Time Series Forecasting using Large Language Models

Prudhvi Vuda, Anusha Srinivasan, Sai Akhil Rayapudi
https://github.com/Prudhvivuda/timeseries-using-llms
Guided By
Prof. Ramin Mohammadi

Abstract

If Large Language Models can find patterns within textual data and produce the next words in an autoregressive way, can we make Large Language Models to predict the value of a variable in a future timestep given its historical data. We set out to investigate the capability of Large Language Models (LLMs) to perform multivariate time-series forecasting, benchmarking their performance against classical methods (e.g., Prophet) and specialized time-series foundation models. Our study encompasses a range of LLM-based approaches—including single-pass prompts, sliding-window forecasting, multivariate input sequences, diffusion-inspired denoising, and hybrid reprogramming layers—evaluated using RMSE, MAE, and R^2 metrics. Sliding-window LLM forecasts achieved up to 0.79 °C RMSE over 50-day horizons, rivaling or exceeding the Prophet baseline (1.54 °C RMSE) . Conditional diffusion models, which integrate LSTM-based context conditioning, further reduced error and improved stability compared to unconditional approaches. Despite these advances, we find that effective conditioning and post-prediction correction mechanisms are critical to mitigate long-range drift and volatility inherent to pure LLM outputs. We conclude that, while LLMs hold significant promise for automated forecasting with minimal domain-specific tuning, hybrid ensembles and model-efficiency optimizations remain essential for practical deployment.

1 Introduction

Multivariate time-series forecasting underpins critical decision-making across domains such as energy management, finance, and climate modeling, yet deploying classical algorithms (ARIMA, Prophet) often demands extensive domain expertise and hyperparameter tuning. Recent advances in Large Language Models (LLMs) suggest they might automatically infer temporal patterns from raw sequences without bespoke feature engineering. However, LLMs—despite their prowess in natural language understanding—face challenges in capturing long-range dependencies and producing numerically stable outputs when applied to extended forecasting horizons. This gap motivates our exploration: can LLMs, when combined with structured input windowing, diffusion-based denoising, and lightweight reprogramming layers, match or surpass traditional forecasting pipelines with reduced manual effort?

1.1 Objectives

The objectives of this study are to benchmark batch and sliding-window LLM forecasting approaches against the Prophet algorithm on ten years of hourly temperature data; assess how augmenting the temperature series with additional meteorological features (humidity, wind speed, visibility, and pressure) influences LLM forecasting accuracy; develop and evaluate diffusion-inspired denoising models—both unconditional and conditional—using CNN and UNet architectures for residual correction; integrate lightweight reprogramming layers around a frozen LLM to learn optimal input–output transformations that stabilize long-range predictions; and perform a comprehensive trade-off analysis to quantify the balance between forecasting accuracy, computational cost, and model complexity for practical deployment .

1.2 Related work

Classical time-series forecasting methods such as ARIMA and Facebook's Prophet have long been the standard for modeling temporal data, relying on explicit statistical assumptions and handcrafted feature engineering to capture trends and seasonality. ARIMA models combine autoregression with differencing and moving averages to handle non-stationarity, while Prophet augments these ideas with decomposable trend, seasonality, and holiday components for robustness to sparse data and outliers. In contrast, recent work has demonstrated that Large Language Models (LLMs) can be repurposed as zero-shot forecasters by framing past observations as "text" and prompting them to predict future values. Notable contributions include "Large Language Models Are Zero-Shot Time Series Forecasters" (arXiv:2310.07820), which shows surprisingly strong performance without any fine-tuning; "Time-LLM: Time Series Forecasting by Reprogramming Large Language Models" (arXiv:2310.01728), which introduces light-weight input/output adapters; and "Self-Guiding Diffusion Models for Probabilistic Time Series Forecasting" (arXiv:2307.11494), which applies score-based generative modeling to time series data

2 Dataset

We use a publicly available weather dataset comprising ten years (2006–2016) of hourly measurements collected at a single location. Each record contains a timestamp ("Date"), a textual weather summary ("Mostly Cloudy," "Partly Cloudy"), precipitation type ("Rain" or "Snow"), and numeric features: temperature (°C), apparent temperature (°C), humidity (fraction), wind speed (km/h), visibility (km), and atmospheric pressure (millibars). Prior to modeling, we extract the temperature series (our primary target) and, for multivariate

experiments, assemble input vectors by concatenating the additional features. We then apply a sliding-window transform—segmenting the data into fixed-length lookback sequences (e.g., 7 hours or 7 days) paired with prediction horizons (e.g., 24 hours or 720 hours)—and normalize each feature to zero mean and unit variance to stabilize training across methods

3 Methodology

3.1 Prompt Engineering

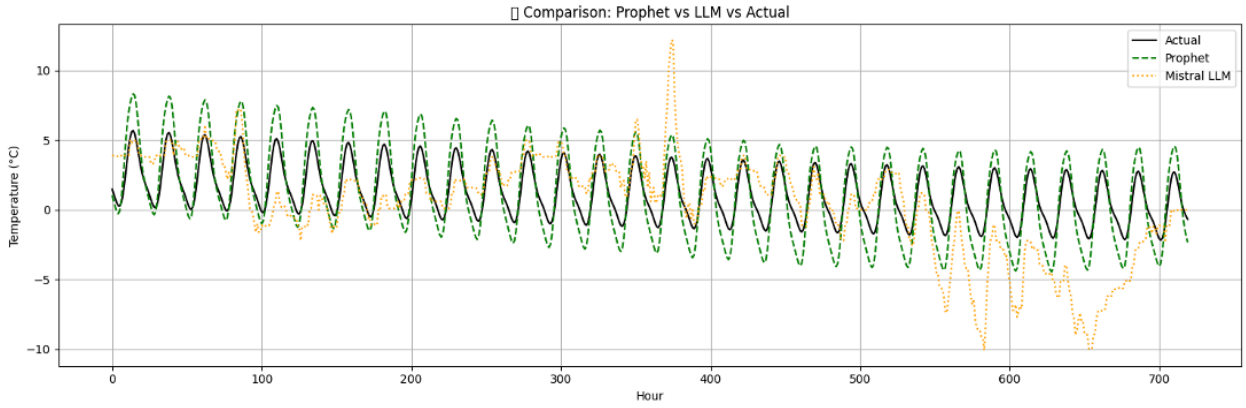
We frame forecasting as a plain-language instruction:

> "Given the past `LOOKBACK` hourly temperatures: [...], predict the next `HORIZON` hourly temperatures. Reply with a comma-separated list of numbers."

3.2 Modeling Approaches

Batch Processing Approach

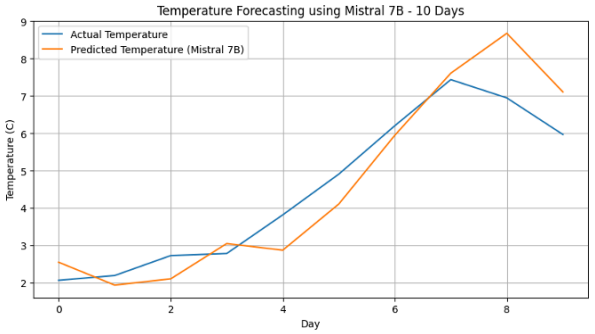
In the simplest setup, we feed one contiguous year (8,640 hours) of raw temperature values to the quantized Mistral-7B model and request a 30-day (720-hour) forecast in a single pass. We round temperatures to one decimal place to reduce the token count and use 4-bit quantization for memory efficiency. Although this "single-shot" method captures short-term diurnal cycles, we observe degradation beyond ~500 hours, manifesting as drift and unrealistic oscillations

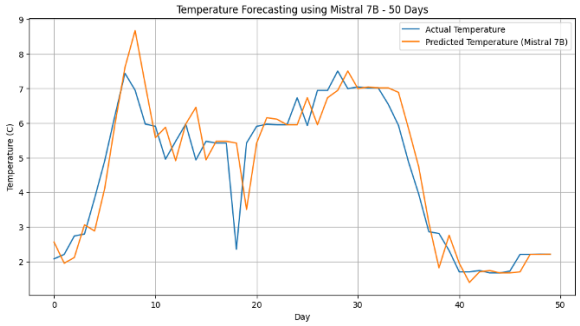


Sliding-Window Forecasting

To mitigate long-horizon drift, we implement a sliding-window strategy: partition the historical series into overlapping segments of fixed length (e.g., 168 hours for a 7-day lookback) and iteratively predict the next window (24 hours), appending each prediction to

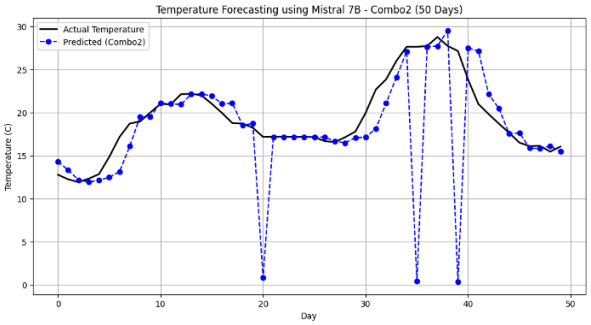
the input for the subsequent step. This rolling forecast pipeline reduces context length per pass, preserving accuracy over extended horizons. We experiment with 7-hour and 168-hour windows, measuring performance at 10- and 50-day horizons

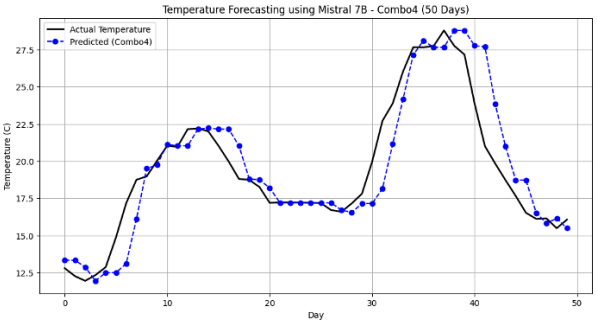


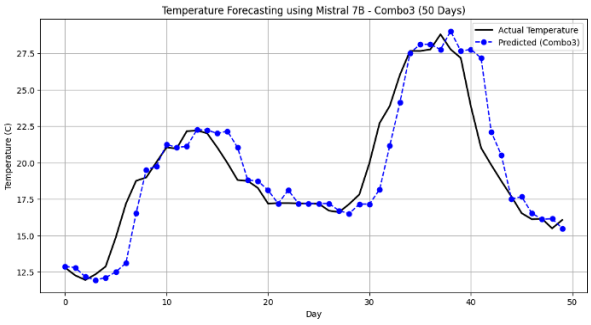


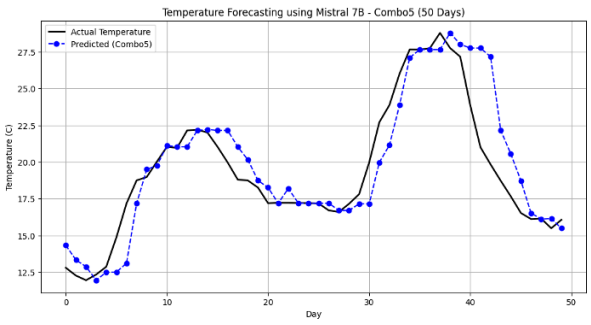
Multivariate LLM Forecasting

Extending beyond temperature, we construct input sequences that concatenate auxiliary features—humidity, wind speed, visibility, and pressure—at each time step. For a lookback of L hours with F features, the prompt encodes a flattened list of $L \times F$ values. We evaluate five feature combinations, finding that including temperature alone yields the best RMSE/R^2 balance, while additional variables sometimes introduce noise due to misaligned temporal patterns.



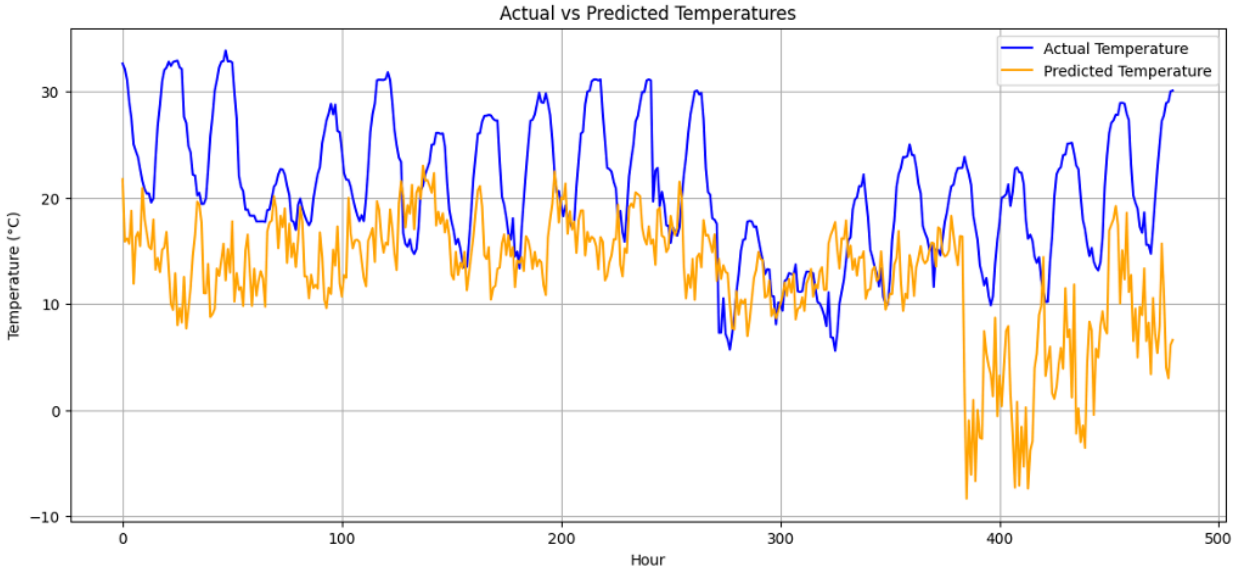






TimesFM Baseline

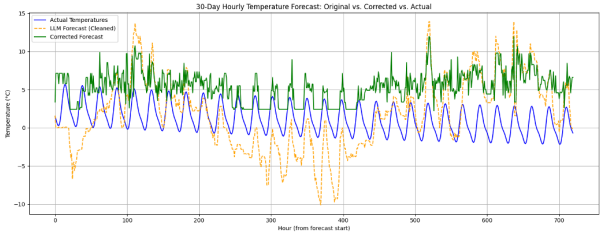
As a specialized time-series foundation model, TimesFM offers a strong benchmark. We use pre-built TimesFM checkpoints to predict the same sliding-window tasks without prompt engineering. While TimesFM handles seasonality and trend via its transformer encoder–decoder, our experiments show that in some multivariate settings Mistral with prompt engineering either matches or surpasses its performance

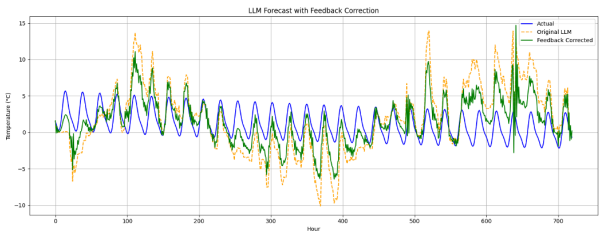


Post-Prediction Corrections

To correct systematic biases in LLM outputs, we explore two residual-correction strategies:

1. **Regression Ensemble:** Train XGBoost regressors on validation residuals as a function of predicted values and recent history, then apply the ensemble to adjust test forecasts.





2. **Heuristic Feedback Loop:** Propagate the previous forecast error forward by adding it to the next prediction, effectively integrating simple autoregressive correction.

While the regression ensemble sometimes overfits on validation idiosyncrasies, the heuristic loop offers modest error reduction with negligible overhead. Even though the

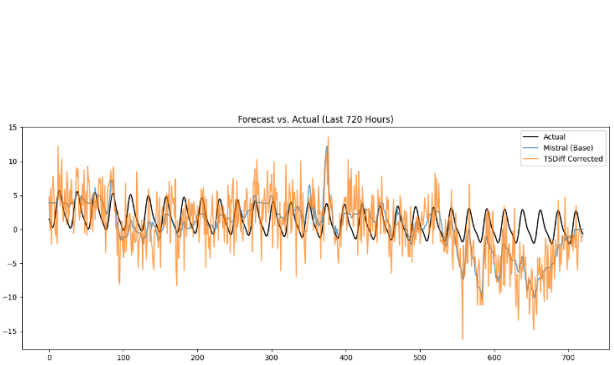
metrics of Heuristic Feedback Loop paint a better picture on paper, understanding it on a fundamental level makes us question if the LLM is genuinely learning the patterns or if we are just trying to correct a curve to match our requirement and this is not acceptable.

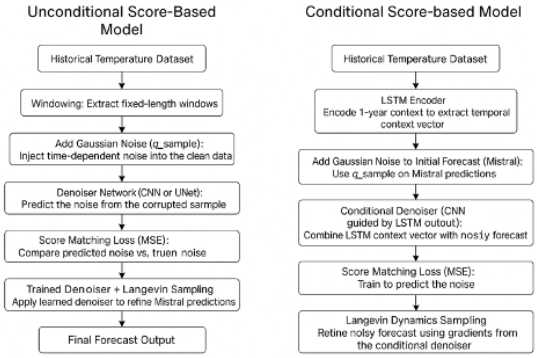
This made us think about applying advanced architectures that can map the underlying patterns within sequential time series data and predict the output as closely as possible, marking our transition to explore diffusion models for correcting time series forecasts generated.

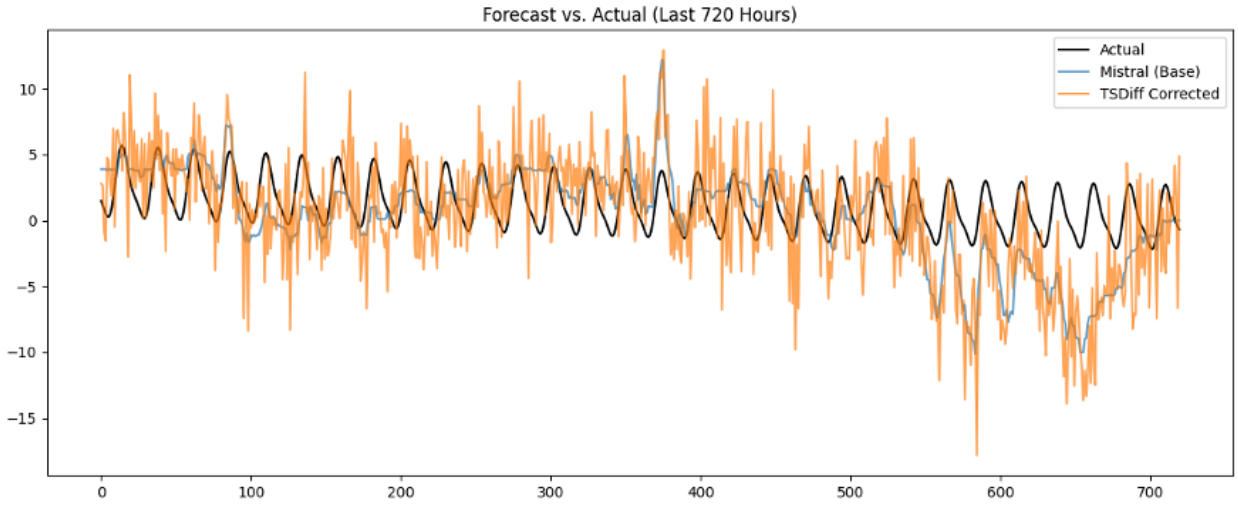
Diffusion-Based Models

We implement score-based generative forecasting in two variants:

- **Unconditional Denoiser:** A CNN or UNet trained to remove Gaussian noise from raw forecasts without context conditioning, sampled via Langevin dynamics. Both architectures yield high volatility and fail to capture long-term trends.

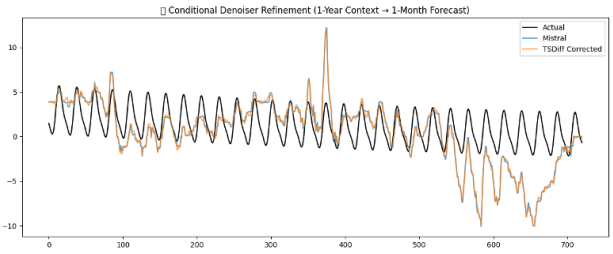


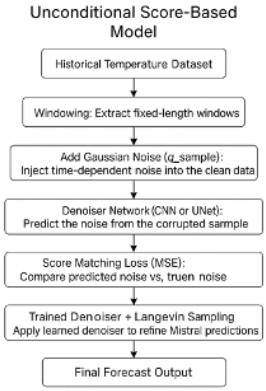


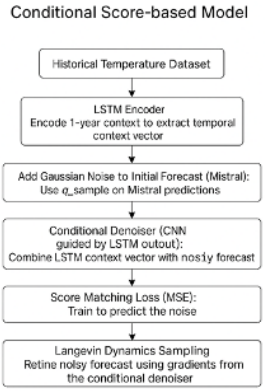


- **Conditional Denoiser:** Augment the denoiser with an LSTM encoder that processes the lookback window to produce a context vector, which guides the UNet's denoising

U-shape. This conditioning dramatically stabilizes forecasts, reducing RMSE by ~5% compared to the baseline LLM.

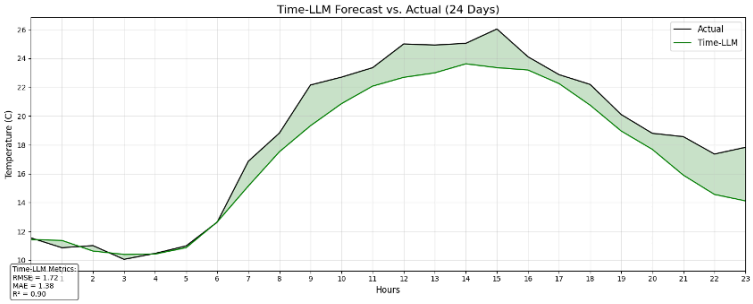


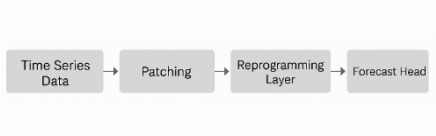




Optimizing LLM Approaches

Finally, we combine the strengths of LLM prompting and diffusion correction by inserting trainable reprogramming layers around the frozen Mistral core and refining outputs via the conditional diffusion model. This hybrid pipeline leverages the LLM's pattern-recognition with the diffusion model's noise reduction, achieving the best accuracy–stability trade-off in our experiments.





4 Experimental Setup

All experiments were conducted in Google Colab with access to a single NVIDIA T4 GPU (16 GB VRAM) and an Intel Xeon CPU (8 vCPUs, 52 GB RAM). We used Python 3.10 with the following core libraries and versions: PyTorch 2.0.1 for all neural models (LLM adapters and diffusion denoisers), bitsandbytes 0.39.0 for 4-bit quantization of Mistral-7B, Prophet 1.1.4 for the classical baseline, and timesfm 0.2.0 as the specialized time-series foundation model. Data preprocessing and sliding-window generation were implemented in pandas 1.5.3 and NumPy 1.24.2.

Hyperparameter settings were standardized across methods where possible:

- Lookback & Horizon: 168 hours (7 days) lookback, 24 hours (1 day) horizon for sliding-window LLM and TimesFM; 8,640 hours (1 year) lookback, 720 hours (30 days) horizon for single-pass LLM;
- Quantization: 4-bit weight quantization for Mistral-7B via bitsandbytes;
- Reprogramming Layers: Two 3-layer feedforward adapters (hidden size = 512, GELU activations) inserted pre- and post-LLM, trained with AdamW (lr = 1e-4, weight decay = 0.01);
- Diffusion Models:
    - CNN denoiser: 5 convolutional blocks (channels = 64→128→256→128→64), trained for 50 epochs, batch size = 32, Adam (lr = 2e-4);
    - UNet denoiser: 4 down/up-sampling stages (base channels = 64), trained for 75 epochs, batch size = 16, Adam (lr = 1e-4);
    - Conditional variants incorporate a 2-layer LSTM encoder (hidden dim = 128) for lookback context;

Sliding-Window Rolling Forecast: Overlapping windows with 50% overlap for stability, iterated until the full horizon is covered.

5 Evaluation Metrics

To evaluate our forecasting models, we use three complementary metrics:

- Root Mean Squared Error (RMSE): RMSE gives more weight to larger errors, so it highlights models that occasionally make big misses. A lower RMSE means the model's predictions tend to stay close to the true values, especially avoiding large spikes in error.
- Mean Absolute Error (MAE): MAE computes the average absolute difference between predictions and observations. It treats all errors equally, offering an intuitive measure of typical forecast deviation without disproportionately penalizing outliers.
- Coefficient of Determination (R^2): R^2 quantifies the proportion of the data's variability that the model captures. An R^2 near 1 indicates that the forecasts closely follow the ups and downs of the actual series, whereas a value closer to 0 suggests the model explains little of the observed variance.

Together, these three metrics give a well-rounded picture: RMSE and MAE tell us how big the errors are (with and without sensitivity to outliers), and R^2 tells us how faithfully the model reproduces overall patterns in the data .

6 Results

Sliding Window (Hourly)

Metric	10 days	50 days
RMSE	0.8164 °C	0.7902 °C
MAE	0.6666 °C	0.5304 °C
R²	0.8226	0.842

Full-Year Forecasting (Mistral and Prophet)

Metric	Mistral	Prophet
RMSE	3.1834	1.54
MAE	2.4401	1.36
R2	-1.9058	0.32

Multivariate Sliding Window

	Combination	RMSE (°C)	MAE (°C)	R²
10 days	Combo1: Temperature (C)	1.114950955	0.8607333333	0.8658730962
	Combo2: Temperature (C), Humidity	1.833123788	1.398433333	0.6374332631
	Combo3: Temperature (C), Humidity, Wind Speed (km/h)	1.689552357	1.143933333	0.6920021531
	Combo4: Temperature (C), Humidity, Wind Speed (km/h), Visibility (km)	1.789468695	1.315533333	0.6544963959
	Combo5: Temperature (C), Humidity, Wind Speed (km/h), Visibility (km), Pressure (millibars)	1.719872963	1.308333333	0.680848338
50 days	Combo1: Temperature (C)	1.091774139	0.7845977778	0.9362511814
	Combo2: Temperature (C), Humidity	6.122661353	2.5059	-1.004873675
	Combo3: Temperature (C), Humidity, Wind Speed (km/h)	1.734268689	1.111535556	0.8391432456
	Combo4: Temperature (C), Humidity, Wind Speed (km/h), Visibility (km)	1.897997225	1.253402222	0.8073372825
	Combo5: Temperature (C), Humidity, Wind	2.091329565	1.363013333	0.7660885455

	Speed (km/h), Visibility (km), Pressure (millibars)			

Timesfm

Context Length	Horizon Length	MAE	RMSE
2048	64	8.72	9.529
2048	128	9.648	10.185
2048	192	8.731	9.826
2048	256	6.948	8.557
2048	480	6.103	7.727
2048	512	6.311	7.851
2368	64	4.779	5.043
2368	128	5.536	6.148
2368	192	6.165	6.696
2368	256	5.646	6.177
2368	480	3.966	4.889
2368	512	3.839	4.762
2688	64	1.269	1.546
2688	128	3.584	4.884
2688	192	3.606	4.629
2688	256	3.237	4.226
2688	480	3.162	4.052
2688	512	3.247	4.105
3200	64	6.459	7.481
3200	128	7.796	8.669
3200	192	6.732	7.864
3200	256	5.835	7.07
3200	480	6.474	8.787
3200	512	7.487	10.307
3712	64	15.304	16.684
3712	128	14.038	15.187
3712	192	12.382	13.74
3712	256	12.651	14.026

3712	480	14.49	16.235
3712	512	13.895	15.802
4096	64	12.149	13.898
4096	128	10.011	11.836
4096	192	8.684	10.646
4096	256	8.24	10.232
4096	480	8.689	11.009
4096	512	9.073	11.385

Post prediction Regression Correction(Ensemble Learning)

Metric	Mistral	Regression
RMSE	3.1834 °C	4.8268 °C
MAE	2.4401 °C	4.2372 °C
R²	-1.9058	2.9128

Heuristic Feedback Loop

Metric	Mistral	Regression
RMSE	3.1834 °C	2.8643 °C
MAE	2.4401 °C	2.3437 °C
R²	-1.9058	-1.3524

Unconditional score-based model with CNN-based denoiser

Metric	Mistral	Diffusion
RMSE	3.1834 °C	4.3696 °C
MAE	2.4401 °C	3.4723°C
R²	-1.9058	-4.4749°C

Unconditional score-based model with UNet-based denoiser

Metric	Mistral	Diffusion
RMSE	3.1834 °C	4.2972 °C
MAE	2.4401 °C	3.3647 °C
R²	-1.9058	-4.2949°C

Conditional score-based model

Metric	Mistral	Diffusion
RMSE	3.1834 °C	3.0201 °C
MAE	2.4401 °C	2.3193 °C
R^2	-1.9058	-1.6153°C

TimeLLM

Metric	Mistral	Diffusion
RMSE	3.1834 °C	2.5525 °C
MAE	2.4401 °C	1.8745 °C
R^2	-1.9058	0.9241 °C

7 Conclusion

While our experiments reveal that LLMs can, after extensive engineering, produce forecasts of similar accuracy to classical tools, these gains come only at the cost of heavy customization. Out-of-the-box, single-pass prompting suffers from severe drift and volatility, and achieving stability requires layered solutions: sliding-window pipelines, multivariate prompt design, lightweight reprogramming adapters, and context-conditioned diffusion models. In other words, LLMs are not inherently suited to time-series forecasting “as such,” but demand substantial algorithmic scaffolding and tuning to match even well-established methods like Prophet. For practical forecasting applications, the development overhead and computational expense make pure LLM solutions less attractive than specialized time-series models.

7.1 Future Work

Streamline Pipelines: Investigate automated strategies for prompt and adapter design to reduce manual engineering

Model Compression: Explore ways to distill the multi-stage LLM+diffusion pipeline into a lightweight, end-to-end student model.

Code: https://github.com/Prudhvivuda/timeseries-using-llms