Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelor Thesis

Title of the Thesis // Titel der Arbeit

Scientific software development of a Convolutional Neural Network for the reconstruction of missing data from a weather measurement station using numerical model data.

Wissenschaftliche Softwareentwicklung eines Convolutional Neuronal Networks für die Rekonstruktion fehlender Daten einer Wettermessstation unter Verwendung von Numerischen Modelldaten

---

Academic Degree // Akademischer Grad
Bachelor of Science (B.Sc.)

Author's Name, Place of Birth // Name der Autorin/des Autors, Geburtsort
Timo Wacke, Hamburg

Field of Study // Studiengang
Computing in Science (Physics Specialization)

Department // Fachbereich
Computer Science // Informatik

First Examiner // Erstprüferin/Erstprüfer
Prof. Dr. Thomas Ludwig

Second Examiner // Zweitprüferin/Zweitprüfer
Dr. Christopher Kadow

Matriculation Number // Matrikelnummer
7434883

Date of Submission // Abgabedatum
10.06.2024

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Eidesstattliche Versicherung

Wacke Timo
Last Name, First Name // Name, Vorname

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem Titel

Wissenschaftliche Softwareentwicklung eines Convolutional Neuronal Networks für die Rekonstruktion fehlender Daten einer Wettermessstation unter Verwendung von Numerischen Modelldaten

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Hamburg, den June 3, 2024
Place, Date, Signature // Ort, Datum, Unterschrift

Abstract

Zusammenfassung

Acknowledgements

Thank you...

Contents

List of Figures

1 Introduction

Weather station density varies greatly across the globe, depending on population density, economic development, and the availability of infrastructure. [**?**] While any weather station can experience downtime, the reliability of weather stations in regions with low station density is often low as well [**?**]. So not only is downtime in regions where data is limited more likely but also more impactful because there are fewer neighboring stations to help compensate for the missing data.

A denser, more reliable network would benefit weather forecasting, helping to evacuate populations timely before natural disasters [**?**] and from a global perspective aiding climate research. For example in East Africa, the weather station density is very low, but the region would be of great interest to the El Niño Southern Oscillation (ENSO) research. The irregular fluctuation between El Niño and La Niña phases affects the climate from the tropics to even higher-latitude regions through teleconnections. [**? ?**] An innovative approach to increase the density of weather stations could be to use low-cost weather stations that could be 3D-printed and assembled by the local population [**?**]. Either way, low-cost weather stations have reliability issues and are prone to downtime which is outlined in section **??**.

In light of the challenges posed by sparse weather station coverage, novel methodologies are required to address the reconstruction of missing weather data. One promising avenue involves the application of machine learning techniques, which offer a departure from traditional numerical reconstruction methods such as kriging, which are reliant on neighboring station data and are often computationally intensive [**?**]. The application of machine learning in this case would be to connect numerical reanalysis data that describes the weather in grid cells meaning it's to some degree blurry, with the local patterns that lead to measurements at a weather station. This would allow for independent operation and would beat numerical methods in terms of needed computational resources needed by orders of magnitude [**? ? ?**] as the application of the trained machine-learning model. By leveraging available local data, these techniques, such as Convolutional Neural Networks (CNNs), can be trained to estimate weather conditions at a designated time by assimilating global numerical weather model data. Despite the inherent blurriness of aerial data provided in grid cells, these models are anticipated to discern and adapt to local weather patterns such that they become capable of transferring knowledge from the meta situation to the local situation. This paper aims to achieve reconstruction using that approach which will be further explained in **??**.

The reanalysis of choice in this endeavor is the ECMWF Reanalysis v5 (ERA5), which

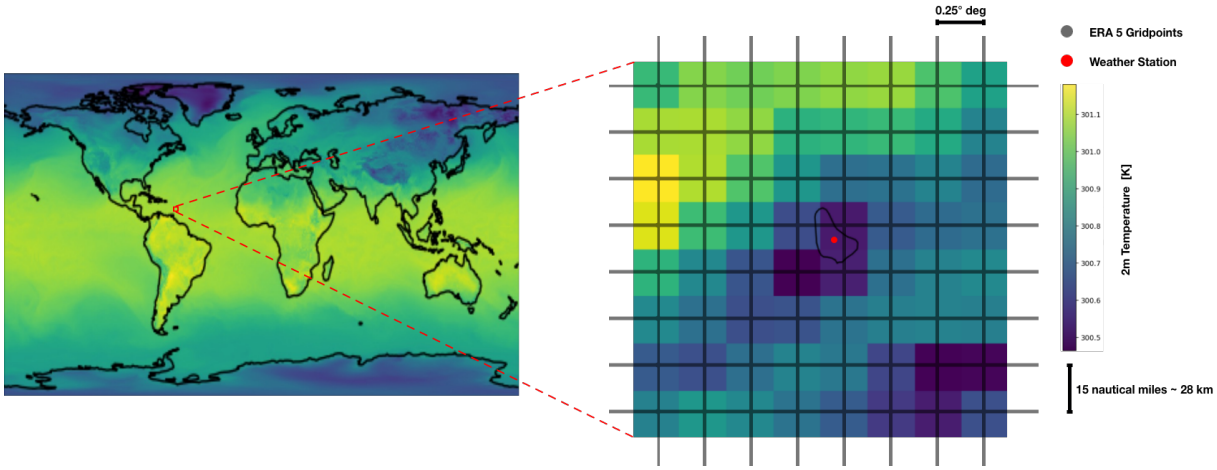


Figure 1: 8x8 grid-points of ERA5 with 2m temperature for 2020-06-23 19:00 UTC in the area of a weather station on Barbados

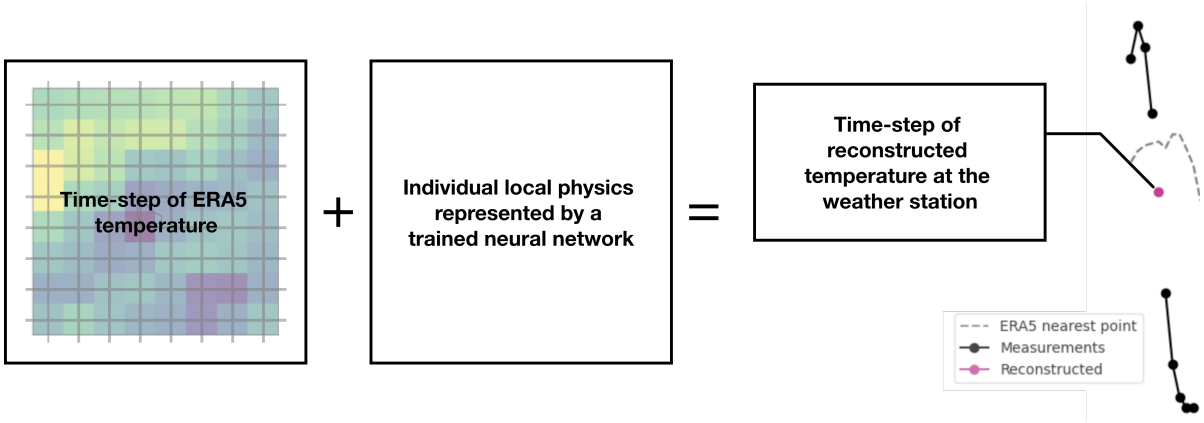


Figure 2: Conceptual Framework of the proposed method

covers the globe in grid cells of 0.25° x 0.25°. The data is available in hourly timesteps from 1940 to the present and contains a wide range of variables, such as temperature, precipitation, wind speed, and many more. [**?**]

To prove the concept it's likely easiest to start with temperature data, meaning the 2m temperature variable from the ERA5 reanalysis will be used as input to the neural net.

2 3D printed Weather Stations

3 Conceptual Framework

3.1 Concept

The conceptual framework of the proposed method is illustrated in **??**. Applying the local bias of the weather station against ERA5 on top of ERA5 would reconstruct the temperature data at the station. The idea is that a Convolutional Neural Network would learn the complex behavior of the bias depending on the aerial situation if you train it with enough available measurements. Once the CNN is trained as a model that represents the local bias of the weather station vs the ERA5 data, applying it to the ERA5 data allows for the reconstruction of missing temperature values at the weather station just by providing the data of the ERA5 temperature in the regional cutout at that timestamp used in training as an input. To illustrate what such a bias could be: In the case of Barbados for example the grid cells of the ERA5 data all lay primarily in the ocean, meaning the diurnal cycle has a much lower amplitude than at the weather station that is placed on land, the neural network would need to learn how to detect based on the 64 grid points in which phase of the diurnal cycle the weather station is and then adjust the temperature values accordingly. Besides this obvious difference between ERA5 and the measurements at Barbados, there are most likely many more local effects to master.
Since the ERA5 data is available everywhere and for any timestep, for any hour without measurements at the weather station that the model is trained for could be reconstructed. To train the model, a supervised learning approach would be used, meaning that the input will be ERA5 data at times when measurements are available and the expected output should be the measurements at the weather station. The model would be trained to minimize the difference between the expected output and the model's prediction through backpropagation. To allow for flexibility in application and simplicity in training, the training has to be done hour by hour, meaning the model is passed during training one timestep at a time and the weights are updated between iterations. When reconstructing missed data, hence the model is applied to the ERA5 data hour by hour as well and the result is a series of hourly predictions that are not connected in time.

3.2 Data Acquisition and Preprocessing

Upon obtaining a dataset from a weather station, it is determined where temperature data is missing. While the weather station dataset is minute-based, data could be missing only

for a few minutes within an hour instead of the full hour. This would raise the question of how many missing values are acceptable to not mark the hour as missing. Sure is, that if all temperature values are missing during an hour, the hour is marked as missing. The ERA5 data then needs to be cropped to the neighboring 8x8 grid cells, while centering the cutout as close to the weather station as possible. The available longitudes and latitudes in the ERA5 model are spaced by 0.25° along the latitude and longitude from each other, when 8x8 grid cells are selected it needs to be assured that the grid points are such selected that the coordinates of the weather station are between the 4th and 5th grid point in each dimension. After cropping the ERA5 dataset geographically, the data needs to be cut and divided along the time axis to match the weather station data, leading to two datasets: one with all the hours marked as missing and one with all the hours marked as present. Until the model is trained, only the dataset with all the hours marked as present will be used.
As a result, we have a dataset pair of weather station measurements and ERA5 data, that are coherent in time and space.
To determine after the training if and to which extent the model learned to reconstruct the missing data, the dataset pair is split again along the time axis into a pair of station with ERA5 data for training and a pair of station with ERA5 data for validation. Thus we can later let the model reconstruct values that have actually been measured but haven't been included in the training so we then can validate how successful the reconstruction is. With the datasets prepared, the next phase involves configuring and training the Convolutional Neural Network (CNN) for the temperature reconstruction task. The CNN architecture is tailored to accept input in the form of 8x8 grid cells centered around the weather station's location. Employing a supervised learning approach, the CNN is trained using pairs of hourly temperature data from the weather station and corresponding grid cell data from ERA5. The training process iteratively feeds batches of data into the CNN, fine-tuning its parameters to minimize prediction errors and optimize accuracy in reconstructing missing temperature values.

3.3 Model Evaluation

Following the training phase, the CNN's performance is evaluated using the validation set. The model's capacity to accurately reconstruct missing temperature data at the weather station is scrutinized against ground truth values. This evaluation step serves to gauge the CNN's proficiency in capturing intricate weather patterns and producing precise temperature estimations. For that, the root mean squared error (RMSE) and the

correlation coefficient are calculated. The RMSE is a measure of the differences between predicted and observed values, while the correlation coefficient quantifies the strength and direction of the linear relationship between the two datasets. An obvious choice as a time range for the evaluation would be to cut out one complete year so that the model can be evaluated over the full range of seasons and weather conditions.

3.4 Application to New Data

Upon successful training and validation, the model is trained again with the measurements that have been excluded before for the benefit of validation. After training on the full data, the CNN can be used to fill gaps. Fundamentally any list of timesteps that the model should be applied for can be requested and then the ERA5 data for the respective timesteps is obtained, cropped in the same way to the geographical region as before and then the model is applied to the data. The result is a list of temperature values that are not connected in time, but are the model's prediction for the temperature at the weather station at the respective time. Since in **??** we split the ERA5 dataset already into two datasets, one with all hours marked as missing and one with all hours marked as present, the model can be applied to the dataset with all hours marked as missing and then directly infilled into the original measurements dataset.

4 Theoretical Background

4.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) have excelled at extracting patterns and features from spatial data such as satellite imagery, radar data, and climate model outputs. This capability has enabled researchers to better understand and predict complex atmospheric and oceanic phenomena. For instance, CNNs have been employed to detect and localize extreme weather events from satellite data, enhancing early warning systems and disaster response efforts. Furthermore, CNNs have improved the ability to identify forced climate patterns and disentangle natural variability from human-induced climate change signals, advancing our understanding of climate dynamics and informing mitigation strategies.
CNNs are a type of deep learning architecture inspired by the visual cortex of animals. They are designed to efficiently capture spatial and temporal dependencies in data through the use of learnable filters and hierarchical feature representations. Through the use of convolutional layers instead of fully connected layers, the architecture is able to preserve

Figure 3: How a convolution operation works. [**?**]

the spatial structure of the input data, making it particularly well-suited for image and video data. This approach not only simplifies pattern detection but also implies a reduction in the number of parameters, which minimizes the necessary computational resources.
Application of CNNs in climate science has yielded several notable contributions, including the reconstruction of the El Nino event of 1877 by Kadow et al. despite extremely limited data availability. [**?**]
In the context of this work an architecture introduced by ([**?**]) is used, which consists of an encoder and a decoder part as seen in **??**. Due to it's shape it is called U-Net. The encoder part consists of convolutional layers and pooling layers, while the decoder part consists of upscaling layers and in this case, skip connections as proposed by [**?**].

Convolutional Layer

The convolutional layer is the driver for feature mapping in a CNN. It applies a convolution operation shown in **??** to the input data. The operation is done element by element while sliding a filter (also called kernel) over the input data such that situations, where the filter overflows the input data ranges, are avoided or taken care of. On each step, the Frobenius Product between the kernel and the submatrix given by the current position and the kernel dimension is calculated and noted in the output matrix. The parameters in the kernel matrix are chosen in such a way that the Frobenius Product is maximized when the kernel is over a feature that the kernel is supposed to detect. In **??** the kernel

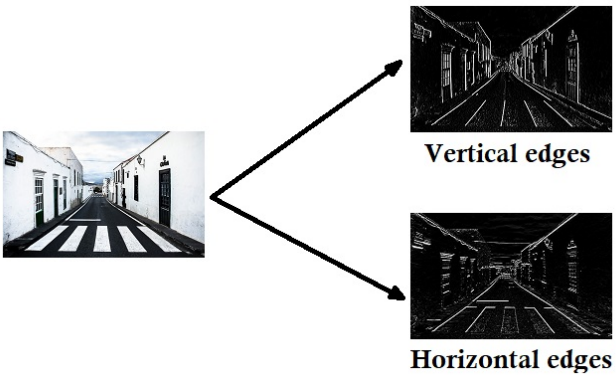


Figure 4: Example of edge detection with a convolutional kernel. [**?**]

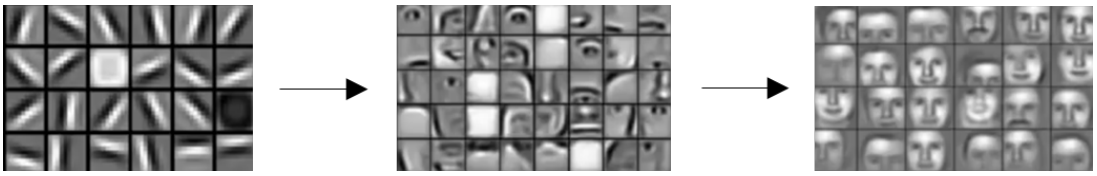
for example is a vertical edge detector, meaning it will output a high amount (positive or negative) when the horizontal gradient in the input data submatrix has a high absolute value. This result is rather trivial, as a positive horizontal gradient as seen in the upper-left 3x3 submatrix of the example leads to a right column that when negatively weighted overweights the positive-weighted left column and thus the output for the upper-left 3x3 submatrix is negative. Conversely, a Fresenius product of the upper-right 3x3 submatrix with the kernel returns a positive value, hinting at a negative horizontal gradient in the input data. The result of such a convolution can be observed in **??**.
In the context of weather data, the convolutional layer can be used to detect any weather patterns not just edges and the kernel itself can be learned by the network.

Activation Function

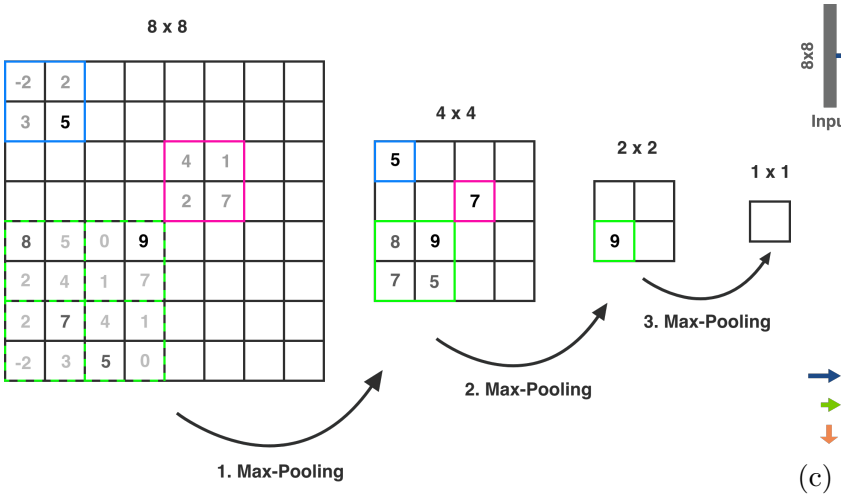
To map the output of the convolutional layer to a meaningful space, and to avoid negative values, an activation function is applied. This introduces non-linearity into the network. The simplest activation function is the ReLU function, which returns the input if it is positive and zero otherwise. It is defined as $f(x) = max(0, x)$. The ReLu function is the activation function used in this work after each convolutional operation.

Pooling Layer

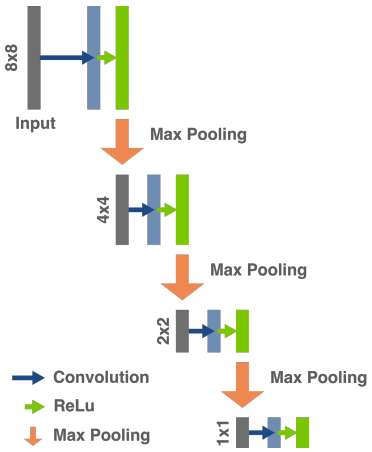
The exact position of a low-level feature in the input data is not so important when it comes to detecting high-level features, it is more important to recognize if a feature is present at all in certain spatial areas of the input data or not. Thus scaling down the resolution of the matrix by combining every 2x2 submatrix into one value can be beneficial. It is most commonly aggregated by taking the maximum value of the submatrix because



(a) Low-level features are combined into high-level features.



(b) Illustration of 3 max pooling operations



(c) Alternation of convolutional and pooling

Figure 5: Abstraction in the encoding part of a CNN.

it works for the mentioned purpose to detect if a feature is present in the area of the submatrix or not. While the convolution layer depending on how the convolution is processed near the borders of the input data reduces the dimensionality of the input data just slightly or not at all, the pooling reduces the dimensionality drastically. So after a 2x2 pooling operation, the output matrix is only a quarter of the size of the input matrix. For the 8x8 grid cells of the ERA5 data, that is used as input in the laid out approach (see **??**), the architecture of the CNN could include a maximum of 3 pooling layers, reducing the input data to a 1x1 matrix as seen in **??**. **??** just illustrates the downsampling of the data, in the actual architecture pooling layers always follow convolutional layers and are never applied directly after each other as seen in **??**.

Upscaling Layer

In the decoding part of the U-Net, the shape of the data is restored by upscaling the encoded layers with the detected feature again. This is simply done through Gaussian interpolation.

?? Question: What else can I explain here?

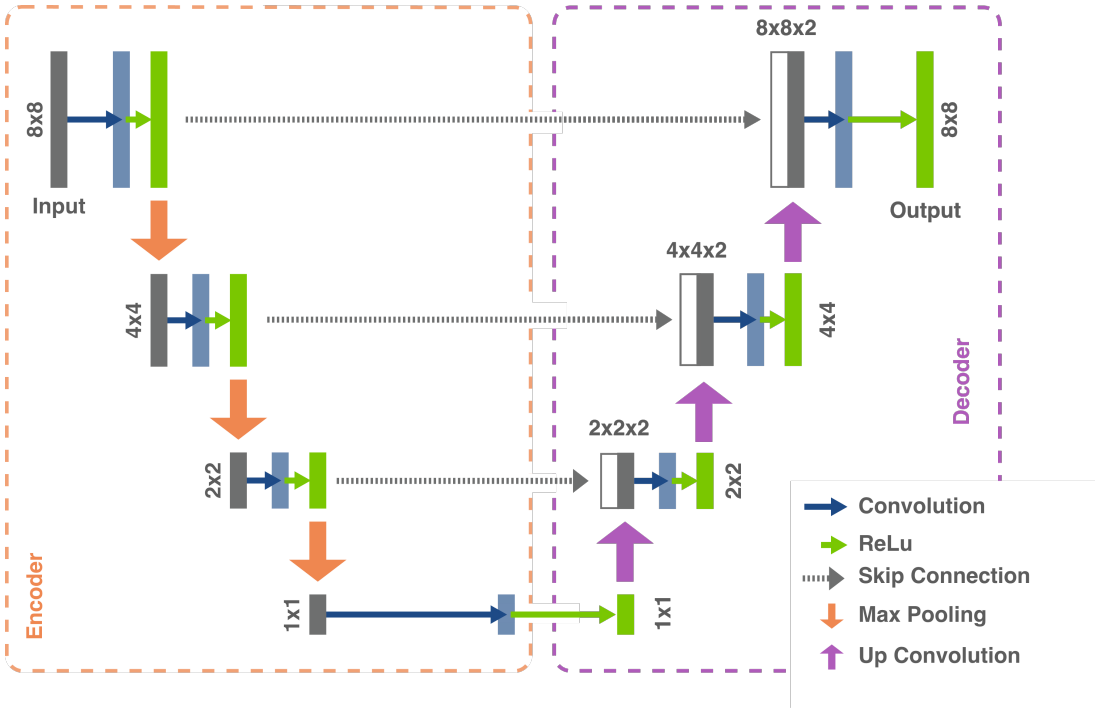


Figure 6: Encoder Decoder Architecture in the U-Net.

A very important part of the decoder is the usage of skip connections ([**?**]).

Skip Connections

??? Question is this even helpful when not infilling but using "target" data as input?
For the skip connections, the outputs on each layer of the encoding part are copied over and directly factored into the decoding part. This helps the network to remember the context of the input data, helping to preserve the context of the input data, which is especially important in spatial infilling tasks, where the reconstruction needs to match the available data around the missing data. To combine the data from the skip connections with the data from the upscaling layers, a specialized convolution is used...
?? Question: Is this correct? Or Convolution->ReLU + Pooling? How is the dimensionality halved again?
The now-explained steps used in the architecture combined will return for any input data a prediction of the same shape as the input data, where the values are the predicted values for the missing data. What the values exactly will be is completely dependent on the parameters used in the convolution kernels.
?? Question: What else weights are there? The architecture does not have fully connected layers, does it? So no weights there. What about the pooling and upscaling layers? Are

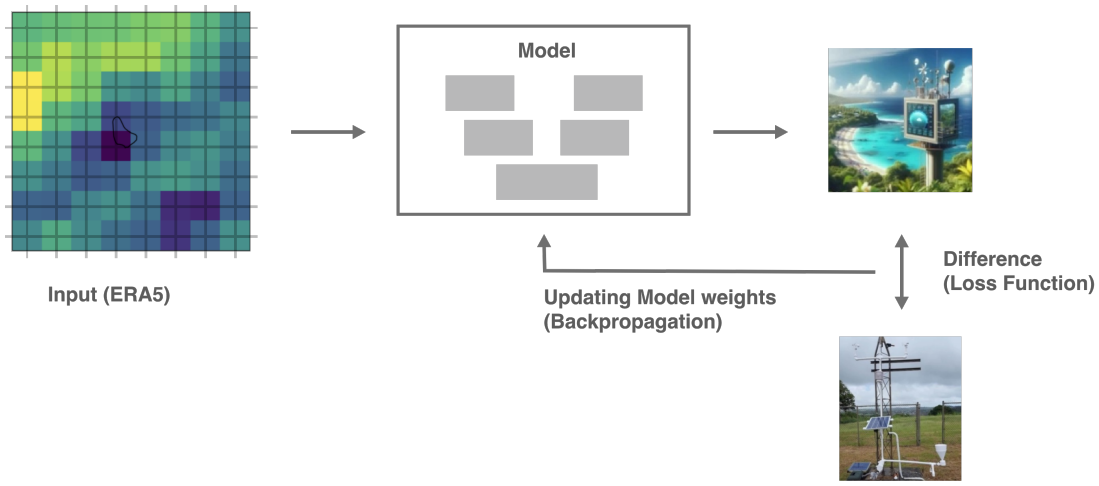


Figure 7: Schema of supervised learning.

there weights in the activation function?
With the correct weights found the prediciton should be ideally the same as the target data. This is what the training process is for which is supervised based on the available measurements, the ground truth (see **??**).

Backpropagation and Loss Function

??? I am using Loss Function 3, what should I explain here in the context of a bachelor thesis about it?
To minimize any differences between ground through and prediction, the differences first need to be quantified in a loss function. Then the partial gradient of the loss function concerning each weight in the network is calculated. This is done through backpropagation, recursively applying the chain rule of calculus. The weights are then updated in the opposite direction of the gradient, such that the loss function is minimized. The learning rate determines how big the adjustments are made and is a crucial hyperparameter in the training process.

4.2 Reanalysis - ERA5

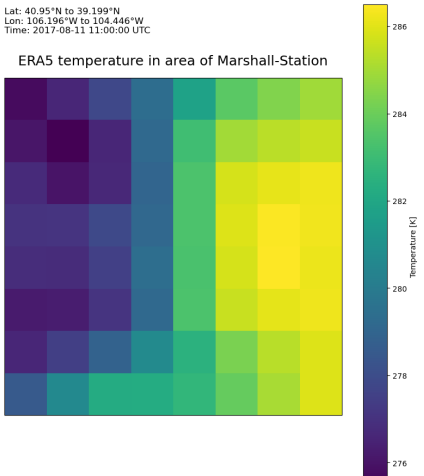
Atmospheric reanalysis is an effort to combine measurements with numerical models to estimate the state of the atmosphere at any given time, opposing missing measurements. The method is based on a complex numerical model, simulating the physical rules of the atmosphere. As with any numerical model, degrees of freedom, allow for adjustments of parameters, which are chosen in such a way that the model output matches the actual

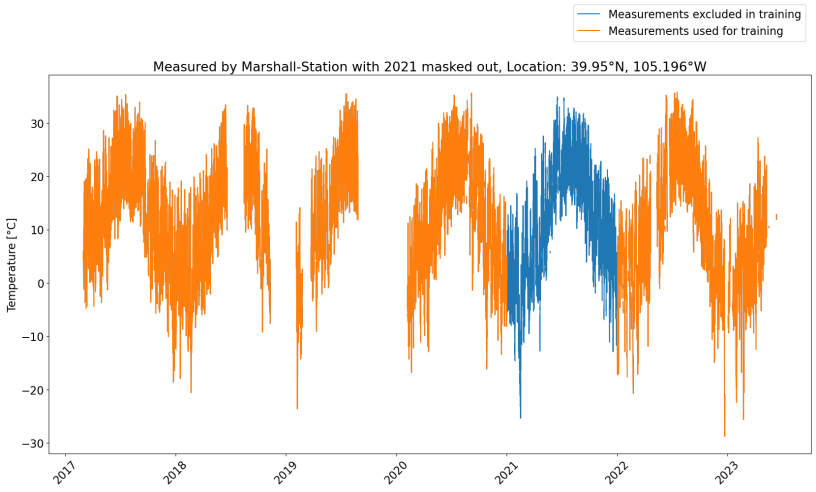
measurements wherever they are available in space and time.
The ERA5 reanalysis is a product of the European Centre for Medium-Range Weather Forecasts (ECMWF) and is the fifth generation of the ERA reanalysis series. It is the most complete and accurate data assimilation project available [**?**]. Which makes it the best choice as input data in this approach. As it is the best available data in remote locations such as in East Africa [**?**], where the 3D printed weather stations are specifically supposed to make an impact it can be used as a type of benchmark to compare the results of the infilling approach to.
As mentioned above the data is available in grid cells of 0.25° x 0.25°, which is approximately 28 km x 28 km at the equator. The data assimilation takes place in 4 dimensions, as not only the latitude and longitude over time are taken into account but also multiple pressure levels in the atmosphere. This gives the techniques used for ERA5 the name 4D-Var [**?**] The available variables in the ERA5 reanalysis are numerous, but for this work, only the 2m temperature is used as input to the neural network. Which is the temperature weather stations aim to measure. However, for similar approaches or extended approaches, other variables such as different wind-(component) speeds, precipitation, or humidity and solar radiation could be used as well.

5 Results

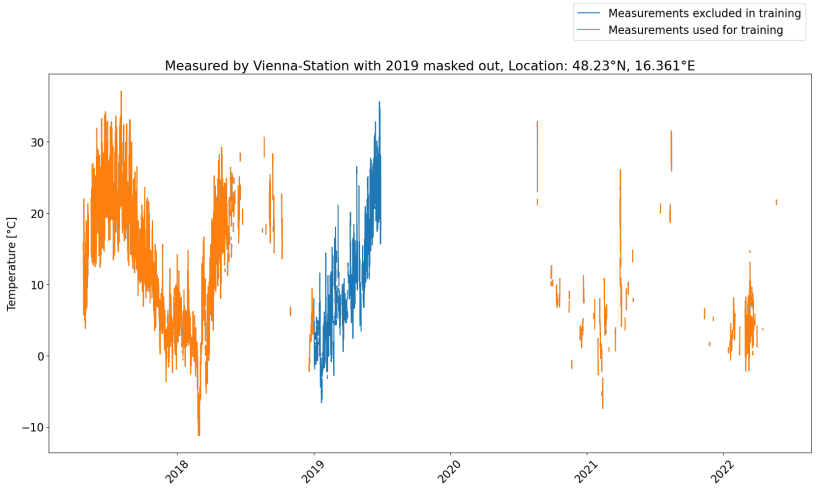
5.1 Included Weather Stations

From the 3D-Paws project of NCAR, measurements from three different weather stations in recent years were obtained. The stations are located in Marshall, Colorado; Vienna, Austria; and Barbados, Caribbean. The measurements span up to late 2023, with the quantity of available temperature data varying significantly between stations. The station in Marshall is one of the oldest in the 3D-Paws project and has the most available measurements among the three stations. The weather station in Vienna also has data dating back to 2017, but it has been almost completely offline since mid-2019. The weather station in Barbados began recording in mid-2020
The station in Marshall is located between Boulder and Denver next to Marshall Lake, on an Elevation of 1743m, less than 10km east of the South Boulder Peak, which is already more than 2600m high [**?**]. That the station is located just about 30 kilometers east of the Colorado Front Range where the Rocky Mountains range in their

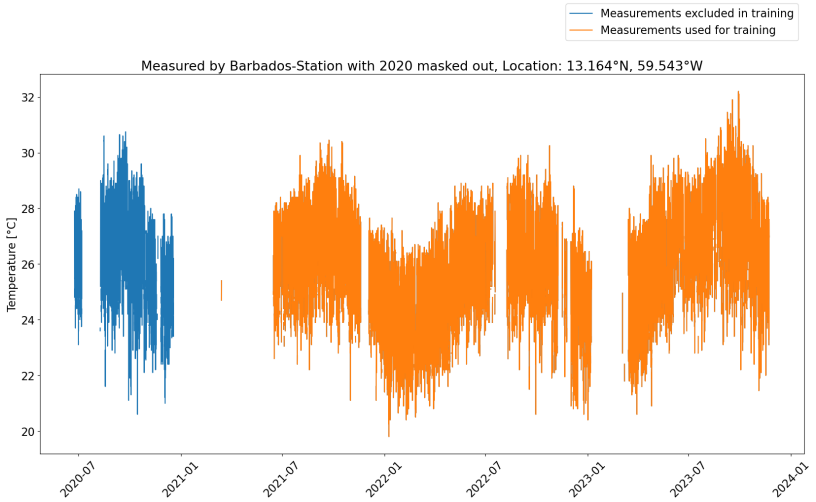




(a) Station in Marshall, Colorado, USA



(b) Station in Vienna, Austria



(c) Station in Barbados, Caribbean

Figure 8: Available data from 3D printed Weather stations

altitude between 3250m - 4000m [**?**] is visible in the ERA5 data **??**.
Vienna is situated to the west within the Vienna Basin, at a relatively low altitude just beyond the Alpine region. The Alps, which rise moderately western of Vienna, reach heights of up to 2000 meters above sea level in a distance of approximately 100km, meaning the ERA5 resolution is high enough to differentiate the atmospheric conditions and topographical features within this region better.
The station in Barbados is located in the parish of St George at an altitude of 274m, the island itself is relatively flat with the highest point being Mount Hillaby at 340m. The island is 34km long and up to 23km wide, which is approximately the size of an ERA5 grid cell, but as can be observed in **??** all surrounding grid cells are ocean heavy, while the stations location on Barbados is close the the point where the landmass is the widest. The ERA5 data is therefore not able to capture the diurnal cycle of the station as it is located on land, while the surrounding grid cells are mostly ocean.

5.2 Data availability

In **??** the measurements of the three stations are displayed, after data cleansing and converting them on an hourly basis. The measurements were provided by the NCAR with most invalid values already marked as such but especially in Barbados the sensors had more noise and occasional invalid measurements such that they needed extra cleansing. To do so, values at temperatures near zero and below were excluded. The conversion from minute to hourly data was done by taking the average of all minute values in that specific hour, but only if there were more than 20 values available and only if the values weren't all the same. It has been observed that the sensors sometimes get stuck and then deliver the same value for a longer period. The dataset for the Marshall station spans from 2017 to 2023, comprising 41,883 data points. These measurements reveal three significant data gaps along with several smaller ones. The larger intervals without data include the midsummer period of 2018, late 2019 to early 2020, and the most extensive gap occurring over five months from September 2019 to January 2020
For the Vienna Station, measurements are available for only 12,477 hours, which is less than a third compared to Marshall. This is primarily due to extended downtimes starting from mid-2018, resulting in few to no measurement data. Except for a brief period from

late 2018 to July 2019, when continuous measurements resumed. After that, there were only sporadic data collection instances, before the station went preliminary offline in April 2022.
The Barbados Station only suffered 2 downtimes that were significantly longer than a month with the biggest being roughly the first half of 2021 and the 2nd biggest being the first quarter of 2023. In total the station has 17,315 hours of measurements, which is still less than half of the Marshall station.

Splitting into Training and Validation Data

As explained in **??**, the measurements are split into a training and validation dataset. The training dataset is used to train the model, while the validation dataset will be kept aside to compare it to what the model predicts after the training to validate if and to which extent the model learned to reconstruct the missing data. If the validation data would be included in the training data, the model would be able to predict the values it has already seen, but it would be impossible to tell if the model learned to generalize the local wather patterns. Starting with the Marshall station, the data was sufficient to extract an entire year as validation data. Excluding a consecutive year from the training data not only allows for a comprehensive analysis of a full yearly cycle but also ensures that the model relies solely on the general weather patterns it learned from the training data to predict the values of the validation data. This approach prevents the model from potentially "cheating" by accessing information from nearby training data points, which could compromise the integrity of the validation process. It is a common practice in the machine learning field to validate data as a contiguous set, as it helps to maintain the independence and integrity of the validation process. 2021 highlighted in blue in **??** was the most complete year in the Marshall dataset thus it was chosen as the validation data. With the mask of 2021 applied the training data for Marshall consists of 34,188 hours of measurements which is about 82% of the total data.
For the Vienna station, the data was split into training and validation data based on the availability of the data. The training data consists of all available data up to 2019, while the validation data is the year 2019. Resulting in 9,593 hours of measurements used as training data, which is about 77% of the total data. The limited availability of measurements didn't allow for a full year of validation data.
The Barbados station also only had 2022 and 2023 as a complete available years, so that the decision to use the available data of 2020 as a training was a result of the lack of data as well. With the mask applied, the training data consists of 14,576 hours of measurements,

which is about 84% of the total data.

5.3 Validation of the Models

For the training and validation data, the corresponding temperature data of the 64 ERA5 grid cells in the regions of each station was obtained. The 8x8 grid cells were chosen to be centered as good as possible around each weather station, given the specific ERA5 coordinates. The ERA5 data also was preprocessed using the code pipeline described in **??** and then the CNN with the architecture described in **??** was trained on the training data. The training was done in batches of 4 datapoints, which are in hour case the hourly steps, at a time. The model was trained for up to 300,000 iterations, which therefore are depending on the dataset size between 8 and 31 epochs. The training was done on the Supercomputer "Levante" of the German Climate Computing Center, which made it possible to complete training runs in just a few hours. However, the training can be done in the same way on almost any machine. During the training, in the scientific process, where the approach is un-proven to work, it is especially important to validate multiple times during the training to calculate not only the loss function over the model for the training data, but over the validation data. This is done to prevent overfitting, a so called phenomena where the model in the aim to minimize the loss for the training data, which is what training using backpropagation is exactly at it's core, starts to learn the noise and any potential specifics of the training data so well, that it loses it's ability to generalize and the loss for the validation data starts to increase. This is a sign that the model is overfitting and the training should be stopped. The model that is then saved is the one that performed best on the validation data. To allow for that, not only was the ERA5 data for the training data provided to the machine learning architecture but also the validation ERA5 data paired with the expected output for these, the masked measurements of the weather station.
Training runs have been completed for all stations with up to 1 million iterations, but the best models were in most cases already found between 300 and 500 thousand iterations. The models were then applied to the ERA5 data at all the hours in the validation data set of each station, to analyze the performance of the model for each station.

Root Mean Squared Error (RMSE)

To quantify how far apart the reconstructed temperature series $\hat{y}$ is from the measured temperature series y, where both series have n different timesteps and thus can be seen as vectors most commonly the root mean squared error (RMSE) comes to help, which is

strongly connected to the Euclidean norm of the difference between both vectors $\hat{y} - y$. However, a simple mean absolute error (MAE), also called the Manhattan norm could seem most intuitive and easiest to understand. It would be calculated just by adding all absolute differences up before dividing it by the length of the series. The RMSE is favored in machine learning and statistics because it penalizes larger differences more than smaller ones, which is more in line with the human perception of errors.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

When converting **??**, it can be seen how the RMSE is connected to the Euclidean norm:

$$\text{RMSE} \cdot \sqrt{n} = \sqrt{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \|y - \hat{y}\|_2 \tag{3}$$

The Euclidean norm $\|y - \hat{y}\|_2$, of a coordinate in our case the error $\hat{y} - y$ can be imagined as the diagonal distance between the origin and the point with these coordinates. For demonstration purposes, a right triangle in 2-dimensional space can be imagined with the errors at 2 validation timesteps as the legs. The Euclidian norm over both errors would be proportional to the hypothenuse of that triangle which is more dependent on the length of the longer leg than the shorter one. The Manhattan norm (**??**), on the other hand, which received its name from the gridded street system of Manhattan is not based on the diagonal distance, but proportional to the distance a taxicab in Manhattan would have to drive to travel along the hypothenuse, the equally weighted sum of the legs.

Pearson Correlation Coefficient

The Pearson correlation coefficient (in the following called correlation coefficient), is a measure of the strength and direction of a linear relationship between two variables. It ranges from -1 to 1, where 1 indicates a perfect positive linear relationship, -1 is a perfect negative linear relationship, and 0 is no linear relationship. The correlation coefficient is calculated as follows:

$$\text{Correlation Coefficient} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2 \sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2}} \tag{4}$$

Where $\bar{y}$ and $\bar{\hat{y}}$ are the mean values of the measured and the model output temperature

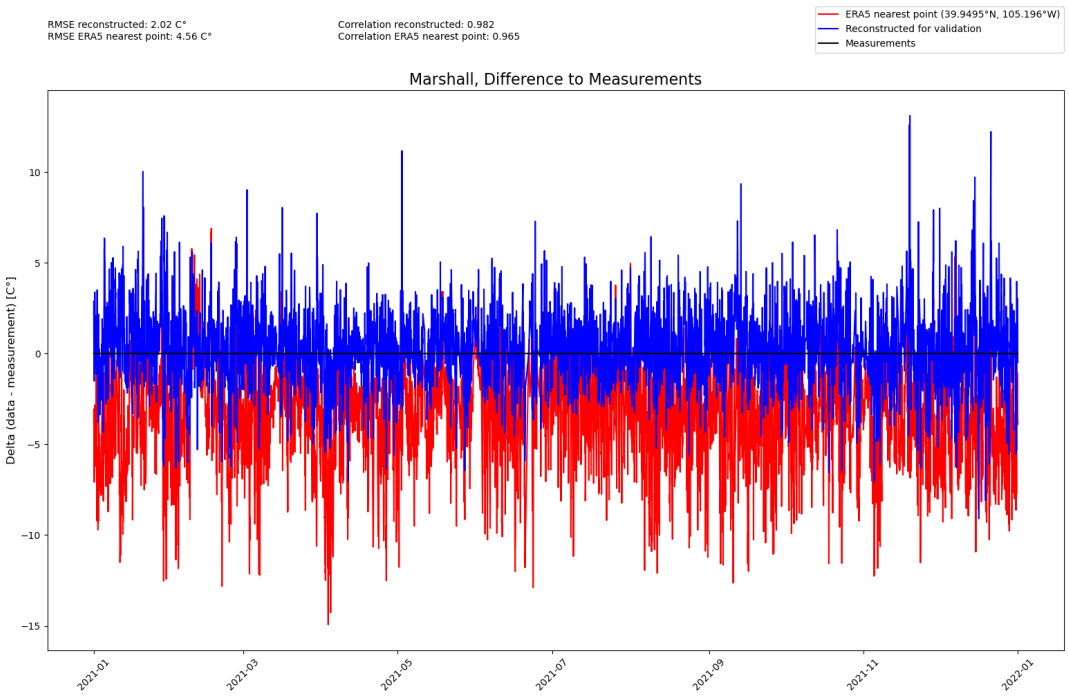


Figure 10: Difference between reconstructed and measured temperature for Marshall-Station

series, respectively. The correlation coefficient is a measure of how well the model output follows the general trend of the measured data. [**?**]

Marshall-Station

To begin with Marshall the occasion with the most available training data, all by the model predicted temperatures for the hourly measurements of the validation dataset have been plotted in **??** with a blue line against the ground truth in black. Given the high number of hourly timesteps along the x-axis spanning the year 2021, for the sake of readability, the temperature differentials are plotted on the y-axis instead of the absolute temperatures. Above the chart the over the whole hourly validation data calculated RMSE and Correlation of the reconstructed series are written. The RMSE for the Marshall station is 2.02°C, which is the highest value among the three weather stations, which models have been trained for. To compare the RMSE to what a direct read out of ERA5 would have been, the plot includes in red the temperature at each timestep at the nearest ERA5 grid point, at the coordinates 39.950° N, 106.196° W. The grid point's location is exactly the location of the weather station in Marshall (see **??**). But the RMSE between the measurements and the temperature time series of that ERA5 grid point, is 4.56°C, which

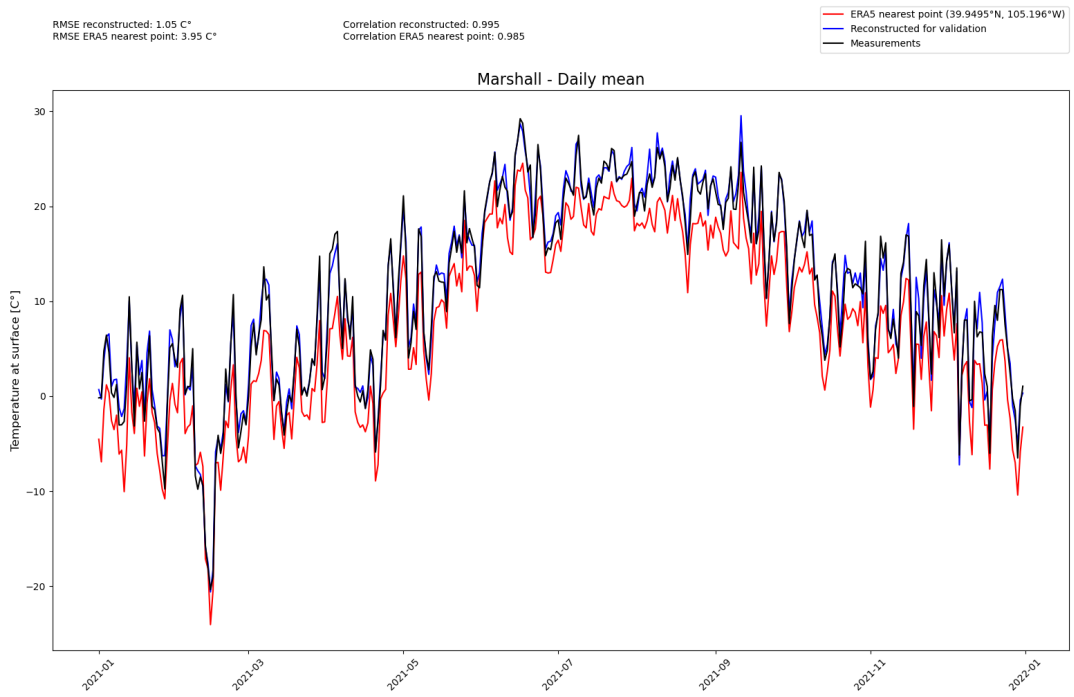


Figure 11: Reconstructed temperature vs measured temperature for Marshall-Station (Daily mean)

is more than twice as high. Considering that the model was trained on the ERA5 data of the 8x8 grid cells around the station, the model was able to reduce the error significantly. The ERA5 data, as seen in the chart has a substantial low-bias compared to the station which directly relates to the fact that the grid cell of the station already reaches into the Rocky Mountains, as described above. Thus it is especially important, that the reconstruction not just has a lower RMSE which could be achieved in this case just by correcting the low-bias but also beats it's training data in terms of correlation. The calculated correlation coefficient over the validation year 2021 on an hourly basis for the reconstructed temperature and the measurements is 0.982 compared to 0.965 for the ERA5 nearest grid point a significant improvement.

Even better is the performance of the model when the hourly values are combined again with daily means, as in **??**. The chart shows the daily mean temperature of the three different series, known from the previous chart over 2021. In this case, the reduced number of timesteps allows for an absolute temperature scale on the y-axis and therefore makes the performance of the model visually appealing as well, the RMSE on a daily basis for the prediction is almost halved the original one at 1.05°C, while the correlation coefficient is even higher as well at 0.995, the ERA5 correlation coefficient improved significantly as

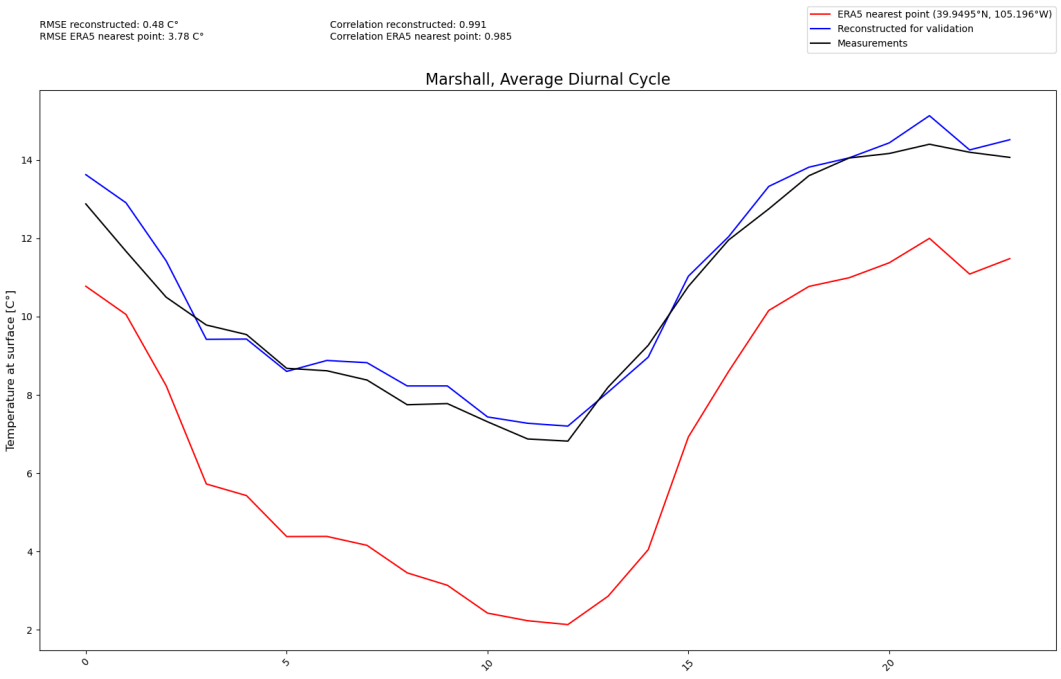


Figure 12: Reconstructed temperature vs measured temperature for Marshall-Station (Average Diurnal Cycle)

well, however, the RMSE didn't as the ERA5 data is impacted drastically by the low-bias. **??** makes clear, that the improvement in RMSE and correlation coefficient of the daily analysis was not caused by potential issues in the model to predict the diurnal cycle correctly but by noise reduction in general. In **??** the average diurnal cycle of the reconstructed temperature is plotted against the measured temperature and the temperature series of ERA5. This means that over all the days in 2021, for each of the 24 hours, an average temperature was calculated and plotted for the three series. As all measurements, have been recorded in universal time code (UTC), which is also the time in ERA5, the x-axis is also using UTC, which is apart from the daylight savings time 6 hours ahead of Colorado. It can be observed that ERA5 has a higher amplitude in the diurnal cycle than the weather station and that the model is capable of correcting that.
It is not surprising, that the RMSE and correlation coefficient improve the more the data is aggregated, so it is also important to look at the hourly values again and go into detail. To allow the details to appear on a plot, the x-axis can be cropped to a shorter period, as in **??**, where the x-axis includes 168 hours, equivalent to 7 days. The chart is chosen to be representative, while some weeks have a better fitting than others. In this case, it can be seen, that even in the measurements of 2021, minor gaps happened. For example

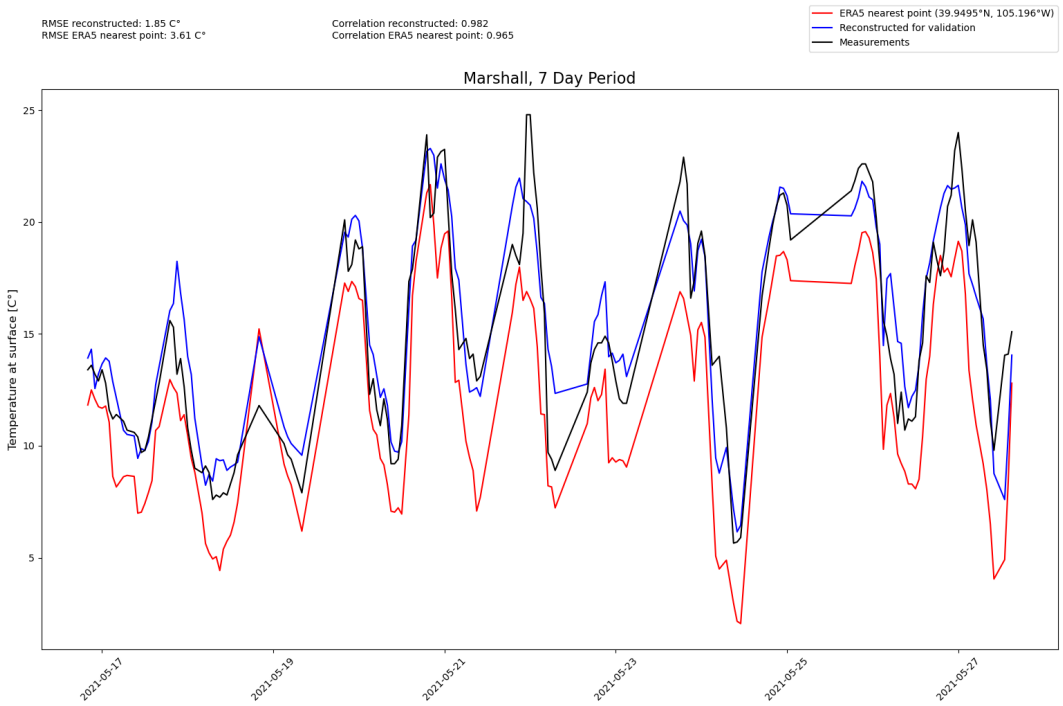


Figure 13: Reconstructed temperature vs measured temperature for Marshall-Station (7 Day Period)

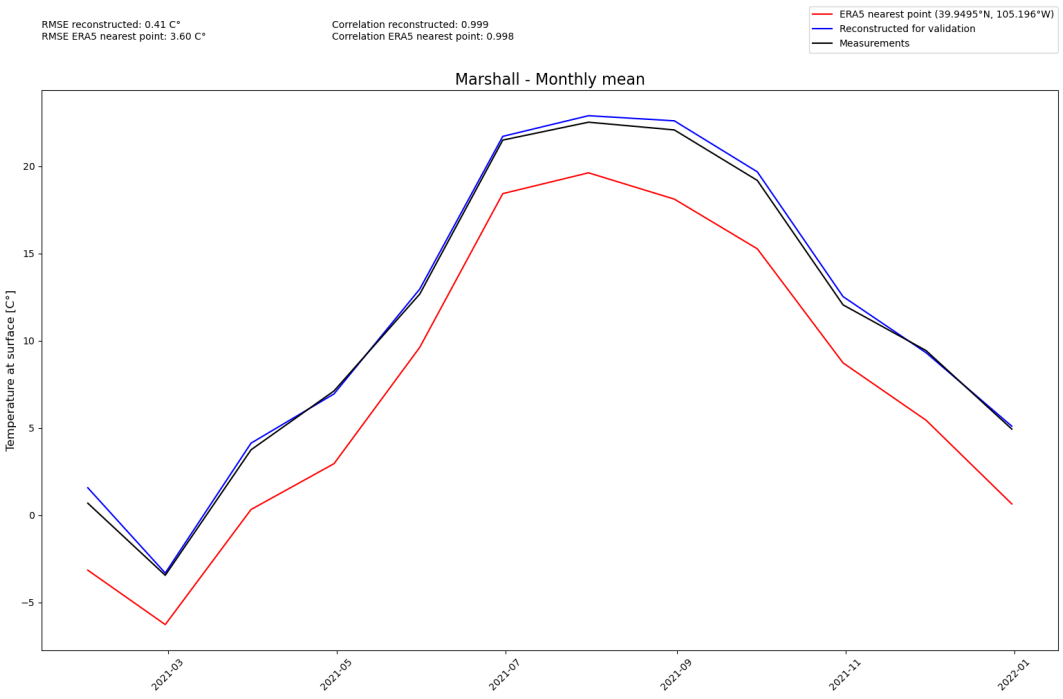


Figure 14: Reconstructed temperature vs measured temperature for Marshall-Station (Monthly mean)

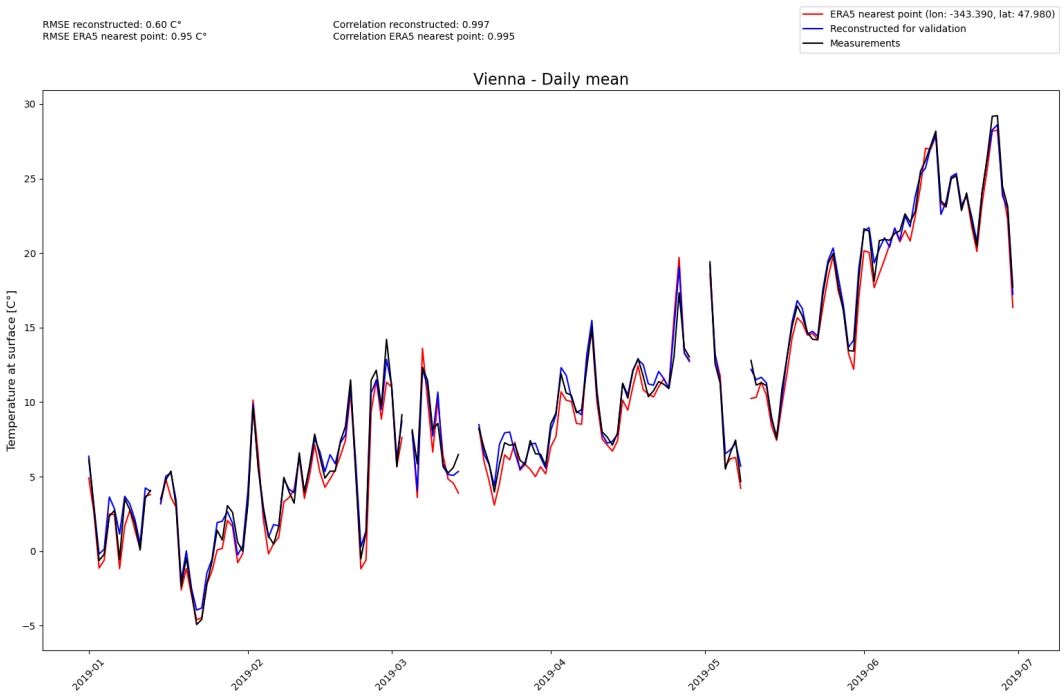


Figure 15: Reconstructed temperature for Vienna-Station (Daily mean)

Vienna Station

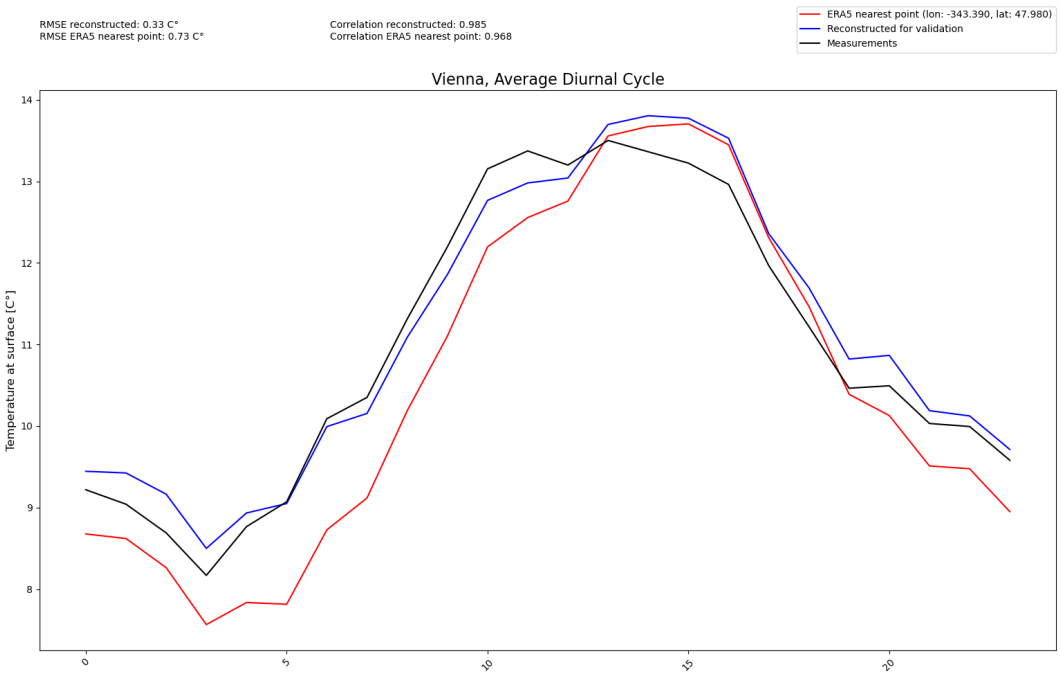


Figure 16: Measured temperature for Vienna-Station (Average Diurnal Cycle)

Barbados Station

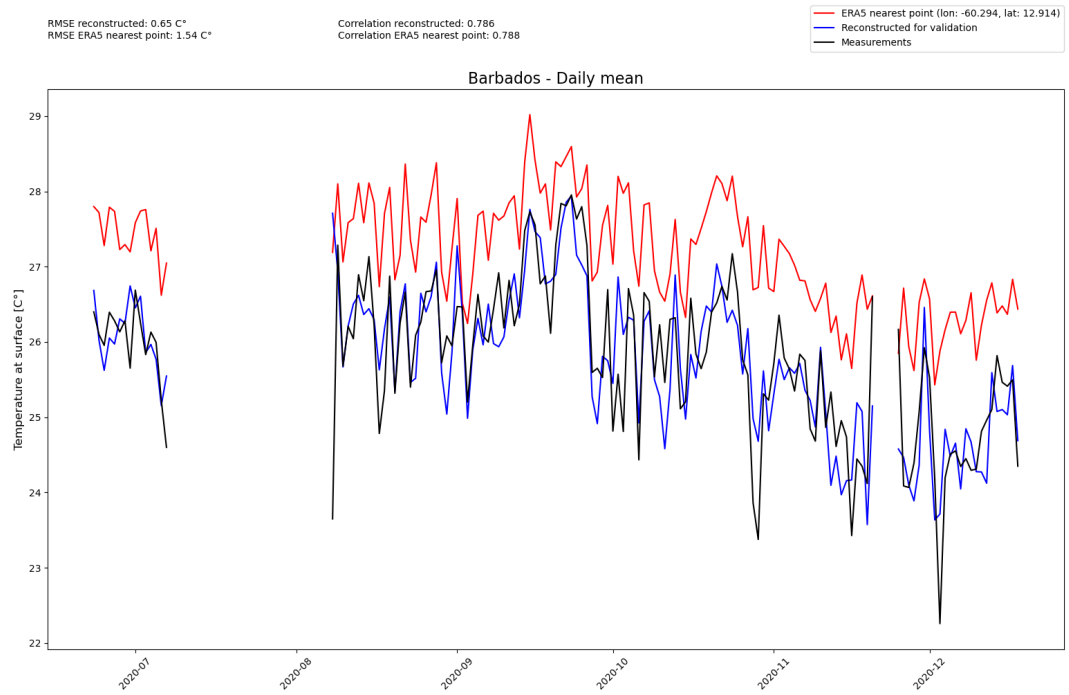


Figure 17: Reconstructed temperature for Barbados-Station (Daily mean)

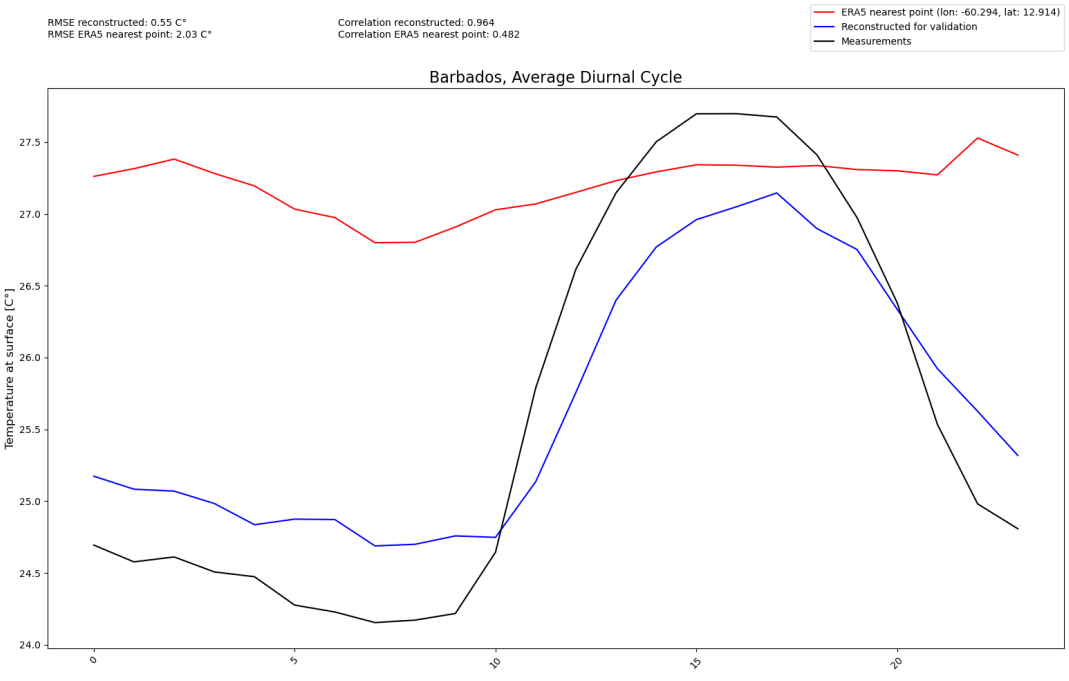


Figure 18: Measured temperature for Barbados-Station (Average Diurnal Cycle)

6 Software Implementation

6.1 Introduction

Given that the machine learning part of the software is modulized as "Climate Reconstruction AI" (CRAI), such that the actual setup of the neural net, training and evaluation afterwards is outsourced. Thus the process for a single specific dataset of one station could be written pretty straightforward with a NetCDF file of the station data ahead, if there is sufficient access to ERA5 Data. A Jupyter notebook could be sufficient as a way to start. However once dealing with different stations, different ERA5 files need to be stored, and the files that have been prepared for submission into CRAI need to have some kind of management and the "training-args" for CRAI need to be adapted each time. Thus it appears natural to implement a set of functions and structure the process through an object-oriented approach. This allows control of different but similar pipelines with a few lines of code, either through a script, initiated by an API or in a Jupyter notebook. This chapter is about laying the foundation through the implementation of the different steps of the pipeline as classes and functions, while in chapter **??** the next abstraction level of the software into classes that handle the execution of the different steps and the interaction with the user through an API and a web interface is described. The following subsections highlight the key steps of the pipeline and how their functionality evolves. It does not cover the detailed embedding of the methods in their classes, nor is explicitly mentioned where temporary folders would be created to establish a philosophy where each step of the pipeline is implemented independently and can be connected in a higher-level class. The methods are not necessarily designed to be used in any possible order and more in the sense of the desired pipelines, but the modularity allows for a better understanding and cleaner code, which is essential when developing another layer of abstraction later.

6.2 Stationdata Submission & Conversion

The station data for the 3D printed weather stations provided by NCAR comes in delimited text files, with one file per day and a text-based metadata file holding information like the station name, the latitude and longitude and the elevation. The data files (.dat files) hold per sensor one column and per minute one row.
A class "DatToNcConverter" is implemented with the following main use cases: First to take in a directory with the .dat files and the metadata file and parse it. Second to process the data including dropping missing values as well as resampling to an hourly frequency, and converting from Celcius to Kelvin to match ERA5. This is easiest in a

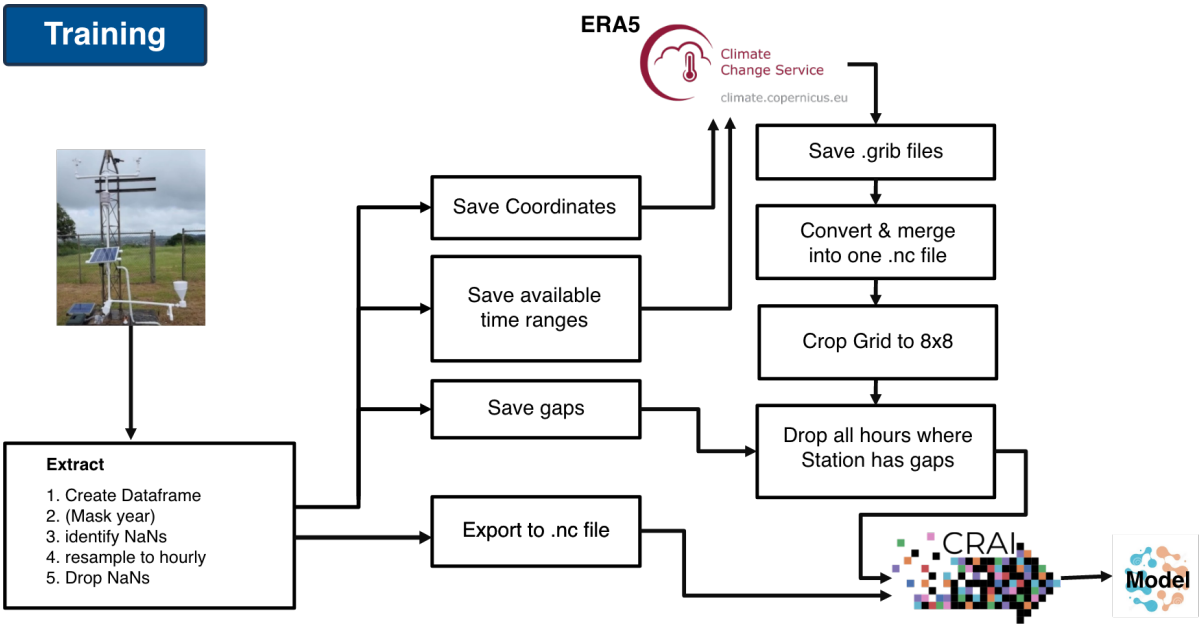


Figure 19: Pipeline to train a model

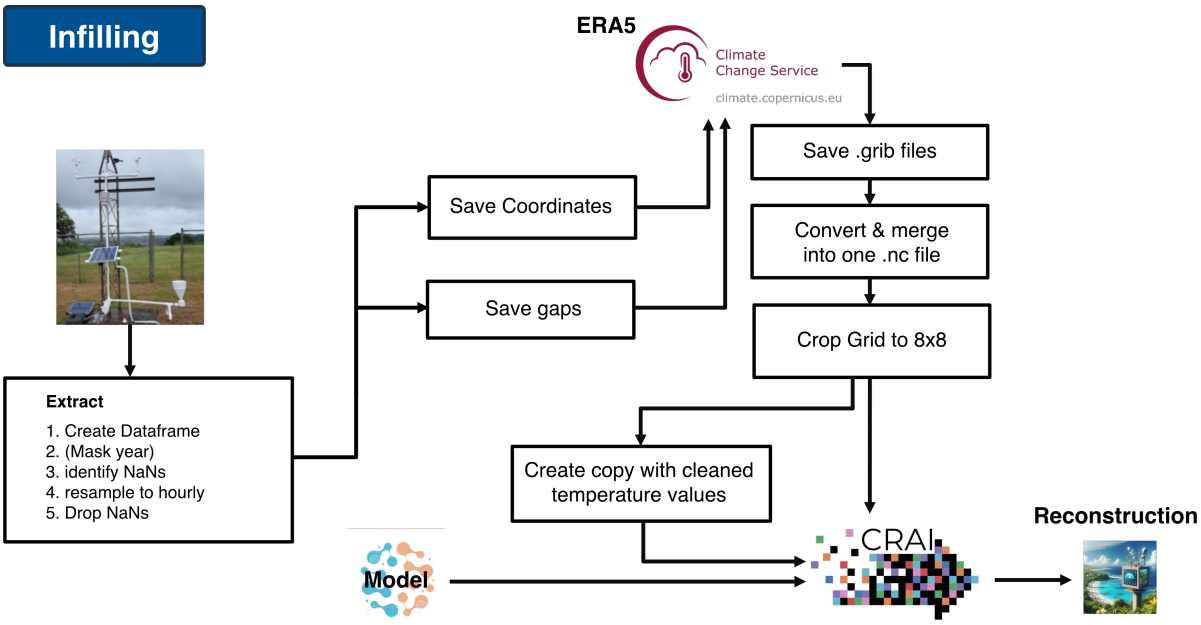


Figure 20: Pipeline to reconstruct weather data using a model

pandas dataframe. Third to convert the data to a NetCDF file, which is the format that is used by the ERA5 data and the CRAI module. The NetCDF file is structured in a way that the data is stored in a 3D array with the dimensions time, latitude and longitude. The metadata is stored as global attributes in the NetCDF file. Fourth to convert some dataframe back to a .dat file after the reconstruction of measurements to match the original format when infilling. Of these use cases the one resampling to hourly frequency is the most complex, or at least could include many design decisions. Missing values are marked with "-999.99", so in the first step, these values can be marked as NaN. However, the data quality is by default not controlled and there could be values that should be marked as missing but are not. Thus the NCAR consulted to mark "0.00" °C as NaN. For stations like Barbados this wouldn't be a realistic value anyway, but even for regions where 0°C degrees are reached often, it's unlikely to measure exactly "0.00", meaning the amount of correct data lost through marking "0.00" as NaN is still limited. Also by agreement with the NCAR everything above or under +/- 45°C is marked as NaN. Additionally to compensate for peaks in the data aggregating the minutes using a median instead of a mean can be a good idea, numpy.median() would return NaN if any value is NaN and numpy.nanmedian() would ignore NaNs. It would be best to have a custom aggregate function using numpy.nanmedian() but assuring prior to that that there are sufficient non-NaN values.
For the usecase of converting the data back to a .dat file, the dataframe is stored directly in the converter object as original dataframe after detecting NaNs and resampling to hourly values, before any transformation of units (Celcius to Kelvin) or renaming of columns takes place and before NaNs are dropped. Fundamentally inbetween the first and last measurements rows for all minutes exist, even if measurements are missing. However if the station had severe issues it's possible that between first and last record there are even rows or files missing. To assure that the original dataframe will have rows for all hours, a handy method provided by the python pandas module is used:

```
1     self.original_df = self.original_df.reindex(
2         pd.date_range(start = self.dataframe.index.min(), end = self.↩
              dataframe.index.max(), freq = "h"))
```

A class "Station" is implemented to hold the metadata and the pandas dataframe of the station data. It holds the converter itself, to minimize lines of code in the main script. The class is mainly used to manage the access to the different files and data, before and after the conversion. And secondly to detect the gaps in the data, for infilling simply in the form of listing the hours where data is missing. And for training use cases additionally in

form of listing the months where at least some data is available which is most convenient for the API applicance to get ERA5 data for the station.

```
def find_gaps(self) -> None:
available_hour_steps = self.df.index
all_hour_steps = self.converter.original_df.index
# find all hours between the first and last hour that are missing
missing_hours = all_hour_steps.difference(available_hour_steps)
return missing_hours.tolist()
```

Code Snippet 1: Gap Detection in Station Class

```
def get_all_months_in_df(self) -> None:
# return all (year, month) tuples in the dataframe
periods = self.df.index.to_period('M').unique().tolist()
month_dict = {}
for period in periods:
    if period.year not in month_dict:
        month_dict[period.year] = []
    month_dict[period.year].append(period.month)
return month_dict
```

Code Snippet 2: Detection of available ranges in Station Class

6.3 Copernicus Climate Data Store - CDS API

The Copernicus Climate Data Store (CDS) is a service by the European Centre for Medium-Range Weather Forecasts (ECMWF). The CDS API provides opensource access to the ERA5 data, allowing users to download the data for a specific location and time period. After creating an account online an API Key can be obtained for free and with the python module 'cdsapi' data can be downloaded then easily in .grib format.

```
import cdsapi

class Era5DownloadHook:

    def __init__(self, lat, lon):
        self.cds = cdsapi.Client(
            url="https://cds.climate.copernicus.eu/api/v2",
            key=f"{os.getenv('UID')}:{os.getenv('API_KEY')}"
```

```
        )
        self.lon, self.lat = lon, lat

    def _download(cds, date_info, save_to_file_path):
        cds.retrieve(
        'reanalysis-era5-single-levels',
        {
            "product_type": "reanalysis",
            "format": "grib",
            "variable": "2m_temperature",
            "area": [
                self.lat + 1, # limit north
                self.lon - 1 % 360, # limit west
                self.lat - 1, # limit south
                self.lon + 1 % 360, # limit east
            ],
            "year": date_info.get("years"),
            "month": [f"{month:02d}" for month in date_info.get("months")],
            "day": [f"{day:02d}" for day in date_info.get("days")],
            "time": [f"{hour:02d}:00" for hour in date_info.get("hours")]

        }, save_to_file_path)
```

Code Snippet 3: Download Hook for ERA5 Data

As seen in the code snippet, the request needs to include besides basic informations such as variable / format / model, the regional selection and the selection in the time dimension. to download a few hours in a day the following can be used. When building requests that download a month or a full year, it becomes obvious why the **??** is so useful.

```
    def download_hours_in_same_day(self, year, month, day, hours, ↩
        target_folder):
        self.download({
            "years": [year],
            "months": [month],
            "days": [day],
            "hours": hours
        }, f"{year}_{month}_{day}.grib")
```

Code Snippet 4: Download Hours in Same Day in Download Hook Class

In the ?? it can be seen that the regional selection is always +/- 1 degree around the Station location. This ensures because of the grid interval of 0.25° that the downloaded area always includes at least 9x9 grid points, such that when cropping to an 8x8 selection the station can always be centered, even without exact knowledge of the coordinates of the desired grid points prior requesting.

6.4 Data Preprocessing

As pointed out in ?? the data is downloaded in .grib format. Using the program 'cdo' the data can be quickly converted to .nc format. With the following simple command

```
cdo -f nc copy {source_path} {nc_path}
```

However to have the temperature at surface variable name unified as "tas" in the NetCDF file, the following command can be used:

```
    import xarray as xr
    import subprocess

    def _rename_variable(self, var_name, tas_name, input, output):
        rename_variable_command = \
            f"cdo chname,{var_name},{tas_name} {input_path} {output_path}"
        subprocess.run(rename_variable_command, shell=True)

    ds = xr.open_dataset(input_path)
    if 'var167' in ds.variables:
        _rename_variable('var167', 'tas', input_path, output_path)
    elif '2t' in ds.variables:
        _rename_variable('2t', 'tas', input_path, output_path)
```

Code Snippet 5: Renaming Variable in NetCDF File

Then the files are merged using

```
    cdo cat {temp_dir_path}/*.nc {era5_target_file_path}
```

before the data is cropped to the 8x8 grid around the station location, as described in ??. For training the data it needs to be assured that the ERA5 data does not include timesteps that the nc file of the station data does not include so that the time dimensions are identical. This is done in two steps, first the ERA5 data is cropped using a start / end date approach using the first and last date of the station data. Then all hours that were missing in the

station dataset, identified by the find gaps method in **??**, are removed from the ERA5 data. This is done using the python module xarray and deleting the timesteps in batches of up to 1000 timesteps, as deleting all at once could cause issues.
The station data should have the same dimensions as the ERA5 data, so a method is applied that coppies the ERA5 dataset and replaces all the temperature values with the measured value from the corresponding hour in the station data.
For evaluating the model, in a validation procedure the same preparation of ERA5 data is done, however instead of passing the station data as expected output to the model, the file will not be filled with the measured values but with NaNs.
When evaluating the model not for validation but to infill missing values, the ERA5 data needs to be prepared differently of course. Instead of requesting monthly data from the cdsapi, it is requesting only the missing hours day by day, such that the time-dimesion is directly as desired, and preprocessing only needs to handle the conversion and geographical cropping.

6.5 Output of the model

Because CRAI is using the encode, decode architecture with connected layers as described in **??**, the output file in NetCDF format that the model produces includes not only one temperature value per timestep but has the same dimensions as the input ERA5 file meaning it uses the same 8x8 grid with 64 temperature values. However since it has been trained on input files where the ground truth was also laid on the 8x8 grid, the 64 temperature values in the output data of the model tend to be extremely similar. They need to be reshaped however to the original dimensions of the station data, which is a 1D array with 1 temperature value per timestep. This is done by taking the mean of the 64 values.

6.6 Plotting for validation

6.7 Infilling for reconstruction

7 Process Orchestration + API & Webinterface

7.1 Executer Classes: Training, Infilling, Validation

The challenge for the foretold Web interface and API solution is to store different station data, independently and to dynamically control processes for each station. The root of the

executer classes is to control the needed processes for Training, Infilling and Validation. Thus begins with passing the corresponding Station object to the executor class and for the process dedicated temporary folders are created, etc. The executors are designed as classes to allow for easy access to the different data of the process entity itself at all steps, from ERA5 download to plotting.
The Training Executer class is structured such that all needed inputs are directly passed on object creation, but the pipeline steps are initiated not from the class constructor but from an extra method called "execute" which allows for a "similar execute_with_sbatch" version which is useful when using the code stackon a super computer like levante. The execution, first runs the download steps for the ERA5 data which is a combination of a few routines described in **??**: Identifying the available timesteps, downloading the data, merging the .grib files, converting to NetCDF files and cropping in the space and time dimensions as described above. This is where the benefit of the work in the previous chapter comes into play, as all of that is controlled by just a few lines and even undisturbed of temp-folder management which is directly included in the routines from **??**. After completion of the preprocessing steps, when the necessary coherent training files are saved in the temp-folders of the executer object, it generates the training-args.txt which is used as the input parameters for the separate machine learning code stack. That file not only specifies how exactly the training should be executed and for how many iterations but also includes information about where there output, respectively the model files should be saved. As this is specified by one of the executer-objects properties, it is troublefree to handle the saving of the model wherever needed.
The Evaluation Executer used for the infilling routine is structured similarly to the Training Executer, but with the difference that the input parameters must include a model path, and that the download routines to obtain ERA5 data use the times where measurements are missing, instead of the times where measurements are available. The class has an execute function as well which runs the different steps after each other, but can be manipulated if certain steps such as the download of the ERA5 data should be run differently. The arguments .txt file for the CRAI module is generated similarly but of course with the parameters needed for evaluation.
The Validation Executer is structured similarly to the Evaluation Executer, but has the download routine of the Training Executor, to obtain the ERA5 data for the times when measurements are available. Also after the successful evaluation of the model, the plots for the comparison of the predictions with the actual measurements are generated and saved in the temp-folder of the executer object. The plots also are generated comfortably with the plotting routine from **??**.

7.2 Webinterface

As written in the Introduction of this chapter **??**, the benefit of establishing the abstraction layer with the executor classes is that with minimal additional effort, API Endpoints for initiation of processes to train/validate a model as well as endpoints to retrieve the results after the processes have finished. Additionally, it is possible to monitor the progress of the process through the API. The details on the API are described in **??** This lays out a sufficient foundation for the implementation of a web interface, which is the main focus of this section. The interface consists of two areas, one where the user is supposed to submit a "dataset", which in this context is meant as a collection of measurements from a weather station and the station's information such as location, and optionally a custom name and third a trained model for the station that can be provided optionally as well. If no model is provided the system can train a model and will attach it.

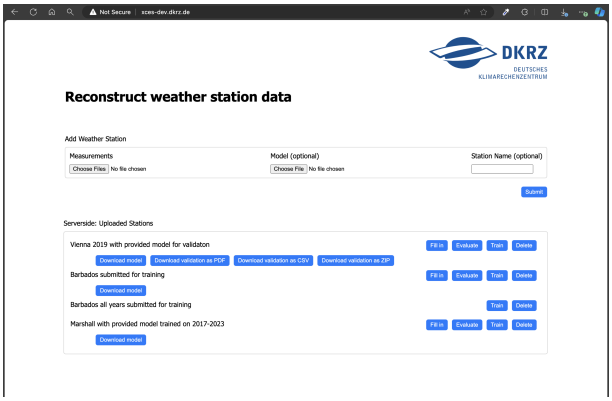


Figure 21: Screenshot of the webinterface

The second area is a list of all datasets that the user has submitted. The API implementation allows for user identification through a unique token that is passed on to the server when the user submits a dataset, such that the user can then ask to see all datasets owned by them using that same token. The web interface automatically generates a token for the user and stores it then in the local storage of the browser, such that the user does not have to remember it.

As examples in **??** show each dataset depending on its state has different options available. The deletion and train buttons all have in common, as no model is necessary to train a model. Datasets where a model has been provided still could be used to train a new model overwriting the old one. The "Evaluate" and "Fill in" buttons to evaluate the model over the timesteps where measurements are available, respectively to evaluate the model over the timesteps where measurements are missing.

Once a process such as training evaluation or infilling has been completed for a dataset the user can download the results anytime through the new buttons that appear in the dataset list. For example, it can be seen in **??** that for the Vienna example, validation has been completed, and the user can now download a PDF with all the plots that compare the predictions with the actual measurements, or a CSV file with the same data, or a zip

file that contains both and the images used in the PDF. The first Barbados example has a model attached that can be downloaded, meaning either it was provided or generated through training on the server itself. The second Barbados example has no model attached yet, meaning only the buttons "Train" and "Delete" are available, because no model was provided and no training has been done yet.

7.3 API Endpoints

POST /data-submission

Stores a new dataset on the server. Each dataset is associated with a unique ID, and a unique token representing the owner. The token of the owner is passed in the request body. The dataset ID is created by the server and returned in the response. The dataset ID is used to refer to the dataset in the following API calls.

GET /available-datasets/<user-token>

For a given user token, this endpoint returns a list of all datasets that the user has submitted. The user token is passed as a URL parameter. The response is a list of dictionaries, each containing information about the dataset, such as the dataset ID, the name, and the status of the dataset (e.g. if it is busy training a model and progress percentage).

GET /train/<data-submission-id>

GET /validate-model/<dataset-id>

GET /fill-in/<dataset-id>

DELETE /delete-dataset/<dataset-id>

GET /download-model/<dataset-id>

GET /download-validation-zip/<dataset-id>

GET /download-infilling/<dataset-id>

8 Discussion

8.1 2 step process

8.2 Use for weather forecasting