CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

The Role of Textual Context in LLM-Based Time Series Forecasting

A Study of Prompt Content and Backbone Scale

Master's thesis in Computer science and engineering

MATTIAS MICHAELSSON

BRIAN NGUYEN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

The Role of Textual Context in LLM-Based Time Series Forecasting

A Study of Prompt Content and Backbone Scale

MATTIAS MICHAELSSON
BRIAN NGUYEN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

The Role of Textual Context in LLM-Based Time Series Forecasting
A Study of Prompt Content and Backbone Scale
MATTIAS MICHAELSSON
BRIAN NGUYEN

Supervisor: Ashkan Panahi, Computer Science and Engineering
Advisor: Nasser Mohammadiha, Ericsson AB
Examiner: Selpi, Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

The Role of Textual Context in LLM-Based Time Series Forecasting
A Study of Prompt Content and Backbone Scale
MATTIAS MICHAELSSON
BRIAN NGUYEN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Large language models (LLMs) are increasingly used as frozen backbones in time series forecasting, yet it remains unclear whether these models can benefit from the textual prompts that accompany the numerical input, and if so, what kind of text is needed and at what scale the benefit emerges. Prior work found that replacing prompts with random text has no effect, but tested only static prompts at small backbone scales. This thesis systematically investigates these questions through a controlled ablation study using a Time-LLM-inspired architecture on hourly bike rental demand. Four prompt configurations (no prompt, static domain description, sample-wise calendar context, and shuffled calendar context) are evaluated at two backbone scales (Qwen2.5-0.5B and Qwen2.5-7B), yielding eight experiments with all other variables held constant.

The findings show that the usefulness of the prompt depends on both the type of information provided and the backbone's scale. Generic dataset-level descriptions offer little value, whereas sample-specific context can improve forecasting by helping the model capture irregular patterns. This benefit is more evident for larger backbones and appears to transfer beyond the main experimental setting. Overall, the results suggest that frozen LLMs can benefit from text in forecasting, but only when that information is relevant to each sample and the backbone is large enough to act on it.

Keywords: Time series forecasting, large language models (LLMs), textual conditioning, frozen backbone models

Acknowledgements

We would like to express our gratitude to our supervisor, Ashkan Panahi, for his support and feedback throughout this thesis.

We would also like to thank our company advisors, Nasser Mohammadiha and Joakim Wennerberg, for their continuous support, helpful advice, and valuable contributions during the project.

Finally, we thank Ericsson for providing the data and computational resources necessary to carry out this project. Their support enabled the analyses and experiments presented in this work.

MATTIAS MICHAELSSON
BRIAN NGUYEN
Gothenburg, 2026-06-10

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background and Motivation

Time series forecasting is a fundamental task in domains ranging from energy management and transportation planning to finance and public health [1]. Traditional approaches, from statistical methods such as ARIMA to modern deep learning architectures, learn temporal patterns solely from historical numerical data [2]. While effective on regular, repeating patterns, these models struggle when demand deviates from historical norms due to other contextual factors that are difficult to infer from the time series alone [3].

Recent advances in large language models (LLMs) have opened new directions in time series research [4]. Pre-trained on vast text corpora, LLMs encode broad world knowledge [5], [6], [7], including information about calendar events and domain-specific patterns, within their learned representations. A growing body of work has explored whether this knowledge can be transferred to time series tasks. One line of research treats time series values as text strings and queries LLMs directly [4], [8], [9], [10], [11]; another freezes the LLM and trains lightweight adaptation layers to map numerical inputs into the model's embedding space [4], [12], [13], [14]. We refer to this latter paradigm as *time-series aligned*: the LLM's parameters remain frozen, and one or more trainable layers project (or *reprogram*) the time-series input into the LLM's token space, enabling it to be processed by the pretrained Transformer. This paradigm is particularly appealing because it preserves the LLM's pretrained knowledge while requiring only a fraction of the parameters to be trained. In addition, the prompt tokens and the aligned time-series tokens appear in the same sequence, which enables testing whether the LLM can bridge the two modalities.[15].

Among time-series aligned methods, Time-LLM [13] introduces a specific projection mechanism called *reprogramming*: time-series patches are mapped into the LLM's token space via cross-attention with learned prototypes, and a natural-language prompt prefix can optionally be prepended to provide task context, for example, "forecast the next 48 points." The authors report that including such a prompt improves performance. Still, the mechanism remains unclear: does the LLM extract useful information from the text, or does the additional token sequence act as a generic conditioning signal? Two recent studies cast doubt on the importance of prompt content. Niu et al. [16] replaced meaningful prompts with random text and random embeddings in a GPT-2-based framework and found no degradation,

concluding that prompts act as implicit adapters rather than conveying semantic information. Tan et al. [17] showed that even replacing a pretrained LLM backbone with a randomly initialised Transformer yields comparable forecasting performance. However, both studies tested only static prompts on small backbone models (GPT-2 and truncated LLaMA), leaving open whether their conclusions hold for sample-wise context or for larger backbones. Tan et al. themselves suggest that LLMs may be better suited to multimodal settings that require textual reasoning.

This thesis investigates the role of textual context in time-series aligned LLMs. Rather than asking whether LLMs are useful for time series in general, we ask a more targeted question: within the time-series aligned paradigm, when a frozen LLM receives both time-series tokens and natural-language tokens in a single sequence, does the textual input matter, and if so, what kind of text is needed? By systematically varying the textual context while holding the architecture constant, we isolate the contribution of prompt content from prompt presence and from backbone capacity. The controlled ablation focuses on hourly bike rental demand [18], a domain where calendar context (holidays, weekdays, time of day) plausibly affects demand in ways that a frozen LLM may encode from pretraining. The best configuration identified by this study is then re-evaluated on an industrial telecom KPI dataset as an external validity check, testing whether the chosen architecture and training recipe transfer to a different domain with shorter history, higher sampling frequency, and operational rather than human-driven signals.

1.2 Problem Statement

Several architectures now repurpose frozen LLMs as time-series forecasters within the time-series aligned paradigm. In this setup, numerical inputs are projected into the LLM's token space so that the frozen Transformer can process them. Yet important questions remain about *what role the textual input plays*. Within this paradigm, methods vary in how they use text: GPT4TS [12] omits text entirely, Time-LLM [13] prepends a static task instruction along with input context, and GPT4MTS [19] pairs each channel with a natural-language variable description. As discussed in Section 1.1, the two studies that directly test prompt effectiveness, Niu et al. [16] and Tan et al. [17], found no evidence that prompt content matters, but both tested only static, dataset-level prompts at small backbone scales. This leaves two questions open: whether their conclusion holds at larger scales where richer pretrained knowledge is available [20], and whether *sample-wise* context, i.e., window-specific calendar text describing when the lookback and forecast horizon occur, behaves differently from static or random text.

This gap matters for two reasons. First, if the textual content is irrelevant, meaning that any token sequence of comparable length produces the same effect, then the reported benefits of prompting may be architectural artefacts rather than evidence of cross-modal knowledge transfer. Second, if textual content matters, understanding *what kind* of text is useful (static domain descriptions vs. sample-wise context) and *at what backbone scale* the effect emerges will help guide the design of future multimodal forecasting systems.

This thesis addresses the gap through a controlled ablation study. We build a forecasting model that uses a frozen LLM backbone with an optional natural-language prefix, and evaluate four context configurations: no context, a static domain/task description, sample-wise calendar context, and shuffled calendar context, at two backbone scales. By holding the architecture and training procedure constant across all eight experiments, we isolate the contribution of textual context and its interaction with backbone capacity.

1.3 Research Questions

The overarching question of this thesis is: *How does textual context affect time-series aligned LLMs for forecasting, and under what conditions does it improve prediction accuracy?* To answer this, the thesis is structured around four sub-questions, each designed to isolate a potential mechanism through which textual context may influence forecasting performance.

RQ1 — Static prompt effect. Does a fixed dataset-level description, stating the domain, known periodicities, and forecasting task, improve performance compared to using the time series alone?

This tests the simplest form of textual input: a prompt that is identical across all samples. If the frozen LLM benefits from a textual framing of the domain, a static prompt should improve over no prompt.

RQ2 — Sample-wise context effect. Does sample-wise contextual text, specifying the calendar position, day of week, and hour partitioning of each forecast window, improve performance beyond a static prompt or no prompt?

Sample-wise context provides information that varies across samples and that the model cannot directly infer from the numerical time series. If the LLM can condition on this information, sample-wise context should outperform both the static prompt and the no-prompt baseline.

RQ3 — Semantic content vs. token conditioning. Does the *content* of the sample-wise context matter, or does any text prefix act as a generic conditioning signal?

We test this via a shuffled-context ablation: the same sample-wise descriptions used in RQ2 are randomly permuted across samples, thereby breaking the correspondence between each time-series window and its textual description. If the shuffled variant performs comparably to the correctly paired variant, then the benefit comes from the presence of text tokens rather than their meaning. If performance degrades, it would indicate that the model is extracting and acting on the specific content.

RQ4 — Backbone scale interaction. How does the size of the frozen LLM backbone interact with the effect of textual context?

By repeating all prompt configurations at two backbone scales (Qwen2.5-0.5B and Qwen2.5-7B), we test whether a larger backbone amplifies, diminishes,

or leaves unchanged the patterns observed at a smaller scale. This addresses under which circumstances scaling the frozen backbone is worthwhile.

1.4 Contributions

This thesis makes the following contributions:

1. **A multimodal time series forecaster.** We design and implement a univariate time-series forecasting model that utilises a frozen LLM backbone for feature extraction and multimodal fusion. The architecture combines patch-based input embedding, inspired by PatchTST [21], with the reprogramming-based LLM integration of Time-LLM [13], enabling controlled experiments on the effects of textual context.
2. **A systematic ablation of textual context.** We evaluate four prompt configurations, no prompt, static domain/task description, sample-wise calendar context, and shuffled calendar context, at two backbone scales (Qwen2.5-0.5B and Qwen2.5-7B), yielding eight experiments with all other variables held constant. This design isolates the effect of prompt content from prompt presence and from backbone capacity.
3. **A shuffled-context control at multiple scales.** Niu et al. [16] tested replacing static dataset-level prompts with random text and random embeddings at small backbone scale (GPT-2, LLaMA-8-layers) and found no effect of prompt content. We extend this investigation with a shuffled ablation that breaks the correspondence between each time-series window and its *sample-wise* calendar description, and apply it at both the 0.5B and 7B scales. This reveals that the conclusion drawn at the small scale does not generalise: at 7B, correctly paired context significantly outperforms shuffled context, indicating that the model extracts and acts on the semantic content of the prompt.
4. **Evidence of a scale–context interaction.** In our experiments, we show that a larger frozen backbone is only beneficial when paired with informative textual context: without text, the 0.5B model significantly outperforms the 7B model; with sample-wise context, the 7B model achieves the best performance of all variants.
5. **A situational advantage on significant dates.** We demonstrate that while the LLM-based model does not significantly outperform PatchTST on aggregate metrics, it outperforms PatchTST on test windows overlapping significant dates such as holidays and special events ($p = 0.043$), where the multimodal fusion enables the model to utilise signals in the textual context.
6. **External validation on industrial data.** As a case study, we re-train the best LLM configuration identified in the controlled ablation, together with all four baselines, on an industrial telecom KPI dataset comprising operational measurements from six LTE cells. The configuration remains competitive with PatchTST on pooled metrics and outperforms the simpler baselines, providing a modest external validity check that the architecture and training recipe are

not specific to the Bike Sharing setting [18]. The four research questions are not re-tested in this case study.

1.5 Thesis Structure

The remainder of this thesis is organised as follows. Chapter 2 provides the technical foundations: time-series forecasting formulations and baselines, the architecture and key properties of large language models, existing approaches that adapt LLMs for time-series tasks, and the statistical testing framework used to compare models. Chapter 3 describes the model architecture, prompt configurations, dataset, baselines, experimental setup, and evaluation methodology. Chapter 4 presents the experimental results, organised by research question, including the baseline comparison, the significant date analysis, and a case study on the Telecom KPI dataset. Chapter 5 discusses the findings, relates them to prior work, identifies limitations, outlines directions for future research, and states the conclusions.

2

Background

This chapter provides the technical foundations needed to understand the methods and results of this thesis. Section 2.1 introduces the time series forecasting problem and the baseline approaches used in this work. Section 2.2 describes the architecture and key properties of large language models. Section 2.3 reviews how LLMs have been adapted for time series tasks. Finally, Section 2.4 presents the statistical testing framework used to compare models.

2.1 Time Series Forecasting

Time series forecasting concerns the prediction of future observations from previously observed values. This section introduces the forecasting formulation used in this thesis and reviews the baseline model families used for comparison, ranging from simple seasonal and linear methods to recurrent and Transformer-based architectures.

2.1.1 Problem Formulation

A univariate time series is an ordered sequence of real-valued observations $x_1, x_2, \ldots, x_T$ recorded at regular intervals. In forecasting, the goal is to predict future values given a window of past observations. More precisely, given a *lookback window* of L consecutive observations $\mathbf{x} = (x_t, x_{t+1}, \ldots, x_{t+L-1})$, the task is to produce a forecast of the next H values $\hat{\mathbf{y}} = (\hat{x}_{t+L}, \ldots, \hat{x}_{t+L+H-1})$, where H is the *forecast horizon*. The model learns a mapping $f : \mathbb{R}^L \to \mathbb{R}^H$ that minimises a loss function between the forecast $\hat{\mathbf{y}}$ and the ground truth $\mathbf{y} = (x_{t+L}, \ldots, x_{t+L+H-1})$.

In practice, a single long time series is converted into many (input, target) pairs using a *sliding window* procedure: a window of length $L + H$ is moved across the series, usually one step at a time, and each position yields one training or evaluation sample. This approach makes efficient use of the available data but introduces statistical dependence between consecutive samples, since neighbouring windows share $L + H - 1$ observations. This overlap has implications for model comparison, which is addressed in Subsection 2.4.1.

2.1.2 Classical and Linear Baselines

Before introducing more complex architectures, it is useful to establish simple reference points that any competitive model should surpass.

Seasonal naive is the simplest forecasting strategy and exploits periodicity: the forecast is a copy of the corresponding period from the recent history. For a series with period P (e.g. $P = 24$ for hourly data with daily seasonality), the forecast at horizon h is $\hat{x}_{t+h} = x_{t+h-P}$. This baseline requires no training and captures the dominant periodic pattern, but it cannot adapt to trends, day-of-week effects, or irregular events. Despite its simplicity, seasonal naive remains a popular baseline for forecasting [22].

Beyond naive predictors, simple trainable linear models have also proven surprisingly strong for time series forecasting. Zeng et al. [23] introduced DLinear, a simple model that first decomposes the input into a trend component, obtained with a moving-average filter, and a residual seasonal component. Each component is then mapped directly from L input steps to H using its own linear layer, and the two outputs are summed. The model has no attention mechanism, no recurrence, and minimal parameters, yet in their experiments, Zeng et al. [23] reported that it matched or outperformed several Transformer-based forecasters.

2.1.3 Recurrent Models

Recurrent neural networks (RNNs) [24] process sequential data by maintaining a hidden state that is updated at each time step, thereby capturing temporal dependencies of variable length. In the standard formulation, the hidden state $\mathbf{h}_t$ is computed as a function of the current input $\mathbf{x}_t$, and the previous state $\mathbf{h}_{t-1}$, and the final hidden state is used to produce the forecast.

In practice, vanilla RNNs struggle to learn long-range dependencies because gradients either vanish or explode during backpropagation through time [25]. The Long Short-Term Memory (LSTM) network addresses this through a gating mechanism: an input gate, a forget gate, and an output gate control the flow of information into, within, and out of a memory cell [24]. This design allows LSTMs to selectively retain or discard information over long sequences, making them effective for time-series tasks [26], where relevant patterns may span hundreds of time steps.

LSTMs remain a relevant baseline: they capture non-linear temporal dependencies that linear models cannot, they are well-understood, and they are computationally inexpensive to train. In this thesis, an LSTM baseline provides a reference point between the linear DLinear model and the Transformer-based approaches.

2.1.4 Transformer-based Models

The Transformer architecture, originally developed for machine translation [27], has been widely adopted for time series forecasting [28]. Its core mechanism is *self-attention*: given a sequence of input tokens, each token computes a weighted sum over all other tokens, where the weights are determined by learned query-key dot products [27]. This allows the model to capture dependencies between arbitrary positions in the sequence without the sequential bottleneck of recurrent models [27].

Early Transformer-based forecasters treated each time step as a separate token.

This produced long sequences (e.g. 96 tokens for a 96-hour lookback) and incurred quadratic attention cost, while each token carried only a single scalar value, a poor match for the attention mechanism's capacity [29]. Nie et al. [21] addressed both issues with PatchTST, which introduced two key ideas:

- **Patching.** The input series is segmented into subseries-level patches of length p, extracted with stride s. Each patch becomes a single token, reducing the sequence length from L to approximately L/s while preserving local temporal structure within each patch. The shorter sequence lowers the quadratic attention cost and allows the model to capture longer-range dependencies within the same computational budget.
- **Channel independence.** In multivariate settings, each variable is treated as a separate univariate series and processed independently through the same Transformer encoder.

PatchTST demonstrated that Transformer-based models can remain competitive with strong linear baselines, and its patching strategy has been adopted in many subsequent time series architectures, including the LLM-based approach used in this thesis.

2.2 Large Language Models

Large language models provide the textual backbone used in this thesis. This section introduces the architectural properties that make LLMs suitable as frozen feature extractors, explains how text is converted into token embeddings, and contrasts different adaptation strategies, with particular emphasis on the fully frozen setting used in this work.

2.2.1 Architecture

A large language model (LLM) is a neural network trained on large text corpora to predict the next token in a sequence [30]. Modern LLMs are built on the *decoder-only* Transformer architecture, which consists of a stack of identical layers, each containing two sub-layers: a *causal self-attention* mechanism and a position-wise feed-forward network.

Causal self-attention is a restricted form of the self-attention described in Subsection 2.1.4. While the Transformer encoder used in PatchTST allows each token to attend to all positions in the sequence, causal attention masks out all positions to the right of the current token: token i can attend only to tokens $1, \ldots, i$. This enforces an autoregressive property: each token's representation depends only on itself and its preceding tokens, which is essential for left-to-right text generation but also has consequences when the architecture is repurposed for other tasks.

At each layer, the attention output and the feed-forward output are combined with residual connections and layer normalisation, forming a deep residual network. Modern LLMs stack tens to hundreds of such layers, with hidden dimensions (d_{llm})

ranging from hundreds to tens of thousands. The result is a model that transforms an input sequence of token embeddings into contextualised representations, in which each position encodes information about all preceding tokens as filtered through the model's learned weights.

A key property of LLMs for downstream applications is that these learned representations encode far more than surface-level language patterns. Through pretraining on diverse text, LLMs acquire structured knowledge about the world, temporal patterns, causal relationships, and domain-specific facts that are distributed across their parameters [6], [7], [13]. This property motivates the use of LLMs as feature extractors for non-language tasks, including time series forecasting.

2.2.2 Tokenisation and Embeddings

Before an LLM can process text, the raw character string must be converted into a sequence of discrete tokens from a fixed vocabulary [31]. Modern LLMs use subword tokenisation algorithms such as Byte Pair Encoding (BPE), which iteratively merge the most frequent character pairs in the training corpus until a target vocabulary size is reached [31], [32]. The result is a vocabulary that balances character-level granularity for rare words and whole-word tokens for common ones. A typical LLM vocabulary contains tens to hundreds of thousands of tokens.

Each token is mapped to a dense vector via a learned *embedding table*: a matrix $\mathbf{W}_e \in \mathbb{R}^{V \times d_{\text{llm}}}$, where V is the vocabulary size and d_{llm} is the model's hidden dimension. The embedding for a given token is simply the corresponding row of this matrix. Positional information is added, either through learned positional embeddings or rotary position encodings, so that the model can distinguish tokens at different sequence positions [27].

The embedding table and the tokeniser are tightly coupled to the LLM, so using a different tokeniser is non-trivial [33]. When an LLM is repurposed for a non-language task, such as time-series forecasting, numerical inputs cannot be passed directly through the embedding layer. They must often be projected into the same d_{llm}-dimensional space through a separate mechanism. How this projection is achieved is the central design problem of LLM-based time series methods, discussed in Section 2.3.

2.2.3 Frozen vs. Fine-tuned Usage

When adapting a pre-trained LLM to a downstream task, a fundamental design choice is which parameters to update. Three strategies span the spectrum:

Full fine-tuning: all parameters of the LLM are updated on the downstream task's training data [34]. This gives the model maximum flexibility to adapt but requires substantial compute and data, risks catastrophic forgetting of pretrained knowledge [35], and makes it difficult to attribute performance to what the model learned during pretraining versus fine-tuning.

Parameter-efficient fine-tuning: A small set of additional parameters (adapters,

low-rank updates, or soft prompts) is inserted into or appended to the frozen model, and only these parameters are trained. Methods such as LoRA [34] inject trainable low-rank matrices into the attention layers while keeping the original weights fixed. This preserves most of the pretrained representations while allowing task-specific adaptation with a fraction of the training cost.

Fully frozen: The LLM's parameters are entirely fixed, only external layers (input projection, output head) are trained. The frozen model acts as a fixed feature extractor, and all task adaptation must happen in the trainable layers surrounding it. This is the most restrictive setting but also the cleanest for studying what the pretrained representations contribute: any benefit from the LLM's knowledge must come from its frozen weights, not from task-specific weight updates.

This thesis uses the fully frozen strategy. By keeping the LLM fixed, the experimental design ensures that any effect of textual context operates through the pretrained representations rather than through parameters adapted to the forecasting task. This enables isolation of the prompt's contribution from the backbone's learned knowledge.

2.3 LLMs for Time Series

As discussed in Subsection 2.2.2, numerical time series cannot be passed through an LLM's embedding table. Several strategies have been proposed to bridge this modality gap. They can be broadly grouped into two paradigms.

In the *prompt-based* paradigm, numerical values are converted into text strings and fed to the LLM as ordinary language input. Gruver et al. [8] showed that LLMs can perform zero-shot forecasting when time series are formatted as comma-separated numbers, with the tokenisation strategy (number of decimal places, spacing) significantly affecting performance. While this demonstrates an intrinsic capacity for numerical extrapolation, the approach is limited by the LLM's tokeniser, which was not designed for numerical precision, and by the inability to train task-specific input representations [36].

In the *time-series aligned* paradigm, the LLM's tokeniser is bypassed entirely: one or more trainable layers project the numerical input into the model's embedding space, producing tokens that the frozen Transformer can process alongside (or instead of) natural-language tokens. This is the paradigm used in this thesis, and it encompasses a range of projection mechanisms of increasing complexity.

2.3.1 Projection Mechanisms within the Time-Series Aligned Paradigm

The simplest projection is a learned linear mapping. Zhou et al. [12] demonstrated this with GPT4TS, which freezes all self-attention and feed-forward layers of GPT-2 and trains only the input embedding, positional embedding, and layer normalisation. The patched time series is mapped into the d_{llm}-dimensional space via a trainable

linear layer, and the frozen Transformer processes these tokens as if they were language. Despite this minimal adaptation, GPT4TS achieved competitive results across forecasting, classification, and imputation tasks, suggesting that the frozen Transformer's representations capture useful temporal structure.

Time-LLM [13] introduces a more expressive projection called *reprogramming*: the patched time-series tokens are aligned with the LLM's embedding space via multi-head cross-attention with a set of learned text prototypes. Each patch token serves as a query and attends to a bank of k trainable prototype vectors residing in the LLM's d_{llm}-dimensional space. The output is a sequence of reprogrammed tokens that the frozen LLM can process through its standard forward pass. In addition, Time-LLM prepends a natural-language prompt prefix, a textual description of the dataset and forecasting task, before the reprogrammed tokens, so that the frozen LLM receives both language and reprogrammed time-series tokens in a single sequence. The authors report that this prompt improves performance, but the contribution of the prompt content relative to the reprogramming mechanism is not systematically investigated.

This thesis adopts the reprogramming mechanism from Time-LLM as the projection layer in the time-series-aligned paradigm. The architecture is described in detail in Chapter 3.

2.3.2 The Role of Prompts and Textual Context

A recurring design element in LLM-based forecasting is the use of natural-language prompts alongside numerical input. The literature now offers a growing, but sometimes contradictory, body of evidence on whether, how, and under what conditions these prompts contribute.

Several methods incorporate text as a core component, spanning different paradigms of LLM usage. Within the time-series aligned paradigm, where a frozen LLM receives numerical input that has been projected into its token space, Time-LLM [13] prepends a static dataset description along with task instructions, before the reprogrammed time-series tokens and reports that removing this "Prompt-as-Prefix" degrades the average MSE by more than 8%, with even larger drops in few-shot settings. GPT4MTS [19] pairs each channel in a multivariate series with a natural-language variable description, using a frozen GPT-2 backbone. TEMPO [14] replaces natural-language prompts with *learnable* prompt tokens conditioned on trend, seasonal, and residual decompositions to guide a frozen GPT-2 backbone. In the *prompt-based* paradigm, where LLMs are queried directly with text-formatted time series, PromptCast [10] frames forecasting entirely as a sentence-to-sentence task, and LSTPrompt [37] generates sample-wise prompts that describe the time series' statistical properties, level, seasonality, and trend, to guide zero-shot predictions from GPT-3.5 and GPT-4.

While these methods demonstrate the breadth of prompting strategies, they operate in paradigms ranging from zero-shot text generation to learnable soft prompts, making direct comparison of prompt mechanisms difficult. The remainder of this

thesis focuses on the time-series-aligned paradigm, in which a frozen LLM receives both natural-language text and reprogrammed time-series tokens within a shared sequence.

Within the time-series aligned paradigm, the question of *why* prompts help has been investigated directly. Niu et al. [16] conducted a systematic study using the OFA/GPT-2 framework [12]. They tested two controls: "Random Prompting," which replaces the static dataset-description prompt with random text of comparable length, and "Random Word Embedding," which replaces the prompt's word embeddings with random vectors while preserving the token count. On the ETT [38] benchmarks, both controls achieved forecasting accuracy comparable to the standard textual prompt, and, for LLaMA [39] (8 layers), random prompting even produced a slight improvement. Niu et al. concluded that (a) adding text prompts is roughly equivalent to introducing additional adapter parameters, and (b) it is the introduction of learnable parameters, rather than the textual information itself, that aligns the LLM with the forecasting task. However, these experiments tested only *static* dataset-level prompts (identical across all samples) and used small backbone scales: GPT-2 (12 layers, 124M parameters) and a truncated LLaMA (8 layers). Whether their conclusion holds for sample-wise context or at larger scales remains unclear.

A related challenge comes from Tan et al. [17], who ablated three time-series aligned methods (OFA, Time-LLM, and CALF) and showed that replacing the pretrained LLM backbone with a randomly initialised Transformer of the same architecture yields comparable performance on standard benchmarks. They further demonstrated that shuffling the time-series input (not the text prompt) has little effect on LLM-based forecasters, suggesting that the models do not acquire meaningful sequential reasoning from pretraining. However, Tan et al. conclude by recommending that LLM-based time series methods should focus on "multimodal applications that require textual reasoning", suggesting that LLMs may be more promising in settings where informative textual context is provided alongside the numerical signal, which is the setting investigated in this thesis.

Taken together, the existing evidence presents a tension: time-series aligned methods that include prompts report better performance than those that omit them [13], yet the two studies that directly investigate the mechanism, Niu et al. and Tan et al., suggest that the benefits may stem from architectural artefacts. Notably, both studies share the same limitations identified in Section 1.2: static prompts at small backbone scales, with no investigation of sample-wise context.

2.4 Statistical Testing

Forecasting models are evaluated not only by their average error, but also by whether observed differences in error are statistically reliable. This section introduces the statistical tests used to compare model performance in this thesis, with particular attention to the dependence introduced by overlapping forecast windows.

w

2.4.1 Diebold-Mariano Test

Comparing forecasting models based solely on aggregate error metrics, such as mean squared error (MSE) or mean absolute error (MAE), can be misleading. Two models may differ in their average error, but the difference may not be statistically significant after accounting for variability across individual forecasts. The Diebold-Mariano (DM) test provides a formal framework for testing whether two models have equal predictive accuracy [40].

The test proceeds as follows. Given n forecast windows, define the *loss differential* for window i as $d_i = \mathcal{L}(\hat{y}_i^A, y_i) - \mathcal{L}(\hat{y}_i^B, y_i)$, where $\mathcal{L}$ is a loss function (typically squared error) and $\hat{y}_i^A$, $\hat{y}_i^B$ are the forecasts of models A and B. The null hypothesis is that the expected loss differential is zero: $H_0 : \mathbb{E}[d_i] = 0$. The DM test statistic is

$$\mathrm{DM} = \frac{\bar{d}}{\hat{\sigma}_{\bar{d}}}, \tag{2.1}$$

where $\bar{d}$ is the sample mean of $\{d_i\}$ and $\hat{\sigma}_{\bar{d}}$ is an estimate of its standard error. Under the null hypothesis, this statistic is asymptotically standard normal, and a two-sided p-value is computed accordingly.

A critical assumption is that the standard error $\hat{\sigma}_{\bar{d}}$ is estimated correctly. When forecast windows are generated by a sliding window with a stride smaller than the forecast horizon, consecutive loss differentials share overlapping forecast periods and are therefore serially correlated. A naive variance estimate would understate the true uncertainty, producing artificially small p-values. Addressing this autocorrelation is the subject of Subsection 2.4.2.

2.4.2 Newey-West HAC Standard Errors

When the loss differentials $\{d_i\}$ are serially correlated, the standard error of $\bar{d}$ must account for the autocovariance structure of the sequence. Newey and West [41] proposed a *heteroskedasticity and autocorrelation consistent* (HAC) variance estimator that incorporates autocovariances up to a chosen lag ℓ:

$$\hat{\sigma}_{\bar{d}}^2 = \frac{1}{n}\left[\hat{\gamma}_0 + 2\sum_{j=1}^{\ell} w(j,\ell)\,\hat{\gamma}_j\right], \tag{2.2}$$

where $\hat{\gamma}_j$ is the sample autocovariance at lag j and $w(j,\ell) = 1 - j/(\ell+1)$ is the Bartlett kernel weight. The Bartlett kernel linearly downweights higher-order lags, ensuring that the resulting variance estimate is positive semi-definite.

The lag truncation parameter ℓ controls how many lags of autocorrelation are accounted for. Setting ℓ too low ignores genuine serial dependence and underestimates the standard error; setting it too high introduces noise from estimating autocovariances at lags where the true correlation is negligible. In the context of overlapping forecast windows, a natural choice is $\ell = \lceil H/s \rceil - 1$, where H is the forecast horizon and s is the sliding-window stride: windows separated by $\lceil H/s \rceil$ or more steps in the sample index correspond to non-overlapping forecast horizons in time, and are

therefore approximately independent. When $s = 1$ this reduces to $\ell = H - 1$. The specific application of this estimator to the experiments in this thesis is described in Subsection 3.6.2.

3

Methods

This chapter describes the model architecture, prompt configurations, dataset, baselines, experimental setup, and evaluation methodology.

Notation: Lowercase bold letters ($\mathbf{x}$, $\mathbf{y}$) denote vectors, uppercase bold letters ($\mathbf{W}$, $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$) denote matrices, and plain italic letters (μ, σ, d_k) denote scalars. Dimension symbols carry upright subscripts: d_{llm}, d_{model}, n_{heads}. Let $\mathbf{x} \in \mathbb{R}^L$ denote the input lookback window and $\hat{\mathbf{y}} \in \mathbb{R}^H$ the forecast horizon. The values of L and H depend on the dataset: $L = 96$, $H = 48$ for the Bike Sharing dataset (a 4-day lookback and a 2-day forecast horizon) and $L = 144$, $H = 96$ for the Telecom KPI dataset (a 36-hour lookback and a 24-hour forecast horizon). All equations describe sample-wise operations unless stated otherwise; the batch dimension B appears only in implementation-level shape annotations, written in parenthesised tuple form (B, L, d). Semicolons in brackets denote concatenation along the sequence axis, e.g. $[\mathbf{A}\,;\,\mathbf{B}]$.

3.1 Model Architecture

A frozen pre-trained LLM is repurposed as a time-series backbone by reprogramming (see 3.1.3) patched time-series embeddings into the LLM's token space via learned cross-attention, with an optional natural-language prompt prepended to provide domain- or sample-level context. The entire LLM remains frozen throughout training; only the reprogramming, patch embedding, and output projection layers are trained.

Qwen2.5 [42] is chosen as the backbone family because it is open-weight, available at multiple scales with a shared architecture, and released under a permissive licence, allowing controlled comparison across scales without confounding architectural differences. Importantly, Qwen2.5 offers a 0.5B variant that shares the same architecture (grouped-query attention, RMSNorm, SwiGLU) as its 7B counterpart, providing a wider scale range within a single family than alternatives such as LLaMA 3 (which has 1B parameters for the smallest model) or Gemma 2 (which has 2B parameters for the smallest model). This 14× parameter ratio between the two scales maximises the sensitivity of the scale–context interaction tested in RQ4. Two sizes are used: **Qwen2.5-0.5B** ($d_{\text{llm}} = 896$, 24 Transformer layers, 151,936 vocabulary tokens) and **Qwen2.5-7B** ($d_{\text{llm}} = 3{,}584$, 28 Transformer layers, 152,064 vocabulary tokens). All Transformer layers are retained in both cases.

Figure 3.1 illustrates the end-to-end pipeline. The input is a univariate time series

of L hourly observations, and the output is an H-step forecast. The following subsections describe each stage in detail.

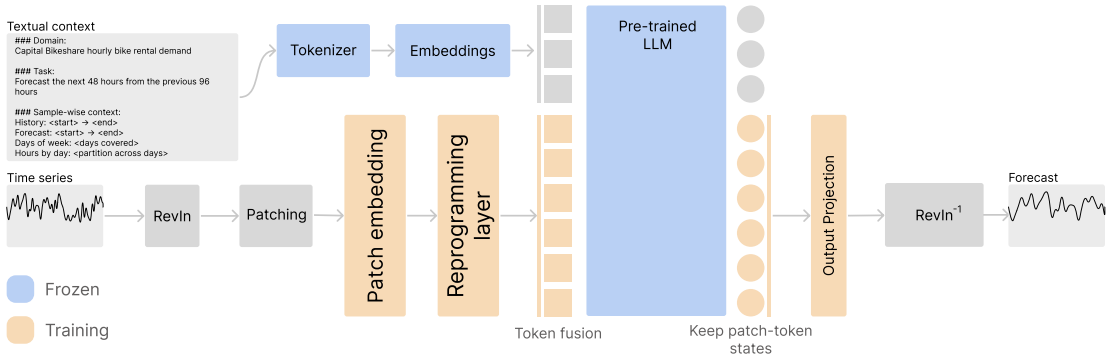


Figure 3.1: End-to-end architecture of the Time-LLM-inspired model. The frozen LLM backbone (shaded) receives reprogrammed patch tokens, optionally preceded by prompt embeddings. Only the reprogramming layer, patch embedding, and output projection are trained.

3.1.1 Reversible Instance Normalisation (RevIN)

Before any processing, each input sample $\mathbf{x} \in \mathbb{R}^L$ is normalised independently to zero mean and unit variance along the time dimension:

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x} - \mu}{\sigma}, \tag{3.1}$$

where μ and σ are the sample-wise mean and standard deviation, computed over the L input time steps and detached from the computation graph. After the model produces a forecast $\hat{\mathbf{y}}_{\text{norm}}$, the output is denormalised by reversing the transformation:

$$\hat{\mathbf{y}} = \hat{\mathbf{y}}_{\text{norm}} \cdot \sigma + \mu. \tag{3.2}$$

This makes the model invariant to the level and scale of each sample, enabling it to learn shape and patterns rather than absolute magnitude [43].

3.1.2 Patch Embedding

The normalised input is divided into overlapping patches rather than processed as individual time steps [21]. Patching reduces the effective sequence length seen by the Transformer (N tokens instead of L), lowering computational cost while preserving local temporal structure within each patch. Overlapping patches ensure that boundary information is not lost between adjacent segments.

Patches of length p are extracted with stride s, producing

$$N = \left\lfloor \frac{L - p}{s} \right\rfloor + 1 \tag{3.3}$$

patches. Before unfolding, the series is end-padded by s time steps (replicating the last observed value) so that a full patch always covers the final observations. This

adds one patch, giving $N = \lfloor (L + s - p)/s \rfloor + 1$. For the Bike Sharing dataset, we use a lookback window of $L = 96$, with $p = 16$ and $s = 8$, which yields $N = 12$ patches, each overlapping its neighbour by $p - s = 8$ time steps.

A 1D convolution maps the patch sequence to a $d_{\text{model}} = 16$-dimensional embedding. Each patch of length $p = 16$ is treated as a p-channel input vector, and a convolutional kernel of size 3 slides over the sequence of N patches. Consequently, the embedding of patch i is computed from patch_{i-1}, patch_i, and patch_{i+1}, with circular padding applied at the boundaries. Each output token, therefore, incorporates local context from neighbouring patches. This embedding dimension matches the Time-LLM default for long-term forecasting [13] and the reduced PatchTST configuration recommended for small datasets [21]. This value was adopted without further tuning, sensitivity to d_{model} is not explored in this thesis.

The result is a patch embedding matrix $\mathbf{E} \in \mathbb{R}^{N \times d_{\text{model}}}$: N patch tokens in a d_{model}-dimensional space.

3.1.3 Reprogramming Layer

The patch embeddings live in a d_{model}-dimensional space, but the frozen LLM expects input in its own d_{llm}-dimensional token space. The reprogramming layer bridges this gap using multi-head cross-attention [13], projecting the patch tokens into the LLM's embedding space, as illustrated in Figure 3.2. The cross-attention requires a

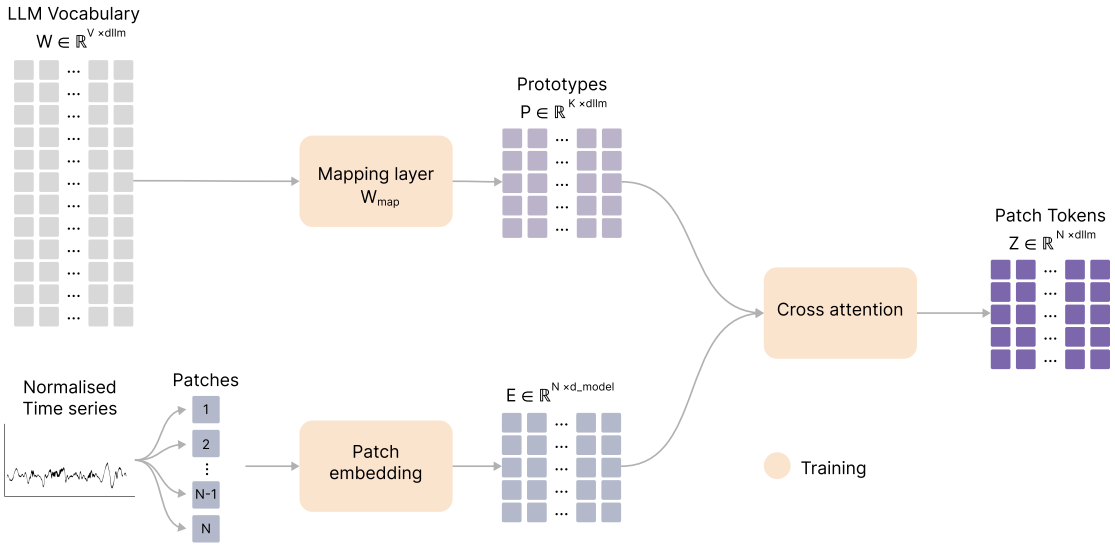


Figure 3.2: Illustration of the reprogramming mechanism from normalised time series into patch tokens

set of key–value vectors that reside in the LLM's embedding space that represent meaningful directions for the patch queries to attend to. These are constructed by compressing the frozen vocabulary embedding matrix $\mathbf{W} \in \mathbb{R}^{V \times d_{\text{llm}}}$ into $k = 1{,}000$ learned *prototypes*:

$$\mathbf{P} = \mathbf{W}_{\text{map}}\,\mathbf{W} \in \mathbb{R}^{k \times d_{\text{llm}}}, \tag{3.4}$$

where $\mathbf{W}_{\text{map}} \in \mathbb{R}^{k \times V}$ is a trainable linear mapping. Each prototype is a learned weighted combination of the LLM's vocabulary embeddings, thereby forming a compact dictionary of directions in LLM space. The choice of $k = 1{,}000$ prototypes follows the Time-LLM default [13]; the original authors tested $k = 500$ and reported worse results. For Qwen2.5-0.5B [44] with $V = 151{,}936$, this mapping layer alone contains approximately 152M parameters, the dominant trainable component in the model (see Table 3.1). With ∼12K training windows, the trainable layers lie in an overparameterised regime relative to the data; whether a smaller k would lower the parameter-to-data ratio without sacrificing performance was not tested.

The N patch embeddings serve as queries, and the k prototypes serve as keys and values. Multi-head attention with $n_{\text{heads}} = 8$ computes, for each head h:

$$\text{head}_h = \text{softmax}\left(\frac{\mathbf{Q}_h \mathbf{K}_h^\top}{\sqrt{d_k}}\right)\mathbf{V}_h, \tag{3.5}$$

where $\mathbf{Q}_h \in \mathbb{R}^{N \times d_k}$ are projected from the patch embeddings ($d_{\text{model}} \to d_k$), $\mathbf{K}_h, \mathbf{V}_h \in \mathbb{R}^{k \times d_k}$ are projected from the prototypes, and $d_k = d_{\text{llm}}/n_{\text{heads}}$ is the per-head dimension, following standard multi-head attention sizing [27]. The heads are concatenated and linearly projected to d_{llm} dimensions. Each patch token is now a d_{llm}-dimensional vector that the LLM can process as if it were a language token.

3.1.4 Prompt Prefix

When a prompt is provided, it is tokenised using the LLM's own tokeniser and its token embeddings are looked up in the frozen embedding table. Because these embeddings already live in $\mathbb{R}^{d_{\text{llm}}}$, the same space the reprogramming layer maps patches into, they can be directly prepended to the reprogrammed patch sequence:

$$\mathbf{H}_{\text{input}} = [\mathbf{H}_{\text{prompt}}\,;\,\mathbf{Z}] \in \mathbb{R}^{(L_{\text{prompt}}+N)\times d_{\text{llm}}}, \tag{3.6}$$

where $\mathbf{H}_{\text{prompt}} \in \mathbb{R}^{L_{\text{prompt}} \times d_{\text{llm}}}$ are the prompt token embeddings, $\mathbf{Z} \in \mathbb{R}^{N \times d_{\text{llm}}}$ are the reprogrammed patch tokens, and L_{prompt} is the number of prompt tokens. The prompt embeddings are not trained; they are used as-is from the frozen LLM.

An attention mask marks real prompt tokens and all N patch positions as active (value 1), while tokeniser padding positions receive value 0. Through the LLM's causal attention, each patch token can attend to all preceding prompt tokens, thereby allowing contextual information to influence the time-series representation. The specific prompt configurations are described in Section 3.2.

3.1.5 Frozen LLM Forward Pass

The combined sequence $\mathbf{H}_{\text{input}}$ is passed through all Transformer layers of the frozen Qwen2.5 backbone (24 layers for 0.5B, 28 layers for 7B). Each layer applies causal self-attention and a feed-forward network, progressively mixing the prompt and patch representations. The LLM's weights remain frozen throughout training; it serves as a fixed feature extractor, whose learned language representations are repurposed for time-series data and multimodal fusion. Gradients flow through the frozen layers during back-propagation, connecting the trainable components on either side.

3.1.6 Output Projection

The LLM produces a hidden state for every input position. The N hidden states corresponding to the patch tokens are extracted, and the prompt positions are discarded. All d_{llm} dimensions of each hidden vector are retained; unlike the original Time-LLM [13], which truncates the hidden states to a smaller dimension, we keep the full representation because the univariate setting produces a modest output projection even at full width. The N hidden vectors are concatenated into a single vector and mapped to the forecast horizon by a fully connected layer:

$$\text{flatten}: \quad \mathbb{R}^{N \times d_{\text{llm}}} \rightarrow \mathbb{R}^{N \cdot d_{\text{llm}}}, \tag{3.7}$$

$$\text{linear}: \quad \mathbb{R}^{N \cdot d_{\text{llm}}} \rightarrow \mathbb{R}^{H}. \tag{3.8}$$

The flattened dimension is $N \cdot d_{\text{llm}}$ and the output dimension is H, both dataset-dependent: for the Bike Sharing dataset ($N = 12$, $H = 48$), this gives 10,752 $\rightarrow$ 48 at the 0.5B scale and 43,008 $\rightarrow$ 48 at the 7B scale.

The forecast is then denormalised using the sample-wise statistics from Subsection 3.1.1 via Equation 3.2, restoring the prediction to the original scale.

3.1.7 Trainable vs. Frozen Parameters

Only four components are trained: the vocabulary mapping layer, the reprogramming cross-attention, the patch embedding convolution, and the output projection head. The frozen LLM backbone accounts for the vast majority of the total parameters but receives no gradient updates.

Table 3.1 summarises the parameter counts for both backbone scales. The mapping layer dominates at both scales, projecting the full vocabulary ($\sim$152K tokens) to 1,000 prototypes. The reprogramming layer is substantially smaller because each head operates in $d_{\text{llm}}/n_{\text{heads}}$ dimensions rather than the full d_{llm}, following standard multi-head attention sizing.

Table 3.1: Parameter counts for both backbone scales. Only the first four components are trainable; the LLM backbone is entirely frozen.

Component	0.5B	7B
Mapping layer (vocab $\rightarrow$ 1,000 prototypes)	$\sim$152M	$\sim$152M
Reprogramming layer (cross-attention)	$\sim$2.4M	$\sim$38.6M
Patch embedding (Conv1d)	$\sim$1K	$\sim$1K
Output projection (FlattenHead)	$\sim$516K	$\sim$2.1M
Total trainable	**$\sim$155M**	**$\sim$193M**
Qwen2.5 backbone (frozen)	$\sim$494M	$\sim$7.6B

3.2 Prompt Configurations

Four prompt configurations are tested at each backbone scale, yielding eight LLM experiments in total. All variants share identical architecture, hyperparameters, and

training procedure; only the prompt input differs. The full prompt texts are provided in Section A.2.

3.2.1 No Prompt (TS Only)

The LLM receives only the N reprogrammed patch tokens with no textual input. This serves as the baseline configuration, isolating the contribution of the time-series alignment mechanism and frozen LLM representations alone.

3.2.2 Static Context

A fixed natural-language description is prepended to every sample. It describes the domain (Capital Bikeshare, hourly rental counts), known periodicities (daily, weekly, seasonal), and the forecasting task. The prompt is identical across all samples and batches, it provides generic domain knowledge but no sample-specific information. The exact prompt is provided in Subsection A.2.1.

3.2.3 Sample-wise Context

The static description is extended with a sample-wise instance description generated from the metadata of each window. This instance description specifies the start and end datetimes of both the history and forecast windows, the days of the week they span, and the partitioning of hours across those days. Each sample in a batch, therefore, receives a unique prompt that explicitly encodes calendar information, day of the week, time of day, and position within the week, which the model cannot infer from the numerical time series alone. An example of a prompt containing sample-wise context is provided in Subsection A.2.2.

3.2.4 Shuffled Sample-wise Context (Ablation)

The same sample-wise instance descriptions from Subsection 3.2.3 are used, but randomly permuted across samples using a fixed seed. This breaks the correspondence between each time-series window and its textual description: a Monday-morning window might receive the context of a Saturday-evening window.

This ablation is the key control for distinguishing semantic content from token-level conditioning. If the shuffled variant performs comparably to the correctly paired variant, then the model benefits from the presence of text tokens regardless of their content. If performance degrades, evidence points toward the model extracting and using specific calendar information in the prompt.

3.3 Dataset

This thesis uses two time-series datasets: a Bike Sharing dataset and a Telecom KPI dataset. The Bike Sharing dataset, based on the Capital Bikeshare system in Washington, D.C. [18], is used in the eight-experiment ablation study. The Telecom

KPI dataset, comprising operational performance metrics from a RAN, is used as a case study for one configuration (see Subsection 3.5.1).

3.3.1 Bike Sharing Dataset

Bike rental demand exhibits properties that make it well-suited for investigating the effect of textual context: strong daily periodicity (commuting peaks on weekdays, midday peaks on weekends), weekly periodicity (weekday vs. weekend patterns), and seasonal trends (higher ridership in summer, lower in winter). Importantly, demand also deviates from these regular patterns on holidays and special events, precisely the situations where calendar context should provide the most value.

All Bike Sharing experiments use the Capital Bikeshare dataset, which records hourly bike rental counts from the Capital Bikeshare system in Washington, D.C., spanning 2011–2012 [18]. The raw data contain 165 missing hours (0.95% of the series), which are imputed with the most recent observed value (forward filling), yielding a continuous series of 17,544 hourly observations.

The raw hourly counts are preprocessed into sliding windows with a lookback of $L = 96$ hours (4 days) and a forecast horizon of $H = 48$ hours (2 days), using a stride of 1 hour. Each window is paired with metadata (anchor datetime, day of week, and hour partitioning) to generate sample-wise context.

A global z-normalisation is fit to the training set's historical values, yielding a single mean and standard deviation. This transformation is applied identically to all splits (train, validation, test). The global step places the series in a consistent numeric range across the dataset, while the sample-wise RevIN normalisation inside the model (Subsection 3.1.1) standardises each input window independently, removing sample-specific level and scale variation. The two normalisation stages are therefore complementary: global normalisation stabilises the overall data distribution, and RevIN ensures that each sample presented to the model has zero mean and unit variance regardless of when in the series it was drawn. Table 3.2 summarises the split sizes.

Table 3.2: Bike Sharing dataset split. All splits are chronological and non-overlapping; the anchor datetime column indicates the first time step of each window's lookback.

Split	Windows	First anchor	Last anchor
Train	12,180	2011-01-01 00:00	2012-05-22 11:00
Validation	1,740	2012-05-24 12:00	2012-08-04 23:00
Test	3,481	2012-08-07 00:00	2012-12-30 00:00
Total	17,401		

The Bike Sharing dataset is split chronologically into training, validation, and test sets using a 70/10/20 ratio to prevent future information leakage.

3.3.2 Telecom KPI Dataset

RAN KPIs typically exhibit daily and weekly seasonality reflecting human activity patterns, but are also subject to abrupt deviations driven by load surges, configuration changes, and external events [45]. Compared to the Bike Sharing dataset, this setting differs along several axes: the time span is much shorter (about 87 days versus 24 months), the sampling frequency is higher (15 minutes versus 1 hour) and the data's behaviour is far more irregular and bursty. Additionally, it contains three KPIs with distinct temporal structures. The dataset is used in Section 4.8 as an external validity check for the best LLM configuration identified on the Bike Sharing dataset; the four research questions are not re-evaluated here (see Subsection 3.5.1).

The Telecom KPI dataset, supplied by Ericsson, contains operational measurements from Long Term Evolution (LTE) [46] and has a 15 minute frequency. The cells are part of a Radio Access Network (RAN) [47], the layer of mobile infrastructure that connects user devices to the operator's core network through base stations and their antennas [47]. Operators monitor the health of the RAN using Key Performance Indicators (KPIs): metrics aggregated at regular intervals that quantify performance dimensions such as throughput, latency, cell availability, and energy consumption. In this dataset, each cell is observed on three KPIs: two data-rate KPIs covering the downlink and uplink directions, and one latency KPI, giving 18 distinct time series of 8,640 timestamps each, or 155,520 observations in total.

Each forecasting sample uses a lookback of $L = 144$ time steps (36 hours) and a forecast horizon of $H = 96$ time steps (24 hours). Sliding windows are extracted for each series with a stride of 8 (one window every two hours). Normalisation is applied per KPI, with separate means and standard deviations fit on the training portion of each KPI, shared across its six cells, placing the three KPIs in comparable numeric ranges while preserving relative magnitudes between cells. The per-window RevIN normalisation from Subsection 3.1.1 is applied inside the model as before. When textual context is used, it follows the same template as the Bike Sharing sample-wise prompt (Subsection 3.2.3): a dataset description, a task description, and an instance description specifying the timestamp range, day-of-week partitioning, and hour partitioning of each forecast window. The KPI prompt also includes a cell-specific section providing information such as the cell identifier and its geographical location. The literal prompt text is confidential and is not provided.

The 18 series (six cells across three KPIs) are forecast as 18 independent univariate problems. A single model is trained on the union of all 18 series and applied to each series at inference time, with the cell-specific section of the prompt providing the only signal that distinguishes one cell from another. No information is shared between cells or across KPIs during forecasting, and the model treats each (cell, KPI) pair as its own forecasting task.

The dataset is split chronologically into training, validation, and test sets using a 70/10/20 ratio over the full timeline. Sliding-window construction over the 18 series at a stride of 8 produces 18,378 candidate windows in total. Windows whose lookback or forecast horizon crosses a split boundary are dropped to avoid leakage, leaving 17,334 windows used in training and evaluation. Table 3.3 summarises the

resulting split sizes.

Table 3.3: Telecom KPI dataset split. The full timeline is partitioned chronologically; windows whose lookback or forecast horizon would cross a split boundary are dropped.

Split	Windows	First anchor	Last anchor
Train	12,708	2025-02-01 00:00	2025-04-03 05:45
Validation	1,368	2025-04-04 05:45	2025-04-12 23:45
Test	3,258	2025-04-13 23:45	2025-04-30 11:45
Total	17,334		

3.4 Baselines

Four baselines of increasing complexity are included to contextualise the LLM-based results. For a detailed description of these baselines, we refer to Section 2.1.

Seasonal Naive: the simplest baseline exploits daily periodicity, each forecast step copies the value from one day earlier, i.e. $\hat{x}_{t+h} = x_{t+h-P}$ for $h = 1, \ldots, H$, where P is the number of samples per day. On the Bike Sharing dataset, $P = 24$ (hourly sampling) and $H = 48$, so the forecast reproduces the most recent two daily cycles from the lookback window. On the Telecom KPI dataset, $P = 96$ (15-minute sampling) and $H = 96$, so the forecast reproduces the most recent daily cycle. This captures the dominant diurnal pattern but cannot adapt to trends, day-of-week effects, or irregular events. It requires no training.

DLinear: in our implementation of DLinear [23], input is decomposed into a trend and a remainder (seasonal) component using a moving average with kernel size 25, and a separate linear layer maps each component from L steps to H steps. The two outputs are summed to produce the forecast (9,312 parameters).

LSTM: a 2-layer LSTM encoder [24], [25] (hidden size 64) reads the L-step input sequentially and maps the final hidden state to an H-step forecast via a linear layer (53,552 parameters). This provides a recurrent baseline that, unlike DLinear, can capture non-linear temporal dependencies.

PatchTST: a Transformer encoder operating on patched time series input with channel independence. PatchTST [21] trains its own Transformer from scratch rather than repurposing a frozen LLM. With 474,928 parameters, it remains among the top-performing pure time-series Transformer models. It serves as the primary baseline for assessing whether the frozen LLM backbone adds value beyond what a purpose-built architecture achieves.

3.5 Experimental Setup

This section specifies the experimental setup. We first list the eight LLM experiments comprising the controlled ablation on the Bike Sharing dataset (Table 3.4). Subsection 3.5.1 then describes the narrower scope of the Telecom KPI case study, which re-trains

only the best LLM configuration alongside the four baselines as an external validity check. Subsection 3.5.2 details the training configuration, which is shared across all experiments and both datasets.

Table 3.4 lists all eight LLM experiments. Each experiment addresses one or more research questions by comparing it with other experiments using pairwise comparisons.

Table 3.4: List of experiments. Each experiment shares an identical architecture and hyperparameters; only the prompt configuration and backbone scale differ.

ID	Backbone	Prompt variant	Answers
E1	Qwen2.5-0.5B	TS only	Baseline
E2	Qwen2.5-0.5B	Static context	RQ1
E3	Qwen2.5-0.5B	Sample-wise context	RQ2
E4	Qwen2.5-0.5B	Sample-wise context (shuffled)	RQ3
E5	Qwen2.5-7B	TS only	RQ4, baseline
E6	Qwen2.5-7B	Static context	RQ1, RQ4
E7	Qwen2.5-7B	Sample-wise context	RQ2, RQ4
E8	Qwen2.5-7B	Sample-wise context (shuffled)	RQ3, RQ4

3.5.1 Scope on the Telecom KPI Dataset

The eight experiments in Table 3.4 are run on the Bike Sharing dataset only. On the Telecom KPI dataset (Subsection 3.3.2), we re-train Qwen2.5-7B with sample-wise context (E7), the largest and most prompt-rich variant in Table 3.4, together with the four baselines. The remaining seven prompt-by-scale combinations are not re-tested, since the case study targets the narrower question of whether E7 generalises to industrial data rather than re-running the full ablation on a second domain; the four research questions of Chapter 1 are therefore answered using only the Bike Sharing dataset. The case study reports E7 against the four baselines in Section 4.8, using pooled and per-KPI metrics.

3.5.2 Training Configuration

All eight experiments are trained with identical hyperparameters. The peak learning rate of (5×10^{-5}) was set based on preliminary trial runs to ensure it was in the right order of magnitude. Because OneCycleLR sweeps the learning rate through a wide range during training, the peak value need not be precisely optimal for stable convergence [48]. The same value was held fixed across all eight experiments to avoid confounding prompt and backbone effects with per-configuration tuning. The batch size of 16 is the maximum that fits in GPU memory for the largest configuration (7B with sample-wise context). To ensure identical training conditions across all eight experiments, the same batch size is used even for smaller configurations where larger batches would fit in memory. The only difference between experiments is the prompt configuration and backbone size.

The OneCycleLR scheduler ramps the learning rate from near zero to 5×10^{-5} over the first 20% of training steps, then anneals it back to near zero over the remaining

80%. All models are trained for 50 epochs; the checkpoint with the lowest validation MSE is saved and used for the final test evaluation. The test set is evaluated only once, after training completes, to prevent any leakage into model selection.

All models are trained on a single NVIDIA H100 GPU. The training configuration is detailed in Table 3.5.

Table 3.5: Training hyperparameters, shared across all experiments.

Setting	Value
Optimiser	Adam [49]
Learning rate	5×10^{-5}
LR scheduler	OneCycleLR [48] (20% warmup)
Loss function	MSE
Epochs	50
Batch size	16
Model selection	Best validation MSE checkpoint

3.6 Evaluation Methodology

This section describes how forecasting performance is evaluated and compared across models. In addition to reporting aggregate error metrics, the evaluation uses statistical tests to assess whether the observed performance differences are statistically significant.

3.6.1 Metrics

Two standard forecasting metrics are reported for all models:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{H} \sum_{t=1}^{H} \left(\hat{y}_{i,t} - y_{i,t} \right)^2 , \tag{3.9}$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{H} \sum_{t=1}^{H} \left| \hat{y}_{i,t} - y_{i,t} \right| , \tag{3.10}$$

where n is the number of test windows (3,481 for the Bike Sharing dataset, 3,258 for the Telecom KPI dataset), $\hat{y}_{i,t}$ is the predicted value, and $y_{i,t}$ is the ground truth. Metrics are computed on normalised values: the Bike Sharing dataset uses the global z-normalisation described in Subsection 3.3.1, while the Telecom KPI dataset uses the per-KPI normalisation described in Subsection 3.3.2, ensuring that the three KPIs contribute comparably to pooled metrics.

3.6.2 Diebold-Mariano Test

Aggregate metrics provide point estimates of model performance but do not indicate whether differences between models are statistically significant. To make rigorous

pairwise claims, model pairs are compared using the Diebold-Mariano (DM) test with Newey-West HAC standard errors (Subsection 2.4.1, Subsection 2.4.2). The loss function $\mathcal{L}$ is squared error, averaged over the H forecast steps. Because the sliding-window stride means that consecutive test windows share most of their forecast horizon, the Newey-West lag truncation parameter is set to $\ell = \lceil H/s \rceil - 1$ (where s is the sliding-window stride), so that only windows with non-overlapping forecast horizons are treated as approximately independent. This gives $\ell = 47$ on the Bike Sharing dataset ($H = 48$, $s = 1$) and $\ell = 11$ on the Telecom KPI dataset ($H = 96$, $s = 8$; see Subsection 3.3.2).

3.6.3 Pairwise Comparisons

Fourteen pairwise Diebold-Mariano (DM) tests are conducted. Table 3.6 lists the ten within-LLM comparisons covering all four research questions. The remaining four tests compare each baseline (Seasonal Naive, DLinear, LSTM, PatchTST) against the strongest LLM variant, as determined by the results in Chapter 4.

Table 3.6: Pairwise DM test comparisons among the eight LLM experiments.

Comparison	Tests	RQ
Within 0.5B		
E1 vs. E2	TS only vs. Static context	RQ1
E1 vs. E3	TS only vs. Sample-wise context	RQ2
E2 vs. E3	Static context vs. Sample-wise context	RQ2
E4 vs. E3	(Shuffled) Sample-wise context vs. Sample-wise context	RQ3
Within 7B		
E5 vs. E6	TS only vs. Static context	RQ1, RQ4
E5 vs. E7	TS only vs. Sample-wise context	RQ2, RQ4
E6 vs. E7	Static context vs. Sample-wise context	RQ2, RQ4
E8 vs. E7	(Shuffled) Sample-wise context vs. Sample-wise context	RQ3, RQ4
Cross-scale		
E1 vs. E5	TS only: 0.5B vs. 7B	RQ4
E3 vs. E7	Sample-wise context: 0.5B vs. 7B	RQ4

All tests are two-sided with significance assessed at $\alpha = 0.05$. In Chapter 4, significance levels are denoted as follows: * for $p < 0.05$, ** for $p < 0.01$, and *** for $p < 0.001$.

No multiple-comparison correction (e.g. Bonferroni [50]) is applied. The fourteen tests are not independent, they share overlapping test windows and models, and the comparisons are structured by the four research questions rather than being exploratory. Nonetheless, results near the significance threshold (particularly p-values between 0.01 and 0.05) should be interpreted with caution, as they might not survive a conservative multiplicity adjustment.

The same Diebold-Mariano procedure with Newey-West HAC standard errors is applied to the Telecom KPI case study in Section 4.8, comparing the re-trained E7 against each of the four baselines, both pooled across KPIs and per KPI. The

Newey-West lag for that dataset ($\ell = 11$) follows from the same $\lceil H/s \rceil - 1$ rule used here.

3.6.4 Significant Date Subset Analysis

To investigate whether performance differences between the LLM variants and PatchTST are concentrated on specific types of samples, the DM test is additionally run on two subsets of the test data:

1. **Significant dates**: a test window is active on a significant date if any timestamp belonging to that date falls within its H-hour forecast horizon. Seven significant dates are defined for the test period (Washington, D.C., 2012): Labour Day (3 Sep), 9/11 Memorial (11 Sep), Columbus Day (8 Oct), Veterans Day (12 Nov), Thanksgiving (22 Nov), Christmas Eve (24 Dec), and Christmas Day (25 Dec) [51]. Each individual date is active for 71 windows; after taking the union and deduplicating windows that overlap consecutive significant dates (primarily Christmas Eve and Christmas Day), this yields 450 of $n = 3{,}481$ test windows.
2. **Normal dates**: the remaining 3,031 test windows whose forecast horizon does not overlap any significant date.

For each subset, a separate DM test is conducted on the MSE-based loss differentials. The same Newey-West lag ($\ell = 47$) is used for both subsets. Because significant-date windows cluster around specific calendar periods rather than being uniformly distributed across the test set, the autocorrelation structure within this subset differs from that of the full test set, and the HAC correction may not fully account for the dependence. Results from the significant-date subset should therefore be interpreted with appropriate caution. This analysis tests the hypothesis that textual context is most informative when demand deviates from regular weekly patterns.

4

Results

This chapter presents the experimental results on the Bike Sharing dataset, except for Section 4.8, which applies the best LLM configuration to the Telecom KPI dataset as a case study. Section 4.1 summarises the performance of all models. Section 4.2 through Section 4.5 address each research question in turn. Section 4.6 compares the LLM variants against the baselines, and Section 4.7 examines performance on significant dates. All reported p-values are from two-sided Diebold-Mariano tests with Newey-West HAC standard errors (Subsection 3.6.2).

4.1 Overview of Results

Table 4.1 summarises the test set performance of all twelve models: eight LLM experiments (E1–E8), three trained baselines (DLinear, LSTM, PatchTST), and one non-parametric baseline (Seasonal Naive). All metrics are computed on the normalised values defined in Subsection 3.6.1.

Table 4.1: Results for all models on the Bike Sharing test set. The best result in each metric column is shown in bold.

Backbone	Prompt variant	ID	Test MSE	Test MAE
Qwen2.5-0.5B	TS only	E1	0.3720	0.4017
Qwen2.5-0.5B	Static context	E2	0.3728	0.4034
Qwen2.5-0.5B	Sample-wise context	E3	0.3661	0.3994
Qwen2.5-0.5B	Sample-wise context (shuffled)	E4	0.3785	0.4071
Qwen2.5-7B	TS only	E5	0.4333	0.4457
Qwen2.5-7B	Static context	E6	0.4162	0.4280
Qwen2.5-7B	Sample-wise context	E7	**0.3562**	**0.3933**
Qwen2.5-7B	Sample-wise context (shuffled)	E8	0.4235	0.4341
Baselines				
Seasonal Naive	—	—	1.4386	0.7777
DLinear	—	—	0.7188	0.5855
LSTM	—	—	0.4728	0.4607
PatchTST	—	—	0.3805	0.4082

Figure 4.1 provides a visual comparison. Several patterns are immediately apparent. First, all LLM variants and PatchTST substantially outperform the simpler baselines

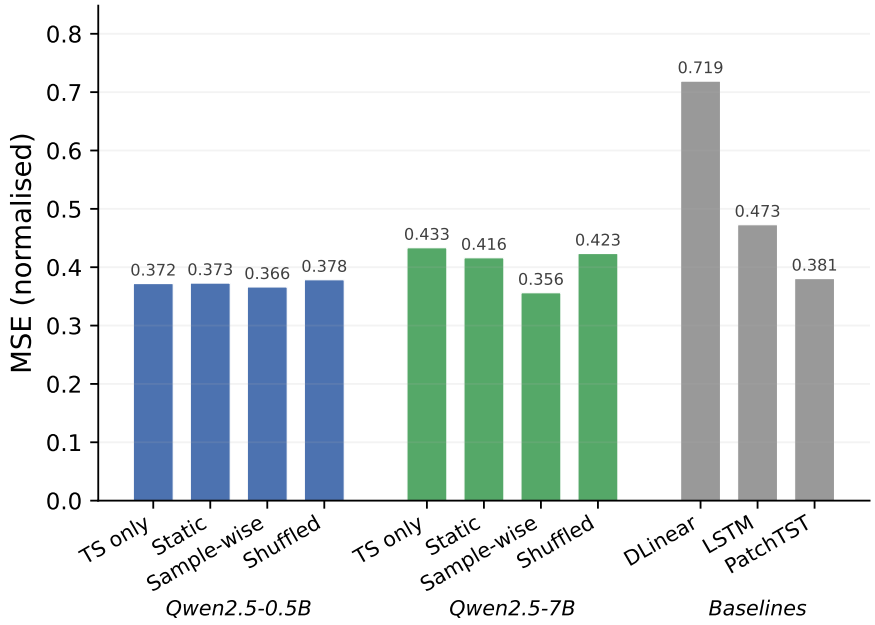
(Seasonal Naive, DLinear, and LSTM). Second, the 0.5B experiments (E1–E4) cluster tightly between 0.366 and 0.379, whereas the 7B experiments (E5–E8) span a much wider range from 0.356 to 0.433. Third, the best overall model is E7 (7B with sample-wise context), while the worst LLM variant is E5 (7B without any textual context), both from the larger backbone.

Figure 4.2 shows the training and validation MSE over the 50-epoch training budget. Most training curves show smooth loss reduction with no signs of overfitting. Some configurations exhibit transient spikes during the learning-rate warm-up in early epochs, the most noticeable being E5 (7B TS only), before recovering and converging normally.

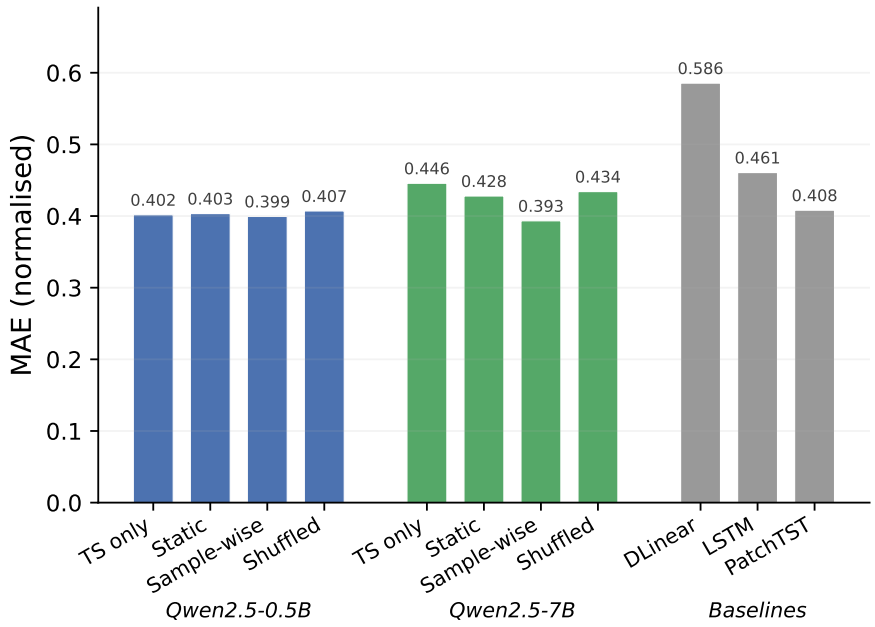
Table 4.2 reports the computational cost of each experiment. Adding context tokens substantially increases both memory usage and training time, particularly at the 7B scale, where sample-wise context requires roughly 11× longer per epoch and nearly twice the VRAM compared to TS-only.

Table 4.2: Computational cost per experiment. Shuffled variants (E4, E8) are identical to their sample-wise counterparts and omitted.

ID	Backbone	Prompt variant	Time/epoch [min]	Peak VRAM [GB]
E1	Qwen2.5-0.5B	TS only	0.6	5.5
E2	Qwen2.5-0.5B	Static context	1.6	9.0
E3	Qwen2.5-0.5B	Sample-wise context	2.6	12.4
E5	Qwen2.5-7B	TS only	3.0	37.6
E6	Qwen2.5-7B	Static context	18.9	54.8
E7	Qwen2.5-7B	Sample-wise context	32.4	71.1

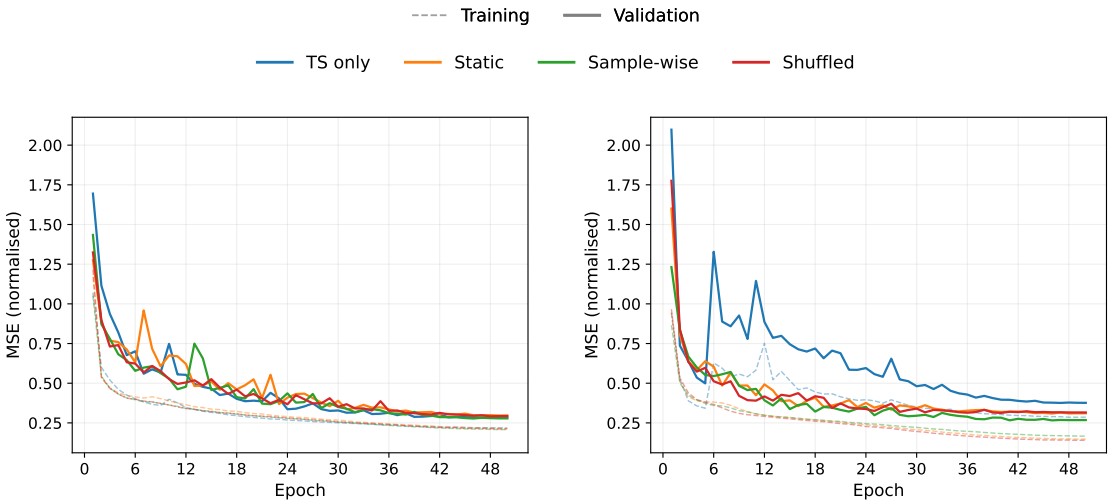


(a) Test MSE.



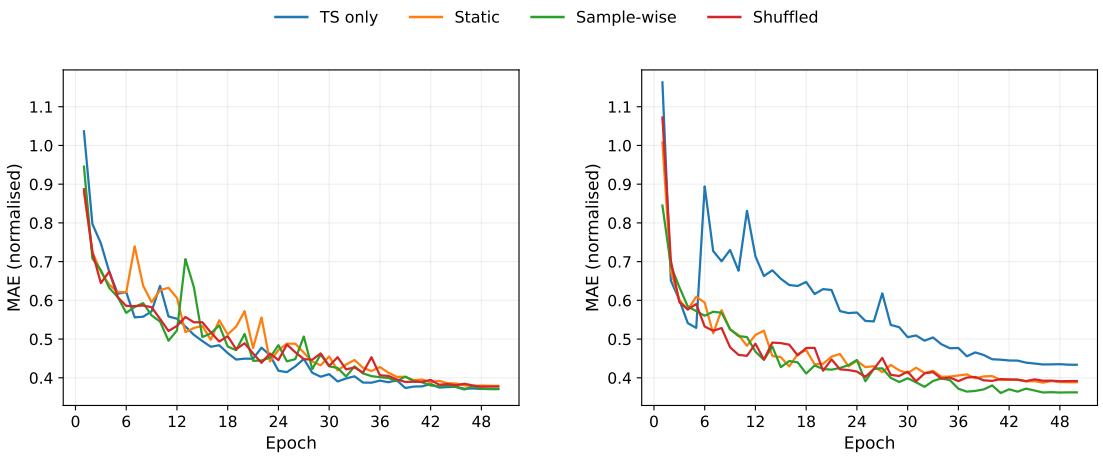
(b) Test MAE.

Figure 4.1: Test MSE (a) and MAE (b) across all models. LLM experiments are grouped by backbone scale; baselines are shown separately. Seasonal Naive is omitted for scale (MSE = 1.4386, MAE = 0.7777).



(a) Qwen2.5-0.5B variants (E1–E4). (b) Qwen2.5-7B variants (E5–E8).

Figure 4.2: Training (dashed) and validation (solid) MSE over 50 epochs for the Qwen2.5-0.5B variants (a) and the Qwen2.5-7B variants (b).



(a) Qwen2.5-0.5B variants (E1–E4). (b) Qwen2.5-7B variants (E5–E8).

Figure 4.3: Validation MAE over 50 epochs for the Qwen2.5-0.5B variants (a) and the Qwen2.5-7B variants (b).

4.2 RQ1: Static Prompt Effect

Does a fixed dataset-level description improve forecasting performance compared to using the time series alone?

Table 4.3 shows the DM test results for adding static context at each backbone scale.

Adding static context has no significant effect at either backbone scale. At 0.5B, static context marginally worsens the MSE (0.3720 to 0.3728), whereas at 7B it marginally improves the MSE (0.4333 to 0.4162). Neither change is statistically significant. A fixed domain description identical across all samples provides no information beyond what the model can already learn from the time-series patterns.

Table 4.3: DM test results for RQ1. A positive statistic indicates that the prompted variant has lower loss.

Comparison	Test	DM stat.	p-value
E1 vs E2	0.5B: TS only vs static	-0.167	0.868
E5 vs E6	7B: TS only vs static	$+1.050$	0.294

4.3 RQ2: Sample-wise Context Effect

Does sample-wise contextual text improve performance compared to a static prompt or no prompt?

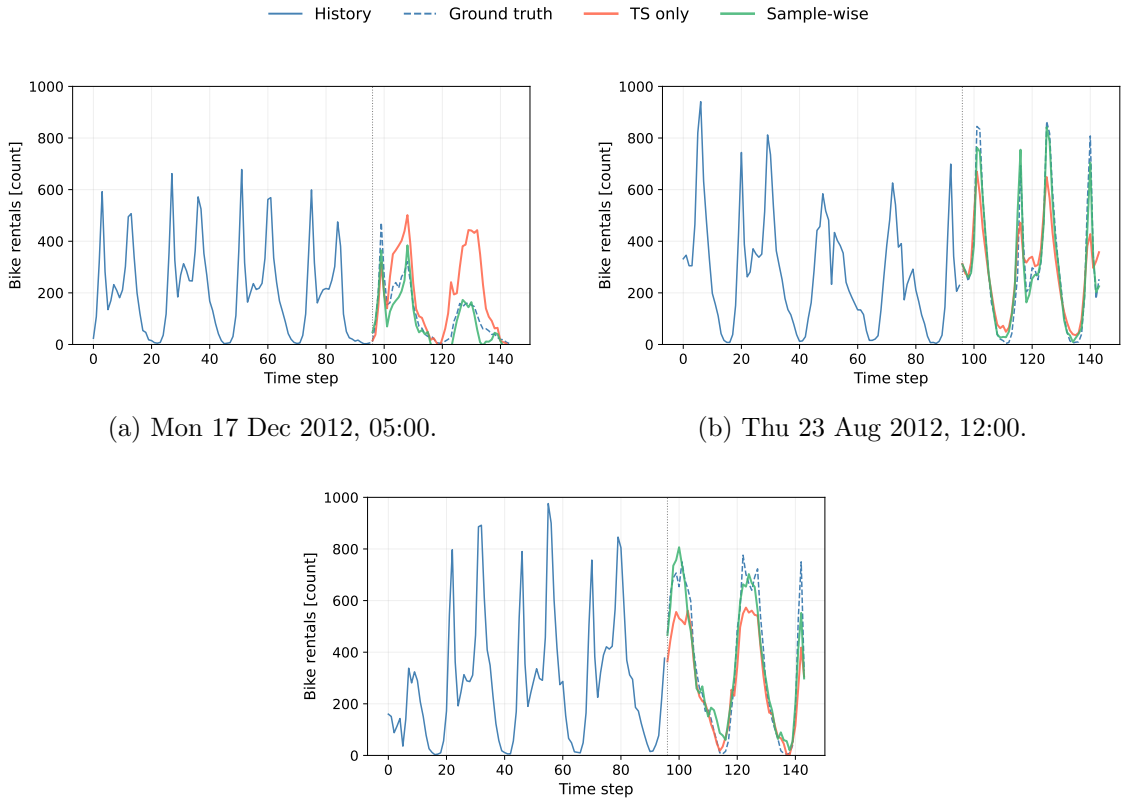
Table 4.4 shows the DM test results for adding sample-wise context at each backbone scale, compared against both the TS-only and static-context baselines.

Table 4.4: DM test results for RQ2. A positive statistic indicates that the sample-wise variant has lower loss.

Comparison	Test	DM stat.	p-value
E1 vs E3	0.5B: TS only vs sample-wise	$+1.038$	0.299
E2 vs E3	0.5B: static vs sample-wise	$+1.584$	0.113
E5 vs E7	7B: TS only vs sample-wise	$+5.763$	$<0.001^{***}$
E6 vs E7	7B: static vs sample-wise	$+4.654$	$<0.001^{***}$

The effect of sample-wise context is scale-dependent. At 0.5B, sample-wise context shows a directional improvement (MSE decreases from 0.3720 to 0.3661), but the difference is not statistically significant. At 7B, sample-wise context produces a large, highly significant improvement: compared to TS only, MSE drops from 0.4333 to 0.3562 (an 18% reduction, $p < 0.001$); compared to static context, MSE drops from 0.4162 to 0.3562 (a 14% reduction, $p < 0.001$).

The 7B model with sample-wise context (E7) achieves the best performance among all variants across both scales. This is particularly striking because the 7B backbone performs *worst* without context (E5, MSE = 0.4333). Figure 4.4 illustrates this on three Bike Sharing test windows: panels (a) and (b) are windows where sample-wise context improves noticeably over TS only, while panel (c) is randomly drawn.



(a) Mon 17 Dec 2012, 05:00.

(b) Thu 23 Aug 2012, 12:00.

(c) Tue 18 Sep 2012, 10:00.

Figure 4.4: Forecast comparison of E5 (Qwen2.5-7B, no context) and E7 (Qwen2.5-7B, sample-wise context) on three test windows from the Bike Sharing dataset. (a) and (b) are windows where sample-wise context produces visibly more accurate forecasts; (c) is a randomly drawn comparator.

4.4 RQ3: Semantic Content vs. Token Conditioning

Does the content of the sample-wise context matter, or does any text prefix act as a generic conditioning signal?

The correctly paired sample-wise context is compared against the shuffled ablation at each backbone scale in Table 4.5.

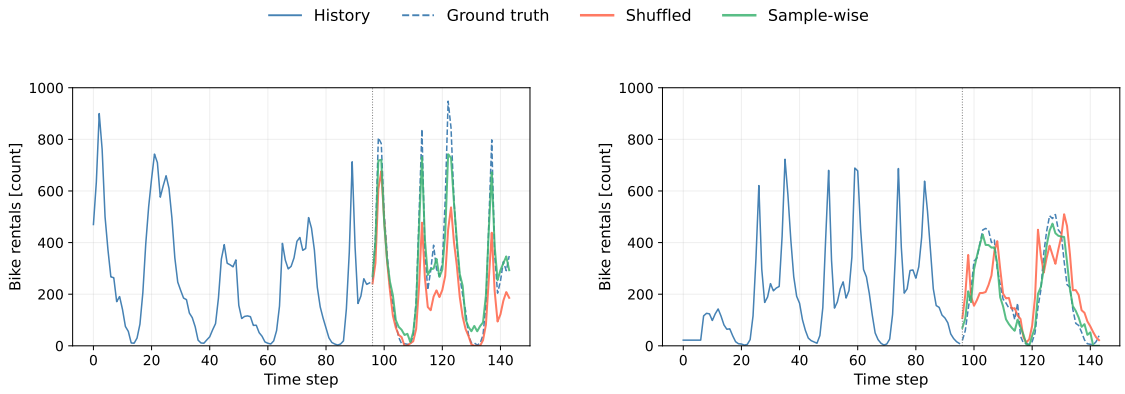
Table 4.5: DM test results for RQ3. A positive statistic indicates that the correctly paired variant has lower loss.

Comparison	Test	DM stat.	p-value
E4 vs E3	0.5B: shuffled vs correct	+1.845	0.065
E8 vs E7	7B: shuffled vs correct	+5.269	$<0.001^{***}$

At 0.5B, the shuffled context (E4, MSE = 0.3785) performs marginally worse than the correctly paired context (E3, MSE = 0.3661), with $p = 0.065$, close to the

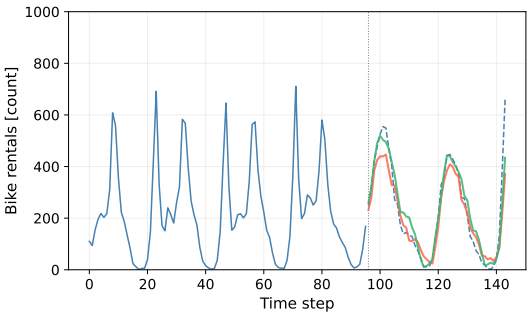
significance threshold but not crossing it. This is consistent with RQ2: the 0.5B backbone does not fully leverage textual context, regardless of whether the context is correct or shuffled.

At 7B, the correctly paired context (E7, MSE = 0.3562) significantly outperforms the shuffled version (E8, MSE = 0.4235), with $p < 0.001$. Shuffling the context, breaking the correspondence between each time series window and its calendar description, degrades performance to the level of the static-context variant (E6, MSE = 0.4162). This indicates that mismatched sample-wise text provides roughly the same benefit as generic text, and no more. The results are consistent with the model conditioning on the specific calendar information in the prompt; when that information is incorrect, the benefit disappears. Figure 4.5 shows three Bike Sharing test windows: panels (a) and (b) are windows where correctly paired context outperforms shuffled context noticeably, while panel (c) is randomly drawn.



(a) Fri 05 Oct 2012, 15:00.

(b) Tue 30 Oct 2012, 06:00.



(c) Tue 13 Nov 2012, 09:00.

Figure 4.5: Forecast comparison of E8 (Qwen2.5-7B, shuffled context) and E7 (Qwen2.5-7B, sample-wise context) on three test windows from the Bike Sharing dataset. (a) and (b) are windows where the mismatched calendar information degrades the shuffled variant's forecast; (c) is a randomly drawn comparator.

4.5 RQ4: Backbone Scale Interaction

How does the size of the frozen LLM backbone interact with the effect of textual context?

Table 4.6 compares the two backbone scales under matched prompt configurations.

Table 4.6: DM test results for RQ4. A positive statistic indicates that the 7B variant has lower loss.

Comparison	Test	DM stat.	p-value
E1 vs E5	TS only: 0.5B vs 7B	−7.630	$<0.001^{***}$
E3 vs E7	Sample-wise: 0.5B vs 7B	+1.099	0.272

Without textual context, the 0.5B model significantly outperforms the 7B model (MSE 0.3720 vs 0.4333, $p < 0.001$). The larger backbone's additional capacity is not beneficial and is even harmful when the model receives only time-series input. With sample-wise context, the 7B model closes the gap entirely: the difference between 0.5B (MSE = 0.3661) and 7B (MSE = 0.3562) is no longer significant ($p = 0.272$), and the 7B point estimate is the lowest of all variants.

Figure 4.6 visualises this interaction. The 0.5B backbone is relatively insensitive to the context configuration, while the 7B backbone spans the full range from worst to best depending on the textual input it receives.

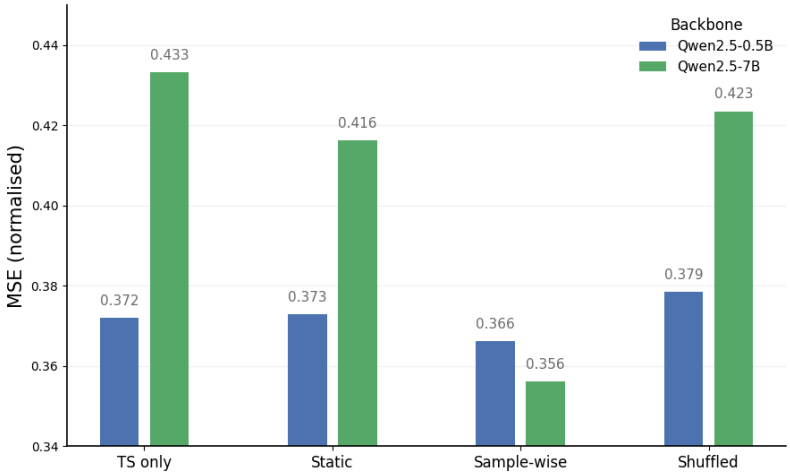


Figure 4.6: Interaction between backbone scale and context configuration.

4.6 Baseline Comparison

Table 4.7 reports the DM test results comparing each baseline against the best LLM variant (E7, 7B with sample-wise context).

Table 4.7: DM test results comparing baselines against E7 (7B sample-wise). A positive statistic indicates that E7 has lower loss.

Baseline	MSE	E7 MSE	DM stat.	p-value
Seasonal Naive	1.4386	0.3562	+12.627	$<0.001^{***}$
DLinear	0.7188	0.3562	+13.475	$<0.001^{***}$
LSTM	0.4728	0.3562	+4.605	$<0.001^{***}$
PatchTST	0.3805	0.3562	+1.578	0.115

All LLM variants significantly outperform Seasonal Naive, DLinear, and LSTM ($p < 0.001$). The comparison with PatchTST is more nuanced: E7 achieves a lower point estimate (MSE 0.3562 vs 0.3805), but the difference is not statistically significant ($p = 0.115$). On aggregate metrics, the LLM-based approach matches but does not clearly outperform a purpose-built time-series Transformer. However, as the next section shows, this aggregate view masks an important distinction between different types of test samples.

4.7 Significant Date Analysis

The aggregate comparison between E7 and PatchTST is not significant ($p = 0.115$). However, sample-wise context should be most valuable on dates when demand deviates from regular weekly patterns, holidays and special events that the frozen LLM may recognise from its pretraining data. To test this, the 3,481 test windows are partitioned into two subsets as defined in Subsection 3.6.4: significant-date windows whose forecast horizon overlaps with a calendar event ($n = 450$), and normal-date windows that do not ($n = 3{,}031$). A separate DM test is run on each subset.

Table 4.8 shows that the advantage of sample-wise context is concentrated on significant-date windows. On normal dates, E7 and PatchTST perform comparably ($p = 0.384$). On significant dates, E7 reduces MSE by 18.4% relative to PatchTST and the difference is significant ($p = 0.043$). The corresponding reduction on normal dates is 3.7%.

Table 4.8: DM test results comparing E7 (7B sample-wise) against PatchTST on significant-date and normal-date subsets. A positive statistic indicates that E7 has lower loss.

Subset	n	E7 MSE	PatchTST MSE	DM stat.	p-value
Significant dates	450	0.4605	0.5644	+2.026	0.043^{*}
Normal dates	3,031	0.3407	0.3539	+0.871	0.384

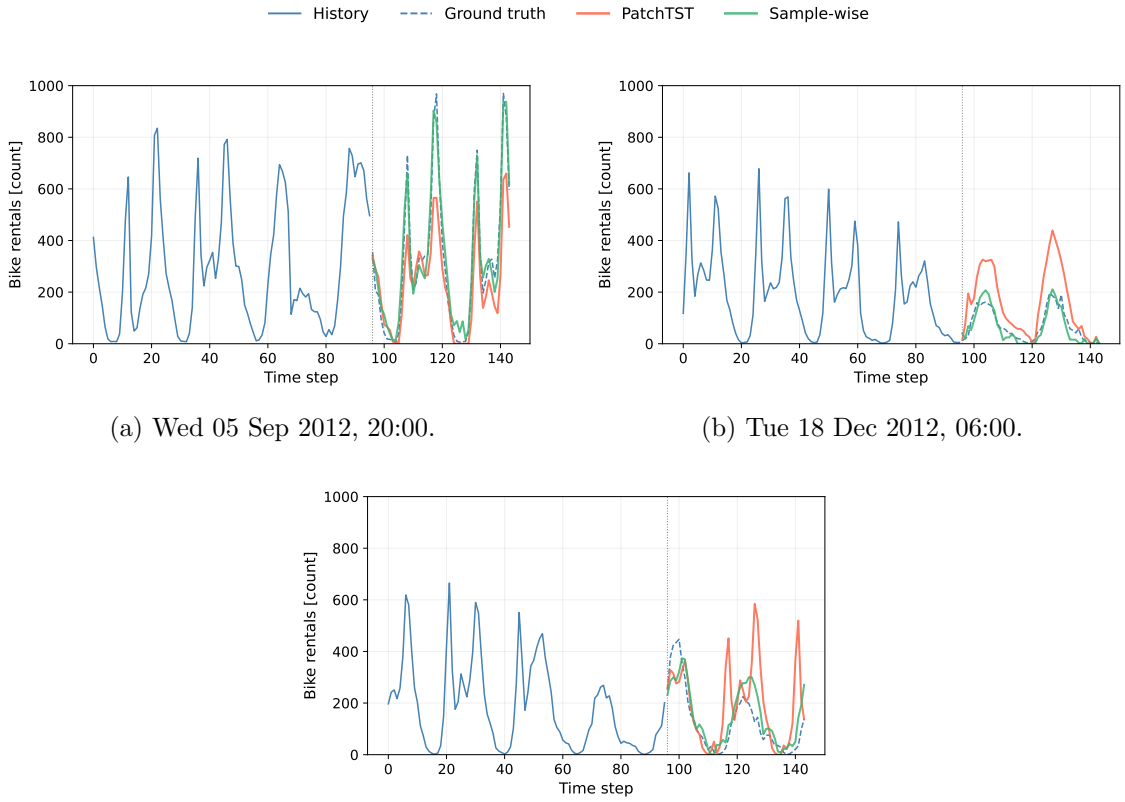
Table 4.9 breaks the significant-date result down by individual date. The effect is heterogeneous rather than uniform: E7 produces large MSE reductions on 9/11 Memorial, Christmas Eve, Christmas Day, and Labor Day; performs comparably on Thanksgiving; and is outperformed by PatchTST on Veterans Day and Columbus

Day. The strongest gains occur on well-known dates associated with disruptions to daily routines, while the negative results suggest that mismatched or weakly informative context can hurt as well as help. We return to this heterogeneity in Subsection 5.1.3.

Table 4.9: Per-date MSE reduction of E7 relative to PatchTST on the seven significant dates. A positive MSE reduction indicates that E7 has lower loss.

Date	n	E7 MSE	PatchTST MSE	MSE reduction
9/11 Memorial	71	0.1914	0.4653	+58.87%
Christmas Eve	71	0.4141	0.7550	+45.15%
Christmas Day	71	0.3687	0.6383	+42.24%
Labor Day	71	0.3894	0.5233	+25.59%
Thanksgiving	71	0.7671	0.7757	+1.12%
Veterans Day	71	0.4590	0.4227	−8.59%
Columbus Day	71	0.6427	0.5293	−21.41%

Figure 4.7 shows forecast comparisons on three significant-date windows.



(a) Wed 05 Sep 2012, 20:00.

(b) Tue 18 Dec 2012, 06:00.

(c) Mon 19 Nov 2012, 11:00.

Figure 4.7: Forecast comparison of PatchTST and E7 (Qwen2.5-7B, sample-wise context) on three significant-date windows from the Bike Sharing dataset. (a) covers 9/11 memorial week; (b) covers Christmas Eve week and (c) covers Thanksgiving week.

4.8 Case Study: Telecom KPI Forecasting

This section presents a case study of the best LLM configuration from Section 4.5 on the Telecom KPI dataset described in Subsection 3.3.2. The configuration is E7 (Qwen2.5-7B with sample-wise context, as defined in Table 3.4); we refer to it as E7 throughout this section. As stated in Subsection 3.5.1, only E7 is re-trained on the KPI data; the four research questions are not re-tested here. The same four baselines (Seasonal Naive, DLinear, LSTM, and PatchTST) are re-trained using the procedure described in Subsection 3.5.2, and evaluated on the same 3,258 test windows. The training and validation loss curves for E7 on the KPI data are provided in Section A.3.

Table 4.10 reports pooled test MSE and MAE across the 3,258 test windows. Pooled metrics use normalised values from the per-KPI scaler described in Subsection 3.3.2, placing all three KPIs on a comparable scale.

Table 4.10: Pooled test performance on the Telecom KPI dataset. Metrics are computed on per-KPI normalised values. The best result in each column is shown in bold.

Model	Test MSE	Test MAE
Seasonal Naive	1.1741	0.7338
DLinear	0.7129	0.5922
LSTM	0.7167	0.5945
PatchTST	**0.6762**	0.5772
E7	0.6785	**0.5686**

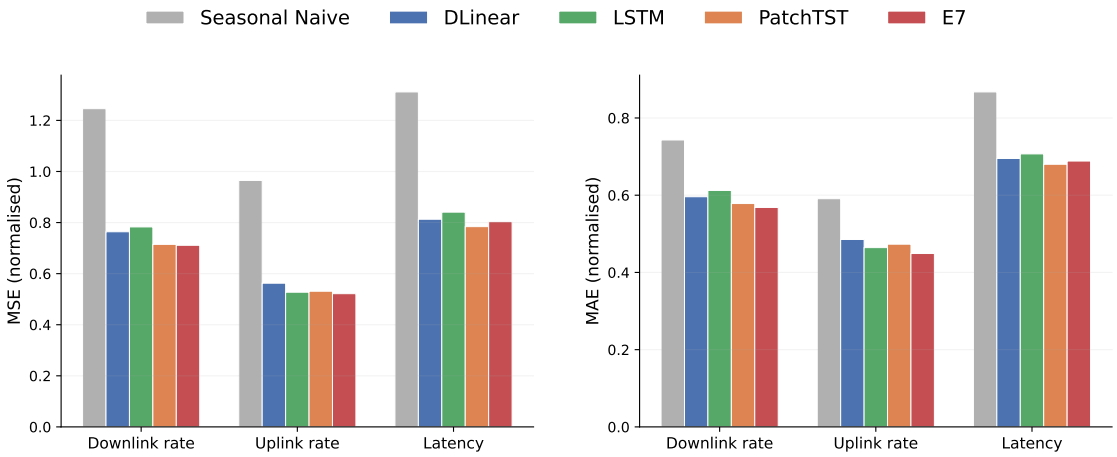
The two Transformer-based models, PatchTST and E7, perform comparably on pooled metrics: PatchTST achieves the lowest MSE (0.6762 vs. 0.6785) while E7 achieves the lowest MAE (0.5686 vs. 0.5772). Both achieve lower MSE and MAE than Seasonal Naive, DLinear, and LSTM.

Since the three KPIs differ in physical interpretation, scale, and shape, we also report per-KPI MSE and MAE on the same normalised scale, see Table 4.11.

Table 4.11: Per-KPI test performance on the Telecom KPI dataset. Each column uses the KPI-specific normaliser fit on the training set. The best result in each column is shown in bold.

	DL rate		UL rate		Latency	
Model	MSE	MAE	MSE	MAE	MSE	MAE
Seasonal Naive	1.2459	0.7430	0.9649	0.5909	1.3114	0.8674
DLinear	0.7639	0.5960	0.5626	0.4853	0.8130	0.6951
LSTM	0.7829	0.6123	0.5273	0.4642	0.8408	0.7069
PatchTST	0.7146	0.5784	0.5311	0.4732	**0.7840**	**0.6801**
E7	**0.7110**	**0.5683**	**0.5215**	**0.4490**	0.8036	0.6884

E7 has the lowest MSE and MAE on both data-rate KPIs; PatchTST has the lowest on latency. Figure 4.8 visualises the per-KPI breakdown.



(a) Test MSE.

(b) Test MAE.

Figure 4.8: Per-KPI test MSE (a) and MAE (b) for the five models across the three KPIs, on the per-KPI normalised scale.

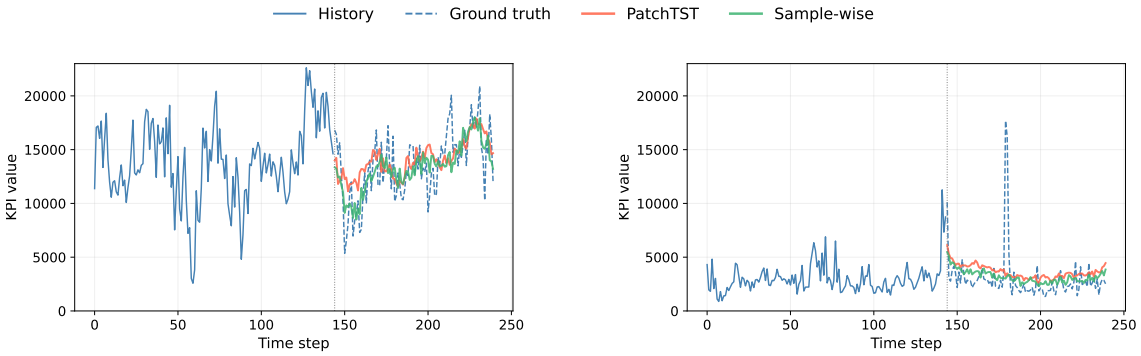
The same Diebold-Mariano procedure used in Section 4.6 is applied to the KPI test set. As specified in Subsection 3.6.2, the Newey-West lag is $\ell = 11$ on this dataset, since $H = 96$ and the sliding stride is $s = 8$. Table 4.12 reports each baseline against E7, per KPI subset and pooled.

Against the three weakest baselines (Seasonal Naive, DLinear; LSTM), E7 achieves notably better performance in almost every case, with only two non-significant cases (DLinear on latency, LSTM on uplink rate). Against PatchTST we have mixed results: E7 significantly outperforms PatchTST on uplink rate ($p = 0.003$) while PatchTST is significantly better on latency ($p = 0.020$), the two are statistically indistinguishable in regards to downlink rate. There is also no significant difference in performance across all KPIs when pooled.

Figure 4.9 shows one randomly selected test window for each of the two data-rate KPIs, and Figure 4.10 shows one for the latency KPI, in both cases comparing PatchTST and E7.

Table 4.12: Diebold-Mariano test results comparing each baseline against E7 on the Telecom KPI dataset. A positive DM statistic indicates that E7 has lower loss.

Baseline	DM stat.	p-value
DL rate		
Seasonal Naive	+7.526	$<0.001^{***}$
DLinear	+5.809	$<0.001^{***}$
LSTM	+8.994	$<0.001^{***}$
PatchTST	+0.475	0.635
UL rate		
Seasonal Naive	+11.321	$<0.001^{***}$
DLinear	+7.827	$<0.001^{***}$
LSTM	+1.547	0.122
PatchTST	+2.992	0.003^{**}
Latency		
Seasonal Naive	+16.639	$<0.001^{***}$
DLinear	+1.214	0.225
LSTM	+3.070	0.002^{**}
PatchTST	−2.335	0.020^{*}
Pooled		
Seasonal Naive	+17.087	$<0.001^{***}$
DLinear	+7.689	$<0.001^{***}$
LSTM	+7.292	$<0.001^{***}$
PatchTST	−0.525	0.600



(a) Downlink rate.

(b) Uplink rate.

Figure 4.9: Forecast comparison of PatchTST and E7 (Qwen2.5-7B, sample-wise context) on the two data-rate KPIs of the Telecom KPI dataset, each a single randomly selected test window. (a) shows downlink rate and (b) uplink rate.

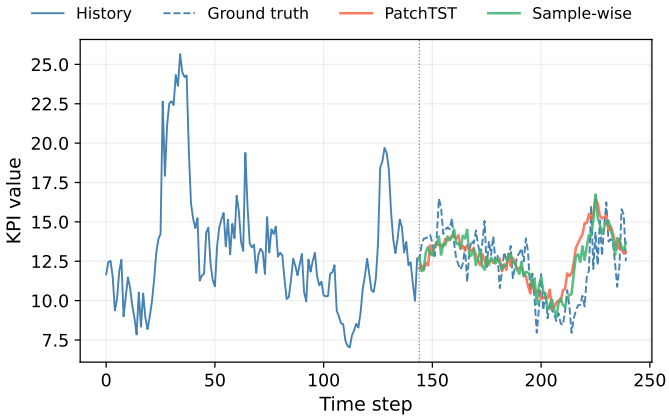


Figure 4.10: Forecast comparison of PatchTST and E7 (Qwen2.5-7B, sample-wise context) on a single randomly selected latency test window from the Telecom KPI dataset.

5

Discussion and Conclusion

5.1 Discussion

This section interprets the experimental results and relates them to the broader question of when textual context is useful in LLM-based time series forecasting. The discussion first summarises the main findings, then examines the interaction between backbone scale and textual context, the conditions under which the LLM-based model offers an advantage, its transferability to industrial KPI data, the study's main limitations, and its relation to prior work.

5.1.1 Summary of Findings

This thesis investigated the role of textual context in LLM-based time series forecasting through a systematic ablation study across four context configurations and two backbone scales. The results are summarised below, organised by research question.

Analysis of the static prompt effect (RQ1) indicates that a fixed dataset-level description does not significantly improve forecasting performance at either backbone scale (0.5B: $p = 0.868$; 7B: $p = 0.294$). The static context is identical across all samples: describing the dataset, the target variable, and the measurement frequency. The null result suggests that a fixed domain description contributes no information beyond what the model can already infer from the time-series patterns alone.

Analysis of the sample-wise context effect (RQ2) indicates that sample-wise textual context significantly improves forecasting performance at the 7B scale. At 0.5B, the improvement is directional but not statistically significant ($p = 0.299$). At 7B, sample-wise context reduces MSE by 18% compared to TS only ($p < 0.001$) and by 14% compared to static context ($p < 0.001$), transforming the 7B backbone from the worst-performing variant (E5, MSE = 0.4333) to the best (E7, MSE = 0.3562).

Analysis of semantic content versus token conditioning (RQ3) suggests that the semantic content of the sample-wise context matters, rather than the text functioning merely as a generic conditioning signal. At 7B, shuffling the context, breaking the correspondence between each time-series window and its calendar description, significantly degrades performance ($p < 0.001$), returning MSE to the level of static context (E8: 0.4235 vs E6: 0.4162). At 0.5B, the difference between shuffled and correct context is marginally non-significant ($p = 0.065$), consistent with the smaller backbone's limited ability to leverage textual input in either case. At the 7B scale,

these results rule out the possibility that any text prefix acts as a generic conditioning signal: the content of the prompt matters, not just its presence.

Analysis of the backbone scale interaction (RQ4) reveals a clear interaction between backbone scale and textual context. Without text, the 0.5B backbone significantly outperforms the 7B backbone ($p < 0.001$); with sample-wise context, the difference is no longer significant ($p = 0.272$), and the 7B point estimate is the lowest overall. The 0.5B backbone is relatively insensitive to the context configuration, with MSE ranging from 0.366 to 0.379 across the four variants. The 7B backbone, by contrast, spans the full range from worst to best (0.433 to 0.356) depending on the textual input. Scaling the backbone is only beneficial when paired with informative textual context, without it, the larger frozen representation space becomes a liability for the trainable reprogramming layer.

The overall comparison between the best LLM variant (E7) and PatchTST is not significant ($p = 0.115$), but partitioning the test set reveals a clear conditional advantage. On the 450 significant-date windows, E7 reduces MSE by 18.4% relative to PatchTST ($p = 0.043$); on the remaining 3,031 normal-date windows, the corresponding reduction is 3.7%, and the two models are statistically indistinguishable ($p = 0.384$). The per-date breakdown is heterogeneous: large MSE reductions on 9/11 Memorial, Christmas Eve, Christmas Day, and Labor Day, comparable performance on Thanksgiving, and small negative results on Veterans Day and Columbus Day.

5.1.2 How Does Scale Interact with Text?

The most striking result of this study is a crossover interaction between backbone scale and textual context. Without textual input, the 7B backbone is significantly worse than the 0.5B backbone ($p < 0.001$). Adding a static dataset description marginally narrows this gap: the 0.5B backbone is unaffected, and the 7B backbone improves slightly but not significantly ($p = 0.294$). Sample-wise calendar context reverses the relationship entirely. At 7B, sample-wise context reduces MSE by 18% over no context ($p < 0.001$) and by 14% over static context ($p < 0.001$), transforming the 7B backbone from the worst LLM variant to the best. At 0.5B, the same sample-wise context produces only a small directional improvement that is not statistically significant. The 0.5B backbone is in fact insensitive to the configuration of textual context in general: its MSE ranges from 0.366 to 0.379 across the four prompt variants, compared with 0.356 to 0.433 for the 7B backbone, which is roughly six times more responsive to what kind of prompt it receives. The 7B model also distinguishes correctly paired from shuffled sample-wise context significantly ($p < 0.001$), whereas the 0.5B model is marginally indistinguishable on the same comparison ($p = 0.065$).

Our interpretation of these results rests on three established facts about LLMs. First, pretrained LLMs encode substantial factual and structural knowledge in their parameters, accessible at inference time without further training [6]. Second, larger LLMs of the same family encode more of this knowledge than smaller ones [20]. Third, natural-language tokens enter the LLM through its own frozen tokeniser and embedding table (Subsection 3.1.4), so they evoke the same internal representations the model learned during pretraining, while reprogrammed numerical tokens do

not have this property. Together, these facts suggest a single mechanism. Without textual context, the trainable reprogramming layer must, on its own, construct token sequences that the frozen LLM processes usefully. This is plausibly harder at 7B than at 0.5B because the larger embedding space (3,584 dimensions versus 896) offers more directions to navigate from the training signal alone, which is consistent with the 7B-TS configuration underperforming the 0.5B-TS configuration. When a sample-wise prompt is prepended, the natural-language tokens evoke representations the LLM has organised during pretraining, and through causal attention [27] the subsequent patch tokens can attend to whatever the prompt activated. The asymmetry between the two scales then follows naturally. The 7B model has organised more nuanced pretrained knowledge than the 0.5B model, for example finer-grained distinctions such as "Thursday December 25" (Christmas Day) being distinct from other Thursdays in December. This is consistent with both the larger effect of correct sample-wise context at 7B and the larger penalty for shuffled context at 7B; the same activation mechanism appears available at 0.5B in principle, but the smaller backbone offers less pretrained knowledge for the prompt to activate.

5.1.3 The Situational Advantage: When LLMs Help

On aggregate metrics, the best LLM variant (E7) does not significantly outperform PatchTST [21] ($p = 0.115$). By only comparing headline MSE numbers, one might reasonably conclude that repurposing a 7-billion-parameter frozen language model offers no advantage over a purpose-built time-series Transformer with fewer than 500K parameters. However, the analysis of significant dates reveals that this conclusion is incomplete.

On the 3,031 normal-date windows, E7 reduces MSE by 3.7% relative to PatchTST, and the two models are statistically indistinguishable ($p = 0.384$). However, on the 450 significant-date windows, E7 reduces MSE by 18.4%, and the difference is significant ($p = 0.043$). The LLM's advantage is *situational*: it is concentrated on samples where demand deviates from patterns learnable from historical data alone.

These results suggest that the value proposition of LLM-based forecasting is not to replace specialised models for routine predictions, but to complement them for irregular events where purely pattern-based approaches struggle, and where the relevant kind of irregularity is encoded in the LLM's pretraining. Such domains include retail demand around major holidays, energy load during extreme weather, and transport demand during public events. In each, this selective advantage could be practically significant even if aggregate metrics show no overall improvement.

5.1.4 Generalisation to Industrial KPI Data

The Telecom KPI case study (Section 4.8) provides a modest external validity check rather than a replication of RQ1–RQ4: only the best configuration (E7) is re-trained, and the four research questions are not re-tested in this domain. The interpretation below therefore concerns whether the architecture and training recipe remain viable in a different setting, not whether the prompt-content and scale findings carry over.

On pooled metrics, E7 and PatchTST are statistically indistinguishable ($p = 0.60$), as on the Bike Sharing dataset ($p = 0.115$ in Section 4.6). Per KPI subset, E7 is significantly more accurate than PatchTST on uplink rate ($p = 0.003$), statistically indistinguishable on downlink rate, and significantly less accurate on latency ($p = 0.020$). The pooled result indicates that the frozen-LLM architecture remains competitive when applied to a domain with a higher sampling frequency (15 minutes vs. 1 hour), a longer lookback (144 vs. 96 time steps), and a longer forecast horizon (96 vs. 48 time steps) than the Bike Sharing setting.

The mixed per-KPI picture is harder to interpret. The Bike Sharing RQ3 result (Section 4.4) showed that mismatched or uninformative text can degrade performance rather than improve it, suggesting that the prompt content is more predictive of uplink rate than of downlink rate or latency in this dataset. Confirming this would require re-running the four-context ablation on KPI data, which lies outside the scope of this thesis.

5.1.5 Limitations

Several limitations constrain the generalisability of these findings. First, the controlled ablation is conducted on a single dataset and uses a single source of sample-wise text. Second, only a single LLM family is considered. Third, each experiment is trained once, so the differences between configurations are not separated from the variance across training runs.

The controlled ablation that answers RQ1–RQ4 is conducted on the Capital Bikeshare hourly series [52], using calendar metadata derived from each window's timestamps as the sample-wise text. Calendar metadata is one specific kind of informative text, chosen because it is easy to generate and plausibly aligned with the LLM's pretrained knowledge of dates and weekdays. Other text types, such as weather forecasts, event listings, free-form descriptions, or operational telemetry, differ in length, structure, and the extent to which they are represented in pretraining, and may interact with the reprogramming layer and backbone scale differently. The findings therefore establish that the mechanism works for one dataset and one kind of sample-wise text, but not that it generalises across the broader space. The Telecom KPI case study provides a limited check on the transferability of the architecture and training recipe, but does not address whether the prompt-content and scale findings replicate.

Only Qwen2.5 is used as the frozen backbone, at two scales (0.5B and 7B). The observed scale interaction, namely that larger backbones benefit more from textual context, may therefore not generalise across LLM families. Models with different pretraining corpora, tokenisation schemes, or architectural choices, such as mixture-of-experts architectures, could respond differently to prompt conditioning. The two scales evaluated also leave open whether the pattern extends to intermediate model sizes, such as 1.5B or 3B, or instead saturates beyond 7B.

Finally, each experiment in this thesis is run once, with a single training seed and a single train/validation/test split. The Diebold-Mariano tests account for variance across test windows but not for variance across training runs. Several of the reported

differences between configurations are small in absolute terms (test MSE values cluster in a 0.06 range across the eight LLM variants), so a fraction of that range could plausibly be attributable to seed-level variation. The large effects reported in this thesis — the 18% MSE reduction from sample-wise context at 7B, the shuffled-vs-correct gap at 7B, and the 0.5B-vs-7B crossover — are unlikely to be artefacts of a single seed given their magnitude and statistical strength. Smaller differences, particularly the non-significant 0.5B comparisons and the marginal significant-date result, should be interpreted with this in mind.

5.1.6 Relation to Prior Work

The relationship between our findings and prior work depends strongly on the backbone scale. At the smaller scales used in earlier studies, our results are largely consistent with those reported. At the 7B scale, our results diverge from several prior conclusions, in ways the remainder of this subsection examines in detail.

Niu et al. [16] showed that replacing meaningful prompts with random text does not degrade forecasting accuracy at small backbone scale (GPT-2 and truncated LLaMA), and argued that prompts function as implicit adapters rather than conveying semantic information. At small scale and with static prompts, our results are consistent with theirs: replacing the no-prompt configuration with a static dataset description produces no significant improvement at either 0.5B or 7B (E1 vs E2, E5 vs E6), and our shuffled-context comparison at 0.5B is marginally non-significant ($p = 0.065$). At 7B, however, our sample-wise experiments contradict their hypothesis. Correctly paired calendar context significantly outperforms shuffled context (E8 vs E7, $p < 0.001$), holding the number of tokens and trainable parameters fixed. The content of the text is therefore doing real work when the backbone is large enough to use it, which is the opposite of what their adapter framing predicts.

The architectural differences between our setup and theirs help explain why their hypothesis is likely incomplete rather than simply wrong. Niu et al.'s adapter framing rests on the observation that prompts coincide with the introduction of learnable parameters: in their setup, replacing one prompt with another changes both the input the LLM sees and the parameters available to adapt to it, so the two effects are confounded. In our architecture this confound is removed. Prompt token embeddings are looked up from the LLM's frozen vocabulary embedding table and used as-is (Subsection 3.1.4); they are not trainable. The trainable components, vocabulary mapping, reprogramming cross-attention, patch embedding, and output projection, are identical across the four prompt configurations, varying only in what they receive as input. When the sample-wise variant at 7B significantly outperforms the no-prompt, static-prompt and shuffled-prompt variants, there is no additional learnable adapter to credit. The only thing that changes between configurations is the textual information the frozen LLM receives. The adapter framing predicts no difference; we observe a large one. The simplest reading is that the adapter mechanism is real for the setups Niu et al. tested, but is not the full story: at sufficient scale, the content of the textual signal contributes independently of any capacity it might carry.

Tan et al. [17] showed that replacing pretrained LLM backbones with randomly

initialised Transformers yields comparable forecasting performance, and concluded that the pretrained representations are not essential for time series forecasting. Our TS-only results at 7B are compatible with this: without textual context, the 7B backbone performs *worse* than the 0.5B backbone, suggesting that the additional pretrained capacity provides no benefit when the model receives only numerical input. However, Tan et al. also recommended that LLM-based methods focus on "multimodal applications that require textual reasoning." Our results are in line with this recommendation: with sample-wise calendar context, the 7B backbone becomes the best-performing variant, and its advantage is concentrated on holidays and special events where semantic reasoning about dates is most informative. The pretrained representations *are* useful, but only when the prompt elicits the relevant knowledge.

Jin et al. [13] report that removing the Prompt-as-Prefix degrades average MSE by over 8%, and their ablations attribute this gain to two static-prompt components: 1) clear task instructions and 2) a dataset description, netting MSE reductions of 7.7% and 9.6% respectively. We do not observe a comparable effect in our static-context experiments. At 0.5B, adding a static dataset description marginally worsens MSE by -0.2% (E1 vs E2), at 7B, it produces a directional improvement of 3.9% (E5 vs E6) but the difference is not statistically significant ($p = 0.294$). One possible explanation is the difference in backbone architectures: Jin et al. used LLaMA-7B, whereas our experiments use Qwen2.5. Another is the domain difference: all their prompt ablations were performed solely on the Electricity Transformer Dataset (ETDataset) [38].

Zhou et al. [12] achieved competitive forecasting performance with a frozen GPT-2 backbone and no textual input, training only the input embedding, positional embedding, and layer normalisation. Our 0.5B results are broadly consistent: at a small backbone scale, the frozen Transformer provides useful representations for time series forecasting even without text. The divergence arises at a larger scale, where our results show that the frozen backbone's additional capacity becomes a liability without textual input. GPT4TS did not test larger backbones, so whether the same pattern would emerge with their architecture remains an open question.

5.2 Future Work

The limitations identified in Subsection 5.1.5 point to four directions for future research.

The most immediate extension is to replicate the four-RQ ablation on additional datasets. Re-running the full eight-experiment matrix on the Telecom KPI dataset is the most direct next step, as the data are already in place, to test whether the prompt-content and scale findings observed on the Bike Sharing dataset carry over to a higher-frequency industrial setting. Additionally, evaluating the model on public benchmarks such as the ETT datasets used by Niu et al. [16] would enable a direct comparison with their findings under matched conditions.

A complementary direction is to vary the kind of sample-wise text. For Bike Sharing,

future work could augment or replace calendar metadata with weather forecasts, event listings, or natural-language summaries of recent demand. For the KPI setting, the sample-wise text could be augmented with operational signals known to drive KPI behaviour, such as alarm logs, operator notes, configuration changes, and maintenance schedules. Combining dataset replication with text-type variation would help isolate which properties of the sample-wise text drive the scale-context interaction.

A third direction is to broaden the analysis across LLM families and model scales. Evaluating intermediate sizes, such as 1.5B or 3B, would clarify whether the scale-context interaction develops gradually or instead reflects a threshold effect. Extending the comparison to other model families, such as LLaMA, Gemma, or Phi, would show whether the findings are specific to Qwen2.5's pretraining corpus and architecture or generalise across frozen LLM backbones.

Finally, multi-seed replication would address the single-seed limitation. Repeating the eight-experiment matrix with several seeds would allow the within-configuration variance to be estimated and sharpen confidence intervals on the smaller differences reported here, particularly the comparisons that sit closer to the significance threshold.

A separate practical concern is computational efficiency. The 7B model with sample-wise context requires 32 minutes per epoch and 71 GB of VRAM, roughly 50× the cost of the 0.5B time-series-only variant and still far more expensive than PatchTST. Since the aggregate improvement over PatchTST is not statistically significant, practical adoption may depend on either demonstrating clear value in high-stakes multimodal forecasting or on reducing computational overhead, for example, through backbone distillation or selective prompting that applies textual context only when it is expected to be informative.

5.3 Conclusion

This thesis investigated whether textual context improves LLM-based time series forecasting and, if so, whether the improvement stems from the semantic content of the text or from the presence of additional tokens. Through a systematic ablation across four context configurations and two backbone scales on hourly bike rental demand, together with a case study on an industrial telecom KPI dataset, six conclusions emerge.

First, a fixed dataset-level description does not improve forecasting performance. Static context provides no sample-wise information, and the frozen LLM gains nothing from a textual restatement of what the reprogramming layer can already learn from the data.

Second, sample-wise calendar context significantly improves performance at the 7B scale, reducing MSE by 18% and transforming the 7B backbone from the worst LLM variant to the best. The 0.5B backbone shows a directional but non-significant benefit from the same context, suggesting that the effect is scale-dependent.

Third, the semantic content of the context matters. Shuffling the sample-wise

descriptions, so that each time-series window receives calendar information from a different window, destroys the benefit at the 7B scale, reverting performance to static-context levels. This rules out the hypothesis that textual tokens act as a generic conditioning signal and confirms that the model extracts and acts on the specific calendar information in the prompt.

Fourth, backbone scale and textual context interact: a larger frozen LLM is only beneficial when paired with informative text. Without context, the additional capacity of the 7B backbone is a liability; with context, it becomes an asset. The practical implication is that scaling the backbone without also providing meaningful textual input is counterproductive.

Fifth, the LLM's advantage over PatchTST is situational and selective. On normal days, the two models perform comparably. On significant dates, E7 reduces the MSE by 18.4% relative to PatchTST ($p = 0.043$), but the per-date breakdown shows that this advantage is concentrated on well-known dates associated with disruptions to daily routines. Less locally relevant dates show no benefit or a slight cost.

Sixth, the configuration that performs best in the controlled ablation transfers to an industrial telecom KPI dataset, where it remains competitive with PatchTST overall and outperforms the simpler baselines. The per-KPI breakdown is mixed: E7 is significantly more accurate on uplink rate, directionally more accurate but not significantly so on downlink rate, and significantly less accurate on latency.

Bibliography

[1] Q. Wen et al., "Transformers in time series: A survey," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, E. Elkind, Ed., Survey Track, International Joint Conferences on Artificial Intelligence Organization, Aug. 2023, pp. 6778–6786. DOI: 10.24963/ijcai.2023/759. [Online]. Available: https://doi.org/10.24963/ijcai.2023/759.

[2] Z. Liu, Z. Zhu, J. Gao, and C. Xu, "Forecast methods for time series data: A survey," *IEEE Access*, vol. 9, pp. 91 896–91 912, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3091162. [Online]. Available: https://ieeexplore.ieee.org/document/9461796/.

[3] S. Chattopadhyay, P. Paliwal, S. S. Narasimhan, S. Agarwal, and S. P. Chinchali, *Context matters: Leveraging contextual features for time series forecasting*, 2025. arXiv: 2410.12672 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2410.12672.

[4] Y. Jiang et al., "Empowering time series analysis with large language models: A survey," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, K. Larson, Ed., Survey Track, International Joint Conferences on Artificial Intelligence Organization, Aug. 2024, pp. 8095–8103. DOI: 10.24963/ijcai.2024/895. [Online]. Available: https://doi.org/10.24963/ijcai.2024/895.

[5] H. Touvron et al., *Llama 2: Open foundation and fine-tuned chat models*, en, 2023. [Online]. Available: https://arxiv.org/abs/2307.09288v2.

[6] F. Petroni et al., "Language models as knowledge bases?" In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2463–2473. DOI: 10.18653/v1/D19-1250. [Online]. Available: https://aclanthology.org/D19-1250/.

[7] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 9459–9474. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[8] N. Gruver, M. Finzi, S. Qiu, and A. G. Wilson, "Large language models are zero-shot time series forecasters," in *Advances in Neural Information Processing*

Systems, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 19 622–19 635. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/3eb7ca52e8207697361b2c0fb3926511-Paper-Conference.pdf.

[9] S. Wu et al., "Bloomberggpt: A large language model for finance," no. arXiv:2303.17564, Dec. 2023, arXiv:2303.17564 [cs]. DOI: 10.48550/arXiv.2303.17564. [Online]. Available: http://arxiv.org/abs/2303.17564.

[10] H. Xue and F. D. Salim, "Promptcast: A new prompt-based learning paradigm for time series forecasting," *IEEE Trans. on Knowl. and Data Eng.*, vol. 36, no. 11, pp. 6851–6864, Nov. 2024, ISSN: 1041-4347. DOI: 10.1109/TKDE.2023.3342137. [Online]. Available: https://doi.org/10.1109/TKDE.2023.3342137.

[11] M. Cheng, J. Wang, D. Wang, X. Tao, Q. Liu, and E. Chen, "Can slow-thinking llms reason over time? empirical studies in time series forecasting," in *Proceedings of the Nineteenth ACM International Conference on Web Search and Data Mining*, 2026. DOI: 10.1145/3773966.3777931.

[12] T. Zhou, P. Niu, X. Wang, L. Sun, and R. Jin, "One fits all: Power general time series analysis by pretrained lm," *Advances in Neural Information Processing Systems*, vol. 36, pp. 43 322–43 355, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/86c17de05579cde52025f9984e6e2ebb-Abstract-Conference.html.

[13] M. Jin et al., "Time-LLM: Time series forecasting by reprogramming large language models," in *International Conference on Learning Representations*, 2024.

[14] D. Cao et al., "TEMPO: Prompt-based generative pre-trained transformer for time series forecasting," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=YH5w12OUuU.

[15] Y. Jiang et al., "Multi-modal time series analysis: A tutorial and survey," en, in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, Toronto ON Canada: ACM, Aug. 2025, pp. 6043–6053, ISBN: 9798400714542. DOI: 10.1145/3711896.3736567. [Online]. Available: https://dl.acm.org/doi/10.1145/3711896.3736567.

[16] P. Niu, T. Yin, T. Zhou, L. Sun, and R. Jin, "Understanding the role of textual prompts in LLM for time series forecasting: An adapter view," *arXiv preprint arXiv:2311.14782*, 2024.

[17] M. Tan, M. A. Merrill, V. Gupta, T. Althoff, and T. Hartvigsen, "Are language models actually useful for time series forecasting?" In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[18] H. Fanaee-T, *Bike sharing*, 2013. DOI: 10.24432/C5W894. [Online]. Available: https://archive.ics.uci.edu/dataset/275.

[19] F. Jia, K. Wang, Y. Zheng, D. Cao, and Y. Liu, "Gpt4mts: Prompt-based large language model for multimodal time-series forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 21, pp. 23 343–23 351, 2024. DOI: 10.1609/aaai.v38i21.30383.

[20] J. Hoffmann et al., "Training compute-optimal large language models," in *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[21] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers," in *International Conference on Learning Representations*, 2023.

[22] A. F. Ansari et al., "Chronos: Learning the language of time series," *Transactions on Machine Learning Research*, 2024, Expert Certification, ISSN: 2835-8856. [Online]. Available: https://openreview.net/forum?id=gerNCVqqtR.

[23] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 11 121–11 128. DOI: 10.1609/aaai.v37i9.26317.

[24] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, Mar. 2020, ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167278919305974.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[26] S. Siami-Namini, N. Tavakoli, and A. Siami Namin, "A comparison of arima and lstm in forecasting time series," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2018, pp. 1394–1401. DOI: 10.1109/ICMLA.2018.00227. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8614252.

[27] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[28] X. Kong et al., "Deep learning for time series forecasting: A survey," en, *International Journal of Machine Learning and Cybernetics*, vol. 16, no. 7, pp. 5079–5112, Aug. 2025, ISSN: 1868-808X. DOI: 10.1007/s13042-025-02560-w. [Online]. Available: https://doi.org/10.1007/s13042-025-02560-w.

[29] H. Zhou et al., *Informer: Beyond efficient transformer for long sequence time-series forecasting*, 2021. arXiv: 2012.07436 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2012.07436.

[30] T. Brown et al., "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[31] S. J. Mielke et al., "Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp," no. arXiv:2112.10508, Dec. 2021, arXiv:2112.10508 [cs]. DOI: 10.48550/arXiv.2112.10508. [Online]. Available: http://arxiv.org/abs/2112.10508.

[32] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, K. Erk and N. A. Smith, Eds., Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. [Online]. Available: https://aclanthology.org/P16-1162/.

[33] B. Minixhofer, E. M. Ponti, and I. Vulić, "Zero-shot tokenizer transfer," *Advances in Neural Information Processing Systems*, vol. 37, pp. 46 791–46 818, 2024.

[34] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9.

[35] S. Chen, Y. Hou, Y. Cui, W. Che, T. Liu, and X. Yu, "Recall and learn: Fine-tuning deep pretrained language models with less forgetting," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds., Online: Association for Computational Linguistics, Nov. 2020, pp. 7870–7881. DOI: 10.18653/v1/2020.emnlp-main.634. [Online]. Available: https://aclanthology.org/2020.emnlp-main.634/.

[36] D. Spathis and F. Kawsar, "The first step is the hardest: Pitfalls of representing and tokenizing temporal data for large language models," *Journal of the American Medical Informatics Association*, vol. 31, no. 9, pp. 2151–2158, Sep. 2024, ISSN: 1527-974X. DOI: 10.1093/jamia/ocae090. eprint: https://academic.oup.com/jamia/article-pdf/31/9/2151/58868315/ocae090.pdf. [Online]. Available: https://doi.org/10.1093/jamia/ocae090.

[37] H. Liu, Z. Zhao, J. Wang, H. Kamarthi, and B. A. Prakash, "LSTPrompt: Large language models as zero-shot time series forecasters by long-short-term prompting," in *Findings of the Association for Computational Linguistics: ACL 2024*, Association for Computational Linguistics, 2024, pp. 7832–7840. DOI: 10.18653/v1/2024.findings-acl.466.

[38] H. Zhou et al., "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 11 106–11 115. DOI: 10.1609/aaai.v35i12.17325. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17325.

[39] K. Rasul et al., "Lag-llama: Towards foundation models for probabilistic time series forecasting," *arXiv preprint arXiv:2310.08278*, 2023.

[40] F. X. Diebold and R. S. Mariano, "Comparing predictive accuracy," *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 253–263, 1995. DOI: 10.1080/07350015.1995.10524599.

[41] W. K. Newey and K. D. West, "A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix," *Econometrica*, vol. 55, no. 3, pp. 703–708, 1987. DOI: 10.2307/1913610.

[42] Qwen Team, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2025.

[43] T. Kim, J. Kim, Y. Tae, C. Park, J.-H. Choi, and J. Choo, "Reversible instance normalization for accurate time-series forecasting against distribution shift," in *International Conference on Learning Representations*, 2022.

[44] Q. Team, *Qwen2.5: A party of foundation models*, Sep. 2024. [Online]. Available: https://qwenlm.github.io/blog/qwen2.5/.

[45] H. Ma, K. Yang, and M.-O. Pun, "Cellular traffic prediction via deep state space models with attention mechanism," *Computer Communications*, vol. 197,

pp. 276–283, 2023, ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2022.10.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014036642200411X.

[46] S. Palat and P. Godin, "Network architecture," in *LTE – The UMTS Long Term Evolution*. John Wiley Sons, Ltd, 2009, ch. 2, pp. 21–50, ISBN: 9780470742891. DOI: https://doi.org/10.1002/9780470742891.ch2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470742891.ch2. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470742891.ch2.

[47] T. O. Olwal, K. Djouani, and A. M. Kurien, "A survey of resource management toward 5g radio access networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 1656–1686, 2016. DOI: 10.1109/COMST.2016.2550765.

[48] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, SPIE, vol. 11006, 2019, pp. 369–386. DOI: 10.1117/12.2520589.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[50] C. E. Bonferroni, "Teoria statistica delle classi e calcolo delle probabilità," *Pubblicazioni del R. Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, vol. 8, pp. 3–62, 1936.

[51] U.S. Office of Personnel Management, *2012 federal holidays*, Accessed: 2026-04-05, 2012. [Online]. Available: https://www.opm.gov/policy-data-oversight/pay-leave/federal-holidays/%5C#url=2012.

[52] H. Fanaee-T and J. Gama, "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, vol. 2, no. 2–3, pp. 113–127, 2014. DOI: 10.1007/s13748-013-0040-3.

[53] T. Guan et al., "Timeomni-1: Incentivizing complex reasoning with time series in large language models," in *The Fourteenth International Conference on Learning Representations*, 2026. [Online]. Available: https://openreview.net/forum?id=kOIclg7muL.

[54] Z. Liu, P. Han, H. Yu, H. Li, and J. You, "Time-r1: Towards comprehensive temporal reasoning in llms," no. arXiv:2505.13508, 2025, arXiv:2505.13508 [cs]. DOI: 10.48550/arXiv.2505.13508. [Online]. Available: http://arxiv.org/abs/2505.13508.

[55] P. Langer et al., "Opentslm: Time-series language models for reasoning over multivariate medical text- and time-series data," no. arXiv:2510.02410, Feb. 2026, arXiv:2510.02410 [cs]. DOI: 10.48550/arXiv.2510.02410. [Online]. Available: http://arxiv.org/abs/2510.02410.

[56] E. J. Hu et al., *Lora: Low-rank adaptation of large language models*, 2021. arXiv: 2106.09685 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2106.09685.

[57] Y. Kong et al., "Time-mqa: Time series multi-task question answering with context enhancement," in *Proceedings of the 63nd Annual Meeting of the Association for Computational Linguistics*, 2025.

A

Appendix 1

A.1 Discussion of a Reasoning-Based Architecture That Was Not Performant

As part of the broader model exploration in this thesis, we also investigated an architectural direction inspired by recent work that combines time-series forecasting with textual prompting and explicit reasoning. This line of work is motivated by the idea that large language models may contribute more than generic sequence modelling capacity: they may also provide a mechanism for incorporating structured instructions, contextual descriptions, and intermediate reasoning into the forecasting process. Several recent papers suggest that this is a promising direction, including work such as *TimeOmni*, *Time-R1*, *Time-R1 Slow Thinking*, and *TimeReasoner* [11], [53], [54]. Related developments have also appeared in time series reasoning for healthcare, especially in classification settings where temporal information is combined with clinical or textual context [55].

From a thesis perspective, this family of methods was interesting for two reasons. First, it represents an attempt to move beyond a richer integration of textual context and time-series forecasting. Second, it provided an opportunity to explain the forecast and motivate it. In particular, we chose to focus on an implementation inspired primarily by *Time-R1 Slow Thinking* [54], since the reported results were promising and the described setup appeared more flexible with respect to prediction length, including a reported setting of 96/96. This was appealing in our experimental setting, where such a context-horizon combination was practical and easy to compare against the rest of our forecasting pipeline.

The implementation objective was not to propose a new reasoning-based architecture from scratch, but rather to investigate whether we could implement it and have it work in our setting, and to evaluate it across different textual contexts. In that sense, this appendix should be read as a discussion of a negative but informative result. The architecture was motivated by the literature, but the reproduction effort revealed that the gap between a promising paper result and a stable implementation can be substantial. This became especially clear once the training setup moved beyond ordinary supervised forecasting and instead depended on several interconnected components, including supervised fine-tuning, reinforcement learning, prompt design, and parameter-efficient adaptation.

A central difficulty was that the supervised fine-tuning stage was not described in enough detail to allow an exact reconstruction. In practice, this meant that important design decisions had to be inferred rather than reproduced directly. These decisions included how to form the training examples, how to align the chain of thought with ground truth, and how to represent the desired outputs. Although such omissions are understandable, they become highly consequential in architectures where the final behaviour may depend strongly on formatting and training-stage interaction.

For the reinforcement learning stage, we used GRPO, as it has been reported as a competitive method, stating that their method "slightly outperforms GRPO" [54]. In addition, we used LoRA to make the adaptation computationally feasible, rather than fully fine-tuning the entire language model backbone [56]. From an engineering standpoint, this was a reasonable and practical choice. From a scientific standpoint, however, it introduced another possible source of deviation from the reported setup. Even if each design choice appeared justified, the accumulated effect of small mismatches may have contributed to the failure to reproduce the published behaviour.

In our experiments, we did not obtain results comparable to those described in the reasoning-based forecasting literature. Instead, the model repeatedly converged to degenerate output behaviour. Most notably, we encountered failure modes that resemble what *TimeReasoner* refers to as *copy-paste repeat* and *constant collapse* [11]. In practical terms, this meant that the model often produced forecasts that were either direct continuations of the last observed value or nearly constant trajectories with very little variation across the prediction horizon. Rather than learning meaningful future dynamics, the model appeared to settle on trivial solutions that were locally stable from the optimiser's perspective but not useful as forecasts.

As an additional diagnostic, we evaluated the base model's behaviour before applying the reasoning-based optimisation procedure. This revealed a telling pattern: approximately 75% of the generated forecasts were classified as constant predictions. Moreover, these constant forecasts achieved better performance than the non-constant forecasts. This suggests that the model was already biased toward a trivial yet locally effective forecasting strategy, in which repeating or flattening the output could reduce the loss without capturing meaningful temporal dynamics.

We tested this issue across multiple settings and repeated runs. In particular, we experimented with different reward formulations intended to discourage trivial forecasts. Some runs included explicit penalties for constant prediction. Others attempted to reward matching the volatility of the ground truth, under the intuition that a model producing overly flat forecasts should be penalised if the true future sequence is more dynamic. We also tested reward-scaling variants to make these constraints more influential during optimisation. However, these modifications did not materially improve the model behaviour.

One interpretation of these results is that the model learned reward-compatible shortcuts that were easier to optimise than genuinely informative forecasts. This fits a broader pattern in reinforcement learning, in which weak or imperfect objectives can yield locally attractive yet ultimately undesirable solutions. In our case, even

when the reward function penalised constant outputs, the model did not shift toward robust forecasting, suggesting that the problem was not confined to reward design. Instead, it likely originated earlier in the pipeline, for example, in the supervised fine-tuning stage, the response format, or in how forecasting was framed as a reasoning task rather than a direct numerical prediction problem.

This also helps explain why the model often converged to trivial or low-variance outputs. Repeating the last observed value, or producing nearly constant continuations, is a comparatively easy strategy when the model is unsure how to generate precise numeric sequences in token form. The collapse patterns we observed, therefore, highlight a mismatch between stable token generation and accurate forecasting: a forecast that is easy for the language model to express is not automatically useful from a time-series perspective.

Taken together, these observations suggest that multiple failure modes interacted. Reward shaping alone could not resolve the issue, indicating deeper problems in the training, while the token-based representation of numerical forecasts likely made the task substantially harder. For this thesis, the key conclusion is therefore methodological. Exploring this architecture clarified the practical limits of reproducibility in this emerging model class and provided a concrete reason not to base the main experimental pipeline on it. Although the idea was conceptually attractive, our implementation repeatedly exhibited unstable or collapsed behaviour and never reached the reliability required for a central model in the thesis. This does not imply that the architecture direction lacks merit; it indicates that, within the time and resource constraints of this project, it could not be made sufficiently robust.

In summary, we explored a reasoning-based forecasting architecture motivated by recent work such as *TimeOmni*, *Time-R1*, *Time-R1 Slow Thinking*, *TimeReasoner*, healthcare-oriented reasoning models, and prompt-based forecasting approaches such as *PromptCast* [11], [53], [54], [55]. Our implementation used a reconstructed supervised fine-tuning stage, GRPO in the reinforcement learning stage, and LoRA for parameter-efficient adaptation [54], [56]. Despite repeated experimentation, the model consistently showed failure modes characterised by copy-paste repetition and constant collapse, where forecasts became trivial continuations of the last observed value [11]. Reward engineering aimed at discouraging such behaviour did not resolve the issue. Consequently, this architecture has been put on hold for now, although the investigation was still valuable, as it exposed practical challenges in reproducing and stabilising reasoning-based time-series models.

A.1.1 Constant Collapse Analysis on Base LLM

Before applying RL, we ran a small set of experiments to better understand when and why the model produces constant forecasts. The goal was not to train the model, but to compare different system-prompt setups and input representations, and assess their effects on forecasting quality and the rate of constant predictions.

We swept over several configurations. First, we compared the base Qwen2.5 model with its instruction-tuned variant. Second, we tested three system prompts: no

prompt, a table-style prompt, and a list-style prompt. Third, we varied how the lookback window was represented in the input, using either a table-style representation or a list-style representation.

For each model, system prompt, and lookback representation, we evaluated the generated forecasts on a subset of the ETT dataset [38] comprising 500 samples. Since this experiment only changed the system prompt and input formatting, no training was performed. We measured standard forecasting errors using MSE and MAE, and additionally computed the percentage of predictions that collapsed to a constant value. Table A.1 summarises the best-performing configuration for each model and lookback format, where the best system prompt is selected based on the lowest constant prediction rate.

A clear observation from this sweep was that constant predictions often achieved lower MSE and MAE than non-constant predictions. This suggests that the LLM may not always generate meaningful forecasts, even when the resulting MSE or MAE appears competitive. In particular, a low forecasting error can sometimes come from collapsed constant predictions rather than from the model capturing the underlying temporal dynamics.

We did not observe a significant performance difference between the base and instruction-tuned Qwen2.5 models. The choice of system prompt and lookback representation appeared to matter more than whether the model was instruction-tuned.

Table A.1: Constant collapse analysis before RL on 500 ETT test samples.

Model	System prompt	Lookback format	Constant (%)	MSE	MAE
Qwen2.5-Instruct	Table	Table	74	1.86	1.09
Qwen2.5-Instruct	None	List	83	1.60	1.06
Qwen2.5	None	Table	83	1.71	1.06
Qwen2.5	List	List	72	2.14	1.13
Qwen2.5-Instruct + TSQA FT	List	List	63	1.94	1.13

Finally, we also fine-tuned the 7B instruction-tuned model on the TSQA dataset [57]. This dataset uses a list-style format, and the fine-tuned model achieved the best overall performance on non-constant prediction. Importantly, it also had the lowest constant-prediction rate, suggesting that supervised fine-tuning on a format aligned with the target prompting setup may help reduce constant collapse while improving forecasting performance.

A.2 Prompts

A.2.1 Static Context

> **Dataset description:** This dataset records hourly bike rental counts from the Capital Bikeshare system in Washington, D.C., USA, from 2011 to 2012. Bike rental demand follows clear daily and weekly patterns,

with different usage behaviour on weekdays and weekends. The series also varies across seasons, with stronger demand in warmer periods and weaker demand in colder periods.

Task description: Forecast the next 48 hourly bike rental counts given the previous 96 hours of rental data.

A.2.2 Sample-wise Context

Dataset description: This dataset records hourly bike rental counts from the Capital Bikeshare system in Washington, D.C., USA, from 2011 to 2012. Bike rental demand follows clear daily and weekly patterns, with different usage behavior on weekdays and weekends. The series also varies across seasons, with stronger demand in warmer periods and weaker demand in colder periods.

Task description: Forecast the next 48 hourly bike rental counts given the previous 96 hours of rental data.

Instance description: The available time series history starts at Friday at 7AM on August 3 and ends at Tuesday at 6AM on August 7.

The history window is partitioned as follows:
Friday: 17 hours
Saturday: 24 hours
Sunday: 24 hours
Monday: 24 hours
Tuesday: 7 hours

The future values to predict start at Tuesday at 7AM on August 7 and end at Thursday at 6AM on August 9.

The forecast window is partitioned as follows:
Tuesday: 17 hours
Wednesday: 24 hours
Thursday: 7 hours

A.3 Telecom KPI Training Curves

Figure A.1 shows the training and validation MSE of E7 (Qwen2.5-7B with sample-wise context) over the 50-epoch training budget on the Telecom KPI dataset. The first epoch is omitted for scale. Both curves decrease smoothly and converge without signs of overfitting.

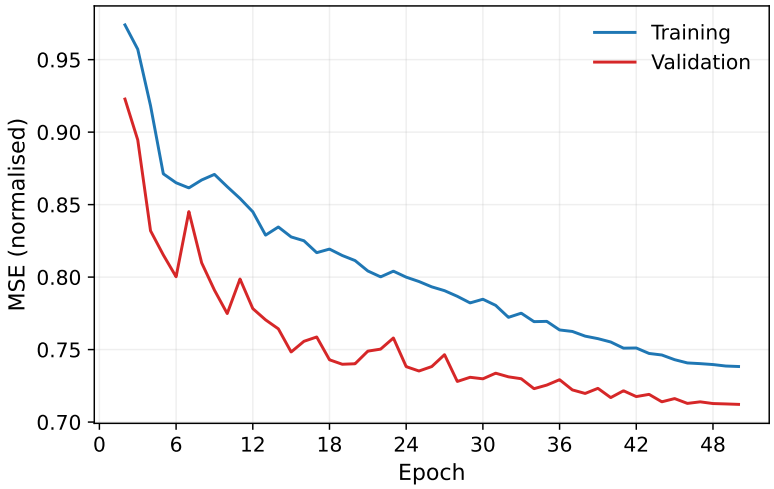


Figure A.1: Training and validation MSE for E7 (Qwen2.5-7B with sample-wise context) over 50 epochs on the Telecom KPI dataset. The first epoch is omitted for scale.