TU WIEN Informatics

Energy-efficient IoT by Continual Learning for Data Reduction in Wireless Sensor Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Alexander Falzberger, BSc
Matrikelnummer 01273054

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Ivona Brandić

Wien, 26. Jänner 2023

Alexander Falzberger	Ivona Brandić

TU WIEN Informatics

Energy-efficient IoT by Continual Learning for Data Reduction in Wireless Sensor Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Alexander Falzberger, BSc
Registration Number 01273054

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Ivona Brandić

Vienna, 26th January, 2023

Alexander Falzberger	Ivona Brandić

Erklärung zur Verfassung der Arbeit

Alexander Falzberger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 26. Jänner 2023

Alexander Falzberger

Acknowledgements

I express my deepest gratitude to Professor Ivona Brandić, who has been an invaluable source of support and guidance throughout my time working on this thesis. Professor Brandić has provided me with valuable encouragement and advice, and I am grateful for her mentorship. I would also like to thank Shashikant Ilager for his invaluable feedback and counsel in times of need.

I am grateful to my family for their support and encouragement throughout my studies. My parents have always been there for me, their love and guidance being a constant source of strength and motivation.

Above all, I would like to thank Klara for her unwavering love and support from when I decided to study informatics until this very day. Klara has been a continuous source of joy and happiness in my life, and I am sincerely grateful to have her by my side.

This work has been supported by a netidee scholarship.

Kurzfassung

Das Internet der Dinge (Internet of Things, IoT) hält immer mehr Einzug in unser tägliches Leben und Sensoren ermöglichen, kostengünstig die Umwelt zu erfassen und darauf zu reagieren. Diese Sensoren haben jedoch häufig nur beschränkte Ressourcen, da sie auf Batterien als Energiequelle angewiesen sind und nur drahtlose Kommunikation zur Verfügung haben, was zu einem höheren Energieverbrauch und auch zu einer Überlastung des Netzwerks führen kann. Die Reduzierung der erforderlichen Kommunikation der Sensoren erhöht ihre Lebensdauer, reduziert die Wartungskosten, und schont Netzwerkressourcen.

Diese Arbeit präsentiert SenseReduce, ein Open Source Framework für Datenreduktion durch Datenvorhersage mit kontinuierlichen Modellaktualisierungen in IoT-Umgebungen. Das Framework ermöglicht die Verwendung neuronaler Netze als multivariate Prognosemodelle, die den Kommunikationsaufwand durch die Vorhersage zukünftiger Sensormessungen mit hoher Genauigkeit erheblich reduzieren können.

Um das Frameworks zu evaluieren, führen wir eine Parameterstudie durch, die auf dem Anwendungsfall der Lufttemperaturüberwachung basiert. Wir entwerfen drei neuronale Netze unterschiedlicher Grüße und Komplexität unter Verwendung von Hyperparameter-Optimierung, und führen mehrere Simulationsszenarien mit verschiedenen Trainingsdatensätzen und Parameterkonfigurationen durch. Die quantitative Auswertung der Simulationsergebnisse zeigt die Effizienz von SenseReduce durch Reduzierung der erforderlichen Nachrichten um bis zu 38% und der Gesamtmenge der übertragenen Daten um bis zu 22% im Vergleich zum Referenzwert eines Prognosemodells basierend auf Trendextrapolation. Kontinuierliche Modellaktualisierungen reduzieren die insgesamt übertragenen Daten in Szenarien mit geringen Trainingsdaten und verbessern die Prognosegenauigkeit im Laufe der Zeit. Darüber hinaus zeigen unsere Simulationen, dass die Effizienz des Frameworks kontextabhängig ist, insbesondere vom Messintervall, der definierten Schwellenwertmetrik und der Modellarchitektur. Wir fassen unsere Erkenntnisse in Empfehlungen für zukünftige Anwendungen der Datenreduktion durch Datenvorhersage, insbesondere im Kontext der Lufttemperaturüberwachung, zusammen.

Abstract

The Internet of Things (IoT) is becoming increasingly prevalent in our daily lives, with sensor devices providing the ability to sense and act on the environment. However, these sensor nodes are often resource-constrained, relying on batteries as an energy source, and constrained by wireless communication, which results in higher energy consumption and network congestion. Reducing required network communication in these devices increases their lifetime, reduces maintenance costs, and preserves network resources.

To tackle this challenge, we propose SenseReduce, an open-source framework for prediction-based data reduction with continual learning and continuous deployment of prediction models in IoT environments. The framework enables using neural networks as multivariate prediction models, which can significantly reduce the required communication by predicting future sensor measurements with high accuracy.

To evaluate the performance of the proposed framework, we conduct a parameter study based on the use case of air temperature monitoring. We design three neural networks with varying architectural and computational complexities using hyperparameter optimization and run multiple simulation scenarios using different training datasets and parameter configurations. The quantitative evaluation of the simulation results demonstrates the effectiveness of SenseReduce in reducing the required messages by up to 38% and the amount of transferred data by up to 22% compared to a baseline using trend extrapolation. Continuous model updates reduce the total data transferred in scenarios with sparse training data and improve model performance over time. Additionally, our simulations show that the effectiveness of prediction-based data reduction highly depends on the context, including measurement frequency, threshold metric, and model architecture. Overall, we provide several recommendations for future applications, particularly for air temperature monitoring.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The proliferation of Internet of Things (IoT) devices in recent years has led to a rapid increase in the number of interconnected devices. According to Ericsson, the number of IoT connections is expected to reach 30.2 billion by 2027 [Eri22]. IoT enables the interconnection of physical devices, vehicles, buildings, and other items embedded with sensors, software, and connectivity, to collect and exchange data, enabling resource sharing, analysis, and management across various industries [LYZ+17]. Wireless *sensor nodes*, which are small and low-cost computing devices equipped with radio antennas and sensors, are an essential component of IoT systems. The constantly increasing performance and decreasing cost of these sensor nodes facilitate ever-expanding fields of application: industrial IoT [SSH+18], precision agriculture [JNG+17], IoT for healthcare [ZB21], and smart cities [PQE+18], for example, are vibrant research areas.

These sensor nodes are capable of sensing a wide range of environmental parameters and are often deployed in the form of *Wireless Sensor Networks* (WSNs) in areas that are difficult for humans to access. Within WSNs, sensor nodes are typically deployed in large numbers in a mesh topology and are used to collect and transmit data wirelessly [ACFP09]. Sensor nodes often rely on batteries as energy sources because the environment does not provide an alternative (e.g., rural environments), or the nodes must be mobile (e.g., body sensors). Reducing energy consumption increases the lifetime of sensor nodes and reduces maintenance costs. Additionally, a lower energy demand implies a lower carbon footprint, a desirable goal by itself in the face of the present climate crisis [RDK+22].

Due to their constrained computing power and hardware limitations, sensor nodes cannot execute high-complexity algorithms. In order to overcome these limitations, many IoT systems rely on devices located in physical proximity to act as gateways between sensor nodes and the wider internet. The distributed computing paradigm of edge

computing brings computing power and data storage closer to sensor nodes or "edge" of a network [DZF+20]. These edge devices are crucial for processing, analyzing, and extracting valuable information from the collected datasets. Still, for wireless sensor nodes, the energy demand of the communication subsystem is significantly higher than that of the computation subsystem, and communication is the single-most energy-demanding task [ACFP09]. Hence, reducing the required communication in resource-constrained WSNs increases the energy efficiency of sensor nodes and is essential for efficacious IoT applications.

The reduction of energy consumption in WSNs has been the subject of numerous studies, resulting in the development of various techniques that operate at different levels of the communication stack [ACFP09]. Data-driven approaches, such as data prediction or data compression, leverage the functionality provided by lower-layer protocols, such as medium access control (MAC) and routing protocols. Furthermore, recent advancements in energy supply methods for sensor nodes, such as energy harvesting and wireless power transfer [SK11], have also significantly improved the overall energy efficiency of WSNs. In this thesis, we focus on *prediction-based data reduction*, where sensor nodes use prediction models to forecast future measurements and do not communicate as long as the difference between predictions and measurements does not violate a defined threshold.

Efforts to increase energy efficiency in WSNs should not be considered mutually exclusive alternatives but complementary techniques that can be utilized together. In this regard, incorporating data prediction into WSNs can provide an additional means of optimizing transmissions in a way neither data compression nor routing protocols can. Prediction-based data reduction eliminates the need for communicating data, reducing exchanged messages and total data transfer without compromising the quality of the measurements made by sensor nodes.

Another critical challenge in wireless networks is medium access control due to the growing number of wireless devices and traffic profiles [BBBK16]. Dense networks are particularly affected, leading to wireless unreliability, channel collision, network congestion, and other issues. These problems can cause transmission failure, increasing energy consumption for retransmission. Additionally, the inefficiency of routing protocols due to the broadcast nature of wireless communications also contributes to higher energy consumption [MZC+21]. Reducing the number of transferred messages in WSNs can alleviate these issues by requiring fewer simultaneous open connections.

In the past, the computational limitations of sensor nodes have imposed constraints on the complexity of prediction algorithms that can be implemented [DBO16]. However, recent advancements in technology, such as TinyML [WS20], have enabled the deployment of neural networks (NNs) on even low-power microcontroller units, enabling the use of NNs for model inference on sensor nodes. This integration of state-of-the-art machine learning techniques holds significant potential for enhancing IoT applications' performance, efficiency, and privacy in general. For prediction-based data reduction, NNs can improve prediction accuracy and thus increase energy efficiency.

One of the challenges associated with prediction-based data reduction is the need for the chosen model to be suitable for representing the phenomenon of interest. Furthermore, using NNs for prediction often requires significant training data to achieve acceptable model performance. This necessitates an a-priori dataset collected by the sensor node, which may not always be feasible. To overcome this limitation, transfer learning [ZQD+21] is a commonly employed technique to improve the performance of NNs by leveraging knowledge from related source domains. This allows for training models with acceptable performance using minimal available training data.

However, deployed models are also subject to issues such as insularity and non-stationarity of the data, which can lead to degradation of the model performance over time, rendering them ineffective or even causing catastrophic failures [AEKB20]. Non-stationarity refers to a phenomenon in which the statistical properties of a time series data change over time and can arise from various factors, including seasonal or periodical effects, thermal drift or aging effects in sensors, hardware or software faults in cyber-physical systems, or changes in user behavior [DRAP15]. Therefore, the need to compute and deploy updated NN models becomes apparent in such scenarios.

Continual learning (CL) addresses the challenge of computing model updates, given its definition of being "the ability to continually learn over time by accommodating new knowledge while retaining previously learned experiences" [PKP+19]. However, training NNs for CL requires significant computational resources and benefits from specialized hardware, such as GPUs, which are often unavailable on IoT devices. Additionally, the accumulation of data from multiple sensor nodes can be exploited to improve the performance of prediction models through the use of additional knowledge.

A distributed CL approach can be used to overcome these limitations, where a central base station handles the computation and deployment of model updates. This can be implemented in various ways, such as utilizing edge devices or cloud servers: The use of edge devices reduces the amount of data transmitted to the cloud, thus reducing communication costs and increasing privacy. However, the use of cloud servers allows for the use of powerful computational resources to train NNs with large datasets, leading to improved model performance. The decision of which approach to use should be based on the application's specific requirements.

Either way, the deployment of NNs to sensor nodes in WSNs poses a significant challenge in terms of data transfer for the sensor node, as it requires a trade-off between improved accuracy and the associated transfer costs. By utilizing better predictions, sensor nodes can further reduce the required communication. However, the net effect of this improvement on overall data transmission and energy efficiency depends on the specific details of the application and the cost of the model transfer. Our proposed prediction-based data reduction is a novel approach that combines NNs as prediction models with continuous model updates in long-term scenarios.

1.2 Contribution

In this thesis, we propose a novel framework, SenseReduce, for prediction-based data reduction with continual learning and continuous deployment of prediction models in IoT systems. The aim of SenseReduce is to enable data reduction using NN-based prediction models in monitoring applications with multivariate measurements. We design an adaptive data reduction algorithm using a dual prediction mechanism and a pull-based communication protocol for sensor nodes to enable high duty-cycling.

We extend the state-of-the-art approaches for prediction-based data reduction by including mechanisms for continual learning and continuous model deployment. Additionally, we focus on customizability by implementing SenseReduce with a modular architecture and abstract interfaces, allowing others to adapt and use the framework in different application scenarios. The implementation of SenseReduce is open-source and available on GitHub[1].

To evaluate the performance of SenseReduce, we conduct a parameter study over a 2-year period, where the deployment of a single sensor node for air temperature monitoring is simulated. We analyze the effects of different training datasets, prediction models, and hyperparameter values on the performance of SenseReduce in terms of prediction accuracy, total transferred data, and the number of exchanged messages. Additionally, we demonstrate the advantages of transfer learning in environments with minimal available training data. In total, we run and analyze 756 scenarios and compare NN-based approaches to baseline implementations, highlighting their power and limitations and providing recommendations for future applications of prediction-based data reduction in IoT systems.

The novel contributions of this thesis are summarized as follows:

- We propose a novel framework, SenseReduce, for prediction-based data reduction with continual learning and continuous deployment of prediction models in IoT systems.
- We design and evaluate different NN-based prediction models for short-term air temperature forecasting, considering their performance, architectural complexity, and computational complexity.
- We present a novel error metric, Weighted Mean Squared Error, as a loss function optimized for training models in applications with prediction-based data reduction.
- We conduct extensive simulations to investigate the impact of different model update strategies and application parameters on top of SenseReduce and provide a comprehensive analysis of their strengths and weaknesses.

[1] `github.com/falzberger/SenseReduce`

1.3 Methodologies

In this thesis, we propose an algorithm for prediction-based data reduction that uses a dual prediction scheme to reduce the amount of data that needs to be transmitted in WSNs. A modular framework was implemented motivated by the use case of air temperature monitoring but allowing for customization to different application areas.

The data reduction algorithm is based on a prediction model, for which we designed three different neural network architectures (MLP, LSTM, CNN) using hyperparameter optimization. The models were evaluated in terms of architectural and computational complexity to find the best trade-off between accuracy and resource usage.

Finally, a parameter study was conducted using simulation data for air temperature monitoring to evaluate the impact of different parameters on the performance of the prediction models. Rehearsal-based continual learning in the form of retraining and transfer learning with freezing layers and fine-tuning was applied to further improve the performance of the prediction models over time.

1.4 Outline

The thesis is organized as follows: In Chapter 2, we provide the theoretical background of the methods used in this research. Chapter 3 reviews related work and state-of-the-art approaches for prediction-based data reduction in WSNs and cost-efficient continuous model deployments.

Our main contributions are presented in the following chapters: Chapter 4 describes the SenseReduce framework and the working principles of the data reduction algorithm. Chapter 5 explains the experimental setup, including the datasets, evaluation metrics, and the model design used in our simulation study. In Chapter 6, we present and discuss the results of our simulations.

Finally, we conclude in Chapter 7 with a summarizing view back on our results and forward into prospective future research directions.

CHAPTER 2

Preliminaries

In this chapter, we present the theoretical foundations of our research. We begin with prediction-based data reduction techniques for energy-efficient wireless sensor networks, introducing dual prediction schemes. We discuss the use of neural networks, emphasizing time series forecasting using Long Short-Term Memory (LSTM) and convolutional networks. Additionally, we introduce the concepts of continual learning and transfer learning, which enable models to adapt to changing data and improve their performance over time. These concepts and techniques are crucial for comprehending the contributions and limitations of our proposed approach.

2.1 Dual Prediction Schemes

Wireless sensor networks (WSNs) are composed of multiple sensor nodes connected to a central base station. The connection is bidirectional: It can be used to collect measurement data from and distribute information to the sensor nodes. Continuous data transfer between sensor nodes results in massive communication overhead and overall high energy consumption.

Data reduction plays an essential role in data-driven approaches for energy conservation in WSNs [ACFP09]. Additional techniques like adaptive sampling, duty cycling, and mobile base stations target other layers in a sensor node's technological stack and can be viewed as complementary. The approaches to data reduction in WSNs are manifold, with prediction-based data reduction reducing required communication to a subset of the measured values without compromising their accuracy [DBO17]. In this thesis, we refer to *prediction* as the process of estimating future values based on historical data, also known as time series forecasting, in contrast to inferring missing values in a dataset.

A *dual prediction scheme* (DPS) utilizes a prediction model deployed on both the sensor node and the base station. This allows both devices to generate predictions of future

values using historical data from the sensor node. The sensor node uses a threshold metric to evaluate the discrepancy between the predicted and measured values to decide whether to transmit its measurements to the base station. If the threshold metric is not violated, the sensor node does not transmit data, thus reducing the amount of communication and increasing energy efficiency. In cases where the sensor node does not transmit data, the base station assumes that the value obtained from the prediction model is within the required error bound. In the case of a threshold violation, the sensor node immediately informs the base station, and the prediction model may be adapted.

In the field of time series forecasting, various methods are available for modeling and predicting future values of a given time series. One popular class of methods is based on the autoregressive (AR) and moving average (MA) models, which can be combined to form the autoregressive-moving average (ARMA) model. These models are beneficial for forecasting time series data that exhibit temporal dependencies and are stationary, i.e., their statistical properties do not change over time.

However, not all time series data are stationary; thus, alternative methods are required for forecasts in nonstationary environments. One such method is the autoregressive integrated moving average (ARIMA) model, which has been shown to reduce communication for sensed data by Li et al. [LGS09]. Despite their effectiveness, the ARIMA model and its subclasses have some limitations. One major limitation is that they cannot capture non-linear patterns in the data. Additionally, these models do not support forecasting multivariate time series data. Both issues are addressed by neural networks, which we will introduce in the next section.

2.2 Neural Networks

Neural networks (NNs) are a powerful class of machine learning algorithms widely used in various application areas. This is due to their ability to discover complex relationships among data, robustness to data uncertainty, and ability to predict multivariate output values in near real-time. However, a significant drawback of NNs is the need for a large amount of representative data to train an accurate forecasting model, which can be challenging to obtain in certain situations. Despite the complexity of their underlying fundamentals, the use of NNs in practice has become increasingly feasible due to the proliferation of open-source libraries and a growing community of experts, along with a high economic demand for their application in various fields.

2.2.1 Multilayer Perceptron

A NN is a computational model composed of layers of interconnected neurons, which are mathematical functions with multiple inputs. The perceptron is the simplest form of a neuron [Ros57]. Given inputs $x_1, \ldots, x_n$, the perceptron computes the weighted sum of all inputs using weights $w_1, \ldots, w_n$, including a bias b. The sum is the input for an activation function $f : \mathbb{R} \to \mathbb{R}$, whose result is the output of the perceptron. In short,

the computation of a single perceptron is given by:

$$y = f(\sum_{i=1}^{n} w_i x_i + b). \tag{2.1}$$

Perceptrons utilize various activation functions f to introduce non-linearity in the model. The selection of an optimal activation function is an active area of research and can often be determined through a process of experimentation and evaluation using a variety of different variants. A commonly applied activation function is the sigmoid logistic function, defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.2}$$

which maps any given input value to the range $[0, 1]$. The hyperbolic tangent function has a similar S-shaped curve but maps to the larger range $[-1, 1]$ and is given by

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.3}$$

The weighting parameters are determined through a process known as training, which is typically performed through several iterations, referred to as *epochs*. During training, the weighting parameters are updated to optimize the network's performance. Multiple algorithms are available for implementing the training process, with the Adam algorithm [KB14] being one of the most widely used. The optimization is performed by minimizing a chosen *loss function*, which is a mathematical expression that quantifies the difference between the predicted value of the network and the actual value. Various loss functions are used in neural network training, with the choice of the loss function depending on the specific task and the nature of the data.

Models in which the data is processed linearly from the input layer through one or more hidden layers to the output layer are referred to as *feed-forward neural networks*. Figure 2.1 shows an example of such a network. Additionally, the layers in this network architecture are fully connected, meaning that each neuron in a given layer is connected to every neuron in the preceding layer. Such layers are also called *dense* layers, and networks of multiple dense layers are referred to as Multilayer Perceptrons (MLPs).

Given a neural network model $\mathcal{M}$, its architectural complexity $p(\mathcal{M})$ is defined by its number of parameters [Won19]. For dense layers consisting of perceptrons, the number of parameters is given by the number of input weights and biases for every perceptron. Given a dense layer $Dense$ with N_{out} perceptrons and N_{in} input units we thus have:

$$p(Dense) = N_{in} \cdot N_{out} + N_{out} = N_{out}(N_{in} + 1). \tag{2.4}$$

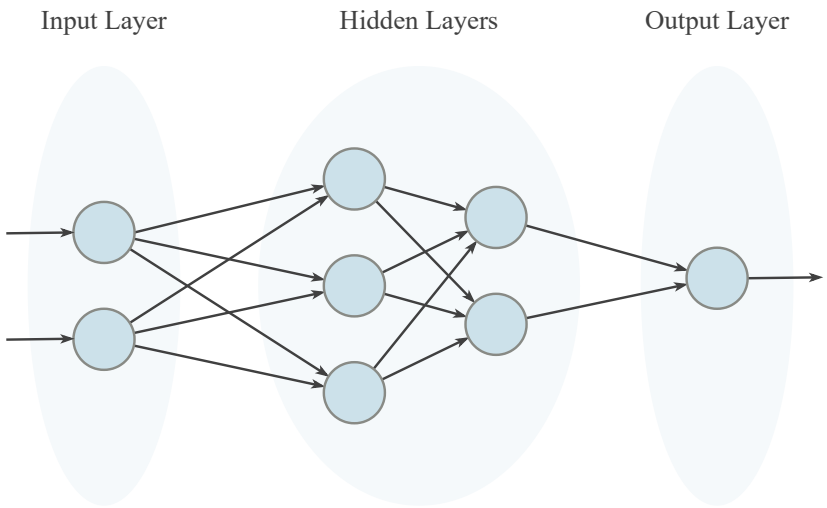


Figure 2.1: Simplified model of a feed-forward neural network with dense layers

The computational efforts for network inference are attributed mainly to the computation of multiply-accumulate (MAC) operations. The number of MAC operations $m(\mathcal{M})$ can be considered as an accurate proxy for quantifying the computational complexity of a neural network $\mathcal{M}$ [Won19]. A MAC operation is a computational step that simultaneously computes the product of two values and accumulates the result into a third value, as represented by the equation $a \leftarrow a+(b \cdot c)$. For a dense layer $Dense$ with N_{out} perceptrons and N_{in} inputs, MAC operations allow perceptrons to accumulate and multiply the inputs by the weights simultaneously, resulting in

$$m(Dense) = N_{out}(N_{in} + 1) = p(DL). \tag{2.5}$$

2.2.2 Long Short-Term Memory

In the context of prediction-based data reduction, the measurements collected by sensor nodes are usually represented as a time series. Sensor nodes can monitor multiple physical phenomenons simultaneously, leading to *multivariate* time series data. Recurrent neural networks (RNNs) are a class of neural networks that are particularly well suited for modeling time series data. RNNs have the ability to incorporate previous outputs as inputs and maintain an internal state that allows them to capture temporal dependencies in the data.

The Long Short-Term Memory (LSTM) network, a specific type of RNN introduced by Hochreiter et al.[HS97], has been shown to be an effective model for prediction-based data reduction in environmental monitoring in the work of Shu et al.[SCBdS19], as detailed in Section 3.2. The ability of LSTMs to effectively capture long-term temporal dependencies in sequential data has been identified as a key factor of their success in this application.

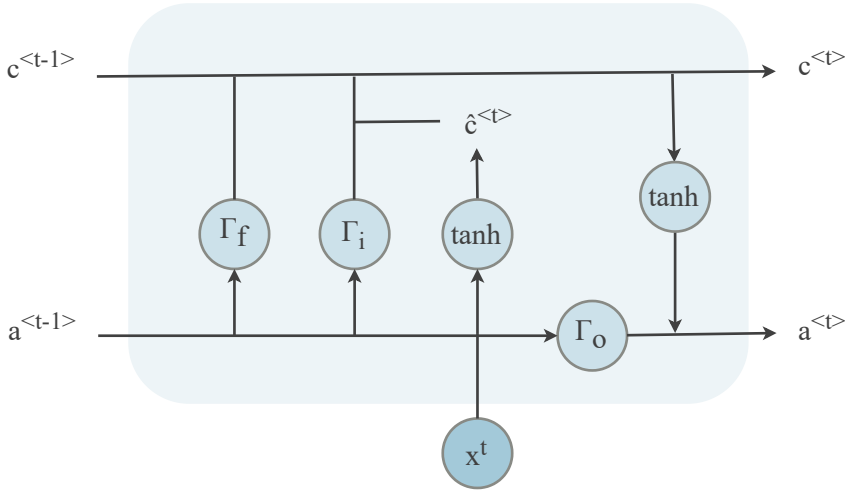


Figure 2.2: Schematic architecture of a Long Short-Term Memory cell

Figure 2.2 illustrates the schematic architecture of a single LSTM cell. Unlike traditional perceptrons, LSTM cells generate two distinct values through a series of mathematical operations and non-linear activations. The first value is the cell state, $c^{<t>}$, and is responsible for carrying and storing information over an extended period of time. The second value, $a^{<t>}$, represents the cell's output. In contrast to feed-forward neural networks, LSTM cells compute a cell output for each time step t of the input sequence x.

An LSTM cell has three main gates: the input gate Γ_i, the forget gate Γ_f, and the output gate Γ_o. Each gate controls the flow of information into and out of the cell state. The output of a single gate unit given input $x^{<t>}$ and previous cell output $a^{<t-1>}$ is defined by the following equation:

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b), \tag{2.6}$$

where W and U are gate-specific input weights, b is an additive bias, and σ the sigmoid activation function as described in Equation 2.2.1. The gates work together to allow LSTM cells to selectively write, read, and forget information over time, which enables them to maintain long-term dependencies in sequential data.

Additionally, the hyperbolic tangent function (Equation 2.2.1) is used to compute the intermediate cell state $\hat{c}^{<t>}$ similar to the other gates:

$$\hat{c}^{<t>} = tanh(W_c x^{<t>} + U_c a^{<t-1>} + b_c), \tag{2.7}$$

where W_c, U_c and b are another set of trainable coefficients. Next, the new cell state $c^{<t>}$ is computed with

$$c^{<t>} = \Gamma_i \circ \hat{c}^{<t>} + \Gamma_f \circ c^{<t-1>}, \tag{2.8}$$

where $\circ$ is the element-wise multiplication, also known as the Hadamard product. Finally, the cell output $a^{<t>}$ is computed using the hyperbolic tangent function:

$$a^{<t>} = \Gamma_o \circ tanh(c^{<t>}). \tag{2.9}$$

A single LSTM cell contains a significantly greater number of parameters than a traditional perceptron. Each gate can be conceptualized as a dense layer with $N_{in} + N_{out}$ input units and N_{out} output units. As a result, the architectural complexity of an LSTM layer is significantly higher than that of a dense layer with the same number of output units. Specifically, for an LSTM layer $LSTM$ with N_{in} input units and N_{out} output units, the number of parameters can be calculated using the following equation:

$$p(LSTM) = 4 \cdot (N_{out}((N_{in} + N_{out}) + 1)) \tag{2.10}$$

Furthermore, the computational complexity in terms of MAC operations also differs from Dense layers, as the Hadamard products must be considered. Additionally, the computational complexity is dependent on the length of the input sequence N_t, as for every $x^{<t>}, 1 \leq t \leq N_t$, a new cell output will be computed, which leads to the following equation:

$$m(LSTM) = 4 \cdot (N_{out}((N_{in} + N_{out}) + 1)) + 2 \cdot N_{out} + N_{out}, \tag{2.11}$$

where every gate requires MAC operations as computed in Equation 2.2.1, and the Hadamard products of Equation 2.2.2 and 2.2.2 require $2 \cdot N_{out}$ and N_{out} MAC operations, respectively. This highlights that LSTMs, compared to dense layers, have a higher computational complexity with similar architectural complexity, as the additional operations required to maintain the temporal dependencies in sequential data increase the number of operations required.

2.2.3 Convolutional Neural Networks

Recent studies [TBKV21, LLS20] have demonstrated the efficacy of using Convolutional Neural Networks (CNNs) for air temperature predictions, particularly with forecast horizons up to 24 hours. CNNs are characterized by their ability to share parameters through the use of convolutional layers, which are able to extract features from multidimensional input data [LBBH98]. Multivariate time series data can be represented as a two-dimensional matrix, where the column axis corresponds to the attributes and the row axis to the temporal dimension. As highlighted in [KMS20], the use of CNNs is motivated by the high degree of correlation among individual measurements and the time-invariant property of convolutional kernels, which enables the extraction of relevant features from the time series data.

A key aspect of CNNs is the convolution layer, which employs a kernel K that is convolved with the input data I to construct a *feature map* O. The kernel is a small two-dimensional filter used to extract features from the input data by scanning it with a stride S and applying an activation function g at each position. A convolution layer consists of multiple filters C, each with its own kernel $K^1, \ldots, K^C$. In this thesis, we use of CNNs with stride

$S = 1$ and zero-padding, such that the feature map is given by:

$$O = (o_{i,j})_{1 \leq i \leq I_x, 1 \leq j \leq C} = g(\sum_{x=1}^{K_x} \sum_{y=1}^{I_y} X_{i+x-1,y} \cdot K_{x,y}^{j} + b), \tag{2.12}$$

where the input X is of size $I_x \times I_y$, the kernel K of size $K_x \times I_y$, and $g(\cdot)$ is an activation function that often is the identity function $g(x) = x$. Similar to the other layer types, a bias term b is added.

In CNNs, parameter sharing is used, where a single kernel is convolved with the input data, resulting in a relatively low architectural complexity. Given a layer $Conv$ with C filters of size $K_x \times I_y$, the number of parameters is given by:

$$p(Conv) = C \cdot K_x \cdot I_y \tag{2.13}$$

However, the computational complexity of CNNs is relatively high, as the number of computations in which a single kernel is involved depends solely on the input data size. Given input data I of size $I_x \times I_y$ the number of MAC operations is

$$m(Conv) = C \cdot I_x \cdot (K_x \cdot I_y + 1). \tag{2.14}$$

2.2.4 Overfitting

In neural networks, overfitting occurs when the model becomes too intricate and starts to adapt to the random variations or noise present in the training data. This leads to poor generalization and decreased performance on new, unseen data. Methods to mitigate overfitting include regularization techniques such as *dropout* and *pooling* layers.

Dropout is a regularization technique that randomly drops out (i.e., sets to zero) a certain proportion of neurons during the training process [SHK+14]. This helps to prevent the co-adaptation of neurons, which can lead to overfitting. The dropout rate, i.e., the proportion of neurons that are dropped out, is a hyperparameter that needs to be set before training the model.

Pooling or subsampling layers reduce the spatial dimensions of the feature maps, which also helps prevent overfitting by reducing the number of parameters in the network [LBBH98]. For instance, *max pooling* works by taking the maximum value from a small window of the feature map. As a result, the model is less sensitive to small translations in the input data, which makes the model more robust and less prone to overfitting.

Another way to prevent overfitting is to use a lower learning rate. The learning rate determines the step size of the optimization algorithm, and a lower learning rate means that the model updates less aggressively during training. This can help to prevent the model from fitting to the noise in the training data.

2.3 Continual Learning

In many applications of neural networks, the goal is to optimize performance on a fixed dataset. However, in the context of wireless sensor networks (WSNs), the dataset continuously expands with new data samples from measurements. This dynamic nature of the dataset can lead to the phenomenon of *concept drift* [DRAP15], where the characteristics of the data change over time. For example, user interests in social networks may adapt to new trends, or climate conditions may shift due to climate change.

Algorithms for addressing concept drift can be classified in various ways, as outlined in [EP11], such as online and batch algorithms or active and passive approaches. Online algorithms, which process one instance at a time, demonstrate enhanced plasticity but inferior stability properties. This results in frequent adjustments to the model parameters, making them infeasible for dual prediction schemes in WSNs. In contrast, batch algorithms benefit from utilizing more extensive amounts of data and possess superior stability properties. Active approaches incorporate a drift detection mechanism and only update the model when drift is detected. On the other hand, passive approaches assume the possibility of steady drift and continuously update the model with new data. As reported in [DRAP15], passive approaches are generally more appropriate for batch learning.

The phenomenon of a model's performance on previously learned tasks deteriorating as it adapts to new data is commonly referred to as *catastrophic forgetting* [KAM+18]. Neural networks suffer from catastrophic forgetting independent from their architecture [SG19]. Continual learning (CL) is the process of neural networks continually adapting to changing input distributions over time. Balancing the need for model adaptation, or plasticity, with the need for model stability is the primary challenge in CL and is often referred to as the *stability-plasticity dilemma* [MBB13]. Various techniques such as regularization methods, architectures, and memory replay have been proposed to address this problem in the literature [PKP+19].

Maltoni and Lomonaco [ML19] have proposed a fuzzy categorization of continual learning (CL) strategies, which can be broadly grouped into three types: architectural strategies, regularization strategies, and rehearsal strategies. Each strategy type comes at different costs, particularly in IoT and edge environments, which we will briefly summarize.

Architectural strategies involve adapting the structure of the model, such as adding new layers or re-training on a new architecture. This type of strategy relies on flexible resource allocation, which can be challenging in IoT environments due to the limited memory and computational power available on these devices. For instance, in [EP11], an ensemble of models is applied, which is continuously extended with new models.

Regularization strategies constrain the extent to which weights in the neural network can change, for example, by differentiating their contributions to already learned tasks. This can lower the memory requirements by only relying on new data for adapting model weights. However, CL algorithms generally are polynomial-time heuristics to

solve an NP-hard problem and require perfect memory for an optimal solution [KHD20]. Consequently, regularization strategies are still prone to catastrophic forgetting in many scenarios [LSF19].

Rehearsal strategies selectively maintain past information and use it for periodic replays to strengthen already learned weights [HKB+21]. Ven et al. [vdVT19] find that rehearsal-based approaches are most reliable across different CL scenarios. They have greater requirements in terms of memory as they require maintaining a representation of previous training data, which can exceed the capabilities of an IoT device [IC18]. This highlights the importance of centralizing model training on base stations, where specialized hardware and centralized data aggregation can be leveraged to support the additional memory requirements of rehearsal strategies.

The taxonomy introduced in [ML19] classifies CL scenarios into two dimensions: the type of task and the type of data updates. In terms of tasks, CL scenarios can be Multi Task (MT), Single Incremental Task (SIT), or Multi Incremental Task (MIT). MT scenarios involve incoming tasks that are different from the previous ones, whereas SIT scenarios involve the same task. In contrast, MIT scenarios involve new and old tasks that may be interleaved and presented again to the model.

Task-incremental (MT, MIT) scenarios assume task labels are available for each sample. However, it is difficult to acquire explicit task labels for each sample in many real-world applications, particularly at test time. Therefore, Single Incremental Task (SIT) scenarios, which do not rely on task labels, are commonly used [CCLB21].

In terms of data updates, there can be New Classes (NC), New Instances (NI), or a combination of both, referred to as New Instances and Classes (NIC). NC scenarios introduce new classes, whereas NI scenarios involve new examples of previously seen classes. In the case of NIC scenarios, new instances and new classes are presented to the model simultaneously.

In the context of time series forecasting, the task of the model is typically a regression task. The focus is on making predictions based on new instances of the same time series rather than introducing new classes or new and unseen instances of classes. In contrast, classification tasks, such as image or speech recognition, involve predicting a categorical value, and the focus is on learning new classes and new instances of previously seen classes. Therefore, in time series forecasting, the notion of classes is less relevant, and the NI scenario is the most applicable. Although research on CL for classification tasks has yielded promising results [RKSL17], the field of CL for regression tasks, such as time series forecasting, is still in its early stages of development [HS21].

2.4 Transfer Learning

The data obtained from sensor nodes in WSNs highly depends on their physical location. This can make it challenging to train NNs with high performance in situations with limited historical data. The cost of acquiring extensive training data before deploying the initial

model may be prohibitively high in practical scenarios. Continual learning strategies can be employed to improve performance after deployment; however, acquiring sufficient data may take considerable time to achieve acceptable performance for prediction-based data reduction. *Transfer learning* (TL) can be an effective solution to this problem by leveraging knowledge acquired from other related tasks to improve the performance of the target task [ZQD+21].

Transfer learning is a technique that utilizes the knowledge gained from one task, known as the source task, to enhance the performance of a new task, referred to as the target task. This is done by utilizing the learned features or representations from the source task as a foundation for the target task, instead of starting the learning process from scratch. Transfer learning can effectively reduce the amount of training data required for the target task and improve the generalization performance of a model as the features or representations learned from the source task are likely to be more robust and generalizable than those learned from a limited amount of training data in the target task. The specific implementation of transfer learning depends on the type of model and the nature of the source and target tasks.

One of the most common techniques used in transfer learning is *freezing layers*. This method involves freezing the weights of specific NN layers, considered to be task-invariant, and only training the remaining layers on the new task. This allows the model to retain the knowledge acquired from previous tasks while adapting to new tasks. The frozen layers act as a prior, providing a good initialization for the new task, while the unfrozen layers can adapt to the new task.

Another transfer learning technique is *fine-tuning*. This technique involves training the entire neural network on the new task but using a lower learning rate than the one used for the previous task. This allows the model to adapt to the new task while retaining the knowledge acquired from previous tasks. The lower learning rate ensures that the model does not forget the previous task by making too significant adjustments to the weights.

2.5 Overview

Property	**ARIMA**	**MLP**	**LSTM**	**CNN**	**TL**	**CL**
Univariate Prediction	•	•	•	•		
Multivariate Prediction		•	•	•		
Non-Linearity		•	•	•		
Temporal Dependencies	•		•			
Limited Training Data	•				•	•
Concept Drift						•

Table 2.1: Comparison of relevant methods for data prediction

In Table 2.1, an overview of the methods for data prediction presented in this chapter is provided. We highlight the aspects each method addresses in WSNs. While capable of incorporating temporal dependencies with limited training data, statistical models, such as ARIMA, are limited to univariate time series and cannot fully exploit the additional information present in multivariate time series. Conversely, prediction models based on different neural network architectures (MLP, LSTM, CNN) can utilize this additional information. LSTM networks are specifically designed to consider temporal dependencies. However, these models require a substantial amount of training data. Both continual learning (CL) and transfer learning (TL) techniques can be employed to address this issue. CL allows updating the model over time, integrating new knowledge, while TL leverages knowledge from similar domains. Altogether, integrating NN-based prediction models with CL and TL leads to the desired system properties.

CHAPTER 3

Related Work

Our proposed prediction-based data reduction framework is a novel approach that combines NN-based prediction models with continual learning and continuous deployment in IoT environments. This chapter will first provide an overview of the state-of-the-art for energy-efficient continuous model deployment before discussing different approaches to prediction-based data reduction with dual prediction schemes in IoT environments.

3.1 Continuous Model Deployment

Today, many cloud vendors offer solutions for state-of-the-art machine learning (ML) operations that aim to continuously integrate, deliver, and train ML systems. For example, Google's Vertex AI[1], Amazon SageMaker[2], and Azure Machine Learning[3] all provide fully automated ML platforms that include automated deployments to IoT devices. However, although they automate many required steps, these solutions are not designed for data reduction in sensor applications and do not support dual prediction schemes.

Continuum [TYW18] represents a general-purpose platform for updating and deploying models in a continual learning setting. The platform is language-agnostic, i.e., it provides an abstraction layer that allows using heterogeneous ML frameworks and supports pluggable and customizable model update strategies. However, the provided default strategies only focus on incorporating data as quickly as possible and reducing training costs; they do not consider network communication or model performance.

Similarly, Derakshan et al. [DMRM19] propose a framework that includes advanced sampling techniques for continual learning of ML models and automatically deploys

[1] `cloud.google.com/vertex-ai`
[2] `docs.aws.amazon.com/sagemaker`
[3] `azure.microsoft.com/services/machine-learning`

updated models. However, the deployment policy only targets low latency and does not consider model performance or communication costs.

Aral et al. [AEKB20] present a network-aware scheduling algorithm for ML model updates called Staleness Control for Edge Data Analytics (SCEDA), based on reinforcement learning. The proposed algorithm is specifically designed for nonstationary environments that exhibit concept drift and intermittent connectivity, where data from edge nodes is collected to train a central ML model. The algorithm considers the model's performance through a data variety metric, which acts as an estimate for model generalization. The primary goal of SCEDA is timeliness and fault tolerance, and the evaluation results demonstrate that it can effectively minimize the age of information. However, the algorithm does not consider the transfer cost of model updates, nor is it designed for sensing applications.

Edge MLOps [RBWA21] is a fully automated framework for "Machine Learning Operations at the Edge". The authors focus on the design of a complete ML framework, including continuous model monitoring, retraining, and deployment, with state-of-the-art technologies. The deployment decision is based on estimating future model performance using a pre-defined error metric. The scenario of indoor air quality prediction is chosen to evaluate the framework. The authors only validate the stability of the framework and neither show an improved model performance in general nor consider communication cost.

The optimization framework introduced in [JZXJ22] aims for cost efficiency with a holistic view of edge and IoT application systems. The framework optimizes data processing, transmission, and deployment decisions, given a cloud-edge system with potentially multiple applications deployed on a single edge node. The system model is generic because it allows retraining models on the edge and the cloud, depending on resource limitations. The framework's goal is to (1) minimize cloud and edge resource costs; and (2) maximize data throughput for best-possible model performance. Unfortunately, the framework does not consider communication costs in its cost model. The authors argue that compressed ML models have minimal model sizes, making their transfer cost negligible. However, this argument does not apply to sensor nodes in WSNs with limited bandwidth and energy limitations.

The Stratum framework [BBK+19] provides life cycle management for ML in IoT analytics. It provides a unified end-to-end framework for designing, building, and deploying ML models. Likewise, the framework in [MKL19] constitutes a fully automated ML pipeline and includes a sophisticated model selection approach to select the best model for an IoT device based on collected data samples. However, neither of these frameworks has any continual learning aspects, i.e., once a model is deployed, there is no performance monitoring, automated retraining, or continuous deployment.

The Deep-Edge framework [BCS+20] distributes the computation of model updates. Some nodes might have access to GPU-enabled hardware optimized for neural network computations in a heterogeneous network of edge nodes. Deep-Edge tries to optimize

the selection of nodes participating in the computation to minimize computational load and resource interference while increasing fault tolerance. “Latent Replay” [PGLM20] is a technique for enabling continual learning with a rehearsal strategy directly on embedded devices. Similarly, [DV21] extends the TensorFlow Lite framework with continual learning capabilities to enable continual learning on edge devices. However, those frameworks do not consider the possibility of computing a model update in the cloud. Thus, centralized data collection and model optimization across multiple inference nodes is impossible. Moreover, in resource-constrained environments, distributed update computations pose a heavy burden regarding the required data transmission.

In [DBT+19], the authors introduce a reference architecture for a complete ML pipeline life-cycle, including model retraining and deployment. However, their work comprises only an outline for possible implementations, with no specific strategies or cost models in place.

3.2 Dual Prediction Schemes

Dias et al. [DBO16] conduct a comprehensive survey of prediction-based data reduction techniques in WSNs. They propose a taxonomy for existing approaches and provide an in-depth analysis of the application of different prediction models, including statistical and probabilistic methods, as well as machine learning techniques utilizing NNs. The survey also covers approaches for different WSN topologies, including those with multiple clusters and cluster heads as intermediaries between sensor nodes and base stations that can apply prediction models and data fusion for data reduction. In this section, we will focus on recent literature that specifically addresses prediction models for sensor nodes, and that has been published since the survey was conducted.

Fathalla et al. [FSM+22] propose using ensemble learning as a prediction model in a dual prediction scheme for wireless sensor networks. The model employs an automatic retraining mechanism based on the number of miss-predicted observations. The proposed approach is evaluated with a simulation based on a univariate time series over seven days with a sampling interval of 1 minute. The results demonstrate that ensemble learning outperforms simpler techniques such as the least mean square (LMS) filter [SR06]. However, the authors note that there is no clear trend on when to update the prediction model, and they do not consider the cost of model transfer in their analysis.

In [MdSBF21], the authors apply three recurrent neural networks and one CNN for multivariate data prediction in dual prediction schemes in WSNs. The efficacy of the models is evaluated with a simulation over 14 days with a sampling interval of 5 minutes, where the best-found model predicts the next 40 minutes using an input sequence of length 256. The threshold metric has been defined as the absolute difference between predicted and measured temperature. The study also examines the impact of varying threshold values on the performance of the models, with a correlation observed between lower threshold values and improved average accuracy, albeit at the cost of increased transmission rates, and vice versa for higher threshold values.

Shu et al. [SCBdS19] propose a hybrid model in a dual prediction scheme to minimize the number and overall duration of transmissions. The hybrid model consists of a gradient descent-based Least Mean Square filter (GD-LMS) in combination with an LSTM network. The GD-LMS filter is designed to have low computational complexity, reduced storage requirements, and aims for fast convergence speed while maintaining high prediction accuracy; however, it is limited to univariate prediction. The proposed approach is evaluated using a univariate indoor air temperature dataset over one day, and the study demonstrates the effect of different threshold metrics on the average prediction accuracy.

Moghadam and Keshmirpour [MK11] combine an autoregressive integrated moving average (ARIMA) model with a neural network (NN) for data prediction. The ARIMA model is used for predictions as long as the threshold metric is not violated. Once the error exceeds the threshold, the NN is employed to generate input for the ARIMA model that aims to represent the non-linear trend of the data. The base station is informed of this change in input data through a beacon signal that is significantly smaller in size compared to an update packet. The results of the evaluation, which used datasets from three different days for indoor temperature monitoring, demonstrate that the proposed hybrid model outperforms either of its individual components.

Yu et al. [YWS17] utilize a dual prediction scheme with Kalman filters (KF) for data reduction in IoT environments. The generated IoT data is sent to fog platforms and is deemed redundant if it falls within the predicted range. Otherwise, the predictions are treated like measurements. The sensor data is modeled as a multivariate normal distribution based on historical data determined by statistical properties. The approach is shown to outperform baselines based on the Grey model and Autoregressive and Moving Average (ARMA) model for indoor temperature, humidity, and brightness monitoring in terms of accuracy and the number of transmitted packets. Deng et al. [DGLZ19] propose a similar approach in fog computing and base their data reduction on an autoregressive model (AR), limited to univariate prediction. Jain et al. [JAK20] propose the extended linear regression technique (ELR), where an autoregression (AR) model is combined with a cosine distance method to incorporate the trend of the current measurement using past measurements, showing similar results.

Santini and Römer [SR06] also propose a least mean square (LMS) adaptive algorithm for a dual prediction mechanism in temperature measurement. The adaptive algorithm provides a tracking capability and is thus able to adapt to a nonstationary environment. With a threshold metric implemented as an absolute difference of 0.5 Celsius, they achieve a communication reduction of up to 92% over three days for indoor air temperature and humidity monitoring compared to constant reporting. Similarly, Fathy et al. [FBT18] developed an Adaptive Method for Data Reduction (AM-DR) based on a convex combination of two decoupled LMS filters with different memory sizes and could increase the reduction rate in the same setting to 95%.

Tayeh et al.[TMLD18] combine a dual prediction scheme with an adaptive sampling rate to further extend the lifetime of WSNs. They use a univariate linear prediction model based on the last value and a computed additive weight. The proposed approach is

evaluated over 35 days for indoor temperature monitoring, and the results are compared to the Data Prediction and Adaptive Sampling (DPCAS) approach introduced by Monteiro et al.[MDP+17]. The evaluation results demonstrate that both the proposed and the DPCAS approaches can effectively reduce data transmission, with minimal differences depending on the observed attribute.

In the study by Salim et al. [SM21], an examination of time series forecasting for temperature and humidity measurements in precision agriculture is conducted to achieve prediction-based data reduction. The authors employ a trend computation approach utilizing previous measurements in both a machine learning algorithm and a correlation-based prediction algorithm that leverages inter- and intra-node data correlation of temperature and humidity measurements. The effectiveness of the proposed approach is evaluated using temperature data over three days with a sampling interval of 30 minutes, and the results demonstrate a decrease in transmission rate and energy consumption in each sensor node. Razafimandimby et al. [RLVN17] apply a Bayesian Inference Approach (BIA) for outdoor weather monitoring, leveraging the spatio-temporal correlated data of densely deployed nodes in agriculture networks. They focus on heterogeneous data generated by IoT devices, utilizing temperature data to infer humidity over an evaluation period of eighteen hours.

Almalki et al. [AOAS21] propose the Energy Efficient Routing Protocol using Dual Prediction Model (EERP-DPM) for up to 17 biomedical sensors in the context of healthcare IoT. The proposed protocol combines prediction-based data reduction with a routing protocol that assigns a priority to each packet based on whether it is related to a medical emergency. However, the authors do not explain the prediction model used in the proposed protocol in detail.

Publication	**UI**	**MI**	**CL**	**TM**	**MC**
Fathalla et al. [FSM+22]	•		•	•	
Morales et al. [MdSBF21]	•	•		•	
Shu et al. [SCBdS19]	•		•	•	
Moghadam and Keshmirpour [MK11]	•				•
Yu et al. [YWS17]	•	•	•	•	N/A
Santini and Römer [SR06]	•		•		N/A
Fathy et al. [FBT18]	•		•	•	N/A
Tayeh et al.[TMLD18]	•				N/A
Salim et al. [SM21]		•	•		N/A
SenseReduce	•	•	•	•	•

Table 3.1: Qualitative assessment of the relevant literature

In Table 3.1, a qualitative assessment of selected studies from the literature in the field

of prediction-based data reduction is presented concerning key design considerations and evaluation methodologies. The former include the challenges addressed by the proposed SenseReduce framework, such as support for univariate input (UI), multivariate input (MI), and continual learning (CL). The latter include evaluation metrics such as varying threshold metrics (TM) and the consideration of model transfer costs (MT). The table uses a symbol notation, where the presence of a bullet (•) indicates that the work addresses the criterion, a gap indicates otherwise, and (N/A) indicates that the criterion does not apply to the selected work.

This thesis conducts an in-depth examination of the cost-performance trade-offs of prediction-based data reduction over an extended period of 2 years. We extend the state-of-the-art by analyzing the performance of different neural network architectures as prediction models combined with various strategies for continual learning and continuous model deployment in different application scenarios. This research aims to provide a comprehensive understanding of the trade-offs and potential optimizations that can be made in deploying NN-based prediction models with continuous model updates for data reduction in IoT environments.

CHAPTER 4

SenseReduce Framework

This chapter presents our proposed prediction-based data reduction algorithm based on a dual prediction scheme for energy-efficient IoT. Based on this algorithm, we propose SenseReduce, an open-source, state-of-the-art framework that supports continual learning and continuous model deployment. The implementation of SenseReduce is motivated by the use case of air temperature monitoring, and we will discuss its components and interfaces in detail.

4.1 Data Reduction Algorithm

We introduce common notation before discussing the working principles of SenseReduce in detail: Let a sensor node collect measurements $x \in \mathbb{R}^f$ with a defined measurement interval ρ. Each measurement x consists of $f \geq 1$ measured values and has an associated timestamp t_x.

Prediction-based data reduction requires forecasts provided by a prediction model $\mathcal{M}$. Given input matrix $X \in \mathbb{R}^{k \times l}$, model $\mathcal{M}$ outputs a prediction matrix $Y \in \mathbb{R}^{m \times n}$. The dimensions of inputs and outputs can vary depending on the application. Moreover, the number of input features l can differ from the number of measured values f (e.g., through feature engineering) and from the number of predicted features n. However, every predicted value must have a corresponding measurement value, i.e., $n \leq l$, and the function $\mu : \mathbb{N} \to \mathbb{N}$ maps the prediction column index to its corresponding measurement column index.

In both matrices, each row $x_{i,*}$ represents (predicted) measurements correlated with timestamp t_{x_i}. Moreover, the rows are sorted by their timestamps in ascending order, i.e., $t_{x_i} < t_{x_j}$ for all $i < j$.

The model predictions $Y \in \mathbb{R}^{m \times n}$ range from timestamp t_{y_1} to t_{y_n}. For prediction-based data reduction, SenseReduce constructs the *prediction horizon* $H \in \mathbb{R}^{(m+1) \times n}$ by

adding the last measurement $x_{k,*}$ of input matrix $X \in \mathbb{R}^{k \times l}$ as first element to the predictions Y:

$$H = (h_{ij})_{1 \leq i \leq (m+1), 1 \leq j \leq n} = \begin{cases} x_{k,\mu(j)} & \text{if } i = 1 \\ y_{i-1,j} & \text{if } i > 1 \end{cases} \tag{4.1}$$

Clearly, the prediction horizon must contain at least two rows, i.e., $m + 1 \geq 2$. We denote the *prediction horizon length* η of model $\mathcal{M}$ as the length of the prediction interval $[t_{h_1}, t_{h_{m+1}}]$.

Given two rows $x_{i,*}, x_{j,*}$, the difference between their corresponding timestamps t_{x_i}, t_{x_j} is called the data aggregation period δ. For simplicity, we assume that δ is an integer multiple of the measurement interval ρ, i.e., $mod(\delta, \rho) = 0$. Intuitively, $\eta = \delta * m$ holds for any model predictions $Y \in \mathbb{R}^{m \times n}$, and thus $m = \eta / \delta$.

A threshold metric $\mathcal{T}$ is a Boolean function with two inputs: a measurement $x \in \mathbb{R}^f$ and a prediction $\hat{y} \in \mathbb{R}^n$. Depending on the application context, the threshold metric defines when the prediction deviates from the measurement to such an extent that the base station must be notified.

4.1.1 Prediction Interpolation

For every sensor node measurement x at timestamp t_x, prediction-based data reduction requires a prediction $\hat{y}$ as input to the threshold metric $\mathcal{T}$. However, as defined above, the measurement interval ρ can be smaller than the data aggregation period δ. Hence, it can be the case that the prediction horizon H does not contain a row $h_{i,*}$ with timestamp $t_{h_i} = t_x$.

Linear interpolation is used to compute predictions for all timestamps that do not match the timestamp of a row in the prediction horizon: Given prediction horizon $H \in \mathbb{R}^{(m+1) \times n}$ with prediction interval $[t_{h_1}, t_{h_{m+1}}]$ and data aggregation period δ, the prediction $\hat{y}_t$ at timestamp t is given by

$$\hat{y}_t = h_{i,*} \cdot \frac{t - t_{h_i}}{\delta} + h_{j,*} \cdot \frac{t_{h_j} - t}{\delta}, \tag{4.2}$$

where i and j are the smallest and largest indices, respectively, such that $t_{h_i} < t_{h_j}$ and $t_{h_i} \leq t \leq t_{h_j}$ hold.

4.1.2 Prediction Horizon Adjustment

As soon as the sensor node collects a measurement that causes a threshold violation or is not within the current prediction interval, the prediction model $\mathcal{M}$ is used to compute a new prediction horizon. However, if measurement x_t at time t causes a threshold violation within the first and second timestamp of the prediction horizon H, i.e., $t_{h_1} \leq t < t_{h_2}$, no new prediction horizon can be created as no new data samples have been collected. Still,

it is desirable to improve the current prediction horizon based on the new information. To this end, we adjust the prediction horizon H with

$$H = H \oplus (x_t - \hat{y}_t), \tag{4.3}$$

where $\oplus$ is the addition of $(x_t - \hat{y}_t)$ to each row $h_{i,*}, 1 \leq i \leq (m+1)$. By shifting the prediction horizon to match the current measurement, we decrease the number of threshold violations within the current data aggregation period as it incorporates the most recent observations prior to the execution of the prediction algorithm.

4.1.3 Sensor Node

After introducing the essential concepts of the dual prediction scheme, we now present the data reduction mechanism from the sensor node's perspective: In Algorithm 4.1, we list the algorithm executed on a single sensor node. Initially, the node registers itself at the base station by informing it about its threshold metric $\mathcal{T}$ and receives its prediction model $\mathcal{M}$ and the first prediction horizon H as a result. The measurement process of the sensor node does not have a stopping criterion; hence we use an infinite loop with a sleep timer of the interval ρ for the node to collect and evaluate measurements.

Algorithm 4.1: Data Reduction Algorithm on Sensor Node

Input: Base station $\mathcal{B}$, threshold metric $\mathcal{T}$, data aggregation period δ, and measurement interval ρ

```
1  M, H ← register_at_base_station(B, T);
2  while node is running do
3      t ← current_datetime();
4      x_t ← current_measurement();
5      if not in_prediction_horizon(H, t) then
6          data ← data_since_last_event(δ);
7          M ← send_update(B, t, data);
8          H ← update_prediction_horizon(M, t, data);
9      end
10     ŷ_t ← predict(H, t);
11     if is_threshold_violation(T, x_t, ŷ_t) then
12         data ← data_since_last_event(δ);
13         M ← send_violation(B, t, data, x_t);
14         H ← update_prediction_horizon(M, t, data);
15         H ← adjust_prediction_horizon(H, t, x_t, ŷ_t);
16     end
17     sleep(ρ);
18 end
```

In Line 5, the sensor node checks whether the current measurement is still within the prediction horizon. If not, the node first collects all measurements acquired since the last prediction horizon has been computed with the data aggregation interval δ; for instance, if δ is set to one hour, then `data` contains hourly measurements. The collected data are then sent to the base station $\mathcal{B}$ with the current timestamp in Line 7 before a new prediction horizon is computed in Line 8.

In Line 10, the prediction horizon H is used to compute the prediction $\hat{y}_t$ for the current timestamp t according to Equation 4.1.1. In case a threshold violation is detected in Line 11, the node again retrieves the aggregated data and communicates the violation to the base station in Line 13. After updating the prediction horizon, the node adjusts it to the measurement as described in Equation 4.1.2 in Line 15.

When communicating threshold violations or horizon updates (Lines 7 and 13), the node can receive a new model from the base station. This depends on the active model update strategy in the base station. If it does not assign a new model to the node, the functions return the currently active prediction model $\mathcal{M}$ instead.

In the background, the sensor node maintains a limited-size buffer that contains the most recent measurements. The prediction horizon gives the lower bound of the buffer size, i.e., the buffer must be large enough to hold all measurements within one horizon. The upper bound is the length of the input sequence for the prediction model $\mathcal{M}$.

4.1.4 Example

To provide a clear understanding of the prediction-based data reduction algorithm, we present an example in Figure 4.1. The plot illustrates the application of the algorithm for one week in a sensor node with a measurement interval of $\rho = 10$ minutes. During the visualized period, the sensor node collected $7 \cdot 24 \cdot 6 = 1008$ measurements; however, it only transmitted 13 messages to the base station, consisting of 9 threshold violations and 4 horizon updates.

The prediction horizon length is $\eta = 24$ hours, as demonstrated by the horizon updates from July 4 to July 6. Since the predictions match the measurements without violating the threshold, no communication is required between these horizon updates. In the case of threshold violations, we can observe that the predictions are immediately shifted due to either computing a new prediction horizon or adjusting the existing one. Furthermore, the first threshold violation on July 2 resulted in an updated model immediately utilized to compute a new prediction horizon.

4.2 Architecture

After presenting the prediction-based data reduction algorithm, we will delve into the implementation details of the proposed SenseReduce framework. The framework not only supports prediction-based data reduction but also incorporates the capabilities for

Figure 4.1: Example of prediction-based data reduction over one week for a univariate measurement, where the threshold metric is defined using the absolute difference between measurement and prediction.

continual learning and continuous deployment of prediction models. The implementation of the framework is motivated by the use case of temperature monitoring, a typical application area for WSNs, which we will also use as a case study for our simulation-based evaluations in Chapters 5 and 6.

In Figure 4.2, we provide a Unified Modeling Language (UML) deployment diagram of the proposed SenseReduce framework. The diagram illustrates the functional components of the framework and their interactions. The central component of the framework is the *base station*, which is responsible for managing the communication and data aggregation of potentially multiple *sensor nodes* located at the network edge. This design represents a standard deployment of WSNs but can also be applied to IoT environments where sensor devices are connected to edge nodes or cloud-based processing units. For instance, in smart agriculture [ERO+18], multiple battery-driven sensor nodes can be deployed on an agricultural field to monitor the micro-climate. The base station can predict the current measurements with the accuracy defined by the threshold metric without requiring any energy-intensive communication with the sensor nodes.

The central component of each sensor node is its Monitor, which is responsible for running the data reduction algorithm as described in Algorithm 4.1. A sensor node can consist of multiple sensors, i.e., hardware units responsible for collecting measurements from the physical environment with a Sensing Unit. For every measurement, the Monitor retrieves a prediction from the Predictor and uses it together with the Threshold Metric for data reduction.

The base station contains a *node manager*, which maintains a representation of each sensor node connected to it with a Node component. Essentially, a Node is a virtual copy of a deployed sensor node, including its Predictor and all collected measurement data. The node manager uses the DeploymentStrategy and ContinualStrategy to manage continuous model updates for every node, which we will explain in Section 4.3.3.

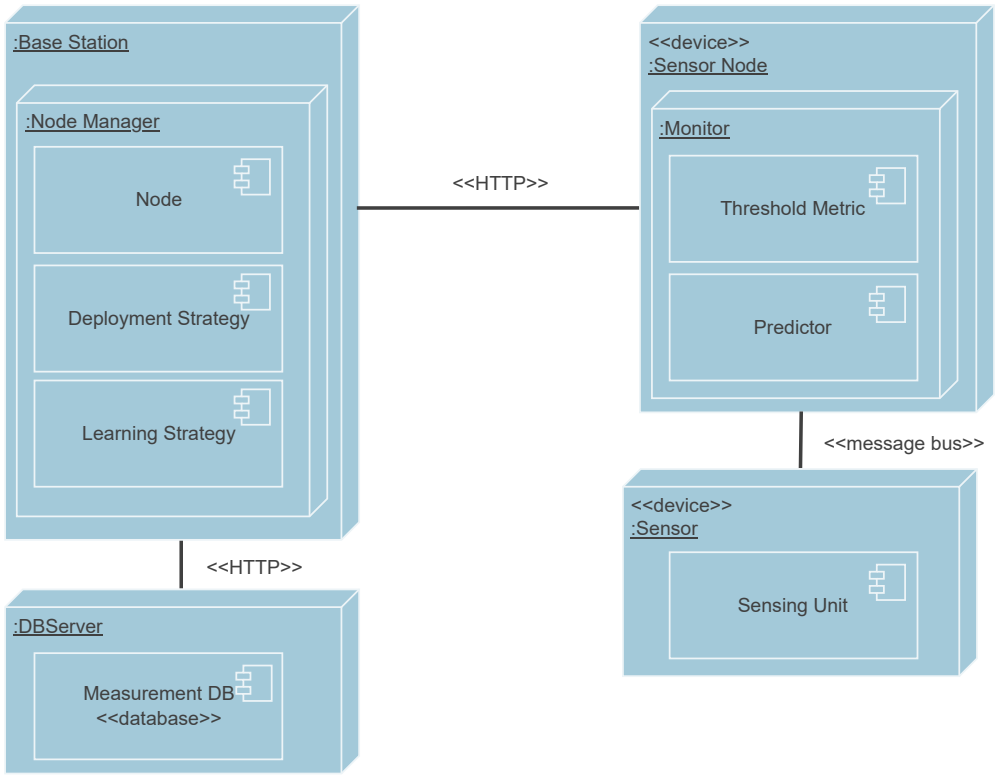


Figure 4.2: SenseReduce deployment diagram

The design goal of SenseReduce is to be adaptable for different application scenarios. To this end, the framework relies on the design principle of abstraction. We discuss the interfaces of the customizable classes in Section 4.3.

4.2.1 Software Dependencies

We implemented SenseReduce using the Python programming language and made the source code freely available via an open-source repository[1]. The dependencies of the framework are listed in Table 4.1, and our choices are briefly discussed as follows:

- We chose Python as a commonly used language for data analysis and machine learning tasks. The open-source libraries `NumPy` and `pandas` provide high-performance implementations for data transformation and are widely supported in the data science community.
- `TensorFlow` is used as a machine learning framework due to its support for TensorFlow Lite, which enables easy conversion of models for inference on mobile, embedded, and edge devices.

[1] `github.com/falzberger/sense-reduce`

- The `Flask` and `requests` libraries were used for implementing network communication efficiently.

Python 3.9	programming language
NumPy 1.23.4 [HMvdW+20]	scientific computing library
pandas 1.5.1 [WM10]	data analysis and manipulation tool
TensorFlow 2.11 [ABC+16]	open-source machine learning framework
Flask 2.2.2[2]	web application framework
requests 2.28.1[3]	HTTP request library

Table 4.1: Software dependencies of SenseReduce

4.2.2 Communication

The primary objective of the SenseReduce framework is to optimize energy efficiency by reducing the communication between sensor nodes and the base station. To accomplish this, the framework employs a limited set of communication interfaces consisting of only four distinct message types. Additionally, the framework utilizes a pull-based approach for sensor node communication, where the sensor nodes initiate all communication and do not have to maintain a constant connection with the base station. This pull-based approach enables the sensor nodes to power down their communication hardware during idle periods, referred to as duty cycling [ACFP09], resulting in increased energy efficiency. Furthermore, this approach also helps to mitigate communication congestion and reduce the overall power consumption in the sensor nodes, which is a crucial factor in IoT applications where the sensor nodes are battery-powered.

In detail, SenseReduce facilitates communication between sensor nodes and the base station using the following message types:

- **Register**: After start-up, a sensor node registers itself at the base station by sending its threshold metric. The base station assigns a new unique identifier to the sensor node and returns it together with the initial data required for computing the first prediction horizon.

- **Get Model**: Due to pull-based communication, the sensor node fetches the prediction model from the base station itself. By sending the base station its unique identifier, the sensor node receives its currently assigned prediction model as a response. The base station reacts by also changing its prediction model currently employed in the dual prediction scheme.

[2] `github.com/pallets/flask`
[3] `github.com/psf/requests`

- **Threshold Violation**: The sensor node immediately informs the base station when a measurement causes a threshold violation. In addition to the current timestamp and measurement, the message contains measurement data to compute a new prediction horizon as described in Section 4.1.3. Depending on the active model deployment strategy, the base station's response may inform the sensor node that it must update its prediction model.
- **Horizon Update**: If a measurement is outside the current prediction horizon, the sensor node will compute a new one. However, the base station also requires accumulated measurements to update its prediction horizon. Therefore, the sensor node sends a message to the base station containing the required measurement data (see Section 4.1.3).

The communication protocol utilized in the SenseReduce framework is HTTP (Hypertext Transfer Protocol). The base station provides RESTful endpoints using the Flask framework, which the sensor nodes access using the requests library. While the MQTT (Message Queueing Telemetry Transport) protocol would provide advantages in an IoT environment, we have chosen to use HTTP due to its capability to transfer large prediction models to sensor nodes. MQTT is a lightweight protocol optimized for sending small amounts of data and does not have native support for resuming or retrying transfers, which can be problematic when dealing with large data transfers.

However, the HTTP protocol poses scalability limitations for the framework in heterogeneous networks, as some IoT devices may only support MQTT or other custom protocols. Additionally, HTTP has a higher message overhead due to its text-based nature. One possible improvement in future versions of the framework would be to support multiple communication protocols or to adapt the protocol depending on the message type. This will increase the scalability and allow the selection of the most efficient protocol depending on the message type to improve overall communication efficiency.

4.3 Interfaces

The implementation of our data reduction framework, SenseReduce, is based on the principles of object-oriented programming, specifically utilizing the features of the Python programming language. Through encapsulation, we can achieve a high level of modularity within our framework. In addition, we utilize polymorphism in the form of subtyping, which allows for the implementation of abstract classes that define essential interfaces, ultimately resulting in a high degree of customizability for various application areas.

In Figure 4.3, we provide a UML class diagram of the core logic within our framework. Abstract classes are denoted by the keyword 'abstract' for easy identification. It is worth noting that while the implementation of our prediction-based data reduction algorithm is fixed as previously described in Section 4.1, its behavior can be adapted by implementing custom subtypes. In the subsequent sections, we will provide detailed information on the interfaces of all abstract classes in our framework.

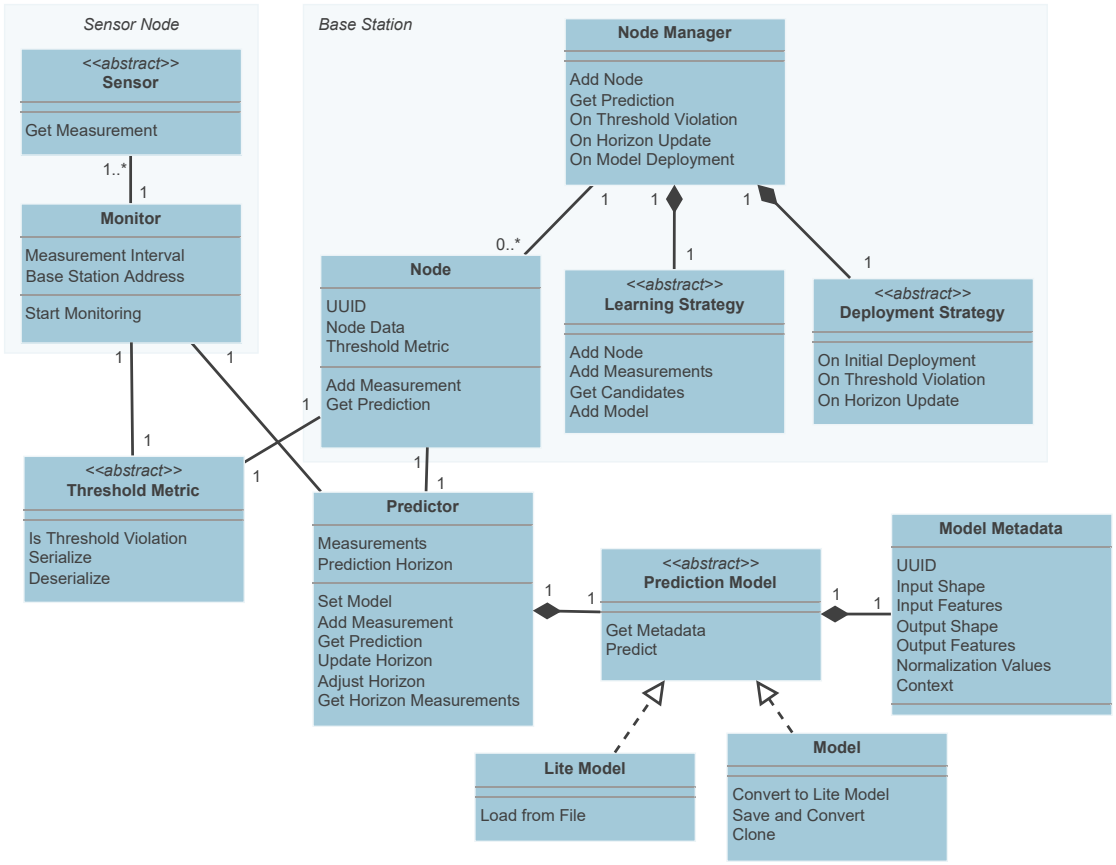


Figure 4.3: Class diagram of SenseReduce's core components

4.3.1 Threshold Metric

An implementation of the abstract ThresholdMetric class defines what constitutes a threshold violation in a given application scenario. Its interface consists of a single function:

- `is_threshold_violation(measurement, prediction)`: As described in Section 4.1, the function returns a Boolean value given a measurement and a prediction, indicating whether the measurement lies within the prediction's threshold. The implementation of this function should be tailored to the specific requirements of the application scenario and must be based on a thorough analysis of the underlying data. Furthermore, this function must be stateless and as efficient as possible to reduce computational overhead.

4.3.2 Prediction Model

The design of appropriate neural networks for a given task is considered a complex and multi-faceted task within the field of machine learning, with some researchers noting that it is often more of an art than science [Cho17]. Furthermore, different prediction tasks often require distinct neural network architectures. In light of this, it is important to design our framework to be architecture-agnostic to maximize its applicability across a wide range of domains.

The abstract class PredictionModel defines the minimal set of functions required for a prediction model within the SenseReduce framework. The interface of this class is composed of two functions:

- `get_metadata()`: This function returns a description of the prediction model, including information on its input and output dimensions, data required for value normalization, and any additional context.
- `predict(input)`: The core function of the model, providing predictions based on the input passed to it.

In addition to the abstract class, our framework also provides two subtypes of the PredictionModel class: Model and LiteModel. The Model class uses a TensorFlow model and is intended for use in a base station. In contrast, the LiteModel class utilizes a lightweight TensorFlow Lite model and is intended for use on sensor nodes. These subtypes demonstrate the adaptability of the framework to different requirements of the application scenario.

4.3.3 Update Strategy

Continuous model updates are a crucial aspect of the SenseReduce framework as they enable models deployed on sensor nodes to adapt and improve over time, ultimately leading to more accurate prediction-based data reduction and increased energy efficiency. The node manager within the framework utilizes two interfaces, the LearningStrategy and DeploymentStrategy, to devise an *update strategy* for any given sensor node.

The LearningStrategy class defines the continual learning method applied to the prediction models to allow them to adapt to new data. This class has several interface functions that include:

- `add_node(node, data)`: Allows the addition of new sensor nodes with the specified initial measurement data, which may be empty, to be maintained by the strategy.
- `add_measurements(node, measurements)`: Adds collected measurements of the specified node to the strategy that can be considered for updating prediction models.

- `get_candidates(node)`: Returns possible prediction models for the specified node. The number of models returned depends on the implementation of the strategy.
- `add_model(model)`: Allows for the addition of prediction models to the strategy.

It is worth noting that it depends on the implementation of LearningStrategy how many prediction models are maintained simultaneously and when to trigger continual learning of existing prediction models. For instance, if many resources are available, a separate model for each sensor node can be maintained and updated every time new measurements are available. On the contrary, in some scenarios, a single prediction model for all sensor nodes that is updated once a year may be sufficient.

The DeploymentStrategy class is responsible for continuous model deployment by deciding when prediction models devised by the LearningStrategy class are deployed to sensor nodes. The node manager accesses this event-based interface through the following functions:

- `on_initial_deployment(t, candidates)`: Triggered when a new sensor node registers at the base station at time t. The strategy selects a prediction model among the candidates to deploy at the sensor node.
- `on_threshold_violation(t, node, candidates)`: Called when a threshold violation occurred at time t on the specified node. The node object contains all measurements collected so far, and the strategy can select a model among the candidates to be deployed on the sensor node based on this information. However, it is up to the strategy to decide whether to deploy a new model at all.
- `on_horizon_update(t, node, candidates)`: The behavior is equally defined as in the `on_threshold_violation` function but allows the strategy to differentiate between threshold violations and horizon updates.

The interface is generic and allows for many different implementations. For instance, model deployments can happen in regular intervals or if data drift is detected in the node's collected measurements. The framework provides two default implementations for both LearningStrategy and DeploymentStrategy, which will be used in Section 5.4 to devise different update strategies for our simulation study.

CHAPTER 5

Parameter Study

In this chapter, we present the methodology for the parameter study that evaluates the performance of the SenseReduce framework in the application scenario of temperature monitoring. We first provide an overview of the experimental setup, including the datasets and models used. We then describe the evaluation metrics and methods employed to assess the performance of model update strategies in different simulated deployment scenarios. We also present the process for model selection, including the criteria used and strategies employed for the simulation. Finally, we outline the explored parameter space, providing details on the range of values and settings used to evaluate the framework's performance.

5.1 Experimental Setup

5.1.1 Hard- and Software

All experiments were conducted on a single desktop computer. We list the hardware specifications in Table 5.1.

CPU	Intel Core i9-11950H (2.60GHz)
RAM	64 GB
GPU	NVIDIA RTX A3000 Mobile
OS	Ubuntu 22.04.1
GPU Driver	520.61.05
CUDA Version	11.8

Table 5.1: Hardware used for the simulations

Graphical Processing Units (GPUs) are preferred over Central Processing Units (CPUs) for training ANNs because they can perform vector and matrix operations more efficiently. This allows for faster training of neural networks and reduces inference time. The TensorFlow library supports NVIDIA GPUs, which require the CUDA toolkit to enable GPU acceleration.

In addition to the software used by SenseReduce (Table 4.1), we used the libraries listed in Table 5.2 in our simulations for model optimization and data visualization. Additionally, we provide all code required for running the simulations as open source[1].

KerasTuner 1.1.3 [OBL+19]	hyperparameter optimization framework
matplotlib 3.6.2 [Hun07]	visualization library
seaborn 0.12.1 [Was21]	statistical data visualization

Table 5.2: Additional software libraries used for the simulations

5.1.2 Deployment Scenario: Air Temperature Monitoring

We conduct a simulation study based on the use case of environmental monitoring. Specifically, we will use SenseReduce to increase the energy efficiency of a single sensor node utilized for air temperature monitoring. We will simulate different measurement intervals ρ for the sensor node: 10, 30, and 60 minutes.

We define the prediction horizon length η to be 24 hours. Recent work has shown that NN-based prediction models work best for short-term intervals of this length [TBKV21]. Additionally, a horizon of 24 hours makes the implementation of baselines straightforward because they can exploit the daily periodicity naturally displayed by weather data (Section 5.2.2).

The data aggregation period δ is 1 hour, i.e., the base station will collect hourly measurements, which will be used as inputs for the prediction models. Consequently, prediction horizon updates can only occur at full hours.

The precision with which the currently measured temperature from the sensor node must be known can vary depending on the use case. Therefore, we will also simulate different threshold metrics $\mathcal{T}$ based on the absolute difference between predicted and measured air temperature. More specifically, given threshold value t, prediction $\hat{y}$, and measurement x, the threshold metric is defined as

$$\mathcal{T}_t = \begin{cases} \text{True} & \text{if } |\hat{y} - x| \geq t \\ \text{False} & \text{otherwise} \end{cases} \tag{5.1}$$

The threshold values t in our simulations will be 0.5 (`TL_low`), 1.0 (`TL_medium`), and 2.5°C (`TL_high`).

[1] `github.com/falzberger/sense-reduce`

5.1.3 Dataset

We base our experiments on data provided by the *Austrian Central Institution for Meteorology and Geodynamics (ZAMG) Data Hub*[2]. Table 5.3 provides an overview of the utilized datasets. We will use the data from the Vienna Inner City station to simulate the deployment of one sensor node with SenseReduce for two years, from January 1, 2020, to December 31, 2021. Along the classification introduced in Section 2.3 for continual learning, the simulation is a Single Incremental Task (SIT) scenario with New Instances (NI).

Stations	Vienna Inner City (5925)
	Linz City (3202)
Features	air temperature in 2 meters (TL)
	air pressure (P)
	relative humidity (RF)
	sunshine duration (SO)
Duration	2010-01-01 - 2021-12-31
Interval	10 min

Table 5.3: ZAMG dataset description

The feature of interest is the air temperature in 2 meters (TL), i.e., our threshold metrics evaluate the difference between measured and predicted air temperature. However, the sensor node observes additional variables, namely the air pressure (P), relative humidity (RF), and sunshine duration (SO). These are utilized as additional input variables to the prediction models to improve their performance.

Our goal is to also consider different availability of training data as a parameter in our simulations. As shown in Table 5.4, all training datasets range until December 31, 2019, i.e., the day before the deployment simulation starts (inclusive). We use three subsets of varying lengths of the dataset from Vienna Inner City as training data: *vienna_2010_2019* starting from January 1, 2010, *vienna_2019_2019* starting from January 1, 2019, and *vienna_201907_201912* starting from July 1, 2019. Additionally, we simulate a transfer learning scenario using data from another station, Linz City. The *linz_2010_2019* dataset will be described in more detail in the following section.

Analysis and Preprocessing

We first shortly discuss our process of data analysis and the preprocessing involved. We aim to optimize the performance of the prediction models and will show that the data from Linz City is feasible for the simulation scenario with transfer learning.

[2] `data.hub.zamg.ac.at`

Dataset Name	Station	Start Date	End Date
vienna_2010_2019	Vienna Inner City	2010-01-01	2019-12-31
vienna_2019_2019	Vienna Inner City	2019-01-01	2019-12-31
vienna_201907_201912	Vienna Inner City	2019-01-01	2019-12-31
linz_2010_2019	Linz City	2010-01-01	2019-12-31

Table 5.4: Definition of training datasets

The datasets from both stations contain 12 years of weather data in 10-minute intervals, resulting in 631,152 measured data points each. Both contain approximately 0.1% of missing data. We impute most missing values using linear interpolation for gaps up to 3 hours. For both datasets, a gap from April 6, 2017, to April 9, 2017, remains, where we replace each missing value by taking the average of the values from 24, 48, and 72 hours before.

The sampling interval of the datasets provided by ZAMG is 10 minutes. This allows us to simulate the deployment of a sensor node with a measurement interval ρ of 10 minutes with real-world data. However, we will use a data aggregation interval, δ, of 60 minutes in our simulations. Hence, we assume to have only hourly measurements available for model training. To that end, we downsample all training datasets using only the measurements at the full hours. Since the sunshine duration (SO) denotes the seconds of sunshine in the last 10 minutes, we downsample it by computing the sum over the last hour.

As mentioned above, we will use the historical weather data from Linz City in a simulation scenario to demonstrate the effect of transfer learning. We choose the Austrian city Linz as an approximation of Vienna which has similar but still different weather characteristics. Both stations are situated at the Danube river; however, Linz is located 150 kilometers west of Vienna. The Linz station lies at an altitude of 262 meters, and Vienna at 177 meters. In Figures 5.1 and 5.2, we visualize and compare the feature distributions, which we will briefly discuss.

Given the boxplots in Figure 5.1, we can observe that the value distributions are similar but centered around different means. The interquartile ranges do not have any noteworthy differences. For the sunshine duration (SO), we plot the hours of sunshine for every day; for all other attributes, we plot the distribution of all measurements. Pressure (P) and relative humidity (RF) display some outliers that are within the expected variance given the vast number of data points.

In Figure 5.2, we plot the daily averages over all years for every station. For all attributes except sunshine duration (SO), we plot the average over all years of the daily average. For SO, we plot the average over all years of the daily *sum*, i.e., the total hours of sunshine on a given day. Both datasets exhibit a strong yearly periodicity typical for the mid-European climate.

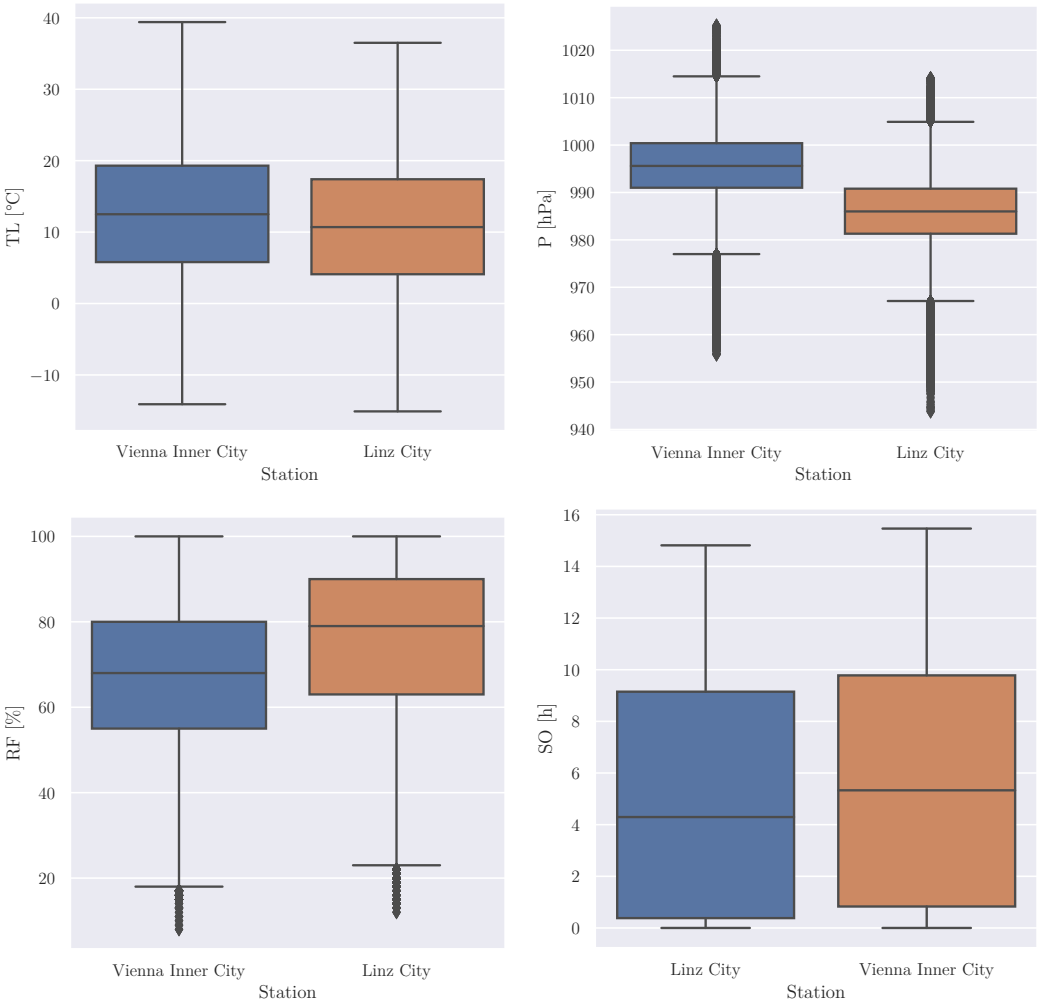


Figure 5.1: Feature distributions of the training datasets

We can observe that the yearly averages move in parallel but are shifted due to the different means. The pressure (P) displays the most significant difference, which the different altitudes of the stations can explain. We provide the Pearson correlation coefficient (of the daily averages or sum) for every attribute in Table 5.5. We can observe a very high correlation (> 0.95) for features TL and P. For RF and SO, the correlation is lower but still high (> 0.75). Given these observations, we conclude that the Linz City station is a good approximation of the Vienna Inner City station for our transfer learning scenario.

Intuitively, weather data exhibits a strong daily and yearly periodicity. However, neither the timestamp nor the time in seconds is a valuable input for NN models. Given a date

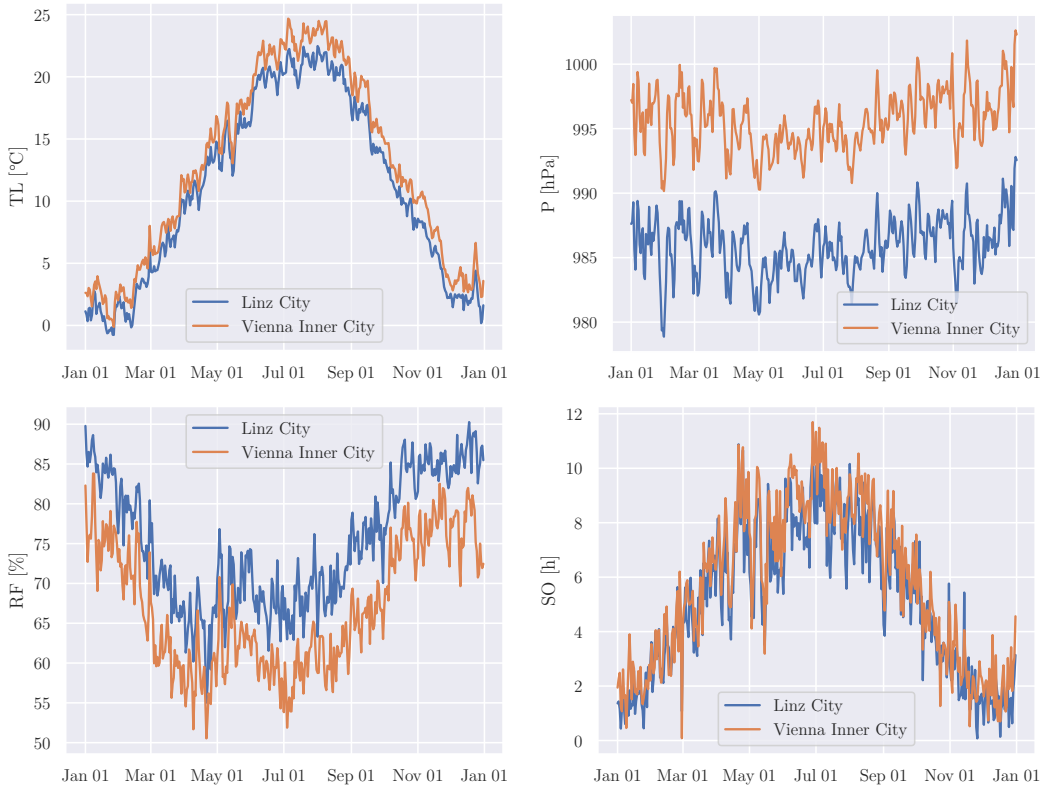


Figure 5.2: Yearly averages of the training datasets

Feature	Correlation Coefficient
TL	0.97088
P	0.97250
RF	0.75975
SO	0.82744

Table 5.5: Pearson correlation coefficients of dataset features from both stations

and time encoded in a timestamp t in seconds, we derive the following additional features:

$$Day_{sin} = sin(t \cdot \frac{2\pi}{24 \cdot 60 \cdot 60})$$

$$Day_{cos} = cos(t \cdot \frac{2\pi}{24 \cdot 60 \cdot 60})$$

$$Year_{sin} = sin(t \cdot \frac{2\pi}{365.2425 \cdot 24 \cdot 60 \cdot 60})$$

$$Year_{cos} = cos(t \cdot \frac{2\pi}{365.2425 \cdot 24 \cdot 60 \cdot 60})$$

We encode the timestamps using sine and cosine functions, which map radian (rad) values to the range $[-1, 1]$. Intuitively, the encoding allows us to provide the model with the information that 23:59h is close to 00:00h and December 31 is close to January 1.

As the last step, we apply feature normalization, which is essential for training well-performing neural networks. We transform all features to be normally distributed with zero mean and unit variance by using standard normalization:

$$z = \frac{x - \mu}{\sigma}, \tag{5.2}$$

where μ is the mean of the feature and σ its standard deviation. We arrive at complete and normalized datasets for our simulations with four input features (TL, P, RF, and SO) and encoded periodicity with sine and cosine functions.

5.2 Evaluation

5.2.1 Evaluation Metrics

We require evaluation metrics not only for comparing the results of different simulation scenarios but also as loss functions for training NN models. In the context of our simulation study, we are only predicting a single feature (air temperature), which allows us to use univariate metrics in the following definitions.

Let $\hat{y}$ be the vector of predicted values. The vector of true target values, i.e., the measurements, is given by x. Both $\hat{y}$ and x have length $|\hat{y}| = |x| = n$, and $\hat{y}_i$ (x_i) denotes the predicted (true) scalar value at position i. Given the data aggregation period δ of 1 hour, if any model outputs prediction vector $\hat{y}$ at time $t_{\hat{y}}$, the value $\hat{y}_1$ denotes the predicted air temperature at the next full hour *after* $t_{\hat{y}}$.

A simple measure to evaluate a model's accuracy with intuitive interpretation is the average over the absolute differences of all predictions. The *mean absolute error* (MAE) is defined as

$$MAE(x, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |x_i - \hat{y}_i|. \tag{5.3}$$

However, in the context of SenseReduce, prediction errors do not affect the data reduction algorithm as long as they do not cause a threshold violation. Hence, it would be better to have many small prediction errors than a single large error. The *mean squared error* (MAE) considers this by squaring the errors and hence increasing their impact with higher differences:

$$MSE(x, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{y}_i)^2. \tag{5.4}$$

By taking the square of the errors, the MSE changes the unit of measurement. Although this might not make a difference when using the metric for training NN models, the

resulting values are more difficult to interpret. The *root mean squared error* (RMSE) mitigates this issue by taking the square root of the MSE:

$$RMSE(x, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{y}_i)^2}. \tag{5.5}$$

In SenseReduce, using the above error metrics poses a problem. All values $y_i, 1 < i < n$ contributing equally to the metric implies that a correct prediction for the first hour in the prediction horizon is as important as for the last hour. However, if the first prediction y_1 already leads to a threshold violation, the last prediction y_n is never actually used. Therefore, we derive a weighted MSE metric that penalizes errors less the further they are in the future. The *weighted mean squared error* (WMSE) is defined as

$$WMSE(x, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{y}_i)^2 \left(10 + (i-1)\left(\frac{\frac{1}{n} - 10}{n-1}\right)\right). \tag{5.6}$$

Each squared error $(x_i - \hat{y}_i)^2$ is weighted by a linearly distributed penalty value ranging from 10 to $\frac{1}{n}$. Hence, given our prediction horizon of 24 hours, the first hour will be 240 times more important than the last hour in the WMSE metric. We will use the WMSE as a loss function for training NN models and as a metric to compare the performances of different model architectures.

5.2.2 Baselines

Using NN models for data prediction on sensor devices requires a non-trivial amount of data transfer to deploy and update the models. Hence, we want to ensure that the benefit outweighs the cost compared to simpler approaches. To this end, we define a simple baseline (without machine learning) that extrapolates the trend from the past 24 hours into the future. The `repeat` baseline is defined as follows:

Let x_t be the air temperature at horizon update time t, and x be a vector of hourly air temperature measurements of the past 24 hours. Moreover, x_1 represents the first element of x, i.e., the air temperature 24 hours before t. The prediction vector y for the next 24 hours is given by

$$y = x \oplus (x_t - x_1) \tag{5.7}$$

where $\oplus$ is the element-wise addition applied to vector x. Intuitively, `repeat` forecasts the *trend* of the last day one day into the future. Thus, it exploits the daily periodicity of the air temperature.

In addition, we will use two other baselines that serve as lower and upper bounds in the parameter study:

- `naive` baseline: The sensor node communicates every measurement directly to the base station without prediction-based data reduction. This approach results in

the most communication since the number of sent messages equals the number of measurements, and no data aggregation is applied.

- `oracle` baseline: We assume a perfect prediction model, i.e., $\hat{y} = x$ for all predictions $\hat{y}$ and (future) measurements x. This approach poses a theoretical lower bound. In practice, random noise in measurements makes it impossible to achieve a prediction model that perfectly forecasts all measurements.

5.3 Prediction Models

Finding a neural network architecture for a specific use case can be difficult because there are many types of architectures to choose from, each with its strengths and weaknesses. This requires domain expertise, knowledge of the latest research and techniques, and experimentation to determine the best solution. Moreover, more complex models may have increased performance but are typically larger in size and have a higher communication cost. Therefore, we will also have different neural network architectures as a parameter in our simulation scenarios.

We want to compare the effectiveness of simpler, more generic NN models against more complex architectures. Before we present the three chosen models in detail, we discuss our methodology in model design. As discussed in Section 4.2, SenseReduce supports the TensorFlow library for prediction models by default; therefore, we will also TensorFlow to implement our NN models. Moreover, by utilizing TensorFlow Lite for mobile and edge devices, we can reduce the size of deployed models without developing custom model compression techniques.

5.3.1 Model Design

Air temperature forecasting is an established research area, and NN-based models have shown promising results in recent work [TBKV21, CMBR20, LLS20]. However, reviews of the state-of-the-art conclude that "it is still difficult to pick the best methodology for air temperature forecasting" [TBKV21].

In recent research by Kreuzer et al. [KMS20], it has been shown that multivariate LSTMs outperform simpler neural networks and univariate Seasonal Autoregressive Integrated Moving Average (SARIMA) for temperature forecasting with a prediction horizon of up to 6 hours. For prediction horizons greater than 6 hours, they showed that convolutional LSTMs performed even better. The authors used datasets from four German cities with different climate characteristics. Since their climate is similar to our datasets, we can transfer these findings to our scenario.

Aligned with the state-of-the-art discussed in [TBKV21], and the results presented in [KMS20], we designed three different architectures:

- **Dense**: A simple generic model with two densely connected layers

- **LSTM**: An LSTM network specialized in time series forecasting
- **ConvLSTM**: An LSTM network with convolutional layers, similar to [KMS20]

After initial experiments, we defined a search space for every model variant. We then utilized KerasTuner [OBL+19], a hyperparameter optimization framework, to optimize the architecture of each model. The search has been conducted based on the dataset definition in Table 5.6. We do not use data from the simulation period (January 1, 2020, to December 31, 2021) for the model architecture search since this would introduce a bias based on future information in our model design.

Loss Function	WMSE
Dataset	vienna_2010_2019
Validation Data	2018-01-01 - 2019-12-31
Maximum Input Length	5 days
Maximum Epochs	150
Patience Value	20
Batch Size	32

Table 5.6: Hyperparameters for model architecture search

Additionally, we applied two callbacks during model training after every epoch to avoid overfitting the model to the training data. Therefore, although we defined an equal number of maximum epochs for all models, the actual number of epochs depends on the progress of the loss function (WMSE) during model training. Given the patience value p as listed in Table 5.6, we use the following callbacks:

- **EarlyStopping**: Early stopping is a technique that halts the training process if the loss function does not decrease over a set number of consecutive iterations, specified as p epochs. The callback will return the best model found during all epochs.
- **ReduceLROnPlateau**: If the loss function does not decrease for $\lfloor p/2 \rfloor$ epochs, the callback will reduce the current learning rate by a factor of 5 (i.e., multiply by 0.2).

We list the detailed search space of every model for the hyperparameter optimization in Appendix A. In the remainder of this section, we discuss the final design and compare the model performances.

5.3.2 Model Architectures

In Figure 5.3, we provide an overview of the design of every NN. For all layers, we use the TensorFlow default weight initialization method. Except for the *dense1* layer in the

Dense model, all Dense layers use a linear activation function. All convolutional layers in the ConvLSTM model have equal kernel size $k = 3$ and use zero-padding such that the output dimension does not change. For the Dense and LSTM layers, the numbers in parentheses indicate the number of units. For the Conv1D layers, it is the number of filters, and for the dropout and pooling layers, the dropout rate or pool size, respectively. As shown in Table 5.7, the hyperparameter optimization led to different optimizers and learning rates for every model.

Model	**Optimizer**	**Learning Rate**
Dense	RMSprop	$2.3994 \cdot 10^{-3}$
LSTM	RMSprop	$5.0287 \cdot 10^{-4}$
ConvLSTM	Adam	$2.4407 \cdot 10^{-4}$

Table 5.7: Optimizers and learning rates of the prediction models

The different numbers of layers in the NN visualizations give a sense of the differences in architectural complexity. Due to its design, an LSTM layer with an equal number of units as a Dense layer requires approximately four times more parameters. Also, the length of the input sequence varies across the models: While the hyperparameter optimization led to the Dense model only using the last 4 hours, the LSTM model uses the last 48 hours, and the ConvLSTM model uses the last 120 hours as input data. This influences the number of parameters in the first network layer, which we will analyze in more detail later in this section.

Model	**MAE**	**MSE**	**RMSE**	**WMSE**
`constant`	0.8673	1.0568	1.0280	5.3057
`last`	0.3210	0.1838	0.4287	0.8392
`previous`	0.2693	0.1285	0.3584	0.6447
`repeat`	0.2833	0.1599	0.3999	0.5542
Dense	0.1816	0.0630	0.2511	0.2337
LSTM	0.1763	0.0602	0.2453	0.2189
ConvLSTM	0.1736	0.0576	0.2400	0.2108

Table 5.8: Evaluation metrics of all prediction models including baselines

In Table 5.8, we provide the performance metrics on the validation dataset of the prediction models and additional baselines. In addition to the `repeat` baseline introduced in Section 5.2.2, we use the following:

- `constant`: Predict the average temperature from the training dataset for 24 hours
- `last`: Predict the last measured air temperature for 24 hours

- `previous`: Repeat the previous 24 hours of measurements

Overall, we can observe that the additional model complexity leads to better performance across all metrics. Considering the WMSE, LSTM can improve by 6% compared to the Dense, and ConvLSTM by 10%. Compared to the baselines, all NN prediction models show a significant improvement. For example, Dense performs 36% better in terms of MAE and 58% better in terms of WMSE compared to the `repeat` baseline, which suggests that the prediction-based data reduction of SenseReduce benefits in particular from the increased performance.

Notably, the `previous` strategy performs better in terms of the MAE, MSE, and RMSE metrics, while the `repeat` strategy exhibits superior performance according to our custom WMSE metric. This discrepancy can likely be attributed to the fact that the `previous` strategy is better suited for short-term predictions due to its ability to extrapolate trends, while the `repeat` strategy is better suited for long-term predictions. In the context of SenseReduce, the `repeat` strategy is, therefore, the more desirable choice.

Although the LSTM and ConvLSTM models show better performances, they come at the cost of higher architectural and computational complexity. Recent work [Won19] has introduced *information density* and NetScore as metrics to assess which NN models offer the best trade-off between prediction performance and complexity. Since the NetScore is an extension of information density, we will first introduce the latter:

Given a NN model $\mathcal{M}$, its information density $D(\mathcal{M})$ is given by

$$D(\mathcal{M}) = \frac{a(\mathcal{M})}{p(\mathcal{M})}, \tag{5.8}$$

where $p(\mathcal{M})$ is the number of parameters of the NN as described in Section 2.2 and $a(\mathcal{M})$ is the model's accuracy.

The accuracy is generally defined as the percentage of samples correctly attributed to their classes in a classification task. However, since air temperature forecasting is a regression task, we need to redefine the concept of accuracy to calculate the NetScore metric for the models.

In [CDF21], the authors also applied the NetScore metric to compare models for air temperature forecasting. To derive an accuracy metric, they used an arbitrary threshold for the prediction error of 1°C. In our simulations, the temperature threshold varies, making using a constant threshold impractical. Hence, we derive a *regression accuracy* metric:

Let $\epsilon_{\mathcal{M}}$ be the MAE of a neural network prediction model $\mathcal{M}$ used for a regression task, then its accuracy $a(\mathcal{M})$ is defined as

$$a(\mathcal{M}) = (1 - \frac{\epsilon_{\mathcal{M}}}{\epsilon_{\mathcal{B}}}) \cdot 100, \tag{5.9}$$

where $\epsilon_{\mathcal{B}}$ is the MAE of an upper-bound baseline $\mathcal{B}$. Theoretically, the accuracy could become negative; however, in that case, the model would not provide any advantages over the baseline. Choosing an appropriate baseline $\mathcal{B}$ that depicts an appropriate worst-case performance is thus essential. We will use the `constant` baseline (Table 5.8) in our evaluation; a NN model performing worse than this baseline does not have any use for prediction-based data reduction.

The metric of information density only accounts for the structural complexity of a model, based solely on the number of parameters, $p(\mathcal{M})$. However, as stated in Section 2.2, the complexity of a network's architecture does not necessarily reflect the amount of computation required to run inferences on the network. To address this, the NetScore metric was introduced by [Won19]. It is defined as:

$$\Omega(\mathcal{M}) = 20\ log\left(\frac{a(\mathcal{M})^{\alpha}}{p(\mathcal{M})^{\beta} m(\mathcal{M})^{\gamma}}\right), \tag{5.10}$$

where $m(\mathcal{M})$ represents the number of multiply-accumulate (MAC) operations required for a single inference computation, i.e., the computational complexity. The coefficients α, β, and γ determine the relative importance The authors propose default values of $\alpha = 2$ and $\beta = \gamma = 0.5$, which we will also use for our evaluation.

We arrive at the complexity metrics for all prediction models as shown in Table 5.9. We computed each model's computational and architectural complexity as described in Section 2.2. First, we can observe that the number of parameters required for model representation increases by a factor of 6 from Dense to LSTM and by a factor of 4 from LSTM to ConvLSTM. However, the MAC operations for model inference increase by a much greater factor ($\times$355 from Dense to LSTM, $\times$7 from LSTM to ConvLSTM). This is due to the higher computational complexity of the LSTM and Conv1D layers: With Dense layers, every parameter is involved in a single computation during network inference; in LSTM and Conv1D layers, a parameter is used for every element in the input time series sequence.

The models' computed accuracy, $a(\mathcal{M})$, demonstrates only minimal differences, similar to the MAE metric. Therefore, the information density displays the same relations as the number of parameters, with decreases by factors of 6 and 4, respectively. On the contrary, the NetScore also considers the MAC operations and shows a greater difference between the models. Information density and NetScore metrics show that the Dense model has a much better performance-complexity trade-off due to its small architectural and computational complexity.

The model size $|\mathcal{M}|$ is the number of bytes required to represent the model after compressing it with the TensorFlow Lite library. Since the model parameters constitute the main part of the required information, the size is proportional to $p(\mathcal{M})$. Additionally, TensorFlow Lite can decrease the file size with increasing parameters, as the size increases only by factors 5 and 3 from Dense to LSTM and LSTM to ConvLSTM, respectively, compared to factors 4 and 6 of the parameters.

Model ($\mathcal{M}$)	Dense	LSTM	ConvLSTM
Parameters ($p(\mathcal{M})$)	936	6,040	26,016
MAC operations ($m(\mathcal{M})$)	936	355,608	2,507,400
Accuracy ($a(\mathcal{M})$)	79.061	79.673	79.984
Information Density	0.084	0.013	0.003
NetScore	16.493	-17.268	-32.025
Model size ($\lvert\mathcal{M}\rvert$)	7,204	35,036	119,736

Table 5.9: Complexity of all prediction models in terms of number of parameters, MAC operations, Information Gain, NetScore, and TensorFlow Lite model size

5.4 Update Strategies

Model update strategies can greatly influence the prediction performance and model transfer cost over the simulation period. Therefore, in addition to the simulation variables introduced in the sections above, we will compare multiple strategies for continual learning and continuous deployment of prediction models. Although each strategy has a different approach to continual learning, they all use the callbacks defined in Section 5.3.1 with a patience value of 20 and at most 100 training epochs. The strategies are implemented in SenseReduce and described as follows:

- `static`: After initial deployment, the sensor node receives no model updates.
- `retrain_*`: A new model gets trained on all collected data. The most recent 30% of the data is used as validation data during training. For every model, we use its defined optimizer and learning rate. We simulate two variants, `retrain_short` updates the model quarterly (after 91 days), and `retrain_long` after half a year (183 days).
- `fine_tune_*`: All model weights get fine-tuned with a reduced learning rate. The model-specific optimizer is used, and we define a reduced learning rate for every model: $2 \cdot 10^{-5}$ (Dense), $5 \cdot 10^{-6}$ (LSTM), and $2 \cdot 10^{-6}$ (ConvLSTM).

  The `fine_tune_*` strategies maintain a limited buffer: After every update, the collected data used as training data is removed, while the validation data is kept to be used in the next update. Like the other strategies, we simulate two variants, `fine_tune_short` and `fine_tune_long`, with quarterly and half-yearly updates, respectively. The last four weeks of validation data are used for quarterly updates, and for half-yearly updates, the last eight weeks are used.
- `transfer_*`: Transfer learning is applied to the models by freezing a model-specific set of layers and only training a subset of layers on newly collected data. For the Dense model, we freeze the first dense layer (*dense1*); for the LSTM

model, only the LSTM layer (*lstm*) is frozen, and in the ConvLSTM model, all convolutional layers (*conv1* - *conv4*) keep their model weights.

With these strategies, we aim to maintain already learned features in higher NN layers while adapting to new data with minimal effort. The more data is available for the initial model to train, the better the strategies are expected to work.

The `transfer_*` strategies apply the optimizer and learning rate defined for each model in Table 5.7. The update intervals, validation data, and buffer size of the `transfer_*` strategies are defined equally as in the `fine_tune_*` strategies.

5.5 Parameter Space

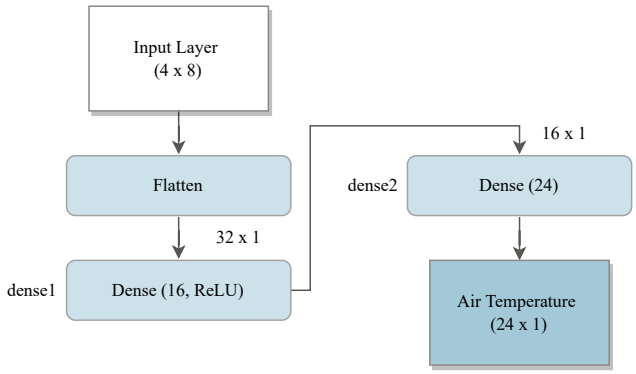
Prediction Horizon	24 h
Aggregation Interval	1 h
Model	Dense, LSTM, ConvLSTM
Training Data	vienna_2010_2019,
	vienna_2019_2019,
	vienna_201907_201912,
	linz_2010_2019
Threshold Metric	`TL_high`, `TL_medium`, `TL_low`
Stride	10 min, 30 min, 60 min
Strategy	`static`,
	`retrain_short`, `retrain_long`,
	`fine_tune_short`, `fine_tune_long`,
	`transfer_short`, `transfer_long`,

Table 5.10: Parameter space of the simulations

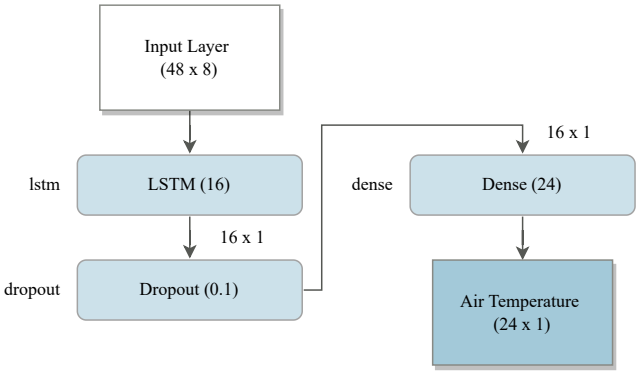
An overview of the parameter space for all simulation scenarios is given in Table 5.10. Considering the cardinality of every parameter, the total number of simulation scenarios is

$$3 \cdot 4 \cdot 3 \cdot 3 \cdot 7 = 756.$$

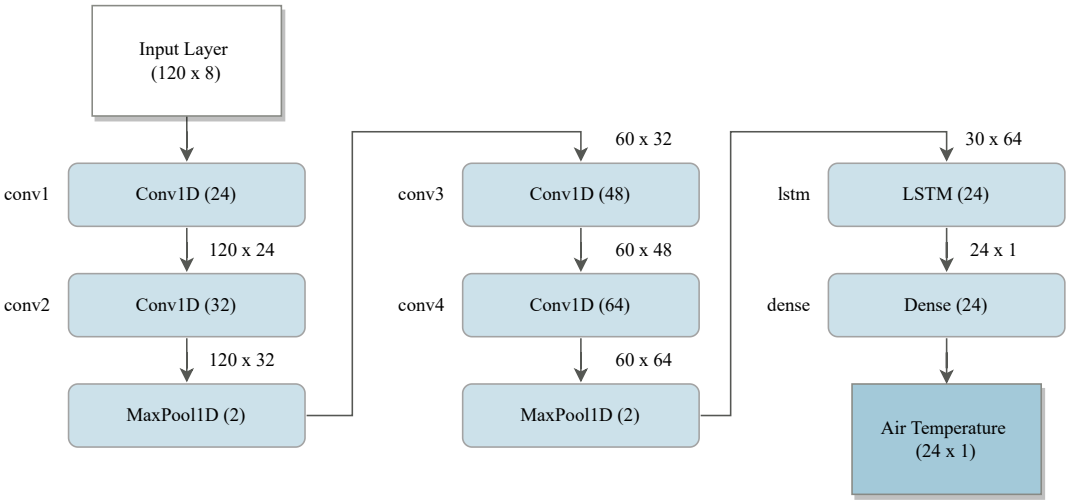
Employing the hardware described in Section 5.1.1, running all simulations required approximately 92 hours to complete. We will describe our analytical approach and discuss the results in the next chapter.



(a) Dense model architecture



(b) LSTM model architecture



(c) ConvLSTM model architecture

Figure 5.3: Neural network architectures of the prediction models

CHAPTER 6

Results

In this chapter, we present the results of the simulation scenarios described in the previous chapter. Our focus is on energy efficiency through data reduction and, consequently, on the number of exchanged messages and the amount of transferred data in every scenario. We begin by explaining our method for analyzing network communication, followed by a brief discussion of the baseline performances. We then present the results for each model architecture and, finally, provide a summary of the findings across all simulation scenarios. We highlight the most interesting findings in this chapter and provide tables with detailed results for every simulation scenario in Appendix B.

Communication Analysis

We assume the sensor nodes are deployed in a standard IEEE 802.11 (Wi-Fi) WLAN to compute the transferred data in our simulations. The nodes use the IPv4 protocol for communication based on the TCP protocol (TCP/IP). In practice, further optimizations in network communication are possible using Low Power Wide Area Network (LPWAN) protocols like LoRaWAN. However, their choice depends on the application's requirements and the available infrastructure. Moreover, they can require custom protocol implementations due to maximum allowed packet size limitations.

We base the analysis of the total transferred data in our parameter study on the following: The TCP and IPv4 headers have a fixed size of at least 20 bytes per network packet [Pos81a, Pos81b]. The minimum overhead for a data frame in a Wi-Fi network is 32 bytes [IEE16]. Hence, the packet overhead p_h is 72 bytes in total.

Each data point consists of a timestamp and the four features described in Table 5.3. In our setup, the timestamps will always be a full hour by definition and can thus be inferred from the message reception time, i.e., they are implicitly encoded. Each of the four features is encoded with a 32-bit float value, resulting in a data point size p_d of 16 bytes. Given a prediction horizon length η of 24 hours and data aggregation period δ of 1

hour, the maximum payload size of a single data exchange is $(\eta/\delta) \cdot p_d = 24 \cdot 16 = 384B$. Since the 802.11 standard gives the smallest maximum transmission unit (MTU) of all communication protocols with 2304 bytes, we can assume that a single packet will always be sufficient for a single data exchange.

In total, given a set of V threshold violations, n_u horizon updates, and n_d model deployments with a model of size $|\mathcal{M}|$, we estimate the data transferred in total $\mathcal{D}$ with the equation

$$\mathcal{D} = \sum_{v \in V} (p_h + elapsed(v) \cdot p_d) + n_u(p_h + h \cdot p_d) + n_d(p_h + |\mathcal{M}|), \tag{6.1}$$

where $elapsed(\cdot)$ is a function such that given a threshold violation v, it computes the elapsed hours since the last threshold violation or horizon update event, i.e., the number of (new) data points transferred with the threshold violation.

By design, $elapsed(v) < (\eta/\delta)$ for any given threshold violation v, prediction horizon length η, and data aggregation period δ. SenseReduce would trigger a horizon update if no threshold violation occurs within the defined prediction horizon. Consequently, threshold violation messages are affected more by the packet overhead p_h than horizon update messages. Hence, although decreasing the number of threshold violations can lead to increased horizon updates, the amount of data transferred is reduced in total.

Given the number of threshold violations $n_v = |V|$, we can further compute the total number of message exchanges $\mathcal{E}$ in a deployment scenario by summarizing the number of events:

$$\mathcal{E} = n_v + n_u + n_d. \tag{6.2}$$

Most related work focuses only on this communication metric, neglecting the network overhead and model transfer cost. Nevertheless, reducing the number of messages is an essential complementary metric as it can improve the duty cycle of sensor nodes by allowing them to power down their wireless communication for extended periods.

The difference between data transmission and reception is not considered in analyzing the transferred data in our parameter study. Instead, the focus is on the total amount of data transferred between the sensor node and the base station. The difference in energy consumption between transmission and reception is determined by the sensor node's hardware, making it difficult to accurately estimate the energy consumption for all possible application scenarios. Moreover, the broadcast nature of wireless networks can result in sensor nodes receiving messages that are not intended for them, thereby increasing the energy consumption associated with data reception in dense networks. Therefore, we apply the data transferred in total combined with the number of message exchanges as a metric for the overall energy efficiency.

It is important to note that our estimation of transferred data simplifies the required communication in a Wi-Fi network by only using the TCP/IP protocol. The cost of a single message exchange may be significantly higher in practice due to overhead from

additional layers in the communication setup, such as TLS or HTTPS. Additionally, message exchanges typically occur at sparse intervals in SenseReduce, leading to temporary disconnections. Establishing a connection can be resource-intensive, particularly in congested networks with a high density of sensor nodes, and may further increase the overhead for a single message exchange. However, the network infrastructure depends on the application requirements, making a general analysis challenging. Our communication analysis aims to show that prediction-based data reduction can reduce network communication even in an environment with minimal overhead.

Baselines

Before we present the simulation results for every prediction model, we shortly discuss the performances of the baselines introduced in Section 5.2.2 since they are equal across all simulation scenarios.

Baseline	ρ [s]	$\mathcal{T}$	n_v	n_u	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
`naive`	600	-	105264	0	9263232	-	-	-
`naive`	1800	-	35088	0	3087744	-	-	-
`naive`	3600	-	17544	0	1543872	-	-	-
`oracle`	-	-	0	731	333336	0	0	0

Table 6.1: `naive` and `oracle` baselines

We provide the results for the `naive` and `oracle` baselines in Table 6.1. For both baselines, the performance is independent of the applied threshold metric. No predictions are made for the `naive` baseline, and thus the error metrics (MAE, MSE, RMSE) are not applicable. For the `oracle` baseline, we assume perfect predictions (i.e., $\hat{y} = x$) independent of the measurement interval ρ, and hence the error metrics are equal to zero.

The `naive` approach shows the upper bound for the number of exchanged messages and the amount of transferred data. Each measurement will be communicated to the base station, which is equal to each measurement resulting in a threshold violation. Hence, the number of threshold violations n_v equals the total number of measurements taken over the simulation period.

On the contrary, given the `oracle` baseline, we can observe a theoretical lower bound of approximately 333 kB of transferred data in our simulation scenario. Due to the prediction horizon length η of 24 hours, the sensor node sends daily messages containing hourly measurements of the elapsed day. Without any communication overhead, these data alone would require $p_d \cdot 24 \cdot (366 + 365) = 280,704$ bytes. The additional packet overhead leads to a total sum of 333,336 bytes. However, we do not consider the data required for model transfer in this sum, as we do not know the minimum size of a perfect prediction model. Therefore, the actual lower bound must be greater in our defined simulation scenario.

ρ [s]	$\mathcal{T}$	n_v	n_u	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	1811	203	431201	0.9538	1.3923	1.1799
600	TL_medium	6530	20	757034	0.4562	0.3185	0.5644
600	TL_low	15102	0	1372704	0.2809	0.1309	0.3617
1800	TL_high	1641	212	418856	1.0475	1.7152	1.3097
1800	TL_medium	5472	16	680400	0.5745	0.5561	0.7457
1800	TL_low	10970	1	1075264	0.4243	0.3447	0.5872
3600	TL_high	1494	199	404032	1.1677	2.2669	1.5056
3600	TL_medium	4618	20	618952	0.7544	1.0747	1.0367
3600	TL_low	8173	2	873904	0.6377	0.8589	0.9268

Table 6.2: Simulation results for `repeat` baseline

In Table 6.2, we present the results of the `repeat` baseline. The exhibited performance can be intuitively explained across multiple parameter dimensions:

- The lower the threshold of the applied metric $\mathcal{T}$, the higher the number of threshold violations n_v and the lower the number of horizon updates n_u. Since threshold violations lead to an adjustment of the prediction horizon, the error metrics improve similarly. For `TL_low`, we observe that it is rarely the case that a full day passes without any threshold violation, i.e., n_u is close to (or exactly) 0. Moreover, with `TL_low` and a measurement interval of 3600 seconds, almost every second measurement (47%) leads to a threshold violation.
- As the measurement interval increases, the number of threshold violations n_v decreases. Since fewer model updates occur, the error metrics worsen. Interestingly, the number of horizon updates n_u seems to be (almost) unaffected by the measurement frequency.
- The higher the number of threshold violations n_v, the higher the amount of total data transferred $\mathcal{D}$. As shown in Equation 6, threshold violations have a more significant share in data transfer than horizon updates due to packet overhead. Hence, although the number of horizon updates n_u is decreasing, they cannot compensate for the increase in n_v in the `repeat` baseline.

For every measurement interval ρ, we can already observe that the `repeat` baseline significantly reduces transferred data compared to the `naive` baseline. For instance, with a measurement interval of 600 seconds, `repeat` reduces the transferred data by 85% to 95% and the number of exchanged messages by 86% to 98%. Clearly, the higher the defined threshold metric, the more relative improvement can be expected.

The `repeat` baseline requires 21% more data and 131% more messages compared to the `oracle` baseline in the best scenario ($\rho = 3600$, $\mathcal{T} =$ `TL_high`). In the worst scenario ($\rho = 600$, $\mathcal{T} =$ `TL_low`), 312% more data and 1966% more messages are needed.

We can conclude that the trend extrapolation of the `repeat` baseline can already lead to a significant data reduction in our simulation scenario compared to the baseline without any data reduction. In the remainder of this chapter, we will analyze the performance of prediction-based data reduction using neural networks by comparing the number of message exchanges and the amount of transferred data for every model update strategy against the `repeat` baseline.

6.1 Dense model

The first prediction model in our simulation analysis is the Dense model, which has a model size of 7204 bytes after being converted to a TensorFlow Lite model. We will examine the results for each dataset separately and provide more detailed information in Tables B.2 through B.5.

6.1.1 vienna_2010_2019

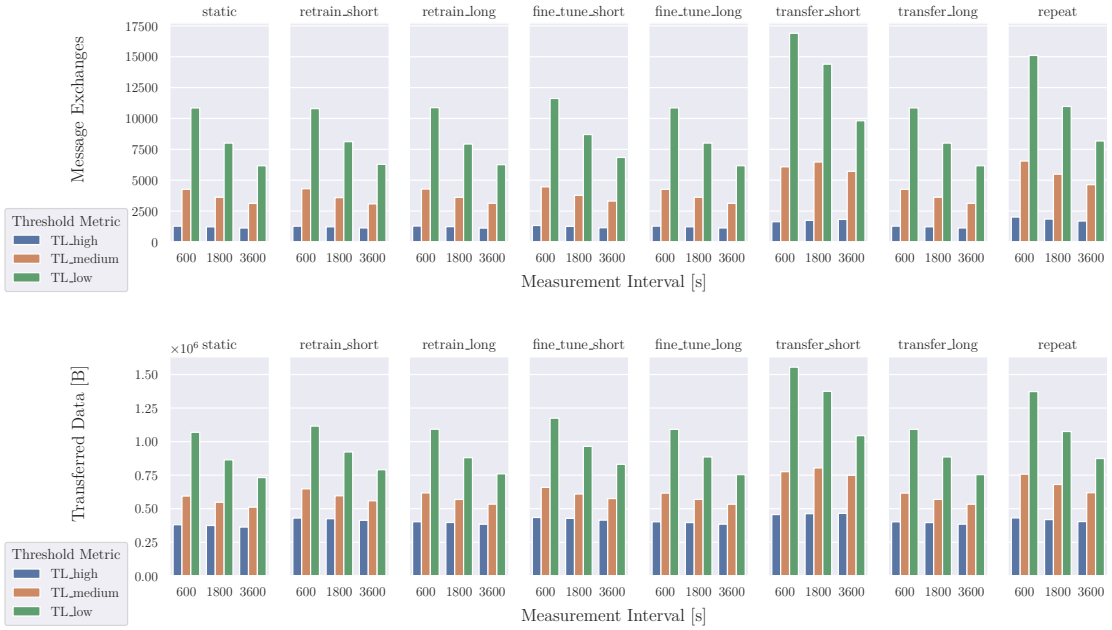


Figure 6.1: Results for Dense model with vienna_2010_2019 dataset

In the simulations conducted on the vienna_2010_2019 dataset, several strategies demonstrated similar performance in terms of message exchanges, as shown in Figure 6.1. We achieve a reduction in the number of messages by up to 37% compared to the `repeat` baseline with `retrain_`$\star$ and `static` strategies ($\rho = 600$, $\mathcal{T} =$ `TL_high`). The `static` strategy required the least amount of data transfer in all scenarios, with a potential reduction of up to 22% compared to the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$

TL_low). Moreover, in the best case, the static strategy requires only 9% more data than the oracle baseline ($\rho = 3600$, $\mathcal{T} =$ TL_high). The strategies with frequent model deployments (*_short) did often perform worse than their counterparts with less frequent model deployments (*_long). Additionally, the model transfer cost increased data transfer compared to the repeat baseline.

The transfer_short strategy exhibited the poorest performance in all scenarios, even performing worse than the repeat baseline in scenarios with a threshold metric other than TL_high. This can be attributed to the small model size and limited training data for transfer learning, resulting in overfitting and reduced performance.

6.1.2 vienna_2019_2019

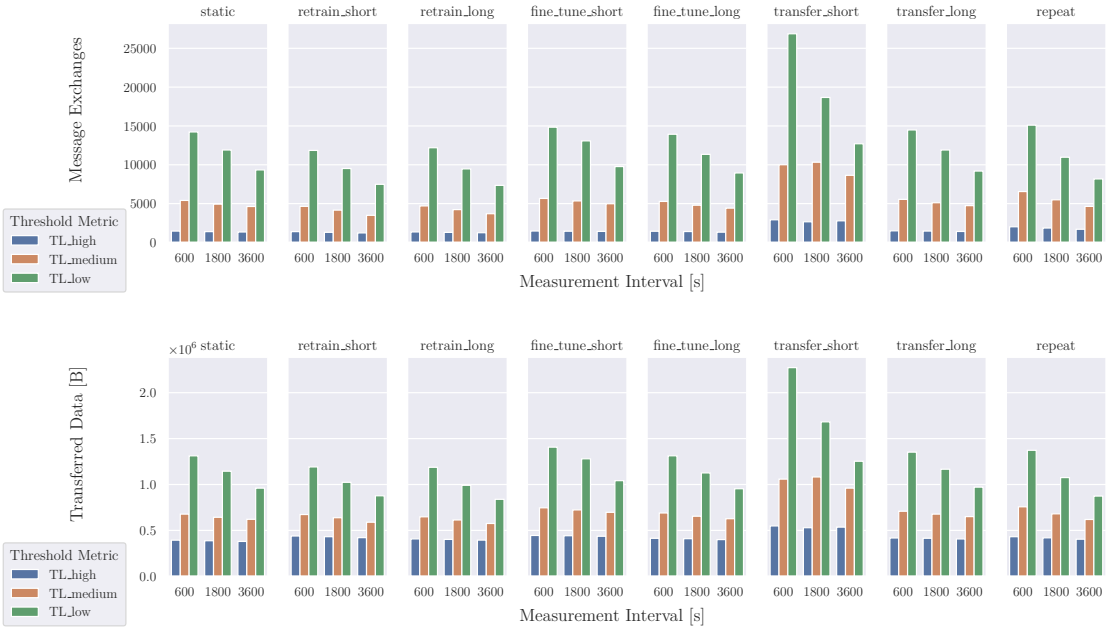


Figure 6.2: Results for Dense model with vienna_2019_2019 dataset

As shown in Figure 6.2, when using the Dense model with training data from the year 2019 only, all strategies experience a decrease in performance compared to the results obtained when using the entire training dataset. However, it is still possible to reduce the number of message exchanges by up to 33% ($\rho = 600$, $\mathcal{T} =$ TL_high) and the amount of transferred data by up to 14% ($\rho = 600$, $\mathcal{T} =$ TL_low) compared to the repeat baseline. As the measurement interval increases, the relative performance gain decreases.

Intuitively, the retrain_* baselines show the best performance in terms of message exchanges across all scenarios. They can fully utilize the accumulated new data by training the models anew. However, considering the data transferred, the static strategy is best when applying the TL_high threshold metric. Otherwise, retrain_long outperforms

both `static` and `retrain_short`. In particular, only the `retrain_long` strategy outperforms the `repeat` baseline regarding transferred data and message exchanges in the scenario with $\rho = 3600$ and $\mathcal{T} =$ `TL_low`. Similar to the scenarios with the vienna_2010_2019 dataset, the `transfer_short` has the poorest performance.

Given the reduced dataset, we can observe that static deployment of the Dense model performs worse than the `repeat` baseline if the threshold metric is low and the measurement interval increased ($\rho \geq 1800$, $\mathcal{T} =$ `TL_low`). This already highlights one of the limitations of NN-based prediction models, namely the requirement of sufficient training data.

6.1.3 vienna_201907_201912

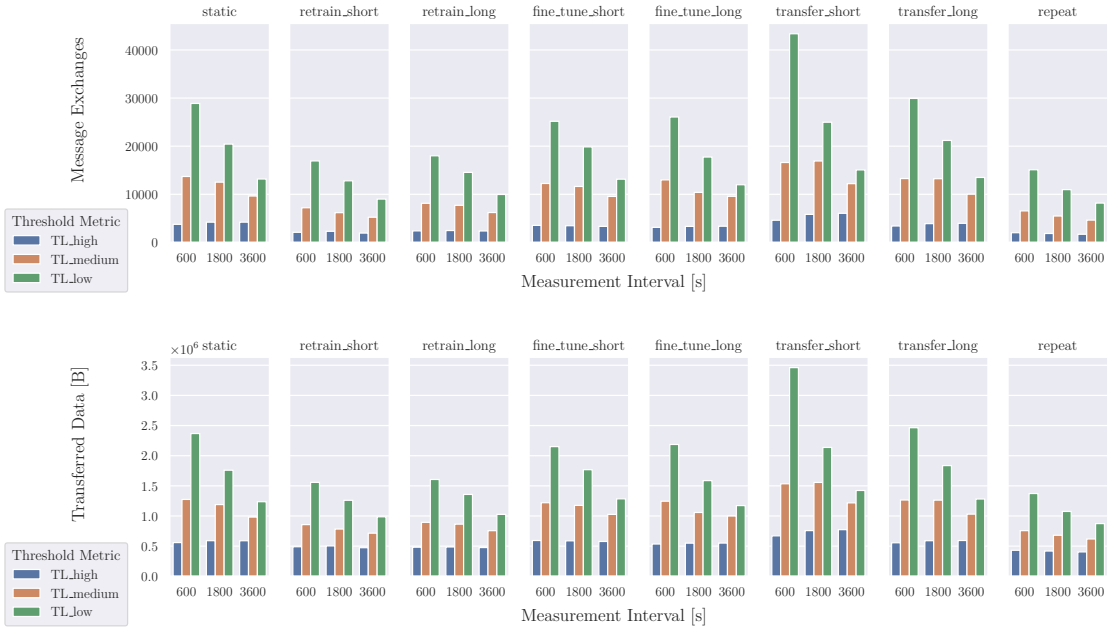


Figure 6.3: Results for Dense model with vienna_201907_201912 dataset

The simulation results when limiting the available training data to half a year are shown in Figure 6.3. Overall, the performance is significantly lower than when using an entire year or multiple years of data, and the trend observed with the vienna_2019_2019 dataset continues.

The `retrain_`$\star$ strategies perform the best but still require at least 4% more message exchanges or 12% more transferred data compared to the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`). Across all scenarios, none of the strategies outperform the `repeat` baseline in terms of exchanged messages or transferred data. Therefore, even for the neural network with the lowest architectural complexity, we require at least a full year of data for efficient prediction-based data reduction.

6.1.4 linz_2010_2019

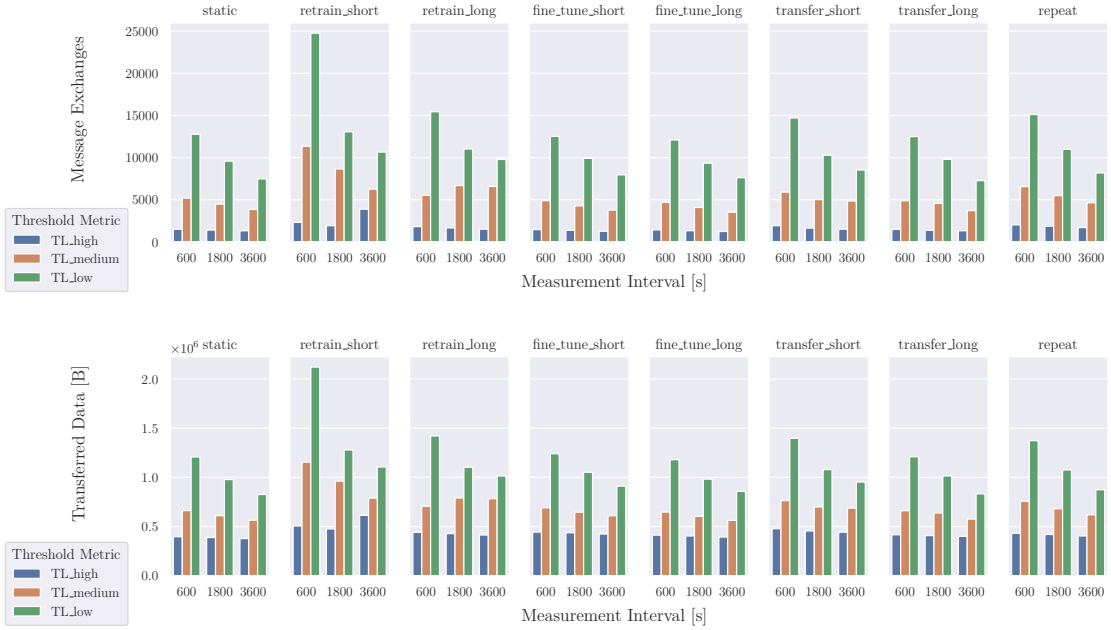


Figure 6.4: Results for Dense model with linz_2010_2019 dataset

With the linz_2010_2019 dataset, we demonstrate the effects of transfer learning, simulating a deployment scenario where only an estimation of the historical measurements is available for model training. The results in Figure 6.4 show that we can significantly improve the performance compared to both scenarios with reduced training data from the exact location. Overall, we can reduce the exchanged messages by up to 30% ($\rho = 600$, $\mathcal{T} =$ `TL_high`) and the transferred data by up to 15% ($\rho = 600$, $\mathcal{T} =$ `TL_low`) compared to the `repeat` baseline.

Because the `retrain_*` strategies do not use data from the linz_2010_2019 dataset for continual learning but only the newly accumulated data during the simulation, they exhibit the poorest performance. They do not acquire enough data during the two simulated years to train a model with adequate performance.

In scenarios with a measurement interval of 3600 seconds and `TL_low` threshold metric, the `transfer_long` strategy requires the fewest message exchanges. Moreover, it outperforms the best-performing strategy for the same scenario with the vienna_2019_2019 dataset.

If the `TL_high` metric is applied, the `static` strategy requires the least data transfer. However, in terms of message exchanges, the `fine_tune_long` strategy performs best in 8 out of 9 scenarios and has the least amount of transferred data for all threshold metrics other than `TL_high`. These results demonstrate the effectiveness of transfer

learning techniques in scenarios with limited training data.

6.2 LSTM model

In the following analysis, we will examine the simulation results for the LSTM model. The TensorFlow Lite version of this model has a size of 35036 bytes, which is five times larger than the size of the Dense model. While the model will require more data transfer, we expect it to reduce the message exchanges by providing better predictions. We will present the results for each dataset individually and provide detailed results in Tables B.6 through B.9.

6.2.1 vienna_2010_2019

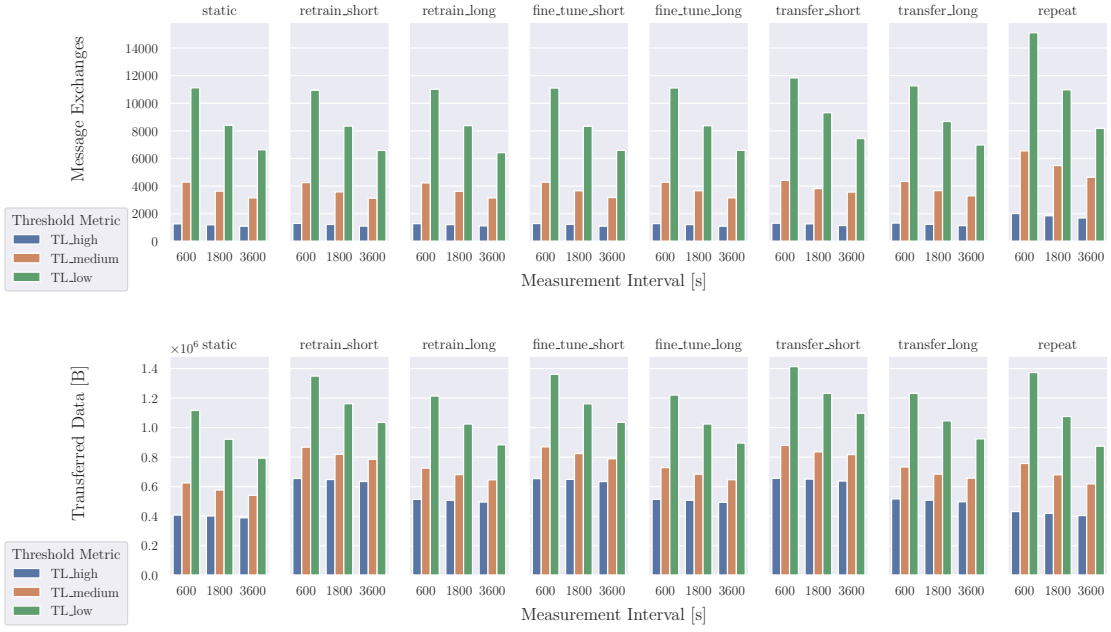


Figure 6.5: Results for LSTM model with vienna_2010_2019 dataset

Given the entire training dataset from the deployment location, all model update strategies perform similarly to the Dense model (Figure 6.5). Only in scenarios with higher threshold metrics ($\mathcal{T} \neq$ `TL_low`) can we observe a performance increase compared to the Dense model in terms of message exchanges. Although the `transfer_short` strategy still has the poorest performance among all strategies, it is better compared to the Dense model in all scenarios, showing the effect of the implementation adapted to the NN architecture.

All strategies outperform the `repeat` baseline regarding exchanged messages. For the `retrain_*` and `static` strategies, we can again achieve a reduction by up to 37% ($\rho = 600$, $\mathcal{T} =$ `TL_high`). However, we can reduce the transferred data by at most 19%

due to the increase in model size ($\rho = 600$, $\mathcal{T} =$ `TL_low`), and the `static` strategy is the only strategy that outperforms the `repeat` baseline across all scenarios. The higher the defined threshold, the more strategies require more data transfer than the baseline.

6.2.2 vienna_2019_2019

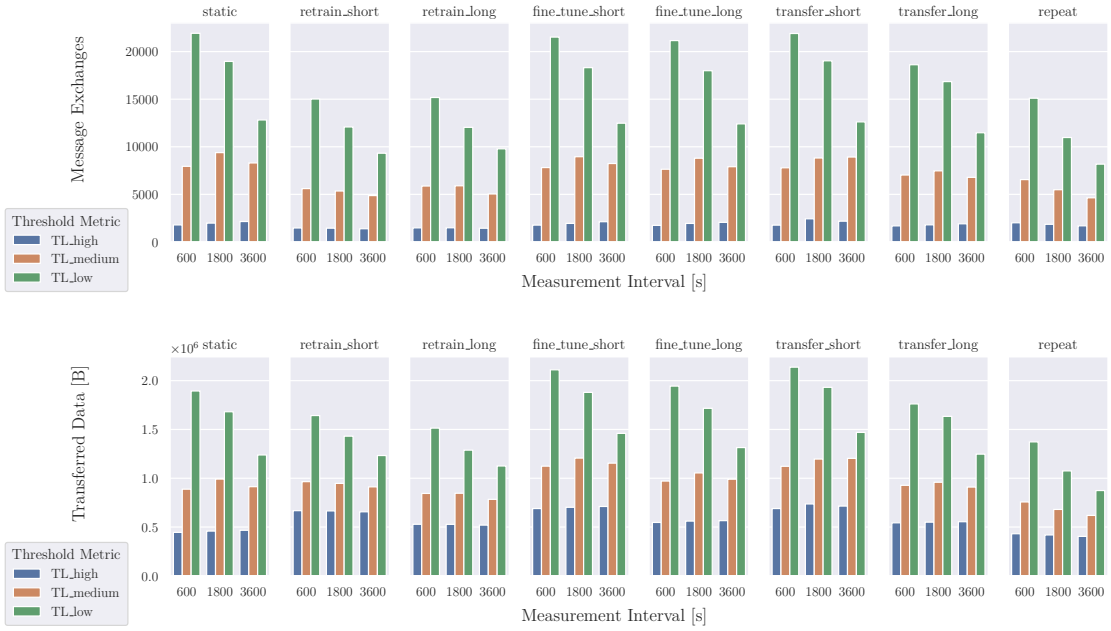


Figure 6.6: Results for LSTM model with vienna_2019_2019 dataset

When the dataset is limited to one year with the vienna_2019_2019 dataset, the performance of the LSTM model significantly decreases, as shown in Figure 6.6. The performance degradation is worse compared to the Dense model, which is likely due to the fact that the LSTM model has a higher architectural complexity and is, therefore, more sensitive to the lack of available training data. Upon directly comparing the performances of the best strategies with Dense and LSTM models across all scenarios, the Dense model consistently outperforms the LSTM model.

Nevertheless, the `retrain_short` strategy demonstrates the most significant reduction in message exchanges in all scenarios. This strategy enables improved performance by allowing the model to incorporate accumulated data as quickly as possible. However, the strategy's effectiveness is limited, with a maximum reduction of message exchanges by 17% compared to the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`). Additionally, the `retrain_short` strategy performs worse than the `repeat` baseline in the scenarios ($\rho = 1800$, $\mathcal{T} =$ `TL_low`), ($\rho = 3600$, $\mathcal{T} =$ `TL_medium`), and ($\rho = 3600$, $\mathcal{T} =$ `TL_low`).

If the threshold metric is lower than `TL_high`, the accuracy gain from model updates in the `retrain_*` strategies outweighs the cost of data transfer of model updates, i.e.,

they outperform the `static` strategy. Nonetheless, all strategies still require at least 3% more data transfer than the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`).

6.2.3 vienna_201907_201912

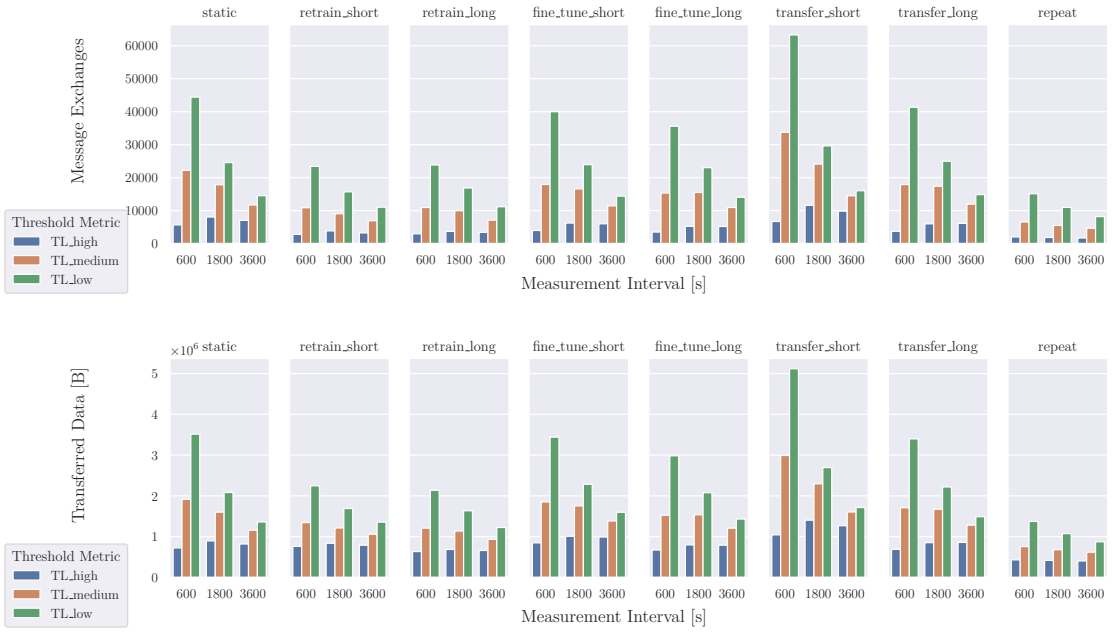


Figure 6.7: Results for LSTM model with vienna_201907_201912 dataset

In the simulation scenario with a limited training dataset of only half a year, the performance trend observed with reduced training data persists (Figure 6.7). None of the strategies can outperform the `repeat` baseline in terms of either message exchanges or transferred data.

Of the strategies analyzed, the `retrain_short` strategy requires the least number of exchanged messages, but still at least 38% more than the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`). In terms of transferred data, the best-case scenario with the `retrain_long` strategy still requires at least 47% more data than the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`).

Compared to the Dense model, the LSTM model, therefore, experiences a far greater decline in performance, similar to the scenarios with one entire year of training data. However, all scenarios and update strategies require still fewer data transferred in total than the `naive` baseline, showing that prediction-based data reduction using NNs still provides benefits in the lack of alternatives.

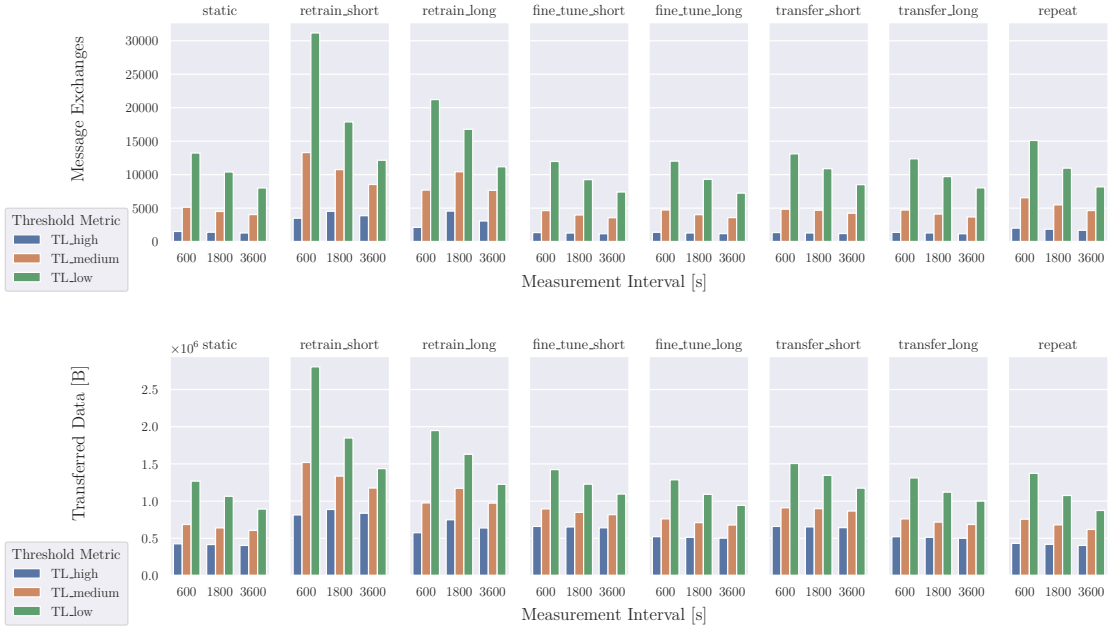


Figure 6.8: Results for LSTM model with linz_2010_2019 dataset

6.2.4 linz_2010_2019

In the simulation scenarios using the training data from Linz, transfer learning significantly improves the performance of the LSTM model (Figure 6.8). Across all scenarios, the best-performing strategy outperforms the best-performing strategy in equivalent scenarios using the vienna_2019_2019 or vienna_201907_201912 datasets. However, the transfer learning approach still performs worse than using the entire dataset from Vienna in all scenarios.

Similar to the Dense model, the `fine_tune_*` strategies perform best with transfer learning, while the `retrain_*` strategies struggle with limited training data. Compared to the `repeat` baseline, we can reduce the number of message exchanges by up to 33% using the `fine_tune_short` strategy ($\rho = 600$, $\mathcal{T} =$ `TL_high`). Considering the transferred data, only the `static` strategy can improve upon the `repeat` baseline in 8 out of 9 scenarios with a maximum reduction of 8% ($\rho = 600$, $\mathcal{T} =$ `TL_high`).

In all scenarios, the LSTM model performs better than the Dense model with the same training dataset in terms of message exchanges, achieving a relative reduction of up to 5% ($\rho = 600$, $\mathcal{T} =$ `TL_high`). However, due to its smaller model size, the Dense model leads to better results regarding the transferred data, with the LSTM model requiring at least 6% more ($\rho = 1800$, $\mathcal{T} =$ `TL_medium`). Hence, choosing a model with higher architectural complexity for transfer learning requires a trade-off between message exchanges and data transfer, which depends on the application context.

6.3 ConvLSTM model

In the final simulation scenarios, the ConvLSTM model provides the predictions. The reduced TensorFlow Lite model has a size of 119,736 bytes, roughly three times larger than the LSTM model and almost 17 times larger than the Dense model. We will present the results for each dataset and provide additional details in Tables B.10 through B.13.

6.3.1 vienna_2010_2019

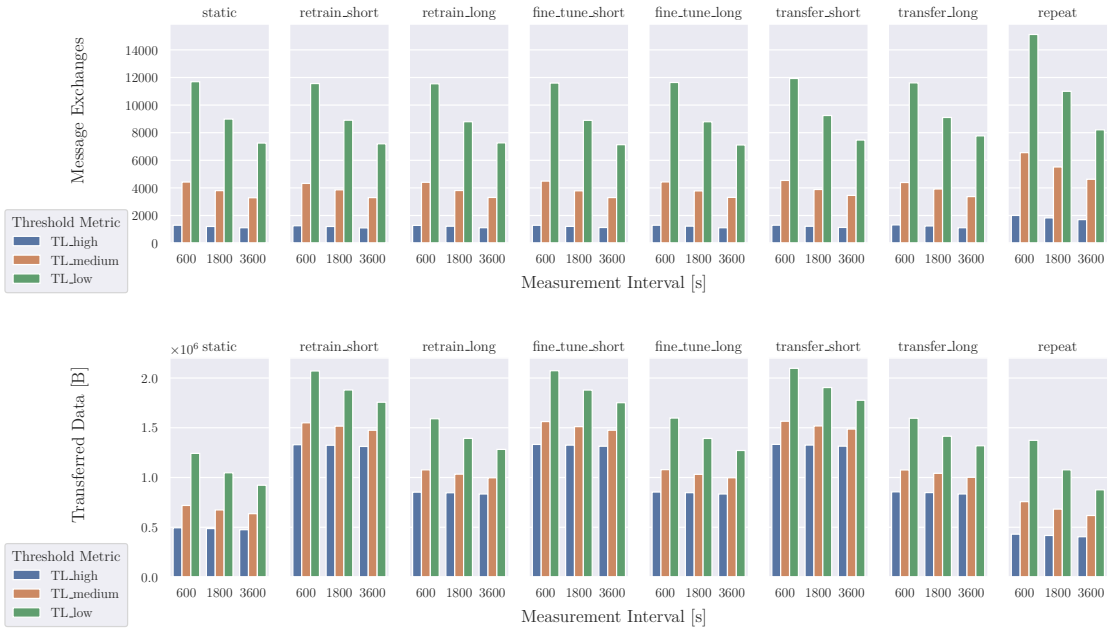


Figure 6.9: Results for ConvLSTM model with vienna_2010_2019 dataset

Similar to the Dense and LSTM models, all strategies demonstrate similar performance regarding message exchanges (Figure 6.9). Moreover, the ConvLSTM model manages to provide the best predictions given a large training dataset and can reduce the messages by up to 38% compared to the `repeat` baseline, the largest improvement observed in all simulations ($\rho = 600$, $\mathcal{T} =$ `TL_high`). Compared to the `oracle` baseline, it requires only 72% more messages and reduces the number of messages compared to the `naive` baseline by 99.8%.

Due to the large model size, all strategies require more data transfer than the `repeat` baseline if they deploy the model more than once. Therefore, only the `static` strategy requires less transferred data than the `repeat` baseline (up to a 5% reduction for $\rho = 600$ and $\mathcal{T} =$ `TL_low`). The relative improvement worsens with increasing threshold metrics or measurement intervals.

Among the best-performing strategies are the `static`, `retrain_*`, and `fine_tune_*` strategies, with only minimal differences among them. As the threshold metric decreases, more improvement can be expected by updating the model regarding the number of exchanged messages. However, in scenarios $\rho = 3600$ and $\mathcal{T} =$ `TL_low`, all strategies with short deployment intervals require more data transfer than the `naive` baseline, making them an expensive trade-off for minimizing the number of message exchanges.

6.3.2 vienna_2019_2019

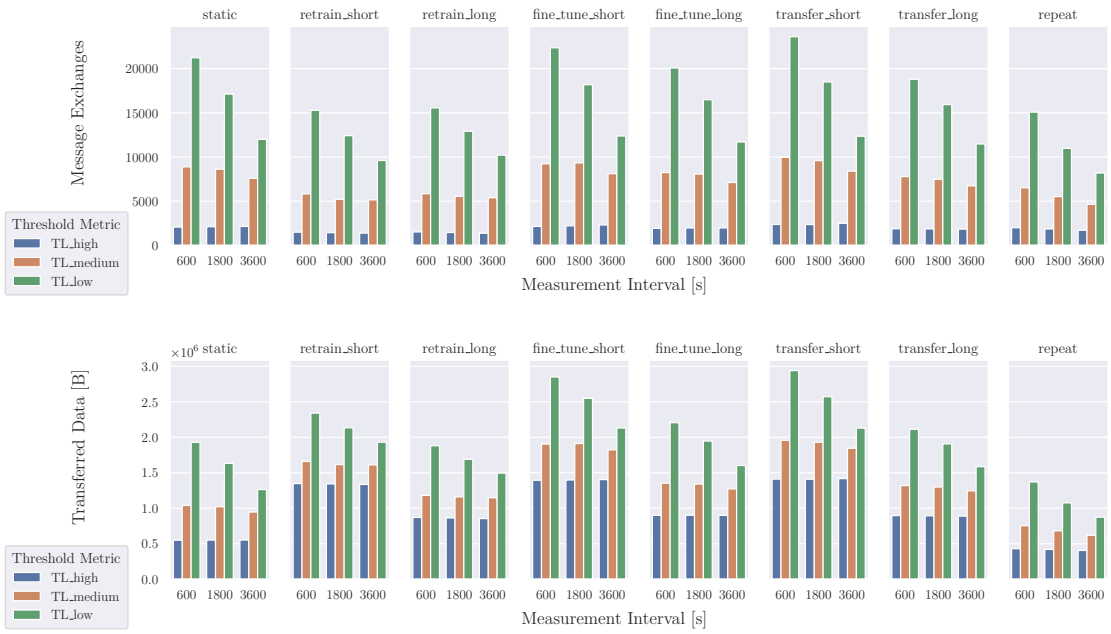


Figure 6.10: Results for ConvLSTM model with vienna_2019_2019 dataset

As observed with the LSTM model, prediction models with higher architectural complexity require more training data for adequate performance. As demonstrated in Figure 6.10, the ConvLSTM model performs poorly compared to the `repeat` baseline when only one year of training data is available. Like the LSTM model, the Dense model outperforms the ConvLSTM model in all scenarios.

Of the strategies analyzed, only the `retrain_*` strategies were able to reduce the number of message exchanges compared to the `repeat` baseline, achieving a reduction of up to 25% ($\rho = 600$, $\mathcal{T} =$ `TL_high`). However, for the `TL_low` threshold metric, the performance of these strategies is worse than the baseline in all scenarios. In terms of transferred data, the best-performing strategy still requires an increase of at least 28% compared to the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`).

For scenarios with hourly measurement intervals ($\rho = 3600$) and lower threshold metrics ($\mathcal{T} \neq$ `TL_high`), 8 out of 14 strategies require more data transfer than the `naive`

baseline while performing worse than the `repeat` baseline regarding message exchanges. Therefore, the ConvLSTM model becomes unfeasible for continuous deployment in such scenarios.

6.3.3 vienna_201907_201912

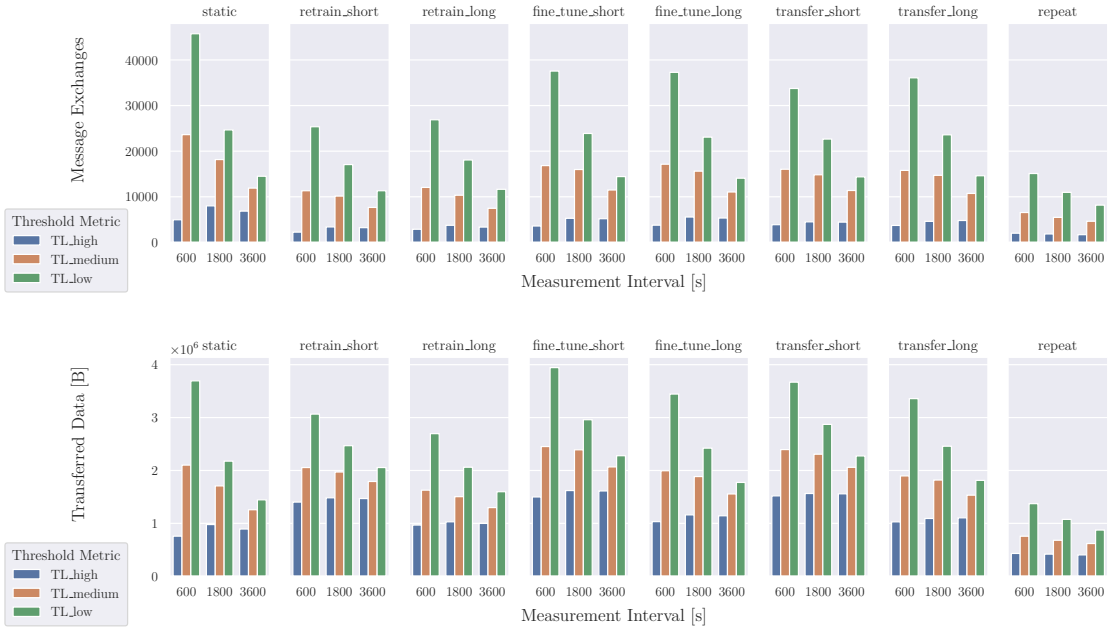


Figure 6.11: Results for ConvLSTM model with vienna_201907_201912 dataset

As anticipated, the performance further deteriorates when only minimal training data is available. As shown in Figure 6.11, no strategy can outperform the `repeat` baseline in any scenario. Intuitively, the `retrain_*` strategies can adapt the fastest to newly accumulated data; however, this does not compensate for the poor predictions made during the initial period of the simulation. The increase in message exchanges is at least 12%, and the increase in transferred data is at least 76% due to the large model compared to the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`).

It is worth noting that, in contrast to the simulations with the vienna_2019_2019 dataset, the best-performing strategies for the ConvLSTM model outperform the best-performing strategies for the LSTM model in scenarios with threshold metric `TL_high`. However, the Dense model still demonstrates the best performance among all models in all scenarios with only half a year of training data.

As observed with the vienna_2019_2019 dataset, the ConvLSTM model requires more data transfer than the `naive` baseline in scenarios with $\rho = 3600$ for most strategies with continuous model deployment.

6.3.4 linz_2010_2019

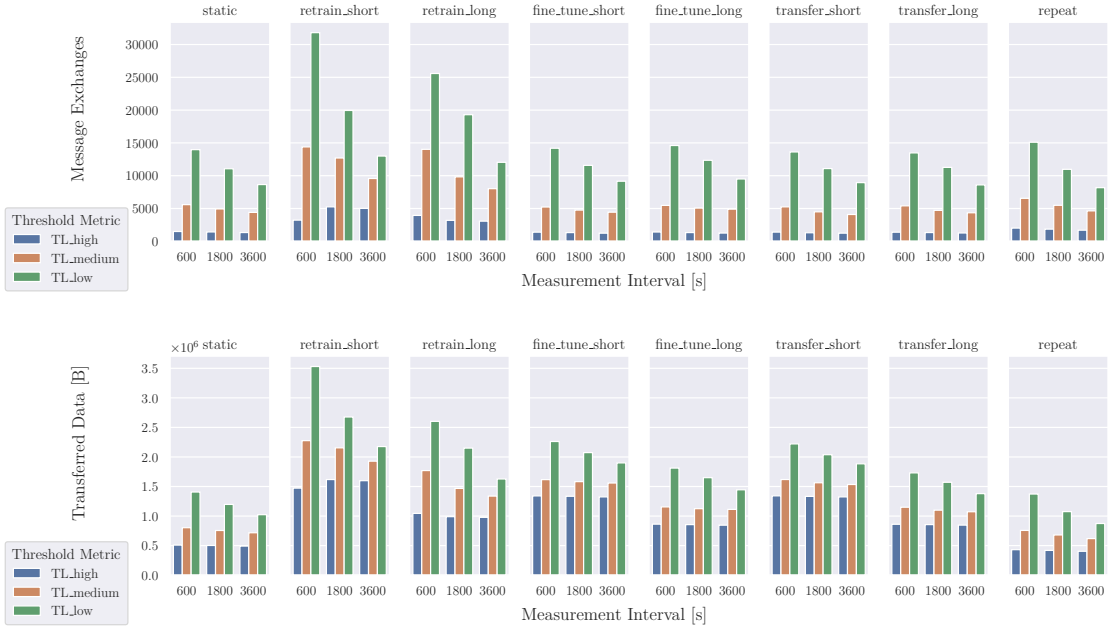


Figure 6.12: Results for ConvLSTM model with linz_2010_2019 dataset

Similar to the other models, transfer learning with the dataset from Linz improves the performance of the ConvLSTM model. In all scenarios, the best-performing strategy using the linz_2010_2019 dataset outperformed the best-performing strategies using the vienna_2019_2019 and vienna_201907_201912 datasets for the same scenario. However, the results are not better than those obtained with the vienna_2010_2019 dataset.

The `transfer_*` and `fine_tune_*` strategies performed particularly well with the ConvLSTM model, with the former demonstrating strong performance at high threshold metrics. In the most favorable scenario, we observe a reduction in exchanged messages by 31% compared to the `repeat` baseline ($\rho = 600$, $\mathcal{T} =$ `TL_high`). However, the ConvLSTM model requires at least 18% more data transferred across all scenarios.

In contrast to the Dense and LSTM models, the ConvLSTM does not outperform the `repeat` baseline with threshold metric `TL_low` and a measurement interval of 3600 seconds, suggesting the quality of the training data is not sufficient to train the complex model. Even though the `transfer_long` strategy can improve against the `static` strategy, the training data seems insufficient for this scenario. Moreover, although the performance improves compared to the scenarios with limited training data, the ConvLSTM still requires more data transfer than the `naive` baseline in scenarios with $\rho = 3600$ and short deployment intervals.

6.4 Discussion

After discussing the simulation results for every model and dataset individually, we will conclude this chapter with a general discussion of the parameter study. In the following, we will synthesize and analyze the results, drawing conclusions and implications from our findings.

Training Dataset	Dense	LSTM	ConvLSTM
vienna_2010_2019	37%	37%	38%
vienna_2019_2019	33%	17%	25%
vienna_201907_201912	-4%	-38%	-12%
linz_2010_2019	30%	33%	31%

Table 6.3: Best-case reduction in message exchanges compared to `repeat` baseline

Training Dataset	Dense	LSTM	ConvLSTM
vienna_2010_2019	22%	19%	5%
vienna_2019_2019	14%	-3%	-28%
vienna_201907_201912	-12%	-47%	-76%
linz_2010_2019	15%	8%	-18%

Table 6.4: Best-case reduction in transferred data compared to `repeat` baseline

Overview

Tables 6.3 and 6.4 provide an overview of the achievable data reduction rates in the best-performing scenarios for each model-dataset combination compared to the `repeat` baseline.

Overall, the ConvLSTM model provides the best predictions given a large training dataset and can reduce the messages by up to 38%, the largest improvement observed in all simulations. Compared to the `naive` baseline, it can reduce the message exchanges by up to 99.8%. Regarding the total data transfer, the Dense model demonstrates the best results due to its smaller model size. It can reduce the overall communication by up to 22% compared to the `naive` baseline and by up to 92% compared to the `naive` baseline.

With the exceptions discussed for the results of the ConvLSTM model (Section 6.3), all models outperform the `naive` baseline in terms of message exchanges and transferred data across all scenarios. In the remaining section, we will discuss various aspects of the parameter study.

Training Dataset

The quality and quantity of the available training dataset play an essential role in the efficiency of prediction-based data reduction. As we have observed in our simulations, the performance worsens significantly if only limited training data, such as an entire year or half a year of the deployment location, is available. Even models with low architectural complexity (Dense model) require more than half a year of training data to outperform the `naive` baseline.

Statically deployed prediction models trained with limited training data can still perform better than the `repeat` baseline in scenarios with small measurement intervals and low threshold metrics. Continuous model updates can improve the performance of these models, leading to fewer message exchanges. However, it often requires a trade-off between message exchanges and transferred data because even the deployment of models with low architectural complexity (Dense model) requires more data transfer in total compared to the `repeat` baseline.

Threshold Metrics

The results of our simulations with the vienna_2010_2019 dataset suggest that as the threshold metric increases, more complex models tend to perform better regarding the number of message exchanges. When the threshold metric is set to `TL_low`, the Dense model performs the best, while the LSTM model performs the best when the threshold is set to `TL_medium` (except for the scenario with hourly measurement intervals). When the threshold is set to `TL_high`, the ConvLSTM model demonstrates the best performance. This is likely due to the fact that the increased complexity of these models leads to improved long-term predictions, which do not necessarily offer an advantage when threshold violations occur in the short term. Similar observations have been made in [KMS20], where more complex models lead to better predictions for prediction horizons greater than 6 hours.

Measurement Interval

In general, the parameter study demonstrates that the more the measurement interval of the sensor node decreases, the more prediction-based data reduction can minimize communication. All best-case reduction rates depicted in Tables 6.3 and 6.4 originate from scenarios with a measurement interval of 600 seconds, the lowest value in our parameter study.

Model Complexity

The Dense model is the best choice if (1) reducing transferred data is more important than reducing message exchanges and (2) insufficient training data is available. Across all scenarios, the Dense model requires less transferred data than the LSTM or ConvLSTM models. Although they may require fewer message exchanges, the larger model size always

cancels out the increase in performance. Additionally, when only limited training data is available, the best-performing strategies with the Dense model are never outperformed by any strategy using another model.

Considering the computational complexities of the prediction models analyzed in Section 5.3.2, the Dense model provides further benefits by requiring less than 1% of the operations to compute a prediction horizon compared to the LSTM and ConvLSTM models. Depending on the hardware capabilities of the sensor nodes, this reduction in required computation can further increase the efficiency of less complex models. However, if reducing the number of message exchanges is paramount, complex model architectures can still be valuable through increased prediction performance.

Transfer Learning

Transfer learning with a similar dataset is worthwhile if limited training data is available. For the more complex LSTM and ConvLSTM models, using the historical measurements from Linz leads to better results than using the limited training datasets from the deployment location. The smaller Dense model seems to adapt quickly enough, so transfer learning only leads to better performance compared to the scenarios with half a year of training data. However, using the linz_2010_2019 dataset, the best-performing strategies with the LSTM model outperform the Dense model strategies with the vienna_2019_2019 dataset in 7 out of 9 scenarios. Hence, choosing a model with higher architectural complexity for transfer learning requires a trade-off between message exchanges and data transfer, which depends on the application context.

Applying transfer learning with a similar dataset demonstrated a comparable reduction in message exchanges to using the entire vienna_2010_2019 dataset, as depicted in Table 6.3. However, this is not the case regarding the transferred data. Since the strategies that reduce the message exchanges the most rely on continuous model deployment, they require more data transferred in total.

Continuous Model Updates

The various model update strategies demonstrate different advantages and disadvantages depending on the available training data and the complexity of the model architecture. In scenarios with limited training data, the `retrain_*` strategies tend to produce the best results due to their ability to quickly incorporate new data. However, in scenarios with the full vienna_2010_2019 dataset, the models may also benefit from short-term adaptations using `retrain_*` or `fine_tune_*` strategies. In most cases, `retrain_short` outperforms `retrain_long` in terms of exchanged messages, but at the cost of increased transferred data due to more frequent model updates. However, in the case of transfer learning, the `retrain_*` strategies fail due to a severe lack of available training data in the initial simulation period.

When utilizing transfer learning, the effectiveness of the model update strategies depends on the model architecture. For the linz_2010_2019 dataset, the LSTM model and the

Dense model demonstrate the best results with the `fine_tune_*` strategies, while the more complex ConvLSTM model benefits from the `transfer_*` strategies. Although the `fine_tune_*` strategies work comparably well in scenarios with `TL_high` metric, they perform worse than `static` strategies if a different threshold is applied. Contrary to expectations, the best-performing `fine_tune_*` strategies with the ConvLSTM model are outperformed by the best-performing `fine_tune_*` strategies with the LSTM model.

When sufficient training data is available (vienna_2010_2019 dataset), the `static` strategies always require the least data transferred in total. However, other strategies demonstrate a reduction in message exchanges by up to 3% in the same scenario. It depends on the application context whether the data transfer cost outweighs the higher number of messages. Given less training data, the performance gain through model updates often compensates for the cost of the model transfer, leading to less data transferred in total compared to a static deployment. Hence, continuous model updates can increase the energy efficiency of sensor nodes in prediction-based data reduction, especially with sparse training data.

CHAPTER 7

Conclusion

In the final chapter of this thesis, we present a summary of our key findings and discuss the applicability and limitations of the proposed framework. We conclude the thesis by providing recommendations for future research in the field of prediction-based data reduction, highlighting the potential for further extensions of the proposed approach to increase energy efficiency in IoT environments.

7.1 Summary

In this thesis, we presented SenseReduce, an open-source state-of-the-art framework for prediction-based data reduction in IoT environments with continuous model updates. We presented its data reduction algorithm based on a dual prediction scheme and the mechanisms for continual learning and continuous deployment. The framework's modular architecture allows for high customization to different application scenarios.

Using an extensive parameter study, we evaluated SenseReduce in the context of air temperature monitoring. We designed three prediction models based on neural networks using hyperparameter optimization and showed that their architectural and computational complexities depend on the applied technique. We used four training datasets of varying length and quality for a simulated deployment over two years, demonstrating the effects of different model update strategies, threshold metrics, and measurement intervals.

We significantly reduced the required communication using neural networks as multivariate prediction models. We have shown that prediction-based data reduction can reduce the number of exchanged messages by up to 38% and the amount of transferred data by up to 22% compared to a baseline using univariate trend extrapolation. Compared to constant reporting, our proposed approach can reduce the exchanged messages by up to 99.8% and the data transfer by up to 92%.

The result analysis has shown that optimizing prediction-based data reduction is highly context-dependent. Continuous model updates improve the performance of prediction models over time, especially if training data is sparse. However, the best-performing model update strategy is influenced by the measurement frequency, the threshold metric, and the utilized model architecture.

Additionally, the results demonstrated that the quantity and quality of training data are crucial factors for the performance of neural network models. While optimizing model architectures can lead to improved performance, simpler models with larger amounts of training data often surpass these gains. This highlights the importance of acquiring and preprocessing high-quality data for machine learning applications.

We conclude with the following recommendations for future applications of prediction-based data reduction, in particular for air temperature monitoring:

- If only limited training data is available, using transfer learning to build a model on a similar dataset leads to better results than relying on continual learning to improve the performance over time.
- More complex model architectures can provide benefits for scenarios with transfer learning. If enough training data is available, they only improve performance in scenarios with more coarse-grained threshold metrics.
- The higher the measurement frequency, the more relative data reduction can be expected from using more sophisticated prediction models like neural networks compared to baseline approaches.
- Continuous model updates reduce the total data transferred if training data is sparse; otherwise, it only helps reduce the number of message exchanges. The increase in prediction performance rarely compensates the model transfer cost.

7.2 Limitations and Applicability

We anticipate potential limitations and applicability of our proposed SenseReduce framework. Thus, in this section, we discuss and provide possible solutions to them.

Multidimensional Data

The SenseReduce framework is intended for two-dimensional time series data, making it suitable for various application scenarios such as environmental monitoring, industrial control, and smart building systems. However, although it supports scenarios where multiple attributes are measured and observed simultaneously by supporting multivariate prediction models, it is not designed to be used with sequential image data such as image classification or video analysis. Adapting the framework to handle such data would require defining appropriate threshold metrics and either reducing the dimensionality

of image data or increasing the dimensionality of prediction model inputs and outputs. Due to the absence of a relevant application scenario, we did not include it within the scope of this thesis.

Additionally, the proposed framework requires each connected sensor node to monitor the same attributes. This design choice is based on the typical application scenarios of WSNs. However, it is possible to operate multiple node managers within a single base station to support multiple application scenarios with one physical device.

Communication Efficiency

As highlighted in our parameter study, implementing prediction-based data reduction may increase communication compared to a basic baseline that does not utilize neural networks. The point at which the SenseReduce framework demonstrates advantages in terms of energy efficiency is highly dependent on the context, and we will briefly discuss the relevant considerations.

Prediction-based data reduction should improve the overall communication efficiency compared to the `naive` baseline discussed in Section 6. In particular, the total number of messages can be reduced as long as a given prediction model can accurately predict at least one measurement during each model deployment, as outlined in Equation 6 In terms of overall data reduction, we first look at the total data transferred with the `naive` baseline. Given a deployed sensor node where Δt is the duration of deployment, ρ is the measurement interval, p_d is the size of a data point, and p_h is the packet overhead, the total amount of transferred data in bytes is given by

$$D_{naive} = \frac{\Delta t}{\rho}(p_d + p_h). \tag{7.1}$$

Similarly, for prediction-based data reduction, we discussed the computation of the transferred data in Equation 6. Combining both equations, prediction-based data reduction leads to a decrease in overall transferred data if the following inequality holds:

$$\frac{\Delta t}{\rho}(p_d + p_h) < \sum_{v \in V}(p_h + elapsed(v) \cdot p_d) + n_u(p_h + h \cdot p_d) + n_d(p_h + |\mathcal{M}|). \tag{7.2}$$

However, the resulting threshold violations V are dependent on both the defined threshold metric $\mathcal{T}$ and the performance of the prediction model $\mathcal{M}$. The number of horizon updates depends on the defined prediction horizon length η, and the number of model deployments n_d depends on the chosen model update strategy. Additionally, the packet overhead p_h depends on the chosen communication protocol, and the data point size p_d is determined by the monitored data. These parameters must be defined for a specific application and chosen appropriately based on the underlying data distribution and application requirements.

Cost of Continual Learning

In addition to the parameters discussed, the cost of continual learning for prediction models must also be considered in evaluating the overall performance of the proposed framework. The cost of continual learning is determined by multiple factors, including the neural network architecture, deployment strategy, and the specific continual learning method implemented, and thus cannot be generalized across all cases. Nevertheless, although a significant factor, the continual learning cost is often insignificant in practice. This is because continual learning is performed on the base station, where computational resources are readily available, and the neural network models used for prediction tend to be smaller than other types of neural networks that require many resources for training.

Generic Applicability

The modular design of SenseReduce with generic interfaces allows for high customization to different prediction models, threshold metrics, and model update strategies. Therefore, the framework is applicable to various use cases in the context of wireless sensor networks and IoT sensor applications in general, not limited to air temperature monitoring. The essential requirement is that the observed data can be predicted with adequate accuracy using neural network models.

The proposed prediction-based data reduction mechanism builds on the general advantages neural network models provide. In addition to being able to handle non-linear data, the large number of trainable parameters enables neural networks to extract underlying patterns in multivariate input data without the need for domain-specific expert knowledge. In certain applications, such as climate prediction, neural networks have been shown to be a viable alternative to traditional scientific models while also offering reduced computational costs [RDK+22]. However, it should be noted that training neural network models requires a sufficient dataset of historical measurements or an approximation thereof, as demonstrated through our simulation scenarios with transfer learning. In cases where such data is not readily available, prediction models that require minimal or no training data are preferable. In light of these considerations, our proposed methodology, SenseReduce, represents a versatile tool for a wide range of applications, particularly in addressing the challenges of energy efficiency in IoT environments.

7.3 Future Work

Numerous avenues for further research could build upon the work presented in this thesis. First and foremost, exploring use cases in other domains will provide valuable insights. By making the SenseReduce framework available as open source, we aim to encourage other researchers to explore these possibilities in greater depth. In the following section, we will outline other areas for future study to conclude this thesis.

Multiple Sensor Nodes

Our simulations focused on a network with a single sensor node, though SenseReduce can be used with multiple sensor nodes. In such cases, it may be possible to enhance model deployment strategies by aggregating data from multiple nodes. Centralized federated learning [DZF+20] could be employed to distribute model update computations across the sensor nodes, or prediction performance could benefit from centralized data collection and data fusion. Additionally, sensor nodes may directly utilize measurements from neighboring nodes to improve their predictions, as shown in [SM21].

Data Reduction

The proposed framework can be further optimized for data reduction through model and data compression techniques. Model compression methods, such as quantization and pruning, reduce the size and inference latency of prediction models [HMD16]. Additionally, various data compression techniques can be employed to reduce the size of transmitted data, depending on the application domain. For instance, symbolic representation for time series data [LKWL07] or the transfer of differences between subsequent measurements instead of the entire measurement can be used to reduce the size of messages.

Deployment Strategies

One potential avenue for future research is to investigate dynamic strategies for model deployment time. This could involve utilizing techniques such as change detection [DRAP15] and cost-performance optimization techniques, such as linear programming or reinforcement learning, to determine the optimal deployment time. However, it is important to consider the computational complexity of these techniques. Additionally, advanced continual learning methods, such as dynamic adaptation of model architecture and techniques like random or buffered replay, can be investigated to improve prediction performance and handle sparse training data.

Improving Prediction Performance

In future work, it may be worthwhile to investigate other model architectures, such as ensemble learning techniques and univariate models like ARIMA, while considering the suitability of each architecture for specific use cases. Given the challenge of limited training data in our parameter study, simpler prediction models (like the `repeat` baseline) could be employed until enough training data has been gathered to train a neural network. Another idea would be to use transfer learning combined with fine-tuning on a sample of historical measurements to train a prediction model before the first deployment.

Additionally, data augmentation and feature engineering techniques could be employed to increase the available training data and enhance performance. Furthermore, hyperparameter optimization of the proposed framework can be conducted to determine optimal values for parameters such as prediction horizon, data aggregation period, and hyperparameters related to continual learning, such as learning rate and validation split.

Fault Tolerance

In the proposed framework, fault tolerance for sensor nodes is incorporated to a limited extent. The expected communication between sensor nodes and the base station is restricted to the horizon update, which depends on the defined prediction horizon length. However, if sensor nodes fail in between these updates, the base station may not be aware of the failure; on the contrary: it assumes that the current predictions match the sensor node's measurements. One approach to increase resiliency is to decrease the prediction horizon length, which may also increase required communication.

The required level of resiliency is highly dependent on the specific application scenario. For instance, in a densely deployed wireless sensor network in an agricultural field, the failure of a single node may be insignificant. However, in scenarios where fault tolerance is paramount, sensor nodes may implement a heartbeat signal independent of the framework to ensure their status is known. Hence, the trade-off between the required level of resiliency and the communication overhead must be carefully evaluated for each use case.

Security

In this thesis, we did not address security concerns related to the data transfer from sensor nodes to the base station. The implementation of encryption can result in additional computational overhead and energy consumption. The necessary level of security in IoT environments is highly dependent on the specific use case, making it challenging to develop a universally applicable technique. For example, using the HTTP/S protocol can provide the required level of security but increases packet overhead and computational complexity. In some cases, symmetric key encryption with manually deployed keys may be a lightweight and feasible option, but key storage and management can pose issues in low-capacity devices [SSH+18].

APPENDIX A

Hyperparameter Optimization

As discussed in Section 5.3.1, we used the KerasTuner library for hyperparameter optimization during our model design process. More specifically, we applied the provided implementation of the Hyperband algorithm [LJD+18] with the parameters listed in Table A.1. The `val_loss` objective is equal to the WMSE metric described in Section 5.2.1.

Parameter	**Value**
Objective	`val_loss`
Max Epochs	150
Factor	3
Hyperband Iterations	3

Table A.1: Hypberband parameters for model optimization

In Tables A.2, A.3, and A.4, we list the search space used as input to KerasTuner for the Dense, LSTM, and ConvLSTM model, respectively. We denote multiple options within the search space with the list notation $[\cdot]$. The $logRange(x, y)$ function samples with equal probabilities from each order of magnitude range between x and y.

Layer	Parameter	Values
Input	Sequence Length	[1,2,3,4,5,6]
dense1	Units	[8,16,24,32]
	Activation	[relu, tanh, linear]
dense2	Units	24
	Activation	linear
Optimizer	Learning Rate	logRange(1e-8, 1e-2)
	Optimizer	[adam, rmsprop]

Table A.2: Search space for Dense model

Layer	Parameter	Values
Input	Sequence Length	[3, 6, 12, 18, 24, 36, 48, 72, 96, 120]
lstm	Units	[8,16,24,32,40,48,56,64]
	Recurrent Dropout	[0, 0.25, 0.5]
dropout	rate	[0, 0.1, 0.2, 0.3, 0.4, 0.5]
dense2	Units	24
	Activation	linear
Optimizer	Learning Rate	logRange(1e-8, 1e-2)
	Optimizer	[adam, rmsprop]

Table A.3: Search space for LSTM model

Layer	Parameter	Values
Input	Sequence Length	120
conv1	Kernel Size	3
	Filters	[8, 16, 24, 32]
conv2	Kernel Size	3
	Filters	[16, 24, 32]
conv3	Kernel Size	3
	Filters	[32, 48, 64]
conv4	Kernel Size	3
	Filters	[64, 128]
MaxPool1D	Pool Size	2
lstm	Units	[16, 24, 32]
dense	Units	24
	Activation	linear
Optimizer	Learning Rate	logRange(1e-8, 1e-3)
	Optimizer	[adam, rmsprop]

Table A.4: Search space for ConvLSTM model

APPENDIX B

Parameter Study Results

For reference, we provide the results for every scenario simulated in our parameter study. We group the result tables by prediction model and training dataset; Table B.1 presents an overview for orientation.

Training Dataset	**Dense**	**LSTM**	**ConvLSTM**
vienna_2010_2019	Table B.2	Table B.6	Table B.10
vienna_2019_2019	Table B.3	Table B.7	Table B.11
vienna_201907_201912	Table B.4	Table B.8	Table B.12
linz_2010_2019	Table B.5	Table B.9	Table B.13

Table B.1: Overview of detailed result tables

The variables in the table headers are as defined in the previous chapters, with $data_d$ representing the sum of transferred data in bytes required only for deploying the prediction models.

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	941	338	1	7160	380608	0.8736	1.1758	1.0843
600	TL_high	retrain_short	937	332	8	57280	430408	0.8783	1.1884	1.0901
600	TL_high	retrain_long	938	348	4	28640	402762	0.8660	1.1665	1.0801
600	TL_high	fine_tune_short	992	333	8	57280	434517	0.8919	1.2231	1.1059
600	TL_high	fine_tune_long	939	339	4	28640	402240	0.8733	1.1755	1.0842
600	TL_high	transfer_short	1374	254	8	57280	456293	0.9848	1.4411	1.2005
600	TL_high	transfer_long	939	339	4	28640	402240	0.8733	1.1755	1.0842
600	TL_medium	static	4229	28	1	7160	594205	0.4204	0.2737	0.5232
600	TL_medium	retrain_short	4267	36	8	57280	648165	0.4219	0.2755	0.5249
600	TL_medium	retrain_long	4252	28	4	28640	617549	0.4225	0.2761	0.5254
600	TL_medium	fine_tune_short	4426	26	8	57280	658850	0.4219	0.2768	0.5261
600	TL_medium	fine_tune_long	4230	28	4	28640	615973	0.4203	0.2737	0.5232
600	TL_medium	transfer_short	6062	15	8	57280	775821	0.4648	0.3214	0.5669
600	TL_medium	transfer_long	4230	28	4	28640	615973	0.4203	0.2737	0.5232
600	TL_low	static	10849	0	1	7160	1068986	0.2538	0.1087	0.3297
600	TL_low	retrain_short	10791	0	8	57280	1115434	0.2519	0.1080	0.3286
600	TL_low	retrain_long	10870	1	4	28640	1092266	0.2538	0.1089	0.3301
600	TL_low	fine_tune_short	11609	1	8	57280	1174400	0.2578	0.1119	0.3346
600	TL_low	fine_tune_long	10851	0	4	28640	1090826	0.2538	0.1087	0.3297
600	TL_low	transfer_short	16891	0	8	57280	1554645	0.2958	0.1390	0.3729
600	TL_low	transfer_long	10851	0	4	28640	1090826	0.2538	0.1087	0.3297
1800	TL_high	static	865	358	1	7160	374840	0.9068	1.3167	1.1475
1800	TL_high	retrain_short	863	355	8	57280	425440	0.9228	1.3511	1.1624
1800	TL_high	retrain_long	895	340	4	28640	397296	0.9128	1.3297	1.1531
1800	TL_high	fine_tune_short	889	362	8	57280	427800	0.9392	1.3858	1.1772
1800	TL_high	fine_tune_long	865	358	4	28640	396536	0.9068	1.3167	1.1475
1800	TL_high	transfer_short	1465	271	8	57280	462944	1.0943	1.7692	1.3301
1800	TL_high	transfer_long	865	358	4	28640	396536	0.9068	1.3167	1.1475
1800	TL_medium	static	3579	35	1	7160	547728	0.5032	0.4239	0.6510
1800	TL_medium	retrain_short	3545	36	8	57280	595976	0.5045	0.4256	0.6523
1800	TL_medium	retrain_long	3567	41	4	28640	568912	0.5006	0.4202	0.6483
1800	TL_medium	fine_tune_short	3728	36	8	57280	609520	0.5108	0.4339	0.6587
1800	TL_medium	fine_tune_long	3579	35	4	28640	569424	0.5032	0.4239	0.6510
1800	TL_medium	transfer_short	6450	16	8	57280	803680	0.6176	0.5986	0.7737
1800	TL_medium	transfer_long	3579	35	4	28640	569424	0.5032	0.4239	0.6510
1800	TL_low	static	8003	1	1	7160	864136	0.3513	0.2389	0.4888
1800	TL_low	retrain_short	8113	1	8	57280	922680	0.3539	0.2422	0.4921
1800	TL_low	retrain_long	7924	2	4	28640	880216	0.3511	0.2401	0.4900
1800	TL_low	fine_tune_short	8688	2	8	57280	964152	0.3642	0.2508	0.5008
1800	TL_low	fine_tune_long	8001	1	4	28640	885688	0.3512	0.2389	0.4888
1800	TL_low	transfer_short	14388	1	8	57280	1374504	0.4945	0.4076	0.6384
1800	TL_low	transfer_long	8001	1	4	28640	885688	0.3512	0.2389	0.4888
3600	TL_high	static	774	354	1	7160	363104	0.9570	1.5567	1.2477
3600	TL_high	retrain_short	769	356	8	57280	413480	0.9533	1.5454	1.2432
3600	TL_high	retrain_long	769	348	4	28640	384104	0.9567	1.5539	1.2466
3600	TL_high	fine_tune_short	792	350	8	57280	414800	0.9810	1.6079	1.2680
3600	TL_high	fine_tune_long	774	354	4	28640	384800	0.9570	1.5567	1.2477
3600	TL_high	transfer_short	1567	249	8	57280	464944	1.2168	2.2662	1.5054
3600	TL_high	transfer_long	774	354	4	28640	384800	0.9570	1.5567	1.2477
3600	TL_medium	static	3080	39	1	7160	511560	0.6083	0.6691	0.8180
3600	TL_medium	retrain_short	3032	42	8	57280	558880	0.6083	0.6682	0.8174
3600	TL_medium	retrain_long	3082	46	4	28640	533808	0.6081	0.6702	0.8187
3600	TL_medium	fine_tune_short	3269	34	8	57280	575512	0.6299	0.7038	0.8389
3600	TL_medium	fine_tune_long	3080	39	4	28640	533256	0.6083	0.6691	0.8180
3600	TL_medium	transfer_short	5689	19	8	57280	748880	0.8274	1.1321	1.0640
3600	TL_medium	transfer_long	3080	39	4	28640	533256	0.6083	0.6691	0.8180
3600	TL_low	static	6166	2	1	7160	731920	0.4788	0.4747	0.6890
3600	TL_low	retrain_short	6273	4	8	57280	790360	0.4839	0.4793	0.6923
3600	TL_low	retrain_long	6251	2	4	28640	759736	0.4829	0.4789	0.6920
3600	TL_low	fine_tune_short	6833	2	8	57280	830552	0.5120	0.5191	0.7205
3600	TL_low	fine_tune_long	6166	2	4	28640	753616	0.4788	0.4747	0.6890
3600	TL_low	transfer_short	9805	1	8	57280	1044480	0.7618	1.0293	1.0145
3600	TL_low	transfer_long	6166	2	4	28640	753616	0.4788	0.4747	0.6890

Table B.2: Detailed results for Dense model with vienna_2010_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	1185	274	1	7204	393513	0.9381	1.3180	1.1481
600	TL_high	retrain_short	1078	303	8	57632	438938	0.9100	1.2527	1.1193
600	TL_high	retrain_long	1026	320	4	28816	407149	0.9029	1.2375	1.1124
600	TL_high	fine_tune_short	1193	268	8	57632	444557	0.9294	1.2978	1.1392
600	TL_high	fine_tune_long	1154	280	4	28816	413533	0.9216	1.2839	1.1331
600	TL_high	transfer_short	2746	160	8	57632	548506	1.0532	1.6081	1.2681
600	TL_high	transfer_long	1235	252	4	28816	417274	0.9459	1.3323	1.1543
600	TL_medium	static	5391	15	1	7204	677236	0.4477	0.3036	0.5510
600	TL_medium	retrain_short	4620	17	8	57632	672786	0.4329	0.2877	0.5363
600	TL_medium	retrain_long	4677	20	4	28816	647973	0.4309	0.2853	0.5342
600	TL_medium	fine_tune_short	5649	6	8	57632	746040	0.4515	0.3075	0.5546
600	TL_medium	fine_tune_long	5262	13	4	28816	689624	0.4456	0.3012	0.5489
600	TL_medium	transfer_short	9992	4	8	57632	1058592	0.5036	0.3772	0.6142
600	TL_medium	transfer_long	5529	11	4	28816	708685	0.4504	0.3068	0.5539
600	TL_low	static	14227	0	1	7204	1312305	0.2790	0.1263	0.3554
600	TL_low	retrain_short	11841	0	8	57632	1191442	0.2616	0.1140	0.3376
600	TL_low	retrain_long	12190	1	4	28816	1187530	0.2646	0.1160	0.3406
600	TL_low	fine_tune_short	14834	0	8	57632	1406941	0.2817	0.1285	0.3585
600	TL_low	fine_tune_long	13928	0	4	28816	1312605	0.2765	0.1244	0.3527
600	TL_low	transfer_short	26865	0	8	57632	2273173	0.3573	0.1950	0.4416
600	TL_low	transfer_long	14483	0	4	28816	1352562	0.2797	0.1276	0.3572
1800	TL_high	static	1091	298	1	7204	387316	0.9916	1.5065	1.2274
1800	TL_high	retrain_short	974	318	8	57632	431232	0.9649	1.4421	1.2009
1800	TL_high	retrain_long	944	343	4	28816	401768	0.9544	1.4153	1.1897
1800	TL_high	fine_tune_short	1154	281	8	57632	441824	1.0053	1.5382	1.2402
1800	TL_high	fine_tune_long	1104	293	4	28816	409680	0.9900	1.5076	1.2278
1800	TL_high	transfer_short	2459	190	8	57632	529152	1.1732	2.0483	1.4312
1800	TL_high	transfer_long	1201	258	4	28816	414144	1.0203	1.5810	1.2574
1800	TL_medium	static	4923	8	1	7204	642940	0.5645	0.5104	0.7144
1800	TL_medium	retrain_short	4126	25	8	57632	638048	0.5287	0.4571	0.6761
1800	TL_medium	retrain_long	4190	31	4	28816	613584	0.5305	0.4600	0.6783
1800	TL_medium	fine_tune_short	5335	3	8	57632	723184	0.5777	0.5320	0.7294
1800	TL_medium	fine_tune_long	4772	15	4	28816	654376	0.5575	0.5004	0.7074
1800	TL_medium	transfer_short	10317	5	8	57632	1082040	0.7576	0.8812	0.9387
1800	TL_medium	transfer_long	5090	16	4	28816	677328	0.5699	0.5229	0.7231
1800	TL_low	static	11902	0	1	7204	1144900	0.4328	0.3275	0.5722
1800	TL_low	retrain_short	9512	0	8	57632	1023752	0.3822	0.2710	0.5206
1800	TL_low	retrain_long	9465	0	4	28816	991256	0.3818	0.2705	0.5201
1800	TL_low	fine_tune_short	13070	0	8	57632	1279928	0.4544	0.3533	0.5944
1800	TL_low	fine_tune_long	11338	0	4	28816	1126120	0.4216	0.3146	0.5609
1800	TL_low	transfer_short	18662	0	8	57632	1682552	0.6268	0.6241	0.7900
1800	TL_low	transfer_long	11898	0	4	28816	1166440	0.4323	0.3278	0.5725
3600	TL_high	static	1060	280	1	7204	379676	1.0756	1.8424	1.3573
3600	TL_high	retrain_short	889	328	8	57632	420936	1.0170	1.6924	1.3009
3600	TL_high	retrain_long	934	311	4	28816	394168	1.0367	1.7365	1.3178
3600	TL_high	fine_tune_short	1150	260	8	57632	435968	1.1113	1.9393	1.3926
3600	TL_high	fine_tune_long	1036	280	4	28816	399760	1.0730	1.8354	1.3548
3600	TL_high	transfer_short	2605	169	8	57632	535616	1.3778	2.9221	1.7094
3600	TL_high	transfer_long	1150	256	4	28816	406624	1.1058	1.9444	1.3944
3600	TL_medium	static	4614	12	1	7204	620812	0.7417	0.9356	0.9672
3600	TL_medium	retrain_short	3450	33	8	57632	589096	0.6448	0.7407	0.8606
3600	TL_medium	retrain_long	3666	27	4	28816	575224	0.6643	0.7710	0.8780
3600	TL_medium	fine_tune_short	4965	9	8	57632	696848	0.7653	0.9826	0.9912
3600	TL_medium	fine_tune_long	4396	17	4	28816	627224	0.7235	0.8986	0.9480
3600	TL_medium	transfer_short	8628	7	8	57632	960472	1.1929	2.2908	1.5135
3600	TL_medium	transfer_long	4714	18	4	28816	650176	0.7472	0.9465	0.9729
3600	TL_low	static	9338	0	1	7204	960284	0.6797	0.8302	0.9111
3600	TL_low	retrain_short	7474	1	8	57632	877064	0.5550	0.6019	0.7758
3600	TL_low	retrain_long	7343	1	4	28816	838512	0.5464	0.5840	0.7642
3600	TL_low	fine_tune_short	9762	0	8	57632	1041744	0.7109	0.8828	0.9396
3600	TL_low	fine_tune_long	8944	0	4	28816	953744	0.6487	0.7706	0.8778
3600	TL_low	transfer_short	12699	0	8	57632	1253208	1.1988	2.3222	1.5239
3600	TL_low	transfer_long	9186	1	4	28816	971224	0.6711	0.8182	0.9045

Table B.3: Detailed results for Dense model with vienna_2019_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	3650	108	1	7204	558868	1.0774	1.6908	1.3003
600	TL_high	retrain_short	1828	262	8	57632	489936	0.9656	1.4053	1.1854
600	TL_high	retrain_long	2145	246	4	28816	482506	0.9779	1.4418	1.2008
600	TL_high	fine_tune_short	3480	53	8	57632	593413	1.0156	1.5633	1.2503
600	TL_high	fine_tune_long	3053	73	4	28816	535074	1.0388	1.5887	1.2604
600	TL_high	transfer_short	4556	46	8	57632	670474	1.1534	1.8840	1.3726
600	TL_high	transfer_long	3293	118	4	28816	555784	1.0630	1.6542	1.2862
600	TL_medium	static	13704	0	1	7204	1274633	0.5448	0.4379	0.6618
600	TL_medium	retrain_short	7173	22	8	57632	856968	0.4654	0.3281	0.5728
600	TL_medium	retrain_long	8113	10	4	28816	894637	0.4735	0.3424	0.5851
600	TL_medium	fine_tune_short	12241	0	8	57632	1220213	0.5166	0.4038	0.6355
600	TL_medium	fine_tune_long	12992	0	4	28816	1245152	0.5238	0.4178	0.6464
600	TL_medium	transfer_short	16603	0	8	57632	1534280	0.5913	0.5068	0.7119
600	TL_medium	transfer_long	13267	3	4	28816	1265197	0.5436	0.4354	0.6598
600	TL_low	static	28884	0	1	7204	2367596	0.3648	0.2026	0.4502
600	TL_low	retrain_short	16929	0	8	57632	1557752	0.2930	0.1393	0.3732
600	TL_low	retrain_long	17998	0	4	28816	1605618	0.2992	0.1463	0.3826
600	TL_low	fine_tune_short	25165	0	8	57632	2150757	0.3429	0.1824	0.4271
600	TL_low	fine_tune_long	26063	0	4	28816	2186309	0.3487	0.1887	0.4344
600	TL_low	transfer_short	43361	0	8	57632	3460858	0.4606	0.3102	0.5570
600	TL_low	transfer_long	29921	0	4	28816	2464085	0.3728	0.2107	0.4590
1800	TL_high	static	4103	103	1	7204	590388	1.3137	2.5736	1.6042
1800	TL_high	retrain_short	2046	252	8	57632	503608	1.0915	1.8415	1.3570
1800	TL_high	retrain_long	2208	255	4	28816	486272	1.1184	1.9157	1.3841
1800	TL_high	fine_tune_short	3403	58	8	57632	588136	1.2184	2.2702	1.5067
1800	TL_high	fine_tune_long	3256	68	4	28816	549488	1.2342	2.2823	1.5107
1800	TL_high	transfer_short	5777	42	8	57632	757584	1.4955	3.2737	1.8093
1800	TL_high	transfer_long	3776	124	4	28816	590096	1.2948	2.5066	1.5832
1800	TL_medium	static	12522	0	1	7204	1189516	0.8475	1.0880	1.0431
1800	TL_medium	retrain_short	6159	18	8	57632	783528	0.6057	0.6048	0.7777
1800	TL_medium	retrain_long	7692	15	4	28816	864624	0.6633	0.7222	0.8498
1800	TL_medium	fine_tune_short	11625	0	8	57632	1175864	0.8120	1.0022	1.0011
1800	TL_medium	fine_tune_long	10398	0	4	28816	1058416	0.7684	0.9470	0.9731
1800	TL_medium	transfer_short	16915	0	8	57632	1556752	1.0445	1.5826	1.2580
1800	TL_medium	transfer_long	13239	2	4	28816	1263104	0.8765	1.1489	1.0719
1800	TL_low	static	20436	0	1	7204	1759332	0.7143	0.8189	0.9049
1800	TL_low	retrain_short	12796	1	8	57632	1260240	0.4814	0.4276	0.6539
1800	TL_low	retrain_long	14552	0	4	28816	1357488	0.5282	0.5011	0.7079
1800	TL_low	fine_tune_short	19849	0	8	57632	1768000	0.6565	0.6768	0.8227
1800	TL_low	fine_tune_long	17741	0	4	28816	1587112	0.6342	0.6934	0.8327
1800	TL_low	transfer_short	24973	0	8	57632	2136928	0.9646	1.4059	1.1857
1800	TL_low	transfer_long	21195	0	4	28816	1835808	0.7391	0.8602	0.9275
3600	TL_high	static	4111	98	1	7204	589364	1.6462	4.1664	2.0412
3600	TL_high	retrain_short	1677	255	8	57632	473792	1.1916	2.3247	1.5247
3600	TL_high	retrain_long	2121	250	4	28816	476376	1.2668	2.6693	1.6338
3600	TL_high	fine_tune_short	3264	63	8	57632	577336	1.5150	3.5386	1.8811
3600	TL_high	fine_tune_long	3299	64	4	28816	550824	1.5234	3.5784	1.8917
3600	TL_high	transfer_short	6010	31	8	57632	773272	2.0822	6.6735	2.5833
3600	TL_high	transfer_long	3848	115	4	28816	593192	1.6298	4.0839	2.0209
3600	TL_medium	static	9642	1	1	7204	982212	1.3576	2.9713	1.7238
3600	TL_medium	retrain_short	5207	21	8	57632	714944	0.8297	1.2520	1.1189
3600	TL_medium	retrain_long	6194	14	4	28816	756512	0.9502	1.6640	1.2900
3600	TL_medium	fine_tune_short	9553	0	8	57632	1026680	1.2588	2.4486	1.5648
3600	TL_medium	fine_tune_long	9580	0	4	28816	999520	1.2781	2.5631	1.6010
3600	TL_medium	transfer_short	12207	0	8	57632	1217768	2.0014	6.2693	2.5039
3600	TL_medium	transfer_long	10013	3	4	28816	1030864	1.4158	3.1890	1.7858
3600	TL_low	static	13179	0	1	7204	1236820	1.2896	2.7717	1.6649
3600	TL_low	retrain_short	8998	0	8	57632	986720	0.7322	1.0654	1.0322
3600	TL_low	retrain_long	9960	0	4	28816	1026880	0.8624	1.4736	1.2139
3600	TL_low	fine_tune_short	13136	0	8	57632	1284656	1.1854	2.2324	1.4941
3600	TL_low	fine_tune_long	11984	0	4	28816	1172608	1.1212	2.2637	1.5045
3600	TL_low	transfer_short	15062	0	8	57632	1423328	2.0758	6.7238	2.5930
3600	TL_low	transfer_long	13496	0	4	28816	1281472	1.3819	3.1127	1.7643

Table B.4: Detailed results for Dense model with vienna_201907_201912 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	1219	278	1	7204	396417	0.9237	1.3059	1.1428
600	TL_high	retrain_short	2079	230	8	57632	505704	0.9632	1.4064	1.1859
600	TL_high	retrain_long	1564	243	4	28816	440685	0.9575	1.3758	1.1729
600	TL_high	fine_tune_short	1138	307	8	57632	443834	0.9240	1.3024	1.1412
600	TL_high	fine_tune_long	1119	297	4	28816	412552	0.9132	1.2697	1.1268
600	TL_high	transfer_short	1679	237	8	57632	477661	0.9874	1.4644	1.2101
600	TL_high	transfer_long	1193	281	4	28816	416621	0.9141	1.2878	1.1348
600	TL_medium	static	5180	14	1	7204	661988	0.4397	0.2969	0.5449
600	TL_medium	retrain_short	11325	9	8	57632	1154994	0.5151	0.4151	0.6443
600	TL_medium	retrain_long	5492	16	4	28816	706445	0.4461	0.3036	0.5510
600	TL_medium	fine_tune_short	4860	16	8	57632	690042	0.4356	0.2906	0.5390
600	TL_medium	fine_tune_long	4657	21	4	28816	646706	0.4336	0.2883	0.5370
600	TL_medium	transfer_short	5883	10	8	57632	763261	0.4465	0.3062	0.5534
600	TL_medium	transfer_long	4866	7	4	28816	660661	0.4336	0.2891	0.5376
600	TL_low	static	12764	0	1	7204	1206964	0.2650	0.1163	0.3410
600	TL_low	retrain_short	24745	0	8	57632	2120544	0.3761	0.2752	0.5246
600	TL_low	retrain_long	15429	1	4	28816	1420765	0.2830	0.1314	0.3624
600	TL_low	fine_tune_short	12512	0	8	57632	1239741	0.2661	0.1165	0.3414
600	TL_low	fine_tune_long	12081	0	4	28816	1179632	0.2615	0.1138	0.3374
600	TL_low	transfer_short	14686	0	8	57632	1396266	0.2791	0.1277	0.3574
600	TL_low	transfer_long	12486	0	4	28816	1208786	0.2637	0.1155	0.3399
1800	TL_high	static	1090	310	1	7204	388300	0.9860	1.5090	1.2284
1800	TL_high	retrain_short	1696	213	8	57632	475664	1.0459	1.7027	1.3049
1800	TL_high	retrain_long	1372	277	4	28816	428056	1.0422	1.6561	1.2869
1800	TL_high	fine_tune_short	1068	302	8	57632	436688	0.9939	1.5223	1.2338
1800	TL_high	fine_tune_long	1014	307	4	28816	404024	0.9683	1.4577	1.2073
1800	TL_high	transfer_short	1387	238	8	57632	455176	1.0357	1.6497	1.2844
1800	TL_high	transfer_long	1080	289	4	28816	407520	0.9746	1.4849	1.2185
1800	TL_medium	static	4466	13	1	7204	610396	0.5323	0.4648	0.6817
1800	TL_medium	retrain_short	8634	9	8	57632	961168	0.7525	1.0778	1.0382
1800	TL_medium	retrain_long	6657	14	4	28816	790048	0.6174	0.6181	0.7862
1800	TL_medium	fine_tune_short	4247	13	8	57632	645944	0.5329	0.4621	0.6798
1800	TL_medium	fine_tune_long	4051	22	4	28816	602944	0.5203	0.4457	0.6676
1800	TL_medium	transfer_short	4996	7	8	57632	699072	0.5607	0.5036	0.7096
1800	TL_medium	transfer_long	4542	17	4	28816	637960	0.5444	0.4808	0.6934
1800	TL_low	static	9572	0	1	7204	977148	0.3804	0.2657	0.5155
1800	TL_low	retrain_short	13048	0	8	57632	1278352	0.4897	0.4434	0.6659
1800	TL_low	retrain_long	11014	0	4	28816	1102784	0.4155	0.3105	0.5573
1800	TL_low	fine_tune_short	9895	0	8	57632	1051320	0.3876	0.2733	0.5228
1800	TL_low	fine_tune_long	9332	1	4	28816	981744	0.3783	0.2647	0.5145
1800	TL_low	transfer_short	10273	0	8	57632	1078552	0.3949	0.2802	0.5294
1800	TL_low	transfer_long	9784	0	4	28816	1014248	0.3874	0.2742	0.5236
3600	TL_high	static	1011	300	1	7204	377524	1.0730	1.8535	1.3614
3600	TL_high	retrain_short	3640	226	8	57632	613600	1.5943	4.6493	2.1562
3600	TL_high	retrain_long	1254	249	4	28816	413992	1.1108	1.9873	1.4097
3600	TL_high	fine_tune_short	938	310	8	57632	423792	1.0693	1.8261	1.3513
3600	TL_high	fine_tune_long	904	321	4	28816	392856	1.0332	1.7357	1.3175
3600	TL_high	transfer_short	1253	241	8	57632	442576	1.1253	2.0118	1.4184
3600	TL_high	transfer_long	1048	262	4	28816	399888	1.0698	1.8497	1.3600
3600	TL_medium	static	3836	8	1	7204	564588	0.6664	0.7613	0.8725
3600	TL_medium	retrain_short	6230	16	8	57632	788352	1.0092	1.9703	1.4037
3600	TL_medium	retrain_long	6558	10	4	28816	782528	1.2466	3.6977	1.9229
3600	TL_medium	fine_tune_short	3747	17	8	57632	609600	0.6637	0.7617	0.8728
3600	TL_medium	fine_tune_long	3491	25	4	28816	562528	0.6452	0.7288	0.8537
3600	TL_medium	transfer_short	4833	9	8	57632	687376	0.7486	0.9121	0.9550
3600	TL_medium	transfer_long	3686	14	4	28816	575952	0.6485	0.7338	0.8566
3600	TL_low	static	7467	0	1	7204	825572	0.5361	0.5456	0.7387
3600	TL_low	retrain_short	10642	0	8	57632	1105120	1.0985	2.7643	1.6626
3600	TL_low	retrain_long	9773	0	4	28816	1013432	1.1134	3.1535	1.7758
3600	TL_low	fine_tune_short	7941	0	8	57632	910648	0.5645	0.5917	0.7692
3600	TL_low	fine_tune_long	7608	0	4	28816	857536	0.5464	0.5705	0.7553
3600	TL_low	transfer_short	8518	0	8	57632	952176	0.6034	0.6566	0.8103
3600	TL_low	transfer_long	7258	0	4	28816	832352	0.5292	0.5366	0.7325

Table B.5: Detailed results for Dense model with linz_2010_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	930	338	1	35036	407468	0.8647	1.1595	1.0768
600	TL_high	retrain_short	958	337	8	280288	655306	0.8656	1.1624	1.0782
600	TL_high	retrain_long	949	331	4	140144	513906	0.8692	1.1715	1.0824
600	TL_high	fine_tune_short	949	334	8	280288	654354	0.8749	1.1808	1.0867
600	TL_high	fine_tune_long	948	335	4	140144	513994	0.8846	1.1994	1.0952
600	TL_high	transfer_short	999	308	8	280288	656133	0.8822	1.2007	1.0958
600	TL_high	transfer_long	1014	308	4	140144	516813	0.8800	1.1990	1.0950
600	TL_medium	static	4261	33	1	35036	624905	0.4245	0.2772	0.5265
600	TL_medium	retrain_short	4219	31	8	280288	867440	0.4231	0.2756	0.5250
600	TL_medium	retrain_long	4192	35	4	140144	725421	0.4208	0.2738	0.5233
600	TL_medium	fine_tune_short	4238	39	8	280288	869469	0.4233	0.2759	0.5253
600	TL_medium	fine_tune_long	4240	37	4	140144	729050	0.4230	0.2755	0.5249
600	TL_medium	transfer_short	4381	31	8	280288	879208	0.4291	0.2819	0.5310
600	TL_medium	transfer_long	4295	36	4	140144	732925	0.4250	0.2782	0.5274
600	TL_low	static	11110	4	1	35036	1115993	0.2555	0.1097	0.3313
600	TL_low	retrain_short	10928	4	8	280288	1348626	0.2539	0.1088	0.3299
600	TL_low	retrain_long	11004	3	4	140144	1213589	0.2551	0.1094	0.3308
600	TL_low	fine_tune_short	11088	3	8	280288	1360098	0.2554	0.1097	0.3312
600	TL_low	fine_tune_long	11096	3	4	140144	1220242	0.2554	0.1097	0.3313
600	TL_low	transfer_short	11822	3	8	280288	1412936	0.2619	0.1141	0.3378
600	TL_low	transfer_long	11251	3	4	140144	1231394	0.2573	0.1106	0.3326
1800	TL_high	static	852	346	1	35036	401380	0.9248	1.3502	1.1620
1800	TL_high	retrain_short	873	346	8	280288	648176	0.9186	1.3443	1.1594
1800	TL_high	retrain_long	886	326	4	140144	507768	0.9191	1.3436	1.1591
1800	TL_high	fine_tune_short	880	347	8	280288	649232	0.9229	1.3488	1.1614
1800	TL_high	fine_tune_long	867	345	4	140144	507704	0.9317	1.3624	1.1672
1800	TL_high	transfer_short	933	327	8	280288	651248	0.9441	1.3934	1.1804
1800	TL_high	transfer_long	896	333	4	140144	508520	0.9332	1.3680	1.1696
1800	TL_medium	static	3597	39	1	35036	577348	0.5074	0.4265	0.6530
1800	TL_medium	retrain_short	3528	44	8	280288	818448	0.5051	0.4227	0.6501
1800	TL_medium	retrain_long	3587	37	4	140144	681736	0.5043	0.4230	0.6504
1800	TL_medium	fine_tune_short	3615	40	8	280288	824432	0.5085	0.4281	0.6543
1800	TL_medium	fine_tune_long	3622	40	4	140144	684488	0.5053	0.4247	0.6517
1800	TL_medium	transfer_short	3780	34	8	280288	835856	0.5203	0.4436	0.6660
1800	TL_medium	transfer_long	3636	33	4	140144	685040	0.5084	0.4284	0.6545
1800	TL_low	static	8397	3	1	35036	920956	0.3579	0.2422	0.4921
1800	TL_low	retrain_short	8337	2	8	280288	1161920	0.3588	0.2442	0.4941
1800	TL_low	retrain_long	8371	6	4	140144	1024224	0.3593	0.2427	0.4927
1800	TL_low	fine_tune_short	8314	5	8	280288	1160880	0.3578	0.2419	0.4919
1800	TL_low	fine_tune_long	8364	5	4	140144	1024048	0.3581	0.2422	0.4922
1800	TL_low	transfer_short	9303	4	8	280288	1231624	0.3806	0.2659	0.5157
1800	TL_low	transfer_long	8671	5	4	140144	1045760	0.3655	0.2504	0.5004
3600	TL_high	static	748	350	1	35036	388996	0.9669	1.5535	1.2464
3600	TL_high	retrain_short	750	351	8	280288	634952	0.9682	1.5677	1.2521
3600	TL_high	retrain_long	768	350	4	140144	495760	0.9649	1.5561	1.2474
3600	TL_high	fine_tune_short	744	350	8	280288	634464	0.9638	1.5442	1.2427
3600	TL_high	fine_tune_long	744	350	4	140144	494032	0.9623	1.5371	1.2398
3600	TL_high	transfer_short	811	328	8	280288	638056	0.9963	1.6312	1.2772
3600	TL_high	transfer_long	808	331	4	140144	497576	0.9862	1.5978	1.2640
3600	TL_medium	static	3104	44	1	35036	541604	0.6193	0.6759	0.8221
3600	TL_medium	retrain_short	3067	49	8	280288	784960	0.6151	0.6716	0.8195
3600	TL_medium	retrain_long	3099	42	4	140144	646440	0.6100	0.6611	0.8131
3600	TL_medium	fine_tune_short	3126	43	8	280288	788872	0.6171	0.6705	0.8188
3600	TL_medium	fine_tune_long	3107	42	4	140144	647016	0.6180	0.6740	0.8210
3600	TL_medium	transfer_short	3523	38	8	280288	817192	0.6480	0.7314	0.8552
3600	TL_medium	transfer_long	3256	39	4	140144	657576	0.6263	0.6876	0.8292
3600	TL_low	static	6621	6	1	35036	792828	0.4976	0.4872	0.6980
3600	TL_low	retrain_short	6567	7	8	280288	1034736	0.4914	0.4817	0.6940
3600	TL_low	retrain_long	6415	4	4	140144	883192	0.4854	0.4741	0.6886
3600	TL_low	fine_tune_short	6577	5	8	280288	1035360	0.4955	0.4846	0.6961
3600	TL_low	fine_tune_long	6578	5	4	140144	895000	0.4961	0.4851	0.6965
3600	TL_low	transfer_short	7432	5	8	280288	1096920	0.5442	0.5659	0.7522
3600	TL_low	transfer_long	6977	2	4	140144	923560	0.5171	0.5177	0.7195

Table B.6: Detailed results for LSTM model with vienna_2010_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	1570	219	1	35092	445169	0.9611	1.3626	1.1673
600	TL_high	retrain_short	1174	287	8	280736	667704	0.9456	1.3328	1.1545
600	TL_high	retrain_long	1189	287	4	140368	528218	0.9221	1.2854	1.1338
600	TL_high	fine_tune_short	1555	208	8	280736	689421	0.9512	1.3512	1.1624
600	TL_high	fine_tune_long	1526	213	4	140368	547002	0.9550	1.3568	1.1648
600	TL_high	transfer_short	1538	222	8	280736	689314	0.9809	1.4080	1.1866
600	TL_high	transfer_long	1445	240	4	140368	543168	0.9604	1.3758	1.1729
600	TL_medium	static	7945	1	1	35092	887937	0.4820	0.3491	0.5908
600	TL_medium	retrain_short	5578	20	8	280736	965106	0.4515	0.3072	0.5543
600	TL_medium	retrain_long	5849	24	4	140368	844309	0.4559	0.3132	0.5597
600	TL_medium	fine_tune_short	7805	1	8	280736	1124040	0.4770	0.3432	0.5858
600	TL_medium	fine_tune_long	7635	1	4	140368	971144	0.4782	0.3437	0.5862
600	TL_medium	transfer_short	7779	5	8	280736	1122493	0.4824	0.3484	0.5902
600	TL_medium	transfer_long	7016	13	4	140368	927498	0.4753	0.3376	0.5810
600	TL_low	static	21922	0	1	35092	1894241	0.3261	0.1682	0.4102
600	TL_low	retrain_short	15008	0	8	280736	1642565	0.2841	0.1311	0.3621
600	TL_low	retrain_long	15173	0	4	140368	1513800	0.2854	0.1322	0.3635
600	TL_low	fine_tune_short	21517	0	8	280736	2111229	0.3231	0.1657	0.4070
600	TL_low	fine_tune_long	21153	0	4	140368	1944365	0.3212	0.1639	0.4049
600	TL_low	transfer_short	21894	0	8	280736	2138373	0.3289	0.1713	0.4139
600	TL_low	transfer_long	18616	0	4	140368	1761701	0.3082	0.1518	0.3896
1800	TL_high	static	1759	221	1	35092	458596	1.0908	1.8012	1.3421
1800	TL_high	retrain_short	1136	303	8	280736	665240	1.0090	1.5528	1.2461
1800	TL_high	retrain_long	1186	298	4	140368	527488	1.0356	1.6171	1.2717
1800	TL_high	fine_tune_short	1709	231	8	280736	702280	1.0819	1.7812	1.3346
1800	TL_high	fine_tune_long	1719	223	4	140368	560960	1.0867	1.7839	1.3356
1800	TL_high	transfer_short	2232	192	8	280736	736488	1.2028	2.1000	1.4491
1800	TL_high	transfer_long	1542	252	4	140368	549904	1.0907	1.7810	1.3345
1800	TL_medium	static	9384	1	1	35092	991564	0.7296	0.8348	0.9137
1800	TL_medium	retrain_short	5328	17	8	280736	947168	0.5845	0.5513	0.7425
1800	TL_medium	retrain_long	5878	19	4	140368	846256	0.6063	0.5871	0.7662
1800	TL_medium	fine_tune_short	8947	1	8	280736	1206632	0.7158	0.8091	0.8995
1800	TL_medium	fine_tune_long	8792	3	4	140368	1054568	0.7100	0.7942	0.8912
1800	TL_medium	transfer_short	8811	3	8	280736	1196584	0.7161	0.8081	0.8989
1800	TL_medium	transfer_long	7458	9	4	140368	958912	0.6664	0.7086	0.8418
1800	TL_low	static	18967	0	1	35092	1681476	0.6370	0.6619	0.8136
1800	TL_low	retrain_short	12074	0	8	280736	1431320	0.4458	0.3601	0.6001
1800	TL_low	retrain_long	12037	0	4	140368	1288008	0.4493	0.3672	0.6060
1800	TL_low	fine_tune_short	18305	0	8	280736	1879960	0.6206	0.6355	0.7972
1800	TL_low	fine_tune_long	17990	0	4	140368	1716624	0.6094	0.6168	0.7853
1800	TL_low	transfer_short	19024	0	8	280736	1931728	0.6597	0.7187	0.8478
1800	TL_low	transfer_long	16844	0	4	140368	1634112	0.5727	0.5535	0.7440
3600	TL_high	static	1935	210	1	35092	466660	1.2784	2.6137	1.6167
3600	TL_high	retrain_short	1069	311	8	280736	656080	1.0912	1.9216	1.3862
3600	TL_high	retrain_long	1136	301	4	140368	519704	1.0948	1.9387	1.3924
3600	TL_high	fine_tune_short	1892	227	8	280736	710664	1.2733	2.5981	1.6119
3600	TL_high	fine_tune_long	1833	221	4	140368	565424	1.2685	2.5702	1.6032
3600	TL_high	transfer_short	1960	215	8	280736	714888	1.3055	2.6853	1.6387
3600	TL_high	transfer_long	1675	228	4	140368	554440	1.2554	2.4638	1.5696
3600	TL_medium	static	8301	0	1	35092	913508	1.1457	2.2116	1.4872
3600	TL_medium	retrain_short	4841	23	8	280736	911824	0.7804	1.0909	1.0445
3600	TL_medium	retrain_long	5019	19	4	140368	783760	0.8053	1.1621	1.0780
3600	TL_medium	fine_tune_short	8233	0	8	280736	1154760	1.1354	2.1679	1.4724
3600	TL_medium	fine_tune_long	7901	1	4	140368	990256	1.0985	2.0442	1.4298
3600	TL_medium	transfer_short	8912	3	8	280736	1203816	1.2388	2.5389	1.5934
3600	TL_medium	transfer_long	6772	12	4	140368	909584	0.9783	1.6814	1.2967
3600	TL_low	static	12817	0	1	35092	1238660	1.1573	2.2288	1.4929
3600	TL_low	retrain_short	9315	1	8	280736	1232720	0.7157	0.9941	0.9970
3600	TL_low	retrain_long	9781	0	4	140368	1125560	0.7538	1.0773	1.0379
3600	TL_low	fine_tune_short	12469	0	8	280736	1459752	1.1177	2.1170	1.4550
3600	TL_low	fine_tune_long	12403	0	4	140368	1314344	1.0937	2.0386	1.4278
3600	TL_low	transfer_short	12607	0	8	280736	1469688	1.1832	2.3856	1.5446
3600	TL_low	transfer_long	11470	0	4	140368	1247168	0.9733	1.6926	1.3010

Table B.7: Detailed results for LSTM model with vienna_2019_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	5511	148	1	35092	724286	1.1793	1.9770	1.4061
600	TL_high	retrain_short	2546	238	8	280736	763090	1.0157	1.5325	1.2379
600	TL_high	retrain_long	2732	226	4	140368	634885	1.0225	1.5535	1.2464
600	TL_high	fine_tune_short	3806	157	8	280736	848024	1.1027	1.7541	1.3244
600	TL_high	fine_tune_long	3299	188	4	140368	673208	1.0725	1.6693	1.2920
600	TL_high	transfer_short	6560	130	8	280736	1044405	1.2234	2.1813	1.4769
600	TL_high	transfer_long	3526	183	4	140368	689213	1.1090	1.7563	1.3253
600	TL_medium	static	22197	2	1	35092	1914158	0.6589	0.6309	0.7943
600	TL_medium	retrain_short	10849	18	8	280736	1344482	0.5222	0.4159	0.6449
600	TL_medium	retrain_long	10921	13	4	140368	1208634	0.5229	0.4147	0.6440
600	TL_medium	fine_tune_short	17912	4	8	280736	1851968	0.6089	0.5467	0.7394
600	TL_medium	fine_tune_long	15321	3	4	140368	1524677	0.5771	0.4930	0.7021
600	TL_medium	transfer_short	33739	0	8	280736	2991194	0.8067	0.9538	0.9766
600	TL_medium	transfer_long	17888	0	4	140368	1709264	0.6143	0.5489	0.7409
600	TL_low	static	44423	0	1	35092	3514300	0.4817	0.3497	0.5913
600	TL_low	retrain_short	23402	1	8	280736	2247008	0.3374	0.1881	0.4337
600	TL_low	retrain_long	23835	0	4	140368	2137453	0.3402	0.1911	0.4372
600	TL_low	fine_tune_short	39981	0	8	280736	3440624	0.4396	0.2892	0.5378
600	TL_low	fine_tune_long	35580	0	4	140368	2983096	0.4124	0.2559	0.5059
600	TL_low	transfer_short	63269	0	8	280736	5117360	0.7198	0.7824	0.8845
600	TL_low	transfer_long	41345	0	4	140368	3398176	0.4492	0.2952	0.5433
1800	TL_high	static	7930	126	1	35092	896060	1.6561	4.0350	2.0087
1800	TL_high	retrain_short	3610	230	8	280736	838088	1.2423	2.4346	1.5603
1800	TL_high	retrain_long	3462	217	4	140368	686248	1.2475	2.4177	1.5549
1800	TL_high	fine_tune_short	6082	133	8	280736	1009744	1.4965	3.3431	1.8284
1800	TL_high	fine_tune_long	5068	162	4	140368	797672	1.4203	3.0130	1.7358
1800	TL_high	transfer_short	11482	103	8	280736	1404176	2.0831	6.5826	2.5657
1800	TL_high	transfer_long	5829	153	4	140368	852592	1.5040	3.3481	1.8298
1800	TL_medium	static	17836	2	1	35092	1600148	1.2318	2.4459	1.5639
1800	TL_medium	retrain_short	9032	9	8	280736	1212888	0.7731	1.0660	1.0325
1800	TL_medium	retrain_long	9958	9	4	140368	1138920	0.8101	1.1480	1.0715
1800	TL_medium	fine_tune_short	16571	1	8	280736	1755160	1.1092	1.9254	1.3876
1800	TL_medium	fine_tune_long	15486	2	4	140368	1536440	1.0243	1.6349	1.2786
1800	TL_medium	transfer_short	24099	0	8	280736	2297104	1.9491	5.9735	2.4441
1800	TL_medium	transfer_long	17389	0	4	140368	1673328	1.1233	1.9061	1.3806
1800	TL_low	static	24559	0	1	35092	2084084	1.1549	2.2906	1.5135
1800	TL_low	retrain_short	15680	0	8	280736	1690936	0.6574	0.9047	0.9511
1800	TL_low	retrain_long	16860	0	4	140368	1635248	0.6927	0.9661	0.9829
1800	TL_low	fine_tune_short	23923	0	8	280736	2284440	1.0266	1.7410	1.3195
1800	TL_low	fine_tune_long	23012	0	4	140368	2078192	0.9191	1.4016	1.1839
1800	TL_low	transfer_short	29605	0	8	280736	2693544	1.9570	6.0372	2.4571
1800	TL_low	transfer_long	24965	0	4	140368	2218808	1.0634	1.7889	1.3375
3600	TL_high	static	6915	117	1	35092	820140	2.3937	9.4143	3.0683
3600	TL_high	retrain_short	3002	224	8	280736	790512	1.5109	4.1716	2.0424
3600	TL_high	retrain_long	3168	219	4	140368	661528	1.5551	4.2837	2.0697
3600	TL_high	fine_tune_short	5872	122	8	280736	991472	2.0825	6.9737	2.6408
3600	TL_high	fine_tune_long	5031	138	4	140368	791160	1.8872	5.7080	2.3891
3600	TL_high	transfer_short	9754	75	8	280736	1268344	3.7074	23.0089	4.7968
3600	TL_high	transfer_long	5982	130	4	140368	859184	2.0862	6.8676	2.6206
3600	TL_medium	static	11708	2	1	35092	1158908	2.2352	8.8171	2.9694
3600	TL_medium	retrain_short	6904	13	8	280736	1059784	1.1922	3.1361	1.7709
3600	TL_medium	retrain_long	7098	13	4	140368	933096	1.2398	3.3321	1.8254
3600	TL_medium	fine_tune_short	11437	0	8	280736	1385432	1.9977	6.7787	2.6036
3600	TL_medium	fine_tune_long	10932	2	4	140368	1208528	1.7656	5.2907	2.3001
3600	TL_medium	transfer_short	14475	2	8	280736	1604280	3.8793	24.1355	4.9128
3600	TL_medium	transfer_long	11955	2	4	140368	1282184	2.0356	6.7026	2.5889
3600	TL_low	static	14515	0	1	35092	1360900	2.2208	8.7701	2.9614
3600	TL_low	retrain_short	11033	0	8	280736	1356344	1.1474	3.0468	1.7455
3600	TL_low	retrain_long	11157	0	4	140368	1224616	1.1908	3.2579	1.8050
3600	TL_low	fine_tune_short	14384	0	8	280736	1597616	1.9759	6.6754	2.5837
3600	TL_low	fine_tune_long	14033	0	4	140368	1431688	1.7422	5.2063	2.2817
3600	TL_low	transfer_short	16015	0	8	280736	1715048	3.9523	24.9487	4.9949
3600	TL_low	transfer_long	14841	0	4	140368	1489864	2.0424	6.6801	2.5846

Table B.8: Detailed results for LSTM model with vienna_201907_201912 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	1238	279	1	35092	425734	0.9275	1.3142	1.1464
600	TL_high	retrain_short	3257	230	8	280736	813834	1.0248	1.5837	1.2585
600	TL_high	retrain_long	1884	238	4	140368	574925	0.9941	1.4572	1.2071
600	TL_high	fine_tune_short	1030	311	8	280736	659421	0.9091	1.2608	1.1228
600	TL_high	fine_tune_long	1073	311	4	140368	521802	0.9038	1.2463	1.1164
600	TL_high	transfer_short	1043	300	8	280736	659461	0.8934	1.2285	1.1084
600	TL_high	transfer_long	1061	313	4	140368	521189	0.8816	1.1996	1.0953
600	TL_medium	static	5109	23	1	35092	685420	0.4408	0.2955	0.5436
600	TL_medium	retrain_short	13282	8	8	280736	1518917	0.5531	0.4821	0.6943
600	TL_medium	retrain_long	7679	13	4	140368	975197	0.4826	0.3514	0.5928
600	TL_medium	fine_tune_short	4588	36	8	280736	895040	0.4321	0.2861	0.5349
600	TL_medium	fine_tune_long	4682	37	4	140368	761170	0.4322	0.2865	0.5352
600	TL_medium	transfer_short	4810	24	8	280736	910152	0.4358	0.2901	0.5386
600	TL_medium	transfer_long	4690	22	4	140368	760672	0.4344	0.2877	0.5364
600	TL_low	static	13226	0	1	35092	1268105	0.2695	0.1195	0.3457
600	TL_low	retrain_short	31150	0	8	280736	2804808	0.4190	0.3260	0.5710
600	TL_low	retrain_long	21206	0	4	140368	1948184	0.3243	0.1756	0.4191
600	TL_low	fine_tune_short	11962	0	8	280736	1423245	0.2626	0.1145	0.3384
600	TL_low	fine_tune_long	12013	0	4	140368	1286264	0.2619	0.1139	0.3376
600	TL_low	transfer_short	13097	0	8	280736	1504984	0.2698	0.1205	0.3471
600	TL_low	transfer_long	12359	1	4	140368	1311250	0.2646	0.1158	0.3403
1800	TL_high	static	1102	299	1	35092	415916	0.9859	1.5118	1.2296
1800	TL_high	retrain_short	4303	219	8	280736	887672	1.3383	2.8858	1.6988
1800	TL_high	retrain_long	4324	234	4	140368	749192	1.3105	2.9068	1.7049
1800	TL_high	fine_tune_short	918	354	8	280736	652696	0.9568	1.4304	1.1960
1800	TL_high	fine_tune_long	958	327	4	140368	513368	0.9448	1.4039	1.1848
1800	TL_high	transfer_short	956	325	8	280736	653440	0.9533	1.4301	1.1958
1800	TL_high	transfer_long	973	317	4	140368	513392	0.9608	1.4398	1.1999
1800	TL_medium	static	4478	23	1	35092	639788	0.5433	0.4784	0.6916
1800	TL_medium	retrain_short	10739	10	8	280736	1335904	0.9211	1.7554	1.3249
1800	TL_medium	retrain_long	10414	13	4	140368	1172024	0.8550	1.3522	1.1628
1800	TL_medium	fine_tune_short	3938	31	8	280736	848416	0.5217	0.4477	0.6691
1800	TL_medium	fine_tune_long	3986	35	4	140368	710712	0.5219	0.4493	0.6703
1800	TL_medium	transfer_short	4643	32	8	280736	898472	0.5529	0.4990	0.7064
1800	TL_medium	transfer_long	4078	34	4	140368	717664	0.5286	0.4577	0.6765
1800	TL_low	static	10397	1	1	35092	1064492	0.3970	0.2844	0.5333
1800	TL_low	retrain_short	17868	0	8	280736	1848504	0.8139	1.5561	1.2474
1800	TL_low	retrain_long	16768	0	4	140368	1628648	0.7432	1.3440	1.1593
1800	TL_low	fine_tune_short	9249	0	8	280736	1227912	0.3769	0.2623	0.5121
1800	TL_low	fine_tune_long	9298	0	4	140368	1090784	0.3760	0.2616	0.5114
1800	TL_low	transfer_short	10885	0	8	280736	1345720	0.4161	0.3131	0.5595
1800	TL_low	transfer_long	9703	2	4	140368	1120088	0.3866	0.2739	0.5233
3600	TL_high	static	994	294	1	35092	403868	1.0754	1.8545	1.3618
3600	TL_high	retrain_short	3633	228	8	280736	836328	1.7349	6.3805	2.5260
3600	TL_high	retrain_long	2831	244	4	140368	638808	1.5269	4.5662	2.1369
3600	TL_high	fine_tune_short	841	328	8	280736	640872	1.0115	1.6798	1.2961
3600	TL_high	fine_tune_long	873	315	4	140368	501808	0.9993	1.6624	1.2893
3600	TL_high	transfer_short	870	327	8	280736	642920	1.0225	1.7114	1.3082
3600	TL_high	transfer_long	847	322	4	140368	500328	1.0120	1.6809	1.2965
3600	TL_medium	static	4007	28	1	35092	605908	0.6884	0.8016	0.8953
3600	TL_medium	retrain_short	8512	10	8	280736	1175424	1.6306	6.2613	2.5023
3600	TL_medium	retrain_long	7648	10	4	140368	972560	1.3210	3.7945	1.9479
3600	TL_medium	fine_tune_short	3526	35	8	280736	817816	0.6542	0.7416	0.8612
3600	TL_medium	fine_tune_long	3545	40	4	140368	678808	0.6469	0.7301	0.8545
3600	TL_medium	transfer_short	4203	32	8	280736	866392	0.7094	0.8872	0.9419
3600	TL_medium	transfer_long	3644	36	4	140368	685712	0.6598	0.7580	0.8707
3600	TL_low	static	8014	0	1	35092	892844	0.5829	0.6377	0.7986
3600	TL_low	retrain_short	12145	0	8	280736	1436440	1.5689	5.7883	2.4059
3600	TL_low	retrain_long	11173	0	4	140368	1225800	1.2193	3.5948	1.8960
3600	TL_low	fine_tune_short	7396	2	8	280736	1094608	0.5417	0.5667	0.7528
3600	TL_low	fine_tune_long	7230	5	4	140368	942168	0.5331	0.5538	0.7441
3600	TL_low	transfer_short	8497	1	8	280736	1173840	0.6664	0.8439	0.9187
3600	TL_low	transfer_long	8024	2	4	140368	999168	0.5807	0.6313	0.7945

Table B.9: Detailed results for LSTM model with linz_2010_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	954	342	1	119736	494576	0.8807	1.1911	1.0914
600	TL_high	retrain_short	886	365	8	957888	1329714	0.8628	1.1514	1.0730
600	TL_high	retrain_long	952	329	4	478944	852680	0.8788	1.1866	1.0893
600	TL_high	fine_tune_short	942	342	8	957888	1332445	0.8792	1.1844	1.0883
600	TL_high	fine_tune_long	955	337	4	478944	853853	0.8712	1.1756	1.0843
600	TL_high	transfer_short	950	341	8	957888	1332992	0.8822	1.1945	1.0929
600	TL_high	transfer_long	989	327	4	478944	855904	0.8740	1.1808	1.0866
600	TL_medium	static	4394	30	1	119736	719093	0.4271	0.2798	0.5290
600	TL_medium	retrain_short	4291	26	8	957888	1549954	0.4234	0.2759	0.5253
600	TL_medium	retrain_long	4378	25	4	478944	1076952	0.4238	0.2771	0.5264
600	TL_medium	fine_tune_short	4460	25	8	957888	1562154	0.4245	0.2779	0.5272
600	TL_medium	fine_tune_long	4397	32	4	478944	1078930	0.4252	0.2785	0.5278
600	TL_medium	transfer_short	4503	30	8	957888	1565658	0.4263	0.2793	0.5285
600	TL_medium	transfer_long	4354	33	4	478944	1075920	0.4252	0.2786	0.5278
600	TL_low	static	11693	1	1	119736	1242458	0.2586	0.1123	0.3351
600	TL_low	retrain_short	11553	0	8	957888	2070914	0.2567	0.1111	0.3334
600	TL_low	retrain_long	11540	0	4	478944	1590749	0.2577	0.1116	0.3340
600	TL_low	fine_tune_short	11587	0	8	957888	2073410	0.2576	0.1116	0.3340
600	TL_low	fine_tune_long	11634	0	4	478944	1597570	0.2579	0.1118	0.3344
600	TL_low	transfer_short	11918	1	8	957888	2097317	0.2604	0.1135	0.3369
600	TL_low	transfer_long	11606	0	4	478944	1595546	0.2580	0.1119	0.3344
1800	TL_high	static	844	362	1	119736	486568	0.9309	1.3582	1.1654
1800	TL_high	retrain_short	846	352	8	957888	1324240	0.9317	1.3571	1.1649
1800	TL_high	retrain_long	870	351	4	478944	846688	0.9260	1.3563	1.1646
1800	TL_high	fine_tune_short	851	354	8	957888	1325512	0.9349	1.3663	1.1689
1800	TL_high	fine_tune_long	877	349	4	478944	847424	0.9380	1.3747	1.1725
1800	TL_high	transfer_short	863	350	8	957888	1326136	0.9405	1.3817	1.1755
1800	TL_high	transfer_long	910	330	4	478944	848432	0.9356	1.3704	1.1706
1800	TL_medium	static	3774	28	1	119736	674120	0.5153	0.4378	0.6617
1800	TL_medium	retrain_short	3813	40	8	957888	1516336	0.5125	0.4357	0.6600
1800	TL_medium	retrain_long	3764	43	4	478944	1033776	0.5130	0.4349	0.6595
1800	TL_medium	fine_tune_short	3745	36	8	957888	1511272	0.5155	0.4366	0.6608
1800	TL_medium	fine_tune_long	3739	40	4	478944	1031832	0.5087	0.4297	0.6555
1800	TL_medium	transfer_short	3847	30	8	957888	1518200	0.5149	0.4377	0.6616
1800	TL_medium	transfer_long	3899	19	4	478944	1041960	0.5198	0.4437	0.6661
1800	TL_low	static	8987	0	1	119736	1047544	0.3726	0.2609	0.5107
1800	TL_low	retrain_short	8898	0	8	957888	1879768	0.3690	0.2572	0.5072
1800	TL_low	retrain_long	8791	1	4	478944	1393288	0.3683	0.2546	0.5045
1800	TL_low	fine_tune_short	8885	1	8	957888	1878928	0.3696	0.2562	0.5061
1800	TL_low	fine_tune_long	8789	0	4	478944	1392712	0.3684	0.2557	0.5057
1800	TL_low	transfer_short	9237	0	8	957888	1904200	0.3764	0.2636	0.5135
1800	TL_low	transfer_long	9098	0	4	478944	1414960	0.3731	0.2600	0.5099
3600	TL_high	static	778	336	1	119736	475296	0.9679	1.5733	1.2543
3600	TL_high	retrain_short	746	350	8	957888	1312144	0.9679	1.5550	1.2470
3600	TL_high	retrain_long	769	344	4	478944	834248	0.9759	1.5875	1.2600
3600	TL_high	fine_tune_short	778	345	8	957888	1314440	0.9808	1.5893	1.2607
3600	TL_high	fine_tune_long	774	336	4	478944	834432	0.9778	1.5884	1.2603
3600	TL_high	transfer_short	799	329	8	957888	1315072	0.9958	1.6312	1.2772
3600	TL_high	transfer_long	779	332	4	478944	834568	0.9887	1.6047	1.2668
3600	TL_medium	static	3246	41	1	119736	636488	0.6266	0.6953	0.8339
3600	TL_medium	retrain_short	3235	55	8	957888	1475072	0.6246	0.7002	0.8368
3600	TL_medium	retrain_long	3263	41	4	478944	997056	0.6222	0.6891	0.8301
3600	TL_medium	fine_tune_short	3250	42	8	957888	1475488	0.6262	0.6916	0.8316
3600	TL_medium	fine_tune_long	3261	49	4	478944	997440	0.6223	0.6861	0.8283
3600	TL_medium	transfer_short	3416	33	8	957888	1486936	0.6334	0.7069	0.8408
3600	TL_medium	transfer_long	3337	30	4	478944	1001832	0.6322	0.6989	0.8360
3600	TL_low	static	7248	1	1	119736	922392	0.5349	0.5526	0.7433
3600	TL_low	retrain_short	7185	3	8	957888	1756576	0.5282	0.5401	0.7349
3600	TL_low	retrain_long	7257	2	4	478944	1282472	0.5366	0.5596	0.7481
3600	TL_low	fine_tune_short	7122	2	8	957888	1752032	0.5255	0.5375	0.7332
3600	TL_low	fine_tune_long	7106	1	4	478944	1271592	0.5238	0.5338	0.7306
3600	TL_low	transfer_short	7459	1	8	957888	1776240	0.5447	0.5651	0.7517
3600	TL_low	transfer_long	7766	0	4	478944	1319056	0.5601	0.5888	0.7673

Table B.10: Detailed results for ConvLSTM model with vienna_2010_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	1908	165	1	119920	550533	0.9607	1.3835	1.1762
600	TL_high	retrain_short	1223	262	8	959360	1347978	0.9172	1.2675	1.1258
600	TL_high	retrain_long	1275	243	4	479680	870453	0.9087	1.2567	1.1210
600	TL_high	fine_tune_short	1977	158	8	959360	1394986	0.9578	1.3761	1.1731
600	TL_high	fine_tune_long	1739	185	4	479680	899808	0.9432	1.3414	1.1582
600	TL_high	transfer_short	2225	133	8	959360	1410989	0.9835	1.4437	1.2015
600	TL_high	transfer_long	1680	196	4	479680	896402	0.9461	1.3496	1.1617
600	TL_medium	static	8875	3	1	119920	1039912	0.4823	0.3534	0.5944
600	TL_medium	retrain_short	5804	14	8	959360	1659520	0.4438	0.3020	0.5496
600	TL_medium	retrain_long	5834	7	4	479680	1181194	0.4468	0.3041	0.5514
600	TL_medium	fine_tune_short	9219	0	8	959360	1904389	0.4875	0.3593	0.5994
600	TL_medium	fine_tune_long	8227	4	4	479680	1353285	0.4752	0.3427	0.5854
600	TL_medium	transfer_short	9951	0	8	959360	1957104	0.4926	0.3685	0.6070
600	TL_medium	transfer_long	7771	6	4	479680	1320602	0.4704	0.3361	0.5797
600	TL_low	static	21222	0	1	119920	1928674	0.3178	0.1599	0.3999
600	TL_low	retrain_short	15270	0	8	959360	2340042	0.2814	0.1294	0.3597
600	TL_low	retrain_long	15552	0	4	479680	1880354	0.2827	0.1307	0.3616
600	TL_low	fine_tune_short	22344	0	8	959360	2849402	0.3261	0.1670	0.4087
600	TL_low	fine_tune_long	20067	0	4	479680	2205493	0.3113	0.1541	0.3925
600	TL_low	transfer_short	23597	0	8	959360	2939621	0.3340	0.1734	0.4164
600	TL_low	transfer_long	18788	0	4	479680	2113400	0.3032	0.1475	0.3841
1800	TL_high	static	1958	156	1	119920	552272	1.0804	1.7995	1.3414
1800	TL_high	retrain_short	1158	270	8	959360	1342696	0.9726	1.4735	1.2139
1800	TL_high	retrain_long	1174	274	4	479680	863736	0.9933	1.5130	1.2300
1800	TL_high	fine_tune_short	2040	161	8	959360	1398448	1.0991	1.8528	1.3612
1800	TL_high	fine_tune_long	1800	167	4	479680	901616	1.0707	1.7551	1.3248
1800	TL_high	transfer_short	2226	123	8	959360	1409280	1.1208	1.9222	1.3864
1800	TL_high	transfer_long	1664	196	4	479680	893872	1.0522	1.7019	1.3046
1800	TL_medium	static	8621	2	1	119920	1021544	0.6948	0.7654	0.8749
1800	TL_medium	retrain_short	5192	12	8	959360	1615208	0.5697	0.5262	0.7254
1800	TL_medium	retrain_long	5540	8	4	479680	1160032	0.5785	0.5479	0.7402
1800	TL_medium	fine_tune_short	9319	2	8	959360	1911736	0.7219	0.8165	0.9036
1800	TL_medium	fine_tune_long	8064	2	4	479680	1341416	0.6768	0.7270	0.8526
1800	TL_medium	transfer_short	9583	0	8	959360	1930608	0.7315	0.8453	0.9194
1800	TL_medium	transfer_long	7479	4	4	479680	1299416	0.6565	0.6898	0.8306
1800	TL_low	static	17116	0	1	119920	1633040	0.5733	0.5384	0.7337
1800	TL_low	retrain_short	12403	0	8	959360	2133616	0.4425	0.3416	0.5845
1800	TL_low	retrain_long	12907	0	4	479680	1689936	0.4571	0.3625	0.6021
1800	TL_low	fine_tune_short	18168	0	8	959360	2548728	0.6050	0.5899	0.7680
1800	TL_low	fine_tune_long	16479	0	4	479680	1947152	0.5521	0.5033	0.7094
1800	TL_low	transfer_short	18484	0	8	959360	2571480	0.6131	0.6046	0.7776
1800	TL_low	transfer_long	15922	0	4	479680	1907048	0.5361	0.4789	0.6920
3600	TL_high	static	1987	155	1	119920	552424	1.2792	2.6428	1.6257
3600	TL_high	retrain_short	1102	280	8	959360	1335328	1.0819	1.9051	1.3802
3600	TL_high	retrain_long	1082	289	4	479680	854408	1.0748	1.8913	1.3752
3600	TL_high	fine_tune_short	2151	152	8	959360	1404008	1.3190	2.7538	1.6595
3600	TL_high	fine_tune_long	1809	164	4	479680	900088	1.2505	2.5128	1.5852
3600	TL_high	transfer_short	2359	130	8	959360	1417752	1.3592	2.9031	1.7038
3600	TL_high	transfer_long	1641	195	4	479680	889728	1.2108	2.3728	1.5404
3600	TL_medium	static	7586	2	1	119920	946984	1.0443	1.8167	1.3478
3600	TL_medium	retrain_short	5132	9	8	959360	1610584	0.7821	1.0548	1.0270
3600	TL_medium	retrain_long	5376	10	4	479680	1148240	0.8006	1.1039	1.0506
3600	TL_medium	fine_tune_short	8095	3	8	959360	1823632	1.1030	1.9919	1.4114
3600	TL_medium	fine_tune_long	7108	3	4	479680	1272600	0.9909	1.6487	1.2840
3600	TL_medium	transfer_short	8392	1	8	959360	1844904	1.1320	2.0912	1.4461
3600	TL_medium	transfer_long	6729	5	4	479680	1245424	0.9465	1.5158	1.2312
3600	TL_low	static	11989	0	1	119920	1263888	0.9950	1.6669	1.2911
3600	TL_low	retrain_short	9596	0	8	959360	1931488	0.7135	0.9186	0.9584
3600	TL_low	retrain_long	10206	0	4	479680	1495456	0.7588	1.0243	1.0121
3600	TL_low	fine_tune_short	12366	0	8	959360	2130976	1.0678	1.8773	1.3701
3600	TL_low	fine_tune_long	11696	0	4	479680	1602768	0.9497	1.5279	1.2361
3600	TL_low	transfer_short	12343	0	8	959360	2129320	1.0543	1.8363	1.3551
3600	TL_low	transfer_long	11465	0	4	479680	1586136	0.9048	1.3955	1.1813

Table B.11: Detailed results for ConvLSTM model with vienna_2019_2019 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	4871	92	1	119920	758413	1.0787	1.7406	1.3193
600	TL_high	retrain_short	2049	194	8	959360	1402344	0.9474	1.3621	1.1671
600	TL_high	retrain_long	2683	195	4	479680	968218	0.9792	1.4487	1.2036
600	TL_high	fine_tune_short	3506	82	8	959360	1499397	1.0478	1.6288	1.2762
600	TL_high	fine_tune_long	3672	101	4	479680	1032746	1.0409	1.6132	1.2701
600	TL_high	transfer_short	3803	71	8	959360	1519869	1.0285	1.5909	1.2613
600	TL_high	transfer_long	3608	108	4	479680	1028658	1.0316	1.5915	1.2616
600	TL_medium	static	23625	0	1	119920	2101693	0.6538	0.6459	0.8037
600	TL_medium	retrain_short	11307	11	8	959360	2055400	0.5108	0.4059	0.6371
600	TL_medium	retrain_long	12042	7	4	479680	1628048	0.5183	0.4183	0.6468
600	TL_medium	fine_tune_short	16830	0	8	959360	2452394	0.5801	0.5052	0.7107
600	TL_medium	fine_tune_long	17140	0	4	479680	1994749	0.5821	0.5128	0.7161
600	TL_medium	transfer_short	16033	0	8	959360	2395010	0.5629	0.4826	0.6947
600	TL_medium	transfer_long	15783	2	4	479680	1897200	0.5653	0.4870	0.6978
600	TL_low	static	45743	0	1	119920	3694189	0.5116	0.4128	0.6425
600	TL_low	retrain_short	25350	0	8	959360	3065805	0.3503	0.2076	0.4556
600	TL_low	retrain_long	26882	0	4	479680	2696144	0.3624	0.2187	0.4676
600	TL_low	fine_tune_short	37553	0	8	959360	3944453	0.4293	0.2828	0.5318
600	TL_low	fine_tune_long	37277	0	4	479680	3444613	0.4294	0.2865	0.5353
600	TL_low	transfer_short	33736	0	8	959360	3669629	0.4034	0.2549	0.5049
600	TL_low	transfer_long	36079	0	4	479680	3358357	0.4192	0.2713	0.5209
1800	TL_high	static	7923	86	1	119920	977488	1.6103	4.0683	2.0170
1800	TL_high	retrain_short	3169	214	8	959360	1483288	1.1696	2.2089	1.4862
1800	TL_high	retrain_long	3539	208	4	479680	1029392	1.2045	2.3362	1.5285
1800	TL_high	fine_tune_short	5195	81	8	959360	1620224	1.3991	2.9994	1.7319
1800	TL_high	fine_tune_long	5461	99	4	479680	1160608	1.4070	3.0658	1.7509
1800	TL_high	transfer_short	4408	89	8	959360	1564392	1.2993	2.6556	1.6296
1800	TL_high	transfer_long	4466	143	4	479680	1092480	1.2998	2.6801	1.6371
1800	TL_medium	static	18162	0	1	119920	1708352	1.3441	3.0687	1.7518
1800	TL_medium	retrain_short	10135	11	8	959360	1970920	0.8262	1.2604	1.1227
1800	TL_medium	retrain_long	10334	4	4	479680	1504816	0.8384	1.3040	1.1419
1800	TL_medium	fine_tune_short	15973	0	8	959360	2390688	1.0692	1.8188	1.3486
1800	TL_medium	fine_tune_long	15616	0	4	479680	1885016	1.0771	1.8971	1.3774
1800	TL_medium	transfer_short	14818	0	8	959360	2307528	1.0029	1.6327	1.2778
1800	TL_medium	transfer_long	14704	0	4	479680	1819352	1.0140	1.6893	1.2997
1800	TL_low	static	24662	0	1	119920	2176352	1.2687	2.9019	1.7035
1800	TL_low	retrain_short	17056	0	8	959360	2468632	0.7010	1.0245	1.0122
1800	TL_low	retrain_long	18069	0	4	479680	2061600	0.7474	1.1384	1.0670
1800	TL_low	fine_tune_short	23874	0	8	959360	2959560	0.9772	1.6150	1.2708
1800	TL_low	fine_tune_long	23077	0	4	479680	2422208	0.9854	1.6970	1.3027
1800	TL_low	transfer_short	22648	0	8	959360	2871288	0.9058	1.4266	1.1944
1800	TL_low	transfer_long	23585	0	4	479680	2458784	0.9262	1.4420	1.2008
3600	TL_high	static	6775	82	1	119920	893072	2.5436	11.5895	3.4043
3600	TL_high	retrain_short	3012	212	8	959360	1469024	1.5284	4.5721	2.1382
3600	TL_high	retrain_long	3170	186	4	479680	999008	1.5674	4.7507	2.1796
3600	TL_high	fine_tune_short	5106	81	8	959360	1612792	1.9673	6.5403	2.5574
3600	TL_high	fine_tune_long	5242	99	4	479680	1143624	1.9992	6.9262	2.6318
3600	TL_high	transfer_short	4346	92	8	959360	1558688	1.8033	5.7373	2.3953
3600	TL_high	transfer_long	4700	96	4	479680	1104432	1.8539	5.8986	2.4287
3600	TL_medium	static	11905	0	1	119920	1257840	2.4810	11.3274	3.3656
3600	TL_medium	retrain_short	7652	7	8	959360	1791784	1.3635	4.1891	2.0467
3600	TL_medium	retrain_long	7466	6	4	479680	1298368	1.3336	4.0269	2.0067
3600	TL_medium	fine_tune_short	11484	0	8	959360	2067472	1.8873	6.1404	2.4780
3600	TL_medium	fine_tune_long	11050	0	4	479680	1556256	1.9063	6.4913	2.5478
3600	TL_medium	transfer_short	11373	0	8	959360	2059480	1.8140	5.6216	2.3710
3600	TL_medium	transfer_long	10730	0	4	479680	1533216	1.7814	5.6916	2.3857
3600	TL_low	static	14504	0	1	119920	1444968	2.4405	11.1737	3.3427
3600	TL_low	retrain_short	11316	0	8	959360	2055328	1.2638	3.8251	1.9558
3600	TL_low	retrain_long	11636	0	4	479680	1598416	1.2976	3.9170	1.9791
3600	TL_low	fine_tune_short	14422	0	8	959360	2279008	1.8600	6.0539	2.4605
3600	TL_low	fine_tune_long	14073	0	4	479680	1773912	1.8673	6.3475	2.5194
3600	TL_low	transfer_short	14363	0	8	959360	2274760	1.7751	5.4727	2.3394
3600	TL_low	transfer_long	14611	0	4	479680	1812648	1.8182	5.5760	2.3614

Table B.12: Detailed results for ConvLSTM model with vienna_201907_201912 dataset

ρ [s]	$\mathcal{T}$	strategy	n_v	n_u	n_d	$data_d$ [B]	$\mathcal{D}$ [B]	MAE [°C]	MSE [°C^2]	RMSE [°C]
600	TL_high	static	1213	280	1	119920	508962	0.9152	1.2812	1.1319
600	TL_high	retrain_short	3046	175	8	959360	1472914	1.0075	1.5287	1.2364
600	TL_high	retrain_long	3751	190	4	479680	1045069	1.0265	1.6014	1.2655
600	TL_high	fine_tune_short	1075	311	8	959360	1341157	0.8903	1.2184	1.1038
600	TL_high	fine_tune_long	1092	320	4	479680	863197	0.9038	1.2489	1.1175
600	TL_high	transfer_short	1081	314	8	959360	1341957	0.8850	1.2016	1.0962
600	TL_high	transfer_long	1072	312	4	479680	861120	0.8933	1.2238	1.1062
600	TL_medium	static	5571	16	1	119920	803016	0.4402	0.2976	0.5455
600	TL_medium	retrain_short	14379	3	8	959360	2276133	0.5545	0.4947	0.7033
600	TL_medium	retrain_long	14014	2	4	479680	1769818	0.5453	0.4685	0.6845
600	TL_medium	fine_tune_short	5200	20	8	959360	1616530	0.4322	0.2889	0.5375
600	TL_medium	fine_tune_long	5449	21	4	479680	1154941	0.4418	0.2982	0.5461
600	TL_medium	transfer_short	5232	11	8	959360	1618130	0.4334	0.2896	0.5381
600	TL_medium	transfer_long	5381	21	4	479680	1149669	0.4358	0.2933	0.5416
600	TL_low	static	13971	1	1	119920	1406672	0.2725	0.1230	0.3507
600	TL_low	retrain_short	31801	0	8	959360	3530304	0.4370	0.3738	0.6114
600	TL_low	retrain_long	25577	0	4	479680	2602210	0.3660	0.2463	0.4963
600	TL_low	fine_tune_short	14179	0	8	959360	2261520	0.2760	0.1257	0.3545
600	TL_low	fine_tune_long	14598	1	4	479680	1811792	0.2800	0.1287	0.3587
600	TL_low	transfer_short	13613	0	8	959360	2220757	0.2699	0.1212	0.3481
600	TL_low	transfer_long	13473	0	4	479680	1730709	0.2703	0.1216	0.3488
1800	TL_high	static	1124	301	1	119920	502336	0.9788	1.4964	1.2233
1800	TL_high	retrain_short	5095	149	8	959360	1617720	1.3834	3.3335	1.8258
1800	TL_high	retrain_long	3009	182	4	479680	989632	1.1689	2.2094	1.4864
1800	TL_high	fine_tune_short	1003	316	8	959360	1334824	0.9534	1.4237	1.1932
1800	TL_high	fine_tune_long	995	331	4	479680	855304	0.9610	1.4362	1.1984
1800	TL_high	transfer_short	958	336	8	959360	1332872	0.9460	1.4046	1.1852
1800	TL_high	transfer_long	1006	322	4	479680	855504	0.9440	1.4080	1.1866
1800	TL_medium	static	4917	17	1	119920	755848	0.5502	0.4980	0.7057
1800	TL_medium	retrain_short	12686	5	8	959360	2154344	0.9842	1.8810	1.3715
1800	TL_medium	retrain_long	9817	4	4	479680	1467752	0.8134	1.2790	1.1309
1800	TL_medium	fine_tune_short	4731	21	8	959360	1582648	0.5490	0.4994	0.7067
1800	TL_medium	fine_tune_long	5034	27	4	479680	1124976	0.5626	0.5187	0.7202
1800	TL_medium	transfer_short	4468	17	8	959360	1563472	0.5390	0.4816	0.6940
1800	TL_medium	transfer_long	4695	19	4	479680	1099992	0.5439	0.4912	0.7009
1800	TL_low	static	11059	0	1	119920	1196928	0.4163	0.3136	0.5600
1800	TL_low	retrain_short	19939	0	8	959360	2676240	0.9798	2.1861	1.4785
1800	TL_low	retrain_long	19297	0	4	479680	2150048	0.9397	2.1267	1.4583
1800	TL_low	fine_tune_short	11570	0	8	959360	2073664	0.4315	0.3371	0.5806
1800	TL_low	fine_tune_long	12345	2	4	479680	1649624	0.4504	0.3604	0.6003
1800	TL_low	transfer_short	11080	0	8	959360	2038384	0.4167	0.3125	0.5590
1800	TL_low	transfer_long	11262	1	4	479680	1571592	0.4213	0.3209	0.5665
3600	TL_high	static	1047	287	1	119920	492120	1.0624	1.8416	1.3571
3600	TL_high	retrain_short	4855	152	8	959360	1598664	2.1402	10.1559	3.1868
3600	TL_high	retrain_long	2878	189	4	479680	978440	1.4978	4.2497	2.0615
3600	TL_high	fine_tune_short	918	312	8	959360	1324192	1.0314	1.7390	1.3187
3600	TL_high	fine_tune_long	943	309	4	479680	845856	1.0223	1.7149	1.3096
3600	TL_high	transfer_short	924	310	8	959360	1324496	1.0151	1.6970	1.3027
3600	TL_high	transfer_long	969	294	4	479680	846872	1.0298	1.7482	1.3222
3600	TL_medium	static	4388	18	1	119920	717608	0.7199	0.9019	0.9497
3600	TL_medium	retrain_short	9556	8	8	959360	1929088	2.0582	10.0124	3.1642
3600	TL_medium	retrain_long	8019	1	4	479680	1338080	1.3696	4.4394	2.1070
3600	TL_medium	fine_tune_short	4405	26	8	959360	1559240	0.7277	0.9395	0.9693
3600	TL_medium	fine_tune_long	4871	27	4	479680	1112880	0.7671	1.0307	1.0153
3600	TL_medium	transfer_short	4069	10	8	959360	1534152	0.6981	0.8560	0.9252
3600	TL_medium	transfer_long	4320	25	4	479680	1073096	0.7161	0.9131	0.9556
3600	TL_low	static	8648	0	1	119920	1023336	0.6418	0.7672	0.8759
3600	TL_low	retrain_short	12990	0	8	959360	2175904	1.9824	8.8711	2.9784
3600	TL_low	retrain_long	12036	0	4	479680	1627248	1.3197	4.2522	2.0621
3600	TL_low	fine_tune_short	9151	0	8	959360	1899496	0.6949	0.8934	0.9452
3600	TL_low	fine_tune_long	9498	1	4	479680	1444568	0.7282	0.9691	0.9844
3600	TL_low	transfer_short	8936	0	8	959360	1884016	0.6458	0.7586	0.8710
3600	TL_low	transfer_long	8594	0	4	479680	1379424	0.6537	0.8158	0.9032

Table B.13: Detailed results for ConvLSTM model with linz_2010_2019 dataset

List of Figures

List of Tables

Bibliography

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283. USENIX Association, November 2016.

[ACFP09] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, May 2009.

[AEKB20] Atakan Aral, Melike Erol-Kantarci, and Ivona Brandić. Staleness control for edge data analytics. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(2):1–24, June 2020.

[AOAS21] Faris A. Almalki, Soufiene Ben Othman, Fahad A. Almalki, and Hedi Sakli. EERP-DPM: Energy efficient routing protocol using dual prediction model for healthcare using IoT. *Journal of Healthcare Engineering*, 2021:1–15, May 2021.

[BBBK16] Boris Bellalta, Luciano Bononi, Raffaele Bruno, and Andreas Kassler. Next generation IEEE 802.11 wireless local area networks: Current status, future directions and open challenges. *Computer Communications*, 75:1–25, February 2016.

[BBK+19] Anirban Bhattacharjee, Yogesh Barve, Shweta Khare, Shunxing Bao, Zhuangwei Kang, Aniruddha Gokhale, and Thomas Damiano. STRATUM: A BigData-as-a-service for lifecycle management of IoT analytics applications. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, December 2019.

[BCS+20] Anirban Bhattacharjee, Ajay Dev Chhokra, Hongyang Sun, Shashank Shekhar, Aniruddha Gokhale, Gabor Karsai, and Abhishek Dubey. DeepEdge: An efficient framework for deep learning model update on heterogeneous edge. In *IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, May 2020.

[CCLB21] Andrea Cossu, Antonio Carta, Vincenzo Lomonaco, and Davide Bacciu. Continual learning for recurrent neural networks: An empirical evaluation. *Neural Networks*, 143:607–627, November 2021.

[CDF21] Gaia Codeluppi, Luca Davoli, and Gianluigi Ferrari. Forecasting air temperature on edge devices with embedded AI. *Sensors*, 21(12):3973, June 2021.

[Cho17] Francois Chollet. *Deep Learning with Python*. Manning Publications, 2017.

[CMBR20] Jenny Cifuentes, Geovanny Marulanda, Antonio Bello, and Javier Reneses. Air temperature forecasting using machine learning techniques: A review. *Energies*, 13(16), August 2020.

[DBO16] Gabriel Martins Dias, Boris Bellalta, and Simon Oechsner. A survey about prediction-based data reduction in wireless sensor networks. *ACM Computing Surveys*, 49(3), November 2016.

[DBO17] Gabriel Martins Dias, Boris Bellalta, and Simon Oechsner. The impact of dual prediction schemes on the reduction of the number of transmissions in sensor networks. *Computer Communications*, 112:58–72, November 2017.

[DBT+19] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. Continual learning in practice. *CoRR*, abs/1903.05202, March 2019.

[DGLZ19] Hao Deng, Ziyan Guo, Rongheng Lin, and Hua Zou. Fog computing architecture-based data reduction scheme for WSN. In *1st International Conference on Industrial Artificial Intelligence (IAI)*. IEEE, July 2019.

[DMRM19] Behrouz Derakhshan, Alireza Rezaei Mahdiraji, Tilmann Rabl, and Volker Markl. Continuous deployment of machine learning pipelines. In Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi, editors, *22nd International Conference on Extending Database Technology (EDBT)*, pages 397–408. OpenProceedings.org, March 2019.

[DRAP15] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, November 2015.

[DV21] Giorgos Demosthenous and Vassilis Vassiliades. Continual learning on the edge with tensorflow lite. *CoRR*, abs/2105.01946, May 2021.

[DZF+20] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, August 2020.

[EP11] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, October 2011.

[Eri22] Ericsson. Massive IoT shows strong growth in 2021. *Ericsson Mobility Report*, June 2022.

[ERO+18] Olakunle Elijah, Tharek Abdul Rahman, Igbafe Orikumhi, Chee Yen Leow, and MHD Nour Hindia. An overview of internet of things (IoT) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of Things Journal*, 5(5):3758–3773, October 2018.

[FBT18] Yasmin Fathy, Payam Barnaghi, and Rahim Tafazolli. An adaptive method for data reduction in the internet of things. In *IEEE 4th World Forum on Internet of Things (WF-IoT)*. IEEE, February 2018.

[FSM+22] Ahmed Fathalla, Ahmad Salah, Mohamed Ali Mohamed, Nur Indah Lestari, and Mahmoud Bekhit. A novel dual prediction scheme for data communication reduction in IoT-based monitoring systems. In *7th International Conference on Internet of Things as a Service (IoTaaS)*, pages 208–220. Springer, December 2022.

[HKB+21] Tyler L. Hayes, Giri P. Krishnan, Maxim Bazhenov, Hava T. Siegelmann, Terrence J. Sejnowski, and Christopher Kanan. Replay in Deep Learning: Current Approaches and Missing Biological Elements. *Neural Computation*, 33(11):2908–2950, October 2021.

[HMD16] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio, Yann LeCun, Yoshua Bengio, and Yann LeCun, editors, *4th International Conference on Learning Representations (ICLR)*, May 2016.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer

Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[HS21] Yujiang He and Bernhard Sick. Clear: An adaptive continual learning framework for regression tasks. *AI Perspectives*, 3(2), January 2021.

[Hun07] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[IC18] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence (AAAI), April 2018.

[IEE16] IEEE Standards Committee. IEEE standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, 2016.

[JAK20] Khushboo Jain, Arun Agarwal, and Anoop Kumar. A novel data prediction technique based on correlation for data reduction in sensor networks. In *Proceedings of International Conference on Artificial Intelligence and Applications (ICAIA)*. Springer, July 2020.

[JNG+17] Haider Jawad, Rosdiadee Nordin, Sadik Gharghan, Aqeel Jawad, and Mahamod Ismail. Energy-efficient wireless sensor networks for precision agriculture: A review. *Sensors*, 17(8):1781, August 2017.

[JZXJ22] Lin Jia, Zhi Zhou, Fei Xu, and Hai Jin. Cost-efficient continuous edge learning for artificial intelligence of things. *IEEE Internet of Things Journal*, 9(10):7325–7337, 2022.

[KAM+18] Ronald Kemker, Angelina Abitino, Marc McClure, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence (AAAI), April 2018.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KHD20] Jeremias Knoblauch, Hisham Husain, and Tom Diethe. Optimal continual learning has perfect memory and is NP-hard. *CoRR*, June 2020.

[KMS20] David Kreuzer, Michael Munz, and Stephan Schlüter. Short-term temperature forecasts using a convolutional neural network — an application to different weather stations in germany. *Machine Learning with Applications*, 2(100007), December 2020.

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LGS09] Ming Li, Deepak Ganesan, and Prashant Shenoy. PRESTO: Feedback-driven data management in sensor networks. *IEEE/ACM Transactions on Networking*, 17(4):1256–1269, August 2009.

[LJD+18] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research (JMLR)*, 18(185):1–52, 2018.

[LKWL07] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, April 2007.

[LLS20] Sungjae Lee, Yung-Seop Lee, and Youngdoo Son. Forecasting daily temperatures with different time interval data using deep neural networks. *Applied Sciences*, 10(5):1609, February 2020.

[LSF19] Timothée Lesort, Andrei Stoian, and David Filliat. Regularization shortcomings for continual learning. *CoRR*, abs/1912.03049, December 2019.

[LYZ+17] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142, October 2017.

[MBB13] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, 4, 2013.

[MDP+17] Leonardo C. Monteiro, Flavia C. Delicato, Luci Pirmez, Paulo F. Pires, and Claudio Miceli. DPCAS: Data prediction with cubic adaptive sampling for wireless sensor networks. In *Green, Pervasive, and Cloud Computing (GPC)*. Springer, May 2017.

[MdSBF21] Carlos R. Morales, Fernando R. de Sousa, Valner Brusamarello, and Nestor C. Fernandes. Multivariate data prediction in a wireless sensor network based on sequence to sequence models. In *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, May 2021.

[MK11] Reza Askari Moghadam and Mehrnaz Keshmirpour. Hybrid ARIMA and neural network model for measurement estimation in energy-efficient wireless sensor networks. In *Proceedings of the Inernational Conference on Informatics Engineering and Information Science*. Springer, November 2011.

[MKL19] Jaewon Moon, Seungwoo Kum, and Sangwon Lee. A heterogeneous IoT data analysis framework with collaboration of edge-cloud computing: Focusing on indoor PM10 and PM2.5 status prediction. *Sensors*, 19(14):3038, July 2019.

[ML19] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, August 2019.

[MZC+21] Wenliang Mao, Zhiwei Zhao, Zheng Chang, Geyong Min, and Weifeng Gao. Energy-efficient industrial internet of things: Overview and open issues. *IEEE Transactions on Industrial Informatics*, 17(11):7225–7237, November 2021.

[OBL+19] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. KerasTuner. `https://github.com/keras-team/keras-tuner`, 2019.

[PGLM20] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, October 2020.

[PKP+19] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, May 2019.

[Pos81a] Jon Postel. Internet protocol. *RFC*, 791, September 1981.

[Pos81b] Jon Postel. Transmission control protocol. *RFC*, 793, September 1981.

[PQE+18] Charith Perera, Yongrui Qin, Julio C. Estrella, Stephan Reiff-Marganiec, and Athanasios V. Vasilakos. Fog computing for sustainable smart cities. *ACM Computing Surveys*, 50(3):1–43, May 2018.

[RBWA21] Emmanuel Raj, David Buffoni, Magnus Westerlund, and Kimmo Ahola. Edge MLOps: An automation framework for AIoT applications. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 191–200. IEEE, 2021.

[RDK+22] David Rolnick, Priya L. Donti, Lynn H. Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont,

Natasha Jaques, Anna Waldman-Brown, Alexandra Sasha Luccioni, Tegan Maharaj, Evan D. Sherwin, S. Karthik Mukkavilli, Konrad P. Kording, Carla P. Gomes, Andrew Y. Ng, Demis Hassabis, John C. Platt, Felix Creutzig, Jennifer Chayes, and Yoshua Bengio. Tackling climate change with machine learning. *ACM Computing Surveys*, 55(2):1–96, February 2022.

[RKSL17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017.

[RLVN17] Cristanel Razafimandimby, Valeria Loscri, Anna Maria Vegni, and Alessandro Neri. Efficient bayesian communication approach for smart agriculture applications. In *IEEE 86th Vehicular Technology Conference (VTC-Fall)*. IEEE, September 2017.

[Ros57] Frank Rosenblatt. The perceptron—a perceiving and recognizing automaton. In *Tech. Rep. 85-460-1*. Cornell Aeronautical Laboratory, 1957.

[SCBdS19] Tongxin Shu, Jiahong Chen, Vijay K. Bhargava, and Clarence W. de Silva. An energy-efficient dual prediction scheme using LMS filter and LSTM in wireless sensor networks for environment monitoring. *IEEE Internet of Things Journal*, 6(4):6736–6747, August 2019.

[SG19] Monika Schak and Alexander Gepperth. A study on catastrophic forgetting in deep LSTM networks. In *Lecture Notes in Computer Science*, pages 714–728. Springer International Publishing, 2019.

[SHK+14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SK11] Sujesha Sudevalayam and Purushottam Kulkarni. Energy harvesting sensor nodes: Survey and implications. *IEEE Communications Surveys & Tutorials*, 13(3):443–461, 2011.

[SM21] Christian Salim and Nathalie Mitton. K-predictions based data reduction approach in WSN for smart agriculture. *Computing*, 103(3):509–532, 2021.

[SR06] Silvia Santini and Kay Römer. An adaptive strategy for quality-based data reduction in wireless sensor networks. In *Proceedings of the 3rd International Conference on Networked Sensing Systems (INSS)*. Transducer Research Foundation, May 2006.

[SSH+18] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734, November 2018.

[TBKV21] Trang Thi Kieu Tran, Sayed M. Bateni, Seo Jin Ki, and Hamidreza Vosoughifar. A review of neural networks for air temperature forecasting. *Water*, 13(9), May 2021.

[TMLD18] Gaby Bou Tayeh, Abdallah Makhoul, David Laiymani, and Jacques Demerjian. A distributed real-time data prediction and adaptive sensing approach for wireless sensor networks. *Pervasive and Mobile Computing*, 49:62–75, September 2018.

[TYW18] Huangshi Tian, Minchen Yu, and Wei Wang. Continuum: A platform for cost-aware, low-latency continual learning. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, page 26–40. ACM, October 2018.

[vdVT19] Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *CoRR*, abs/1904.07734, April 2019.

[Was21] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[WM10] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference (SciPy)*, pages 56 – 61. SciPy.org, June 2010.

[Won19] Alexander Wong. NetScore: Towards universal metrics for large-scale performance analysis of deep neural networks for practical on-device edge usage. In *Proceedings of 16th International Conference on Image Analysis and Recognition (ICIAR)*, pages 15–26. Springer, August 2019.

[WS20] Pete Warden and Daniel Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly, 2020.

[YWS17] Tianqi Yu, Xianbin Wang, and Abdallah Shami. A novel fog computing enabled temporal data reduction scheme in IoT systems. In *2017 IEEE Global Communications Conference (GLOBECOM)*. IEEE, December 2017.

[ZB21] Sherali Zeadally and Oladayo Bello. Harnessing the power of internet of things based connectivity to improve healthcare. *Internet of Things*, 14(100074), June 2021.

[ZQD+21] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, January 2021.