TimeLLM: Time Series Forecasting by Reprogramming Large Language Models

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, Qingsong Wen

Presenter: Shovito Barua Soumma

Date: October 16, 2024

- ICLR 2024, [96 Citations as today]
- Code Base: https://github.com/KimMeen/Time-LLM
- Easy Use: https://nixtlaverse.nixtla.io/neuralforecast/models.timellm.html

Introduction

TimeGPT-1

Azul Garza*, **Cristian Challu***, **Max Mergenthaler-Canseco** *
Nixtla
San Francisco, CA, USA
{azul,cristian,max}@nixtla.io

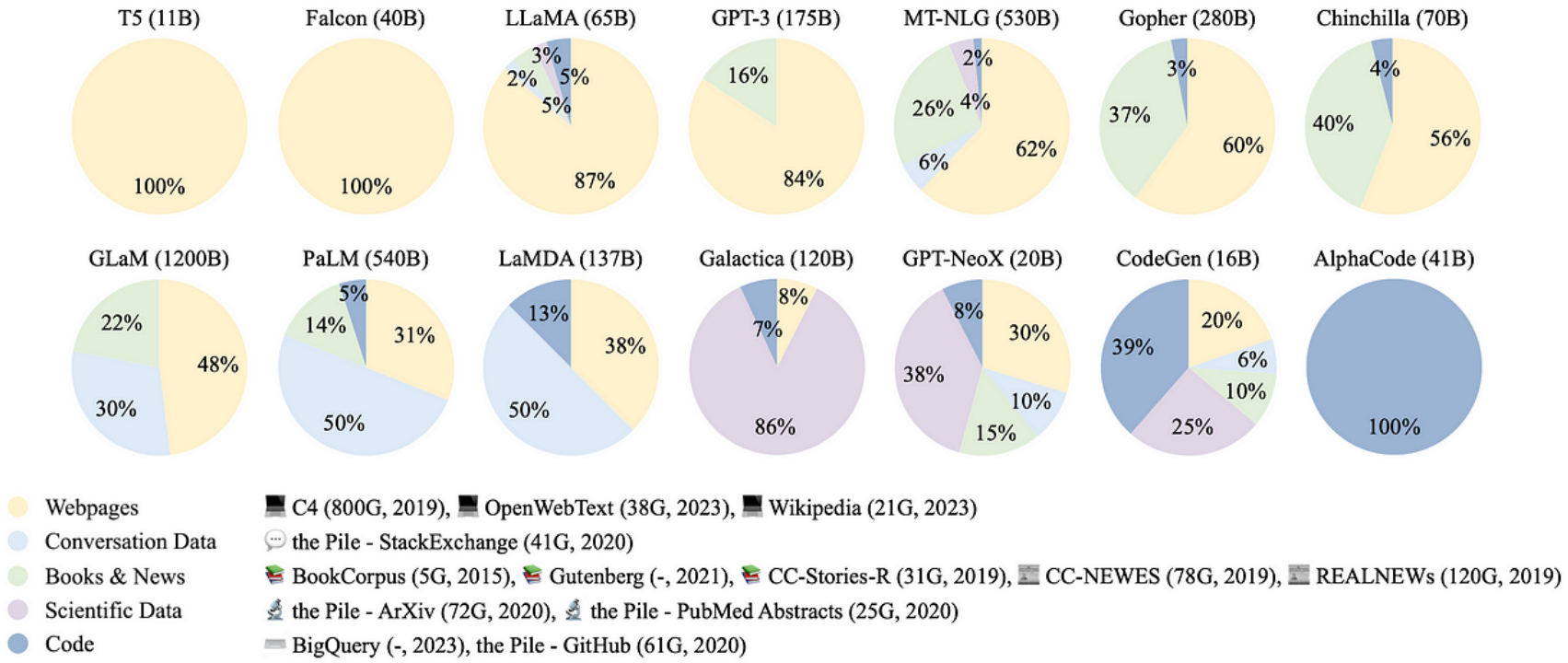
Abstract

In this paper, we introduce TimeGPT, the first foundation model for time series, capable of generating accurate predictions for diverse datasets not seen during training. We evaluate our pre-trained model against established statistical, machine learning, and deep learning methods, demonstrating that TimeGPT zero-shot inference excels in performance, efficiency, and simplicity. Our study provides compelling evidence that insights from other domains of artificial intelligence can be effectively applied to time series analysis. We conclude that large-scale time series models offer an exciting opportunity to democratize access to precise predictions and reduce uncertainty by leveraging the capabilities of contemporary advancements in deep learning.

- Time series forcasting is important in financial market.
- But freqeuenct changes in data distribution makes it challenging.
- Small amount of historical data
- Models are not generalizable for different task (weather vs stock)
- Training all the parameters using huge amount of dataset
- TimeGPT is a foundation model for time series forcasting => needs huge resource to retrain model

Dataset Statistics

- Lama (decoder only architecture)
- Not capable to handle timeseries data



A Comprehensive Survey of Large Language Models and Multimodal Large Language Models in Medicine, Hanguang Xiao et al 2024

TimeLLM

- Existing LLM model is not changed [No Finetuning]
- Introduces reprogramming to use the existing backbone w/o finetuning
- ReVINE Normalization method

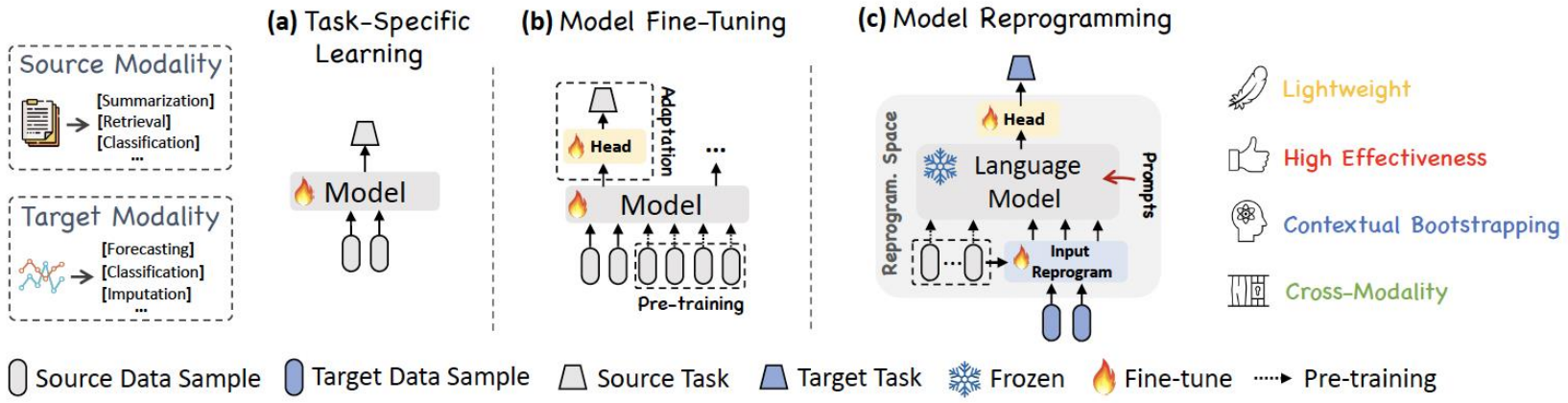
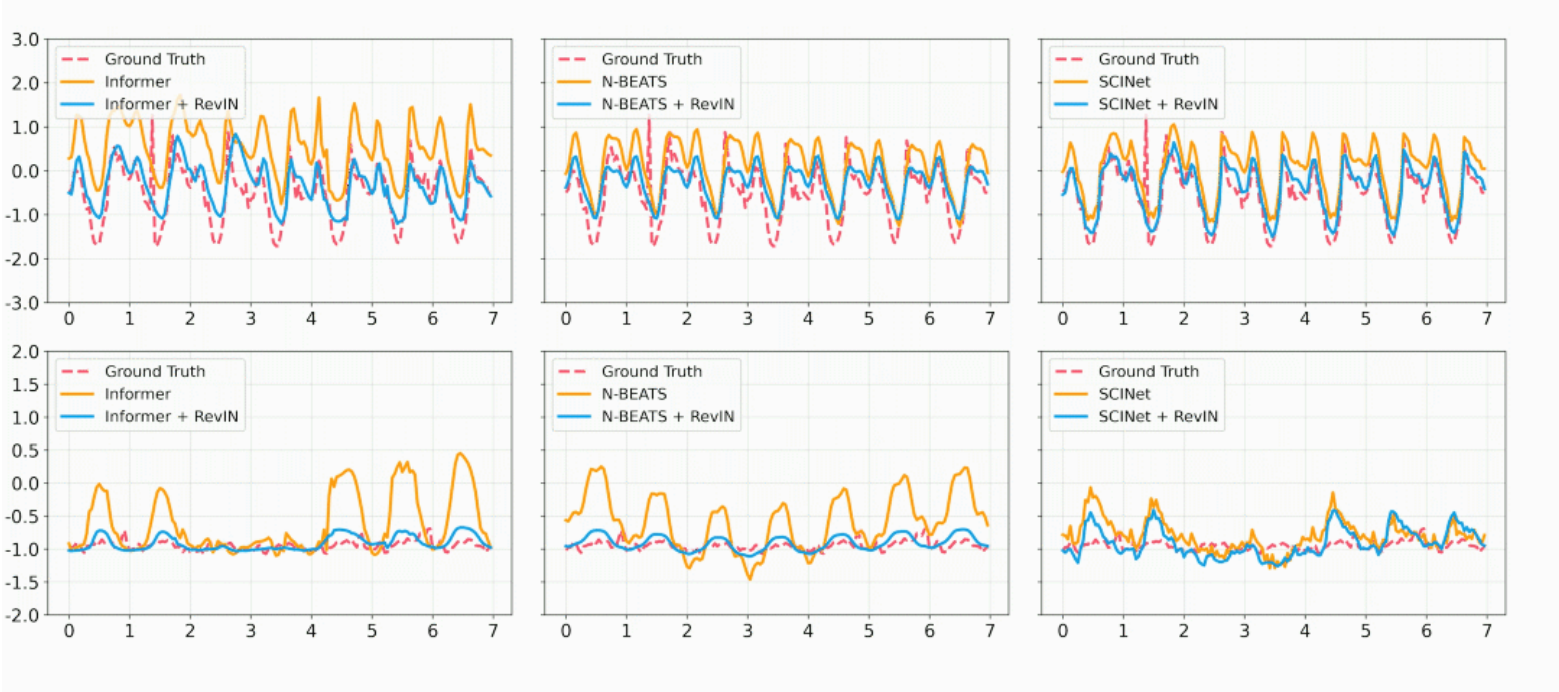


Figure 1: Schematic illustration of reprogramming large language models (LLMs) in comparison of **(a)** task-specific learning and **(b)** model fine-tuning. Our proposal investigates and demonstrates **(c)** how to effectively reprogram open-sourced LLMs as powerful time series learners where well-developed time series pre-trained models are not readily available.

RevIN (ICLR 2022)

- **Reversible Instance Normalization**
- https://github.com/ts-kim/RevIN



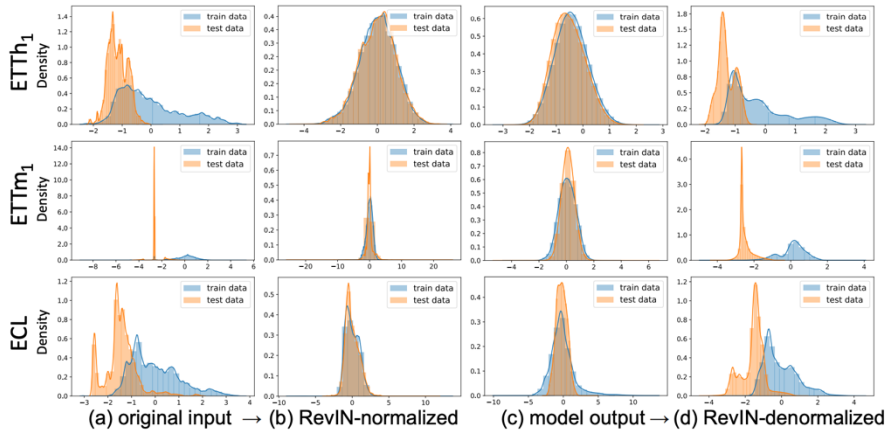
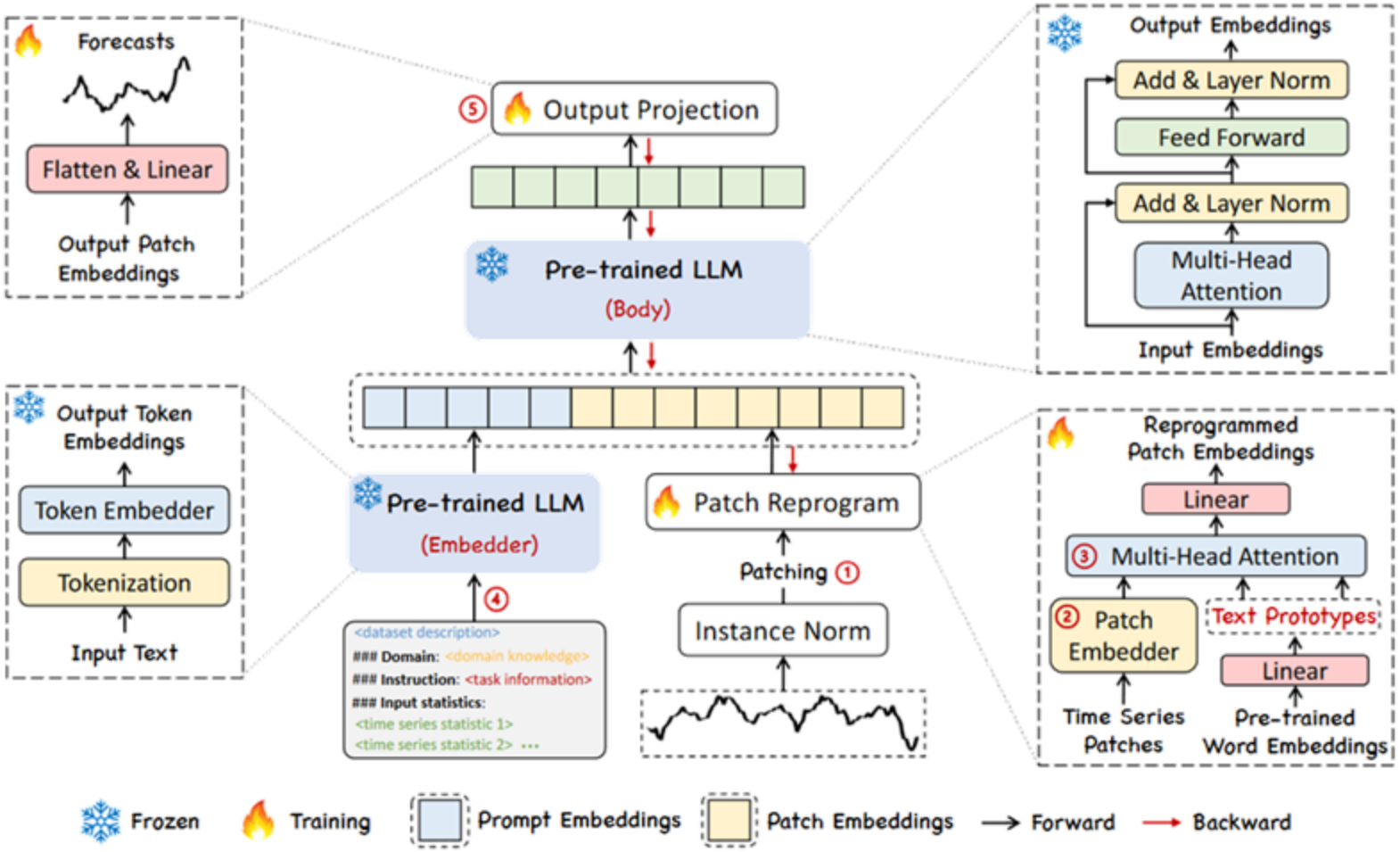


Figure 3: **Effect of RevIN on distribution discrepancy between training and test data.** From left to right columns, we compare the training and test data distributions of a variable on each step of the sequential process in RevIN: (a) the original input x, (b) the input $\hat{x}$ normalized by RevIN, (c) the model prediction output $\tilde{y}$, and (d) the output $\hat{y}$ denormalized by RevIN, the final prediction. The analysis is conducted on the ETT and ECL datasets using SCINet (Liu et al., 2021) as the baseline.

Motivation

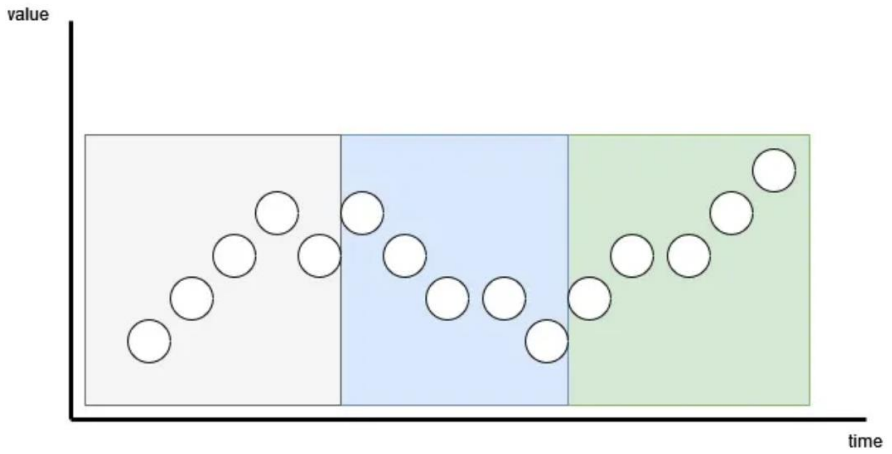
Note that this is different from fine-tuning the LLM. Instead, we teach the LLM to take an input sequence of time steps and output forecasts over a certain horizon. This means that the LLM itself stays unchanged.



- At a high level, Time-LLM starts by tokenizing the input time series sequence with a customized patch embedding layer. These patches are then sent through a reprogramming layer that essentially translate the forecasting task into a language task
- we can also pass a prompt prefix to augment the model's reasoning ability. Finally, the output patches go through the projection layer to ultimately get forecasts.

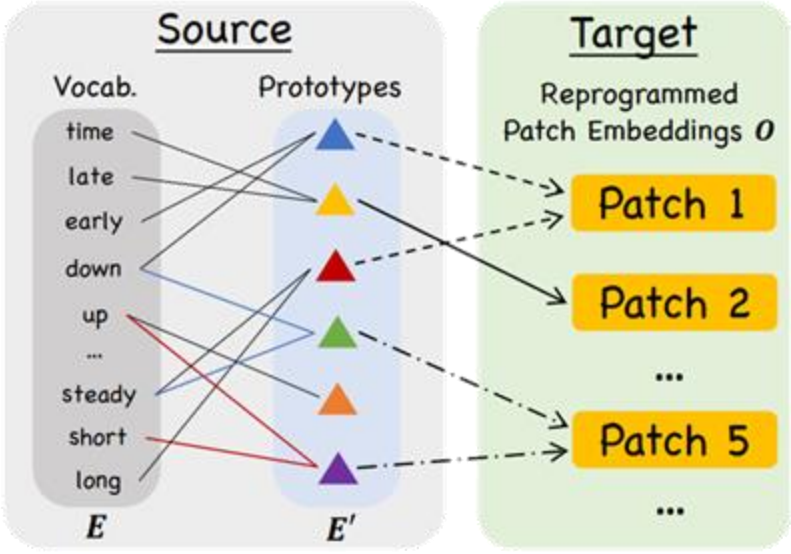
Input Patching

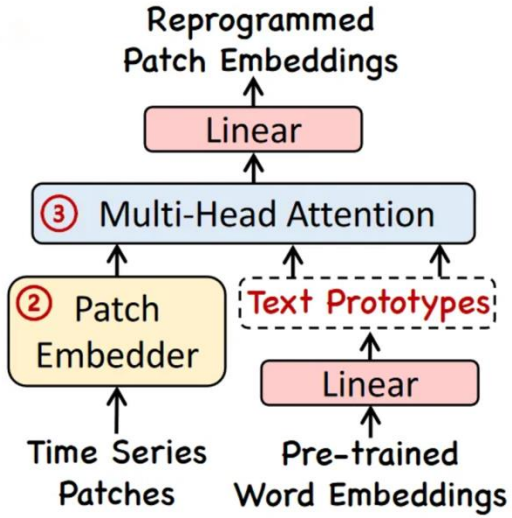
- Patch by patch (instead of looking at a single time stamps)
- Preserves the temporal semantic meaning
- Each patch is a token
  - Reduce the computation as lower no of token
- Once patching is done, the input sequence is sent to the reprogramming layer.



Transforming Input into Language Task

- Reprogramming Layer : It essentially maps the input time series into a language task, allowing us to leverage the capabilities of the language model.
- Once this is done, the translated patches are sent to a multi-head attention mechanism and a linear projection is done to align the dimension of the reprogrammed patches to the dimension of the LLM backbone.

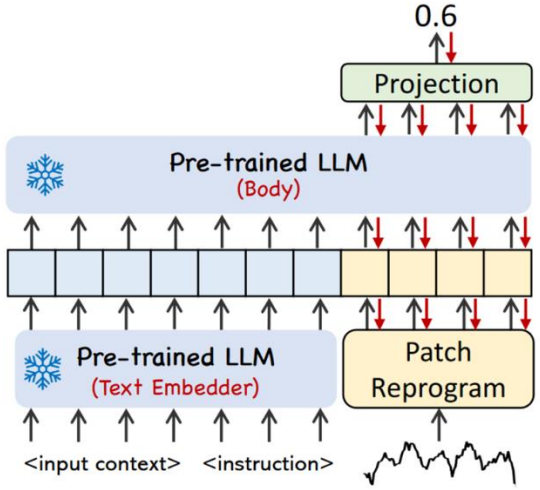




Augment the input with Prompt-as-Prefix

Constrains using "Patch-as-Prefix"

- **LLMs have limited precision** with exact numbers (like time series data) since they are primarily designed for text.
- **They struggle with long-term forecasting**, where precise numerical trends are essential for accuracy.
- **Inconsistent numerical formatting** when generating numbers can lead to misinterpretation of the forecasts.
- For example, a model could output "0.6" as
  - ["0", ".", "6"] or
  - as ["0", ".", "60"], making it tricky to standardize the predictions.

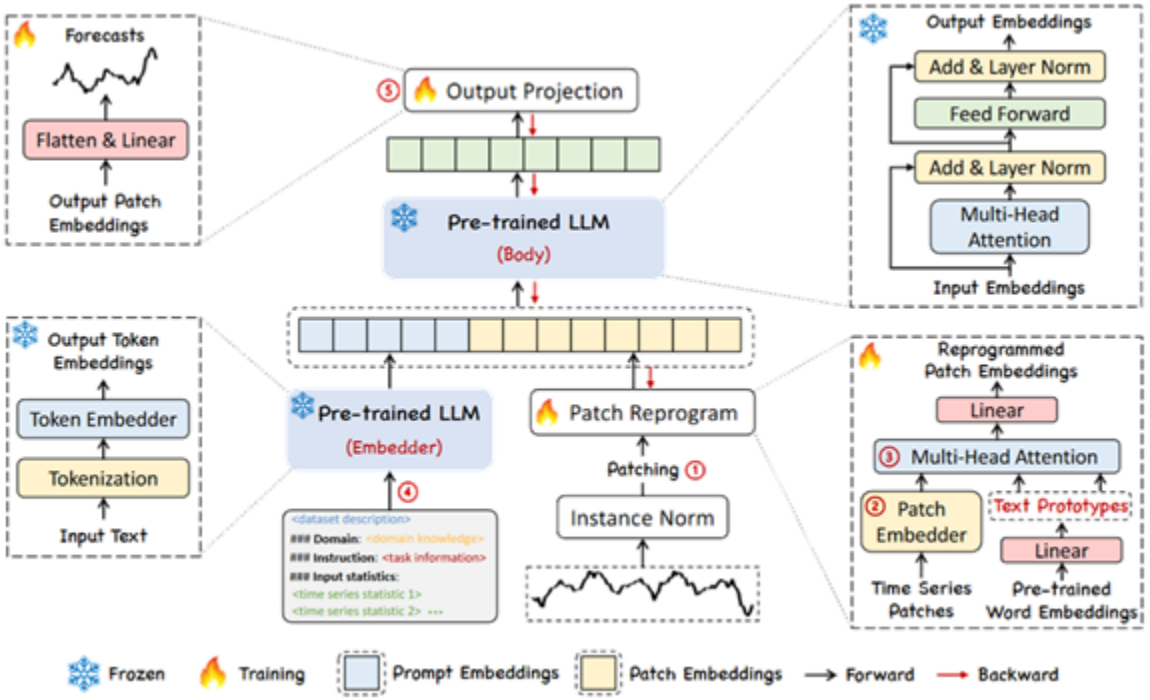


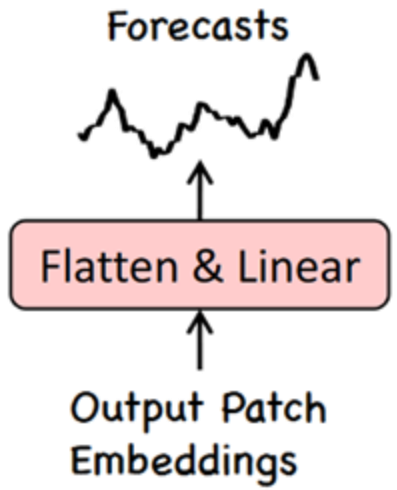
The Electricity Transformer Temperature (ETT) indicates the electric power long-term deployment. Each data point consists of the target oil temperature and 6 power load features ... Below is the information about the input time series:

[BEGIN DATA]
***
[Domain]: We usually observe that electricity consumption peaks at noon, with a significant increase in transformer load
***
[Instruction]: Predict the next <H> steps given the previous <T> steps information attached
***
[Statistics]: The input has a minimum of <min_val>, a maximum of <max_val>, and a median of <median_val>. The overall trend is <upward or downward>. The top five lags are <lag_val>.
[END DATA]

Final Step: Output projection

- Once the prompt prefix and reprogrammed patches are sent to the LLM, it outputs patch embeddings.
- This output must then be flattened and projected linearly to derive the final forecasts, as shown below.





Results Outperform all prior baseline works

- Backbone: Llama-7B
- Long term forecasting
  - Input: 512 length
  - Output horizo, H = {96,192,336,720}
  - Evaluation metric: MAE, MSE
- Short term forecasting
  - Input: 512 length
  - Output horizon, H = {6,48}

Table 6: Ablations on ETTh1 and ETTm1 in predicting 96 and 192 steps ahead (MSE reported). **Red**: the best.

Variant	Long-term Forecasting				Few-shot Forecasting			
	ETTh1-96	ETTh1-192	ETTm1-96	ETThm1-192	ETTh1-96	ETTh1-192	ETTm1-96	ETThm1-192
A.1 Llama (**Default**; 32)	**0.362**	**0.398**	**0.272**	**0.310**	**0.448**	**0.484**	**0.346**	**0.373**
A.2 Llama (8)	0.389	0.412	0.297	0.329	0.567	0.632	0.451	0.490
A.3 GPT-2 (12)	0.385	0.419	0.306	0.332	0.548	0.617	0.447	0.509
A.4 GPT-2 (6)	0.394	0.427	0.311	0.342	0.571	0.640	0.468	0.512
B.1 w/o Patch Reprogramming	0.410	0.412	0.310	0.342	0.498	0.570	0.445	0.487
B.2 w/o Prompt-as-Prefix	0.398	0.423	0.298	0.339	0.521	0.617	0.432	0.481
C.1 w/o Dataset Context	0.402	0.417	0.298	0.331	0.491	0.538	0.392	0.447
C.2 w/o Task Instruction	0.388	0.420	0.285	0.327	0.476	0.529	0.387	0.439
C.3 w/o Statistical Context	0.391	0.419	0.279	0.347	0.483	0.547	0.421	0.461

- Few shot learning (1^{st} 10% predictions vs 1^{st} 5% prediction)
- Zero shot learning (optimized in one dataset then used it in another dataset w/o any prior examples)
  - For both cases: only long term horizon was tested

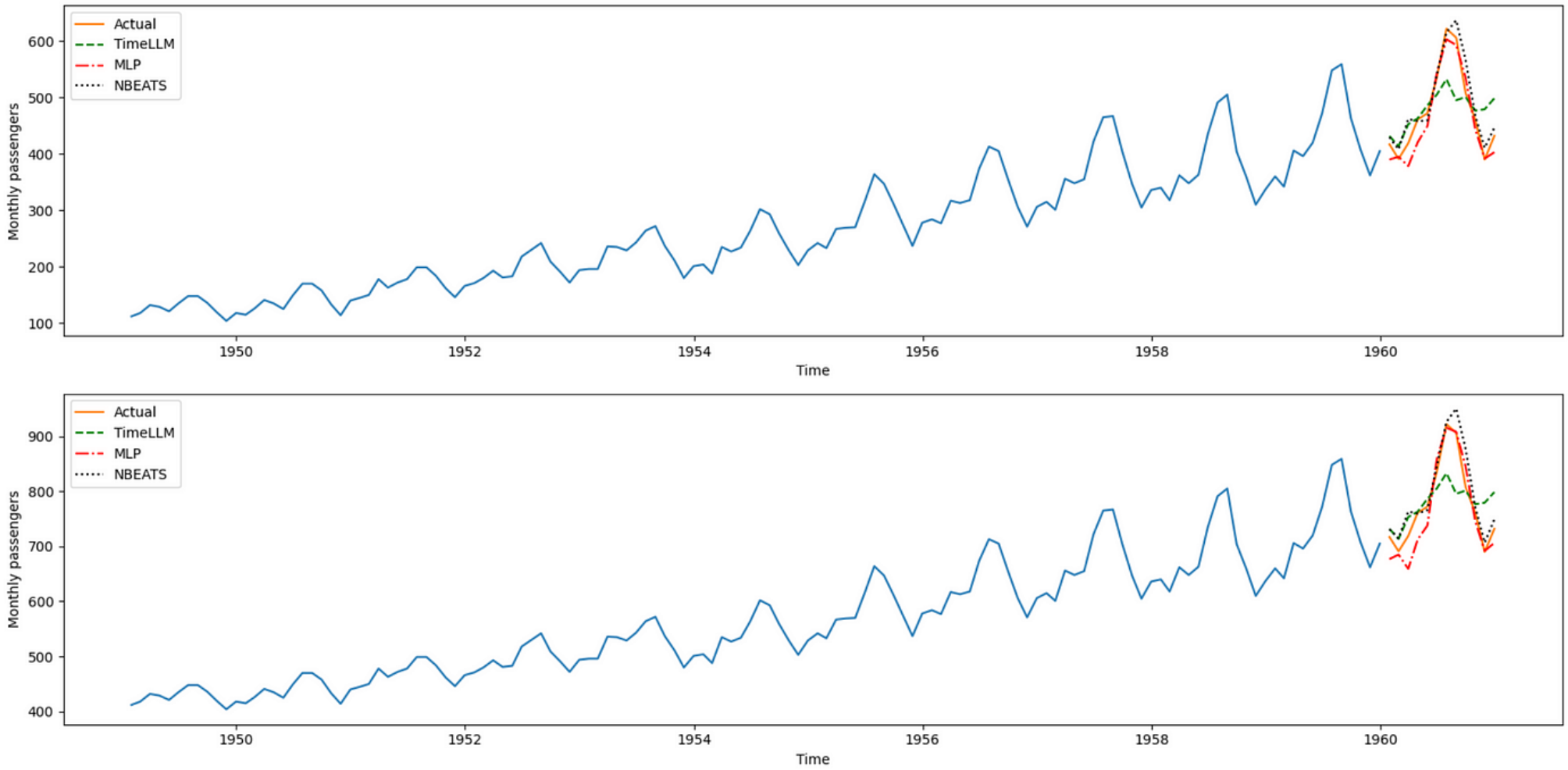
Summary

- An input series is first patched and reprogrammed as a language task.
- Appended a prompt prefix specifying the context of the data, the instructions for the LLM, and input statistics.
- The combined input is sent to the LLM.
- The output embeddings are flattened and projected to generate predictions.

Things to do for better results

- Training the model for longer.
  - I trained for 100 epochs, but the paper uses 1000 epochs.
- Change LLM.
  - I used GPT-2,
  - but *LLaMA*, which was used in the paper, is much better.
- Better prompt engg.
  - My prompt is very minimal, and perhaps we can better engineer it.
- https://colab.research.google.com/drive/1q9PBQrcKeBaHHWWc3f5P2wOcBGH5KzUu#scrollTo=sBQOWsp3-BX_
-

Using MLP and NBEATS



Opinion

- would I use Time-LLM in a forecasting project?
  - Probably not.
- The reality is that Time-LLM requires a lot of computing power and memory. After all, we are working with an LLM.
- In fact, when reproducing the results from the paper using their script, training the model on a single dataset for 1000 epochs takes approximately 19 hours using a GPU!
- Plus, LLMs take a lot of memory space, with billions of parameter usually weighing a few gigabytes for the very large models. In comparison, we can train lightweight deep learning models in a few minutes and get very good forecasts.
- For those reasons, I think that the tradeoff between a possible increase in forecasts accuracy and the computing power and memory storage required to run such model is not worth it.