Time-LlaMA: Adapting Large Language Models for Time Series Modeling via Dynamic Low-rank Adaptation

Anonymous ACL submission

Abstract

Time series modeling holds significant importance in many industrial applications and has been extensively studied. A series of recent studies have demonstrated that large language models (LLMs) possess robust pattern recognition and semantic understanding capabilities over time series data. However, the current literature have yet striked a high-quality balance between (a) effectively aligning the time series and natural language modalities and (b) keeping the inference efficiency for industrial deployment. To address the above issues, we now propose the Time-LlaMA framework. Time-LlaMA first converts the time series input into token embeddings through a linear tokenization mechanism. Second, the time series token embeddings are aligned with the text prompts. Third, to further adapt the LLM backbone for time series modeling, we have developed a dynamic low-rank adaptation technique (DynaLoRA). DynaLoRA dynamically chooses the most suitable LoRA modules at each layer of the Transformer backbone for each time series input, enhancing the model's predictive capabilities. Our experimental results on an extensive collection of challenging open and proprietary time series tasks confirm that our proposed method achieves the state-of-the-art (SOTA) performance and have potentials for wide industrial usages.[1]

1 Introduction

Time series forecasting (TSP) represents a crucial modeling endeavor (Jin et al., 2023b), spanning a wide array of practical applications such as climate modeling, inventory management, and energy demand prediction. Typically, each forecasting task demands specialized domain expertise and bespoke model architectures. This requirement has precluded the development of a robust foundational model (FM) capable of few-shot or zero-shot learning, akin to GPT-3 (Brown et al., 2020), GPT-4 (OpenAI, 2023), and Claude-3[2], within the time series domain. Despite the fact that time series modeling has yet to witness similar groundbreaking advancements, the remarkable capabilities of large language models (LLMs) have fueled interest in their application to time series forecasting tasks (Zhou et al., 2023).

Despite the advancements in the literature on Large Language Model (LLM)-based Time Series (TS) modeling (Zhou et al., 2023; Jin et al., 2023a), several limitations remain, hindering their industrial usages. Firstly, the successful integration of time series data with natural language in LLM-based TS modeling depends heavily on the appropriate alignment of their respective modalities. Current approaches primarily rely on text prompts and cross-attention mechanisms, which do not effectively leverage the vocabulary. Secondly, recent studies adopt a methodology similar to PatchTST (Nie et al., 2022), transforming a univariate time series into a sequence of patches that are then treated as tokens input into Transformer blocks. This approach necessitates converting multivariate Time Series Prediction (TSP) tasks into multiple univariate TSP subtasks, leading to increased inference latency. Lastly, the current works maintains the LLM backbone in a frozen state and refrains from incorporating additional trainable components within the Transformer blocks (Jin et al., 2023a), which may limit the models' ability to adapt to specific tasks more effectively.

To address the above issues, we introduce Time-LlaMA, an innovative framework designed to harness large language models for time series forecasting. Our approach diverges from prior methodologies (Zhou et al., 2023; Jin et al., 2023a) in the following aspects. First, we treat each channel within multivariate time series data as an individual token. Furthermore, we employ a trainable cross-attention

[1] Codes will be made public upon acceptance.

[2] https://claude.ai/

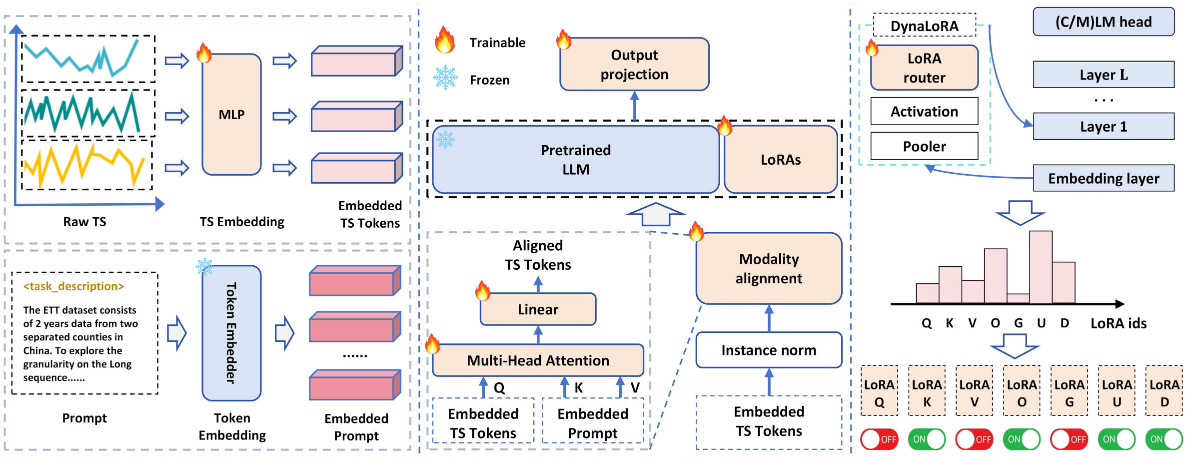

Figure 1: Schematic illustration of our Time-LlaMA framework.

module to align the tokenized time series data with the embeddings of the text prompt, rather than the entire vocabulary, thereby enhancing the model's focus on relevant information. Notably, the text prompt is not passed through the Transformer backbone to minimize inference delay. Additionally, we present DynaLoRA, a novel variant of the LoRA technique that incorporates a mixture-of-experts mechanism. DynaLoRA dynamically assigns distinct sets of LoRA modules to various input samples, leading to improved performance across the board. Extensive experimentation has proved that our Time-LlaMA method surpasses recent state-of-the-art baseline methods. The contributions of our work are summarized as follows:

- We propose a novel framework Time-LlaMA. By aligning to text prompts and fine-tuning the LLMs with a novel DynaLoRA method, our work pushs the limit of LLM based TS modeling methods.
- Time-LlaMA consistently exceeds state-of-the-art performance in TS forecasting tasks, especially in few-shot and zero-shot scenarios. Moreover, this superior performance is achieved while maintaining excellent inference efficiency, making our method suitable for industrial usage.

2 Related Work

Time series modeling. The progressive advancements in natural language processing and computer vision have led to the development of sophisticated Transformer (Vaswani et al., 2017) variants tailored for a wide array of time series forecasting applications (Zhou et al., 2021; Wu et al., 2021). Central to these innovations is the methodology by which Transformers handle time series data. For instance, I-Transformer (Liu et al., 2023b) treats each univariate time series as a distinct token, forming multivariate time series into sequences of such tokens. More recently, PatchTST (Nie et al., 2022) adopts an assumption of channel independence, transforming a univariate time series into multiple patches, which are subsequently treated as tokens and processed through a Transformer encoder. This approach has yielded notable results on various benchmark datasets for time series. Nevertheless, these forecasting models are trained end-to-end using task-specific datasets. A recent trend involves the developments of Transformer-based foundational models for time series analysis (Das et al., 2023; Goswami et al., 2024) via pre-training, capable of being swiftly adapted to diverse downstream tasks.

Cross-modal transfer learning using language models Recent investigations have highlighted the efficacy of transferring Transformer models (Vaswani et al., 2017), which are pretrained on extensive textual corpora, to other modalities. (Lu et al., 2022) employs a frozen pretrained Transformer across a spectrum of sequence classification tasks encompassing numerical computation, vision, and protein structure prediction, training only the newly introduced classification heads. ORCA (Shen et al., 2023) adopts an align-then-refine workflow to adapt to target tasks. Specifically, given the target input, ORCA initially learns an embedding network that aligns the feature distribution of the embedded data with that of the pretraining modality. Subsequently, the pretrained model is fine-tuned on the aligned data to harness cross-modal knowledge. Building upon these capabilities, recent studies have successfully adapted large language models (LLMs) for time series analysis through the use of a reprogramming module and a tokenization technique, while maintaining the LLMs in a frozen state (Zhou et al., 2023; Jin et al., 2023a). Our contribution to this body of research is twofold: (a) we conceptualize each time series variable as a token, enabling simultaneous predictions for all variables within a single forward pass, thereby enhancing efficiency. (b) We introduce a novel LoRA methodology that fine-tunes the LLM backbone in a parameter-efficient manner, advancing the state-of-the-art in LLM-based time series modeling.

Parameter efficient fine-tuning for pretrained Transformer models Parameter-efficient fine-tuning (PEFT) optimizes a small portion of added parameters when fine-tuning a LLM and keeps the backbone model frozen (Ding et al., 2022; Zhang et al., 2023b). LoRA (Hu et al., 2021) is inspired by (Aghajanyan et al., 2021) and (Li et al., 2018), and hypothesizes that the change of weights during model fine-tuning has a low intrinsic rank and optimizes the low-rank decomposition for the change of original weight matrices. LoRA (Hu et al., 2021) is proven to be effective and yield stable results when applied to both relatively small pretrained backbones and large language models (Dettmers et al., 2023; Zhu et al., 2023). However, the original LoRA paper does not specify how to add LoRA modules of different ranks to the Transformer backbones for adapting different tasks. In this work, we propose a novel LoRA variant that can help the LLM backbone to better adapt to the time series prediction tasks and achieve state-of-the-art performance.

3 Methodology

This section elaborates on the model architecture of our Time-LlaMA framework as illustrated in Figure 1. In this study, we address the challenge of multivariate time series prediction. Given a sequence of historical observations $\mathbf{X} \in \mathcal{R}^{N \times T_L}$ consisting of N different 1-dimensional variables across T_L time steps, we aim to adapt a large language model $f(\cdot)$ to understand the input time series and accurately forecast the values at T_P future time steps, denoted by $\mathbf{Y} \in \mathcal{R}^{N \times T_P}$.

3.1 Preliminaries

Transformer model As depicted in Figure 1, each Transformer layer of a LLM with L layers such as LlaMA-2 (Touvron et al., 2023) consists of a multi-head self-attention (MHA) module and a fully connected feed-forward (FFN) sub-layer. MHA contains four linear modules, which are the Query (Q), Key (K), Value (V), and Output (O) modules. FFN contains three linear modules: Gate (G), Up (U), and Down (D). For notation convenience, we will refer to the number of modules in a Transformer block as N_{mod}. Thus, in LlaMA-2, $N_{mod} = 7$.

LoRA For any linear module $m \in$ {Q, K, V, O, G, U, D} in the Transformer layer, the LoRA method adds a pair of low-rank matrices to reparameterize its weights. Formally, the forward calculation of module m in layer l with LoRA is:

$$x^{'} = xW_{m,l} + g_{m,l} * xW^{A}_{m,l}W^{B}_{m,l} + b_{m,l}, \quad (1)$$

where $W_{m,l} \in \mathbf{R}^{d_1 \times d_2}$ is the weight matrix of module m, $b_{m,l}$ is its bias term. $W^{A}_{m,l} \in \mathbb{R}^{d_1 \times r}$ and $W^{B}_{m,l} \in \mathbb{R}^{r \times d_2}$ are the low-rank matrices for the LoRA module, and $r \ll \min(d_1, d_2)$. r is the rank of the two matrices and will also be referred to as the rank of the LoRA module. Here, we include a binary gate $g_{m,l} \in \{0, 1\}$ to conveniently control the inclusion of LoRA m in the forward calculation. For the vanilla LoRA method, all the LoRA gates $g_{m,l}$ are set to 1.

3.2 Time-LlaMA

We now describe the forward calculation process of Time-LlaMA

Token Embedding In order to seamlessly apply the LLM to time series prediction, we consider the i-th variate $X_{i,:}$'s whole series as a token (Liu et al., 2023b), and embed it with:

$$\mathbf{h}_i^{TS,0} = \text{TSEmb}(X_{i,:}), \quad (2)$$

where TSEmb : $\mathcal{R}^{T} \mapsto \mathcal{R}^{d_m}$ denotes the time-series token embedding module, d_m denotes the hidden size of the LLM backbone. And $\mathbf{H}^{TS,0} = \{\mathbf{h}_1^{TS,0}, ..., \mathbf{h}_N^{TS,0}\}$ denotes the whole token sequences of the input time series.

Modality Alignment Note that time series is different from the language modality, making it difficult for the LLM to understanding time series. To close this gap, we propose to align the time-series token embeddings $\mathbf{H}^{0}$ with the prompts' embeddings $\mathbf{H}^{P,0}$. To realize this alignment, we utilize a multi-head cross-attention (MHCA) layer where $\mathbf{H}^0$ acts as the query tensor and $\mathbf{H}^{P,0}$ acts as the key and value tensor. Specifically, for each attention head $k \in \{1, 2, ..., K\}$, we define the query tensors as $Q_k = \mathbf{H}^0 W_k^Q$, the key tensors as $K_k = \mathbf{H}^{P,0} W_k^K$, and the value tensors as $V_k = \mathbf{H}^{P,0} W_k^V$, where $W_k^Q, W_k^K, W_k^v \in \mathcal{R}^{d_m \times d_{head}}$ are the weight matrices, $d_{head} = d_m / K$ is the hidden dimension on each head. Then the time-series token embeddings are aligned to the natural language representation via the following equations:

$$\begin{aligned} A_k &= \text{Softmax}\left(\frac{Q_k K_k^\intercal}{\sqrt{d_{head}}}\right) \\ \mathbf{H}^0 &\leftarrow \mathbf{H}^0 + \text{Concat}([A_1, ..., A_K]) W^O, \end{aligned} \tag{3}$$

where Concat() is the concatenation operation, and $W^O \in \mathcal{R}^{d_m \times d_m}$ is the attention output projection matrix. Then the input for the LLM's Transformer blocks $\mathbf{H}^0$ is obtained by projecting $\mathbf{H}^0$ to dimension d_{model}, the hidden dimension of the LLM.

LLM backbone Time-LlaMA utilize a pre-trained LLM backbone to encode the input tokens. Different from the previous works, we install our novel DynaLoRA module on each Transformer layer. The details are presented in the next subsection.

Output layer and loss calculation After $\mathbf{H}^0$ is encoded by the LLM, we obtain the output representation $\mathbf{H}^L$. Then $\mathbf{H}^L$ will go through a linear layer to obtain the predictions for the future T_P time steps:

$$\hat{\mathbf{Y}} = \mathbf{H}^L W^P + b^P, \tag{4}$$

where $W^P \in \mathcal{R}^{d_m \times T_P}$ is the weight matrix, and $b^P \in \mathcal{R}^{1 \times T_P}$ is the bias term.

Following the standard practice for the time-series prediction tasks, the objective is to minimize the mean square errors between the ground truths $\mathbf{Y}$ and predictions$\hat{\mathbf{Y}}$:

$$\mathcal{L}_{mse} = ||\mathbf{Y} - \hat{\mathbf{Y}}||_F^2. \tag{5}$$

Following (Fedus et al., 2022), to better train our DynaLoRA module, we add a load balancing loss to the training loss function. Consider a training batch B with N_B samples, let f_i^l represent the proportion of prompts assigned to the i-th LoRA expert in layer l,

$$f_i^l = \frac{1}{N_B} \sum_{x \in B} \mathbf{1}\{\arg\max_j p_j^l(x) = i\}, \tag{6}$$

where p_j^l is the probability of expert j, output by the router l. Let $\hat{p}_i^l$ be the average of probability masses received by the i-th expert, $\hat{p}_i^l = \frac{1}{N_B} \sum_{x \in B} p_i^l(x)$. Then, the load balancing loss is given by:

$$\mathcal{L}_{lb} = N_{mod} \sum_{l=1}^{L} \sum_{i=1}^{N_{mod}} f_i^l \cdot \hat{p}_i^l. \tag{7}$$

The $\mathcal{L}_{lb}$ loss term is added to the cross entropy loss with a coefficient $\lambda_{lb} \geq 0$:

$$\mathcal{L} = \mathcal{L}_{mse} + \lambda_{lb} * \mathcal{L}_{lb}. \tag{8}$$

3.3 DynaLoRA

In the previous works (Zhou et al., 2023; Jin et al., 2023a) on applying LLM backbones to the time series tasks, the LLMs are kept entirely frozen, making it convenient for task adaptation. However, this setting restricts the expressiveness of the whole model. Inspired by the recent works on parameter-efficient fine-tuning in the LLM research, we propose to fine-tune the LLM backbone in a parameter-efficient manner when adapting it to time-series tasks. However, through initial experiments, we find that the vanilla LoRA method (Hu et al., 2021) does not perform well on all the time-series prediction tasks. We hypothesize that when adapted to different time-series tasks, how to set the LoRA modules should differ significantly. In this work, we take a step further and propose an input-adaptive dynamic LoRA (DynaLoRA) method (on the right hand side of Figure 1), which dynamically assign LoRA modules to the different Transformer modules based on the input.

We now present the details of our DynaLoRA method. The core of DynaLoRA is the input-dependent LoRA assignment mechanism, as shown in Figure 1. Under this mechanism, a LoRA router takes the input's hidden states as input and outputs the assigned LoRA experts for the current layer. Denote the hidden state of the input right before the Transformer layer l as $\mathbf{H}^{l-1} \in \mathbf{R}^{N \times d_m}$. Then a pooling operation transforms it to a single vector $\mathbf{h}_{pooled}^l \in \mathbf{R}^{1 \times d_m}$:

$$\mathbf{h}_{pooled}^l = \text{Pooler}(\mathbf{H}^{l-1}). \tag{9}$$

Consistent with (Radford et al., 2018) and (Lewis et al., 2019), Pooler() takes the vector representation of the last token in the input as $\mathbf{h}_{pooled}^l$. Then, $\mathbf{h}_{pooled}^l$ will go through an activation function g and then the LoRA router R^l right before layer l. R^l assigns the current input to the most suitable LoRA modules. This router contains (a) a linear layer that computes the probability of $\mathbf{h}^l$ being routed to each LoRA module LoRA_m ($m \in$ {Q, K, V, O, G, U, D}), (b) a softmax function to model a probability distribution over the LoRA modules, and finally, (c) a Top_K$(\cdot, n)$ function that choose the top $n > 0$ experts with the highest probability masses. Formally,

$$R^l(\mathbf{h}^l) = \text{Top\_K}(\text{Softmax}(g(\mathbf{h}^l)W_r^l), n), \quad (10)$$

where $W_r^l \in \mathbf{R}^{d_m \times N_{mod}}$ is the router's weight. $R^l(\mathbf{h}^l)$ is a N_{mod}-dim vector, in which the m-th element is a binary value in $\{0, 1\}$ and is assigned to $g_{m,l}$ to activate or deactivate LoRA m:

$$g_{m,l} \leftarrow R^l(\mathbf{h}^l)[m], \quad (11)$$

and $\sum_{m=1}^{N_{mod}} g_{m,l}$ equals n. The LoRA router dynamically selects and activates the best $n > 0$ experts for each input during inference.

Different from the standard LoRA method (Hu et al., 2021), our work: (a) determines the assigned LoRA modules at the Transformer's layer level, selecting which Transformer module should be modified by its corresponding LoRA module. (b) The decision on selecting LoRA modules are conditioned on the input data, and different test samples could set LoRA modules differently. (c) Note that for a test input, different Transformer layers may choose to assign different LoRA modules. (d) Note that we can adjust the number of assigned LoRA modules n per layer, making inference more efficient than the vanilla LoRA method or previous dynamic LoRA methods (Liu et al., 2023a).

4 Experiments

4.1 Baselines

We compare our Time-LlaMA method with the SOTA time series models: (a) Time-LLM (Jin et al., 2023a), (b) GPT4TS (Zhou et al., 2023), (c) PatchTST (Nie et al., 2022), (d) DLinear (Zeng et al., 2023), and (e) TimesNet (Wu et al., 2022).

4.2 Datasets and evaluation metrics

For long-term time series forecasting, we assess our Time-LlaMA framework on the following datasets, in accordance with (Wu et al., 2022): ETTh1, ETTm1, Weather, ECL, and Traffic. For short-term time series forecasting, we employ the M4 benchmark (Makridakis et al., 2018). Detailed introductions to data sets and evaluation metrics are in the Appendix A.

4.3 Experimental setups

We use Llama-3 1B (Grattafiori et al., 2024) as the default LLM backbone unless stated otherwise, thus $d_m = 2048$. We utilize the first $L = 6$ Transformer blocks of the LLM for our Time-LlaMA framework. For the alignment module, the number of attention heads is $K = 8$. For DynaLoRA, the LoRA rank is set to $r = 4$, and each layer will select $n = 4$ LoRA modules during inference.

The Adam optimizer (Loshchilov, 2017) is employed throughout all experiments. The loss objective is MSE for the long-term forecasting tasks, and SMAPE for the short-term forecasting tasks. The learning rate is denoted as LR. We utilize the LlaMA-2 7B (Touvron et al., 2023) model, maintaining the backbone model layers at 8 across all tasks. Denote the lookback window's length as T_L, the prediction horizon as T_P. And the heads K correlate to the multi-head cross-attention utilized for time-series data reprogramming. For the LoRA modules, the number of ranks r is set to 8. Each Transformer block's LoRA router activates $n = 4$ LoRA modules. We detail the configurations for each task in Table 7 of Appendix A.

4.4 Main results

Results for long-term forecasting For the long-term forecasting tasks, the input time series length T_L is set as 512, and we use four different prediction horizons $T_P \in \{96, 192, 336, 720\}$ ($H \in \{24, 36, 48, 60\}$ for the ILI task). The evaluation metrics include mean square error (MSE) and mean absolute error (MAE). In Table 1, we report the average score over four different horizons.

The experimental results demonstrate that our Time-LlaMA method outperforms the baselines on most of the (task, prediction horizon) pairs. The comparison against Time-LLM (Jin et al., 2023a) and GPT4TS (Zhou et al., 2023) is particularly meaningful. These two are very recent works on adapting large language models to the time-series forecasting tasks. When compared to the pre vious state-of-the-art (SOTA) model PatchTST which is trained from scratch on each task, Time-LlaMA can also achieves advantages.

Results for short-term forecasting To demonstrate that our method works in the short-term forecasting tasks, we utilize the M4 benchmark (Makri-

Methods		Time-LlaMA		TIME-LLM		GPT4TS		PatchTST		DLinear		TimesNet	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	96	0.377	0.398	0.386	0.409	0.376	0.397	0.378	0.405	**0.375**	0.399	0.384	0.402
	192	0.410	0.426	0.414	0.421	0.416	0.418	0.413	0.421	**0.405**	0.416	0.436	0.429
	336	0.421	0.437	0.423	0.436	0.442	0.433	**0.422**	0.436	0.439	0.443	0.491	0.469
	720	**0.443**	0.464	0.481	0.478	0.477	0.456	0.447	0.466	0.472	0.490	0.521	0.500
ETTm1	96	0.291	0.343	0.298	0.356	0.292	0.346	**0.290**	0.342	0.299	0.343	0.338	0.375
	192	**0.326**	0.366	0.334	0.377	0.332	0.372	0.332	0.369	0.335	0.365	0.374	0.387
	336	**0.352**	0.384	0.365	0.389	0.366	0.394	0.366	0.392	0.369	0.386	0.410	0.411
	720	**0.405**	0.416	0.413	0.418	0.417	0.421	0.416	0.420	0.425	0.421	0.478	0.450
Weather	96	0.151	0.207	0.154	0.208	0.162	0.212	**0.149**	0.198	0.176	0.237	0.172	0.220
	192	**0.193**	0.240	0.198	0.247	0.204	0.248	0.194	0.241	0.220	0.282	0.219	0.261
	336	**0.242**	0.287	0.251	0.282	0.254	0.286	0.245	0.282	0.265	0.319	0.280	0.306
	720	**0.313**	0.332	0.317	0.338	0.326	0.337	0.314	0.334	0.333	0.362	0.365	0.359
ECL	96	**0.128**	0.224	0.137	0.235	0.139	0.238	0.129	0.222	0.140	0.237	0.168	0.272
	192	**0.152**	0.247	0.158	0.242	0.153	0.251	0.157	0.240	0.153	0.249	0.184	0.289
	336	**0.161**	0.256	0.164	0.261	0.169	0.266	0.163	0.259	0.169	0.267	0.198	0.300
	720	0.198	0.292	0.204	0.293	0.206	0.297	**0.197**	0.290	0.203	0.301	0.220	0.320
Traffic	96	0.379	0.270	0.382	0.274	0.388	0.282	**0.378**	0.269	0.410	0.282	0.593	0.321
	192	**0.396**	0.279	0.404	0.285	0.407	0.290	0.398	0.280	0.423	0.287	0.617	0.336
	336	**0.404**	0.282	0.410	0.291	0.412	0.294	0.406	0.282	0.436	0.296	0.629	0.336
	720	**0.446**	0.306	0.456	0.308	0.450	0.312	0.448	0.307	0.466	0.315	0.640	0.350

Table 1: Results for the long-term forecasting tasks. The prediction horizon T_P is one of $\{24, 36, 48, 60\}$ for ILI and one of {96, 192, 336, 720} for the others. Lower value indicates better performance. **Bold** values represent the best MSE score, while Underlined means the second best MSE score.

Methods	Time-LlaMA	TIME-LLM	GPT4TS	PatchTST	DLinear	TimesNet
SMAPE	**11.96**	12.01	12.69	12.06	13.63	12.88
MSAE	**1.656**	1.663	1.808	1.683	2.095	1.836
OWA	**0.881**	0.896	0.942	0.905	1.051	0.955

Table 2: Results for the short-term time series forecasting task, M4. The forecasting horizons are in {6, 48}. Lower value indicates better performance. **Bold** values represent the best score, while Underlined means the second best.

dakis et al., 2018). Table 2 reports the SMAPE, MSAE and OWA scores. Our experimental results demonstrate that our Time-LlaMA method consistently surpasses all baselines when conducting short-term time series predictions.

Results for the few-shot setting Note that a great property of large language models is its great few-shot learning capability. And it is interesting to investigate whether this capability still stands when they are adapted to model time series. We experiment on the scenarios in which limited training data are available for training, that is, only 5% of the training time steps in the original training set are utilized for training. We experiment with the Weather and ETTh1 tasks, and the results are presented in Table 3.

From Table 3, we can observe that Time-LlaMA excels over all the strong baseline methods. The comparison between Time-LlaMA and the non-LLM method like PatchTST demonstrates the advantage of utilizing a pre-trained large language model. The pre-trained LLM contains rich world and semantically knowledge, thus providing a high-quality model parameter initialization for the time-series models. The results underscore the prowess

Methods		Time-LlaMA		TIME-LLM		PatchTST	
Metric		MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	0.166	0.220	0.169	0.223	0.175	0.230
	192	0.219	0.268	0.224	0.272	0.227	0.276
	336	0.272	0.297	0.276	0.303	0.286	0.322
	720	0.355	0.360	0.362	0.368	0.366	0.379
ETTh1	96	0.531	0.497	0.538	0.501	0.543	0.506
	192	0.685	0.546	0.698	0.557	0.748	0.580
	336	0.738	0.573	0.752	0.591	0.754	0.595
	720	-	-	-	-	-	-

Table 3: Results for the few-shot setting. The first 5% of the training sets used in Table 1 are used for training. '-' means that 5% time series is not sufficient to constitute a training set.

Methods		Full-data setting		Few-shot setting	
		Time-LlaMA	Time-LLM	Time-LlaMA	Time-LLM
Results for Gemma 2B					
Weather	96	0.153	0.157	0.169	0.173
	192	0.198	0.204	0.226	0.231
ETTh1	96	0.379	0.401	0.553	0.566
	192	0.421	0.432	0.706	0.718
Results for GPT-2 large (0.5B)					
Weather	96	0.164	0.169	0.187	0.199
	192	0.205	0.211	0.235	0.243
ETTh1	96	0.387	0.398	0.581	0.594
	192	0.432	0.438	0.727	0.742

Table 4: Results on the other LLMs. For the few-shot setting, 5% of the original training set is utilized for training. We report the MSE scores.

of LLMs as a powerful time series model. The comparison against Time-LLM and GPT4TS emphasize our method's advantage in both knowledge activation and task adaptation, which are directly due to the input-adaptive DynaLoRA module and the modality alignment module.

4.5 Ablation studies and analysis

Ablation on the LLM backbones To validate our framework's wide applicability, we experiment on two representative backbones Gemma 2B (Banks and Warkentin, 2024) and GPT-2 large (Radford et al., 2019). The results on the Weather and ETTh1 under the full-data and few-shot setting are reported in Table 4. The Time-LlaMA method also outperforms Time-LLM by clear margins, under both the full-data and few-shot settings, demonstrating the effectiveness of our method with different LLM backbones.

Ablation studies of our Time-LlaMA method In order to understand the superiority of our Time-LlaMA framework (as in Table Table 1, 2, and 3), we now conduct ablation studies on our Time-LlaMA method. We consider the following variants for Time-LlaMA: (a) Time-LlaMA-1, which removes the modality alignment module (Eq 3), and directly feed the time series tokens to the LLM backbone. (b) Time-LlaMA-2, which concatenate the text prompt to the left of the time-series tokens, serving as prefix. (c) Time-LlaMA-3 keeps the LLM backbone entirely frozen. (d) Time-LlaMA-4 substitutes our DynaLoRA mechanism to the vanilla LoRA method. (e) Time-LlaMA-5 substitutes DynaLoRA to a representative LoRA variant, AdaLoRA (Zhang et al., 2023a). (f) Time-LlaMA-6 substitutes DynaLoRA to MOELoRA (Liu et al., 2023a).

Methods	Weather		ETTh1	
	96	192	96	192
Time-LlaMA	**0.166**	**0.219**	**0.531**	**0.685**
Time-LlaMA-1	0.172	0.226	0.538	0.697
Time-LlaMA-2	0.165	0.221	0.533	0.685
Time-LlaMA-3	0.178	0.232	0.542	0.705
Time-LlaMA-4	0.174	0.227	0.537	0.696
Time-LlaMA-5	0.179	0.231	0.540	0.703
Time-LlaMA-6	0.171	0.227	0.536	0.695

Table 5: Results for the ablation study.

The experiments are presented in Table 5. From Table 5, we can observe that: (a) The comparison between Time-LlaMA-1 and Time-LlaMA demonstrates the necessity of the modality alignment module. (b) Time-LlaMA-2 performs closely to Time-LlaMA, demonstrating that with our modality alignment module, the text prompts containing the task information are no longer needed. (c) The comparison between Time-LlaMA-3 and Time-LlaMA shows that fine-tuning the LLM backbone in a parameter-efficient style helps our Time-LlaMA to achieve superior performance. (d) The comparisons among Time-LlaMA-4, Time-LlaMA-5, Time-LlaMA-6 and Time-LlaMA demonstrate the superiority of our method to the recent LoRA variants. Our DynaLoRA module adaptively adjust which LoRA modules are used to conduct inference for the current test sample, achieving stronger generalization capabilities.

Effects on the number of selected LoRA modules

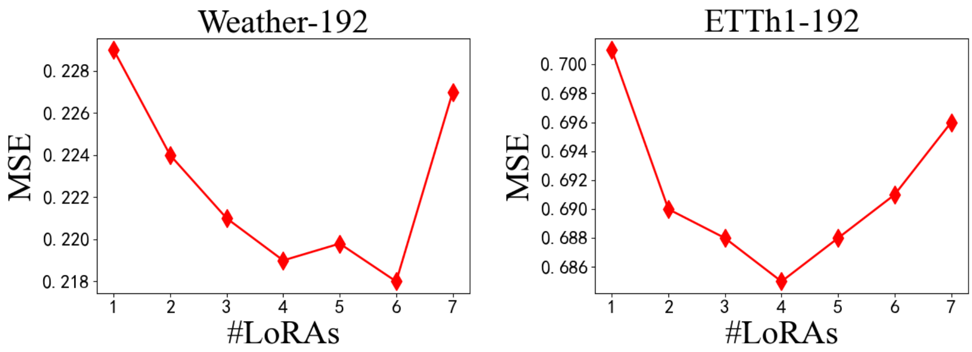

Figure 2: Performances under different numbers of selected LoRAs per Transformer block.

n We now alter the number of selected LoRA modules n to $\{1, 2, 3, 5, 6, 7\}$, and investigate how this hyper-parameter affects our Time-LlaMA method. The results are demonstrated in Figure 2. From the experiments, one can see that when n changes from 1 to 7, the performance first becomes better, and then drops. The observations are consistent with ALoRA (Liu et al., 2024), which demonstrates that reduce the number of LoRA modules per block is beneficial for the LLM's downstream adaptation.

Efficiency analysis In our main experiments (Table 1), we only utilize the first 6 blocks of the LlaMA-3 1B model to encode the time-series information and make predictions. Thus, its inference speed is 10.47 test samples per second on the test set of the Traffic task. Note that in the industrial applications, efficiency is an important factor. Thus, it is of value to compare the latency of our method and the non-LLM method PatchTST. Note that PatchTST transforms the multi-variate time series task like Traffic into multiple single-variate time series tasks. Thus, it has to conduct inference for 862 single-variate series for a single sample in Traffic. Following its original implementations, PatchTST's inference speed is 13.24 samples per second. Time-LLM (Jin et al., 2023a) also utilizes the patching mechanism in PatchTST. Thus, its inference speed is 3.51 samples per second. The comparisons demonstrate that through our Time-LlaMA method is actually very efficient, even with LLM backbones.

Distributions of the selected LoRAs We now compare the distribution of LoRA modules across all Transformer layers on the Weather and ETTh1 tasks' test sets (with $T_P = 192$) in Figure 3. We can observe that: (a) different Transformer layers choose to select different LoRA experts via their corresponding routers, and the maximum proportion a LoRA expert can achieve is less than 25%. The results are intuitive since Transformer layers of different depths represent different knowledge, requiring different LoRA experts to express. (b) the LoRA distributions on different tasks are different. For example, more layers activate LoRA G or LoRA U on the Weather task than on the ETTh1 task.

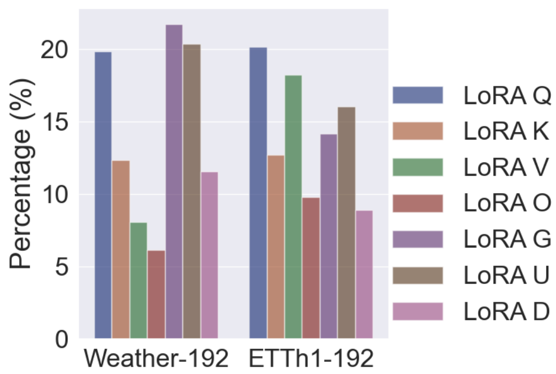

Figure 3: Distribution of activated LoRA experts.

5 Conclusion

In this work, we propose a novel framework, Time-LlaMA. First, Time-LlaMA tokenizes each time series sample by considering each variate as a token. Then we align the time series tokens to the language modality by attending to text prompts' embeddings. Third, the LLM backbone is fine-tuned by a novel LoRA method, DynaLoRA, that adaptively selecting different LoRA modules for different time series samples. Extensive experiments have demonstrated that Time-LlaMA can outperform the recent SOTA baselines. In addition, our method demonstrates inference efficiency, making it applicable for the industry.

Limitations

In this work, we introduced the Time-LlaMA framework to enhance the time series forecasting performance when using LLM backbones as encoders. To address the drawbacks in the recents works on LLM-based time series forecasting models, a novel LoRA method, DynaLoRA is proposed. We have conducted experiments on various real-world time series forecasting tasks, and the experimental results demonstrate that our Time-LlaMA method can outperform the recent baselines.

However, we acknowledge the following limitations: (a) the more super-sized open-sourced LLMs, such as 7B, 14b or 30B models, are not experimented due to limited computation resources. (b) Other time series modeling tasks are not explored, like time series classification, anomaly detection. But our framework can be easily transferred to other backbone architectures and different types of tasks. It would be of interest to investigate if the superiority of our method holds for other large-scaled backbone models and other types of time series tasks. And we will explore it in future work.

Ethical statement

In this research, we have carefully considered the ethical implications of developing Time-LlaMA, a framework for time series forecasting using large language models (LLMs). We ensured data privacy by using only publicly available, anonymized, or permitted datasets, avoiding sensitive or proprietary information. To address potential biases, we employed diverse datasets and rigorous testing across domains. We minimized environmental impact by using efficient training techniques like DynaLoRA and energy-efficient hardware. Transparency and reproducibility were prioritized through detailed methodology descriptions and plans to release code and model weights. We also acknowledged dual-use concerns, encouraging responsible application of our work, and fostered inclusivity through collaborative and open research practices. These steps align our research with ethical AI development principles.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Jeanine Banks and Tris Warkentin. 2024. Gemma: Introducing new state-of-the-art open models. *Google. Available online at: https://blog.google/technology/developers/gemma-open-models/(accessed 6 April, 2024).*

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. 2023. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688.*

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Fine-tuning of Quantized LLMs. *arXiv e-prints*, page arXiv:2305.14314.

Ning Ding, Yujia Qin, Guang Yang, Fu Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Haitao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juan Li, and Maosong Sun. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *ArXiv*, abs/2203.06904.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. 2024. Moment: A family of open time-series foundation models. *arXiv preprint arXiv:2402.03885.*

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783.*

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685.*

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. 2023a. Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728.*

Ming Jin, Qingsong Wen, Yuxuan Liang, Chaoli Zhang, Siqiao Xue, Xue Wang, James Zhang, Yi Wang, Haifeng Chen, Xiaoli Li, et al. 2023b. Large models for time series and spatio-temporal data: A survey and outlook. *arXiv preprint arXiv:2310.10196*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the Intrinsic Dimension of Objective Landscapes. *arXiv e-prints*, page arXiv:1804.08838.

Qidong Liu, Xian Wu, Xiangyu Zhao, Yuanshao Zhu, Derong Xu, Feng Tian, and Yefeng Zheng. 2023a. Moelora: An moe-based parameter efficient fine-tuning method for multi-task medical applications. *arXiv preprint arXiv:2310.18339*.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. 2023b. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*.

Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. 2024. Alora: Allocating low-rank adaptation for fine-tuning large language models. *arXiv preprint arXiv:2403.16187*.

I Loshchilov. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. 2022. Frozen pretrained transformers as universal computation engines. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 7628–7636.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2018. The m4 competition: Results, findings, conclusion and way forward. *International Journal of forecasting*, 34(4):802–808.

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

OpenAI. 2023. GPT-4 Technical Report. *arXiv e-prints*, page arXiv:2303.08774.

Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2019. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Junhong Shen, Liam Li, Lucio M Dery, Corey Staten, Mikhail Khodak, Graham Neubig, and Ameet Talwalkar. 2023. Cross-modal fine-tuning: Align then refine. In *International Conference on Machine Learning*, pages 31030–31056. PMLR.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. 2022. Timesnet: Temporal 2d-variation modeling for general time series analysis. *arXiv preprint arXiv:2210.02186*.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in neural information processing systems*, 34:22419–22430.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128.

Qingru Zhang, Minshuo Chen, Alexander W. Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023a. Adaptive budget allocation for parameter-efficient fine-tuning. *ArXiv*, abs/2303.10512.

Yuming Zhang, Peng Wang, Ming Tan, and Wei-Guo Zhu. 2023b. Learned adapters are better than manually designed adapters. In *Annual Meeting of the Association for Computational Linguistics*.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115.

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

Wei Zhu, Xiaoling Wang, Huanran Zheng, Mosha Chen, and Buzhou Tang. 2023. PromptCBLUE: A Chinese Prompt Tuning Benchmark for the Medical Domain. *arXiv e-prints*, page arXiv:2310.14151.

A Appendix: Experimental settings

Now we provide more details for the experiments presented in the main contents.

A.1 Implementation

We mainly follow the experimental configurations in (Jin et al., 2023a) across all baselines within a unified evaluation pipeline in the Time-Series-Library[3] for fair comparisons. We use Llama-2 7B (Touvron et al., 2023) as the default backbone model, unless stated otherwise. All our experiments are repeated three times and we report the averaged results. Our method is implemented on PyTorch (Paszke et al., 2019) with all experiments conducted on NVIDIA L20 GPUs (48 GB RAM).

A.2 Datasets

We evaluate the long-term forecasting (ltf) performance on the well-established eight different benchmarks, including four ETT datasets (including ETTh1, ETTh2, ETTm1, and ETTm2) from (Zhou et al., 2021), Weather, Electricity, Traffic, and ILI from (Wu et al., 2021). For short-term time series forecasting (STF), we employ the M4 benchmark (Makridakis et al., 2018).

ETT The Electricity Transformer Temperature (ETT) is a crucial indicator in the electric power long-term deployment. This dataset consists of 2 years data from two separated counties in China. To explore the granularity on the Long sequence time-series forecasting (LSTF) problem, different subsets are created, ETTh1, ETTh2 for 1-hour-level and ETTm1 for 15-minutes-level. Each data point consists of the target value ”oil temperature” and 6 power load features. The train/val/test is 12/4/4 months.

ECL Measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Different electrical quantities and some sub-metering values are available.This archive contains 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months).

Traffic Traffic is a collection of hourly data from California Department of Transportation, which describes the road occupancy rates measured by different sensors on San Francisco Bay area freeways.

Weather Weather is recorded every 10 minutes for the 2020 whole year, which contains 21 meteorological indicators, such as air temperature, humidity, etc.

ILI The influenza-like illness (ILI) dataset contains records of patients experiencing severe influenza with complications.

M4 The M4 benchmark comprises 100K time series, amassed from various domains commonly present in business, financial, and economic forecasting. These time series have been partitioned into six distinctive datasets, each with varying sampling frequencies that range from yearly to hourly. These series are categorized into five different domains: demographic, micro, macro, industry, and finance.

The datasets' statistics are presented in Table 6.

A.3 Evaluation metrics

We now specify the evaluation metrics we used for comparing different models. We utilize the mean square error (MSE) and mean absolute error (MAE) for long-term forecasting. For the short-term forecasting task on M4 benchmark, we adopt the symmetric mean absolute percentage error (SMAPE), mean absolute scaled error (MASE), and overall weighted average (OWA), following (Oreshkin et al., 2019). The calculations of these

[3] https://github.com/thuml/Time-Series-Library

Tasks	Dataset	Dim.	Series Length	Dataset Size	Frequency	Domain
Long-term Forecasting	ETTm1	7	{96, 192, 336, 720}	(34465, 11521, 11521)	15 min	Temperature
	ETTm2	7	{96, 192, 336, 720}	(34465, 11521, 11521)	15 min	Temperature
	ETTh1	7	{96, 192, 336, 720}	(8545, 2881, 2881)	1 hour	Temperature
	ETTh2	7	{96, 192, 336, 720}	(8545, 2881, 2881)	1 hour	Temperature
	Electricity	321	{96, 192, 336, 720}	(18317, 2633, 5261)	1 hour	Electricity
	Traffic	862	{96, 192, 336, 720}	(12185, 1757, 3509)	1 hour	Transportation
	Weather	21	{96, 192, 336, 720}	(36792, 5271, 10540)	10 min	Weather
	ILI	7	{24, 36, 48, 60}	(617, 74, 170)	1 week	Illness
Short-term Forecasting	M4-Yearly	1	6	(23000, 0, 23000)	Yearly	Demographic
	M4-Quarterly	1	8	(24000, 0, 24000)	Quarterly	Finance
	M4-Monthly	1	18	(48000, 0, 48000)	Monthly	Industry
	M4-Weakly	1	13	(359, 0, 359)	Weakly	Macro
	M4-Daily	1	14	(4227, 0, 4227)	Daily	Micro
	M4-Hourly	1	48	(414, 0, 414)	Hourly	Other

Table 6: Dataset statistics. The dimension indicates the number of time series (i.e., channels), and the dataset size is organized in (training, validation, testing).

metrics are as follows:

$$\mathrm{MSE} = \frac{1}{H}\sum_{h=1}^{T}(\mathbf{Y}_h - \hat{\mathbf{Y}}_h)^2, \tag{12}$$

$$\mathrm{MAE} = \frac{1}{H}\sum_{h=1}^{H}|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|, \tag{13}$$

$$\mathrm{SMAPE} = \frac{200}{H}\sum_{h=1}^{H}\frac{|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|}{|\mathbf{Y}_h| + |\hat{\mathbf{Y}}_h|}, \tag{14}$$

$$\mathrm{MAPE} = \frac{100}{H}\sum_{h=1}^{H}\frac{|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|}{|\mathbf{Y}_h|}, \tag{15}$$

$$\mathrm{MASE} = \frac{1}{H}\sum_{h=1}^{H}\frac{|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|}{\frac{1}{H-s}\sum_{j=s+1}^{H}|\mathbf{Y}_j - \mathbf{Y}_{j-s}|}, \tag{16}$$

$$\mathrm{OWA} = \frac{1}{2}\left[\frac{\mathrm{SMAPE}}{\mathrm{SMAPE}_{\mathrm{Naive}}} + \frac{\mathrm{MASE}}{\mathrm{MASE}_{\mathrm{Naive}}}\right], \tag{17}$$

$$\tag{18}$$

where s is the periodicity of the time series data. H denotes the number of data points (i.e., prediction horizon in our cases). $\mathbf{Y}_h$ and $\hat{\mathbf{Y}}_h$ are the h-th ground truth and prediction where $h \in \{1, \cdots, H\}$.

A.4 Configurations for training

We detail the configurations for each task in Table 7.

Task-Dataset	Model Hyperparameter						Training Process			
	Layers	T_L	T_P	K	r	n	**LR***	**Loss**	**Batch Size**	**Epochs**
LTF - ETTh1	8	512	{96, 192, 336, 720}	8	8	4	10^{-3}	MSE	16	20
LTF - ETTm1	8	512	{96, 192, 336, 720}	8	8	4	10^{-3}	MSE	16	20
LTF - Weather	8	512	{96, 192, 336, 720}	8	8	4	10^{-3}	MSE	16	20
LTF - Electricity	8	512	{96, 192, 336, 720}	8	8	4	10^{-2}	MSE	16	20
LTF - Traffic	8	512	{96, 192, 336, 720}	8	8	4	10^{-2}	MSE	12	20
LTF - ILI	8	96	{24, 36, 48, 60}	8	8	4	10^{-2}	MSE	16	20
STF - M4	8	$2 \times T_P$	{6, 48}	8	8	4	10^{-3}	SMAPE	32	30

Table 7: An overview of the experimental configurations for TIME-LlaMA. LTF and STF denote long-term and short-term forecasting, respectively.