LLM4SERIES: STRUCTURED PROMPTING FOR TIME SERIES FORECASTING WITH LLMS

Wesley Barbosa[1], **Fernanda Scarcela**[1], **Zairo Bastos**[1], **João P. V. Madeiro**[2], **Wellington Franco**[1], **Carlos Caminha**[1]

[1]Kunumi Lab, Universidade Federal do Ceará (UFC), Fortaleza, CE, Brazil
[2]MDCC, Universidade Federal do Ceará (UFC), Fortaleza, CE, Brazil
{wesley.barbosa, fernandascla, zairo.vianahd}@alu.ufc.br
caminha@ufc.br, wellington@crateus.ufc.br, jpaulo.vale@dc.ufc.br

ABSTRACT

We introduce `LLM4Series`, an open-source Python library that standardizes time series forecasting with Large Language Models (LLMs). Despite the growing interest in applying LLMs to temporal tasks, the lack of dedicated tooling hinders systematic experimentation. `LLM4Series` addresses this gap through a modular and extensible pipeline encompassing data serialization, structured prompt construction, and automated evaluation. We validate the framework on three real-world datasets, comparing it against statistical and deep learning baselines. Results show that our structured methodology achieves competitive accuracy while substantially reducing the complexity of LLM-based forecasting. Source code is available at the **project repository**.

Track: Research

1 INTRODUCTION

Time series forecasting (TSF) has numerous real-world application areas, including finance, retail, healthcare, energy, and disease prevention and control (Andersen et al., 2005; Böse et al., 2017; Topol, 2019; Salinas et al., 2020; Wu et al., 2020). While established approaches like ARIMA (Shumway & Stoffer, 2017; Mondal et al., 2014) and supervised learning (Freitas et al., 2023; Zhang et al., 2024; Yadav & Thakkar, 2024) have played a key role in this field, recent advances in general-purpose Large Language Models (LLMs) have demonstrated strong generalization across diverse domains (Wang et al., 2024; Gu, 2023; Bastos et al., 2025). This progress has motivated the investigation of LLMs as an alternative paradigm for TSF, leveraging in-context learning and structured textual representations of temporal (Gruver et al., 2023; Xue & Salim, 2023).

While several robust toolkits for time series forecasting are currently available, most of them emphasize classical statistical methods (Seabold et al., 2010; Herzen et al., 2022; Middlehurst et al., 2024), traditional machine learning approaches (Alexandrov et al., 2020; Löning et al., 2019), or other time series tasks such as classification and clustering (Faouzi & Janati, 2020; Tavenard et al., 2020). However, to the best of our knowledge, no widely adopted Python library currently provides a standardized framework for systematically orchestrating LLMs as forecasting models.

To bridge this gap, we introduce `LLM4Series`, a Python library specifically designed for time series forecasting with Large Language Models. The framework provides modular components for data representation, preprocessing, prompt construction, model interfacing, evaluation, and visualization within a unified pipeline. Our main contributions are: (i) A modular architecture that integrates LLM-based forecasting into the Python time series ecosystem; (ii) Built-in prompt engineering utilities enriched with descriptive statistics and seasonal decomposition; (iii) A unified abstraction layer supporting multiple LLM providers with structured output enforcement; and (iv) An empirical comparison against statistical, deep learning, and LLM-based baselines across real-world datasets.

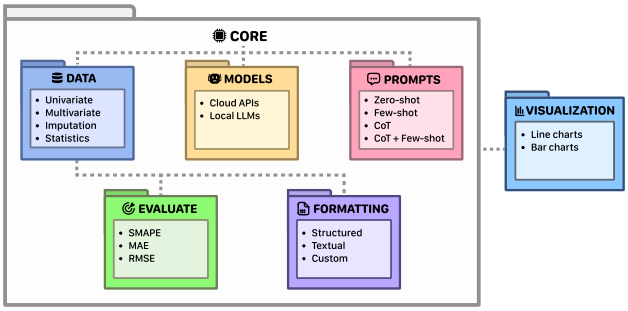


Figure 1: **Overview of the `LLM4Series` library.** The architecture emphasizes modularity, reusability, and ease of use, organized into a primary **core** module for predictive tasks and a **visualization** module for interactive plotting.

2 Library Design and Implementation

2.1 Data

Data Representation. A *time series* is an ordered sequence of observations indexed in time. To represent this structure, `LLM4Series` extends the `Pandas` ecosystem (McKinney et al., 2011) by abstracting its data structures and incorporating time series–specific operations. The library implements two core classes: `UniTimeSeries` (extending `pandas.Series`) for univariate data, and `MultiTimeSeries` (extending `pandas.DataFrame`) for multivariate scenarios. Both inherit from the abstract `TimeSeries` base class, which provides essential utilities such as frequency normalization, data splitting, and sliding-window sampling strategies. This design ensures seamless integration with existing Python data science pipelines while providing specialized structures for temporal modeling.

Data Input and Format Conversion. For data reading, the library supports common formats (CSV, Parquet, JSON, Excel) and direct `DataFrame` ingestion. When specified, the index column is converted to `datetime` and the data is automatically sorted, ensuring the series is in ascending temporal order. Single-column series are represented as `UniTimeSeries`, while multiple columns result in `MultiTimeSeries` objects. Given evidence that tokenization strategies significantly impact numerical reasoning performance (Gruver et al., 2023), the framework facilitates conversion of numerical series into textual formats like `plain`, `csv` and `json`. This design, grounded in recent findings (Fons et al., 2024), allows for a systematic assessment of how data representation influences forecasting accuracy.

Data Preprocessing and Statistical Analysis. A common challenge in time series analysis is the presence of missing data (Pratama et al., 2016). To address this, `LLM4Series` integrates imputation methods directly into the time series objects. Missing values can be handled via diverse strategies, from simple moving averages and backfilling to linear and spline interpolation. The availability of these methods within the data structure simplifies experimentation and comparison between strategies. Beyond imputation, the framework provides built-in statistical utilities for diagnostics and context enrichment, including descriptive statistics (mean, median, dispersion, and quartiles) and STL decomposition to expose trend and seasonal structure (Cleveland et al., 1990). Crucially, these extracted features serve as auxiliary context for the Prompt Engineering module, where they are injected into templates to provide the LLM with a high-level statistical summary and enhance pattern recognition.

2.2 Prompt Engineering

A core contribution of `LLM4Series` is a systematic framework for prompt engineering. Since Large Language Models operate over textual inputs, forecasting performance depends critically on how temporal information is serialized and contextualized. Thus, the library builds the prompt using four explicit components to reduce ambiguity: (i) the forecasting horizon; (ii) the historical series

data (serialized in the selected textual format, e.g., `plain` or `csv`); (iii) the descriptive statistics extracted in the previous step (e.g., trend and mean); and (iv) a specific role definition (e.g., "You are an expert forecaster"). This structure provides a robust quantitative context, helping the model focus on data patterns rather than generating hallucinations.

To guarantee reliable automated evaluation, the framework enforces strict output constraints. All templates instruct the model to generate *only* numerical predictions enclosed within `<out>` tags. This mechanism effectively isolates numerical values from explanatory text, preventing parsing ambiguities caused by extraneous natural language content and ensuring that the evaluation pipeline operates on structured, machine-readable outputs.

Finally, the library supports multiple inference strategies, including *zero-shot* (Kojima et al., 2022), *Chain-of-Thought* (CoT) (Wei et al., 2022) to induce intermediate reasoning, and *few-shot* learning (Khot et al., 2022). For the latter, `LLM4Series` implements flexible sampling policies, such as *front-end*, *back-end*, *random*, and *uniform* sliding windows, to optimize how historical examples are selected. Users may also define custom prompt templates, preserving flexibility without sacrificing structural consistency. Example prompt templates are provided in Appendix B.

2.3 Unified Interface for LLM Providers

For forecasting, the library defines a unified `Model` interface that standardizes interactions with OpenAI, Azure, and LM Studio. This abstraction provides a consistent interface across providers, facilitating systematic evaluation of different LLM backends. The system normalizes outputs into a `ModelResponse` object that aggregates model completion, extracted values, and execution metadata (latency, token usage) for subsequent analysis. To ensure reliability, strict generation rules enforce output formatting and continuity immediately following the last observation.

2.4 Model Evaluation and Interactive Visualization

Finally, to assess predictive accuracy, the library integrates standard error metrics directly into the time series objects, including *Mean Absolute Error* (MAE), *Root Mean Square Error* (RMSE), and *Symmetric Mean Absolute Percentage Error* (sMAPE). The formal definitions of these metrics are reported in Appendix C. Complementing this quantitative analysis, the visualization module leverages `Plotly` to generate interactive figures, ranging from statistical bar charts to detailed STL decomposition plots. Furthermore, the library provides high-level utilities for overlaying multiple forecasts with ground truth data, facilitating visual error diagnosis and model comparison.

3 Experiments

To evaluate the practical effectiveness of `LLM4Series` in univariate forecasting scenarios, we conducted a comparative analysis with established Python packages that represent different forecasting paradigms: statistical models, deep learning, and other LLM-based agents.

Datasets and Protocol. The evaluation uses three datasets: **ETTh2** (Zhou et al., 2021) (17,420 hourly oil temperature observations), **Electricity** (Trindade, 2015) (26,304 hourly electricity consumption measurements from 321 customers) and **Mortality**[1] (2,191 daily death records in Brazil). Forecasting horizons were set to $h = 24$ and $h = 7$ for the hourly and daily datasets, respectively. Regarding context availability, statistical and deep learning methods were trained on the entire historical data available up to the train-test split. The only exception was `AutoARIMA` on the Electricity dataset, where memory constraints required the reduction of the training period to the interval from 28-06-2013 to 06-09-2013. Conversely, to simulate realistic context window constraints in LLM-based approaches, we restricted the input history to a lookback window of 70 days for hourly data and 1 year for daily data. Further information regarding dataset details and splitting is available in Appendix C.

Baselines and Configuration. We evaluated `LLM4Series` against four established libraries, selecting specific representative models from each:

[1] `https://dadosabertos.saude.gov.br/dataset/sim`

- **Prophet (Taylor & Letham, 2018):** Using this library's native additive regression model, we enabled daily, weekly, and yearly seasonality to handle non-linear trends and multiple seasonal components.
- **Darts (Herzen et al., 2022):** From this library, we selected the `TCNModel` (Temporal Convolutional Network). It was trained for 100 epochs using the `Adam` optimizer (learning rate 1×10^{-3}), standard scaling, and an input chunk length of 96.
- **Sktime (Löning et al., 2019):** Within this framework, we utilized the `AutoARIMA` estimator. We used the default configuration for parameter optimization, explicitly setting the seasonal period (sp) to 24 for hourly data and 7 for daily series.
- **TimeCopilot (Garza & Rosillo, 2025):** An agentic framework for autonomous temporal reasoning, instantiated with GPT-5. In this configuration, the LLM acts as an orchestrator, selecting and executing a specialized forecasting model. The agent assigned `Prophet` to ETTh2, `SeasonalNaive` to Electricity, and `AutoETS` to the Mortality dataset.

`LLM4Series` Setup: We used the open-source **Qwen3-32B** (Team, 2025) model with temperature 0.7 (default) and evaluated four prompt strategies: *zero-shot*, *few-shot*, *Chain-of-Thought* (CoT), and *CoT + few-shot*. Table 1 reports, for each dataset, the results of the best-performing strategy. Data serialization was restricted to `plain` and `csv` formats, which showed the highest accuracy in prior work (Fons et al., 2024). Only textual time-series representations were used, following prior evidence of their effectiveness (Gruver et al., 2023). The code example is provided in Appendix A.

	LLM4Series			Prophet			Darts			Sktime			TimeCopilot		
Dataset	**SMAPE**	**MAE**	**RMSE**	**SMAPE**	**MAE**	**RMSE**	**SMAPE**	**MAE**	**RMSE**	**SMAPE**	**MAE**	**RMSE**	**SMAPE**	**MAE**	**RMSE**
ETTh2	**3.57**	**1.15**	**1.47**	5.78	1.94	2.48	5.06	1.56	1.95	7.25	2.51	3.30	6.48	2.17	2.61
Electricity	7.45	0.07	0.12	44.62	0.34	0.38	7.68	0.07	0.12	**6.71**	**0.06**	**0.10**	7.45	0.07	0.12
Mortality	2.82	120.09	132.95	11.66	540.41	556.76	6.18	280.18	300.30	**0.78**	**33.90**	**41.86**	0.86	37.40	44.87

Table 1: Performance metrics across different forecasting libraries

Results and Discussion. Table 1 details the performance metrics. `LLM4Series` outperformed all other baselines on ETTh2, highlighting the robustness of the automated prompt construction framework. On Electricity, it delivered competitive performance close to `Darts` and `TimeCopilot`, although `Sktime` obtained the lowest errors. For Mortality, while `Sktime` and `TimeCopilot` ranked highest, `LLM4Series` clearly outperformed `Prophet` and `Darts`. Qualitative comparisons are presented in Figure 3, visualizing the forecasts against ground-truth data.

Crucially, `LLM4Series` results reflect the best prompt strategy identified via the library's built-in optimization tools (full ablation of strategies are provided in Table 5). In contrast, baselines operated under automated (`Sktime`, `TimeCopilot`) or standard regimes. Notably, although the `TimeCopilot` agent correctly selected `Prophet` for ETTh2, it yielded higher errors (6.48 sMAPE) than our standalone implementation (5.78). This illustrates a limitation of fully autonomous agents in fine-tuning low-level hyperparameters compared to specialized frameworks. While baselines like `Darts` might improve with exhaustive search, we demonstrate that prompt engineering alone facilitates highly competitive performance.

4 Conclusion and Future Work

In this work, we introduced `LLM4Series`, a Python library designed to lower the barrier to applying Large Language Models (LLMs) to time series forecasting. By providing a unified interface, it streamlines the experimental workflow and enables controlled evaluation of LLM-driven forecasting strategies. Beyond an implementation framework, LLM4Series serves as an experimental environment for studying prompt-driven model behavior.

Integrated with the Pandas ecosystem, the library makes in-context learning approaches more accessible within the time series analysis landscape. Future work will focus on synthetic data generation for robustness evaluation, expanded evaluation metrics and data format support, and improved multivariate forecasting capabilities.

5 ACKNOWLEDGMENTS

We gratefully acknowledge the Kunumi Institute for funding this research and for providing the computational infrastructure and support for the presentation of this work.

REFERENCES

Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner TÃ¼rkmen, and Yuyang Wang. Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, 21(116):1–6, 2020. URL http://jmlr.org/papers/v21/19-820.html.

Torben G Andersen, Tim Bollerslev, Peter Christoffersen, and Francis X Diebold. Volatility forecasting, 2005.

Zairo Bastos, João David Freitas, José Wellington Franco, and Carlos Caminha. Prompt-driven time series forecasting with large language models. In *Proceedings of the 27th International Conference on Enterprise Information Systems*, pp. 309–316, 2025.

Joos-Hendrik Böse, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Dustin Lange, David Salinas, Sebastian Schelter, Matthias Seeger, and Yuyang Wang. Probabilistic demand forecasting at scale. *Proceedings of the VLDB Endowment*, 10(12):1694–1705, 2017.

Robert B Cleveland, William S Cleveland, Jean E McRae, Irma Terpenning, et al. Stl: A seasonal-trend decomposition. *J. off. Stat*, 6(1):3–73, 1990.

Johann Faouzi and Hicham Janati. pyts: A python package for time series classification. *Journal of Machine Learning Research*, 21(46):1–6, 2020.

Elizabeth Fons, Rachneet Kaur, Soham Palande, Zhen Zeng, Tucker Balch, Manuela Veloso, and Svitlana Vyetrenko. Evaluating large language models on time series feature understanding: A comprehensive taxonomy and benchmark. *arXiv preprint arXiv:2404.16563*, 2024.

João David Freitas, Caio Ponte, Rafael Bomfim, and Carlos Caminha. The impact of window size on univariate time series forecasting using machine learning. In *Symposium on Knowledge Discovery, Mining and Learning (KDMiLe)*, pp. 65–72. SBC, 2023.

Azul Garza and Renée Rosillo. Timecopilot. *arXiv preprint arXiv:2509.00616*, 2025.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew G Wilson. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36:19622–19635, 2023.

Qiuhan Gu. Llm-based code generation method for golang compiler testing. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 2201–2203, 2023.

Julien Herzen, Francesco LÃ¤ssig, Samuele Giuliano Piazzetta, Thomas Neuer, LÃ©o Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan KoÅcisz, Dennis Bader, FrÃ©dÃ©rick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and GaÃl Grosch. Darts: User-friendly modern machine learning for time series. *Journal of Machine Learning Research*, 23(124):1–6, 2022. URL http://jmlr.org/papers/v23/21-1177.html.

Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

Vladik Kreinovich, Hung T Nguyen, and Rujira Ouncharoen. How to estimate forecasting quality: A system-motivated derivation of symmetric mean absolute percentage error (smape) and other similar characteristics. 2014.

Dong Kyu Lee, Junyong In, and Sangseok Lee. Standard deviation and standard error of the mean. *Korean journal of anesthesiology*, 68(3):220–223, 2015.

Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J. Király. sktime: A unified interface for machine learning with time series, 2019. URL https://arxiv.org/abs/1909.07872.

Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

Matthew Middlehurst, Ali Ismail-Fawaz, Antoine Guillaume, Christopher Holder, David Guijo-Rubio, Guzal Bulatova, Leonidas Tsaprounis, Lukasz Mentel, Martin Walter, Patrick Schäfer, et al. aeon: a python toolkit for learning from time series. *Journal of Machine Learning Research*, 25(289):1–10, 2024.

Prapanna Mondal, Labani Shit, and Saptarsi Goswami. Study of effectiveness of time series modeling (arima) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications*, 4(2):13, 2014.

Irfan Pratama, Adhistya Erna Permanasari, Igi Ardiyanto, and Rini Indrayani. A review of missing values handling methods on time-series data. In *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, pp. 1–6, 2016. doi: 10.1109/ICITSI.2016.7858189.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International journal of forecasting*, 36(3):1181–1191, 2020.

Skipper Seabold, Josef Perktold, et al. Statsmodels: econometric and statistical modeling with python. *SciPy*, 7(1):92–96, 2010.

Robert H Shumway and David S Stoffer. Arima models. In *Time series analysis and its applications: with R examples*, pp. 75–163. Springer, 2017.

Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. Tslearn, a machine learning toolkit for time series data. *Journal of machine learning research*, 21(118):1–6, 2020.

Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Eric J Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44–56, 2019.

Artur Trindade. ElectricityLoadDiagrams20112014. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C58C86.

Wenhai Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong Lu, Jie Zhou, Yu Qiao, et al. Visionllm: Large language model is also an open-ended decoder for vision-centric tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020.

Hao Xue and Flora D Salim. Promptcast: A new prompt-based learning paradigm for time series forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):6851–6864, 2023.

Hemant Yadav and Amit Thakkar. Noa-lstm: An efficient lstm cell architecture for time series forecasting. *Expert Systems with Applications*, 238:122333, 2024. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2023.122333. URL https://www.sciencedirect.com/science/article/pii/S095741742302835X.

Kexin Zhang, Qingsong Wen, Chaoli Zhang, Rongyao Cai, Ming Jin, Yong Liu, James Y Zhang, Yuxuan Liang, Guansong Pang, Dongjin Song, et al. Self-supervised learning for time series analysis: Taxonomy, progress, and prospects. *IEEE transactions on pattern analysis and machine intelligence*, 46(10):6775–6794, 2024.

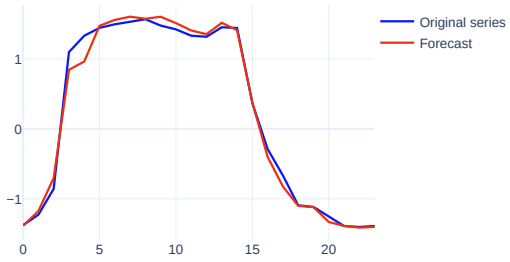
Qianru Zhang, Peng Yang, Honggang Wen, Xinzhu Li, Haixin Wang, Fang Sun, Zezheng Song, Zhichen Lai, Rui Ma, Ruihua Han, et al. Beyond the time domain: Recent advances on frequency transforms in time series analysis. *arXiv preprint arXiv:2504.07099*, 2025.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*, volume 35, pp. 11106–11115. AAAI Press, 2021.

A USAGE EXAMPLE

The code snippet below demonstrates the complete workflow for forecasting using `LLM4Series`, including data loading, model configuration, and visualization.

```
import llm4series as l4s

ts = l4s.read_file("Electricity.csv", index_col="date")
train, test = ts.split(start="2013-06-28", end="2013-09-06", periods=24)
model = l4s.LMStudio("qwen3-32b")

prompt = l4s.prompt(
    ts=train,
    tsformat="plain",
    tstype="textual",
    type="cot_few",
    sampling="uniform",
    examples=3,
    forecast_horizon=24,
    stl=train.stl(freq="h"))

data = train.to_str(format="plain", type="textual")
response = model.predict(prompt, data, temperature=0.7)

predicted = l4s.from_str(response.predicted, format="plain")
l4s.lineplot(test, predicted, groups=["Original series", "Forecast"])

metrics = test.metrics(predicted)["value"]
print(metrics["smape"], metrics["mae"], metrics["rmse"])
```



B PROMPTS

This section provides illustrative examples of the prompt structures implemented in `LLM4Series`. All prompts follow a structured design that combines the forecasting horizon, serialized historical data, descriptive statistics extracted during preprocessing, and a specific role specification (e.g., "You are an expert forecaster"). This composition establishes a quantitative context that steers the model toward pattern-based reasoning rather than unconstrained text generation.

B.1 SYSTEM PROMPT (EXPERT PERSONA)

The system prompt defines the operational constraints and the technical persona, ensuring the model acts as a rigorous statistical engine.

System Prompt (Expert Persona)

You are a specialist in statistical modeling and machine learning, with expertise in time series forecasting.

Constraints:
1. The forecast must start immediately after the last observed timestamp.
2. Provide only predicted values. No prose, explanations, or code.
3. Enclose the output strictly within `<out></out>` tags.
4. Maintain the input's exact data representation and grammar. If a header is present in the input, it MUST be replicated in the output.

B.2 INFERENCE STRATEGIES

The library supports four primary inference strategies to leverage different LLM capabilities:

1. Zero-Shot (ZS): Prompts the model to generate forecasts using only the structured context derived from the target series.

Zero-Shot Template

Objective: Predict the next `{forecast_horizon}` values based on the historical series (`{input_len}` data points).
Statistical Context: `{statistics}`
Execution Steps:
1. Analyze the series trend and seasonality internally.
2. Generate the forecast for the next `{forecast_horizon}` steps.
3. Output the result inside `<out></out>` tags.
Input Data: `{input_series}`

2. Few-Shot (FS): Extends the prompt with example input–output forecasting pairs to demonstrate the expected output structure and prediction behavior.

Few-Shot Template

...
Examples: `{forecast_example}`
Input Data: `{input_series}`

3. Chain-of-Thought (CoT): Introduces guided reasoning instructions, encouraging the model to internally analyze structural characteristics before producing the forecast.

CoT Reasoning Block

Reasoning Instructions:
Before generating the forecast, analyze the historical series step by step, considering:
- *Trend:* Identify the overall direction (increasing, decreasing, stable) and the trend strength.
- *Seasonality:* Patterns that repeat at regular intervals (e.g., daily, weekly, monthly).
- *Outliers:* Possible outliers or abrupt changes.
- *Cycles:* Not seasonal long-term patterns.
- *Noise reduction:* Apply a technique to reduce noise when necessary.
- Consistency with the provided descriptive statistics (mean, median, etc.).
- Adjustment for data frequency and contextual events (holidays, promotions, etc.).

4. CoT + Few-Shot: Combines both approaches by providing demonstration examples alongside explicit reasoning guidance.

Example CoT+FS

Objective: Predict the next 7 values based on the historical series (364 data points).
Statistical Context:
- *Mean*: 70.1202
- *Median*: 47.26
- *Standard Deviation*: 47.772
- *Minimum Value*: 0.0
- *Maximum Value*: 243.18
- *First Quartile (Q1)*: 36.8787
- *Third Quartile (Q3)*: 102.9538
- *Trend Strength (STL)*: 0.0244
- *Seasonality Strength (STL)*: 0.8829

Example CoT+FS

Reasoning Instructions:
Before generating the forecast, analyze the historical series step by step, considering:

- *Trend:* Identify the overall direction (increasing, decreasing, stable) and the trend strength.
- *Seasonality:* Patterns that repeat at regular intervals (e.g., daily, weekly, monthly).
- *Outliers:* Possible outliers or abrupt changes.
- *Cycles:* Not seasonal long-term patterns.
- *Noise reduction:* Apply a technique to reduce noise when necessary.
- Consistency with the provided descriptive statistics (mean, median, etc.).
- Adjustment for data frequency and contextual events (holidays, promotions, etc.).

Execution Steps:
1. Analyze the series trend and seasonality internally.
2. Generate the forecast for the next `7` steps.
3. Output the result inside `<out></out>` tags.

Examples:
- Example 1:
Input (history):
date: 2017-01-01 00:00:00, value: 0 . 0
date: 2017-01-02 00:00:00, value: 1 5 8 . 1 2 0
date: 2017-01-03 00:00:00, value: 1 5 9 . 4 3 0
date: 2017-01-04 00:00:00, value: 3 8 . 3 5 0
date: 2017-01-05 00:00:00, value: 3 8 . 7 8 5
date: 2017-01-06 00:00:00, value: 3 0 . 5 5 0
date: 2017-01-07 00:00:00, value: 5 2 . 0 6 5

Output (forecast):
<out>
date: 2017-01-08 00:00:00, value: 4 4 . 0 2 9
date: 2017-01-09 00:00:00, value: 1 2 0 . 8 6 0
date: 2017-01-10 00:00:00, value: 1 0 7 . 6 2 0
date: 2017-01-11 00:00:00, value: 2 6 . 7 1 0
date: 2017-01-12 00:00:00, value: 2 7 . 3 3 5
date: 2017-01-13 00:00:00, value: 3 3 . 7 2 5
date: 2017-01-14 00:00:00, value: 4 5 . 9 0 5
</out>

- Example 2: ...
- Example 3: ...

Input Data:
date: 2017-06-25 00:00:00, value: 5 3 . 9 6 4
date: 2017-06-26 00:00:00, value: 1 2 9 . 0 6 5
date: 2017-06-27 00:00:00, value: 1 5 3 . 6 5 4
date: 2017-06-28 00:00:00, value: 3 0 . 3 3 5
date: 2017-06-29 00:00:00, value: 3 1 . 7 2 5
date: 2017-06-30 00:00:00, value: 4 9 . 5 5 9
date: 2017-07-01 00:00:00, value: 9 5 . 6 9 5
...

C EXPERIMENTS

C.1 DATASET DETAILS AND TRAINING SPLITS

Table 2 details the temporal granularity, target variables, and specific train-test intervals adopted for Qwen3-32B. For the baseline libraries evaluated in this study, as described in the main paper, we used the full training set up to the forecasting horizon, with the exception of `AutoARIMA`, whose configuration follows the procedure previously outlined.

In contrast, for Qwen3-32B, the limited context window of frontier LLMs requires restricting the amount of historical data provided at inference time. To address this constraint, we adopt a selection method based on the **Continuous Wavelet Transform (CWT)** to choose the most relevant data intervals. This approach allows us to identify periods that contain rich patterns and structural changes across different scales, which might be missed by simple truncation (Zhang et al., 2025). We standardized the evaluation windows to 70 days for hourly data and one year for daily data. This setup ensures that we can compare results fairly across different datasets while staying within token limits and maintaining consistency with other baseline forecasting libraries.

Dataset	#Vars	Timestamps	Freq.	Target	Horizon	Training Start	Split (End Train)	End Test
ETTh2	7	17420	Hourly	OT	24	2016-07-12 06:00	2016-09-20 05:00	2016-09-21 05:00
Electricity	321	26304	Hourly	channel_293	24	2013-06-28 07:00	2013-09-06 06:00	2013-09-07 06:00
Mortality	1	2191	Daily	value	7	2021-02-23 00:00	2022-02-21 00:00	2022-02-28 00:00

Table 2: Summary of the datasets used in the experiments, including intrinsic time series characteristics.

C.1.1 Context Window Selection

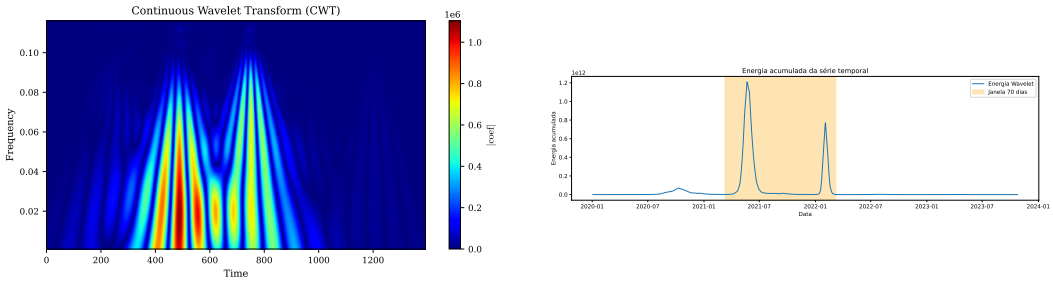


Figure 2: Time–Frequency Scalogram obtained via the Continuous Wavelet Transform.

All wavelet computations are implemented in Python using the `PyWavelets` library. The following code snippet illustrates the computation of wavelet coefficients, energy aggregation, and window selection:

```
import numpy as np
import pywt

# Configuration
wavelet = "db4"
sampling_period = 7  # Weekly seasonality
scales = np.arange(1, 128)
# Continuous Wavelet Transform
coefficients, frequencies = pywt.cwt(signal, scales, wavelet,
    sampling_period)
# Energy computation
energy = np.abs(coefficients) ** 2
energy_per_time = np.sum(energy, axis=0)
# Window selection
window_size = L
window_energy = np.convolve(energy_per_time, np.ones(window_size), mode="
    valid")
best_window_start = np.argmax(window_energy)
```

C.2 Evaluation Metrics

We evaluate forecasting performance using four standard error metrics. Let y_t represent the observed series, $\hat{y}_t$ the model forecast, and n the length of the evaluation horizon.

1. **Mean Absolute Error (MAE)**(1): Measures the average absolute deviation between predicted and observed values, providing a scale-dependent assessment of forecasting accuracy (Hyndman & Athanasopoulos, 2018).

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t| \tag{1}$$

2. **Root Mean Squared Error (RMSE)**(2): Quantifies the square root of the mean squared deviations, penalizing larger errors more heavily due to the quadratic term (Hyndman & Athanasopoulos, 2018).

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{2}$$

3. **Symmetric Mean Absolute Percentage Error (SMAPE)**(3): Computes a normalized percentage-based error by scaling absolute deviations relative to the average magnitude of actual and predicted values, enabling comparison across series with different scales (Kreinovich et al., 2014).

$$SMAPE = \frac{100\%}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{(|y_t| + |\hat{y}_t|)/2} \tag{3}$$

4. **Standard Error of the Mean (SEM)**(4): Used **only** for Qwen3-32B to quantify variability across five independent inference runs. SEM reflects the uncertainty of the reported average performance (Lee et al., 2015).

$$SEM = \frac{s}{\sqrt{k}} \tag{4}$$

C.3 Detailed Hyperparameters

Table 3 summarizes the configurations for all models used in our comparative study. While Section 3 provides a high-level overview, here we specify exact parameters to ensure reproducibility.

Library/Model	Hyperparameters & Settings
Prophet	Seasonality: Daily, Weekly, Yearly; Growth: Linear
Darts (TCN)	Epochs: 100; Opt: Adam ($1e-3$); Input Chunk: 96; Scaling: Standard
Sktime (AutoARIMA)	sp: 24 (hourly), 7 (daily); Auto-search: Enabled
TimeCopilot	Engine: GPT-5; Orchestration: Default Agentic Policy
LLM4Series (Qwen3-32B)	Temp: 0.7; Sampling: 5 runs; Format: `plain/csv`

Table 3: Model configurations and training parameters.

C.4 Infrastructure and Reproducibility

All experiments were conducted on a dedicated local workstation running Ubuntu 24.04.3 LTS (Kernel 6.14.0). For Qwen3-32B, inference was executed via `LM Studio` with full GPU offloading.

Component	Specification
CPU	AMD Ryzen 7 8700G with Radeon 780M Graphics (16 threads), 5.177 GHz
Memory	64 GB system RAM
GPU	NVIDIA RTX PRO 6000 Blackwell Workstation Edition, 96 GB GDDR6 VRAM, 24,064 CUDA cores

Table 4: Hardware specifications used in the experiments.

		ETTh2			Electricity			Mortality		
Prompt	**Format**	**SMAPE**	**MAE**	**RMSE**	**SMAPE**	**MAE**	**RMSE**	**SMAPE**	**MAE**	**RMSE**
Zero-shot	CSV	13.99±3.44	4.83±1.30	5.26±1.27	7.53±0.11	0.07±0.00	0.12±0.00	**2.82±0.88**	**120.09±36.02**	**132.95±38.76**
	Plain	12.67±2.67	4.20±0.89	4.98±1.02	10.09±1.07	0.09±0.01	0.14±0.01	4.75±2.02	217.20±94.15	251.10±110.51
Few-shot	CSV	6.52±1.79	2.15±0.60	2.46±0.63	**7.45±0.00**	**0.07±0.00**	**0.12±0.00**	4.61±1.20	193.66±49.02	210.68±52.66
	Plain	5.09±0.57	1.66±0.20	1.94±0.24	8.95±1.53	0.08±0.01	0.13±0.01	4.84±1.29	202.66±52.37	221.12±56.09
CoT	CSV	11.24±3.43	3.87±1.23	4.37±1.37	10.05±1.02	0.09±0.01	0.14±0.01	5.27±2.10	236.86±101.45	260.32±113.02
	Plain	10.80±2.02	3.66±0.71	4.19±0.67	14.20±4.65	0.12±0.04	0.22±0.08	4.83±1.66	217.74±77.13	248.23±90.21
CoT + Few-shot	CSV	4.66±0.88	1.47±0.29	1.68±0.29	10.59±1.83	0.09±0.01	0.15±0.02	8.38±1.47	360.71±70.20	385.85±75.30
	Plain	**3.57±0.21**	**1.15±0.10**	**1.47±0.11**	13.06±1.48	0.11±0.01	0.17±0.01	5.51±1.22	229.91±50.18	251.43±52.22

Table 5: Performance metrics for experiments on the ETTh2, Electricity, and Mortality datasets using Qwen3-32B.

Table 5 presents the results, offering several key insights into LLM behavior for forecasting:

1. **Complexity requires Reasoning:** For the **ETTh2** dataset, which exhibits complex nonlinear patterns and high volatility, the hybrid **CoT-Few** strategy significantly outperformed simpler approaches. This suggests that for intricate temporal dynamics, the model benefits from both in-context examples (to grasp the output distribution) and intermediate reasoning steps (to analyze trend direction).
2. **Simplicity favors Zero-Shot:** Conversely, on the **Mortality** dataset, the **Zero-Shot** strategy achieved the lowest error (sMAPE 2.82). We hypothesize that for short, noisy series with lower frequency (daily), adding complex reasoning instructions or few-shot examples might introduce noise or cause the model to overfit to unrepresentative examples in the prompt, leading to poorer generalization.
3. **Impact of Serialization:** Consistent with prior literature (Gruver et al., 2023), the `plain` text format generally yielded lower errors compared to `csv`, particularly in the high-performing CoT-Few configurations. The space-delimited format appears to facilitate better tokenization of numerical magnitudes for the Qwen tokenizer compared to the density of comma-separated strings.

Based on these findings, the main results reported in Section 3 utilize the best-performing configuration per dataset: `CoT-Few/plain` for ETTh2, `Few-Shot/csv` for Electricity, and `Zero-Shot/csv` for Mortality.

C.4.1 Qualitative Forecast Comparison

To visually assess the quality of the predictions, Figure 3 contrasts the forecasts generated by `LLM4Series` against the ground truth and baseline models.

- **ETTh2 (Left Panel):** The plot highlights the robustness of our framework (red line). While statistical baselines (e.g., Prophet) struggle to capture the sharp peak at the end of the window, `LLM4Series` closely follows the ground truth trajectory, correctly identifying the inflection point.

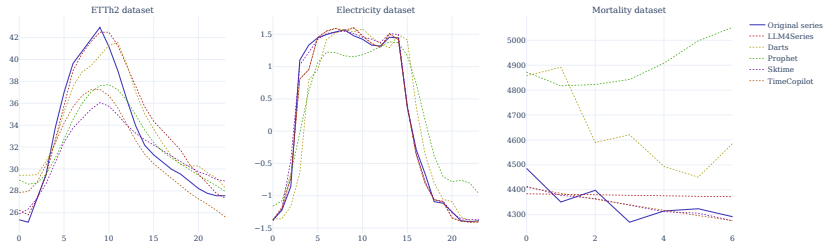


Figure 3: Forecasting results for the ETTh2, Electricity, and Mortality datasets. The figure contrasts the original time series with forecasts produced by `LLM4Series`, `Darts`, `Prophet`, `Sktime`, and `TimeCopilot`.

- **Electricity (Center Panel):** Qwen3-32B successfully captures the daily seasonality and the magnitude of the peaks. The alignment with the ground truth is comparable to the deep learning baseline (Darts), demonstrating that the LLM can effectively parse periodic structures.
- **Mortality (Right Panel):** This dataset poses a challenge due to its lack of strong seasonality. Notably, some baselines (e.g., TimeCopilot/Prophet) "hallucinate" a strong upward or downward trend that does not exist. `LLM4Series` maintains a conservative prediction that aligns closer to the stable nature of the actual data, resulting in the lower sMAPE observed in the quantitative analysis.

D Limitations and Future Work

Although `LLM4Series` establishes a systematic framework for LLM-based forecasting, we acknowledge certain limitations in the current experimental scope which define our immediate roadmap for future development:

- **Experimental Evaluation:** Our experiments focused on establishing the validity of the framework. Consequently, while the library's `ModelResponse` object natively captures execution metadata, we did not provide an extensive analysis of token consumption and inference latency. Future works will leverage these built-in tools to provide a comprehensive cost-benefit analysis of different LLM backends.
- **Data Contamination:** This study did not explicitly investigate the impact of data contamination in common public datasets. However, `LLM4Series` is designed precisely to facilitate such investigations; future work will involve integrating synthetic data generation modules to evaluate model performance on "unseen" temporal dynamics.
- **Robustness to Malformed Outputs:** While our tag-based extraction mechanism is effective, it remains sensitive to model hallucinations. Improving the robustness of the output parser is a priority for upcoming releases.