A DEEP LEARNING MODEL TO IMPUTE MISSING DATA IN TIME SERIES

Wenjie Du

A thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science
Concordia University
Montréal, Québec, Canada

November 2021

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Wenjie Du

Entitled: A Deep Learning Model to Impute Missing Data in Time Series

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. D. Qiu

_____ Examiner
Dr. R. Glitho (CIISE)

_____ Examiner
Dr. D. Qiu

_____ Thesis Supervisor(s)
Dr. Y. Liu

_____ Thesis Supervisor(s)

Approved by _____
Dr. Yousef Shayan, Chair
Department of Electrical and Computer Engineering

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Date _____

Abstract

A Deep Learning Model to Impute Missing Data in Time Series

Wenjie Du

Missing data in time series is a pervasive problem that puts obstacles in the way of advanced analysis. A popular solution is imputation, where the fundamental challenge is to determine what values should be filled in. In this thesis, we study imputing missing data in time series with deep learning. We first present a concrete case in telecommunication domain, where we use machine learning models to handle missing data and forecast Imminent Loss of Signal (ILOS) that is going to occur in optical networks. Subsequently, we further propose a novel model, called SAITS (Self-Attention-based Imputation for Time Series), to impute missing values in multivariate time series. SAITS uses a joint-optimization training approach to learn missing values from a weighted combination of two diagonally-masked self-attention (DMSA) blocks. DMSA explicitly captures both the temporal dependencies and feature correlations between time steps, which improves imputation accuracy and training speed. The **contributions** of this thesis are **1)** In the motivation case, we develop a deep learning methodology based on BRITS to learn a good representation from data with massive missing values and forecast 3% ILOS with 65% precision; **2)** We design a joint-optimization training approach to train self-attention models on the imputation task. Trained by this approach, Transformer achieves up to 25% smaller mean absolute error than BRITS; **3)** We propose SAITS, a new imputation model based on self-attention, specifically for the time-series imputation task. Compared to the state-of-the-art (SOTA) model BRITS, SAITS obtains 12%∼38% smaller mean absolute error and 2.0∼2.6 times faster training speed. Experimental results demonstrate that SAITS achieves the new SOTA position on the time-series imputation task.

Acknowledgments

I am enormously grateful to be supervised by Dr. Yan Liu, whose academic guidance and advice are invaluable to me. I also appreciate my internship mentor Dr. David Côté at Ciena, whose technical insights provided us fruitful discussions.

I would like to thank my parents for their unconditional love and support in my life. They are my heroes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multivariate time-series data is ubiquitous in many application domains, for instance, transportation [3, 4], economics [5, 6], healthcare [7, 8, 9], and meteorology [10, 11, 12]. Due to all kinds of reasons, including failure of collection sensors, communication error, and unexpected malfunction, missing values are common to see in time series. They impair the interpretability of data and pose challenges for advanced analysis and downstream applications, such as classification and forecasting.

Traditional missing value processing methods fall into two categories. One is deletion, which removes samples or features that are partially observed. However, deletion makes data incomplete and can yield biased parameter estimates [13]. The other one is data imputation that estimates missing data from observed values [14]. The problem of imputation is what values should be filled in. Amounts of prior work are proposed to solve this problem with statistics and machine learning methods [4, 15, 16, 17, 18, 19]. However, most of them require strong assumptions on missing data [2]. Recently, much literature utilizes deep learning to solve this imputation problem and achieves state-of-the-art (SOTA) results. Non-time series imputation models, which are not in the scope of this thesis, can be referred to [20, 21, 22, 23, 24, 25]. We give a more detailed description of time-series imputation models in Section 2, including [2, 26, 27, 28, 29, 30, 31, 32, 33, 34].

In Chapter 3, we start from a motivation case in telecommunication to show the importance of handling missing values. In the motivation case, our goal is to forecast loss of signal (LOS). LOS represents a significant cost for operators of optical networks. Forecasting LOS accurately and taking maintenance measures in advance

can prevent network service outrages. However, high missing rate (>70%) in collected data stops us from the goal of forecasting. In our work, we leverage BRITS [2], a state-of-the-art (SOTA) deep learning model on the time series imputation task, to handle missing data firstly and then do LOS forecasting. By studying large sets of real-world Performance Monitoring (PM) data collected from six international optical networks, we find that it is possible to forecast LOS events with good precision 1-7 days before they occur, albeit at relatively low recall, with supervised learning. Our study covers twelve facility types, including 100G lines and ETH10G clients. We show that the precision for a given network improves when training on multiple networks simultaneously relative to training on an individual network. Furthermore, we show that it is possible to forecast LOS from all facility types and all networks with a single model, whereas fine-tuning for a particular facility or network only brings modest improvements. Hence our machine learning models remain effective for optical networks previously unknown to the model, which makes them usable for commercial applications.

In Chapter 4, we propose a self-attention-based model called SAITS to learn missing values by a joint-optimization training approach of imputation and reconstruction. This training approach consists of two learning tasks corresponding to an imputation loss and a reconstruction loss. The self-attention mechanism is now widely applied, whereas its application on time-series imputation is still limited. Previous SOTA time-series imputation models are mostly based on recurrent neural networks (RNN), such as [2, 26, 27, 28, 29]. Among them, [2, 26, 27, 28] are autoregressive models that are highly susceptible to compounding error [29]. Although [29] is not autoregressive, the multi-resolution imputation algorithm it proposed consists of a loop, which can greatly slow the imputation speed. The self-attention mechanism, which overcomes RNNs' drawbacks of slow speed and memory constraints and is non-autoregressive, can avoid compounding error and be helpful to achieve better imputation performance and speed. In this work, we make the following contributions:

I. We develop a deep learning methodology based on BRITS in the motivation case, which can learn a good representation from data with missing values and use it to forecast 3% ILOS 1-7 days before they occur in commercial optical networks with 65% precision.

II. We design a joint-optimization training approach of imputation and reconstruction for self-attention models to perform missing value imputation for multivariate time series. Transformer trained with this approach achieves up to 25% smaller mean absolute error than BRITS.

III. We design a novel model called SAITS, which consists of a weighted combination of two diagonally-masked self-attention (DMSA) blocks. DMSA mechanism emancipates SAITS from RNN and enables SAITS to capture temporal dependencies and feature correlations between time steps explicitly. Compared to BRITS [2] on three real-world public datasets, SATIS demonstrates 12%∼38% improvement in imputation performance (in terms of mean absolute error) and 2.0∼2.6 times faster training speed. Furthermore, experimental results show that SAITS outperforms Transformer and achieves the new SOTA position.

We start by reviewing related work in Chapter 2, give a motivation case from telecommunication in Chapter 3, introduce our joint-optimization training approach and the SAITS model in Chapter 4. Experiments, limitations and conclusions are presented in Chapter 5, Chapter 6 and 7, respectively.

Chapter 2

Related Work

We review prior related work of time-series imputation in the following four categories:

RNN-based Che et al. [35] propose GRU-D, a gated recurrent unit (GRU) variant, to handle missing data in time series classification problems. The concept of time decay on the last observation is firstly proposed by [35] and continues to be used in [2, 26, 27, 28]. M-RNN [26] and BRITS [2] impute missing values according to hidden states from bidirectional RNN. However, M-RNN treats missing values as constants, while BRITS treats missing values as variables of the RNN graph. Furthermore, BRITS takes correlations among features into consideration while M-RNN does not.

GAN-based Models in [27, 28, 29] are also RNN-based. However, considering they adopt the generative adversarial network (GAN) structure, we list them separately as GAN-based. Luo et al. [27] propose GRUI (GRU for Imputation) to model temporal information of incomplete time series. Both the generator and the discriminator in their GAN model are based on GRUI. Moreover, based on [27], Luo et al. [28] propose E^2GAN, which is an end-to-end method, comparing to the method in [27] having two stages. E^2GAN adopts an auto-encoder based on GRUI to form its generator to ease the difficulty of model training and improve imputation performance. Liu et al. [29] propose a non-autoregressive model called NAOMI for spatiotemporal sequence imputation, which consists of a bidirectional encoder and a multiresolution decoder. NAOMI is further enhanced by adversarial training.

VAE-based Inspired by GPPVAE [36] and the non-time-series imputation model HI-VAE [23], Fortuin et al. [32] propose GP-VAE, a variational auto-encoder (VAE) architecture for time series imputation with a Gaussian process (GP) prior in the latent space. The GP-prior is used to help embed the data into a smoother and more explainable representation. L-VAE [33] uses an additive multi-output GP-prior to accommodate auxiliary covariate information other than time. To support sparse GP approximations based on inducing points and handle missing values in spatiotemporal datasets, Ashman et al. [34] propose SGP-VAE.

Self-Attention-based Ma et al. [30] apply self-attention jointly from three dimensions (time, location, and measurement) to impute missing values in geo-tagged data, namely spatiotemporal datasets. Bansal et al. [31] propose DeepMVI for missing value imputation in multidimensional time-series data. Their model includes a Transformer with a convolutional window feature and a kernel regression. Related work of self-attention-based models for time-series imputation is very limited. There are some non-time series imputation models based on self-attention, such as AimNet [24] and MAIN [25].

Chapter 3

A Motivation Case

3.1 Problem Statement

Optical networks form the backbone of the global information and communication infrastructure, which supports a huge ecosystem of technologies and services. Thus, the high availability of the network infrastructure is critical both economically and socially. Although today's networks are remarkably reliable, Loss of Signal (LOS) still occurs. When it happens, either (1) the network protects itself automatically or (2) the LOS propagates to cause signal interruption. This latter situation degrades service quality for end-users, incurs labor and maintenance costs for the network operator and can even have catastrophic effects. For instance, in 2016, Southwest Airline canceled more than 2,000 flights and lost tens of millions of dollars due to a massive network system failure caused by a faulty router [37, 38]. Predicting LOS events before they occur enables proactive actions to prevent such network outages.

Autonomous networking is the vision pursued by world-leading network technology companies, including the Adaptive Network [39] proposed by Ciena, the Autonomous Driving Network proposed [40] by Huawei, the Digital Network Architecture proposed [41] by Cisco, the self-driving network proposed by Juniper [42], and the Zero-Touch Network [43] proposed by Ericsson. This activity is largely focused on automating reactive processes, but a few applications such as Ciena's Network Health Predictor (NHP) [44] and Juniper's HealthBot [45] can also forecast network events before they occur to strengthen service reliability.

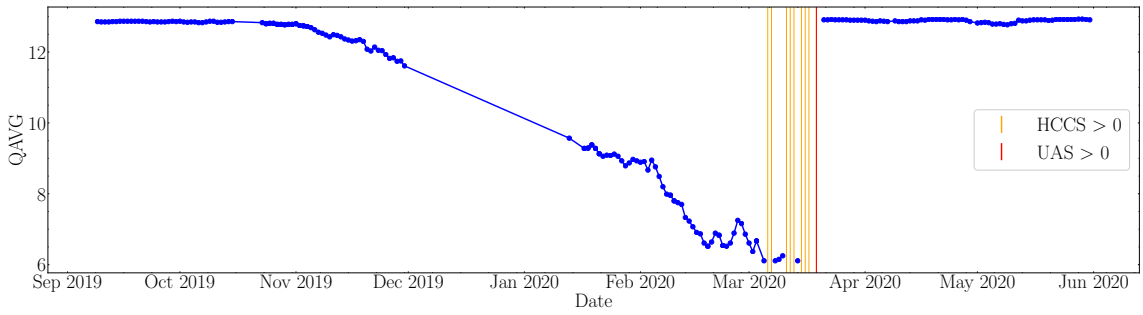
In optical networks, service-impacting LOS often comes from intrinsically unpredictable fiber cuts. But other root-causes like equipment aging, loose connectors, or system mis-configuration can create predictable LOS events. Hence it is possible to forecast some LOS in advance, though not all of them. Figure 1 shows one compelling example of predictable LOS reported from the 100G line receiver port of a Ciena 6500 device in a production network. In the left-most part of the graph, the signal quality is good (high QAVG) and stable (low QSTDEV). Here Q is a factor representing the signal quality. QAVG is the average score of Q (reflecting the signal quality) and QSTDEV is the standard deviation of Q (reflecting the signal stability). Then, over a period of several months, the signal quality becomes incrementally unstable and degraded. In March 2020, orange lines indicate that the port experiences transient LOS for a few seconds. Finally, a red vertical line indicates an outage where the LOS lasts over an hour before the issue gets fixed and the ports re-start reporting good and stable signal quality again. From these findings, we speculate that machine learning (ML) models may be able to forecast Imminent Loss of Signal (ILOS) from receiver ports in multi-vendor packet-optical networks that report similar data.

Recently, ML has been applied in solving telecommunication problems. The exploratory experiments by Côté in [46] validated the speculation that the random forest model is applicable to abnormal element detection in optical networks. However, it used laboratory data and it lacked the ability to predicate ILOS for 1 to 7 days in the future. In this work, we focus on six sets of real-world data collected from large commercial optical networks geographically distributed around the world.

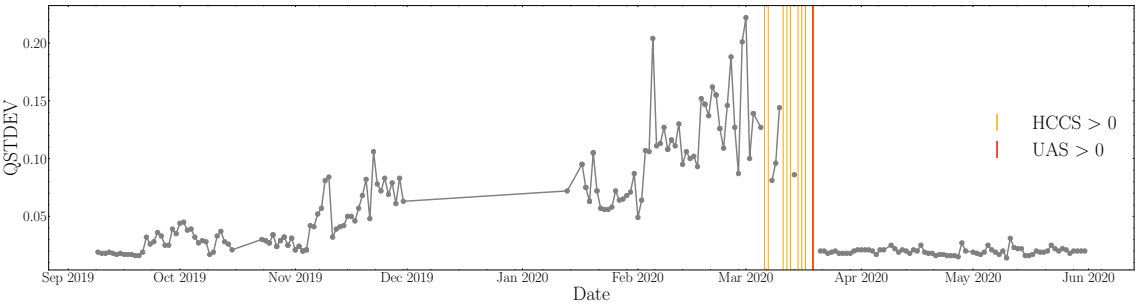
Other researchers have reported success in using ML to reduce nonlinear phase noise [47, 48], estimate quality of transmission [49, 50, 51], detect anomalies [52, 53, 54], forecast equipment failures [55, 56, 57, 58, 59], and enable AI-based routing [60] or self-optimizing networks [61, 62]. Building on this valuable work, we aim to develop a general engineering framework that can handle telecommunication data for production-grade applications. This real-world data comes with practical challenges for ML best practices today, notably: (1) it is unlabeled, (2) it has imperfect data-quality, and (3) it is not readily available in large sets.

To overcome these challenges, we have developed a framework that can:

I. **Encode domain knowledge**



(a) Time series trend of PM QAVG



(b) Time series trend of PM QSTDEV

Figure 1: A typical example of predictable Imminent Loss of Signal (ILOS) from a production network. Figures 1(a) and 1(b) show signal quality (QAVG) and stability (QSTDEV), respectively, over a time period ranging from September 2019 to May 2020. Here Q is a factor representing the signal quality. QAVG is the average value of Q on that day and QSTDEV is the standard deviation of Q value on that day. Orange vertical lines indicate days with transient LOS occurring during a few seconds (HCCS $\in [1, 60]$ sec). The red vertical line indicates an outage where the LOS lasted over an hour (UAS > 3600 sec). This LOS event could have been predicted beforehand given these PMs.

Automatic rules developed by experts can identify problematic network elements (NE) from performance monitoring (PM) data after the fact. We use these rules to process historical datasets and label the status of each NE over time. Then we use labeled data samples to train supervised ML models that predict good → bad transitions before they occur.

II. **Handle imperfect data quality**

Our input data is not always continuous and homogeneous. While not ideal, this is beyond our control and we have to be pragmatic about it. Notably, we can miss data because of zero-suppression, because of error conditions, or because the data was never collected. As a result, our overall matrix of features has a sparse structure with a high missing rate (>70%). Traditional imputation methods do not perform well in this situation, which led us to develop sophisticated methods to learn from the data present without being biased by the data missing.

III. **Combine datasets from diverse sources**

Deep learning methods generally benefit from the largest possible data samples, especially for solving needle-in-the-haystack problems like predicting optical network outages. Nevertheless, Network Management Systems in production typically keep data for only a few days or weeks. Hence we have collected data multiple times from multiple networks to accumulate big datasets. However, commercial optical networks are complex and diverse, hence we put in place a substantial machinery to combine data from all sources into a single *mega-dataset*.

Equipped with the above, we have been able to train supervised ML models that can predict ILOS 1-7 days before they occur in commercial optical networks. Furthermore, we show that this task can conveniently be accomplished with a single ML model that covers all Layer 1 and 2 devices across a full packet-optical network.

In Section 3.2, we review the prior work related to this topic. In Section 3.3, we describe our data pre-processing and ML methodology. In Section 3.4, we describe the various ML experiments that we performed during our analysis, and in the same section, we present our final results. Finally, in Section 3.5, we summarize our main conclusions and give an outlook on the next steps.

3.2 Background

3.2.1 Handling Missing Values

Missing values are ubiquitous in time series datasets because of communication interruption or sensor malfunction during data collection, especially for industry data. Imputing missing values with mean, median or fixed values such as 0 are basic methods to process missing data. Most of such simple imputation methods impose strong assumptions on missing data. For example, zero imputation presumes missing values are zero suppression. However, a high missing rate makes it very challenging for these conventional imputation methods to be effective because a high missing rate results in diverse missing patterns, making these assumptions hard to meet [2].

In recent years, some researchers attempted to solve this problem of missing values with ML. XGBoost proposed by Chen et al. [63] can directly process input with missing values. As a tree-based algorithm, XGBoost has a learning mechanism called sparsity-aware split finding to assign a default direction to each tree. When XGBoost encounters missing values, the branch takes the default direction to continue the decision path. Marek Smieja et al. [64] model the uncertainty on missing attributes by probability density functions using the Gaussian mixture model (GMM), then replace the typical neuron's response in the first hidden layer by its expected value. Recurrent neural networks (RNN) are always popular in the time series topic, and this missing value problem is not an exception. Che et al. [35] propose GRU-D, which introduces a decay mechanism into a GRU so that the influence of input variables is decayed over time if the variable has been missing for a while. Luo et al. [27] propose the Gated Recurrent Unit for data Imputation (GRUI) to model the incomplete time series and integrate the GRUI into a generative adversarial network (GAN) to treat imputation as a data generating task. BRITS (Bidirectional Recurrent Imputation for Time Series) proposed by Cao et al. [2] directly learns missing values in a bidirectional recurrent dynamical model, where imputed values are treated as variables of RNN. At the time of writing, BRITS achieves the state-of-the-art (SOTA) performance on the PhysioNet Challenge 2012 dataset [7], a time-series health-care dataset with 78% of the values missing [2]. Hence BRITS is the neural network model we apply to solve missing data problem in this work.

3.2.2 Detecting and Forecasting Anomalies in Optical Networks

Recently, ML has been utilized for anomaly detection and forecasting in optical networks. To detect anomaly, Chen et al. [53] employ an unsupervised density-based clustering algorithm to analyze monitoring data patterns and then utilize a self-taught mechanism to transfer learned patterns to a supervised data regression and classification module. Rafique et al. [54] build a proactive fault detection (PFD) engine for autonomous anomaly detection. The PFD engine firstly use the generalized extreme studentized deviate (ESD) test to identify all potential faults and then send them into a neural network to classify true anomalies. Shahkarami et al. [52] define a detection framework for Bit Error Rate anomaly and propose a ML method to discriminate different sources of soft failure. To predict the risk of an equipment failure, Zhilong Wang et al. [57] propose a performance monitoring and failure prediction method based on support vector machine (SVM) and double exponential smoothing (DES). They use DES to converge values of features indicating the status of equipment and then predict future values sent into the SVM model to classify if the facility will fail in the near future. With regards to labeling, they apply a threshold on the indicator 'Unusable Time' and assume equipment has failed if this indicator's value exceeds the threshold. The ORCHESTRA network presented in [58] is developed to predict, detect, and diagnose health issues automatically, for example, soft failures like quality of transmission (QoT) degradation. Self-Optimizing Optical Networks (SOON) proposed by Zhao et al. [61] is a network architecture based on software-defined networking with ML for applications with tidal traffic forecasting, alarm prediction, and anomaly action detection. Due to inferior data quality, Zhuang et al. [56] leverage a GAN to augment their dataset and train a neural network on the augmented data to predict alarm in optical networks.

The above methods do not specifically focus on handling missing data, the small size of data samples, or combining imputation with ML. For example, in Zhilong Wang's experiments [57], their dataset has only 15 features and 14,080 samples without missing data, and sample classes also get balanced intentionally. There is even no need to distinguish facility types in their data. In the research of Yan et al. [55], they demonstrate a dirty-data-based alarm prediction method for their specific network structure based on SOON. Data is collected from commercial large-scale optical

networks and has characteristics similar to the data used here, such as imbalanced classes and missing data. However, their work is based on the SOON architecture, not generally applicable to other networks. Additionally, they work on short-term prediction, which takes in data collected from the past 3 hours to predict alarms in the future 15 minutes. Such short-term prediction does not leave enough time for operators to respond and perform maintenance.

3.2.3 Applying Transfer Learning to Optical Networking

Transfer learning transfers the learned knowledge from the source domain to the target domain. It emerges to tackle the problem of building deep learning models in the domain of interest by leveraging the sufficient data in another related domain [65]. In the work of Yu et al. [66], transfer learning is adopted to enable applying their neural network models to different optical systems with only a small size of new training samples. Xia et al. [67] propose a deep neural network to estimate optical-signal-to-noise ratio (OSNR). They leverage transfer learning to remodeling if system parameters change. Their method is able to reduce the training time by a factor of four and requires only one-fifth the training set size without any performance loss. To detect new cyber-attack behaviors in a network, Zhao et al. [68] propose a transfer learning-enhanced approach called CeHTL to find out relations between new attacks and known attacks. Azzimonti et al. [69] utilize Gaussian process to estimate bit error rate in an unestablished lightpath. In their work, transfer learning is adopted to further optimize the accuracy on small-sized datasets.

3.3 The Learning Method

Our methodology consists of three aspects: (1) data pre-processing and labeling, (2) core modeling with missing data handling embedded into the learning model, and (3) transfer learning across networks. For data pre-processing, we produce port-level time-series datasets to represent the status of one port on one day. The labels of data samples are generated by our rule-based method. For modeling, we formalize the ILOS prediction problem into a binary classification modeling task solved with supervised ML. Furthermore, we compare classifier algorithms with three different approaches to missing data: random forest [70], XGBoost [63] and BRITS [2].

For transfer learning, we seek to improve the learning performance on small datasets that lack sufficient training data by leveraging a well trained model with a similar feature space. Our solution is to combine all datasets from six networks into one mega-dataset. The mega-dataset enables these networks to share feature space across all datasets. Models pre-trained on the mega-dataset learn the general feature representation of ILOS signatures from all datasets, akin to knowledge sharing. Our experiments observe that such general ILOS representation improves the performance up to 36.4% as compared to the deep learning model on networks with small datasets and up to 75.8% as compared to XGBoost.

3.3.1 Data Pre-processing

In our data pre-processing methodology, we first introduce properties of our collected data and then illustrate our six pre-processing steps in details.

Dataset Description

The datasets used for this research consist of daily-binned Performance Monitoring metrics (PMs) collected from equipment in production network environments, spanning layers 1 and 2. Each device typically has multiple ports, where each port reports PMs for some facility, where, in turn, a facility is a low-level object mapping to a physical or virtual component or service of the Optical Transport Network (OTN) such as e.g. Ethernet (ETH) or Optical Transport Module (OTM).

Our analysis uses every PM reported by the port as a feature input to the model. These PMs report a wide variety of information related to the signal quality, describing e.g. the number of input and output frames, number of code violations, optical power, delay measurements, number of errored seconds, failure counts, number of forward error corrections, signal quality ("Q") and other metrics related to low-level errors in the signal.

An overview of our datasets is shown in Table 1, where datasets for Network1 to Network6 are sorted by values of positive sample rate in incremental order from left to right. The mega-dataset is the one combining all 6 datasets for transfer learning. It is worth noting that classes in our datasets are imbalanced. The overall positive sample rate in the mega-dataset is around 10%. In Network1, the positive sample rate

is less than 5%. A small number of positive samples is consistent with the observation mentioned above that equipment failures are rare in optical networks.

As summarized in Table 1, we consider twelve facility types and nine protocols to characterize the state and performance of the entire network, from physical equipment (layer-0) to optical (layer-1) and Ethernet (layer-2) signals.

Table 1: Details of the six network datasets used in this work, sorted by positive sample rate in ascending order from left to right. Each dataset includes all 12 facility types of layer-1 and layer-2 ports. The right-most column summarized the "mega-dataset", which is a combination of all six network datasets. The missing rate of the mega-dataset is the largest due to the fact that some networks have features that do not exist in others, resulting in nearly empty columns when merging the network datasets into the mega-dataset.

	Network1	Network2	Network3	Network4	Network5	Network6	Mega-dataset
Number of protocol types	6	9	8	6	8	8	9
Time range (days)	139	222	204	299	301	208	1,373
Number of ports	3,000	19,000	5,000	35,000	16,000	16,000	94,000
Number of samples	394,158	3,540,472	762,268	6,140,585	3,140,132	2,478,716	16,456,331
Number of features	76	83	113	72	115	91	125
Missing rate	75.2%	72.4%	79.2%	70.1%	77.6%	79.8%	82.2%
Positive sample rate	4.3%	5.7%	5.9%	8.6%	15.8%	17.0%	10.4%

Pre-processing Steps

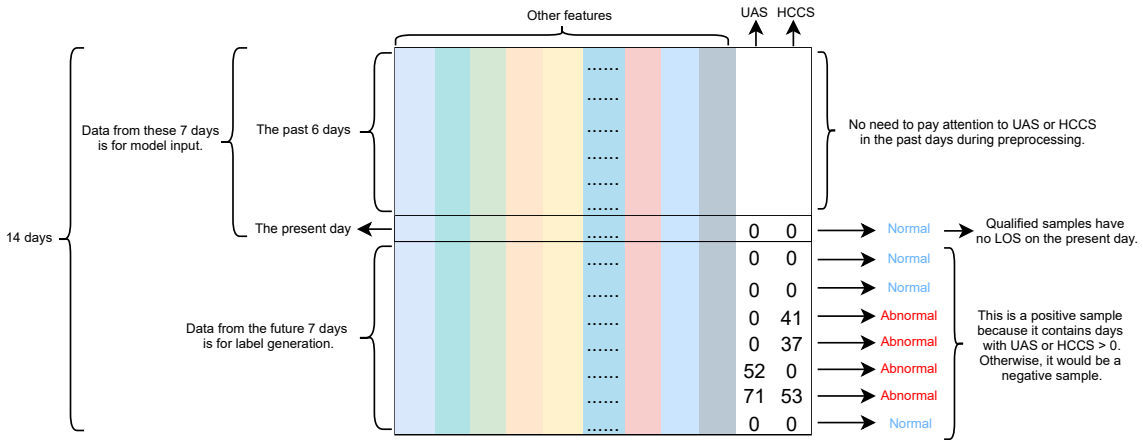


Figure 2: Overview of a qualified complete sample and labeling method.

There are six steps to pre-process our data, listed as follows:

1. Re-organize samples to port-level. Considering that our experiments need port-level datasets, raw data needs to be re-organized into port-level. Facility-level

samples are grouped by their port IDs and the collection timestamps. This means these facility samples are collected from the same port on the same day. These facility-level samples are merged into one line to compose port-level samples. In the merging operation, only maximum values are kept if one feature has multiple values because of more than one facility in the port. Facility types are converted into one-hot encoding. As a result, each sample in the datasets can represent the status of one port on one day.

2. Produce time-series data for deep learning. To generate time-series datasets for deep learning, samples of each port are sorted by collection timestamps firstly. Considering that the collection process may be interrupted for different reasons, missing days are first filled, with all PM values as NaN, so that all samples of one port are time-continuous without interruption. As a result, a 14-day window slides on the chronological data of each port to produce time-series samples. Each produced sample contains data collected from one port in 14 consecutive days. The generated samples contain some days that are filled artificially as NaN.

Each sample is of a 14-day span that is further divided into three parts, namely: (1) *the present day* as the seventh record; (2) *the past six days* as the first six records; (3) *future seven days* as the last seven records. The first seven days of all the samples are used for model training, while the future days of all the samples are used for labeling. The labels indicate whether the port encounters an ILOS on each day of the last seven days. The model learns the features in the first seven days about the status of the port and then predicts if an ILOS may occur on the port in the next 7 days.

3. Generate labels. We rely on data labels to interpret PM values into binary category positive (with ILOS) or negative (without ILOS), which is necessary for status analysis of facilities or ports. However, accurate labels are difficult to obtain in large telecommunication network datasets because of the specialized domain expertise that is required [46]. In this work, we have used rules provided by optical network experts to label historical data automatically. The labeling logic uses the fact that a port is generally suffering from LOS when it experiences errors for a sufficient period of time. For our 6500 devices, this error metric is well-represented by two PMs: Unavailable Seconds (UAS) and High Correction Count Seconds (HCCS). UAS counts the number of seconds during which a facility was unavailable and unable

to perform its task, while HCCS counts the number of seconds per day where the number of errors to be corrected by "forward error correction" techniques exceeds some threshold. Indeed, for a smaller dataset for which alarm data was available, we have verified empirically that these two PMs are highly correlated with LOS alarms emitted by the devices. Thus, if a sample contains UAS or HCCS larger than zero on any of its future 7 days, we label the sample as positive, indicating there is a LOS in the future. We plot Figure 2 to further show a concrete example of how our labeling method works.

4. Filter out defective samples. Samples that meet the following conditions are filtered: (1) not carrying traffic in the present day. There are protocol features in PMs indicating whether the port is carrying traffic. If it is not, we assert this port is not working and filter such a sample; (2) having LOS in the present day may already trigger maintenance operations. Therefore predicting ILOS for such samples is a special case. To focus on the prediction of ILOS in general, we filter out the data samples with LOS in the present day; (3) no data for all of the first 7 days or future 7 days. In this extreme case with no data for all of the first and last 7 days, rather than filling in data, we filter out such samples.

5. Split into different parts. We split our datasets into 3 parts, including the training set, the validation set, and the test set. Samples in datasets are sorted according to date timestamps. Then the partition of the datasets is approximately training set 70%, validation set 10%, and test set 20%.

6. Normalize data. For neural network models depending on gradient descent, feature normalization is necessary. We apply Z-score normalization to feature vectors.

3.3.2 Model Details and Implementation

The core principle of our solution is to augment ML models with missing data handling. We examine and expand each representing model as follows. Random forest is a classical boosting tree algorithm that does not have a default mechanism for handling missing data. Hence we apply two conventional imputation methods of zero imputation and median value imputation to process missing values. XGBoost, on the other hand, can handle missing data by default. BRITS is an RNN-based deep learning algorithm. Missing values in feature vectors are located by missing masks and then get imputed by zeros. Concatenations of imputed feature vectors and missing

masks are input into BRITS to let BRITS learn missing values by itself.

Random Forest as the Baseline Model

Random forest is adopted in the work of Côté [46] to discriminate states of network elements as being 'Normal' or 'Abnormal'. For simplicity, random forest is also adopted in our methodology to observe the learning effects of our proposed models. In addition, we expand the random forest model with two conventional imputation methods to process missing values in the input data. One method uses fixed value (0) imputation and the other method adopts the median value imputation. The comparison between these two variants to the random forest model shows how selecting different traditional imputation methods can affect model performance on our datasets. Considering that random forest is not a time-series model, each sample containing 7-day sequence data (namely 7 rows) is expanded into 1 row before being input into the model.

Random forest in our experiments is the implementation in the python library scikit-learn [71]. For this tree-based model, we find that the parameter *number of trees* has the greatest impact on our results. Therefore, we search for the most appropriate parameter value in a range from 100 to 3,000 in intervals of 100 and validate each one's performance on the validation set to pick out the best.

XGBoost

XGBoost is a scalable end-to-end tree boosting system. XGBoost has a wide range of applications in ML challenges and has achieved strong results effectively. We consider XGBoost because it is a sparsity-aware algorithm [63]. As discussed in Section 3.2, XGBoost assigns missing values to the default direction to continue the decision path. Moreover, the optimal default direction is found by trying both directions in a splitting node so that XGBoost can minimize the training loss. Therefore, missing values in the input of XGBoost can just be left as NaN for XGBoost to handle by itself. The input of XGBoost also needs to be reshaped into 1 row.

In our work, the implementation of XGBoost is from the open-source package mentioned in the original paper [63]. For XGBoost, we also perform parameter searching on *number of trees* to optimize the performance.

BRITS: Bidirectional Recurrent Imputation for Time Series

BRITS model is proposed by Cao et al. [2] to handle missing values in multivariate time series data. BRITS learns missing values in its bidirectional neural network system without imposing any specific assumptions on the data generating process, which means it is generally applicable to time series imputation tasks. As a multi-task learning model, BRITS is trained on both the imputation task and the classification task. BRITS is able to make imputations for missing values and solve classification at the same time.

BRITS model structure. BRITS model is a bidirectional model. It consists of 2 RITS (Recurrent Imputation for Time Series) models. One RITS takes in data in the forward direction and the other RITS takes in data reversed, namely in the backward direction. In the training period, both models' outputs should agree with each other after the backward one's output gets reversed, or the model gets a penalty that is the consistency loss. The consistency loss is the discrepancy between imputations from 2 models, which is measured by mean absolute error (MAE).

As shown in Figure 3, each RITS model is made of 2 parts. One part of a RITS is the imputer, which consists of a LSTM [72] layer and regression layers to learn the representation of features across time and correlation between features. An imputation task matches observed feature values with imputations generated by the imputer and calculates MAE between them to generate the estimation loss. The other part of a RITS is the classifier that has a fully connected layer. This fully connected layer takes the last hidden state output from the LSTM layer in the imputer to produce *logits*. Then the *sigmoid* function converts the *logits* into classification probabilities. Errors made by the classifier result in the classification loss. Considering that BRITS consists of 2 RITS models and each model has its results for losses, imputations, and classification probabilities, the results that BRITS finally yields are averaged values of each RITS model. The total loss in BRITS is the sum of the above mentioned consistency loss, estimation loss and classification loss.

BRITS data input. Data input into BRITS consists of 4 parts: feature vectors, missing masks, time gap vectors, and classification labels. It is worth mentioning that in the original work of Cao et al. [2], they randomly masked out an extra 10% data in their datasets that was held out for comparing BRITS' imputation ability with other models in the test period. In this work, BRITS is utilized to help us handle

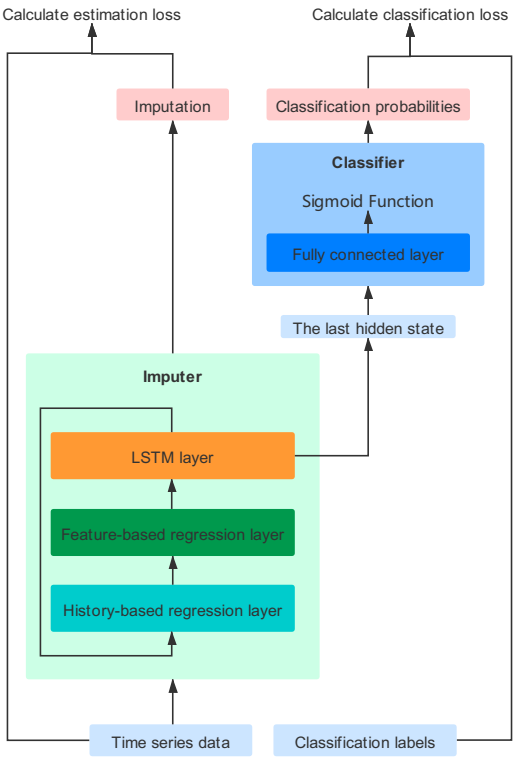


Figure 3: Structure overview of the Recurrent Imputation for Time Series (RITS) neural network model.

missing data, hence such extra-masked data do not exist in our input. Feature vectors contain all processed feature values. Missing masks indicate missing values in feature vectors. As shown in Equation 1, if x_t^d is missing that means feature d is missing at time t, then m_t^d, the mask value of feature d at time t, is set as 0. If x_t^d is observed, m_t^d is set as 1. Time gap vectors contain each feature's time gap from the last observation to the current timestamp. They are calculated from missing masks and are used to generate temporal decay factors to decay hidden states in the LSTM layer. Equation 2 illustrates the calculation of time gap vectors δ_t from the last observation to the current timestamp s_t in detail. The working principle of temporal decay factors is that if values are missing for a long time, then the last observations have less correlation with values at the current time step. As a consequence, corresponding hidden states are expected to be decayed more.

$$m_t^d = \begin{cases} 0 & x_t^d \; is \; missing \\ 1 & x_t^d \; is \; observed \end{cases} \tag{1}$$

$$\delta_t = \begin{cases} s_t - s_{t-1} + \delta^d_{t-1} & t > 1, m^d_{t-1} = 0 \\ s_t - s_{t-1} & t > 1, m^d_{t-1} = 1 \\ 0 & t = 0 \end{cases} \tag{2}$$

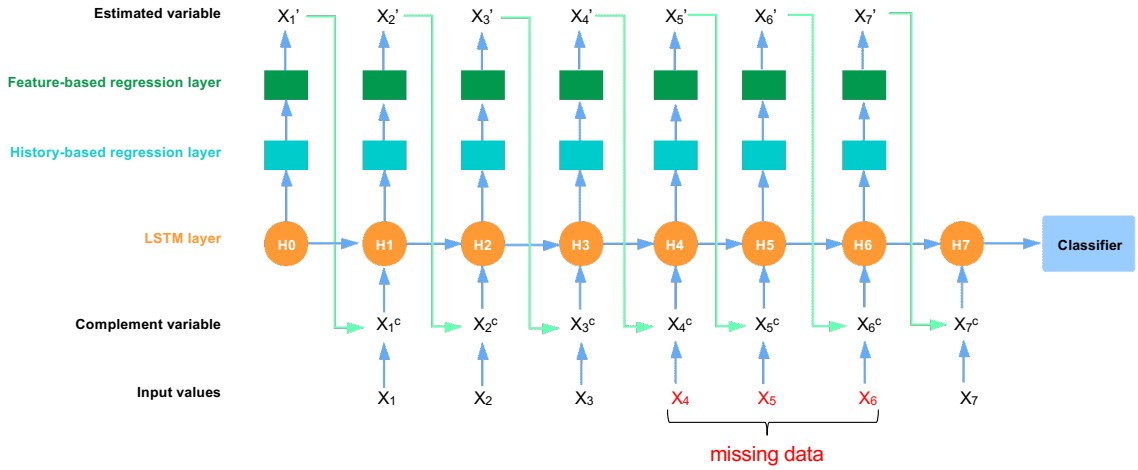


Figure 4: Detailed processing steps of the RITS NN architecture from the forward direction (from X_1 to X_7). For the backward RITS, data flows from X_7 to X_1.

BRITS processing flow. We plot Figure 4 to display detailed processing steps in the forward RITS that takes in data in the order from X_1 to X_7. Likewise, the backward RITS processes data in the order from X_7 to X_1. For each step of data, its hidden state is sent into the history-based regression layer to produce the estimation of missing values based on history values. The first step has no hidden states before it, hence H_0 here is initialized as a 0 vector. The output of the history-based layer is sent into the feature-based regression layer to produce the feature-based estimation. When calculating the estimation of missing values in one feature, the feature-based regression layer takes the correlation between this feature and others into consideration. Subsequently, the model combines the history-based estimation and the feature-based estimation by learned weights to form estimated variable X'. X' is used to impute missing values in X to form the complement variable X^c. The LSTM layer takes X^c to process temporal information and produce a hidden state for the next step. Such a cycle repeats for the whole time series. Finally, the classifier takes the last hidden state to produce the probability of ILOS. Our BRITS implementation is from the open-source code repository in the original paper [2]. We make modifications to

fit our datasets. In our experiments, BRITS obtains the best performance when the RNN hidden size equals 256. Therefore, we fix this most important hyper-parameter as 256 for all BRITS models used in this work.

3.3.3 Transfer Learning

Transfer learning is adopted in our work to help our models obtain better performance on small datasets. We design the transfer learning with two stages: pre-training and fine-tuning. In practice, the whole transfer learning methodology is usually applied on neural networks only, such as BRITS. Although random forest and XGBoost have no way to get tuned like BRITS, they still can be pre-trained to learn general knowledge across our network datasets. Therefore, pre-training here can also be considered as knowledge sharing, which shares the general representation of ILOS signatures across networks.

To collect the general representation of data, we merge six datasets into one, called the mega-dataset. Training sets, validation sets, and test sets from each network dataset are merged into the corresponding mega-training set, the mega-validation set, and the mega-test set. For features that are different between networks, we take the union of these features. Consequently, the feature number of the mega-dataset is 125, as shown in Table 1. We also keep a column to record which network the sample belongs to. This column is not a feature and not visible to models, which is only used to filter out other networks' data while fine-tuning models on the network-specific dataset.

As shown in Figure 5, BRITS is firstly pre-trained on the mega-dataset to learn general representation and then fine-tuned on single-network datasets to learn network-specific knowledge. In pre-training, BRITS is just trained on the mega-dataset rather than single-network datasets, and the learning rate is set as 10^{-3}.

Fine-tuning is only to keep a specific network's data and let pre-trained BRITS adjust its parameters on these network-specific samples. The learning rate in the fine-tuning stage is halved to 5×10^{-4}. The general practice of fine-tuning is to tune the classifier but keep other parts frozen. In our case, this means freezing parameters in the imputer and only retraining the classifier of BRITS on network-specific datasets. In another strategy, we also expand the fine-tuning to the imputer, which means parameters of the entire BRITS model are adjusted on network-specific data during

the fine-tuning stage. For random forest and XGBoost, these two models only have the pre-training stage without the fine-tuning stage.

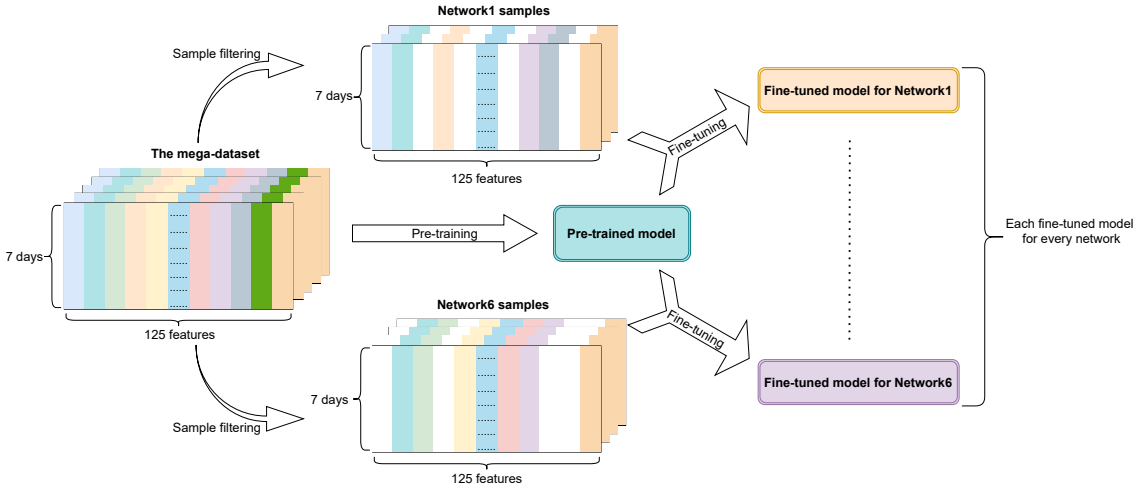


Figure 5: Overview of our transfer learning methodology. Combining all six network datasets, our "mega-dataset" has 125 features for each of the past 7 days. Taking the datasets for Network1 and Network6 as an example, some features in Network1 may not exist in Network6, and vice versa (white/blank columns in the figure). Our model is initially pre-trained on the mega-training set and subsequently fine-tuned on network-specific samples to generate fine-tuned models for each network, resulting in a total of 6 models.

3.4 Experiments

In this section, we design three experiments to validate our methodology. First we compare across models trained on individual datasets. Second, we discuss the feasibility of transferring the general representation of ILOS signatures across networks. In particular, we observe how much improvement transfer learning can bring to our models on small datasets. Third, we benchmark the performance of our models on the two main commercial use-cases of NHP to show our work is valuable in practical applications.

It is worthwhile to mention that, in this section, models mentioned in italic font represent models produced in our experiments, such as *BRITS (zero imputation)*.

3.4.1 Evaluation Metric

As mentioned before, our real-world data has data quality issues, notably, missing data and rare positive samples. Moreover, in collected positive samples, some of them are intrinsically unpredictable, such as fiber cuts. All of these factors result in our predictions having low recalls.

Considering our imbalanced datasets, we select Area Under Precision-Recall Curve (PR-AUC) as the metric, which is not sensitive to the number of samples. Formal definitions of precision and recall are displayed in Equation 3. From another aspect, we weigh precision more than recall, thus targeting high precision at the expense of low recall. Therefore, we focus on the PR-AUC to recall 0.1, and consequently, the full score of our evaluation metric is 0.1. As formulated in Equation 4, we define our evaluation metric as an integral of area under the curve, where D is our evaluation metric, namely the area under PR curve to recall 0.1, axis x is the recall, axis y is the precision, and $f(x)$ represents the PR curve. Our evaluation metric is used for model selection and model evaluation.

$$\begin{aligned} Precision &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \\ Recall &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \end{aligned} \tag{3}$$

$$D = \int_0^{0.1} \mathrm{dx} \int_0^{f(x)} \mathrm{xydy} \tag{4}$$

3.4.2 Experiment Settings

Comparison of Models on Individual Datasets

Missing data processing. We apply zero-imputation and median-imputation separately on datasets for random forest. For XGBoost and BRITS, we also use zero-imputation on their inputs to make a fair comparison to random forest. Furthermore, XBoost and BRITS both have designed mechanisms to handle missing values, so we let them process missing data by themselves to show how much improvement these two mechanisms can bring.

Hyper-parameter searching. For tree-based algorithms random forest and XGBoost, we run parameter searching for the hyper-parameter *number of trees* on both of them. The value range of *number of trees* is from 100 to 3,000 in intervals of 100

and a total of 30 values. For each value, we train random forest and XGBoost on the training set. After that, we use the validation set to pick out the best model and run on the test set to generate test scores.

In the training of BRITS, we first train it only on the imputation task until the estimation loss on the validation set tends to be steady. Next, we continue optimizing the model on both the imputation task and classification task. BRITS is trained on each network dataset by an Adam optimizer with the learning rate as 10^{-3} and batch size as $1,024$.

Transfer Learning

Transfer learning's first stage, namely pre-training, is applicable to all models on the mega-dataset. After the pre-training finishes, we begin to fine-tune the pre-trained model on network-specific samples. In the fine-tuning period, we lower the learning rate to 5×10^{-4}. Generally, we only allow the last few layers in the neural network to update parameters during fine-tuning. For instance, in classification tasks, only the classifier should be fine-tuned but with other parts frozen to keep the learned representation fixed and only adjust the classification border in the classifier. This is how the model *BRITS (fine-tune the classifier only)* is fine-tuned. In addition, we have another fine-tuning model, *BRITS (fine-tune the entirety)* that fine-tunes the entire BRITS model to observe whether adjusting the learned representation together with the classification can help obtain more improvement.

Performance on Main Use-cases

NHP has been applied on two important commercial use-cases, namely 100G OTN line cards and 10G Ethernet clients. To test our model performance on both use-cases, we extract their respective samples in the mega-test dataset and run trained models of XGBoost and BRITS to obtain test scores. Furthermore, we train XGBoost and BRITS separately on 100G OTN line cards and 10G Ethernet clients. The purpose is to compare the learning performance between models trained on single facility type and all 12 facility types.

3.4.3 Experiment Results

Our model performance results on each single-network dataset are listed in Table 2. Table 3 contains the results of transfer learning across networks. Table 4 lists model performance on two main use-cases in NHP: 100G OTN line cards and 10G Ethernet clients. In all tables, bold values are the best results on the dataset.

Comparison Across Models Trained on Individual Datasets

The results in Table 2 show that *Random Forest (zero imputation)* and *Random Forest (median imputation)* achieve comparable results on datasets Network1, Network2, and Network5. On Network3, Network4, and Network6, we observe that *Random Forest (median imputation)* is inferior to *Random Forest (zero imputation)* by 61.1% of the weighted average score. The results indicate that (1) imputation methods can affect model performance significantly for random forest on our datasets; (2) the median imputation works on some of features, but not on the others since our datasets from different networks have different distributions in the feature space.

Models of XGBoost and BRITS all produce higher learning accuracy than random forest. Both *XGBoost* and *BRITS* deomstrate a 72.4% performance improvement over baseline *Random Forest (zero imputation)* on the weighted average score. It is observed that *XGBoost (zero imputation)* produces the same learning accuracy as *BRITS (zero imputation)* in terms of the weighted average score. Likewise, *XGBoost* and *BRITS* have the same overall performance. Furthermore, missing data handling mechanisms of XGBoost and BRITS can bring further improvement (with 4.2% improvement on the weighted average score). Overall, *XGBoost* and *BRITS* have comparable learning performance on datasets Network3, Network4, and Network5. On Network2, *BRITS* is 11.5% better than *XGBoost*. On Network6, *XGBoost* outperforms *BRITS* by 7.1%.

We further observe the model performance on individual networks with small datasets, in particular on Network1. Network1 has the least number of samples and the least number of positive samples, leading to least ratio of positive samples. Both models of BRITS outperform XGBoost and random forest. XGBoost's performance on this small dataset and a low ratio of positive samples is close to baseline models. This also indicates BRITS is a more suitable solution to address the challenge of low positive data ratio and the small sample size. The result of *BRITS (zero imputation)*

is 13.6% better than *BRITS*, indicating zero-imputation is more effective than BRITS' missing data processing mechanism on dataset Network1. Moreover, *BRITS (zero imputation)* gains 38.9% improvement than the baseline model of *Random Forest (zero imputation)*. It also outperforms *XGBoost (zero imputation)* by 47.1%. As the size of data samples increases, the performances of XGBoost and BRITS become comparable.

Table 2: Performance comparison across models trained on single-network datasets. The assessment metric used is PR-AUC to recall 0.1, defined in Subsection 3.4.1. The rightmost column shows each model's scores averaged over the mega-datasets that represent models' overall performance. The average scores are weighted by the number of samples in each test dataset. The best result of each column is highlighted in bold.

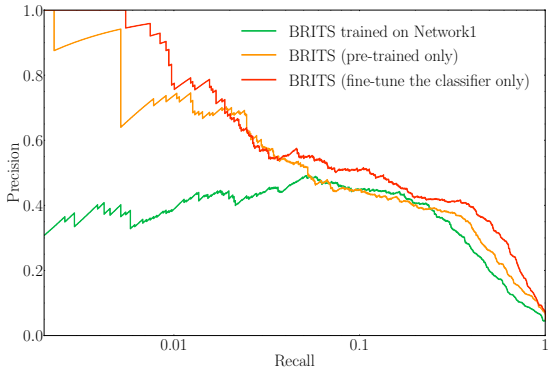
Model	Network1	Network2	Network3	Network4	Network5	Network6	Weighted Avg.
Random Forest (zero imputation)	0.036	0.010	0.018	0.023	0.046	0.042	0.029
Random Forest (median imputation)	0.035	0.007	0.003	0.012	0.049	0.005	0.018
XGBoost (zero imputation)	0.034	0.023	**0.087**	0.039	0.066	0.056	0.048
BRITS (zero imputation)	**0.050**	0.026	0.078	0.039	0.065	0.053	0.048
XGBoost	0.033	0.026	**0.087**	**0.041**	0.066	**0.060**	**0.050**
BRITS	0.044	**0.029**	**0.087**	0.040	**0.067**	0.056	**0.050**

Table 3: Performance comparison across models trained on mega-datasets. The assessment metric used is PR-AUC to recall 0.1. The rightmost column shows each model's scores averaged over the mega-datasets that represent models' overall performance. The average scores are weighted by the number of samples in each test dataset. The best result of each column is highlighted in bold.

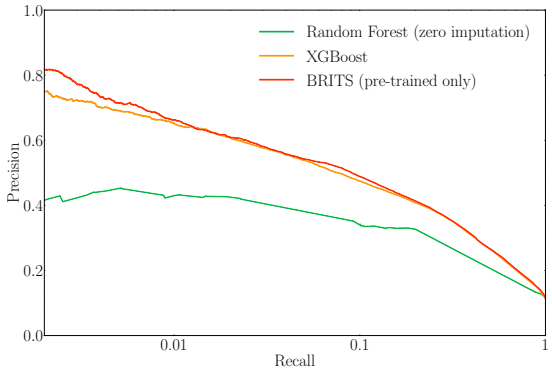
Model	Network1	Network2	Network3	Network4	Network5	Network6	Weighted Avg.
Random Forest (zero imputation)	0.024	0.009	0.009	0.022	0.044	0.028	0.024
XGBoost	0.058	0.023	**0.089**	**0.040**	0.064	0.055	0.048
BRITS (pre-trained only)	0.056	0.026	0.088	0.038	0.066	0.056	0.049
BRITS (fine-tune the classifier only)	**0.060**	0.026	0.085	0.038	**0.068**	0.056	0.050
BRITS (fine-tune the entirety)	0.055	**0.030**	0.087	**0.040**	**0.068**	**0.057**	**0.052**

Transfer Learning

Transfer learning is adopted to improve the learning performance on networks with insufficient data samples. Transfer learning normally consists of two stages, namely pre-training and fine-tuning. Models of *Random Forest (zero imputation)* and *XGBoost* only have the pre-training stage without the fine-tuning stage. We can consider pre-training on mega-datasets as the knowledge sharing of six networks. As shown in Table 3, knowledge sharing through pre-training on mega-datasets allows *XGBoost* to improve the learning performance on Network1 by 75.8% compared to Table 2.



(a) BRITS performance comparison on samples from Network1



(b) Model performance comparison on the overall mega-test set

Figure 6: Performance as a function of recall for various models in this study. We apply log scaling on the recall axis. In subfigure 6(a), we compare the performance of BRITS models evaluated on data from Network1 only. The model is pretrained on the mega-dataset and fine-tuned on Network1. In subfigure 6(b), we compare the performance of *Random Forest (zero imputation)*, *XGBoost* and *BRITS (pre-trained only)*(see Table 3), evaluated on the mega-test set.

The Random Forest Model does not benefit from the knowledge sharing practices with even degraded results. One possible reason is the increased missing rate in the mega-dataset. As shown in Table 1, the mega-dataset's missing rate is the highest as a result of merging 125 feature columns from six network datasets.

BRITS (pre-trained only) is pre-trained on the mega-dataset but not fine-tuned on any network-specific samples. Meanwhile, the second variation of the transfer learning model is *BRITS (fine-tune the classifier only)* that has its imputer frozen and its classifier fine-tuned on corresponding network-specific data samples. This means the representation learned from the mega-dataset is fixed while the classifier is fine-tuned. The third variation of the transfer learning model on BRITS is *BRITS (fine-tune the entirety)* with both its imputer and classifier being fine-tuned.

In the case of Network1 with the smallest dataset, *BRITS (pre-trained only)* achieves 27.3% improvement in Table 3 compared with *BRITS* trained only singular networks in Table 2. This indicates that transfer learning from mega-dataset to singular network is effective with relatively small statistics. *BRITS (fine-tune the classifier only)* further improves learning performance by 36.4%. However, the performance of *BRITS (fine-tune the entirety)* degrades compared to *BRITS (pre-trained only)*. This indicates that fine-tuning for Network1 helps the classifier but not the imputer of the

BRITS model. We plot Precision-Recall curves in Figure 6(a) to demonstrate the learning performance improvements.

In contrast, on Network2/3/4/5/6 with larger datasets, all the models produce comparable learning performances in Table 2 *vs* Table 3. This illustrates that knowledge transfer from the mega-dataset to a singular network stops improving the models' accuracy with sufficiently large statistics. However, the models' performance does not degrade either, which indicates that a single pre-training from the mega-dataset can produce ML models usable for all networks at once.

To demonstrate the model performance on the overall mega-dataset, Precision-Recall curves plotted in Figure 6(b) display a comparison between *Random Forest (zero imputation)*, *XGBoost*, and *BRITS (pre-trained only)*. *XGBoost* and *BRITS (pre-trained only)* both outperform *Random Forest (zero imputation)*. In comparison, *BRITS (pre-trained only)* has comparable learning performance to *XGBoost*.

Performance on Main Use-cases

Table 4: Comparison of model performance on two important use-cases. Models presented in this table are all trained on the mega-dataset, but on different subsets of the 12 facilities. For example, *XGBoost trained on 100G OTN line cards* is trained on samples collected from only line-facing ports of 100G OTN line cards. The evaluation metric used is PR-AUC to recall 0.1. Models trained on 12 facility types obtain close performance with models specifically trained on 100G lines or ETH10G clients.

Model	100G OTN line cards	10G Ethernet clients
XGBoost trained on 100G OTN line cards	0.049	/
XGBoost trained on 10G Ethernet clients	/	0.047
XGBoost trained on all 12 facility types	0.049	**0.053**
BRITS trained on 100G OTN line cards	0.054	/
BRITS trained on 10G Ethernet clients	/	0.050
BRITS trained on all 12 facility types	**0.055**	0.052

To validate XGBoost and BRITS' performance on two main commercial use-cases, we compare models trained on all 12 facility types with models trained on only 100G lines or ETH10G clients. As the results show in Table 4, for both XGBoost or BRITS, models trained on 12 facility types produce comparable results with models specifically trained on 100G lines or ETH10G clients. Model *XGBoost trained on all 12 facility types* in Table 4 is equivalent to *XGBoost* in Table 3; and *BRITS trained on all 12 facility types* in Table 4 is equivalent to *BRITS (pre-trained only)*

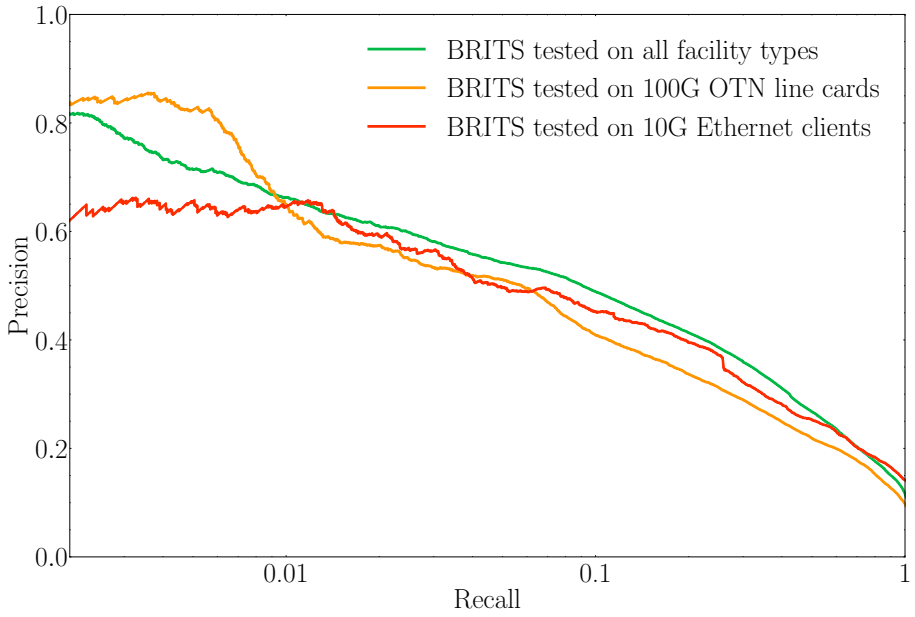


Figure 7: Precision as a function of recall for the model *BRITS (pre-trained only)* in Table 3 evaluated on on all facility types (green) the 100G line-side facilities (orange) and 10G Ethernet client-side facilities (red).

in Table 3. XGBoost and BRITS achieve comparable learning performance on the use case of ETH10G clients. On the use case of 100G lines, BRITS models perform approximately 10% better than XGBoost models.

Figure 7 contains PR curves of *BRITS (pre-trained only)* tested separately on samples of all 12 facility types, 100G lines, and ETH10G clients. Over all facility types, our model achieves 1% recall at 65% precision. For 100G lines, the precision can go above 80% at low recall. As a result, 3% of network outages are reported at least once 1-7 days before they occur. In practical large-scale networks, this corresponds to roughly a dozen alerts per day. Given the limited resources that network operators have to investigate forecasted failures, as opposed to managing current actual failures, this is a practicable number. Operators benefit from receiving a relatively small number of forecasted alerts, which have a high probability of being correct, rather than being flooded with a large number of alerts, which would require many resources to investigate.

Discussion

Our rule-based labeling method encodes the domain knowledge to identify ILOS events automatically. The generated datasets with ILOS labels enable our supervised ML methodology for forecasting. XGBoost is an out-of-box model on the ILOS prediction task with the benefit of easy and fast training. As a deep learning model, BRITS obtains similar results to XGBoost and performs better on small datasets. With transfer learning, BRITS obtains the best weighted average score. Furthermore, knowledge sharing further improves XGBoost's performance on small datasets as well. To address the missing data problem, zero-imputation works relatively well on our datasets because zero-suppression is the dominating factor for missing data in counter PMs. However, we observe that more sophisticated missing-data handling in XGBoost and BRITS brings additional improvements and produces the optimal learning performance. Last but not least, the singular model of XGBoost or BRITS trained on mega-datasets can handle all 12 facility types from all six networks without the need of training the model for each facility type or each network specifically.

3.5 Summary

This work has developed a supervised ML methodology to forecast Imminent Loss of Signal (ILOS) in the long term, 1-7 days before they occur in optical networks. To achieve this goal, we have developed ML best practices tailored for real-world telecommunications data. We have analyzed PM data collected from optical network equipment in six commercial networks of different sizes, totalling 1,373 days and 16,456,331 data samples. As expected, we discover that the general robustness of optical networks and the prevalence of unpredictable events such as fiber cuts in the field make LOS events relatively rare and unpredictable. Still, our models are able to forecast 3% of ILOS events from all receiver ports of the optical network with 65% precision per prediction. For 100G OTN lines, our precision reaches over 80%, albeit at low recall. Furthermore, we achieve nearly optimal results with a single ML model for all facilities and all networks, making it usable for commercial products that work out-of-the-box. Hence this work enables preventive actions by network operators to avoid network outages.

The suppression of zero values and the diversity of metrics reported by ports on

these networks result in extremely sparse input datasets to our ML models, of which 70%∼80% are null values. To overcome this obstacle, we have experimented with XGBoost and BRITS ML architectures, which both can handle null input values. With all missing values imputed with 0, we find that our XGBoost and BRITS models both outperform the baseline random forest model significantly. Furthermore, letting XGBoost and BRITS handle missing data by themselves can further obtain a small improvement. Without transfer learning, BRITS obtains similar results to XGBoost. While XGBoost is easy to use and fast to train, the ability of the BRITS architecture to impute missing values enables other application scenarios, such as generating complete data for other tasks. With the help of transfer learning, *BRITS (fine-tune the entirety)* achieves the best results on our datasets. For network datasets with insufficient data to train a robust ML model, knowledge sharing across all six networks can improve the model performance significantly (*XGBoost* improved by 75.8%, *BRITS (fine-tune the classifier only)* improved by 36.4%). Fine-tuning on BRITS deliver improvement, but not as much as knowledge sharing that shares general representation of ILOS signatures across networks.

In future work, we plan to improve our ML methodology by considering multiple prediction classes corresponding to different root-causes and failure modes in optical networks. This will enable better separation of intrinsically unpredictable events from predictable ILOS and better optimization of the ML training for the latter. We will also leverage data with higher time-resolution and loosen the strict distinction between future-tense forecasting vs present-tense assurance, which will increase the recall and multiply the added value of the application for network operation centers. Finally, we plan to extend our analysis to multi-vendor equipment, as we expect our methodology to apply qualitatively to any optical network from which similar performance monitoring data can be obtained.

Chapter 4

Methodology: Developing Self-Attention-based Imputation for Time Series

Our methodology is made up of two parts: (1). the joint-optimization training approach of imputation and reconstruction; (2). the SAITS model, a weighted combination of two DMSA blocks.

4.1 Joint-optimization Training Approach

We first present the definition of multivariate time series data as:

Definition of Multivariate Time Series Data Given a collection of multivariate time series with T time steps and D dimensions, we denote it as $X = \{x_1, x_2, ..., x_t, ..., x_T\} \in \mathbb{R}^{T \times D}$, where the t-th observation $x_t = \{x_t^1, x_t^2, ..., x_t^d, ..., x_t^D\}$. Therefore, we use X_t^d to represent the d-th dimension variable of the t-th step observation in X. To represent the missing variables in X, we introduce the missing mask vector $M \in \mathbb{R}^{T \times D}$, where

$$M_t^d = \begin{cases} 1 & \text{if } X_t^d \text{ is observed} \\ 0 & \text{if } X_t^d \text{ is missing} \end{cases}$$

To well train self-attention-based imputation models (such as Transformer shown in Figure 8) on the above defined multivariate time-series data, we design a joint-optimization training approach of imputation and reconstruction. This approach

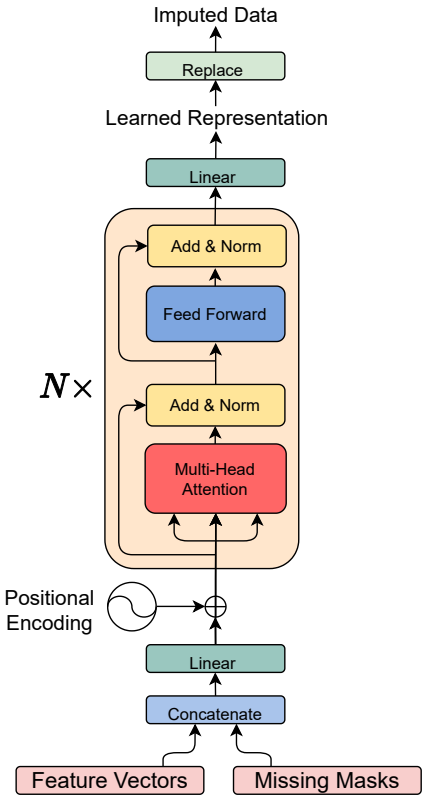


Figure 8: Structure of Transformer used in our work. The whole Transformer [1] is an auto-encoder, which is a generative model consisting of an encoder and a decoder. Here we only need the encoder part because the imputation task in our work is not taken as a generative task.

consists of two learning tasks: Masked Imputation Task (MIT) and Observed Reconstruction Task (ORT). Correspondingly, the training loss is accumulated from two losses: the imputation loss of MIT and the reconstruction loss of ORT. The details are as follows:

Task #1: Masked Imputation Task (MIT) MIT is a prediction task on artificially-masked values, which explicitly forces the model to predict missing values. In this task, for every batch input into the model, we artificially mask some percentage (such as 20% in our work) of observed values at random. These values are not visible to the model, namely, missing to the model. After the model imputes all missing values, the imputation loss is calculated between these artificially-masked values and respective imputations. We use the mean absolute error (MAE) as the imputation loss.

After artificially masking, the actual input feature vector is denoted as $\hat{X}$, and its

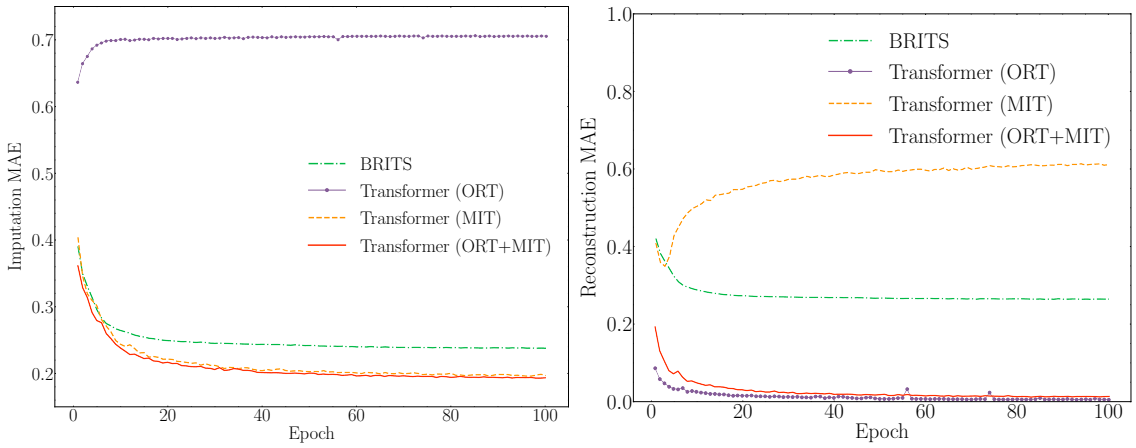
corresponding missing mask vector is $\hat{M}$. To distinguish artificially-missing values and originally-missing values, we introduce the indicating mask vector I. Math definitions of $\hat{M}$ and I are:

$$\hat{M}_t^d = \begin{cases} 1 & \text{if } \hat{X}_t^d \text{ is observed} \\ 0 & \text{if } \hat{X}_t^d \text{ is missing} \end{cases}, \quad I_t^d = \begin{cases} 1 & \text{if } \hat{X}_t^d \text{ is artificially masked} \\ 0 & \text{otherwise} \end{cases}$$

Note that learning tasks similar to MIT, which mask some objects and then predict them, are commonly used to train models in NLP (Natural Language Processing) field, such as the Cloze task [73], and the Masked Language Modeling (MLM) used to pre-train BERT [74]. MIT is inspired by MLM, but the differences are: (1). MLM is an unsupervised task, while MIT is supervised; (2) MLM predicts missing tokens (time steps), while MIT predicts missing values in time steps; (3). One disadvantage of MLM is that it causes pretrain-finetune discrepancy because masking symbols used during pretraining are absent from real data in finetuning [75]. However, the original objective of imputation is to predict missing or masked values. Therefore, MIT does not cause such discrepancies.

Task #2: Observed Reconstruction Task (ORT) ORT is a reconstruction task on the observed values. It is widely applied in the training of imputation models for both time-series and non-time series [2, 27, 28, 32, 20, 21]. After model processing, observed values in the output are different from their original values, and they are called reconstructions. In our work, the reconstruction loss is MAE between the observed values and their respective reconstructions.

In our training approach, MIT and ORT are integral. MIT is utilized to force the model to predict missing values as accurately as possible, and ORT is leveraged to ensure the model converging to the distribution of observed data. To further illustrate the effects, we take BRITS [2] as a baseline to compare with Transformer on the imputation MAE and the reconstruction MAE in two cases: Transformer (ORT) and Transformer (MIT+ORT). Figure 9(a) shows that the imputation MAE of Transformer (ORT) goes up from the beginning and is much larger than BRITS'. However, in Figure 9(b), Transformer (ORT)'s reconstruction MAE is much smaller than BRITS'. We notice ORT can only ensure that models get well trained on observed values. In other words, there is no guarantee that models can predict missing values accurately. This explains why Transformer (ORT) performs greatly on the reconstruction task but poorly on the imputation task. After appending MIT into



(a) Imputation MAE (b) Reconstruction MAE

Figure 9: Imputation MAE and reconstruction MAE in the validation stage. Three models are trained on the same data. BRITS is trained with ORT, namely in the same way as the original paper [2]. Transformer (ORT) is trained with only ORT as well, namely without MIT. Transformer (MIT) is trained on only MIT. Transformer (ORT+MIT) is trained with the joint-optimization approach, namely with both ORT and MIT.

the training of Transformer, the imputation loss of MIT becomes the guarantee. As shown in Figure 9(a), the imputation MAE of Transformer (ORT+MIT) drops steadily. Taking a look at Transformer (MIT), we can find that MIT makes the main contribution to decreasing the imputation MAE. Comparing with Transformer (MIT+ORT), Transformer (MIT) has a slightly higher imputation MAE. This also proves that ORT can help models further optimize performance on the imputation task. On the reconstruction MAE, Transformer (MIT) climes up because it is not required to converge on the observed data. Furthermore, in the aspect of the reconstruction MAE in Figure 9(b), Transformer (ORT+MIT) is slightly higher than Transformer (ORT) because the gradient of the reconstruction loss gets influenced by the imputation loss. This is another evidence that our joint-optimization approach works. We also discuss applying our joint-optimization approach in the training of BRITS in Section 5.6.

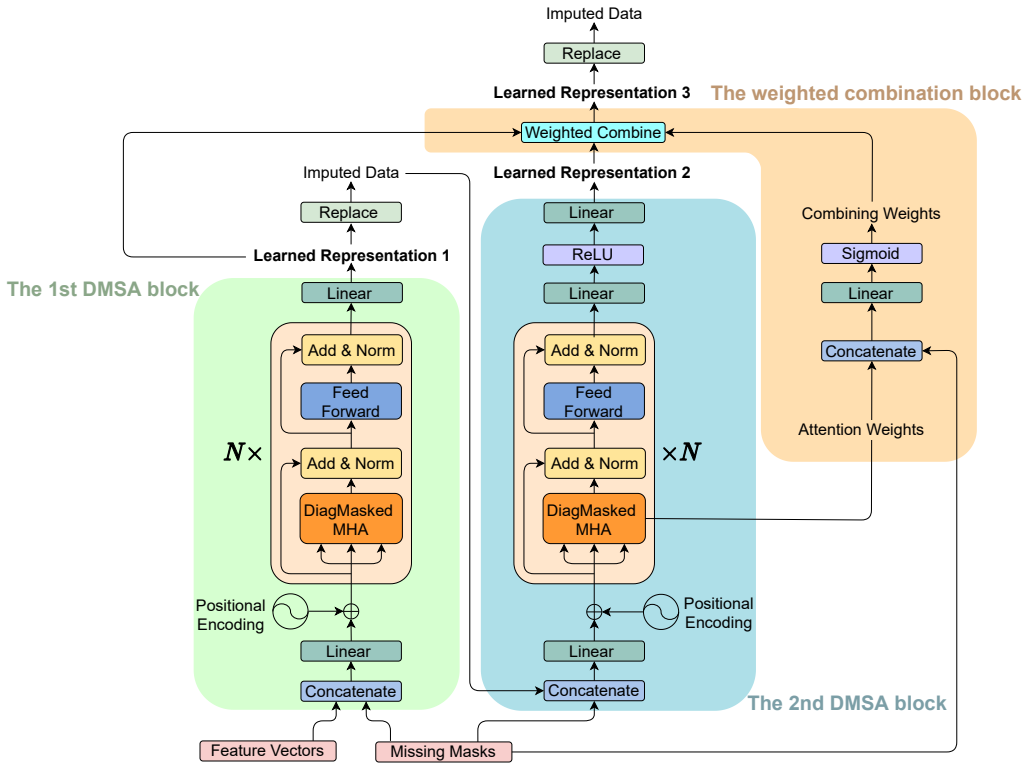


Figure 10: The SAITS model architecture.

4.2 Self-Attention-based Imputation Model

As illustrated in Figure 10, SAITS (Self-Attention-based Imputation for Time Series) is composed of two diagonally-masked self-attention (DMSA) blocks and a weighted combination. We first introduce some fundamental components of SAITS in Subsection 4.2.1 and 4.2.2. Then we illustrate SAITS' three-part structure in Subsection 4.2.3, 4.2.4, and 4.2.5, respectively. Finally, we discuss the loss functions of learning tasks in Subsection 4.2.6.

4.2.1 Diagonally-Masked Self-Attention

The conventional self-attention mechanism is proposed by Vaswani et al. [1] to solve the language translation task. Now it is widely applied in sequence modeling. A given sequence is mapped into a query vector Q of dimension d_k, a key vector K of dimension d_k and a value vector V of dimension d_v. The scaled dot-product can effectively calculate attention scores (or the attention map) between Q and K. After that, a softmax function is applied to obtain attention weights. The final output is

attention-weighted V. The whole process is as shown in Eq. 5 below:

$$\text{SelfAttention}\left(Q, K, V\right) = \text{Softmax}\left(\frac{QK^{\mathsf{T}}}{\sqrt{d_k}}\right)V \tag{5}$$

To enhance SAITS' imputation ability, we apply diagonal masks inside the self-attention. As formulated in Eq. 6 and 7, we set diagonal entries of the attention map ($\in \mathbb{R}^{T\times T}$) as $-\infty$ (set as -1×10^9 in practice) so that after the softmax function, the diagonal attention weights approach 0.

$$\left[\text{DiagMask}\left(x\right)\right]\left(i,j\right) = \begin{cases} -\infty & i = j \\ x\left(i,j\right) & i \neq j \end{cases} \tag{6}$$

$$\begin{aligned} \text{DiagMaskedSelfAttention}\left(Q, K, V\right) &= \text{Softmax}\left(\text{DiagMask}\left(\frac{QK^{\mathsf{T}}}{\sqrt{d_k}}\right)\right)V \\ &= AV \\ &\text{where } A \text{ is attention weights} \end{aligned} \tag{7}$$

Therefore, input values at the t-th step can not see themselves and are prohibited from contributing to their own estimations. Consequently, their estimations depend only on input values from other $(T-1)$ steps. This is the reason why diagonally-masked self-attention (DMSA) can explicitly capture both the temporal dependencies and feature correlations between time steps. Subsequently, the diagonally-masked multi-head attention (DiagMaskedMHA) is formulated:

$$\begin{aligned} \text{DiagMaskedMHA}\left(x\right) &= \text{Concat}\left(head_1, head_2, ..., head_i, ..., head_h\right)W^O \\ &\text{where } head_i = \text{DiagMaskedSelfAttention}\left(xW_i^Q, xW_i^K, xW_i^V\right), \\ &\text{and } h \text{ is the number of heads,} \\ &\text{parameters } W_i^Q \in \mathbb{R}^{d_{\text{model}}\times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}}\times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}}\times d_v}, \text{and } W^O \in \mathbb{R}^{hd_v\times d_{\text{model}}} \end{aligned} \tag{8}$$

To prove the effectiveness of DMSA, we conduct an ablation study in Appendix 5.7.1. Note that attention masks are widely applied in self-attention modeling, especially in NLP field, including the diagonal masks used here, for example [76, 75, 77].

4.2.2 Positional Encoding and Feed-Forward Network

In Transformer, Vaswani et al. [1] apply the positional encoding to make use of the sequence order because there is no notion of sequence order in the original Transformer

architecture. Additionally, there is a fully-connected feed-forward network applied behind each attention layer. In SAITS, we keep both the positional encoding and the feed-forward network.

The positional encoding consists of sine and cosine functions, which is formulated as Eq. 9 below. Note that we use p to refer to the positional encoding in the following equations for brevity.

$$\begin{gathered}\mathrm{PosEnc}(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \quad \mathrm{PosEnc}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \\ \text{where } pos \text{ is the time-step position, } i \text{ is the dimension}\end{gathered} \tag{9}$$

The feed-forward network has two linear transformations with a ReLU activation function between them, as shown in Eq. 10:

$$\begin{gathered}\mathrm{FFN}\left(x\right) = \mathrm{ReLU}\left(xW_1 + b_1\right)W_2 + b_2 \\ \text{where } W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}, b_1 \in \mathbb{R}^{d_{\text{ffn}}}, W_2 \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{model}}}, b_2 \in \mathbb{R}^{d_{\text{model}}}\end{gathered} \tag{10}$$

4.2.3 The First DMSA Block

$$e = \left[\mathrm{Concat}\left(\hat{X}, \hat{M}\right)W_e + b_e\right] + p \tag{11}$$

$$z = \left\{\mathrm{FFN}(\mathrm{DiagMaskedMHA}\left(e\right))\right\}^N \tag{12}$$

$$\tilde{X}_1 = zW_z + b_z \tag{13}$$

$$\hat{X}' = \hat{M} \odot \hat{X} + \left(1 - \hat{M}\right) \odot \tilde{X}_1 \tag{14}$$

In the first DMSA block, we concatenate the actual input feature vector $\hat{X}$ and its missing mask vector $\hat{M}$ as input. Eq. 11 projects the input to d_{model} dimensions and adds up with the positional encoding p to produce e. W_e and b_e are parameters ($W_e \in \mathbb{R}^{D \times d_{\text{model}}}$, $b_e \in \mathbb{R}^{d_{\text{model}}}$). Operation $\{\}^N$ in Eq. 12 means stacking N layers. Eq. 12 transfers e to z with N stacked layers of the diagonally-masked multi-head attention and the feed-forward network [1]. Eq. 13 reduces z from d_{model} dimensions to D dimensions and produces $\tilde{X}_1$ (**Learned Representation 1**). Parameter $W_z \in \mathbb{R}^{d_{\text{model}} \times D}$ and $b_z \in \mathbb{R}^D$. In Eq. 14, we replace missing values in $\hat{X}$ with corresponding

[1]Note that the layer normalization [78] and residual connection [79] are applied after each attention layer and feed-forward layer in the same way as [1]. Fig. 10 shows these details. They are suppressed here for simplicity.

values in $\tilde{X}_1$ and obtain the completed feature vector $\hat{X}'$ with the observed part in $\hat{X}$ kept intact. Here, $\odot$ is the element-wise product.

4.2.4 The Second DMSA Block

$$\alpha = \left[\text{Concat}\left(\hat{X}', \hat{M}\right) W_\alpha + b_\alpha\right] + p \tag{15}$$

$$\beta = \left\{\text{FFN}\left(\text{DiagMaskedMHA}\left(\alpha\right)\right)\right\}^N \tag{16}$$

$$\tilde{X}_2 = \text{ReLU}\left(\beta W_\beta + b_\beta\right) W_\gamma + b_\gamma \tag{17}$$

The second DMSA block takes the output $\hat{X}'$ of the first DMSA block and continues learning. Similar to Eq. 11, Eq. 15 projects the concatenation of $\hat{X}'$ and $\hat{M}$ from D dimensions to d_{model} dimensions and then adds the result together with p to generate α. Parameter $W_\alpha \in \mathbb{R}^{D \times d_{\text{model}}}$, $b_\alpha \in \mathbb{R}^{d_{\text{model}}}$. Eq. 16 performs N times of nested attention functions and feed-forward networks on α and outputs β. In Eq. 17, to obtain $\tilde{X}_2$ (**Learned Representation 2**), we apply two linear projections on β with a ReLU activation in between, where parameter $W_\beta \in \mathbb{R}^{d_{\text{model}} \times D}$, $b_\beta \in \mathbb{R}^D$, $W_\gamma \in \mathbb{R}^{D \times D}$, $b_\gamma \in \mathbb{R}^D$. We find such operation can help achieve better performance than applying a single linear projection here. We do not apply the same transformation to obtain $\tilde{X}_1$ because the learnable parameters in the following weighted combination can dynamically adjust the weights for $\tilde{X}_1$ and $\tilde{X}_2$ to form better $\tilde{X}_3$ (**Learned Representation 3**). Moreover, in practice, we find that applying the same transformation to obtain $\tilde{X}_1$ does not help achieve better results. It validates the effectiveness of our weighted combination, described as below.

4.2.5 The Weighted Combination Block

$$\hat{A} = \frac{1}{h} \sum_i^h A_i \tag{18}$$

$$\eta = \text{Sigmoid}\left(\text{Concat}\left(\hat{A}, \hat{M}\right) W_\eta + b_\eta\right) \tag{19}$$

$$\tilde{X}_3 = (1 - \eta) \odot \tilde{X}_1 + \eta \odot \tilde{X}_2 \tag{20}$$

$$\hat{X}_c = \hat{M} \odot \hat{X} + \left(1 - \hat{M}\right) \odot \tilde{X}_3 \tag{21}$$

To obtain better $\tilde{X}_3$, the weighted combination block is designed to dynamically weigh $\tilde{X}_1$ and $\tilde{X}_2$ according to temporal dependencies and missingness information. $\hat{A}$ ($\in \mathbb{R}^{T\times T}$) in Eq. 18 is averaged from attention weights A output by multi heads in the last layer of the second DMSA block. Eq. 19 takes averaged attention weights $\hat{A}$ and missing masks $\hat{M}$ as references to produce the combining weights η ($\in (0,1)^{T\times D}$) with the learnable parameters W_η ($\in \mathbb{R}^{(T+D)\times D}$) and b_η ($\in \mathbb{R}^D$). Eq. 20 combines $\tilde{X}_1$ and $\tilde{X}_2$ by weights η to form $\tilde{X}_3$. We replace missing values in $\hat{X}$ with corresponding values in $\tilde{X}_3$ to produce the complement vector $\hat{X}_c$, namely the imputed data. To further discuss the rationality of the weighted combination, we perform an ablation experiment in Section 5.7.2.

Moreover, the second DMSA block and the weighted combination block are added to increase our network's depth and to obtain better performance. We do not apply more than two DMSA blocks because the benefit brought is marginal. Experiments and analysis are conducted to prove our points here in Section 5.7.3.

4.2.6 Loss Functions of Learning Tasks

$$\ell_{\text{MAE}}\left(estimation, target, mask\right) = \frac{\sum_d^D \sum_t^T |(estimation - target) \odot mask|_t^d}{\sum_d^D \sum_t^T mask_t^d} \tag{22}$$

$$\mathcal{L}_{\text{ORT}} = \frac{1}{3}\left(\ell_{\text{MAE}}\left(\tilde{X}_1, \hat{X}, \hat{M}\right) + \ell_{\text{MAE}}\left(\tilde{X}_2, \hat{X}, \hat{M}\right) + \ell_{\text{MAE}}\left(\tilde{X}_3, \hat{X}, \hat{M}\right)\right) \tag{23}$$

$$\mathcal{L}_{\text{MIT}} = \ell_{\text{MAE}}\left(\hat{X}_c, X, I\right) \tag{24}$$

$$\mathcal{L} = \mathcal{L}_{\text{ORT}} + \mathcal{L}_{\text{MIT}} \tag{25}$$

We have two learning tasks in the model training: MIT and ORT. The imputation loss of MIT ($\mathcal{L}_{\text{MIT}}$) and the reconstruction loss of ORT ($\mathcal{L}_{\text{ORT}}$) are both calculated by the MAE loss function (ℓ_{MAE}) defined in Eq. 22, which takes three inputs: *estimation*, *target*, and *mask* (all of them $\in \mathbb{R}^{T\times D}$). It calculates MAE between values indicated by *mask* in *estimation* and *target*. *target* and *mask* of $\mathcal{L}_{\text{ORT}}$ in Eq. 23 are the input feature vector $\hat{X}$ and its missing mask vector $\hat{M}$. We let $\tilde{X}_1$ and $\tilde{X}_2$ directly participate in the composition of $\tilde{X}_3$. Therefore, here we make $\mathcal{L}_{\text{ORT}}$ accumulated from three learned representations: $\tilde{X}_1$, $\tilde{X}_2$ and $\tilde{X}_3$. We find such accumulated loss can lead to faster convergence speed. To ensure $\mathcal{L}_{\text{ORT}}$ not too large to dominate the

direction of the gradient, it is reduced by a factor of three, namely averaged. Inputs *estimation*, *target* and *mask* of $\mathcal{L}_{\text{MIT}}$ in Eq. 24 are the complement feature vector $\hat{X}_c$, the original feature vector X without artificially-masked values, and the indicating mask vector I, respectively. At last, Eq. 25 adds $\mathcal{L}_{\text{ORT}}$ and $\mathcal{L}_{\text{MIT}}$ together, and our SAITS model is updated by minimizing the final loss $\mathcal{L}$.

Chapter 5

Evaluation

To ensure the reproducibility of our results, we make our work available to the community. Our data preprocessing scripts, model implementations, as well as hyper-parameter search configurations, are all available in the GitHub repository `https://github.com/SAITS2021/SAITS`.

5.1 Dataset Details

This section contains elaborated descriptions of three public datasets used in this work and their preprocessing details. General information of all datasets is listed in Table 5. Note that standardization is applied in the preprocessing of all datasets.

In order to benchmark the proposed SAITS model, we perform experiments on three public real-world datasets from different domains: PhysioNet 2012 Mortality Prediction Challenge [1] [7], Beijing Multi-Site Air-Quality [2] [80], and Electricity Load Diagrams [3] [81]. We show details of dataset descriptions and preprocessing below.

PhysioNet 2012 Mortality Prediction Challenge (PhysioNet-2012) The PhysioNet 2012 challenge dataset [7] contains 12,000 multivariate clinical time-series samples collected from patients in ICU (Intensive Care Unit). Each sample is recorded during the first 48 hours after admission to the ICU. Depending on the status of patients, there are up to 37 time-series variables measured, such as temperature, heart

[1] https://www.physionet.org/content/challenge-2012
[2] https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data
[3] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

Table 5: General information of three datasets used in this work.

	PhysioNet-2012	Air-Quality	Electricity
Number of total samples	11,988	1,461	1,400
Number of features	37	132	370
Sequence length	48	24	100
Original missing rate	80.0%	1.6%	0%

rate, blood pressure. Measurements might be collected at regular intervals (hourly or daily), and also may be recorded at irregular intervals (only collected as required). Not all variables are available in all samples. Note that this dataset is very sparse and has 80% missing values in total. The dataset is firstly split into the training set and the test set according to 80% and 20%. Then 20% of samples are split from the training set as the validation set. We randomly eliminate 10% of observed values in the validation set and the test set and use these values as ground truth to evaluate the imputation performance of models. Following 12 samples are dropped because of containing no time-series information at all: 147514, 142731, 145611, 140501, 155655, 143656, 156254, 150309, 140936, 141264, 150649, 142998.

Beijing Multi-Site Air-Quality (Air-Quality) This air-quality dataset [80] includes hourly air pollutants data from 12 monitoring sites in Beijing. Data is collected from 2013/03/01 to 2017/02/28 (48 months in total). For each monitoring site, there are 11 continuous time series variables measured (e.g. PM2.5, PM10, SO2). We aggregate variables from 12 sites together so that this dataset has 132 features. There are totally 1.6% missing values in this dataset. The test set takes data from the first 10 months (2013/03 - 2013/12). The validation set contains data from the following 10 months (2014/01 - 2014/10). The training set takes the left 28 months (2014/11 - 2017-02). To generate time series data samples, we select every 24 hours data, namely every 24 consecutive steps, as one sample. Similar to dataset PhysioNet-2012, 10% observed values in the validation set and test set are eliminated and held out as ground-truth for evaluation.

Electricity Load Diagrams (Electricity) This is another widely-used public dataset from UCI [81]. It contains electricity consumption data (in kWh) collected

from 370 clients every 15 minutes and has no missing data. The period of this dataset is from 2011/01/01 to 2014/12/31 (48 months in total). Similar to processing AirQuality, we use the first 10 months of data (2011/01 - 2011/10) as the test set, the following 10 months of data (2011/11 - 2012/08) as the validation set and the left (2012/09 - 2014/12) as the training set. We select every 100 consecutive steps as a sample to generate time-series data for model training. Due to this dataset having no missing values, we vary artificial missing rate from 10%∼90% to eliminate observed values in the training set, validation set, and test set. This can make the comparison between our method and other SOTA models more comprehensive. Artificial missing values in the validation and test set are held out for model evaluation. Experiment results of 10% missing values are displayed in Table 6. Results of 20%∼90% missing values are shown in Table 8.

5.2 Baseline Methods

To obtain a thorough comparison, we compare our method with two naive imputation methods and five SOTA models: (1). **Median**: we fill missing values with corresponding median values from the training set; (2). **Last**: in each sample, missing values are filled by the last previous observations of given features. If there is no previous observation, 0 will be filled in; (3). **GRUI-GAN** [27]; (4). **E^{2}GAN** [28]; (5). **M-RNN** [26]; (6). **GP-VAE** [32]; (7). **BRITS** [2]. These five SOTA models have already been introduced in Chapter 2. Their open-source repositories are listed below:

- **GRUI-GAN** [27]: `https://github.com/Luoyonghong/Multivariate-Time-Series-Imputation-with-Generative-Adversarial-Networks`;
- **E^{2}GAN** [28]: `https://github.com/Luoyonghong/E2EGAN`;
- **M-RNN** [26]: `https://github.com/jsyoon0823/MRNN`;
- **GP-VAE** [32]: `https://github.com/ratschlab/GP-VAE`;
- **BRITS** [2]: `https://github.com/caow13/BRITS`;

5.3 Evaluation Metrics of Imputation Performance

We use three metrics to evaluate the imputation performance of methods: MAE (Mean Absolute Error), RMSE (Root Mean Square Error), and MRE (Mean Relative Error). Math definitions of them are listed below. Note that errors are only computed on the observed values, which are indicated by $mask$.

$$\text{MAE}\left(estimation, target, mask\right) = \frac{\sum_d^D \sum_t^T |(estimation - target) \odot mask|_t^d}{\sum_d^D \sum_t^T mask_t^d}$$

$$\text{RMSE}\left(estimation, target, mask\right) = \sqrt{\frac{\sum_d^D \sum_t^T \left(((estimation - target) \odot mask)^2\right)_t^d}{\sum_d^D \sum_t^T mask_t^d}}$$

$$\text{MRE}\left(estimation, target, mask\right) = \frac{\sum_d^D \sum_t^T |(estimation - target) \odot mask|_t^d}{\sum_d^D \sum_t^T |target \odot mask|_t^d}$$

5.4 Experimental Setup

We fix the batch size as 128 and apply the early stopping strategy in the model training. Training of models is stopped after 30 epochs without any decrease of MAE. To permit a fair comparison between the models, we execute hyper-parameter searches for every model on each dataset, except SAITS (base). For SAITS (base), we fix its hyper-parameters to form a base model and observe its performance. SAITS (base) is also applied in ablation experiments in Section 5.7. All models are trained with the Adam optimizer [82] on *Nvidia Quadro RTX 5000* GPUs. We implement our models with PyTorch [83]. We expose further details of models' hyper-parameter searching as below.

General For all models, the learning rate is log-uniformly sampled between 1×10^{-4} and 1×10^{-2}. If applicable, the dropout rate is sampled from the values (0.0, 0.1, 0.2, 0.3, 0.4, 0.5).

RNN-based models For all RNN-based models (GRUI-GAN, E^{2}GAN, M-RNN, and BRITS), the RNN hidden size is sampled from the values (32, 64, 128, 256, 512, 1024). For GRUI-GAN and E^{2}GAN, the dimension of z is sampled from (32, 64, 128,

256, 512, 1024), the number of pretrain epochs is sampled from (5, 10, 15, 20). For GRUI-GAN, λ is sampled from (0, 0.15, 0.3, 0.45). For E^{2}GAN, λ is sampled from (2, 4, 8, 16, 32, 64). set σ as 1.005.

Self-attention-based models For self-attention models (Transformer and SAITS), we sample the number of layers N from (1, 2, 3, 4, 5, 6, 7, 8), d_{model} from (64, 128, 256, 512, 1024), d_{ffn} from (128, 256, 512, 1024, 2048, 4096), d_v from (32, 64, 128, 256, 512), the number of heads h from (2, 4, 8). d_k is set as the value of d_{model} divided by h.

For model SAITS (base), we fix the learning rate = 0.001, the dropout rate = 0.1, $N = 2, d_{\text{model}} = 256, d_{\text{ffn}} = 128, h = 4, d_v = d_k = 64$.

5.5 Result Analysis

5.5.1 Imputation Performance Comparison

Table 6: Performance comparison between methods on three datasets. 10% observations in the test set are held out for evaluation. Metrics are reported in the order of MAE / RMSE / MRE. The lower, the better. Bold font indicates the best performance. GRUI-GAN and E^{2}GAN have no results on Electricity because they fail in the training due to loss explosion

Method	PhysioNet-2012	Air-Quality	Electricity
Median	0.726 / 0.988 / 103.5%	0.763 / 1.175 / 107.4%	2.056 / 2.732 / 110.1%
Last	0.862 / 1.207 / 123.0%	0.967 / 1.408 / 136.3%	1.006 / 1.533 / 53.9%
GRUI-GAN	0.765 / 1.040 / 109.1%	0.788 / 1.179 / 111.0%	/
E^{2}GAN	0.702 / 0.964 / 100.1%	0.750 / 1.126 / 105.6%	/
M-RNN	0.533 / 0.776 / 76.0%	0.294 / 0.643 / 41.4%	1.244 / 1.867 / 66.6%
GP-VAE	0.398 / 0.630 / 56.7%	0.268 / 0.614 / 37.7%	1.094 / 1.565 / 58.6%
BRITS	0.256 / 0.767 / 36.5%	0.153 / 0.525 / 21.6%	0.847 / 1.322 / 45.3%
Transformer	0.190 / 0.445 / 26.9%	0.158 / 0.521 / 22.3%	0.823 / 1.301 / 44.0%
SAITS (base)	0.192 / 0.439 / 27.3%	0.146 / 0.521 / 20.6%	0.822 / 1.221 / 44.0%
SAITS	**0.186 / 0.431 / 26.6%**	**0.137 / 0.518 / 19.3%**	**0.735 / 1.162 / 39.4%**

Table 6 reports the imputation performance of models on three datasets in three evaluation metrics (MAE / RMSE / MRE). On PhysioNet-2012 and Air-Quality,

GRUI-GAN achieves better results than Last, but is slightly inferior to Median. E^{2}GAN performs better than these three methods. On Electricity, GRUI-GAN and E^{2}GAN both fail because of loss explosion. We find this is caused by the long sequence length. Dataset Electricity's sequence length is 100. If we increase the sequence length of dataset Air-Quality from 24 to 100, both GAN models will be confronted with loss explosion and fail again. M-RNN outperforms both naive imputation methods a lot on PhysioNet-2012 and Air-Quality but gets worse results than Last on Electricity. GP-VAE and BRITS both perform much better than the methods mentioned above. BRITS is the best one among baseline methods. When it comes to self-attention-based models, Transformer surpasses BRITS obviously on PhysioNet-2012 and Electricity, and obtains comparable results to BRITS on Air-Quality. SAITS (base) achieves similar results to Transformer on all datasets. SAITS exceeds all baseline models significantly on all metrics and all datasets, and outperforms Transformer and SAITS (base) as well.

Table 7: Models' parameter number (in million) and training time of each epoch (in seconds) on datasets PhysioNet-2012, Air-Quality, and Electricity are listed from left to right. GRUI-GAN and E^{2}GAN have no results for dataset Electricity because they fail on this dataset due to loss explosion.

Model	# of param	s/epoch	# of param	s/epoch	# of param	s/epoch
GRUI-GAN	0.16M	14.4	2.32M	2.0	/	/
E^{2}GAN	0.08M	22.8	1.13M	2.2	/	/
M-RNN	0.07M	6.8	1.09M	1.3	18.63M	3.9
GP-VAE	0.15M	40.1	0.36M	8.7	13.45M	106.0
BRITS	0.73M	12.8	11.25M	1.9	7.00M	5.2
Transformer	4.36M	3.1	5.13M	0.9	14.78M	2.6
SAITS (base)	1.38M	2.7	1.56M	1.1	2.20M	2.1
SAITS	5.32M	5.0	3.07M	0.9	11.51M	2.6

To show further details of models, we list the parameter number and training speed of models from Table 6 in Table 7. We can see that GP-VAE is the slowest model and consumes the most seconds for each epoch training. RNN-based models are all slower than self-attention-based models. Compared to BRITS that yields the best results in baseline methods, SAITS takes half the training time or even less as BRITS on each epoch. Compared to Transformer, SAITS (base) has only 15%$\sim$30% parameters

of Transformer's, but it still obtains comparable performance to Transformer. This confirms that SAITS' model structure is better than Transformer on the time-series imputation task.

Table 8 shows imputation results of the experiment where we introduce missing values into dataset Electricity at different rates between 20%∼90%. Results of 10% missing rate have been displayed in Table 6. GRUI-GAN and E^{2}GAN are omitted because they fail on dataset Electricity due to loss explosion as we discussed above. Baseline methods are all inferior to self-attention-based models in all cases. SAITS achieves the best performance in eight out of nine cases. SAITS (base) performs better than Transformer in cases of 20%, 30%, and 40%. However, its performance becomes worse than Transformer's in the left cases where missing rates become higher.

Compared to BRITS, Transformer and SAITS perform much better mainly because of two reasons: (1). self-attention has a much stronger ability than RNN to capture long-term dependencies; (2). Transformer and SAITS are not auto-regressive, so they avoid compounding error, which auto-regressive models (such as BRITS) are highly susceptible to. SAITS outperforms Transformer is mainly due to its structure design (DMSA and weighted-combination), which has been clarified in Section 4.2 and been validated in the ablation experiments in Section 5.7.

5.5.2 Downstream Classification Task

Similar to prior work [2, 27, 28, 32, 33], we design a downstream classification experiment on dataset PhysioNet-2012 to further compare the imputation quality of each method. In PhysioNet-2012, each sample has a label indicating if the patient is deceased. There are 1,707 (14.2%) samples with the positive mortality label. We first let each method impute the dataset and then train a classifier on each imputed dataset to obtain classification results. Since this is a time-series dataset, we apply a simple RNN classification model as the classifier. This RNN classifier consists of a GRU layer followed by a fully connected layer. All hyper-parameters are fixed as following: the learning rate (1×10^{-3}), the batch size (128), the RNN hidden size (128), the patience of the early stopping strategy (20). We keep the classifier and the training procedure exactly the same for each imputed dataset. Considering that classes in this dataset are imbalanced, we use ROC-AUC (Area Under ROC Curve),

Table 8: Performance comparison between methods on dataset Electricity across different missing rates from 20%∼90%. Metrics are reported in the order of MAE / RMSE / MRE.

Method	20%	30%	40%
Median	2.053 / 2.726 / 109.9%	2.055 / 2.732 / 110.0%	2.058 / 2.734 / 110.2%
Last	1.012 / 1.547 / 54.2%	1.018 / 1.559 / 54.5%	1.025 / 1.578 / 54.9%
M-RNN	1.242 / 1.854 / 66.5%	1.258 / 1.876 / 67.3%	1.269 / 1.884 / 68.0%
GP-VAE	1.124 / 1.502 / 60.2%	1.057 / 1.571 / 56.6%	1.090 / 1.578 / 58.4%
BRITS	0.928 / 1.395 / 49.7%	0.943 / 1.435 / 50.4%	0.996 / 1.504 / 53.4%
Transformer	0.843 / 1.318 / 45.1%	0.846 / 1.321 / 45.3%	0.876 / 1.387 / 46.9%
SAITS (base)	0.838 / 1.264 / 44.9%	0.845 / 1.247 / 45.2%	0.873 / 1.325 / **46.7%**
SAITS	**0.763 / 1.187 / 40.8%**	**0.790 / 1.223 / 42.3%**	**0.869 / 1.314 / 46.7%**

Method	50%	60%	70%
Median	2.053 / 2.728 / 109.9%	2.057 / 2.734 / 110.2%	2.050 / 2.726 / 109.8%
Last	1.032 / 1.595 / 55.2%	1.040 / 1.615 / 55.7%	1.049 / 1.640 / 56.2%
M-RNN	1.283 / 1.902 / 68.7%	1.298 / 1.912 / 69.4%	1.305 / 1.928 / 69.9%
GP-VAE	1.097 / 1.572 / 58.8%	1.101 / 1.616 / 59.0%	1.037 / 1.598 / 55.6%
BRITS	1.037 / 1.538 / 55.5%	1.101 / 1.602 / 59.0%	1.090 / 1.617 / 58.4%
Transformer	0.895 / 1.410 / 47.9%	**0.891** / 1.404 / **47.7%**	0.920 / 1.437 / 49.3%
SAITS (base)	0.939 / 1.537 / 50.3%	0.969 / 1.565 / 51.9%	0.972 / 1.601 / 52.0%
SAITS	**0.876 / 1.377 / 46.9%**	0.892 / **1.328** / 47.9%	**0.898 / 1.273 / 48.1%**

Method	80%	90%
Mean	2.059 / 2.734 / 110.2%	2.056 / 2.723 / 110.1%
Last	1.059 / 1.663 / 56.7%	1.070 / 1.690 / 57.3%
M-RNN	1.318 / 1.951 / 70.5%	1.331 / 1.961 / 71.3%
GP-VAE	1.062 / 1.621 / 56.8%	1.004 / 1.622 / 53.7%
BRITS	1.138 / 1.665 / 61.0%	1.163 / 1.702 / 62.3%
Transformer	0.924 / 1.472 / 49.5%	0.934 / 1.491 / **49.8%**
SAITS (base)	1.012 / 1.608 / 54.2%	1.001 / 1.630 / 53.6%
SAITS	**0.908 / 1.327 / 48.6%**	**0.933** / **1.354** / 49.9%

PR-AUC (Area Under Precision-Recall Curve), and F1-score to measure the performance. Experiment results are reported in Table 9. Method names annotate that the dataset is imputed by which method. The classifier trained on data imputed by SAITS achieves the best results on all evaluation metrics.

Table 9: Results of the downstream classification task on dataset PhysioNet-2012. Performance metrics of methods are calculated by five independent runs. The reported values are means ± standard deviations. The higher, the better. Values in bold font are the best.

Method	ROC-AUC	PR-AUC	F1-score
Median	0.834 ± 0.004	0.460 ± 0.006	0.385 ± 0.031
Last	0.828 ± 0.003	0.469 ± 0.004	0.395 ± 0.024
GRUI-GAN	0.830 ± 0.002	0.451 ± 0.007	0.388 ± 0.020
E^2GAN	0.830 ± 0.002	0.455 ± 0.005	0.356 ± 0.020
M-RNN	0.822 ± 0.002	0.454 ± 0.006	0.388 ± 0.035
GP-VAE	0.834 ± 0.002	0.481 ± 0.007	0.409 ± 0.033
BRITS	0.835 ± 0.001	0.491 ± 0.004	0.413 ± 0.018
Transformer	0.843 ± 0.005	0.492 ± 0.014	0.412 ± 0.019
SAITS (base)	0.846 ± 0.002	0.498 ± 0.004	0.415 ± 0.020
SAITS	**0.848 ± 0.002**	**0.510 ± 0.005**	**0.427 ± 0.028**

5.6 BRITS Trained by the joint-optimization approach

Table 10: Performance comparison between BRITS trained without MIT and with MIT.

Model	PhysioNet-2012	Air-Quality	Electricity
BRITS (w/o MIT)	0.256 / 0.767 / 36.5%	0.153 / 0.525 / 21.6%	**0.847 / 1.322 / 45.3%**
BRITS (w MIT)	**0.251 / 0.691 / 35.8%**	**0.144 / 0.521 / 20.3%**	0.910 / 1.363 / 48.7%

To discuss how our joint-optimization approach can influence the performance of RNN-based models, we apply it in the training of model BRITS and show experimental results in this section.

BRITS models from Table 6 are used here, namely, hyper-parameters are kept exactly the same. The difference between the training way in the original paper [2] and our joint-optimization approach is whether to apply MIT in training. Therefore, we use suffix "w/o MIT" to represent the original training way and suffix "w MIT" to represent our joint-optimization training approach.

As displayed in Table 10, BRITS (w MIT) outperforms BRITS (w/o MIT) on datasets PhysioNet-2012 and Air-Quality, but achieves worse performance on dataset Electricity. Therefore, applying MIT in the training of BRITS can bring further improvement on some datasets, but this is not necessary, and it depends on the dataset. Note that despite BRITS (w MIT) obtains better results on datasets PhysioNet-2012 and Air-Quality, it is still inferior to Transformer and SAITS.

5.7 Ablation Experiments

In this section, we leverage three ablation experiments to discuss the rationality of SAITS algorithm design. The first one in 5.7.1 is to validate the improvement brought by the diagonally-masked self-attention (DMSA). The second one in 5.7.2 is to prove the necessity of the weighted combination. The third one in 5.7.3 is to explain why we do not apply more than two DMSA blocks.

5.7.1 Ablation Study of the Diagonal Mask in Self-Attention

Table 11: Ablation experiment results of the diagonal mask in self-attention. SAITS (base, w/o) is the exact same with SAITS (base), except it is without the diagonal masks in self-attention layers.

Model	PhysioNet-2012	Air-Quality	Electricity
SAITS (base, w/o)	0.200 / 0.446 / 28.5%	0.148 / 0.528 / 21.3%	0.898 / 1.504 / 48.1%
SAITS (base)	**0.192 / 0.439 / 27.3%**	**0.146 / 0.521 / 20.6%**	**0.822 / 1.221 / 44.0%**

To prove that DMSA has better imputation performance than the conventional self-attention, we make a comparison between SAITS (base) and SAITS (base, w/o) in Table 11. SAITS (base, w/o) is without the diagonal mask. SAITS (base) outperforms SAITS (base, w/o) on all datasets. It demonstrates DMSA does improve SAITS' imputation ability.

5.7.2 Ablation Study of the Weighted Combination

Table 12: Ablation experiment results of the weighted combination. SAITS (base, with only 1 block) does not have the second DMSA block nor the weighted-combination block, and its final representation is directly from the only DMSA block. SAITS (base, R2) directly takes **Learned Representation 2** as the final representation, namely, it has no combination of representations. SAITS (base, Res) applies a residual connection to combine **Learned Representation 1 and 2**.

Model	PhysioNet-2012	Air-Quality	Electricity
SAITS (base, with only 1 block)	0.204 / 0.496 / 29.2%	0.178 / 0.544 / 25.1%	0.876 / 1.381 / 46.9%
SAITS (base, R2)	0.199 / 0.451 / 28.4%	0.149 / 0.522 / 21.0%	0.906 / 1.456 / 48.5%
SAITS (base, Res)	0.200 / 0.477 / 28.5%	0.160 / 0.527 / 22.6%	**0.819** / 1.223 / **43.7%**
SAITS (base)	**0.192 / 0.439 / 27.3%**	**0.146 / 0.521 / 20.6%**	0.822 / **1.221** / 44.0%

After applying diagonal masks to self-attention, we continue to think about how to enhance the imputation ability. Therefore, we add the second DMSA block to increase our model's depth and extend the learning process. Rather than simply raising the layer number of the first DMSA block (which can also increase the network depth), we employ the second DMSA block as the second learner to play a role of verification. Different from the first DMSA block that can only make imputation from scratch, the second DMSA block has its input containing the imputed data from the first DMSA block. Therefore, its learning target is to verify these imputation values. However, there is no guarantee that the second DMSA block can perform better than the first DMSA block, namely, the imputations from the second DMSA block are not necessarily better than those from the first DMSA block. For example, SAITS (base, R2) achieves better performance than SAITS (base, with only 1 block) on datasets PhysioNet-2012 and Air-Quality, but performs worse on dataset Electricity. Hence, taking imputation from either block is not wise. Therefore, we let representations from both blocks form the final imputation together, namely the way of the weighted combination described in Section 4.2.5.

To discuss the rationality of the weighted combination, we compare the weighted combination with other two designs to demonstrate the rationality of the weighted combination. As shown in Table 12, one is no combination, directly taking **Learned Representation 2** as the final representation, referring to SAITS (base, R2). The other is the residual combination, which combines **Learned Representation 1 and 2** by a residual connection, referring to SAITS (base, Res).

With results in Table 12, we can see SAITS (base) obtains the best results on both datasets PhysioNet-2012 and Air-Quality. On these two datasets, SAITS (base, Res) is even inferior to SAITS (base, R2), namely, the residual combination makes results worse. On dataset Electricity, SAITS (base) and SAITS (base, Res) achieve comparable results, and both are better than SAITS (base, R2). In summary, our weighted combination is the most practical design in all of the three.

5.7.3 Ablation Study of the Third DMSA Block

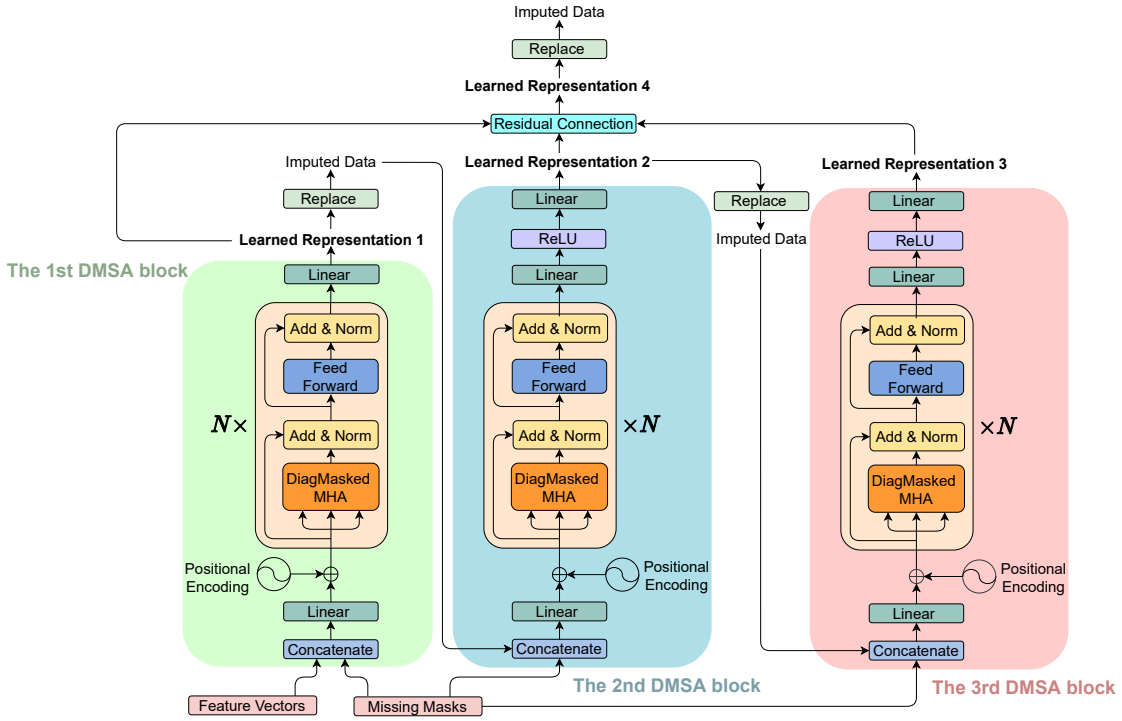
Table 13: Ablation experiment results of the third DMSA block. Results of SAITS here are from Table 6 in this thesis. Both SAITS with three DMSA blocks (residual connected) and SAITS with three DMSA blocks (cascade weighted) apply the same hyper-parameters with SAITS.

Model	PhysioNet-2012	Air-Quality	Electricity
SAITS with three DMSA blocks (residual connected)	0.189 / 0.620 / 27.0%	0.158 / **0.509 / 22.2%**	0.740 / **1.020 / 39.6%**
SAITS with three DMSA blocks (cascade weighted)	**0.185 / 0.418 / 26.4%**	0.146 / 0.512 / 20.5%	0.800 / 1.147 / 42.8%
SAITS	0.186 / 0.431 / 26.6%	**0.137 / 0.518 / 19.3%**	**0.735 / 1.162 / 39.4%**

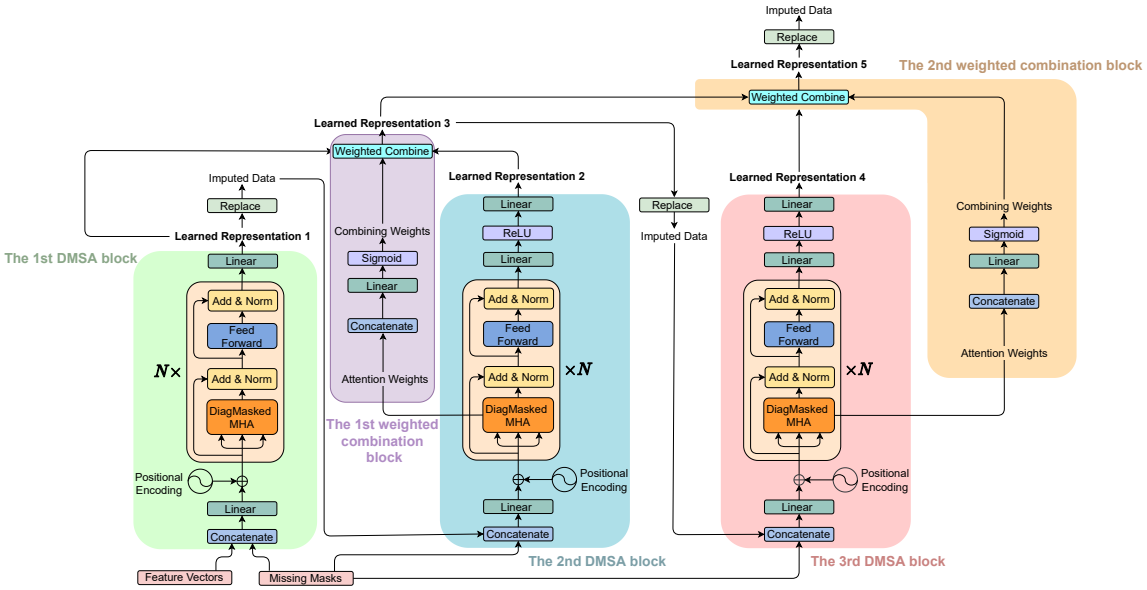
Similar to applying the second DMSA block to obtain better performance, theoretically, we can apply more than two DMSA blocks. However, the benefit is marginal. Taking three DMSA blocks as an example, we conduct experiments and obtain results listed in Table 13 above.

Regarding how to combine representations from three DMSA blocks, we still have two options: residual connection and weighted combination. Residual connection is easy to implement, and SAITS with three DMSA blocks (residual connected) takes this way. The weighted combination can only combine two blocks' representation at a time, so we use cascade-weighted combination here and implement SAITS with three DMSA blocks (cascade weighted). The graphs in Figure 11 are plotted to clearly illustrate both models' structure.

With the results shown in Table 13, we can see, in general, SAITS with three DMSA blocks (residual connected) and SAITS with three DMSA blocks (cascade weighted) do not achieve obviously better results than SAITS, which means adding one more block brings nothing but more parameters and computation resource waste.



(a) SAITS with three DMSA blocks (residual connected)



(b) SAITS with three DMSA blocks (cascade weighted)

Figure 11: Structure illustrations of SAITS with three DMSA blocks.

Chapter 6

Threats to Validity

6.1 Datasets

To increase the robustness of our experimental results, three public real-world datasets used to validate SAITS' performance are selected from three different domains. Additionally, we deliberately make their feature numbers (24, 48, 100) and sequence lengths (37, 132, 370) clear different from each other. Even though, our datasets does not cover all situations and can not represent all kinds of complicated application scenarios in the real world. For example, we do not take spatial-temporal datasets into consideration in this thesis. However, they are also a common form of time series data.

6.2 Modeling

SAITS is based on the original self-attention mechanism proposed in [1]. Therefore, SAITS uses $O(n^2)$ time and space with respect to the sequence length. If samples from the dataset has too long sequence length, this may cause problems such as consuming too much computational power or memory. Solutions proposed in [84, 85] may be worthy of references.

6.3 Experiments

As discussed in Chapter 1, missing values in time series can be MCAR, MAR and MNAR, and this work focuses on the MCAR case. Future work will investigate SAITS' performance in the cases of MAR and MNAR.

Chapter 7

Conclusion

In this thesis, firstly, we study a ILOS-forecasting case from the telecommunication domain to present how to handle missing values with the present machine learning methods. In this case, we develop a deep learning model based on BRITS to process missing values and obtain representation for forecasting. However, we find BRITS' imputation performance is susceptible to the problems of low training speed, memory constraints, and compounding error. To overcome these drawbacks, we propose SAITS, a novel self-attention-based model to impute missing values in multivariate time-series. Specifically, we design a joint-optimization training approach for self-attention-based models to perform on the time-series imputation task. Transformer trained by this approach achieves up to 25% better imputation accuracy to the state-of-the-art (SOTA) models. Moreover, the experimental results in Table 6 show that, compared to BRITS on three real-world datasets, SAITS reduces mean absolute error (MAE) by 12%∼38% and achieves 2.0∼2.6 times faster training speed. SAITS obtains MAE 2%∼13% smaller than Transformer, with comparable training speed of Transformer. Results in Table 8 and 9 further demonstrate that we achieve a new SOTA.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[2] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. BRITS: Bidirectional recurrent imputation for time series. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[3] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, page 95–104, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[5] Tsung-Jung Hsieh, Hsiao-Fen Hsiao, and Wei-Chang Yeh. Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. *Applied Soft Computing*, 11(2):2510–2525, 2011. The Impact of Soft Computing for the Progress of Artificial Intelligence.

[6] Stefan Bauer, Bernhard Schölkopf, and Jonas Peters. The arrow of time in multivariate time series. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2043–2051, New York, New York, USA, 20–22 Jun 2016. PMLR.

[7] Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. *Computing in cardiology*, 39:245, 2012.

[8] Edward Choi, Cao Xiao, Walter Stewart, and Jimeng Sun. Mime: Multilevel medical embedding of electronic health records for predictive healthcare. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[9] Max Horn, Michael Moor, Christian Bock, Bastian Rieck, and Karsten Borgwardt. Set functions for time series. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4353–4363. PMLR, 13–18 Jul 2020.

[10] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[11] Xiuwen Yi, Yu Zheng, Junbo Zhang, and Tianrui Li. ST-MVL: Filling missing values in geo-sensory time series data. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. IJCAI 2016, June 2016.

[12] Ruizhi Deng, Bo Chang, Marcus A Brubaker, Greg Mori, and Andreas Lehrmann. Modeling continuous stochastic processes with dynamic normalizing flows. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7805–7815. Curran Associates, Inc., 2020.

[13] J. Graham. Missing data analysis: making it work in the real world. *Annual review of psychology*, 60:549–76, 2009.

[14] Donald B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

[15] Craig F. Ansley and Robert Kohn. On the estimation of arima models with missing values. In Emanuel Parzen, editor, *Time Series Analysis of Irregularly Observed Data*, pages 9–37, New York, NY, 1984. Springer New York.

[16] D. Kreindler and C. Lumsden. The effects of the irregular sample and missing data in time series analysis. *Nonlinear dynamics, psychology, and life sciences*, 10 2:187–214, 2006.

[17] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 501–508, New York, NY, USA, 2006. Association for Computing Machinery.

[18] D. S. Fung. Methods for the estimation of missing values in time series, 2006.

[19] Melissa J Azur, E. Stuart, C. Frangakis, and P. Leaf. Multiple imputation by chained equations: what is it and how does it work? *International Journal of Methods in Psychiatric Research*, 20, 2011.

[20] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. GAIN: Missing data imputation using generative adversarial nets. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5689–5698. PMLR, 10–15 Jul 2018.

[21] Steven Cheng-Xian Li, Bo Jiang, and Benjamin Marlin. MisGAN: Learning from incomplete data with generative adversarial networks. In *International Conference on Learning Representations*, 2019.

[22] Seongwook Yoon and Sanghoon Sull. GAMIN: Generative adversarial multiple imputation network for highly missing data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[23] Alfredo Nazábal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501, 2020.

[24] Richard Wu, Aoqian Zhang, Ihab Ilyas, and Theodoros Rekatsinas. Attention-based learning for missing data imputation in holoclean. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 307–325, 2020.

[25] Spyridon Mouselinos, Kyriakos Polymenakos, Antonis Nikitakis, and Konstantinos Kyriakopoulos. MAIN: multihead-attention imputation networks. *CoRR*, abs/2102.05428, 2021.

[26] Jinsung Yoon, William R. Zame, and Mihaela van der Schaar. Estimating missing data in temporal data streams using multi-directional recurrent neural networks. *IEEE Transactions on Biomedical Engineering*, 66(5):1477–1490, 2019.

[27] Yonghong Luo, Xiangrui Cai, Ying ZHANG, Jun Xu, and Yuan xiaojie. Multivariate time series imputation with generative adversarial networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[28] Yonghong Luo, Ying Zhang, Xiangrui Cai, and Xiaojie Yuan. E^2GAN: End-to-end generative adversarial network for multivariate time series imputation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3094–3100. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[29] Yukai Liu, Rose Yu, Stephan Zheng, Eric Zhan, and Yisong Yue. NAOMI: Non-autoregressive multiresolution sequence imputation. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[30] Jiawei Ma, Zheng Shou, Alireza Zareian, Hassan Mansour, Anthony Vetro, and Shih-Fu Chang. CDSA: cross-dimensional self-attention for multivariate, geo-tagged time series imputation. *CoRR*, abs/1905.09904, 2019.

[31] Parikshit Bansal, Prathamesh Deshpande, and Sunita Sarawagi. Missing value imputation on multidimensional time series. *CoRR*, abs/2103.01600, 2021.

[32] Vincent Fortuin, Dmitry Baranchuk, Gunnar Raetsch, and Stephan Mandt. GP-VAE: Deep probabilistic time series imputation. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1651–1661. PMLR, 26–28 Aug 2020.

[33] Siddharth Ramchandran, Gleb Tikhonov, Kalle Kujanpää, Miika Koskinen, and Harri Lähdesmäki. Longitudinal variational autoencoder. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3898–3906. PMLR, 13–15 Apr 2021.

[34] M. Ashman, Jonathan So, Will Tebbutt, Vincent Fortuin, Michael Pearce, and Richard E. Turner. Sparse gaussian process variational autoencoders. *ArXiv*, abs/2010.10177, 2020.

[35] Zhengping Che, S. Purushotham, Kyunghyun Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8, 2018.

[36] Francesco Paolo Casale, Adrian Dalca, Luca Saglietti, Jennifer Listgarten, and Nicolo Fusi. Gaussian process prior variational autoencoders. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[37] Mavis D. Boamah. Analysing crisis communication strategies of airline companies in United States: A case study of southwest airline 2016 power outage crisis. *Studies in Media and Communication*, 7:7–16, 2019.

[38] Conor Shine. Cost of southwest's tech outage climbs to at least $54 million. `https://www.dallasnews.com/business/local-companies/2016/08/10/cost-of-southwest-s-tech-outage-climbs-to-at-least-54-million`, 2016.

[39] What is the Adaptive Network? https://www.ciena.com/insights/what-is/What-Is-the-Adaptive-Network.html.

[40] Autonomous Driving Network. https://carrier.huawei.com/en/adn.

[41] Cisco Digital Network Architecture. https://www.cisco.com/c/en_ca/solutions/enterprise-networks/index.html.

[42] The Self-Driving Network. https://www.juniper.net/assets/us/en/local/pdf/pov/3200053-en.pdf.

[43] Towards zero-touch network operations. https://www.ericsson.com/en/blog/2018/9/towards-zero-touch-network-operations.

[44] Blue Planet Network Health Predictor. https://www.blueplanet.com/resources/Blue-Planet-Network-Health-Predictor.html.

[45] HealthBot. https://www.juniper.net/us/en/products-services/sdn/contrail/contrail-healthbot/.

[46] David Côté. Using machine learning in communication networks. *J. Opt. Commun. Netw.*, 10(10):D100–D109, Oct 2018.

[47] Chunpo Pan, Henning Bülow, Wilfried Idler, Laurent Schmalen, and Frank R. Kschischang. Optical nonlinear phase noise compensation for 9×32 -gbaud poldm-16 QAM transmission using a code-aided expectation-maximization algorithm. *Journal of Lightwave Technology*, 33(17):3679–3686, 2015.

[48] Qunbi Zhuge, Xiaobo Zeng, Huazhi Lun, Meng Cai, Xiaomin Liu, Lilin Yi, and Weisheng Hu. Application of machine learning in fiber nonlinearity modeling and monitoring for elastic optical networks. *Journal of Lightwave Technology*, 37(13):3055–3063, 2019.

[49] Takahito Tanimura, Takeshi Hoshida, Tomoyuki Kato, Shigeki Watanabe, and Hiroyuki Morikawa. Convolutional neural network-based optical performance monitoring for optical transport networks. *J. Opt. Commun. Netw.*, 11(1):A52–A59, Jan 2019.

[50] Emmanuel Seve, Jelena Pesic, and Yvan Pointurier. Associating machine-learning and analytical models for quality of transmission estimation: combining the best of both worlds. *J. Opt. Commun. Netw.*, 13(6):C21–C30, Jun 2021.

[51] Cristina Rottondi, Luca Barletta, Alessandro Giusti, and Massimo Tornatore. Machine-learning method for quality of transmission prediction of unestablished lightpaths. *J. Opt. Commun. Netw.*, 10(2):A286–A297, Feb 2018.

[52] Shahin Shahkarami, Francesco Musumeci, Filippo Cugini, and Massimo Tornatore. Machine-learning-based soft-failure detection and identification in optical networks. In *Optical Fiber Communication Conference*, page M3A.5. Optical Society of America, 2018.

[53] Xiaoliang Chen, Baojia Li, Roberto Proietti, Zuqing Zhu, and S. J. Ben Yoo. Self-taught anomaly detection with hybrid unsupervised/supervised machine learning in optical networks. *J. Lightwave Technol.*, 37(7):1742–1749, Apr 2019.

[54] Danish Rafique, Thomas Szyrkowiec, Helmut Grießer, Achim Autenrieth, and Jörg-Peter Elbers. Cognitive assurance architecture for optical network fault management. *J. Lightwave Technol.*, 36(7):1443–1450, Apr 2018.

[55] Boyuan Yan, Yongli Zhao, Sabidur Rahman, Yajie Li, Xiaosong Yu, Dongmei Liu, Yongqi He, and Jie Zhang. Dirty-data-based alarm prediction in self-optimizing large-scale optical networks. *Opt. Express*, 27(8):10631–10643, Apr 2019.

[56] Haotao Zhuang, Yongli Zhao, Xiaosong Yu, Yajie Li, Ying Wang, and Jie Zhang. Machine-learning-based alarm prediction with gans-based self-optimizing data augmentation in large-scale optical transport networks. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 294–298, 2020.

[57] Zhilong Wang, Min Zhang, Danshi Wang, Chuang Song, Min Liu, Jin Li, Liqi Lou, and Zhuo Liu. Failure prediction using machine learning and time series in optical network. *Opt. Express*, 25(16):18553–18565, Aug 2017.

[58] K. Christodoulopoulos, C. Delezoide, N. Sambo, A. Kretsis, I. Sartzetakis, A. Sgambelluri, N. Argyris, G. Kanakis, P. Giardina, G. Bernini, D. Roccato,

A. Percelsi, R. Morro, H. Avramopoulos, P. Castoldi, P. Layec, and S. Bigo. Toward efficient, reliable, and autonomous optical networks: the ORCHESTRA solution. *J. Opt. Commun. Netw.*, 11(9):C10–C24, Sep 2019.

[59] Rui Morais. On the suitability, requisites, and challenges of machine learning [invited]. *J. Opt. Commun. Netw.*, 13(1):A1–A12, Oct 2020.

[60] Zhizhen Zhong, Nan Hua, Zhigang Yuan, Yanhe Li, and Xiaoping Zheng. Routing without routing algorithms: An AI-based routing paradigm for multi-domain optical networks. In *2019 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 2019.

[61] Yongli Zhao, Boyuan Yan, Dongmei Liu, Yongqi He, Dajiang Wang, and Jie Zhang. SOON: self-optimizing optical networks with machine learning. *Opt. Express*, 26(22):28713–28726, Oct 2018.

[62] Thierry Zami, Bruno Lavigne, Ivan Fernandez de Jauregui Ruiz, Marco Bertolini, Yuan-Hua Claire Kao, Oriol Bertran Pardo, Mathieu Lefrançois, Florian Pulka, Sethumadhavan Chandrasekhar, Junho Cho, Xi Chen, Di Che, Ellsworth Burrows, Peter Winzer, Jelena Pesic, and Nicola Rossi. Simple self-optimization of WDM networks based on probabilistic constellation shaping. *J. Opt. Commun. Netw.*, 12(1):A82–A94, Jan 2020.

[63] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016.

[64] Marek Smieja, undefinedukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. Processing of missing data by neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2724–2734, Red Hook, NY, USA, 2018. Curran Associates Inc.

[65] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[66] Jiakai Yu, Weiyang Mo, Yue-Kai Huang, Ezra Ip, and Daniel C. Kilper. Model transfer of qot prediction in optical networks based on artificial neural networks. *J. Opt. Commun. Netw.*, 11(10):C48–C57, Oct 2019.

[67] Le Xia, Jing Zhang, Shaohua Hu, Mingyue Zhu, Yingxiong Song, and Kun Qiu. Transfer learning assisted deep neural network for osnr estimation. *Opt. Express*, 27(14):19398–19406, Jul 2019.

[68] Juan Zhao, Sachin Shetty, J. Pan, C. Kamhoua, and K. Kwiat. Transfer learning for detecting unknown network attacks. *EURASIP Journal on Information Security*, 2019:1–13, 2019.

[69] Dario Azzimonti, Cristina Rottondi, Alessandro Giusti, Massimo Tornatore, and Andrea Bianco. Active vs transfer learning approaches for qot estimation with small training datasets. In *Optical Fiber Communication Conference (OFC) 2020*, page M4E.1. Optical Society of America, 2020.

[70] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2004.

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[72] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[73] W. L. Taylor. Cloze Procedure: A new tool for measuring readability. *Journalism & Mass Communication Quarterly*, 30:415 – 433, 1953.

[74] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[75] Z. Yang, Zihang Dai, Yiming Yang, J. Carbonell, R. Salakhutdinov, and Quoc V. Le. XLNet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

[76] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. DISAN: Directional self-attention network for rnn/cnn-free language understanding. In *AAAI Conference on Artificial Intelligence*, 2018.

[77] Joongbo Shin, Yoonhyung Lee, Seunghyun Yoon, and Kyomin Jung. Fast and accurate deep bidirectional language representations for unsupervised learning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 823–835, Online, July 2020. Association for Computational Linguistics.

[78] Jimmy Ba, J. Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

[79] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[80] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and S. Chen. Cautionary tales on air-quality improvement in beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473, 2017.

[81] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[82] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[83] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dtextquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[84] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768, 2020.

[85] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities. In *WACV*, 2021.