Forecasting Time Series with LLMs via Patch-Based Prompting and Decomposition

Mayank Bumb, Anshul Vemulapalli, Sri Harsha Jella, Anish Gupta, An La, Ryan Rossi, Franck Dernoncourt, Hongjie Chen, Nesreen Ahmed, Yu Wang

Proceedings of the 39th Pacific Asia Conference on
Language, Information and Computation (PACLIC 39)

Emmanuele Chersoni, Jong-Bok Kim (eds.)

2025

Forecasting Time Series with LLMs via Patch-Based Prompting and Decomposition

Mayank Bumb[1], **Anshul Vemulapalli**[1], **Sri Harsha Jella**[1], **Anish Gupta**[1], **An La**[1], **Ryan Rossi**[2], **Franck Dernoncourt**[2], **Hongjie Chen**[3], **Nesreen Ahmed**[4], **Yu Wang**[5]
[1]University of Massachusetts Amherst, [2]Adobe, [3]Dolby Labs, [4]Intel, [5]University of Oregon

Abstract

Recent advances in Large Language Models (LLMs) have demonstrated new possibilities for accurate and efficient time series analysis, but prior work often required heavy fine-tuning and/or ignored inter-series correlations. In this work, we explore simple and flexible prompt-based strategies that enable LLMs to perform time series forecasting without extensive retraining or the use of a complex external architecture. Through the exploration of specialized prompting methods that leverage time series decomposition, patch-based tokenization, and similarity-based neighbor augmentation, we find that it is possible to enhance LLM forecasting quality while maintaining simplicity and requiring minimal preprocessing of data. To this end, we propose our own method, PatchInstruct, which enables LLMs to make precise and effective predictions.

1 Introduction

Time-series forecasting (TSF) has a broad range of applications in agriculture, business, epidemiology, finance, etc. Many of these applications require robust predictions of time series, and accurately modeling the dependencies between variables remains to be a challenge (Shao et al., 2020). Traditional forecasting models such as ARIMA, LSTMs, and even Transformer/Graph-based architectures have displayed a strong performance on these tasks (Zhou et al., 2024).

More recently, Large Language Models (LLMs) have shown a promising future in modeling time series, with accurate predictions that rival state of the art (SOTA) methods, due to their strengths in pattern recognition, sequence modeling, and generalization across tasks. However, current LLM-based methods often rely on complex architectures or require heavy fine-tuning, limiting their scalability to real-world applications.

One prominent approach, S^2IP-LLM (Pan et al., 2024), embeds time series into a semantic space to enhance forecasting performance. While effective, it introduces two key limitations. First, it incurs a high computational cost during inference due to its reliance on complex decomposition and patching pipelines. Second, it does not explicitly model dependencies across related time series, which can be critical in domains such as traffic and energy forecasting where inter-series relationships play a significant role.

We aim to develop a method (see Figure 1) that maintains the predictive strength of LLM-based models while addressing the above limitations of inference speed and generalization. Therefore we guide our experimentation around the idea of whether we can create general-purpose prompts that guide LLMs to forecast time series both accurately and efficiently, without requiring model fine-tuning or architectural changes.

To this end, we introduce PatchInstruct, a prompt-based framework that tokenizes time series data into meaningful patches that encapsulate temporally relevant patterns and guides the LLM via structured natural language instructions to output precise predictions. Unlike prior work, PatchInstruct requires no model retraining or architecture modification and also significantly reduces inference time (in comparison to the baseline and complex architectures) alongside token usage while preserving or improving accuracy.

We compare PatchInstruct with several other prompting strategies—including Zero-shot, Neighbors, and PatchInstruct + Neighbors—and evaluate them on diverse, real-world datasets (Weather and Traffic), primarily using GPT-4 and GPT-4o as the LLM backbones.

Across the datasets and small forecasting horizons we study ($H \leq 12$), PatchInstruct is typically the most accurate among our baselines in MSE/-MAE and, in our setup, reduces inference overhead by 10x–100x compared to S^2IP-LLM while maintaining comparable accuracy. Neighbor augmen-

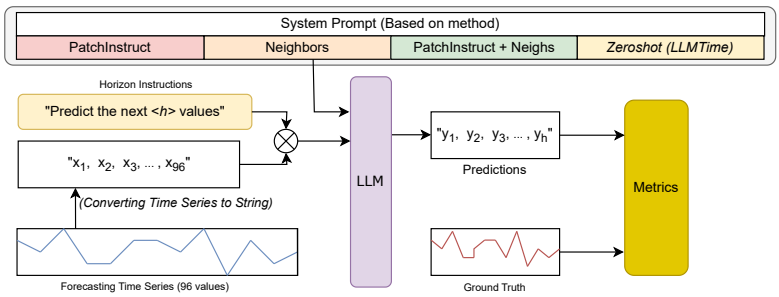

Figure 1: LLM-based Time-Series Forecasting Pipeline

tation is dataset-dependent: it helps on Weather, where retrieved series are highly informative, but can underperform on datasets with weaker cross-series alignment.

Taken together, these results indicate that careful prompt design can substitute for portions of model-specific architecture in our tested setting, enabling scalable and domain-adaptable time-series forecasting with LLMs. Our evaluations standardize on GPT for output-format reliability (other backbones frequently violated the required H-length outputs in pilots), and we fix the neighbor count to ($k = 5$) to fit a 50k input-token budget; broader cross-backbone and budget studies are left for future work.

2 Related Work

2.1 Time Series Foundation Models

Foundation Models (FMs)—large pre-trained models that learn general-purpose representations—have propelled state-of-the-art results in NLP and CV, and the same paradigm is increasingly adapted to time series (Shi et al., 2024). Liang *et al.* provide a useful taxonomy for Time Series Foundation Models (TSFMs) along four axes: data category (standard, spatial, or trajectory/event), model architecture (Transformer-, non-Transformer-, or diffusion-based), pre-training strategy (self-supervised vs. supervised), and application domain (Liang et al., 2024).

While diverse architectures exist, Transformer backbones remain prevalent due to their ability to capture long-range dependencies in sequential data (Miller et al., 2024). Diffusion-style foundations have also emerged, e.g., TimeDiT, which marries diffusion objectives with Transformer blocks for time series analysis (Cao et al., 2024b). Among open and commercial TSFMs, models such as Lag-Llama demonstrate that large-scale pretraining across heterogeneous collections improves adaptability and zero/few-shot forecasting quality (Rasul et al., 2023). In practice, the choice of pre-training signal (contrastive, masked reconstruction, forecasting-style objectives) and coverage of data domains strongly influences downstream generalization.

2.2 LLMs for Time Series Forecasting

Large Language Models (LLMs) have recently been explored for time series by casting numerical forecasting as a language problem via tokenization, prompting, and in-context learning. Foundational work on Transformer-based forecasting for raw continuous inputs includes TST (Zerveas et al., 2020), which applies a Transformer encoder to multivariate sequences. Subsequent advances such as PatchTST (Nie et al., 2022) introduce *patching*—partitioning series into localized segments (patch tokens)—and channel-independence, which together improve efficiency and accuracy. Thus, PatchTST *builds on* earlier Transformer formulations like TST; our discussion reflects this chronology.

A parallel line of research converts real-valued series into discrete tokens to better leverage the next-token prediction strengths of LLMs. Chronos (Ansari et al., 2024) quantizes and scales observations into a fixed vocabulary to enable zero-shot and transfer settings. Digit-level tokenization treats each numeric digit as a token, aligning forecasting with language modeling mechanics (Gruver et al., 2024). Prompt-based formulations go further: PromptCast frames forecasting as sentence-to-sentence generation with task-specific prompts (Xue and Salim, 2023), and GPT4TS demonstrates that a single LLM can address forecasting, anomaly detection, and classification using textual prompts alone (Zhou et al., 2023).

Despite these advances, robustness on hetero-

geneous, irregular, and partially observed series remains challenging. LLM4TS proposes a two-stage pipeline that first aligns pretrained LLMs to standardized time series structures and then fine-tunes for forecasting (Chang et al., 2024). Decomposition-aware prompting (e.g., TEMPO) explicitly models trend/seasonal/residual components to improve interpretability and performance (Cao et al., 2024a). Hybrid systems integrate spatial and relational biases—TPLLM fuses CNNs/GCNs with LLMs for traffic prediction (Ren et al., 2024), while GenTKG combines retrieval-augmented generation with parameter-efficient tuning for temporal knowledge graphs (Liao et al., 2024).

In contrast to methods that rely on heavy fine-tuning or complex multi-component stacks, our work (PATCHINSTRUCT) adopts a training-free prompting framework: we decompose inputs into compact *patch* segments and instruct LLMs directly. This minimizes token usage and implementation overhead while preserving competitive accuracy—particularly on short-horizon forecasts—across diverse domains.

3 Methodology

We propose an approach that leverages Large Language Models (LLMs) for time series forecasting through specialized prompt engineering techniques that eliminates the need for model fine-tuning or architectural modifications.

Our approach begins with a zero-shot baseline, inspired by TimeLLM (Gruver et al., 2024), where the model is prompted with raw historical time series values and tasked with predicting future values. While this baseline offers simplicity and generality, it lacks the inductive bias necessary to capture local temporal dynamics, leading to suboptimal performance in complex forecasting settings. Our second baseline S^2IP-LLM introduces significant inference-time overhead due to its reliance on complex decomposition pipelines and fine-tuning, limiting its scalability in real-world deployments.

To address these limitations, we introduce PatchInstruct (see Figure 2), a prompting strategy that encode temporal structure through patch-based representations, and provide pretrained LLMs more context on the dataset. The core idea is to decompose a time series into fixed-length overlapping patches and provide them to the LLM in a structured format, along with instructions to predict future values.

Additionally, we experimented the model's forecasting ability by supplementing the target time series with a small set of similar time series referred to as Neighbors (Neighs). Specifically, we select the five most similar time series from the dataset's past seen data, referred to as neighbors. The motivation behind this approach is to provide the LLM with additional contextual signals and recurring patterns that may not be fully observable in the target series alone.

We finally also tested a combination of these the Patch-Instruct and Neighbors strategy, enriching the prompt with structurally decomposed information from the target series (via patching), while augmenting it with relevant patterns from similar series (via nearest neighbors).

In the following subsections, we detail the construction of each approach, describe the datasets and evaluation metrics used, and present a comparative analysis of their forecasting performance.

We evaluate our method on two time series datasets: Weather and Traffic. These datasets comprised of continuous measurements sampled at regular intervals. A 96-timestep input window is used to forecast future horizons of 1,2,3,4,5,6 and 12 steps.

3.1 Overview of Framework

Our framework is designed to adapt large language models (LLMs) for time series forecasting without any fine-tuning, using carefully structured prompts that condition the model with temporal data and forecasting instructions. The pipeline is modular and supports multiple prompting strategies, including PatchInstruct, Neighbors, and PatchInstruct + Neighbors, and Zeroshot by modifying the structure of the system prompt and the input representation.

At inference time, a raw time series is converted into a sequence of string-formatted numerical values. Depending on the method, additional transformations are applied—for example, decomposing the sequence into overlapping fixed-length patches (in PatchInstruct), or retrieving similar time series (in Neighs). These inputs are concatenated with forecasting instructions (e.g., "Predict the next h values") and passed to the LLM. The output is parsed into a numerical forecast and compared with the ground truth using standard forecasting metrics.

In addition to forecasting, PatchInstruct also

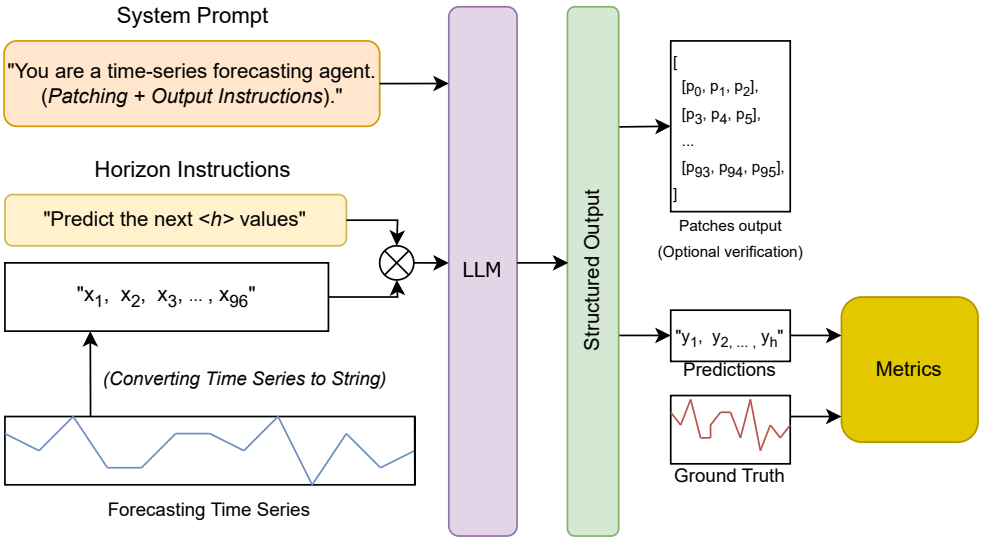

Figure 2: PatchInstruct Forecasting Pipeline

prompts the model to output reconstructed patches from the input, enabling an optional interpretability step. These predicted patches can be compared with the actual ones to assess whether the LLM is learning meaningful temporal structure and capturing local dynamics, thus providing deeper insight into the model's understanding of the task.

3.2 Prompt Design

We now delve deeper into the specific construction of each prompting strategy. This section provides detailed formulations illustrating how time series data, patching instructions, and neighboring trends are encoded within the input to the LLM. The prompts were designed through rigorous empirical testing to ensure clarity and effectiveness. Each prompt consists of a system prompt, which defines the forecasting method and describes how we construct the series, and a user prompt, which contains the actual time series data provided to the LLM for prediction.

PatchInstruct is built upon the zeroshot prompt inspired by LLMTime, this method decomposes the time series into patches and the LLM uses them to form predictions. Below, we outline the structure of the best Patch-Instruct prompt. Additional experiments exploring alternative patching strategies are presented in the Appendix A. The following prompt is a user prompt used for most methods to specify the horizon, and give the context window to the LLM.

Horizon Prompt (Input Time Series)

Continue the following sequence without producing any additional text. Sequence: $<x_1, x_2, x_3, ..., x_{96}>$. Predict the next 3 values.

PatchInstruct System Prompt

You are a forecasting assistant that sees time series data. The sequence represents the total regional humidity measured every 10 minutes. Task: (1) Split the series into overlapping patches with window size 3 and stride 1. (2) Generate the patches in natural order, then reverse the list so the most recent patch appears first. (3) Use these patch tokens to forecast the next 3 values.
Output format:

```
Patches:
[[latest_patch], ..., [oldest_patch]]
Prediction:
[y1, y2, y3]
```

No headings or extra words. Decimals ≤ 4 places; keep leading zeros (e.g., `0.8032`).

For the System prompt, we included both the patching instructions and dataset-specific information, such as the name of the series. Among the variants of prompts tested, we found reverse patching to perform the best. Further details and comparisions of patching strategies are provided in (Section A; see appendix for Table 5).

Neighs is built upon our zero-shot prompt. This method adds closest neighboring series in terms of

euclidean distance over all the past windows of data and construct a composite prompt by giving all the 5 neighboring prompts as additional context. We outline the structure of the Neighs prompt used for the weather dataset in the "Neighs System Prompt" box. We specified the number of neighbors that, and gave the model additional instructions.

PatchInstruct+Neighs integrates the strengths of both PatchInstruct and Neighs approaches. We combined the two methods using the system prompt in the "PatchInstruct+Neighs system prompt" box.

Neighs System Prompt

You are a forecasting assistant that sees time series data. The sequence represents the total regional humidity measured every 10 minutes. You will also be given 5 neighbor time-series similar to the one to forecast. Use it to understand the trends.
Output format: `[y1, y2, y3]`
No headings or extra words. Decimals ≤ 4 places; keep leading zeros (e.g., `0.8032`).

PatchInstruct + Neighs System Prompt

You are a forecasting assistant that sees time series data. The sequence represents the total regional humidity measured every 10 minutes. You will also be given 5 neighbor time-series similar to the one to forecast. Use it to understand the trends.
Task: (1) Split the series into overlapping patches with window size 3 and stride 1. (2) Generate the patches in natural order, then reverse the list so the most recent patch appears first. (3) Use these patch tokens to forecast the next 3 values.
Output format:

```
Patches:
[[latest_patch], ..., [oldest_patch]]
Prediction:
[y1, y2, y3]
```

No headings or extra words. Decimals ≤ 4 places; keep leading zeros (e.g., `0.8032`).

In summary, these prompt-based variants allow us to systematically assess the impact of explicit instructions for time series decomposition, patching, and neighbor augmentation within LLM-based forecasting frameworks. The results, presented in Table 4, provide a comparative analysis of these prompting strategies.

3.3 Evaluation

We evaluate forecasting performance using Mean Squared Error (MSE), Mean Absolute Error (MAE). Runtime Efficiency and Input/Output token usage.

4 Experiments

In this section, we design experiments to investigate our proposed patch instruct framework.

4.1 Backbone selection

PatchInstruct is training-free and works with any instruction-following LLM in principle. In practice, our pilot runs compared GPT, Gemini, and Llama. Both Gemini and Llama exhibited inconsistent decoding for multi-step forecasting—specifically, failing to return exactly H predictions at a given horizon despite explicit instructions and formatting templates. These output-format errors impeded fair comparison and large-scale evaluation. We therefore standardize on GPT for all reported results to ensure consistent generation of horizon-length prediction vectors.

4.2 Datasets

We evaluate our approach on two real-world datasets: Weather and Traffic. Weather captures fast-changing environmental conditions, while Traffic reflects urban flow patterns with spatial and temporal dependencies

The Weather dataset is collected from a meteorological station at the Max Planck Institute for Biogeochemistry (Jena, Germany). It contains 14 meteorological features, including temperature, humidity, and atmospheric pressure measurements. The high-frequency recordings capture intricate weather dynamics critical for testing short-term forecasting precision.

The Traffic dataset consists of sensor network data from Los Angeles, collected between March and June 2012. It records traffic flow rates and congestion patterns across urban arteries. The spatial-temporal correlations in this dataset test the model's ability to capture complex topological dependencies in transportation systems.

We summarize key statistics in Table 1. This selection provides systematic coverage of (1) different sampling frequencies, (2) variable sequence lengths, and (3) heterogeneous feature

interactions—three critical axes for stress-testing tokenization strategies in temporal learning tasks. The datasets' public availability ensures reproducibility, while their domain diversity demonstrates our method's generalizability beyond narrow application contexts.

Dataset	Features	Frequency	Time Span	Samples	Value Range
WEATHER	14	10 minutes	3 years	157,680	0.5–18.13
TRAFFIC	181	Hourly	4 months	34,172	2.5–70

Table 1: Summary of datasets used in our experiments.

4.3 Main Results

For our experiments, we adopt S^{2}IP-LLM as the primary baseline, a method that aligns time series embeddings with the semantic space of a pre-trained LLM through a tokenization framework. While effective, S^{2}IP-LLM suffers from significant computational overhead, requiring extensive training and inference time due to its fine-tuning of LLM components. All methods are evaluated in a consistent zero-shot setting without model retraining to isolate the impact of prompting strategies.

Using various prompting strategies, we instructed a pre-trained LLM to consider time-series patches and utilize them for forecasting without any additional fine-tuning or retraining. Our experiments demonstrate that such patch-based prompting methods can significantly improve forecasting performance across multiple datasets and over shorter horizons. In contrast to models like S^{2}IP-LLM, which rely on explicit decomposition, semantic alignment, and parameter tuning, our approach leverages instruction-tuned LLMs. Among all the strategies evaluated the PatchInstruct technique consistently delivered the best results. This suggests that prompting pre-trained LLMs with thoughtfully structured temporal context can match or even surpass models trained from scratch, offering a lightweight yet effective alternative for time-series forecasting.

Table 3 presents a performance comparison between S^{2}IPLLM (the baseline) and our best-performing Patch Instruct method across multiple time series datasets and forecast horizons. The results clearly indicate that Patch Instruct consistently outperforms the baseline in terms of both MSE and MAE. Finally, Table 3 compares the two methods in terms of input/output token counts and computation time. The analysis reveals that Patch Instruct not only improves forecasting accuracy but also significantly reduces computational overhead. Overall, this comparison highlights the efficiency and effectiveness of the Patch Instruct method.

Dataset	Horizon	S^{2}IP-LLM		Zeroshot		PatchInstruct	
		MSE	MAE	MSE	MAE	MSE	MAE
WEATHER	1	0.0095	0.056	0.0028	0.043	**0.0014**	**0.029**
	2	0.017	0.077	0.0085	0.072	**0.0076**	**0.067**
	3	0.0238	0.0875	**0.0106**	**0.068**	0.0110	0.085
	4	0.0326	0.1051	**0.0115**	**0.085**	0.0236	0.113
	5	0.0371	0.1120	0.0277	0.1	**0.0159**	**0.094**
	6	0.0439	0.1228	0.0204	0.11	**0.0101**	**0.083**
	12	0.0904	0.1823	0.1098	0.221	**0.0436**	**0.137**
TRAFFIC	1	21.0814	**2.4067**	43.49	3.18	**20.05**	2.76
	2	24.0935	2.4919	23.47	2.53	**9.38**	**1.89**
	3	29.9573	2.7849	22.38	2.54	**6.47**	**1.78**
	4	29.8382	2.6147	27.50	2.87	**11.15**	**1.89**
	5	36.0289	2.7971	34.27	3.20	**8.46**	**1.88**
	6	42.3193	3.0444	29.66	3.07	**25.59**	**2.75**
	12	**68.7149**	**3.8473**	296.20	7.72	235.75	5.89

Table 2: Results comparing our approach to baselines.

Dataset	Horizon	S^{2}IP-LLM			Zeroshot			PatchInstruct		
		Time (s)	IT	OT	Time (s)	IT	OT	Time (s)	IT	OT
WEATHER	1	535.42	7	1	1.36	7370	80	1.24	8500	80
	2	518.45	7	2	1.06	7370	120	1.20	8500	120
	3	533.73	7	3	1.20	7370	160	1.01	8500	160
	4	518.37	7	4	1.01	7370	200	1.16	8500	196
	5	537.85	7	5	1.14	7370	240	1.29	8500	216
	6	522.43	7	6	1.35	7370	280	2.05	8500	280
	12	558.67	7	12	1.44	7370	520	1.51	8500	499
TRAFFIC	1	50.94	7	1	2.59	7360	80	1.31	7950	86
	2	52.88	7	2	2.04	7360	116	1.05	7950	134
	3	55.34	7	3	1.17	7360	160	1.11	7950	185
	4	51.00	7	4	2.13	7360	200	1.17	7950	223
	5	49.96	7	5	1.14	7360	240	1.12	7950	239
	6	50.11	7	6	1.38	7360	268	1.23	7950	336
	12	52.66	7	12	1.78	7360	520	1.36	7950	558

Table 3: Token and Time Comparison for Forecasting.

4.4 Cost vs. Performance Analysis

Our instruction-based forecasts deliberately spend more input tokens than the baseline S^{2}IP-LLM. Across the prompt variants, a single prediction consumes about $\approx$ 800 - 1000 input tokens. By contrast, S^{2}IP-LLM needs only the horizon-length of output tokens once its patch encoder has been trained. The extra prompt length therefore represents about a 100 times increase in in front-loaded cost.

Because our method relies on an already-trained LLM and does no task-specific fine-tuning, the end-to-end latency of producing a forecast collapses from minutes to just seconds. For example, on the Weather dataset at horizon $= 1$, S^{2}IP-LLM requires 535 s, whereas Reverse Patch returns the prediction in 0.86 (see Table 3). Thus, even after accounting

for the larger prompt, our approach is two to three orders of magnitude faster in real-time settings.

The additional 800–900 input tokens result in a substantial improvement in short-range forecasting accuracy. On Weather (H=1), mean squared error (MSE) drops from 1.15×10^{-2} to 2.6×10^{-4} (a 97.7% reduction), and mean absolute error (MAE) decreases from 6.52×10^{-2} to 1.4×10^{-2}. Similar improvements are observed on Traffic, where the MSE at the same horizon is reduced by 85%, indicating that the gains generalize across domains.

Given (i) the low marginal price of LLM tokens relative to GPU training hours, and (ii) the consistent short-horizon error reductions that are operationally most valuable, the accuracy and latency benefits comfortably offset the larger prompt size. Hence trading cheap tokens for immediate, higher-quality forecasts yields a more favorable cost–performance envelope than the current state of the art, especially when rapid deployment and low engineering overhead are priorities.

4.5 Neighbor Results

In order to understand whether incorporating neighboring time-series into our PatchInstruct approach leads to better performance we compare our results for PatchInstruct and Neighs across multiple datasets (Table 4).

We cap the input at 50k tokens to control latency and cost across datasets. Given our prompt structure (task description, target series patches, and retrieved neighbors), this budget allows at most ($k = 5$) neighbors without truncation on our longest contexts. We thus fix the number of neighbors for all experiments to maximize usable contextual evidence while remaining within the token limit.

Both the Neighs and PatchInstruct+Neighs prompting strategies demonstrate clear improvements over the S^{2}IP-LLM baseline across datasets. As seen in Table 4, using Neighs alone often improves performance over PatchInstruct, particularly in the Weather dataset. For example, at horizon 2, Neighs reduces the MSE from 0.0076 (PatchInstruct) to 0.0039 and MAE from 0.067 to 0.051. At horizon 4, Neighs again performs better with an MSE of 0.0172 compared to 0.0236. These gains suggest that incorporating neighboring series can help the model infer more accurate trends by providing contextual information beyond the target sequence itself.

However, this is not universally true. In some cases, Neighs and PatchInstruct+Neighs underperform compared to PatchInstruct. For instance, in the Traffic dataset at horizon 5, Neighs shows a significant degradation, increasing the MSE from 8.46 (PatchInstruct) to 43.50, and PatchInstruct+Neighs to 36.43. This indicates that when neighbor series are less correlated, they can introduce confusion rather than useful context.

Despite these exceptions, the PatchInstruct+Neighs strategy still achieves the best overall performance in many cases aswell. But these results also highlight the importance of carefully selecting relevant neighbors to avoid negative transfer and ensure consistent forecasting improvements.

These results underline the strength of combining temporal structuring (via patches) with spatial context (via neighbors), enabling the model to learn more holistic representations and deliver significantly more accurate forecasts than our baseline.

Dataset	Horizon	PatchInstruct		Neighs		PatchInstruct+Neighs	
		MSE	MAE	MSE	MAE	MSE	MAE
Weather	1	**0.0014**	**0.029**	0.0024	0.042	0.0032	0.046
	2	0.0076	0.067	**0.0039**	**0.051**	0.0056	0.056
	3	0.0110	0.085	**0.0083**	**0.065**	0.0138	0.087
	4	0.0236	0.113	0.0172	0.091	**0.0114**	**0.075**
	5	0.0159	0.094	**0.0105**	**0.077**	0.0124	0.088
	6	**0.0101**	**0.083**	0.0116	0.084	0.0338	0.108
	12	0.0436	**0.137**	**0.0371**	0.144	0.0393	0.141
Traffic	1	**20.05**	2.76	35.15	3.05	22.09	**2.72**
	2	**9.38**	**1.89**	18.85	2.50	15.40	2.19
	3	**6.47**	**1.78**	26.67	2.74	13.61	2.10
	4	**11.15**	**1.89**	21.41	2.57	13.22	2.15
	5	**8.46**	**1.88**	43.50	3.39	36.43	3.25
	6	25.59	2.75	40.42	3.35	**14.94**	**2.13**
	12	**235.75**	**5.89**	285.82	7.71	269.68	6.88

Table 4: Forecasting Comparison: PatchInstruct vs Neighs vs PatchInstruct+Neighs.

5 Analysis

This analysis compares the performance of S^{2}IP-LLM (baseline) against our approach across different datasets and forecast horizons. The comparison focuses on error metrics (MSE and MAE) where lower values indicate better performance.

5.1 Performance Overview Across Models

The S^{2}IP-LLM baseline consistently shows higher error rates compared to our methods across datasets. Our methods shows remarkable improvements, with percentage reductions in MSE ranging from approximately 13% to 85% depending on the

dataset and method. For the Weather dataset, all our methods achieve over 80% MSE improvement. Our method achieve orders-of-magnitude faster runtimes than S^2IP-LLM with modest token usage growth, making them highly efficient for inference, especially when balanced with patch or neighbor-based prompts.

5.2 Method-Specific Performance

Our approach exhibits distinct strengths across datasets and forecasting horizons. The PatchInstruct framework demonstrates the most balanced performance, delivering substantial improvements on both the Traffic and Weather datasets—achieving up to 83% and 85% improvement over the baseline, respectively. The Neighs variant, which augments prompts with the closest neighboring time series, performs particularly well on the Weather dataset. Meanwhile, the combined PatchInstruct+Neighs strategy outperforms other methods on the Traffic dataset at longer horizons, highlighting the benefit of incorporating both local structure and external context in more challenging settings. These results suggest that method selection can be guided by the characteristics of the dataset and the specific forecasting task, with PatchInstruct offering a robust default across most conditions.

5.3 Dataset-Specific Analysis

For Weather forecasting, all methods substantially outperform the baseline. Overall MSE values are reduced from 0.009–0.0904 (baseline) to as low as 0.0014–0.043 (our methods).

PatchInstruct deliver substantial improvements, reducing MSE from 21–68 (baseline) to 6.47–20.05. However, Neighs and PatchInstruct+Neighs perform poorly in this scenario.

Across both approaches, forecast accuracy generally decreases as the horizon increases, but this pattern varies by dataset and method. For the Weather dataset, the performance degradation with longer horizons is less pronounced, especially for PatchInstruct+Neighs in multivariate settings.

5.4 Key Insights and Implications

The optimal forecasting method varies notably depending on the characteristics of the dataset and the forecasting horizon. For the Weather dataset, the PatchInstruct and Neighs strategy yields the most accurate results, effectively capturing the contextual signals from related series. In contrast, for the Traffic dataset, our main approach PatchInstruct performs best, suggesting that more complex augmentation may not always be beneficial in settings with high variability or less correlated neighbors.

6 Conclusion

The analysis demonstrates that prompt-based methods generally outperform the S^2IP-LLM baseline across most forecasting scenarios, especially at shorter horizons. The optimal method depends significantly on the specific dataset and forecast horizon, with PatchInstruct dominating in the majority of cases. This suggests that while prompt-based strategies offer a lightweight and effective alternative for time series forecasting.

7 Limitations

While our method achieves competitive accuracy compared to the S^2IP-LLM baseline, several limitations warrant consideration. First, the evaluation is limited to two benchmark datasets, which, though diverse, may not fully represent the diversity of real-world time series scenarios, such as irregular sampling or high-frequency patterns. Second, the framework remains heavily contingent upon carefully engineered prompts, introducing a labor-intensive design process that risks overfitting to specific tasks or datasets without systematic adaptation strategies. Future research should prioritize expanding dataset coverage, and developing adaptive prompting mechanisms.

References

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. 2024. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*.

Defu Cao, Furong Jia, Sercan O Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu. 2024a. Tempo: Prompt-based generative pre-trained transformer for time series forecasting.

Defu Cao, Wen Ye, Yizhou Zhang, and Yan Liu. 2024b. Timedit: General-purpose diffusion transformers for time series foundation model. *arXiv preprint arXiv:2409.02322*.

Ching Chang, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. 2024. Llm4ts: Aligning pre-trained llms as data-efficient time-series forecasters.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew G Wilson. 2024. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36.

Yuxuan Liang, Yue Wu, Sheng Wang, Xiaoyi Zhou, Wang Yang, Rong Xu, Wen Ye, Weizhi Lin, Zhiguo He, Zongyan Li, et al. 2024. Foundation models for time series analysis: A tutorial and survey. *arXiv preprint arXiv:2403.14735*.

Ruotong Liao, Xu Jia, Yangzhe Li, Yunpu Ma, and Volker Tresp. 2024. Gentkg: Generative forecasting on temporal knowledge graph with large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4303–4317.

John A Miller, Mohammed Aldosari, Farah Saeed, Nasid Habib Barna, Subas Rana, I Budak Arpinar, and Ninghao Liu. 2024. A survey of deep learning and foundation models for time series forecasting. *arXiv preprint arXiv:2401.13912*.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2022. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*.

Zijie Pan, Yushan Jiang, Sahil Garg, Anderson Schneider, Yuriy Nevmyvaka, and Dongjin Song. 2024. S^2ip-llm: Semantic space informed prompt learning with llm for time series forecasting.

Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, Marin Biloš, Sahil Garg, Anderson Schneider, Nicolas Chapados, Alexandre Drouin, Valentina Zantedeschi, Yuriy Nevmyvaka, and Irina Rish. 2023. Lag-Llama: Towards foundation models for probabilistic time series forecasting. *arXiv preprint arXiv:2310.08278*.

Yilong Ren, Yue Chen, Shuai Liu, Boyue Wang, Haiyang Yu, and Zhiyong Cui. 2024. Tpllm: A traffic prediction framework based on pretrained large language models.

Xiaofeng Shao, Soumya Ghosh, and Suhasini Subba Rao. 2020. Time-series analysis and its applications in scientific disciplines. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2174):20200209.

Xiaoming Shi, Shiyu Wang, Yuqi Nie, Dianqi Li, Zhou Ye, Qingsong Wen, and Ming Jin. 2024. Time-moe: Billion-scale time series foundation models with mixture of experts. *arXiv preprint arXiv:2409.16040*.

Hao Xue and Flora D. Salim. 2023. Promptcast: A new prompt-based learning paradigm for time series forecasting.

G Zerveas, S Jayaraman, D Patel, A Bhamidipaty, and C Eickhoff. 2020. A transformer-based framework for multivariate time series representation learning. arxiv. *arXiv preprint arXiv:2010.02803*.

Tian Zhou, Peisong Niu, Liang Sun, Rong Jin, et al. 2023. One fits all: Power general time series analysis by pretrained lm. *Advances in neural information processing systems*, 36:43322–43355.

Xinyu Zhou, Zhengyuan Ding, Shuo Ren, Yutao Chen, Xinhui Huang, Jianhao Shi, and Wayne Xin Zhao. 2024. Ditto: A survey on fine-grained alignments of large language models. *arXiv preprint arXiv:2411.05793*.

A Summary of Forecasting Results Across different datasets

Table 5 represents evaluating five prompting strategies—Basic, Non-Overlapping, STR Decompose, Reverse Patches, and Meta Patches—across the Weather and Traffic datasets, for horizons of 1, 3, and 6. While STR Decompose occasionally shows the lowest error for short-term predictions, Reverse Patch Instruct consistently delivers strong performance across all horizons and datasets. Notably, in long-range forecasts (H=6), where prediction becomes more challenging, Reverse Patch Instruct achieves the lowest or near-lowest MAE and MSE in both datasets, highlighting its stability and generalizability. Although it incurs a slightly higher inference time than the most lightweight methods, the trade-off is minimal when weighed against the accuracy benefits. Overall, the results suggest that Reverse Patch Instruct is the most effective and reliable strategy, outperforming other variants in terms of both robustness and predictive accuracy.

A.1 Basic PatchInstruct

The Basic PatchInstruct method employs overlapping sliding windows (size=3, stride=1) to capture local temporal patterns, followed by strategic sequence reversal to prioritize recent context. Unlike conventional approaches that process time series chronologically, this method reverses the generated patches such that the most recent window $[x_{t-2}, x_{t-1}, x_t]$ appears first in the token sequence. This architectural innovation forces the model to attend to immediate temporal patterns before historical context, combining the local sensitivity of patch-based methods (Nie et al., 2022) with explicit recency prioritization. The approach demonstrates particular efficacy in high-frequency electricity demand forecasting where near-term consumption patterns strongly influence subsequent values.

A.2 Non-Overlapping PatchInstruct

This variant utilizes non-overlapping windows where both window size and stride equal the prediction horizon (typically 3). The method partitions the series into discrete blocks like `[8.35, 8.36, 8.32]` followed by `[8.45, 8.35, 8.25]`, eliminating redundant data coverage while maintaining temporal progression. The design trades off some contextual granularity for computational efficiency, making it suitable for scenarios with pronounced periodic patterns. By processing patches in natural order without sequence reversal, the method preserves strict temporal causality, particularly effective when historical seasonal trends dominate the forecasting signal.

A.3 STR Decompose PatchInstruct

Integrating seasonal-trend-residual decomposition, this method first separates raw values into trend (trend_t) and residual (residual_t = series_t - trend_t) components. Each time step becomes a composite token $[T_t, R_t]$, enabling joint modeling of long-term trajectories and short-term fluctuations. These dual-aspect tokens are organized into overlapping windows:

$$[[T_1,R_1], [T_2,R_2], [T_3,R_3]]$$

preserving both local context and decomposition characteristics. The architecture explicitly captures multi-scale temporal dynamics, particularly beneficial for electricity demand series containing both gradual load changes and sudden consumption spikes.

A.4 Reverse Ordered Patches

Building on basic patch inversion, this method systematically prioritizes recent context through full sequence reversal of overlapping windows. The architectural innovation forces models to process the final patch $[x_{94}, x_{95}, x_{96}]$ first, implementing a "recency-first" attention mechanism. This structural bias proves particularly effective for 10-minute interval forecasting where immediate consumption patterns (last 30 minutes) contain stronger signals than older data. The approach maintains patch-based efficiency while adding temporal prioritization through simple sequence manipulation.

A.5 Meta Tokens Patches

This advanced variant enriches temporal representation through explicit time slot encoding. Each value v_t pairs with its absolute position in the daily cycle (0-143 slots) as $(v_t; \text{slot}_{\text{id}})$, creating hybrid tokens like `(8.35;63)`. These meta-tokens are windowed into overlapping patches:

```
[(v1;slot1), (v2;slot2), (v3;slot3)]
[(v2;slot2), (v3;slot3), (v4;slot4)]
. . .
[(v94;slot94),(v95;slot95),(v96;slot96)]
```

enabling joint learning of consumption patterns and their absolute temporal positions. The fixed

Table 5: Ablation Study: Comparing Variants of Patch-Based Prompting Strategies Across Datasets.

Dataset	Horizon	Basic			Non-Overlapping			STR Decompose			Reverse Patches			Meta Patches		
		MAE	**MSE**	**Time**	**MAE**	**MSE**	**Time**	**MAE**	**MSE**	**Time**	**MAE**	**MSE**	**Time**	**MAE**	**MSE**	**Time**
WEATHER	1	0.014	0.0003	0.66	0.012	0.0002	1.6151	**0.009**	**0.0001**	1.2553	0.015	0.0005	1.2290	0.020	0.0007	0.7471
	3	0.050	0.0045	0.989	0.055	0.0064	1.3580	**0.045**	**0.0030**	1.0737	0.053	0.0045	0.9732	0.067	0.0073	0.9269
	6	0.078	0.0116	1.146	0.063	0.0079	3.9016	0.100	0.0230	1.1523	**0.056**	**0.0070**	1.5120	0.060	0.0074	0.9760
TRAFFIC	1	1.43	6.27	0.699	1.35	5.60	1.3404	1.34	4.17	1.2677	**1.15**	**3.69**	1.1221	1.32	5.35	1.3846
	3	1.07	3.36	0.940	1.47	7.17	1.1910	0.99	2.53	1.1801	**0.89**	**1.83**	1.2018	0.94	2.38	1.1584
	6	1.14	4.37	0.910	1.14	3.60	1.3010	1.79	8.71	1.5667	**0.89**	**2.26**	1.1137	1.03	2.95	1.1650

slot indices provide crucial circadian context, helping disambiguate similar patterns occurring at different times (e.g., morning vs. evening peaks). This method adapts positional encoding strategies from language models to time series, grounding predictions in both value sequences and absolute time references.

Basic PatchInstruct

You are a forecasting assistant that sees time series data. The sequence represents the total regional humidity measured every 10 minutes.
Task:
(1) Split the series into overlapping patches with window size 3 and stride 1.
(2) Generate the patches in natural order, then reverse the list so the most recent patch appears first.
(3) Use these patch tokens to forecast the next 3 values.

Output format:

```
Patches:
[[latest_patch], ..., [oldest_patch]]

Prediction:
[y1, y2, y3]
```

No headings or extra words. Decimals ≤ 4 places; keep leading zeros (e.g., `0.8032`).

Non-Overlapping PatchInstruct

Tokenize the given time-series data into non-overlapping patches where a patch is a contiguous subsequence of the time-series. Ensure to use a fixed window size equal to the Horizon size (e.g., 3) and the stride is equal to the window size. This means that each patch starts exactly where the previous one ends and there will be no overlap. Output patches as a list, in order, using square brackets. Each patch becomes a token used to represent local temporal patterns. Use the sequence of patches to predict the next value(s). Below are a few shot examples of non-overlapping patching: Time series data: 8.35, 8.36, 8.32, 8.45, 8.35, 8.25, 8.20, 8.09, 8.13, 8.00, 7.94, 7.86

Patches generated based on Horizon (3), stride = 3:
[8.35, 8.36, 8.32]
[8.45, 8.35, 8.25]
[8.20, 8.09, 8.13]
[8.00, 7.94, 7.86]

Prediction: [7.89, 7.97, 7.94]

STR Decompose PatchInstruct

You are a forecasting assistant that receives STL-decomposed tokens.
Input:
- "series": 96 raw numbers (Humidity demand)
- "horizon" : 3 (fixed)
Task:
1. Decompose the series into
$trend_t$ and $residual_t = series_t - trend_t$
2. For each time-step create a pair token: ($trend_t$, $residual_t$).
3. Split the 96 composite tokens into overlapping patches (window = 3, stride = 1).
4. Use those patches to forecast the next 3 raw values.

Output exactly
`[[T1,R1], [T2,R2], [T3,R3]]`
`[[T2,R2], [T3,R3], [T4,R4]]`
`. . .`
`[[T94,R94], [T95,R95], [T96,R96]]`

`Prediction:`
`[y1, y2, y3]`
No headings or extra words. Decimals ≤ 4 places; keep leading zeros (e.g., `0.8032`).

Reverse Ordered Patches PatchInstruct

You are a forecasting assistant that sees time series data. The sequence represents the total regional humidity measured every 10 minutes.
Input:
- "series": 96 raw numbers (Humidity, 10-min cadence) - "horizon" : 3 (fixed)
Task:
1. Split the series into overlapping patches (window = 3, stride = 1).
2. Generate them in natural order, then reverse the list so the most recent patch appears first.
3. Use those patch tokens to forecast the next 3 normalised values.

Output format:
```
Patches:
[[latest_patch],
... ,
[oldest_patch]]

Prediction:
[y1, y2, y3]
```
No headings or extra words. Decimals ≤ 4 places; keep leading zeros (e.g., `0.8032`).

Meta tokens Patches PatchInstruct

You are a forecasting assistant that sees time series data, where each datapoint is paired with its 10-minute slot index within the day. The sequence represents the total regional humidity measured every 10 minutes.
Input:
- "series": 96 raw numbers (Humidity, 10-min cadence) - "horizon" : 3 (fixed)
Time-slot index - A day is divided into 144 slots (0 $\rightarrow$ 143). - slot = floor((60*HH + MM)/10). Example: 10:30 $\rightarrow$ 63 (because 10*60 + 30 = 630; 630/10 = 63).
Token format (value ; $slot_{id}$)
$slot_{id}$ corresponds to the measurement's clock time
Task:
1. Convert the 96-point series into 96 two-element tokens as above.
2. Split the token stream into overlapping patches (window = 3, stride = 1).
3. Use those patches to forecast the next 3 raw demand values.

Output format:
```
[(v1;slot1), (v2;slot2), (v3;slot3)]
[(v2;slot2), (v3;slot3), (v4;slot4)]
. . .
[(v94;slot94),          (v95;slot95),
(v96;slot96)]

Prediction:
[y1, y2, y3]
```
No headings or extra words. Decimals ≤ 4 places; keep leading zeros (e.g., `0.8032`).