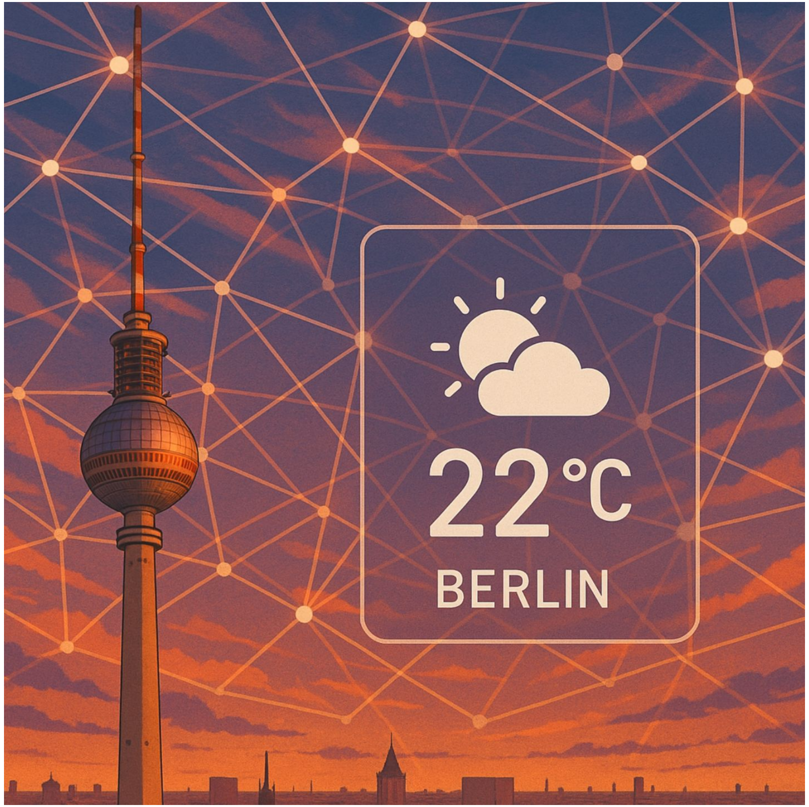
DSW Presentation

By: Nick Chandler, Luisa Kalkert, & Nataliia Remezova

Outline

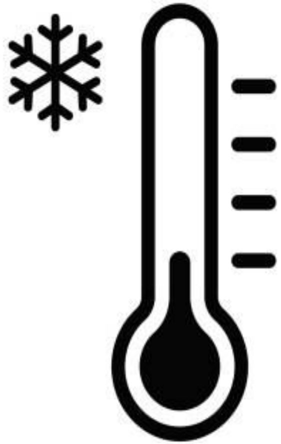
1. Task
2. Dataset
3. Workflow
4. Metrics
5. Methods
    a. Description
    b. Training
    c. Evaluation



Task

Weather forecast: Next day temperature prediction

1. Obtain time series weather data online
2. Visualize and clean the data
3. Apply different models
4. Compare models based on different metrics



Dataset I - What Data?

- We used data from Nick's Friend's BME680 weather sensor he's been running in Germany since 2018!
- It looks like this on the website (https://wx1.slackology.net/data/2024/bme680.dat.20240106):

```
20240106000003  Temp: 1.91 C   Humidity: 92.95 %     Pressure: 99.814 kPa   AirQ: 1312439 Ohms
20240106000101  Temp: 1.92 C   Humidity: 92.96 %     Pressure: 99.816 kPa   AirQ: 1308594 Ohms
20240106000201  Temp: 1.93 C   Humidity: 92.93 %     Pressure: 99.814 kPa   AirQ: 1328048 Ohms
20240106000302  Temp: 1.93 C   Humidity: 92.94 %     Pressure: 99.814 kPa   AirQ: 1313725 Ohms
20240106000401  Temp: 1.94 C   Humidity: 92.94 %     Pressure: 99.812 kPa   AirQ: 1315015 Ohms
20240106000501  Temp: 1.94 C   Humidity: 92.94 %     Pressure: 99.812 kPa   AirQ: 1308594 Ohms
20240106000601  Temp: 1.94 C   Humidity: 92.95 %     Pressure: 99.813 kPa   AirQ: 1318898 Ohms
20240106000702  Temp: 1.95 C   Humidity: 92.92 %     Pressure: 99.819 kPa   AirQ: 1315015 Ohms
20240106000802  Temp: 1.95 C   Humidity: 92.91 %     Pressure: 99.820 kPa   AirQ: 1315015 Ohms
20240106000902  Temp: 1.95 C   Humidity: 92.92 %     Pressure: 99.816 kPa   AirQ: 1321499 Ohms
```

Dataset II - Scraping

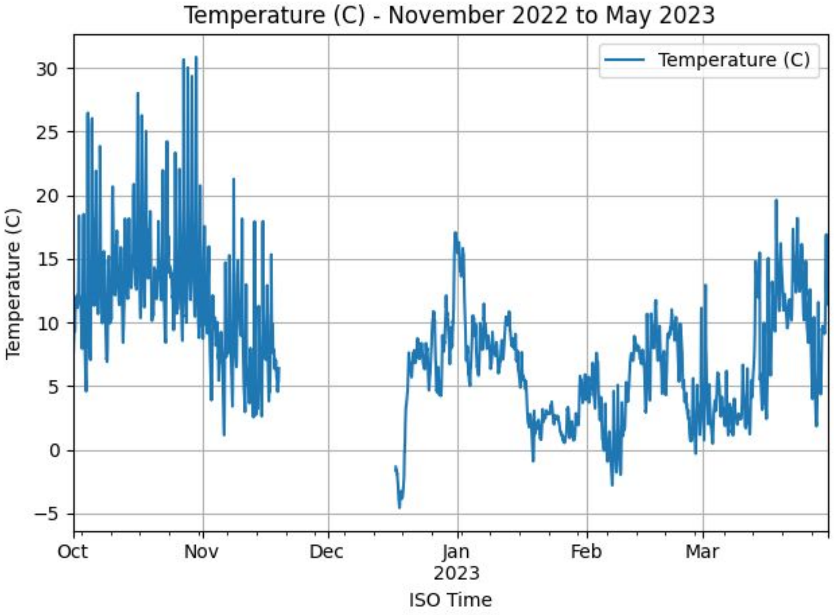
- The data is on a website so we need to scrape it.
- We save to .csv per year of data.

Basic Algorithm:

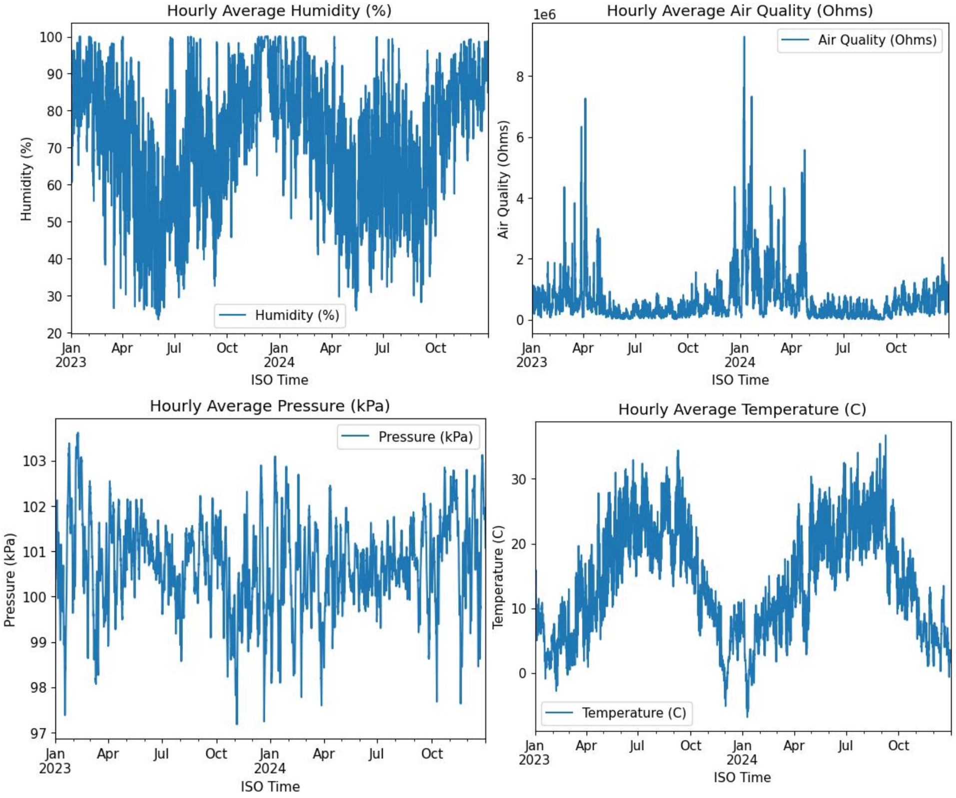
1. Go through each file on the website
2. Fetch webpage content
3. Parse lines & extract data
4. Format data
5. Append to CSV

Dataset III - Cleanup

- There were some anomalies, for example a sudden jump in December of 2022
- It turned out Nick’s friend ran the sensor in Augsburg before moving back to Berlin
- We had enough data on Berlin and that in Augsburg was different enough, so we train on 2023 and 2024 (mostly).
- We do linear interpolation of missing values in the remaining series



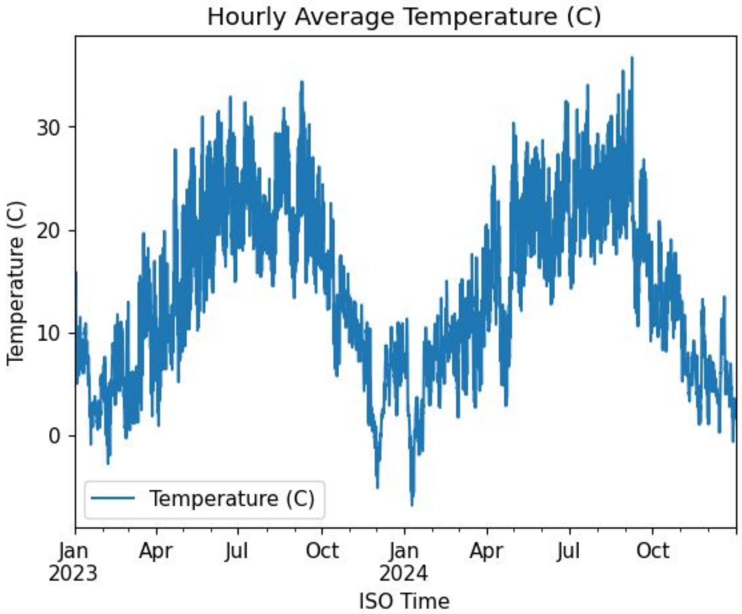
Dataset IV - Visualization



Visualizations of the series

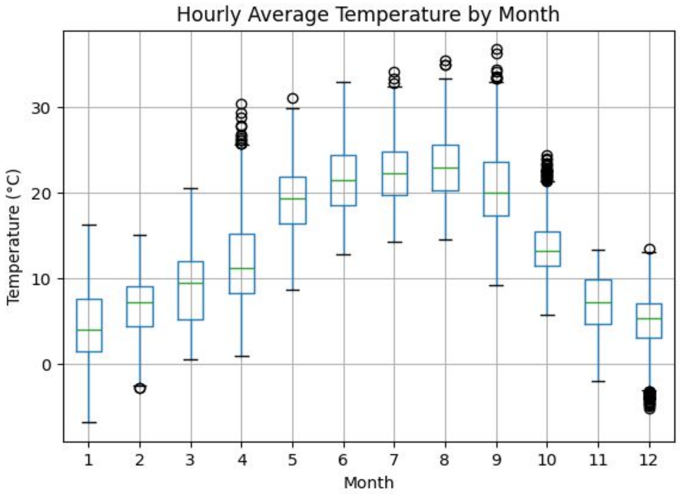
Dataset V - Refinement

- We focus on Temperature in Celsius
- We use 168 hours (1 week) as context to predict the next 24 hours



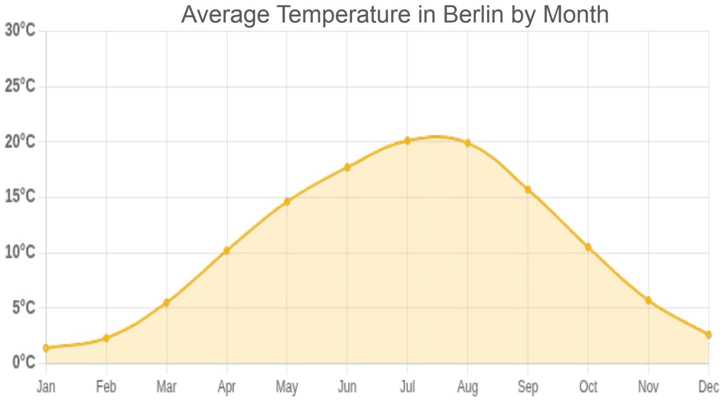
Dataset VI - Sanity Checks

Our Data:



Online Data:

by *World Weather & Climate Information*

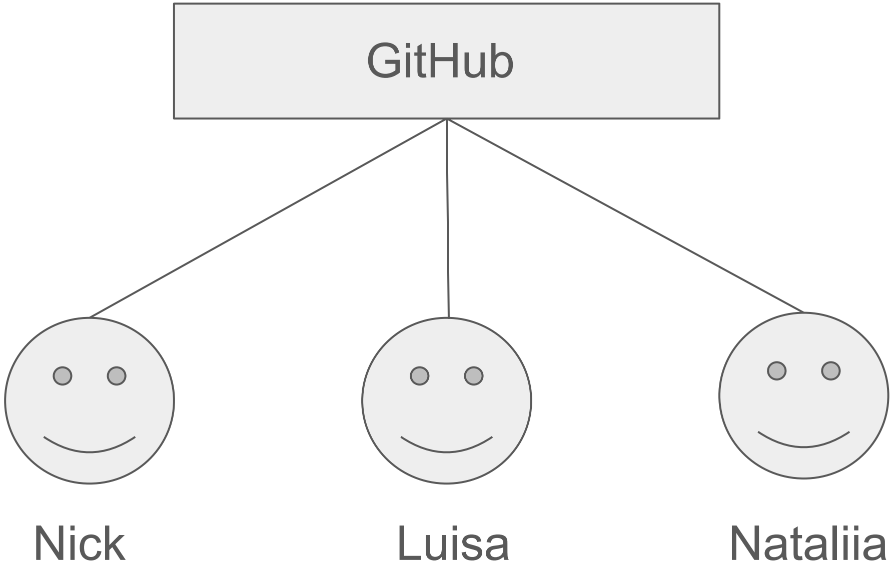


Workflow

GPU Access Process/Collaboration Structure

Steps:

1. Get on BHT Network
2. Scale deployment
3. Port forward
4. Remote SSH in VSCode



Metrics

Metrics I - RMSE

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}},$$

Source: https://c3.ai/glossary/data-science/root-mean-square-error-rmse/

- Absolute measure of error
- Measures the average difference between actual and predicted values
- Standard deviation of the residuals
- Highly used metric

```python
def root_mean_squared_error(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

Metrics II - MAPE

Source: https://www.google.com/search?q=mape

$$M = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

M = mean absolute percentage error

n = number of times the summation iteration happens

A_t = actual value

F_t = forecast value

- Relative measure of error
- Average percentage error between actual and predicted error
- Commonly used in forecasting

```
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    non_zero = y_true != 0
    return np.mean(np.abs((y_true[non_zero] - y_pred[non_zero]) / y_true[non_zero])) * 100
```

Metrics III - MASE

For a non-seasonal time series,[8] the mean absolute scaled error is estimated by

$$\mathrm{MASE} = \mathrm{mean}\left(\frac{|e_j|}{\frac{1}{T-1}\sum_{t=2}^{T}|Y_t - Y_{t-1}|}\right) = \frac{\frac{1}{J}\sum_j |e_j|}{\frac{1}{T-1}\sum_{t=2}^{T}|Y_t - Y_{t-1}|}$$ [3]

where the numerator e_j is the forecast error for a given period (with J, the number of forecasts), defined as the actual value (Y_j) minus the forecast value (F_j) for that period: $e_j = Y_j - F_j$, and the denominator is the mean absolute error of the one-step "naive forecast method" on the training set (here defined as $t = 1..T$),[8] which uses the actual value from the prior period as the forecast: $F_t = Y_{t-1}$[9]

Source: https://en.wikipedia.org/wiki/Mean_absolute_scaled_error

- Absolute measure of error
- Mean error of the model divided by the mean error of a simple baseline (previous temperature value)
- Widely used in forecasting

```python
def mean_absolute_scaled_error(y_true, y_pred, insample):
    """ MASE using naive forecast as denominator (seasonality=1 assumed)"""
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    insample = np.array(insample)
    naive_forecast = np.abs(insample[1:] - insample[:-1])
    denom = np.mean(naive_forecast)
    return np.mean(np.abs(y_true - y_pred)) / denom
```

Methods

- Baseline (Arithmetic Mean)
- ARIMA
- XGBoost
- LSTM
- LSTM-CNN
- Lag-Llama (Time series foundation model)

Baseline I - Description

$$\bar{x}$$

Baseline II - Training

"Train" the model

```
model = np.mean(train_data)
model
```

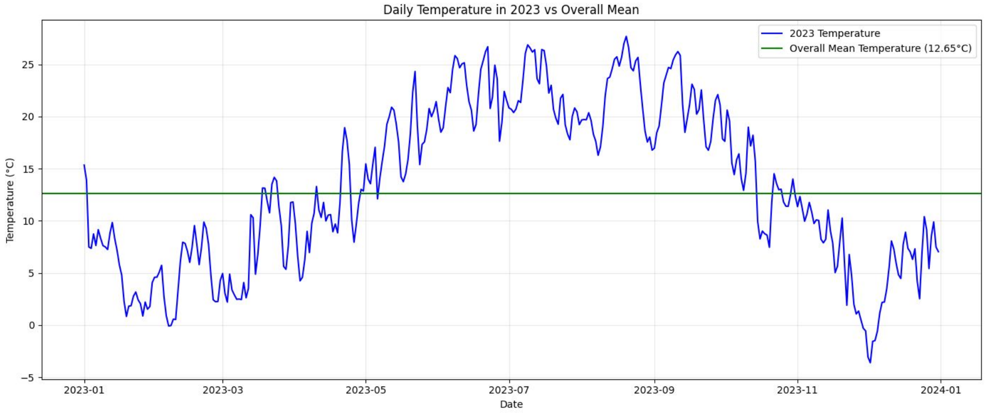
Baseline III - Evaluation

Compute Metrics

```
y_pred = model * np.ones_like(test_data)

print("MAPE:", mean_absolute_percentage_error(test_data, y_pred))
print("RMSE:", root_mean_squared_error(test_data, y_pred))
print("MASE:", mean_absolute_scaled_error(test_data, y_pred, train_data))
```

```
MAPE: 866.755485534668
RMSE: 7.522595002882488
MASE: 14.646165401799582
```



ARIMA

How ARIMA Works

- AR: Predicts via weighted past values
- I: Differences to remove trend
- MA: Smooths via past errors
- (p,d,q):
  - how many past hours to use (p),
  - how many differences to take (d),
  - how many past errors to include (q)
  - via sliding-window CV
- A single set of coefficients —no daily or weekly cycles

Why It Falls Short

- Linear only -> misses curves & jumps
- Seasonality is missing - no SARIMA
- Struggles with drifts and needs stationarity
- Sensitive to gaps & outliers
- Many (p,d,q) settings fail

ARIMA

Pmdarima best result

```
MAPE: 848.1057025729665
RMSE: 7.30677394922854
MASE: 14.17915596284864
```

Darts ARIMA best result

```
MAPE: 1078.7194527365307
RMSE: 7.472889223630581
MASE: 14.437269589545311
```

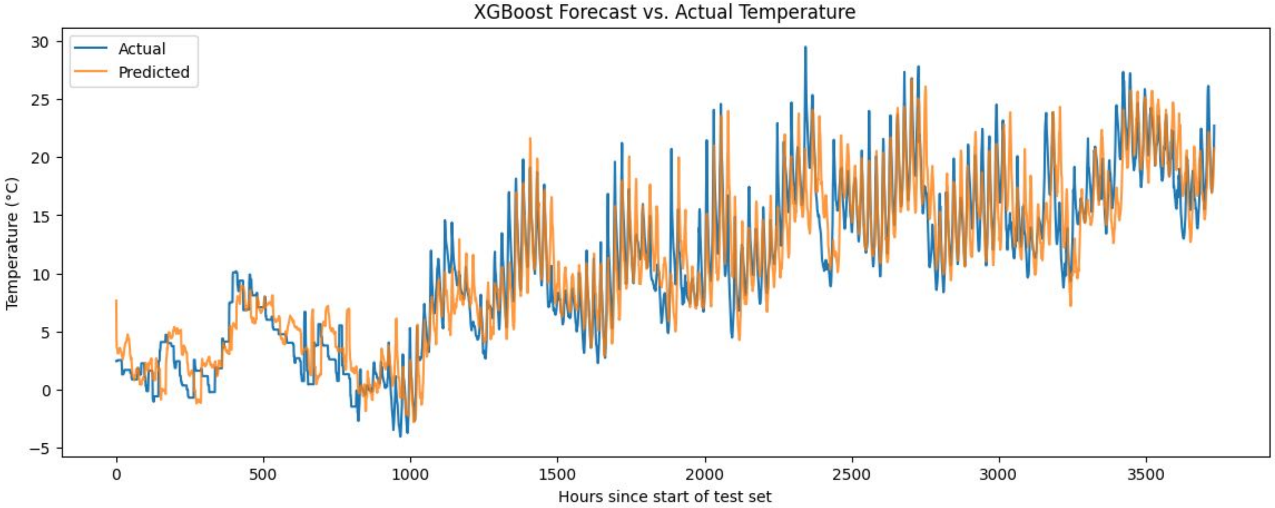
XGBoost I - Description

- 168 lag features capture daily & weekly cycles
- Boosted trees iteratively correct errors
- Nonlinear splits model thresholds & abrupt shifts
- Robust to gaps & noisy readings
- No stationarity requirement—learns any patterns
- Optuna-tuned for minimal RMSE

XGBoost II - Evaluation

Best result

```
MAPE: 108.75%
RMSE: 2.724
MASE: 4.910
```



LSTM I - Description

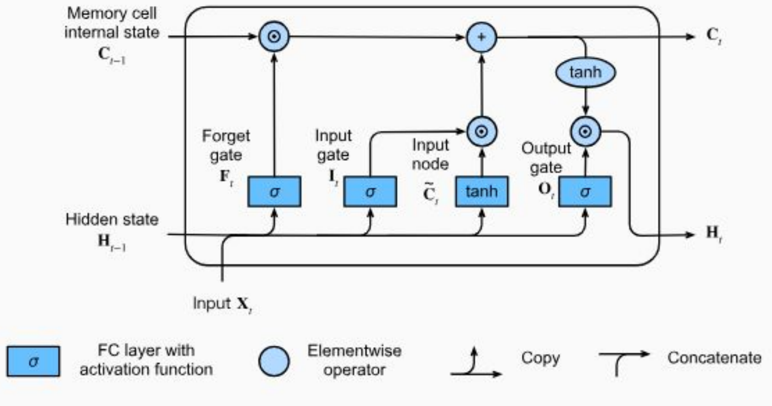


Fig. 10.1.4 Computing the hidden state in an LSTM model.

```python
# LSTM Model
class LSTMModel(nn.Module):
    def __init__(self, input_size=1, hidden_size=50, num_layers=1, output_size=24):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.linear = nn.Linear(hidden_size, output_size)  # output 24 values at once
        self.dropout = nn.Dropout(p=0.2)

    def forward(self, x):
        out, _ = self.lstm(x)             # out shape: (batch, seq_len, hidden_size)
        out = self.dropout(out[:, -1, :])  # take output from last time step only
        out = self.linear(out)            # map to 24 outputs
        out = out.unsqueeze(2)            # reshape to (batch, 24, 1)
        return out
```

Source: https://d2l.ai/chapter_recurrent-modern/lstm.html

LSTM II - Training

Val Metrics and Best Hyperparameters

```
Best trial:
  MAPE: 40.1271
  RMSE: 2.8159
  MASE: 4.3858
  Params:
    hidden_size: 65
    num_layers: 1
    lr: 0.008513973533737033
    wt_decay: 4.954756887512563 4e-05
    batch_size: 64
```

Important Details:

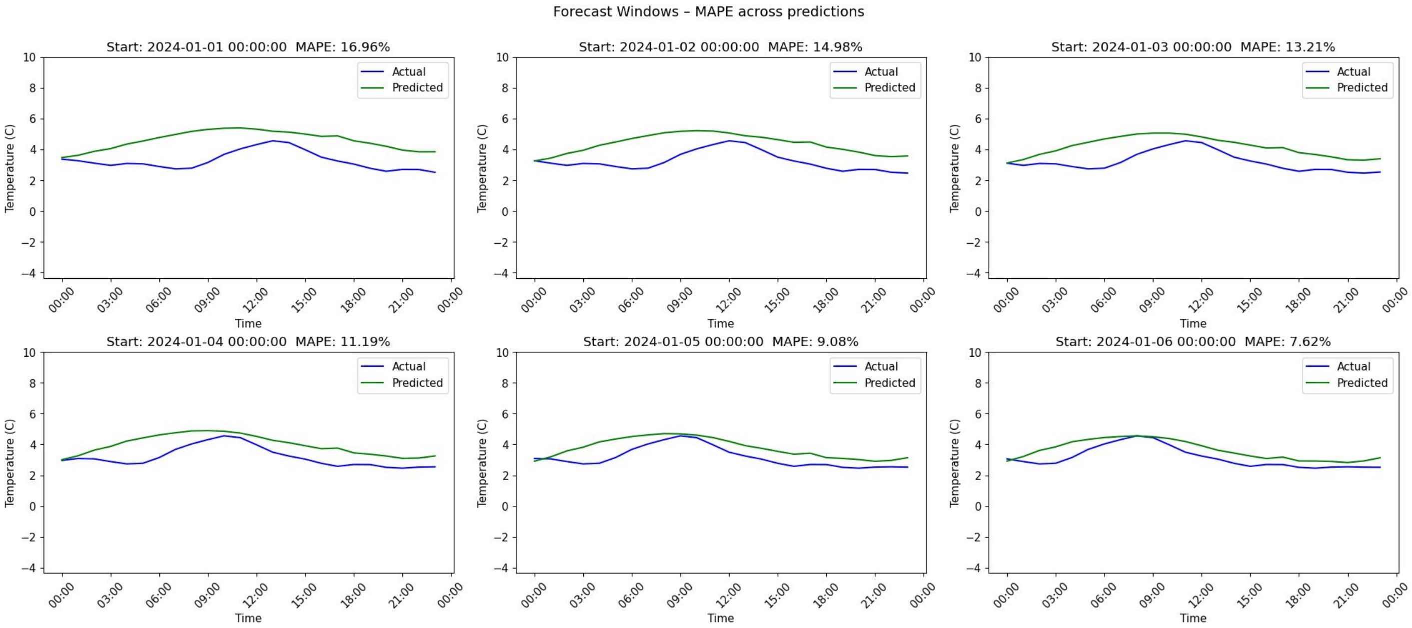
- 10 runs of CV-HPO
- Minimizing MAPE
- Using optuna

LSTM III - Evaluation

Test Metrics

```
RMSE: 2.2229
MASE: 3.2903
MAPE: 57.61%
```

Trained the model on all data used for CV



Sample Predictions

LSTM-CNN I - Description

1D Convolution + Pooling

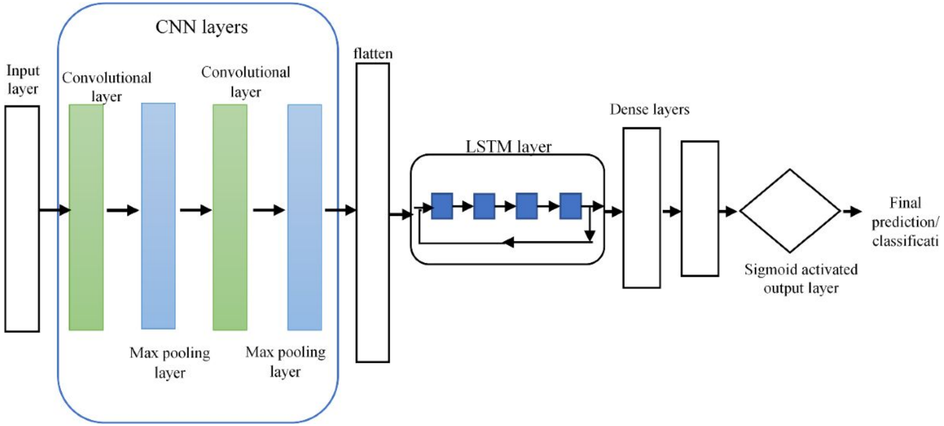
- 1D CNN on 168-h input
- Filters pick up local patterns (daily cycles, spikes)
- MaxPool halves time-steps

LSTM on CNN Features

- Input = learned feature vectors
- Sequence length ≈ 168/2
- fewer time steps—the LSTM works on a richer sequence

Same Final Layer & Output

- Dropout for regularization
- Linear layer → 24-h forecast
- Reshape to (24 × 1)

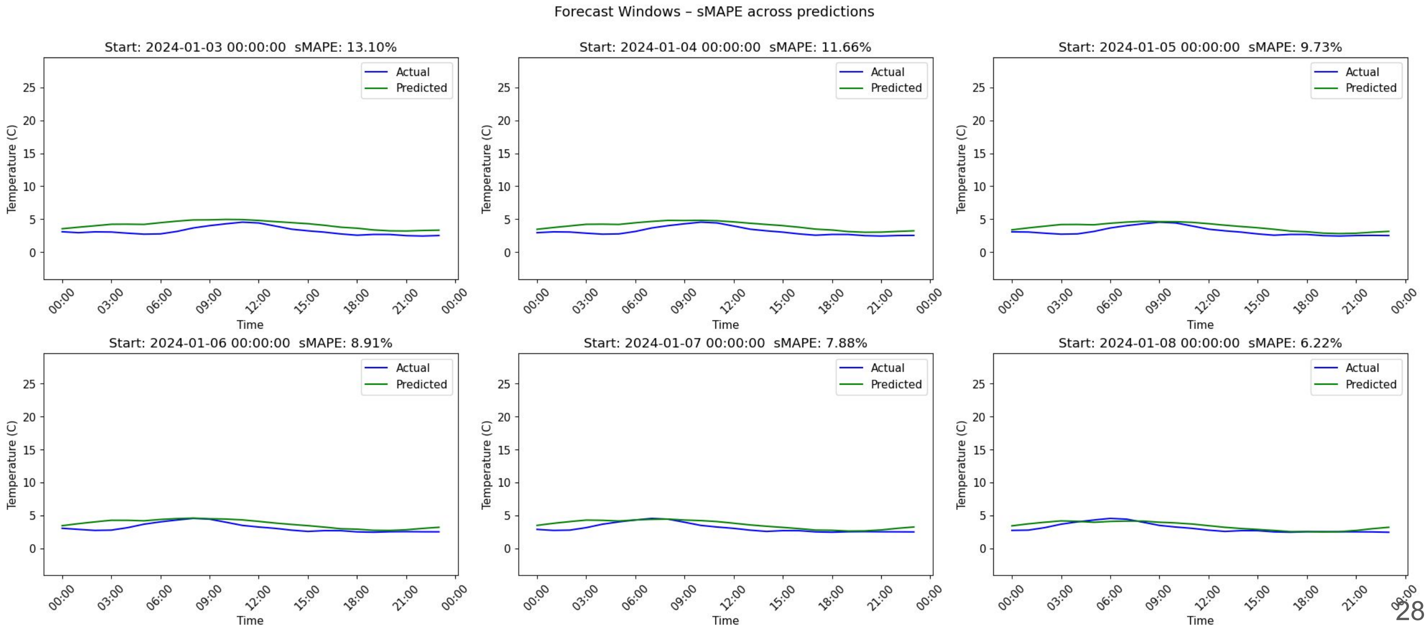


Source: https://link.springer.com/article/10.1007/s11063-024-11687-w

LSTM-CNN II - Evaluation

RMSE: 2.3659
MAE: 3.4284
MAPE: 63.32%

The results are comparable to the original LSTM-HPO, but are not getting better by adding CNN



LSTM-CNN III - Results

- More parameters → overfitting risk
- Pooling may discard fine-grained signals
- CNN filters not guaranteed to match relevant patterns
- Extra hyper-parameters → heavier tuning

Lag-Llama I - Description

Lag-Llama: Towards Foundation Models for Probabilistic Time Series Forecasting

Abstract

Over the past years, foundation models have caused a paradigm shift in machine learning due to their unprecedented capabilities for zero-shot and few-shot generalization. However, despite the success of foundation models in modalities such as natural language processing and computer vision, the development of foundation models for time series forecasting has lagged behind. We present Lag-Llama, a general-purpose foundation model for univariate probabilistic time series forecasting based on a decoder-only transformer architecture that uses lags as covariates. Lag-Llama is pretrained on a large corpus of diverse time series data from several domains, and demonstrates strong zero-shot generalization capabilities compared to a wide range of forecasting models on downstream datasets across domains. Moreover, when fine-tuned on relatively small fractions of such previously unseen datasets, Lag-Llama achieves state-of-the-art performance, outperforming prior deep learning approaches, emerging as the best general-purpose model on average. Lag-Llama serves as a strong contender to the current state-of-art in time series forecasting and paves the way for future advancements in foundation models tailored to time series data.

TLDR:

- Foundation model for time-series
- Uses a decoder-only transformer architecture
- Supposedly decent zero-shot performance

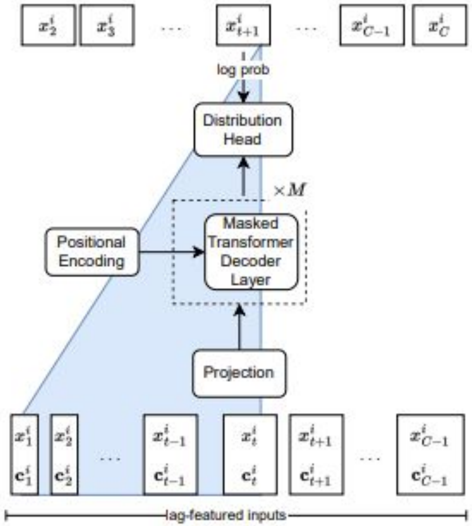


Figure 2: The Lag-Llama architecture. Lag-Llama learns to output a distribution over the values of the next time step based on lagged input features. The input to the model is the token of a univariate time series i at a given timestep, $\mathbf{x}_t^i$, constructed as described in Sec.4.1. Here, we use $\mathbf{c}_t^i$ to refer to all additional covariates used along with the value at a timestep t, which include the $|\mathcal{L}|$ lags, F date-time features, and summary statistics. The inputs are projected through M masked decoder layers. The features are then passed through the distribution head and trained to predict the parameters of the forecast distribution of the next timestep.

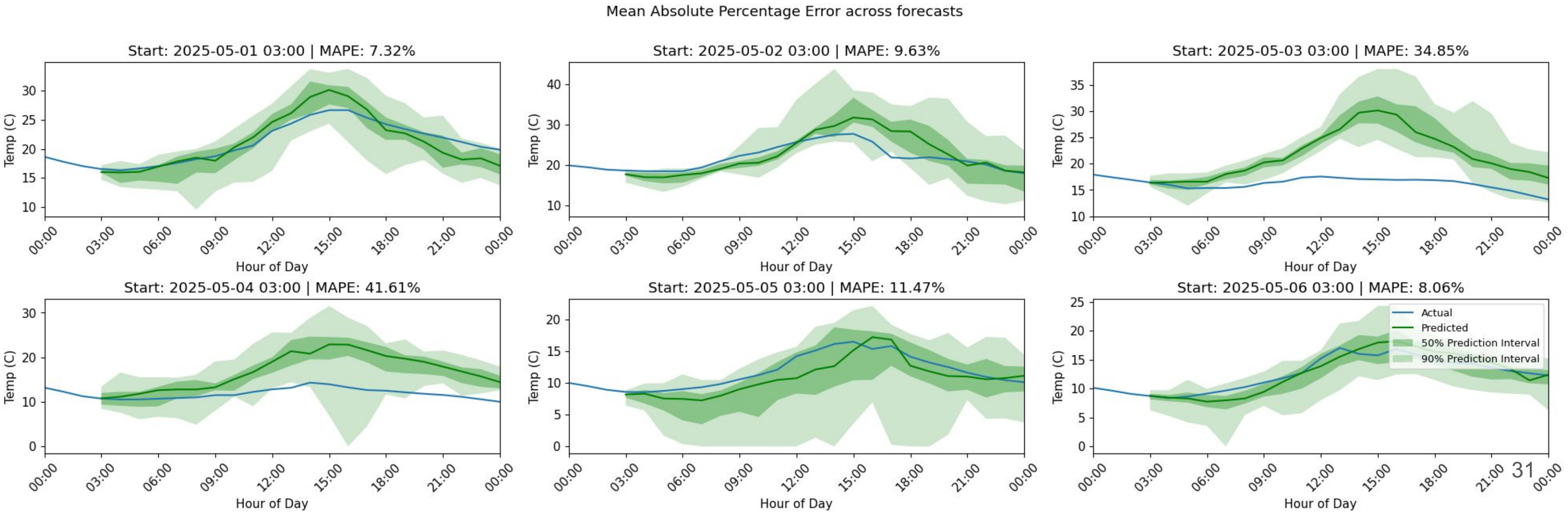
Lag-Llama II - Zero-Shot Evaluation

Test Metrics

```
RMSE: 2.4306
MASE: 1.3988
MAPE: 73.81%
```

Comparable to LSTM, lower MASE, higher MAPE, RMSE

Sample Predictions



Lag-Llama III - Fine Tuning

- Added augmentations (jitter/noise), easily add-able through Lag-LLama package

- Hyperparameter Optimization for learning rate only
- Cross validation, select for best RMSE
- Run on BHT cluster, V100 GPU

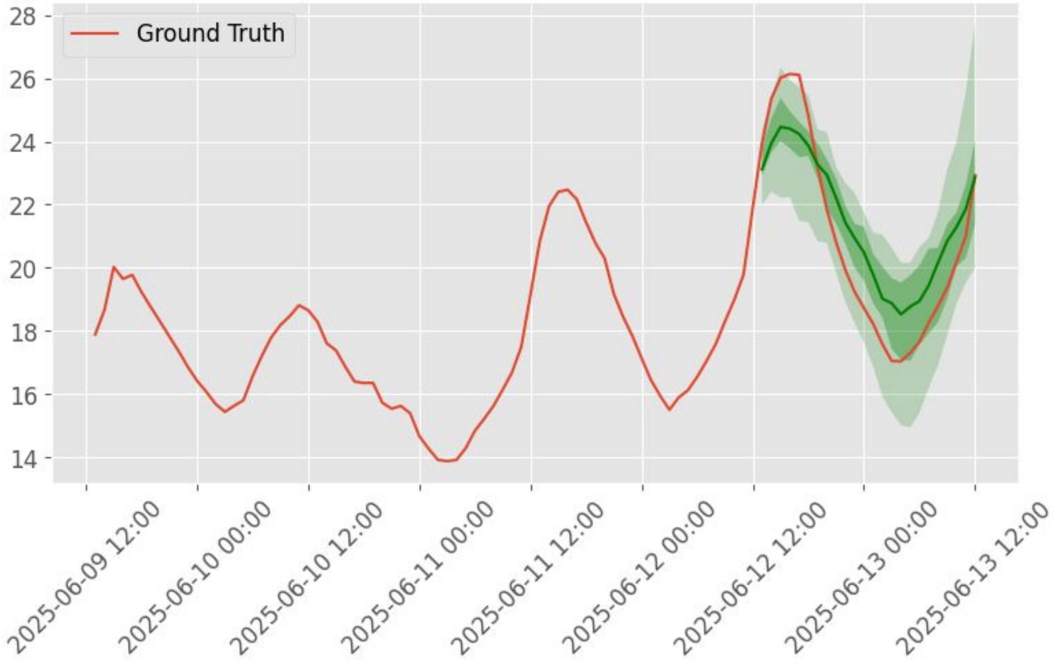
Lag-Llama IV - Fine Tuned Evaluation

RMSE: 1.4122

MASE: 0.7000

MAPE: 6.70%

Best Model over all metrics



Model Comparison

	Baseline	LSTM-HPO	LL-ZS	LL-HPO
RMSE	7.522	2.223	2.431	**1.164**
MASE	14.646	3.290	1.399	**0.648**
MAPE	866.76%	57.61%	73.81%	**6.41%**

Note: MAPE is sensitive to outliers, especially when actual values get close to 0°C

⟹ More of a rough indicator of model performance

Model Comparison

	Baseline	ARIMA	XGBoost	LSTM-HPO	LSTM-CNN	LL-ZS	LL-HPO
RMSE	7.522	7.307	2.724	2.223	2.366	2.431	1.412
MASE	14.646	14.179	4.910	3.290	3.428	1.399	0.700
MAPE	866.76%	848.11%	108.75%	57.61%	63.32%	73.81%	6.70%

Note: MAPE is sensitive to outliers, especially when actual values get close to 0°C. Recall the definition.

Future Work

- Expand on this work a bit (do univariate forecasts for all data collected by BME680) and write an arXiv paper.
- Build a “better-engineered” data pipeline with online learning, compare to batched training.

The End