Time Series Forecasting on Weather Data

AR, RNN, Attention, Transformer across 1h/6h/24h

X

9/1/2025

Table of Contents

Contents

Introduction

The goal of this project is to explore and compare different methods for forecasting weather time series. We use the Weather Long-term Time Series Forecasting dataset from Kaggle, which includes over 52,000 measurements of weather variables. To make predictions, we work with three different time resolutions and evaluate four types of models. The first is an **Autoregressive (AR) model**, which serves as a simple statistical baseline. The other three are sequence models designed to capture temporal dependencies: a **Recurrent Neural Network (RNN)**, an **attention-based model** that focuses on the most relevant time steps, and a **Transformer Encoder model** that leverages self-attention to learn both short- and long-term relationships. We evaluate both single-step (t+1) and multi-step (t+n) forecasting. Results are reported for two input setups: univariate using only T, and multivariate using SWDR, rh, and T to predict T. By comparing these approaches, we aim to highlight the strengths and limitations of traditional and modern forecasting methods under different conditions, from short-term to long-term prediction.

How to Run the Project

The project is on GitHub: https://github.com/GiX007/weather_time_series_forecasting. It was developed on Windows and also runs on macOS.

Requirements

All required packages are listed in requirements.txt (NumPy, Pandas, Matplotlib, Scikit-learn, PyTorch, etc.). Use of Python: 3.10+.

Installation

git clone https://github.com/GiX007/weather_time_series_forecasting.git

cd weather_time_series_forecasting

pip install -r requirements.txt

Project Structure

The structure of the project:

- Weather forecasting Project
    - data/
    - src/
        - config.py
        - eda.py
        - preprocessing.py

        - ar_models.py
        - nn_models.py
        - utils.py
    - main.py
    - auxiliary/
    - results/
    - README.md
    - requirements.txt

Core code is in ***src/*** (config, EDA, preprocessing, AR models, neural models, utilities) and the main script is **main.py**. ***auxiliary/*** contains from-scratch versions, ***results/*** stores metrics, plots and predictions, and ***data/*** is where the dataset is. The project also includes **README.md** and **requirements.txt**.

Running

Run the full pipeline (EDA → preprocessing → training → evaluation):

python main.py

Dataset

The dataset contains **52,696 time-ordered records** at **10-minute intervals** from **January 1, 2020 to January 1, 2021**, with **20 weather variables**. Key fields include atmospheric pressure (p), air temperature (T), relative humidity (rh), wind speed/direction (wv, wd), raining and radiation measures (SWDR, PAR, max. PAR). The index is a **monotonic DatetimeIndex** with **one duplicate timestamp** and **no missing values** reported. Basic stats look reasonable, for example, T mean ≈ 10.8 °C**,** with range −6.44 to 34.8 °C. Full descriptive summaries are provided in **results/*/data_summary.txt**.

EDA and Preprocessing

Data Cleaning, Resampling & EDA

Starting from the 10-minute data described above, we drop the single duplicate and ensure there are no missing values. For analysis and modeling, we resample to **1h**, **6h** and **24h,** using means for the features T, rh, SWDR, and prepare the chronological splits for modeling. The figures show feature distributions, trends and correlations. Detailed per-resolution inputs (shapes, dtypes, etc.) are in **results/*/input_data_summary.txt**.

Figure 1: *1h feature distributions and trends for T, SWDR, rh (box/line/bar).*

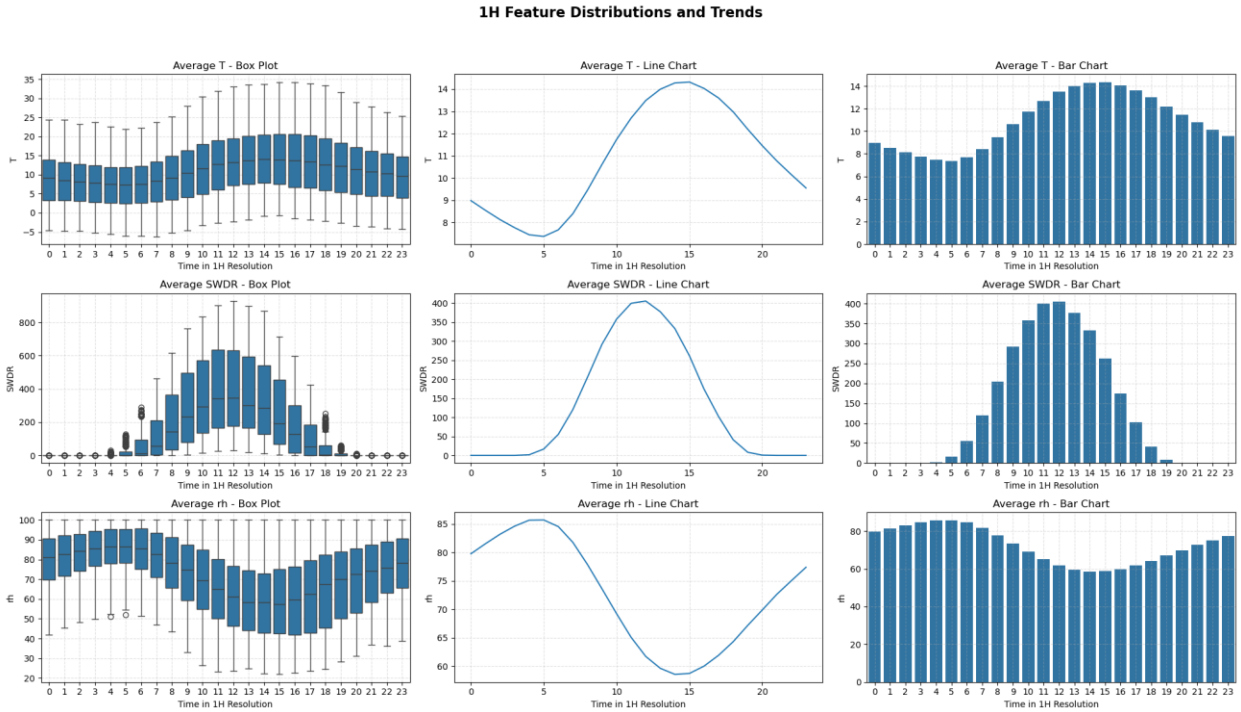
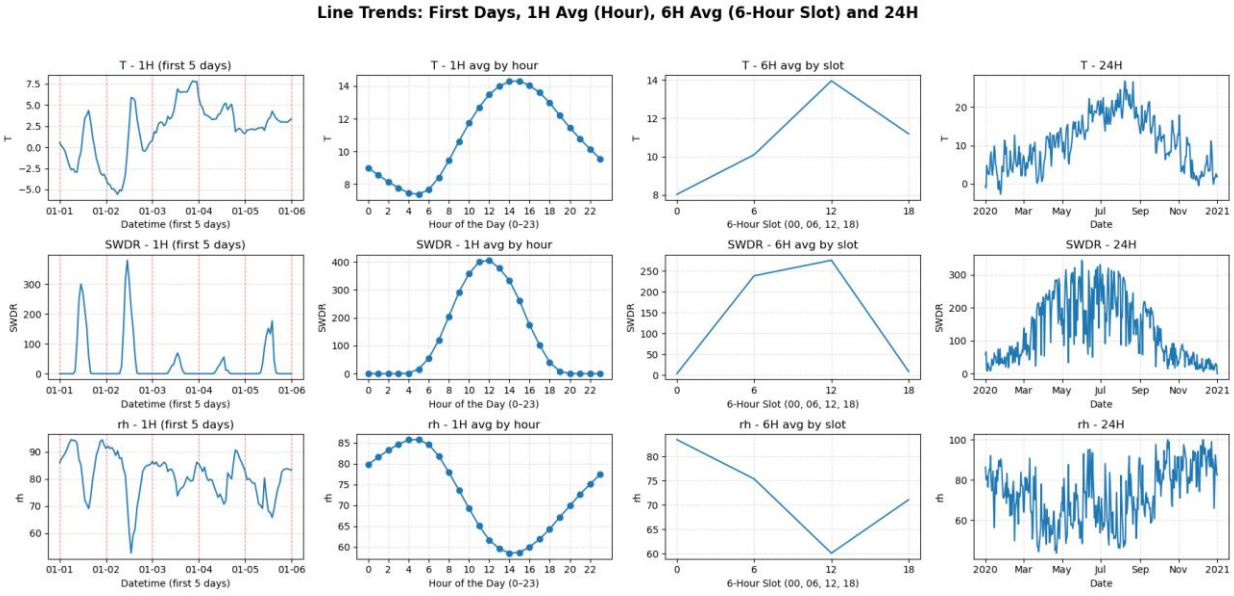


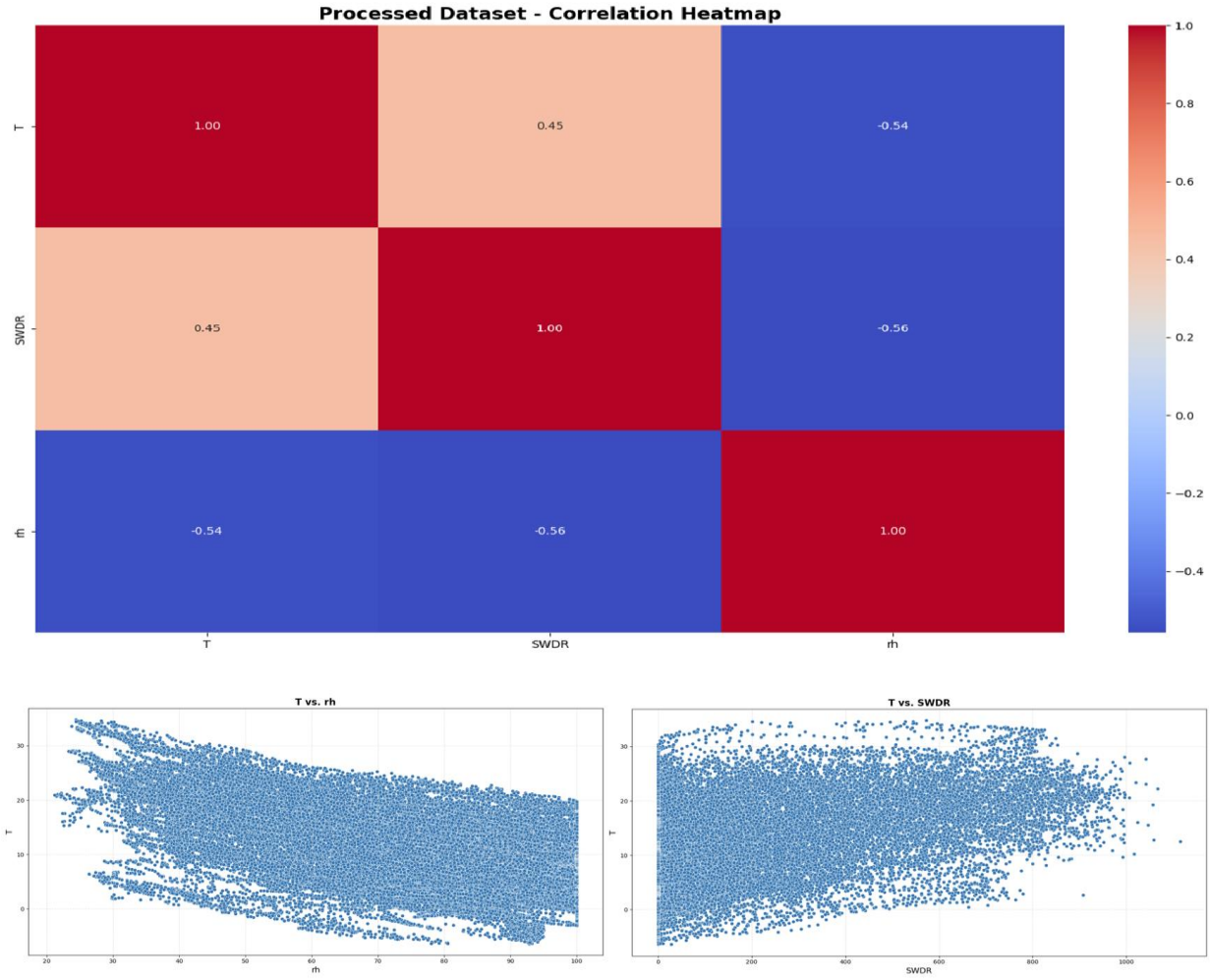
Figure 2: *Multi-resolution overview for the same three features: 1h hourly averages, 6h slots averages and 24h daily series.*



- All three features show a clear daily pattern (*see **Fig. 1***). Over the year, levels change in a consistent way between summer and winter (*see **Fig. 2***).

- **Temperature (T).** Lowest late night/early morning, peaks early afternoon, then declines (*Fig. 1*). At 24h, higher in summer (Jun–Sep), lower in winter, showing a clear seasonal pattern rather than a long-term upward or downward trend (*Fig. 2*).
- **Short-wave radiation (SWDR).** Near zero at night, rises after sunrise, peaks at midday, returns to zero after sunset. Very strong pattern and higher midday values (*Figs. 1-2*).
- **Relative humidity (rh).** Moves opposite to T: higher at night/morning, lowest in the afternoon and generally higher in cooler months (*Figs. 1-2*).
- The relationships are consistent with physics: T and rh are negatively related, while T and SWDR are positively related (see ***Fig. 3a-c***).

Figure 3: *Relationships among key features (1h). a) Correlation heatmap for T, SWDR, rh, b) Scatter: T vs. rh, c) Scatter: T vs. SWDR.*

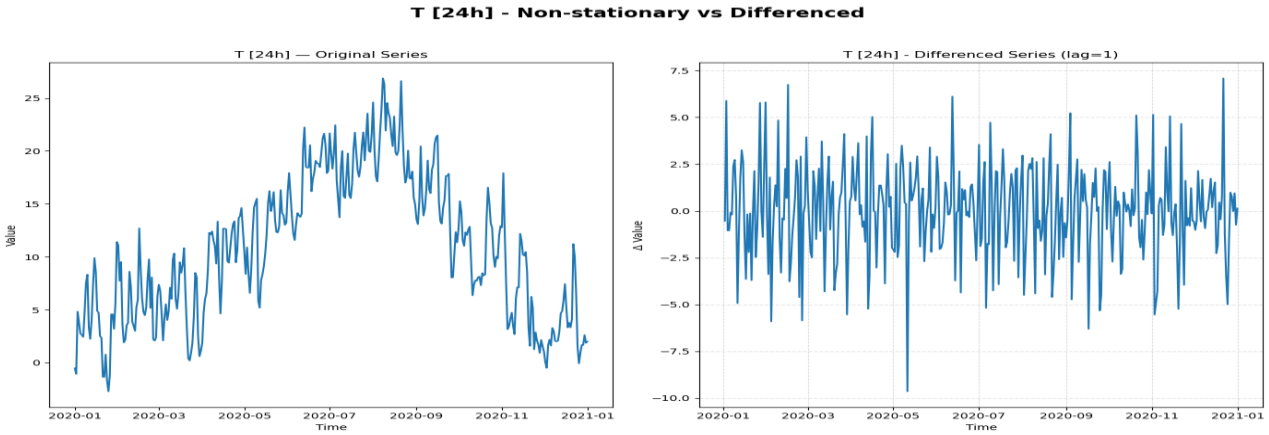


All EDA figures, covering each resolution, stationary vs. non-stationary views, and the resampled/processed series are available in ***results/figures/eda/***.

Stationarity

Stationarity is a key property in time-series modeling. An AR model assumes the series is stationary - its mean and variance are constant over time, with no trend or seasonality. Neural networks do not require stationarity. Because visual checks can be misleading, we use the **Augmented Dickey–Fuller (ADF)** test as our baseline. The results (see **results/stationarity_summary.txt**) show that **T** and **SWDR** are **non-stationary** at the **6h** and **24h** resolutions, confirming that **changing the sampling frequency can affect stationarity**. To make AR applicable, we **difference** these series before fitting. The figure below illustrates this transformation.

Figure 4: *T @ 24h: original (non-stationary) series vs. first-difference (stationary).*



Splits and sequence construction

After making any required series stationary (for AR), we split the data chronologically into train / validation / test (e.g., 70/15/15). The validation set is used for early stopping and any light NN adjustments. For AR, the model is fit on the stationary (differenced) training series, where needed, and evaluated after inverting differences. For neural networks, we form sliding input windows and targets at each resolution. We use two window configurations:

- **Short: {1H: 72, 6H: 28, 24H: 21}**
- **Long: {1H: 120, 6H: 56, 24H: 28}**

Why two sizes? Short windows focus on local/short-term patterns (e.g., daily cycles) and keep sequences manageable, e.g., RNNs handle these well and train faster with lower overfitting risk. Long windows add more (multi-day) context and attention/Transformer models leverage self-attention to connect distant time steps, while classic RNNs often struggle.

Implementation details: windows have shape (L, F) with F=1 for univariate (T only) and F=3 for multivariate (SWDR, rh, T → predict T). DataLoaders yield batches (B, L, F) and targets (B,) for t+1 or (B, n_steps) for t+n. Normalization is fit only on the training split. Full details on univariate/multivariate window shapes, batch formats and DataLoaders for all resolutions are in **results/*/input_data_summary.txt**.

Models

We compare a **classical baseline** (AR) with three **neural sequence models** (RNN, Attention, Transformer). All models support **univariate** (T only) and **multivariate** (SWDR, rh, T → predict T) inputs. Forecasts are evaluated for **t+1** and **t+n** (multi-step) horizons.

- **AR (Autoregressive) – baseline**
  - Assumption: stationary series. Non-stationary series are differenced and later inverted for evaluation.
  - Order (p): start from AR(1), then search p ∈ [1, max_lags] by BIC, and refit keeping only significant lags (drop p-values ≥ 0.05).
  - Forecasting: **rolling forecast origin** with **recursive** updates (each prediction fed back to produce the next).
  - Implementation: **statsmodels** AR fit on the training data and evaluation on test.
- **Vanilla RNN** (sequence → one)
  - Input/Output: (B, L, F) → (B, 1), where F=1 (uni) or F=3 (multi).
  - Core idea: an **nn.RNN** encodes the window, takes the last hidden state → linear head.
  - Strengths: captures short-term dynamics with low overhead, trains fast and tends to overfit less on short windows.
- **Minimal Self-Attention** (sequence → one)
  - Input/Output: (B, L, F) → (B, 1), where F=1 (uni) or F=3 (multi).
  - Architecture: input projection + positional embeddings, stacked **Multi-Head Self-Attention** + FFN with residuals & LayerNorm → mean pooling → linear head.
  - Strengths: attends to the most relevant time steps. It handles longer context better than RNNs.
- **Transformer Encoder** (sequence → one)
  - Input/Output: (B, L, F) → (B, 1), where F=1 (uni) or F=3 (multi).
  - Architecture: input projection → sinusoidal positional encoding → **nn.TransformerEncoder** (self-attention + FFN, residuals, norms) → mean pooling → linear head.
  - Intuition: self-attention links distant points directly, making it effective on longer windows.
- **Training and Inference** (common setup)
  - Windows: two sizes per resolution (short vs. long) to test near-term vs. broader-context learning.
  - Loss and optimizer: SmoothL1 loss and Adam with weight decay (L2), ReduceLROnPlateau scheduler, and early stopping.
  - Multi-step (t+n): recursive rollout: predict next step, append, re-predict, ... up to n.

- **Implementation note:** main experiments use modular, production-ready implementations (PyTorch and statsmodels). In ***auxiliary/***, we also include **from-scratch educational versions** (RNN, Attention, Transformer) with explicit inputs/outputs and visibility into **intermediate representations** (e.g., attention maps), which are used to check behavior and mechanics.
- **Hyperparameters & reporting:**
    - No full hyperparameter search, learning rate, weight decay, optimizer, dropout kept fixed.
    - Only tried varied model size/architectures (layers, heads, FFN width, hidden size) to see how model size affects performance.
    - Shared defaults and per-model settings: **src/config.py**, **src/nn_models.py**.
    - Summaries: **results/training_results.txt** (params, runtime, train/val losses).
    - Per-epoch logs: **results/nn_training_processes.txt**.
    - Learning curves: per-epoch training/validation loss plots are saved in ***results/figures/NNs_training/***.

Evaluation Setup

- **Resolutions & horizons.** We evaluate **t+1** forecasting for all three resolutions (1h, 6h, 24h). We also compare **cross-resolution setups**:
    - **1h model at t+6** (6 hours ahead) **vs. 6h model at t+1** (next 6-hour step).
    - **6h model at t+4** (next day) **vs. 24h model at t+1** (next day).

  These pairs test whether a model trained at a higher-frequency can match a lower-frequency model for the same real-time horizon.

- **Metrics.** We report **MAE**, **RMSE** and **MAPE** on the test split.
- **Rolling-forecast origins** (how predictions are made).
    - We use **walk-forward evaluation**: move the forecast origin forward through time.
    - For multi-step (t+n), we do a **recursive rollout**: predict the next step, append it to the window, and repeat until we reach n.
    - For AR, non-stationary series are **differenced** before fitting and **inverted** back to level space for evaluation.
- **Comparison criteria.**
    - **Performance per resolution:** which models do best at 1h vs. 6h vs. 24h?
    - **t+1 vs. t+n:** how much accuracy is lost as the horizon grows?
    - **1h t+6 vs. 6h t+1, 6h t+4 vs. 24h t+1:** same real-time horizon, different training frequency - can higher-frequency training achieve similar or better results?
    - **Time vs. Parameters:** compare **training time** with the **number of trainable parameters** to assess efficiency.
    - **Model order expectation based on metrics:**
        - Long windows: Transformer best → Attention → RNN → AR.
        - Short windows: RNN best → Attention and Transformer similar → AR.

Results

We report forecasting accuracy across models (AR, RNN, Attention, Transformer), input types (uni vs. multi), horizons (t+1 and t+n), and resolutions (1h, 6h, 24h). Below we show **representative tables for 1h** with **both short and long windows.** Full results for all settings are in **results/evaluation_results.txt** and ***results/predictions/***. Plots (residuals, true vs. predicted) are under ***results/figures/predictions/***.

Table 1: *1h resolution, short window L0: {1h: 72, 6h: 28, 24h: 21}.*

NN configs fixed across runs: **RNN:** hidden=32, layers=1, **Attention:** d_model=32, heads=2, layers=1, dropout=0.1, pool=mean, **Transformer:** d_model=32, heads=2, ffn_dim=64, layers=1, dropout=0.1, pool=mean. **AR:** AR(1) and AR(p) via BIC with insignificant lags dropped.

Model	Resolution	Horizon	Target	Input Type	MAE	RMSE	MAPE (%)	Fit Time	# Params
AR(1)	1h	t+1	T	Univariate	0.87	1.28	145.78	0.003	2
AR(1)	1h	t+6	T	Univariate	2.25	3.05	311.78	0.003	2
AR(8)	1h	t+1	T	Univariate	0.66	0.96	122.74	0.005	9
AR(8)	1h	t+6	T	Univariate	1.58	2.07	307.90	0.005	9
RNN	1h	t+1	T	Univariate	0.41	0.53	82.720	197.7	1153
RNN	1h	t+6	T	Univariate	2.70	3.13	715.44	197.7	1153
RNN_MV	1h	t+1	T	Multivariate	0.36	0.47	71.00	306.0	1217
RNN_MV	1h	t+6	T	Multivariate	2.59	3.20	614.57	306.0	1217
ATT	1h	t+1	T	Univariate	0.43	0.56	92.28	281.0	10945
ATT	1h	t+6	T	Univariate	2.24	2.63	572.59	281.0	10945
ATT_MV	1h	t+1	T	Multivariate	0.44	0.55	97.81	142.6	11009
ATT_MV	1h	t+6	T	Multivariate	2.26	22.75	457.99	142.6	11009
TRANS	1h	t+1	T	Univariate	0.42	0.56	82.25	250.8	8641

TRANS	1h	t+6	T	Univariate	1.92	2.37	458.63	250.8	8641
TRANS_MV	1h	t+1	T	Multivariate	0.46	0.58	104.68	202.7	8705
TRANS_MV	1h	t+6	T	Multivariate	2.11	2.54	458.05	202.7	8705

Figure 5: *Best performing model on short window L0.*

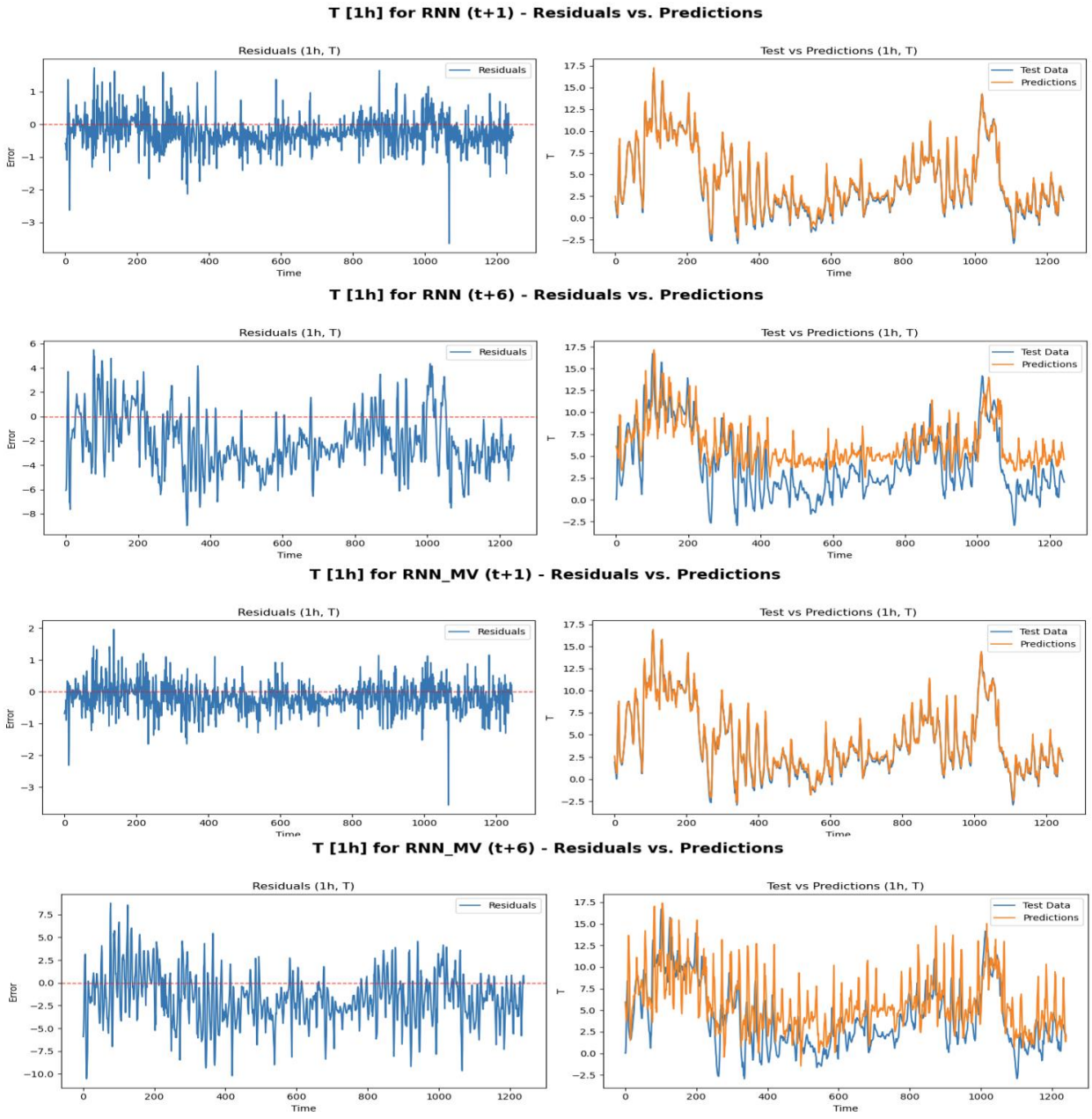
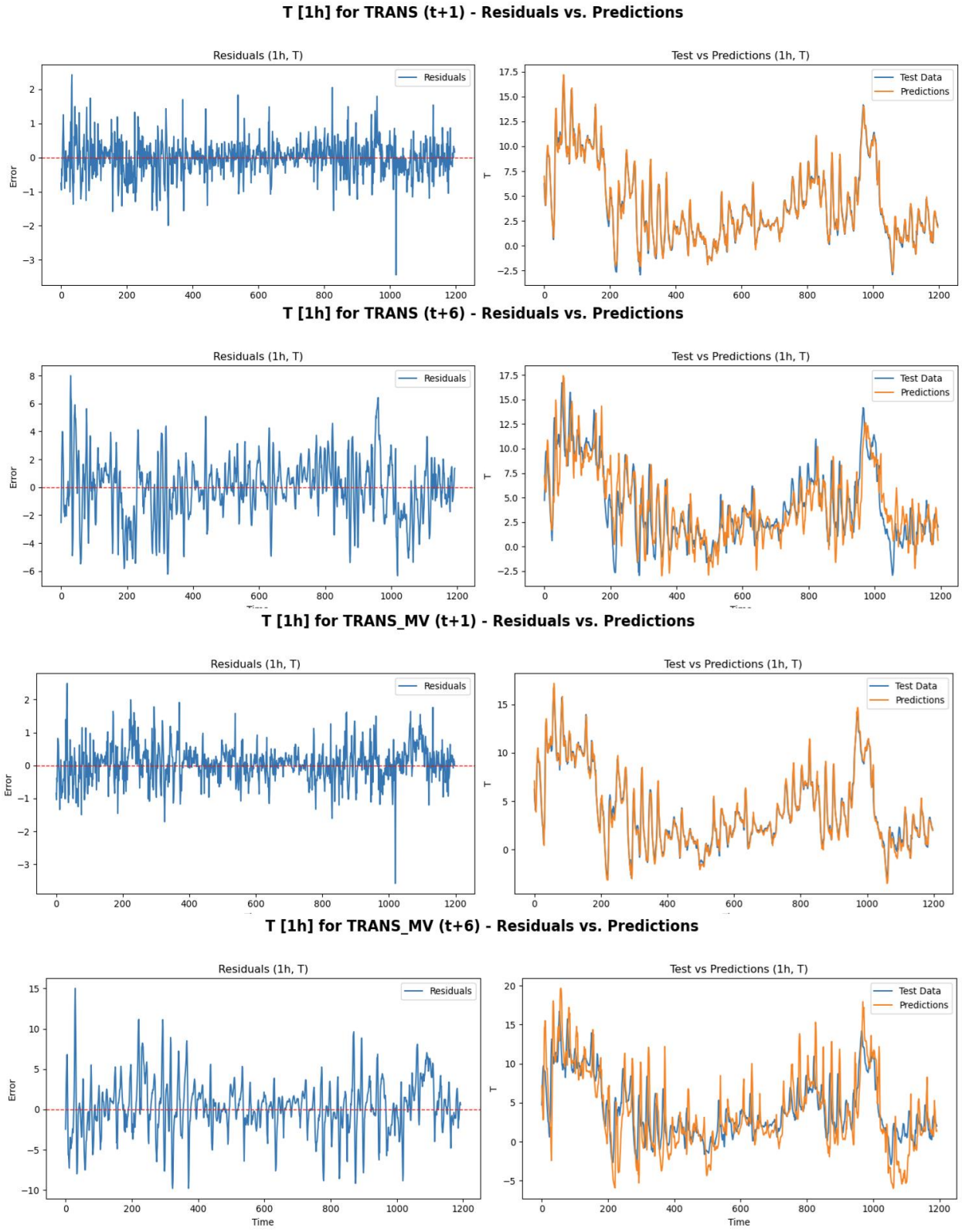


Table 2: *1h resolution, long window L1: {1h: 120, 6h: 56, 24h: 28}.*

NN configs fixed across runs: **RNN:** hidden=32, layers=1, **Attention:** d_model=32, heads=2, layers=2, dropout=0.1, pool=mean, **Transformer:** d_model=48, heads=3, ffn_dim=96, layers=2, dropout=0.1, pool=mean. **AR:** AR(1) and AR(p) via BIC with insignificant lags dropped.

Model	**Resolution**	**Horizon**	**Target**	**Input Type**	**MAE**	**RMSE**	**MAPE (%)**	**Fit Time**	**# Params**
AR(1)	1h	t+1	T	Univariate	0.87	1.28	145.78	0.003	2
AR(1)	1h	t+6	T	Univariate	2.25	3.05	311.78	0.003	2
AR(8)	1h	t+1	T	Univariate	0.66	0.96	122.74	0.005	9
AR(8)	1h	t+6	T	Univariate	1.58	2.07	307.9	0.005	9
RNN	1h	t+1	T	Univariate	0.41	0.53	83.09	225.6	1153
RNN	1h	t+6	T	Univariate	2.73	3.17	706.32	225.6	1153
RNN_MV	1h	t+1	T	Multivariate	0.31	0.42	39.73	238.5	1217
RNN_MV	1h	t+6	T	Multivariate	2.04	2.70	324.1	238.5	1217
ATT	1h	t+1	T	Univariate	0.4	0.54	76.05	667.3	21025
ATT	1h	t+6	T	Univariate	2.48	3.20	463.14	667.3	21025
ATT_MV	1h	t+1	T	Multivariate	0.43	0.57	88.93	246.8	21089
ATT_MV	1h	t+6	T	Multivariate	2.78	3.59	532.47	246.8	21089
TRANS	1h	t+1	T	Univariate	0.36	0.50	48.87	933.6	38065
TRANS	1h	t+6	T	Univariate	1.59	2.10	245.95	933.6	38065
TRANS_MV	1h	t+1	T	Multivariate	0.39	0.53	84.55	585.6	38161
TRANS_MV	1h	t+6	T	Multivariate	2.36	3.22	444.91	585.6	38161

Figure 6: *best performing model on long window L1.*



Key Findings

- **Performance per resolution (1h, 6h, 24h).**
  At 1h (t+1), the best NN outperforms AR. The same happens at 6h and 24h (see /results/evaluation_results.txt). Attention and Transformer are close and, in some cases, change places.
- **Univariate vs. Multivariate.**
  Adding **SWDR and rh** to **T** usually improves **t+1** the accuracy, but gains are smaller at longer horizons.
- **t+1 vs. t+n.**
  Errors rise as **n** grows across all models.
- **Window length (short vs. long).**
  Short windows: RNN generally wins **Attention/Transformer** models and adding complexity (e.g., extra layers didn't help, see results/L0_ATT_HID32_L2_H2_TRANS_HID48_L2_FFN96_H3/evaluation_results.txt).
  Long windows: Attention/Transformer models benefit more and outperform **RNN**. Adding complexity (e.g., layers) in short window did not help
- **Same real-time horizon, different training frequency.**
  1h @ t+6 vs. 6h @ t+1: higher-frequency training **did not match** the 6h model, which performs better.
  6h @ t+4 vs. 24h @ t+1: higher-frequency training **did not match** the 24h model, which again beats the lower-frequency training model.
- **Time vs. parameters** (efficiency).
  Smaller models train faster. Attention/Transformer are **larger** and **slower**, but deliver better accuracy, especially with long windows.
- **AR variants.**
  AR(p) (BIC + significant lags) improves over **AR(1)** at **t+1**, but it's not better than NN models and performance drops quickly at **t+6**.
- **Takeaway.**
  All NNs beat AR. For **short windows**, **RNN** tends to lead, but as context grows, **Transformer ≈ Attention > RNN ≫ AR** in accuracy, at the cost of more parameters and training time.

Conclusion

This study compared a classical baseline (**AR**) with three neural models (**RNN, Attention, Transformer**) on weather time series across **1h, 6h, 24h** resolutions and **t+1 / t+n** horizons. Overall, all neural models outperformed AR with trade-offs between simplicity, accuracy and training time. For **short windows**, **RNN** was usually strongest, while for **long windows**, **Transformer/Attention** made better use of broader context and led the results. Adding **SWDR** and **rh** to **T** gave small but consistent gains at **t+1** and benefits were smaller for longer horizons. Errors increased with horizon, and in the cross-resolution tests the model trained at the **lower data frequency** (e.g., 6h at t+1, 24h at t+1) generally **beat the higher-frequency model** aimed at the same real-time target (e.g., 1h at t+6, 6h at t+4). We enforced stationarity for AR via differencing,

used chronological splits and rolling forecasts, and evaluated with MAE/RMSE/MAPE. No exhaustive hyperparameter search was performed, core optimization settings were fixed, with only size/architecture varied.

Future Work

- **Per-case configurations & tuning.** Define **resolution/horizon-specific configs** (sequence length, feature set for multivariate, split ratios, batch size, epochs, etc.) and perform targeted **hyperparameter tuning** (LR, optimizers, weight decay, dropout, layers/heads/hidden/FFN, etc.) to see how much extra performance each model can reach.
- **Accuracy vs. Inference time.** Record **accuracy and prediction time** side-by-side (CPU/GPU) to show efficiency, including model's size (# params).
- **Feature selection for multivariate.** Use features **most correlated with T** (e.g., **VPmax**, see correlation heatmap in the appendix) to build stronger multivariate inputs.
- **Architectures.** Try casual masking for attention so predictions only use past time steps.
- **Multiple targets.** Try additional targets (e.g., **T and rh**).
- **Deployment.** Package the best model with a simple interface and offer a lightweight app.

Appendix

*A complete list of links and citations is embedded in the **auxiliary/ notebooks**, which provide detailed, step-by-step explorations.*