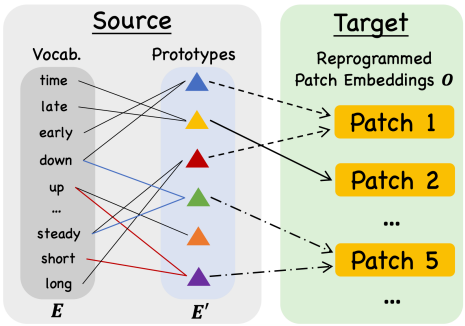
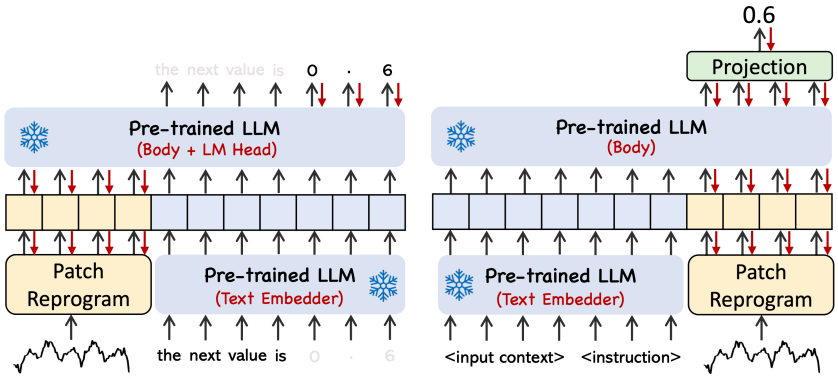
Janghoon Questions

One of the crucial features of TimeLLM is that it does not train or retrain the underlying backbone model. Instead, it trains the input mechanism to convert raw time series data into

Please draw a block diagram for patch embedder in the figure(2). Find out the input shape and output shape of this patch encoder



(a) Patch Reprogramming



(b) Patch-as-Prefix and Prompt-as-Prefix

Input Embedding

each input channel X^i extracted from the time series (the model looks at each uni variate time series channel separately). These channels are first normalized with RevIN. then each channel X^i is divided into **consecutive overlapped or non-overlapped patches** with the total number of input patches equaling:

$$P = \lfloor \frac{T - L_p}{S} \rfloor + 2$$

where S denotes the Stride length. (I don't know where the $+2$ comes from)

Embedded Space

In general we have an input $(n_{channels}, T)$ where n denotes the number of uni variate channels and T represents the number of time steps. Since we have P - patches with length L_p then the sort of internal size of the patch will be (P, L_p).

This is the point where the raw input data is converted to some representation thats an intermediate step to the prototype words.

Output

The patches are then passed into the Patch Embedder that maps each data point at each time step to some **prototype** -> this results in the dimensionality reduction of the raw input data. The output depends on the size of the prototype space (a latent space that represents condensed knowledge and meaningful relationships that the model has learned):

The input is a univariate channel denoted by:

$$X_P^{(i)} \in \mathbb{R}^{P \times L_p}$$

Using the prototypes in the patch embedder, the data is embedded **linearly** into the space of the patch embedder denoted by:

$$\hat{X}_P^{(i)} \in \mathbb{R}^{P \times d_m}$$

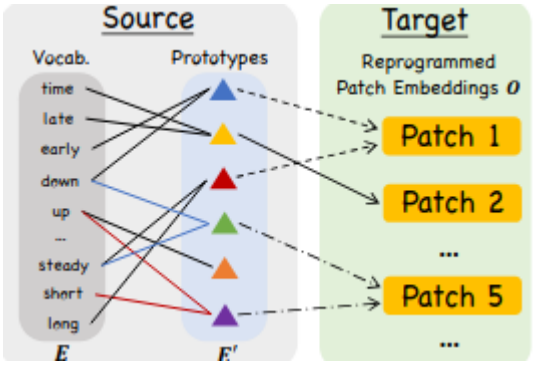
where d_m is the size of the vector after passing through the embedding layer. As I understand it, the embedding layer maps certain inputs to the lower dimension of the prototype space.

In the end, we have a vector output that is a series of patches constructed from some combination of prototypes

Prototypes

These are essentially representations of data in the latent space which is derived from the input tokens. This is how the model condenses the "knowledge"

it gains from the input data. The latent space contains some kinds of abstractions to map terms with similar semantics together into the same prototype?



Try to understand the multi-head attention. What are Q, K, and V?

Reprogrammed Embeddings

A vector of univariate time series data $X_P^{(i)}$ is **reprogrammed** into some representation that the model can understand denoted by

$$\hat{X}_P^{(i)}$$

Vocabulary Dimensionality Reduction

Since the pre-trained model's vocabulary set V is quite large, we take the overall pre-trained word embedding set $E \in \mathbb{R}^{V \times D}$ and create a subset $E' \in \mathbb{R}^{V' \times D}$ where $V' \ll V$ (V' is smaller).

Using linear probing, the model selects **prototypes** that are optimized to better convey the raw data in verbal form to the model.

Multi-Head Attention

this is an approach to designing models that are able to focus on different parts of the input data at the same time - especially from multiple perspectives

(focusing on different attributes and patterns). Each head focuses on a different learned set of attention weights.

The outputs of all the heads are then concatenated and then transformed and passed to the output layer.

Main Idea: each head learns to focus on different patters in the data.

Head Attention Weight Matrices

Each attention head has some trained weight matrix which transforms vectors from one space into its representation in both the key and query space. This is what TimeLLM trains.

Self Attention Matrices

For each Head $k = \{1, \ldots, k\}$, we define the following matrices:

Query Matrix
This matrix represents the mapping of a **specific** patch of raw data into the query space. The goal is to have this query space align as best as possible with the downstream key space which is constructed from the prototypes

$$Q_k^{(i)} = \hat{X}_P^{(i)} W_K^Q$$

Where:

- $\hat{X}_P^{(i)}$ is the reprogramed time series data
- W_K^Q is a learned matrix of weights that will be multiplied by text embedded patches of the raw data $\hat{X}_P^{(i)}$

Key Matrix
This Matrix represents **the mapping of the entire subspace of embedding** E' transformed by their optimized representation in the model space

$$K_k^{(i)} = E' W_k^K$$

where:

- $E' \in \mathbb{R}^{V' \times D} \subseteq E$ is a subspace of all embeddings E

Value Matrix

$$V_K^{(i)} = E' W_K^V$$

- $W_k^V \in \mathbb{R}^{D \times d}$ is of the same dimensionality as the key matrix.
- The value matrix holds the corresponding data that the keys point to.

The value matrices are important because although the model can just use the key and query matrices to determine the importance of parts of the input, we wouldn't be able to map it to the actual output of the model

Attention Function - Softmax

$$Z_K^{(i)} = ATTENTION(Q_k^{(i)}, K_k^{(i)}, V_k^{(i)})^T = SOFTMAX(\frac{Q_k^{(i)} K_k^{(i)}}{\sqrt{d_k}}) V_k^{(i)}$$

Dimensions of weight matrices

- $W_k^K, W_k^V \in \mathbb{R}^{D \times d}$ - where D is the **hidden dimension of the backbone embedding model** and $d = \lfloor \frac{d_m}{K} \rfloor$
- **THE GOAL IS TO TRAIN THE WEIGHT MATRIX W_K^Q SUCH THAT ITS RESULTING QUERY VECTORS ARE ALIGNED ONTO THE KEY SPACE

Softmax produces a normal distribution -> it assigns a weight based on how relevant each key is to each query

Try to find out how outputs with each patch are used for a long sequence to find the output for a long sequence. Did it take

majority decision or averaging from each output?

if we have a long sequence of input data (many time steps), is the output of the patching operation pushed through an averaging function or a majority vote from patches used?

In my understanding, they don't really seem to use either method. Instead the section called *patch reprogramming* states that it uses its concept of **reprogramming**. Using the Query, Key, and Value Matrices the model will assign importance to the inputs based on operations on these objects

Try to find what kind of text will be inserted as input to Patch Program in figure-2. I mean what words or word will be converted to word embeddings?

the raw input data is converted into some specific words from a subset of words that the LLM was trained on - these words capture some kind of information about what the raw data is doing

- "short up"
- "steady down"
- "hold center"
  These word embeddings are then compared to the the prototype space projections to see which prototypes best align with the data in a given patch.

What is the size of the words?

this i dont really know. It seems that we let the model choose the words based on where the projections from the query matrix lands as compared to the

prototypes... the model would choose based on its vocabulary and the where the vector that represents a given word would land

Is linear block applied to each word embedding separately or is it linear layer where multiple word embeddings are input?

the model applies the same weight matrix to all of the word embeddings -> linear layer. This is in step with most other transformer models that do this. Though it might be more accurate to apply the weights over each word embedding, its way more computationally intensive.

In figure-3, patch-as-prefix and prompt as prefix are illustrated. However, while patch as prefix take LLM output directly, prompt as prefix requires additional layer to create the final output. Please find out why they have different architecture just depending on the input composition

patch as prefix doesnt need an extra layer because the data thats being passed into the LLM is already in a verbal format. The can directly operate on the patch as prefix data because it treats it just like any other language input

prompt as prefix requires an additional processing step since the **prompt** requires processing see what its asking for. The combination of the prompt with the actual time series data is used to orient the focus of the model or provide it with some specifc context about the data or maybe some specific aspect that the is trying to be extracted from the data.

1. Verify that first LLM output tensors and Patch Reprogram output are simply concatenated and they are inserted as input to the second LLM as if they are embedding of tokens
2. Think about the network structure if we replace LLM with PLM (pretrained language model) such as BERT