcal{L}_{\text{total}} = \mathcal{L}(Y, \hat{X}) + \mathcal{L}(Y, Z) + \mathcal{L}(Y, Y') \tag{3}$$

In the training setup for my model, I employ the Adam optimizer configured with parameters set to a learning rate of 10^{-3} and a weight decay of 5×10^{-5}.

Weight decay is a regularization technique used to prevent overfitting by discouraging large weights in the model's parameters. It is implemented by adding a penalty term to the loss function, which is proportional to the square of the magnitude of the weights:

$$\text{Regularized Loss} = \text{Original Loss} + \lambda \sum_i w_i^2$$

Here, λ represents the weight decay coefficient, and w_i denotes the coefficients or weights of the model. This addition helps to ensure that the model does not overly fit the training data, thus enhancing its ability to generalize to unseen data.

To further refine the training process, a learning rate scheduler is employed. This scheduler adjusts the learning rate at predefined intervals, reducing the learning rate by a factor of γ every 7 epochs. The adjustment is defined by the formula:

$$\text{Adjusted Learning Rate} = \text{Initial Rate} \times \gamma^{\left\lfloor \frac{n}{\text{step size}} \right\rfloor}$$

In my configuration, γ is set to 0.1, indicating a 90% reduction every 7 epochs. This step decay strategy helps in achieving finer convergence of the optimizer to the optimal set of parameters.

All parameters, including the learning rate, weight decay, and scheduler configuration, were chosen empirically based on extensive experiments.

3.4 Dataset and Model Evaluation

In this study, I utilized the DACMI [5] dataset, derived from the MIMIC-III electronic health records [3], specifically curated for the Data Analytics Challenge on Missing Data Imputation (DACMI). This challenge is recognized for benchmarking advanced imputation methods using complex real-world data. This dataset initially features approximately 7 percent of missing data across 13 different blood laboratory test results for 8,267 patients admitted to intensive care units (ICUs).

Imputation accuracy is assessed using the normalized root mean square deviation (NRMSD) illustrated in Eq 4, where Y represents actual values, $\hat{Y}$ - imputes values, p is a patient index, t - time step index, d - variable index. N_d stands for the overall number of imputed values for a particular variable within all patients and time points. Basically, lower values indicate less residual variance and therefore better performance.

$$NRMSD(d) = \sqrt{\frac{\sum_{p,t}\left(\frac{|Y_{p,t,d} - \hat{Y}_{p,t,d}|}{max(Y_{p,d}) - min(Y_{p,d})}\right)^2}{N_d}} \tag{4}$$

3.5 Algorithm for Generating Missing Values

To ensure the robustness of my developed method, I extended the testing to different missing value rates and types beyond the initial 7 percent. I adopted the approach outlined in the paper [4], following $Algorithm1$ to generate missing values of three missing value types with varying missing rates r.

In MCAR simulation, missing values were generated by randomly selecting $r\%$ of the cells in the dataset and deleting their values. This simulation reflects a scenario where missingness has no systematic pattern and can occur in any data point.

For MNAR generation, all values falling below the $(r/2)$th percentile and above the $[100 - (r/2)]$th percentile were removed. This type of simulation is used to replicate situations where

missingness is related to the distribution of the variables themselves.

MAR missing values were simulated through an iterative process. During each iteration, the same percentiles were obtained on three randomly selected observed variables to identify the indices corresponding to low and high values. These indices are used to remove values in the variable set, excluding these three observed variables. The process was repeated until the cumulative missing rate of $r\%$ was achieved. This simulation represents scenarios where the likelihood of missing data depends on the values of other observed variables, introducing a systematic but indirect pattern of missingness.

These simulations are designed to emulate the complexities and real-world scenarios associated with missing data, thereby providing a testing ground for assessing the efficacy of the developed imputation method.

4 Experiments and Results

The training of the model was conducted on a dataset split into 80% for training and 20% for validation. In my experiments, I employed a batch size of 32.

The results of the imputation on the DACMI dataset are presented in tabular format in Table 1. This table includes the NRMSD metric for each variable.

In evaluating the performance of the developed model, it's important to compare it against existing models. Reference [4] details the performance of CATSI and BRITS when applied to the DACMI dataset, particularly under conditions where 5% of the data is missing of MCAR type. To ensure a fair comparison, I replicated this scenario by employing the missing data generation algorithms described in Section 3.5 of my report, introducing 5% MCAR missingness into the DACMI dataset, thereby creating an equivalent testing environment. The comparative imputation results are summarized in Table 2.

An analysis of the table suggests that the performance of the developed HIIN model showcases strong imputation capabilities. Its average error score is markedly lower than that of BRITS, which emphasizes the added value of integrating the self-attention mechanism into the imputation process. However, it is worth noting that the HIIN model does not quite surpass the CATSI model's performance. This may be partly attributed to constraints in computational resources, CATSI's architecture benefits from a more profound network of hidden layers. Additionally, CATSI's incorporation of a context vector, which encapsulates statistical summaries of the data, may provide the recurrent components of the model with a more informative basis from which to make imputations.

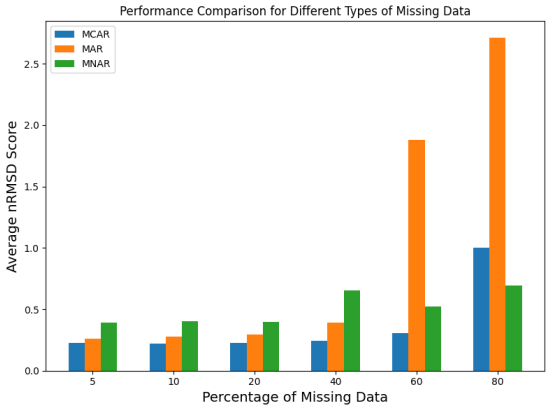

Figure 5: Comparative performance of the HIIN model for MCAR, MAR, and MNAR types of missing data with different missing value rates

4.1 Different Missing Value Types and Rates

To thoroughly test the robustness of the HIIN model across various scenarios, I utilized the algorithms specified in Section 3.5 to create multiple testing environments. These environments were structured with different percentages and types of missing data. Specifically, I generated datasets with 5, 10, 20, 40, 60, and 80 percent of missing values, covering three distinct types of missing data: MCAR, MAR, and MNAR types.

The performance of HIIN across these varied scenarios was recorded and the results are visually presented on the bar chart in Figure 5.

Analyzing the bar chart, several trends can be observed. For MCAR, where the missing data is independent of any factors, the model maintains a relatively stable performance with increasing percentages of missing data. An important observation is the model's superior performance in imputing 60% of missing MCAR data when compared to the CATSI model, as documented in the referenced paper [4] (the bar charts with the models' performance for three different missing value types can be found in Appendix A). However, the model's performance noticeably decreased when tested with extremely high levels of missingness, there is a pronounced spike in the

Method	PCL	PK	PLCO2	PNA	HCT	HGB	MCV	PLT	WBC	RDW	PBUN	PCRE	PGLU	Avg
HIIN	0.248	0.287	0.251	0.258	0.215	0.212	0.285	0.237	0.264	0.254	0.218	0.253	0.433	**0.263**

Table 1: Variable specific imputation error for the DACMI data set based on NRMSD scores

Method	PCL	PK	PLCO2	PNA	HCT	HGB	MCV	PLT	WBC	RDW	PBUN	PCRE	PGLU	Avg
CATSI	0.187	0.261	0.226	0.206	0.137	0.137	0.251	0.176	0.219	0.222	0.183	0.220	0.265	**0.207**
BRITS	0.191	0.269	0.219	0.202	0.145	0.146	0.395	0.351	0.269	0.431	0.264	0.293	0.293	**0.267**
HIIN	0.205	0.25	0.238	0.219	0.183	0.181	0.254	0.212	0.237	0.23	0.205	0.253	0.27	**0.226**

Table 2: Variable specific imputation error for the DACMI data set based NRMSD scores (5% of missing value of MCAR type)

NRMSD score at 80% of missing data.

The model's performance with MNAR data is particularly promising. While the overall error is higher for this type of missingness, which is expected due to the informative nature of MNAR data, the model remains robust, displaying a stable performance even at higher missing value rates. Remarkably, at 80 % of missing data, the model not only sustains its performance but appears to achieve the lowest NRMSD score, showing the best performance. When compared to the results of the BRITS model documented in reference [4] (Appendix A), the HIIN model exhibits a comparable, sometimes even superior ability to handle MNAR data.

However, the MAR type seemed to be challenging for the HIIN model. The big spike in error for data with 60 and higher percentage of missing values may be due to a reduction of available information, which impacts the model's ability to accurately indicate the dependencies between observed values. When compared with the results from [4] (Appendix A), it is apparent that HIIN does not match the performance of the BRITS and CATSI models for MAR type.

5 Future Work

To improve the model performance future experiments could focus on expanding the model's complexity. This could involve increasing not only the number of hidden layers and units within those layers but also experimenting with the quantity and dimensionality of attention heads in the self-attention component. Such adjustments would likely allow the HIIN model to model more complex data relationships that are currently missed. However, it is crucial to balance model complexity with the risk of overfitting, ensuring that improvements in imputation accuracy generalize well to unseen data.

6 Conclusion

In this project, the primary aim was to address the prevalent issue of missing data in multivariate time series with an emphasis on electronic health records. The developed model, named Hybrid Integrative Imputation Network (HIIN), integrates elements from recurrent neural networks (RNNs) and self-attention mechanisms to predict missing data points effectively.

In the scope of this project, the HIIN model underwent extensive testing against various types and rates of missing data. It demonstrated proficiency in handling MCAR and MNAR scenarios, reflecting a stable performance even at high rates of missing data. For the MNAR type, the HIIN model exhibited robustness, maintaining consistent performance with increasing rates of missing data and displaying promising results at 80% of missingness. This suggests that the model is capable of identifying patterns where missingness is not random and may depend on unobserved factors. However, the model's performance was less consistent with the MAR type, showing increased error rates at higher percentages of missing data. This might be due to the study's constraints, particularly regarding computational resources that impacted the depth and breadth of the model's architecture. Future work could focus on expanding the network's complexity and enhancing the self-attention component.

In conclusion, I believe that the outcomes of this project can contribute to the ongoing efforts to improve data imputation techniques.

A Appendix.

References

[1] W. Cao et al. "BRITS: Bidirectional Recurrent Imputation for Time Series". In:

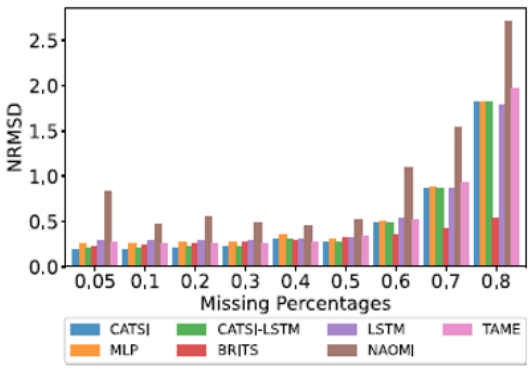

Figure 6: Comparing NRMSD scores of seven deep learning methods for imputing missing values in time series data. The MCAR type is used at varying missing rates on the DACMI dataset. The chart is taken from [4]

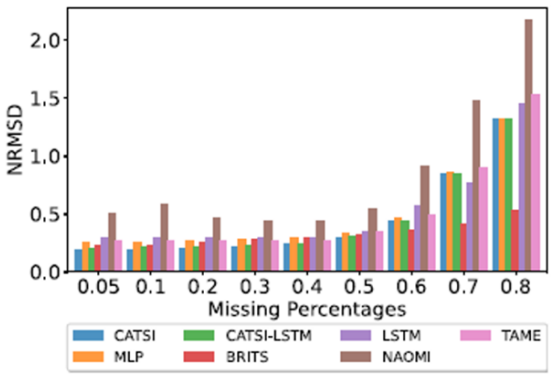

Figure 7: NRMSD scores of seven deep imputation methods. The MAR type is used at varying missing rates on the DACMI dataset. The chart is taken from [4]

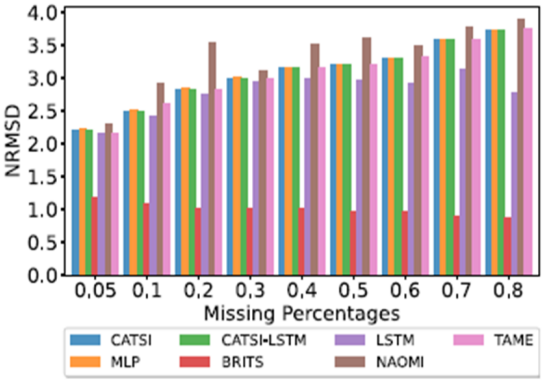

Figure 8: NRMSD scores of seven deep imputation methods. The MNAR type is used at varying missing rates on the DACMI dataset. The chart is taken from [4]

Advances in Neural Information Processing Systems. Vol. 31. 2018.

[2] Wenjie Du, David Cote, and Yan Liu. *SAITS: Self-Attention-based Imputation for Time Series*. https://doi.org/10.48550/arXiv.2202.08516. 2023.

[3] A. E. Johnson et al. "MIMIC-III, a Freely Accessible Critical Care Database". In: *Scientific Data* 3.1 (2016), p. 160035. DOI: 10.1038/sdata.2016.35.

[4] Maksims Kazijevs and Md Dilshadur Samad. "Deep Imputation of Missing Values in Time Series Health Data: A Review with Benchmarking". In: *Journal of Biomedical Informatics* 144 (2023), p. 104440. DOI: 10.1016/j.jbi.2023.104440. URL: https://www.sciencedirect.com/science/article/pii/S1532046423001612.

[5] Luoyuan Lab. *MIMIC Imputation Dataset*. https://github.com/luoyuanlab/mimic_imputation.

[6] K. Yin, L. Feng, and W.K. Cheung. "Context-Aware Time Series Imputation for Multianalyte Clinical Data". In: *Journal of Healthcare Informatics Research* 4 (2020), pp. 411–426. DOI: 10.1007/s41666-020-00075-3. URL: https://doi.org/10.1007/s41666-020-00075-3.