Generalized Prompt Tuning: Adapting Frozen Univariate Time Series Foundation Models for Multivariate Healthcare Time Series

Mingzhu Liu MINGZHUL@ANDREW.CMU.EDU
Angela H. Chen ANGELACHEN@CMU.EDU
George H. Chen GEORGECHEN@CMU.EDU
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Time series foundation models are pre-trained on large datasets and are able to achieve state-of-the-art performance in diverse tasks. However, to date, there has been limited work demonstrating how well these models perform in medical applications, where labeled data can be scarce. Further, we observe that currently, the majority of time series foundation models either are univariate in nature, or assume channel independence, meaning that they handle multivariate time series but do not model how the different variables relate. In this paper, we propose a prompt-tuning-inspired fine-tuning technique, Generalized Prompt Tuning (Gen-P-Tuning), that enables us to adapt an existing univariate time series foundation model (treated as frozen) to handle multivariate time series prediction. Our approach provides a way to combine information across channels (variables) of multivariate time series. We demonstrate the effectiveness of our fine-tuning approach against various baselines on two MIMIC classification tasks, and on influenza-like illness forecasting.

Keywords: time series, foundation models, parameter-efficient fine-tuning

Data and Code Availability We use the MIMIC-III dataset (Johnson et al., 2016) and an influenza-like illness dataset (Wu et al., 2021; Centers for Disease Control and Prevention, 2024), both of which are publicly available. Our code is available at: https://github.com/Ilovecodingforever/Gen-P-Tuning

Institutional Review Board (IRB) This research does not require IRB approval as we perform secondary analyses of publicly available datasets.

1. Introduction

With the rapid development and growing commercial success of large language models (LLMs) recently, there has been a surge of interest in developing similar sorts of foundation models for time series analysis (e.g., Das et al. 2023; Gruver et al. 2024; Goswami et al. 2024). Much like how LLMs are commonly trained on large chunks of the internet spanning many application domains, a number of existing time series foundation models are also trained on a variety of time series data, with the idea that time series across disciplines likely share similar patterns. However, to what extent do such foundation models work for healthcare data such as electronic health records, or public health data such as influenza trends?

For example, electronic health records consist of patient time series, where each time series could be irregularly sampled, have lots of missing entries, and can be of a large number of variables (Xiao et al., 2018). We suspect that this sort of setting is not well-suited for the current time series foundation models that have been developed. In fact, a major limitation of most time series foundation models that have been developed is that they are *univariate* (Ye et al., 2024).

In this paper, our main contribution is to show how to adapt existing *unviariate* time series foundation models to handle *multivariate* time series prediction, specifically for both classification and forecasting. We treat the univariate time series foundation model as frozen, and how we adapt it to handle multivariate time series prediction is as a form of *parameter-efficient fine-tuning* (PEFT). PEFT methods have become popular recently for adapting LLMs to handling various datasets (e.g., Hu et al. 2022). Our proposed PEFT method is a generalization of an existing PEFT method called *prompt tuning*, of which we specifically generalize the variant called *P-tuning v2* by Liu et al. (2022). As such, we call our proposed PEFT method *Generalized Prompt Tuning (Gen-P-Tuning)*. To be clear, the "prompt" here is not a text prompt that a user types. Rather, the prompt for time series forecasting roughly refers to some prefix

that we attach to time series (as if we are including some fictitious extra time steps).

We show that Gen-P-Tuning is competitive in practice against various fine-tuning baselines in experiments on MIMIC in-hospital mortality prediction and phenotyping, and on influenza-like illness forecasting. To the best of our knowledge, our paper is also the first to benchmark various fine-tuning strategies of different time series foundation models on medical datasets. Specifically for our MIMIC classification experiments, we also benchmark against an existing approach called STraTS by Tipirneni and Reddy (2022), which is not a fine-tuning method (all our other baselines are fine-tuning methods).

2. Background

We first set up some basic notation in Section 2.1. We then provide an overview of *univariate* time series foundation models in Section 2.2. We present this overview at a level of detail that is sufficient for understanding our proposed fine-tuning approach, and that is general enough to encompass a number of existing univariate time series foundation models. We then describe ways in which researchers have adapted univariate time series foundation models for *multivariate* time series prediction in Section 2.3. Lastly, we discuss existing work on developing time series foundation models for medical data in Section 2.4.

2.1. Basic Notation

Throughout our paper, it suffices for us to describe how different time series (foundation) models process a single input time series at a time. In practice, for training and for prediction, the models we describe could trivially process a mini-batch of input time series at once rather than a single time series at a time (for training, standard mini-batch gradient descent could be used for learning trainable parameters).

To this end, we denote a single *multivariate* input time series as $\mathbf{X} \in \mathbb{R}^{C \times T}$, where C is the number of channels/variables, and T is the number of time steps. The multivariate case corresponds to $C \geq 2$, whereas in the univariate case ($C = 1$), we represent the time series as a 1D array of length T rather than a 2D array of shape C-by-T. Importantly, for any dataset of interest, we assume that the number of channels C is fixed. Across different datasets, it is possible for C to vary. Meanwhile, the number of time steps T can vary across time series even within the same dataset.

We denote the ground truth prediction target as $\mathbf{Y}$. In this paper, we specifically consider two prediction tasks. First, if the prediction task is classification with $M \geq 2$ classes, then $\mathbf{Y} \in [0, 1]^M$ (the i-th entry of $\mathbf{Y}$ indicates the probability of the i-th class). Second, if the prediction task is forecasting, then $\mathbf{Y} \in \mathbb{R}^{C \times H}$, where H is the time horizon (i.e., the number of future time steps of the input time series that we aim to predict); in the univariate case, we could of course just represent $\mathbf{Y}$ as a 1D array of length H.

2.2. Univariate Time Series Foundation Models

We now state the general form of the univariate foundation models that our proposed fine-tuning strategy can adapt into multivariate time series predictors. Special cases include, for instance, MOMENT (Goswami et al., 2024), GPT4TS (Zhou et al., 2023), and TimesFM (Das et al., 2023).

For an input *univariate* time series $\mathbf{X}^{\text{uni}} \in \mathbb{R}^T$ where T is the number of time steps (since this pretrained foundation model is univariate, there is only a single channel), we assume that the foundation model first applies a preprocessing function f_{emb} to $\mathbf{X}^{\text{uni}}$ to produce a preprocessed array

$$\mathbf{E}^{\text{uni}} = f_{\text{emb}}(\mathbf{X}^{\text{uni}}) \in \mathbb{R}^{P \times D}, \tag{1}$$

where P is the number of patches (commonly, time series foundation models convert the input time series into patches, so that a raw input time series with T time steps gets treated instead as a sequence of P patches (Nie et al., 2023)) and D is the embedding dimension. We leave precisely what preprocessing steps are included in the function f_{emb} unspecified; in practice, this would just depend on whichever univariate foundation model is used.[1]

After preprocessing the raw univariate time series data $\mathbf{X}^{\text{uni}} \in \mathbb{R}^T$ using f_{emb} to produce $\mathbf{E}^{\text{uni}} \in \mathbb{R}^{P \times D}$, the backbone of the univariate foundation model then applies a sequence of transformer layers one after another. The output from one layer becomes the

1. For example, prior to chunking the time series into patches, an additional preprocessing step that could be done is reversible instance normalization (Kim et al., 2021), which standardizes a time series by its mean and variance. An alternative altogether is to preprocess the time series to appear as text data, so that subsequent steps amount to using a large language model (Gruver et al., 2024).

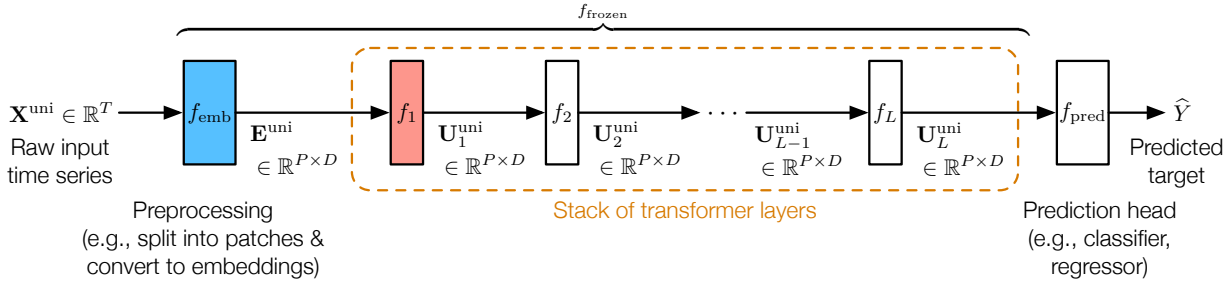

Figure 1: Univariate time series foundation model. The predicted target's dimensions depend on the prediction task ($\widehat{Y} \in [0,1]^M$ for M-way classification, and $\widehat{Y} \in \mathbb{R}^H$ for forecasting the next H time steps). Note that the preprocessing function f_{emb} (in blue) and the first transformer layer f_1 (in pink) are color-coded for ease of exposition as they reappear later on in Figure 2.

input to the next layer, for layers 1 to L:

$$\begin{aligned}
\mathbf{U}_1^{\text{uni}} &\triangleq f_1(\mathbf{E}^{\text{uni}}) \in \mathbb{R}^{P\times D},\\
\mathbf{U}_2^{\text{uni}} &\triangleq f_2(\mathbf{U}_1^{\text{uni}}) \in \mathbb{R}^{P\times D},\\
&\vdots\\
\mathbf{U}_L^{\text{uni}} &\triangleq f_L(\mathbf{U}_{L-1}^{\text{uni}}) \in \mathbb{R}^{P\times D},
\end{aligned}$$

where f_ℓ corresponds to the ℓ-th transformer layer, for $\ell = 1, \ldots, L$.

Finally, a prediction layer f_{pred} (sometimes also called a prediction "head") is applied to the last transformer layer's output $\mathbf{U}_L^{\text{uni}}$ to produce the final prediction:

$$\widehat{\mathbf{Y}} = f_{\text{pred}}(\mathbf{U}_L^{\text{uni}}),$$

where the dimensions of $\widehat{\mathbf{Y}}$ depend on whether we are considering classification or forecasting as the prediction task, as described in Section 2.1.

In summary, the full univariate foundation model could be represented as the function

$$f \triangleq f_{\text{pred}} \circ \underbrace{(f_L \circ f_{L-1} \circ \cdots \circ f_1 \circ f_{\text{emb}})}_{\triangleq f_{\text{frozen}}}, \qquad (2)$$

so that $\widehat{\mathbf{Y}} = f(\mathbf{X}^{\text{uni}}) = f_{\text{pred}}\big(f_{\text{frozen}}(\mathbf{X}^{\text{uni}})\big)$. This entire process of how raw input time series $\mathbf{X}^{\text{uni}} \in \mathbb{R}^T$ is turned into the final predicted output $\widehat{\mathbf{Y}}$ by the univariate foundation model f is depicted in Figure 1.

Our notation in equation (2) emphasizes that we crucially view all the layers prior to the prediction head as frozen—we will not update parameters of f_{frozen}. The reason we exclude the prediction head f_{pred} from f_{frozen} is that when fine-tuning to other datasets, we typically discard the original prediction head f_{pred} of the foundation model and instead use a new prediction head with trainable parameters.

Generalizing beyond transformers Our assumption of the univariate foundation model using repeated transformer layers is mainly because the univariate foundation models we use later in our experiments are transformer-based. However, as it will be apparent when we describe our proposed fine-tuning strategy, the repeated transformer layers could be replaced by some other architecture altogether (i.e., the use of transformers is not actually necessary). This is similar in spirit to how the now-standard Low-Rank Adaptation (LoRA) fine-tuning method (Hu et al., 2022) is also commonly applied to transformer-based architectures although the key idea of LoRA does not require the use of transformers.

2.3. Handling Multivariate Time Series With Channel Independence

A trivial way of adapting a univariate foundation model for multivariate time series prediction is to assume independence across channels (e.g., Nie et al. 2023). Given an input multivariate time series $\mathbf{X} \in \mathbb{R}^{C\times T}$, we separate it out into its different channels' time series $\mathbf{X}_{(1)}, \mathbf{X}_{(2)}, \ldots, \mathbf{X}_{(C)} \in \mathbb{R}^T$. Then we can obtain the final transformer layer's output across the different channels:

$$\mathbf{U}_L^{(c)} \triangleq f_{\text{frozen}}(\mathbf{X}_{(c)}) \qquad \text{for } c = 1, \ldots, C. \qquad (3)$$

Finally, assuming that we have training data for the multivariate time series prediction task, we train a prediction head of our choosing that takes in $\mathbf{U}_L^{(1)}, \ldots, \mathbf{U}_L^{(C)}$ as the inputs (or the average of these across channels as a single input) and outputs the predicted target. As an example, to forecast the next H time steps, we could set the prediction head to be a multilayer perceptron that takes in $\frac{1}{C}\sum_{c=1}^{C} \mathbf{U}_L^{(c)}$ as

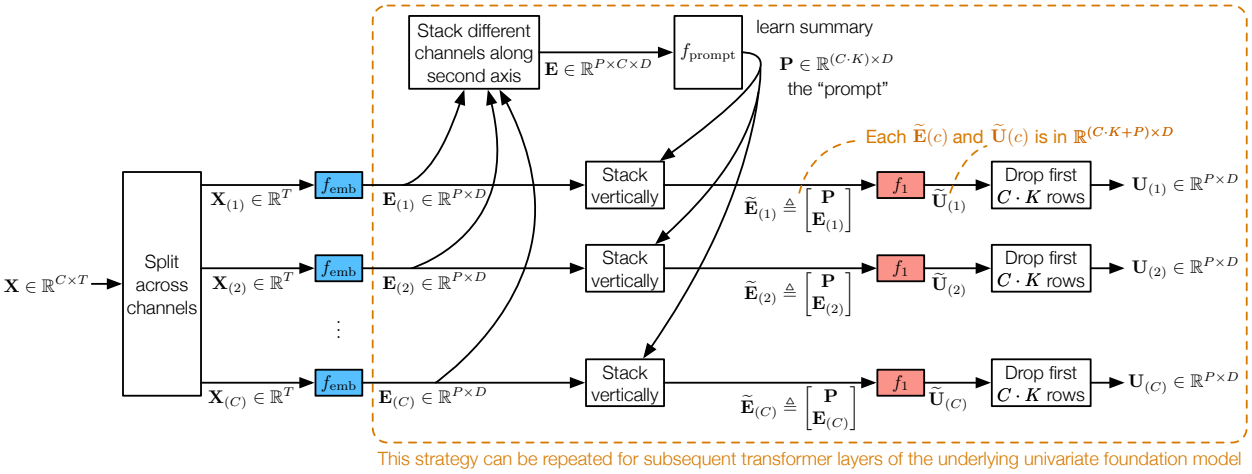

Figure 2: Overview of how the prompt is added to the backbone. Note that there is no subscript in the prompt. The weights are shared across different transformer layers.

input, and outputs a total of $C \cdot H$ numbers (reshaped to be C-by-H); training can proceed via standard minibatch gradient descent with MSE loss.

2.4. Time Series Foundation Models for Medical Data

There has not been extensive work on applications of time series foundation models on medical data, especially longitudinal patient data. Time series foundation models are relatively new (compared to LLMs), and the majority of deep learning methods for patient data have been focusing on non-pretrained models such as LSTMs and GRUs (Pham et al., 2016; Baytas et al., 2017; Che et al., 2017; Baytas et al., 2017).

There is a line of work that aims to build transformer-based foundation models that are pre-trained on EHR data and are able to perform tasks using EHR data. Commonly, the data is encoded as a sequence of tokens, and are used as inputs to a language model. The tokens typically contain a tuple in the form of (timestamp, features) (Yang et al., 2023; Antikainen et al., 2023; McDermott et al., 2023). For example, CEHR-BERT (Pang et al., 2021) encodes the event at each timestamp as (time, age, clinical concept name), passes them through embedding layers, and feeds it to a BERT (pre-trained language) model. Some models are pre-trained on a large cohort of patient data (Yang et al., 2023; Kraljevic et al., 2024). However, in order to protect patient privacy, they typically do not release the pre-trained models or data, as opposed to time series foundation models which are typically open-source.

3. Method

We now explain how our Gen-P-Tuning method works for adapting a pre-trained univariate time series foundation model for multivariate time series prediction. We treat the univariate foundation model as frozen, and our fine-tuning approach introduces trainable elements. We provide an overview first in Section 3.1. In this overview, a key component is called a *Prompt Module*, which contains all the trainable elements aside from what is in the final prediction head. We explain the Prompt Module in more detail in Section 3.2.

3.1. Overview

We give an overview of Gen-P-Tuning first, with an accompanying diagram explaining the high-level steps in Figure 2. For ease of exposition, it suffices to explain what happens when the univariate foundation model only uses a single transformer layer (f_1 that is shaded in pink in Figure 1). Gen-P-Tuning processes a single multivariate input time series $\mathbf{X} \in \mathbb{R}^{C \times T}$ as follows:

1. We separate input $\mathbf{X} \in \mathbb{R}^{C \times T}$ into its different channels' time series $\mathbf{X}_{(1)}, \mathbf{X}_{(2)}, \ldots, \mathbf{X}_{(C)} \in \mathbb{R}^T$. This step is shown in the leftmost part of Figure 2.
2. For each channel $c = 1, \ldots, C$, we preprocess $\mathbf{X}_{(c)} \in \mathbb{R}^T$ using the univariate foundation model's preprocessing function f_{emb} to obtain the 2D arrays $\mathbf{E}_{(1)}, \ldots, \mathbf{E}_{(C)} \in \mathbb{R}^{P \times D}$. This step is shown using the blue boxes of Figure 2 (note that each blue box is the same as the blue box from Figure 1).

3. The 2D arrays $\mathbf{E}_{(1)}, \ldots, \mathbf{E}_{(C)} \in \mathbb{R}^{P \times D}$ are stacked together treating channels as the second axis to produce a 3D array $\mathbf{E} \in \mathbb{R}^{P \times C \times D}$, as shown in the top box labeled "Stack different channels along second axis" in Figure 2.
4. We now introduce a trainable neural network f_{prompt} that takes the 3D array $\mathbf{E} \in \mathbb{R}^{P \times C \times D}$ as input, and outputs a summary 2D array $\mathbf{P} \in \mathbb{R}^{(C \cdot K) \times D}$ that we refer to as the "prompt"; here K is a user-specified hyperparameter that controls the prompt size. Importantly, the prompt $\mathbf{P}$ should encode summary information *across* channels.

   We refer to the function f_{prompt} as the Prompt Module. We provide the architecture we use for the Prompt Module in Section 3.2.
5. Next, the prompt $\mathbf{P} \in \mathbb{R}^{(C \cdot K) \times D}$ is "attached" to each channel c's preprocessed array $\mathbf{E}_{(c)}$ by vertically stacking $\mathbf{P}$ and $\mathbf{E}_{(c)}$ to produce
$$\widetilde{\mathbf{E}}_{(c)} \triangleq \begin{bmatrix} \mathbf{P} \\ \mathbf{E}_{(c)} \end{bmatrix} \in \mathbb{R}^{(C \cdot K + P) \times D}.$$
   Roughly this could be thought of as adding a prefix of $C \cdot K$ fictitious patches prior to the actual P patches of $\mathbf{E}_{(c)}$. This step corresponds to the boxes labeled "Stack vertically" in Figure 2.
6. For each channel $c = 1, \ldots, C$, we feed each array $\widetilde{\mathbf{E}}_{(c)} \in \mathbb{R}^{(C \cdot K + P) \times D}$ as input to the univariate foundation model's first transformer layer f_1 to produce the output $\widetilde{\mathbf{U}}_{(c)} \in \mathbb{R}^{(C \cdot K + P) \times D}$. This step corresponds to the pink boxes in Figure 2.
7. Note that the first $C \cdot K$ rows of $\widetilde{\mathbf{U}}_{(c)} \in \mathbb{R}^{(C \cdot K + P) \times D}$ correspond to the fictitious patches added in step 5. We now remove these rows as to obtain the output $\mathbf{U}(c) \in \mathbb{R}^{P \times D}$, as depicted in the boxes labeled "Drop first $C \cdot K$ rows" in Figure 2.

   Note that $\mathbf{U}_{(1)}, \ldots, \mathbf{U}_{(C)} \in \mathbb{R}^{P \times D}$ could be viewed as the multivariate output produced using the univariate transformer layer f_1 with the help of our Gen-P-Tuning strategy.

If the univariate foundation model has more than one transformer layer, then steps 3–7 could be repeated for the subsequent transformer layers. Supposing for the moment that there is only 1 transformer layer, then after obtaining $\mathbf{U}_{(1)}, \ldots, \mathbf{U}_{(C)} \in \mathbb{R}^{P \times D}$ from step 7, we would simply train a prediction head in the same manner as stated in Section 2.3, when we covered handling multivariate time series prediction with channel independence. Conceptually, each input to the transformer layer f_1 is now provided with information across channels since the prompt $\mathbf{P}$ summarizes information across channels.

We point out two special cases of our approach:

- (Standard prompt tuning) If the Prompt Module f_{prompt} is defined to not actually depend on $\mathbf{E}$ and instead just output an array of $(C \cdot K)$-by-D numbers that are all treated as trainable parameters, then we recover a popular prompt tuning strategy called P-tuning v2 (Liu et al., 2022). This is the main reason we refer to our strategy as Generalized Prompt Tuning.
- (Channel independence) In the degenerate case where the prompt size hyperparameter $K = 0$, so that effectively we do not attach any prefix/prompt $\mathbf{P}$ to each channel's preprocessed array $\mathbf{E}_{(c)}$, then we just recover the same idea as the channel independent strategy of Section 2.3.

3.2. The Prompt Module f_{prompt}

As our overview above indicates, the Prompt Module f_{prompt}'s main goal is to summarize the different preprocessed time series across channels (a total of $P \times C \times D$ numbers) into a single array $\mathbf{P} \in \mathbb{R}^{(C \cdot K) \times D}$ that notably does not depend on the number of patches P. In particular, f_{prompt} needs to accommodate the possibility that different input time series even within the same dataset could have different numbers of patches P (and across different datasets, the number of channels C could vary).

There are many ways to define f_{prompt}. We specifically define it to do the following steps:

1. We apply a transformer module to $\mathbf{E} \in \mathbb{R}^{P \times C \times D}$ by treating the P different patches as if they are different data points (so that each "data point" is in $\mathbb{R}^{C \times D}$, where C is treated as the "time steps" by the transformer module). The output per patch is in $\mathbb{R}^{C \times D}$, and we stack these outputs into a single array $\mathbf{P}' \in \mathbb{R}^{P \times C \times D}$.
2. We use a transformer to map $\mathbf{P}'$ from $\mathbb{R}^{P \times C \times D}$ to $\mathbb{R}^{P \times C \times (K \cdot D)}$, followed by a max pooling operation to map it from $\mathbb{R}^{P \times C \times (K \cdot D)}$ to $\mathbb{R}^{C \times (K \cdot D)}$, and finally a reshape operation to map it from $\mathbb{R}^{C \times (K \cdot D)}$ to $\mathbb{R}^{(K \cdot C) \times D}$. Note that there are other ways of aggregating across the P dimension (to make the output of the Prompt Module not depend on the number of patches). We explain how

to instead use an RNN or an MLP in Appendix B (which includes experimental results with these alternative architectures).

4. Experiments

Our experiments aim to show how different fine-tuning strategies for adapting univariate time series foundation models to multivariate time series classification and forecasting work in practice on clinical data and also on a public health dataset. To this end, we specifically consider univariate time series foundation models, MOMENT (Goswami et al., 2024) and GPT4TS (Zhou et al., 2023) that support both classification and forecasting, and both are special cases of the formulation we presented in Section 2.2.[2] Moreover, we run an experiment to study the effect of increasing the prompt size hyperparameter K, and also point out differences in runtime and the number of parameters used by the different fine-tuning methods.

Datasets Classification experiments are performed on MIMIC-III (Johnson et al., 2016), which is a publicly available electronic health records dataset collected from patients in the intensive care units of the Beth Israel Deaconess Medical Center from 2001 to 2012. We follow the benchmark proposed by Harutyunyan et al. (2019). Specifically, we focus on two tasks. The first is a binary classification task of predicting in-hospital mortality based on the first 48 hours of an ICU stay (referred to as "MIMIC Mortality" in our tables later). The second is a multi-class, multi-label classification task, where we classify which of 25 acute care conditions occurred in an ICU stay ("MIMIC Phenotyping").

For these two MIMIC classification tasks, to simulate a resource-constrained environment, we use only 1000 randomly sampled patients, 60% of the data for training, 10% for validation, and 30% for testing. In the original benchmark, 17 clinical variables are included. We also include as an additional variable the number of hours since the time of admission to the ICU. Instead of one-hot encoding as in the benchmark, we encode categorical variables as ordinal values. The time series are irregularly sampled, and missing values are imputed using forward filling when possible or using the normal values suggested by the benchmark otherwise.

Forecasting experiments are performed on an influenza-like illness dataset (Wu et al., 2021; Centers for Disease Control and Prevention, 2024). This is a weekly sampled dataset with 7 channels/variables. The dataset is split into 60% training, 10% validation, and 30% testing. The forecasting horizon is 60 weeks. We forecast all 7 variables in the dataset.[3]

Baselines We compare Gen-P-Tuning to the following fine-tuning baselines:

1. Full fine-tuning: all parameters are updated.
2. LoRA (Hu et al., 2022): this baseline keeps the pre-trained weight matrices for the transformer blocks frozen but allows for each block to be "updated" by a low-rank matrix in the following manner. Let W denote the weight matrix for a specific transformer block. Then we simply replace W with $W_{\text{new}} = W + AB$, where matrices A and B are low rank. LoRA fine-tuning corresponds to leaving the original W fixed and only learning the values in A and B.
3. Linear probing: this baseline is precisely the channel-independent strategy mentioned in Section 2.3. For MOMENT, the only learnable part is now the prediction head. For GPT4TS, this includes the input embedding layer (since it is not pre-trained) and the prediction head.
4. P-tuning v2 (referred to simply as "Prompt Tuning" in our tables later), where $\mathbf{P}$ is an array of trainable parameters.

As we already stated at the end of Section 3.1, the channel-independent strategy (linear probing) and standard prompt tuning (P-tuning v2) could be viewed as special cases of our Generalized Prompt Tuning approach. However, in our experiments to follow, for simplicity, the results we show for Gen-P-Tuning are specific to when we define the Prompt Module (Section 3.2) in a nontrivial manner so that it does not simply reduce to either linear probing or P-tuning v2. In practice, one could of course tune our Gen-P-Tuning strategy (e.g., based on a validation set evaluation metric) to choose between a nontrivial Prompt Module, a trivial Prompt Module that just

2. At a high-level, the major difference between these two foundation models is that MOMENT is based on T5 (Raffel et al., 2020) whereas GPT4TS is based on GPT2 (Radford et al., 2019).

3. We point out that Goswami et al. (2024) also presented forecasting results on this dataset but they report experimental results only on forecasting one of the 7 variables ("OT"), so the numbers they get are not directly comparable to ours.

Table 1: MIMIC Mortality test set scores (mean ± std. dev. over 5 random seeds). For each univariate foundation model, per column we bold whichever score is highest and underline the second-best score. Note that STraTS is not a univariate foundation model that we fine-tune, so it is provided as a non-foundation-model baseline.

Model	Fine-Tuning Method	Raw Accuracy	AUROC	F1	AUPRC
MOMENT	Full	**0.891** ± 0.012	0.687 ± 0.020	0.508 ± 0.044	0.255 ± 0.038
	LoRA	0.875 ± 0.021	0.720 ± 0.019	0.573 ± 0.050	0.272 ± 0.025
	Linear Probing	0.878 ± 0.013	<u>0.730</u> ± 0.035	0.544 ± 0.043	0.260 ± 0.018
	Prompt Tuning	<u>0.883</u> ± 0.012	0.724 ± 0.035	<u>0.576</u> ± 0.058	<u>0.274</u> ± 0.020
	Gen-P-Tuning	0.881 ± 0.005	**0.754** ± 0.021	**0.591** ± 0.031	**0.292** ± 0.026
GPT4TS	Full	0.886 ± 0.019	**0.743** ± 0.018	0.524 ± 0.052	**0.309** ± 0.023
	LoRA	0.871 ± 0.017	0.708 ± 0.056	**0.588** ± 0.028	0.254 ± 0.024
	Linear Probing	0.859 ± 0.015	<u>0.737</u> ± 0.033	<u>0.584</u> ± 0.037	<u>0.265</u> ± 0.037
	Prompt Tuning	**0.891** ± 0.013	0.689 ± 0.062	0.471 ± 0.004	0.236 ± 0.022
	Gen-P-Tuning	<u>0.887</u> ± 0.016	0.708 ± 0.025	0.499 ± 0.033	0.255 ± 0.038
STraTS		0.900 ± 0.000	0.601 ± 0.039	0.474 ± 0.000	0.159 ± 0.039

outputs a trainable array of numbers (resulting in P-tuning v2), or no Prompt Module (resulting in linear probing).

For MIMIC experiments, we also include a baseline that is not a fine-tuning approach. Specifically, we use STraTS (Tipirneni and Reddy, 2022), a transformer-based model developed on MIMIC-III that applies self-supervised pretraining (forecasting the values in the next two hours). It encodes EHR data as a sequence of triplets (timestamp, variable name, variable value). We pretrain the model using 1000 random samples from the benchmark, and then perform the classification tasks using the same samples.

Evaluation metrics We use accuracy, area under the receiver operating characteristic curve (AUC), F1, and area under the precision-recall curve (AUPRC) for classification tasks. The macro variant of each metric is used for MIMIC Phenotyping.[4] We use mean squared error (MSE) and mean absolute error (MAE) for the forecasting task.

Main benchmark findings We present MIMIC mortality classification performance in Table 1, MIMIC phenotype classification performance in Table 2, and influenza-like illness forecasting performance in Table 3.

4. In the classification tasks, we point out that raw accuracy alone is not a representative metric since there is a class imbalance in MIMIC-III tasks (negative to positive ratio is 7:1 for mortality prediction, and 5:1 on average for phenotype classification).

There is no method that always performs the best. Gen-P-Tuning is often among the best-performing ones. Linear probing sometimes performs very well, which suggests that a channel-independent strategy is sometimes sufficient. Indeed, some existing work has demonstrated that channel-independence sometimes performs well. For example, PatchTST (Nie et al., 2023), which is a channel-independent method, has been shown to outperform "channel-mixing" where channels are concatenated before being fed into the model.

Full fine-tuning sometimes does not perform well, especially for forecasting experiments. This might be a sign of catastrophic forgetting (Goodfellow et al., 2014), which is commonly observed in low-data regimes with deep networks. There has not been previous work on catastrophic forgetting of time series foundation models, but it has been observed in models such as LSTMs (Schak and Gepperth, 2019).

As a reminder, linear probing and standard prompt tuning can be thought of as special cases of Generalized Prompt Tuning, where we set hyperparameters differently. We can choose between these special cases by whichever one achieves the best validation loss. As an illustration of this, for MIMIC mortality prediction, using the MOMENT foundation model, we report the validation losses of linear probing, standard prompt tuning, and Gen-P-Tuning (with a nontrivial Prompt Module) in Table 4. In this case, Gen-P-Tuning achieves the lowest validation loss, and on the true test set, it does indeed outperform the simpler special cases.

Table 2: MIMIC Phenotyping test set scores (mean ± std. dev. over 5 random seeds). This table uses the same formatting as Table 1 (in terms of what bolding and underlining mean).

Model	Fine-Tuning Method	Raw Accuracy	AUROC	F1	AUPRC
MOMENT	Full	0.832 ± 0.007	0.643 ± 0.019	0.070 ± 0.024	0.276 ± 0.021
	LoRA	0.832 ± 0.007	0.640 ± 0.025	0.085 ± 0.023	0.273 ± 0.027
	Linear Probing	0.830 ± 0.006	0.631 ± 0.026	0.071 ± 0.031	0.264 ± 0.022
	Prompt Tuning	0.832 ± 0.008	0.634 ± 0.012	0.069 ± 0.036	0.268 ± 0.015
	Gen-P-Tuning	**0.835** ± 0.004	**0.666** ± 0.015	**0.135** ± 0.017	**0.294** ± 0.012
GPT4TS	Full	0.823 ± 0.009	0.593 ± 0.014	0.060 ± 0.028	0.234 ± 0.014
	LoRA	0.801 ± 0.015	0.596 ± 0.023	0.107 ± 0.024	**0.241** ± 0.015
	Linear Probing	0.789 ± 0.009	0.555 ± 0.016	**0.129** ± 0.029	0.213 ± 0.015
	Prompt Tuning	0.831 ± 0.010	0.581 ± 0.012	0.024 ± 0.017	0.227 ± 0.014
	Gen-P-Tuning	**0.832** ± 0.003	**0.599** ± 0.010	0.020 ± 0.009	0.231 ± 0.010
STraTS		0.835 ± 0.008	0.573 ± 0.020	0.000 ± 0.000	0.217 ± 0.016

Table 3: Influenza-like illness forecasting test set scores (mean ± std. dev. over 5 random seeds). For each univariate foundation model, per column we bold whichever score is best and underline the second-best score.

Model	Fine-Tuning Method	MSE	MAE
MOMENT	Full	3.199 ± 0.102	1.262 ± 0.022
	LoRA	3.109 ± 0.021	1.176 ± 0.006
	Linear Probing	**2.622** ± 0.036	**1.159** ± 0.011
	Prompt Tuning	2.918 ± 0.047	1.200 ± 0.009
	Gen-P-Tuning	3.083 ± 0.080	1.200 ± 0.016
GPT4TS	Full	3.219 ± 0.093	1.240 ± 0.012
	LoRA	3.247 ± 0.337	1.230 ± 0.089
	Linear Probing	3.202 ± 0.303	1.232 ± 0.072
	Prompt Tuning	3.105 ± 0.429	1.253 ± 0.098
	Gen-P-Tuning	**2.939** ± 0.378	**1.209** ± 0.092

Table 4: Validation loss of linear probing, prompt tuning, and Gen-P-Tuning on MOMENT mortality prediction task (mean ± std. dev. over 5 random seeds).

Fine-Tuning Method	Validation Loss
Linear Probing	0.396 ± 0.094
Prompt Tuning	0.397 ± 0.100
Gen-P-Tuning	0.347 ± 0.064

Prompt size We demonstrate the effect of increasing the prompt size on the MIMIC-III mortality prediction task (Table 5). For this task, performance generally becomes better as the prompt size increases.

Computational efficiency To give a sense of the number of trainable parameters for the different fine-tuning methods, we report these numbers specifically for the MIMIC-III mortality prediction in Table 6.

Runtime We present the runtime in seconds in Table 7. The main reason why both prompt tuning and Gen-P-Tuning can take more time than full fine-tuning, LoRA, and linear probing is that the Prompt Module needs to be trained and it depends on all channels. Furthermore, the backbones of MOMENT and GPT4TS are implemented in Huggingface as is the LoRA fine-tuning method. It is possible that Huggingface's implementation of these are optimized for faster training, whereas we did not prioritize compu-

Table 5: MOMENT Mortality test set scores (mean ± std. dev. over 5 random seeds) with varying prompt size. As a reminder, larger prompt size hyperparameter K corresponds to a larger prompt size.

Prompt size hyperparameter K	Raw Accuracy	AUROC	F1	AUPRC
1	0.879 ± 0.011	0.759 ± 0.019	0.574 ± 0.015	0.293 ± 0.037
2	0.871 ± 0.007	0.748 ± 0.033	0.593 ± 0.036	0.297 ± 0.056
4	0.881 ± 0.005	0.754 ± 0.021	0.591 ± 0.031	0.292 ± 0.026

Table 6: Number of trainable parameters in mortality prediction experiments.

Model	Full fine-tune	LoRA	Linear Probing	Prompt Tuning	Gen-P-Tuning
MOMENT	341,651,993 (100%)	666,649 (0.2%)	411,673 (0.1%)	685,465 (0.2%)	619,777 (0.2%)
GPT4TS	60,985,345 (100%)	1,132,033 (1.9%)	139,777 (0.2%)	1,270,881 (2.1%)	1,646,593 (2.7%)

Table 7: Runtime in seconds of all fine-tuning strategies (one experimental repeat).

Model	Experiment	Full fine-tune	LoRA	Linear Probing	Prompt Tuning	Gen-P-Tuning
MOMENT	MIMIC Mortality	1218	1025	418	706	3317
	MIMIC Phenotyping	1287	1084	421	697	3507
	Forecasting	46	42	32	38	70
GPT4TS	MIMIC Mortality	130	119	107	430	913
	MIMIC Phenotyping	137	124	112	432	943
	Forecasting	7	7	6	29	43

tational efficiency in our implementations of Prompt Tuning and Gen-P-Tuning.

5. Discussion

In this paper, we demonstrate the applicability of fine-tuning methods to time series foundation models on disease surveillance and electronic health records data. Moreover, we propose a prompt-tuning method that fine-tunes univariate time series foundation models for multivariate time series classification and forecasting by adding a Prompt Module that combines information across channels.

We highlight some limitations of our work that in turn point toward directions of future research. First, we only evaluated the performance of models on two datasets, with only a fairly limited collection of feature types. The influenza-like illness dataset already resembles many other time series forecasting tasks so that we would expect time series foundation models to work well for it. As for MIMIC, we reused the setup by Harutyunyan et al. (2019), which only considers a relatively small set of features. There are other modalities common in healthcare, such as waveforms, which are recorded more frequently and thus have a longer context available. Importantly, healthcare time series also routinely consist of categorical variables that change over time. We suspect that time-varying categorical variables do not closely resemble the sort of time series that time series foundation models are typically trained on. However, we have not investigated how fine-tuning time series foundation models copes with these categorical variables.

A separate direction that we have not explored is interpreting how the learned Prompt Module combines information across channels. While attention weights of the Prompt Module could be visualized, there is debate on whether attention weights are interpretable (e.g., Serrano and Smith 2019). A direction that could be promising is to synthetically combine univariate time series to form multivariate time series (so we know how channels are combined). We can then check whether the Prompt Module can recover the ground truth channel mixing strategy.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback. G. H. Chen is supported by NSF CAREER award #2047981.

References

Emmi Antikainen, Joonas Linnosmaa, Adil Umer, Niku Oksala, Markku Eskola, Mark van Gils, Jussi Hernesniemi, and Moncef Gabbouj. Transformers for cardiac patient mortality risk prediction from heterogeneous electronic health records. *Scientific Reports*, 13(3517), 2023.

Inci M. Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K. Jain, and Jiayu Zhou. Patient subtyping via time-aware LSTM networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 65–74, 2017.

Centers for Disease Control and Prevention. FluView: Flu activity & surveillance, 2024. https://gis.cdc.gov/grasp/fluview/ fluportaldashboard.html.

Chao Che, Cao Xiao, Jian Liang, Bo Jin, Jiayu Zho, and Fei Wang. An RNN architecture with dynamic temporal matching for personalized predictions of Parkinson's Disease. In *SIAM International Conference on Data Mining (SDM)*, pages 198–206, 2017.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *International Conference on Machine Learning*, 2023.

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *International Conference on Learning Representations*, 2014.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. Moment: A family of open time-series foundation models. In *International Conference on Machine Learning*, 2024.

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

Hrayr Harutyunyan, Hrant Khachatrian, David C. Kale, Greg Ver Steeg, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *Scientific Data*, 6(96), 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Alistair E. W. Johnson, Tom J. Pollard, Lu Shen, Li wei H. Lehman, Mengling Feng, Mohammad Mahdi Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3, 2016.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

Zeljko Kraljevic, Dan Bean, Anthony Shek, Rebecca Bendayan, Harry Hemingway, Joshua Au Yeung, Alexander Deng, Alfred Balston, Jack Ross, Esther Idowu, James T. Teo, and Richard J. B. Dobson. Foresight—a generative pretrained transformer for modelling of patient timelines using electronic health records: a retrospective modelling study. *The Lancet Digital Health*, 6(4):e281–e290, 2024.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. In *Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2022.

Matthew McDermott, Bret Nestor, Peniel Argaw, and Isaac S. Kohane. Event stream gpt: a data pre-processing and modeling library for generative, pre-trained transformers over continuous-time sequences of complex events. In *Advances in Neural Information Processing Systems*, 2023.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023.

Chao Pang, Xinzhuo Jiang, Krishna S. Kalluri, Matthew Spotnitz, RuiJun Chen, Adler Perotte, and Karthik Natarajan. CEHR-BERT: Incorporating temporal information from structured EHR data to improve prediction tasks. In *Machine Learning for Health*, volume 158, pages 239–260. PMLR, 2021.

Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. DeepCare: A deep dynamic memory model for predictive medicine. In *Advances in Knowledge Discovery and Data Mining*, pages 30–41. Springer, 2016.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (140):1–67, 2020.

Monika Schak and Alexander Gepperth. A study on catastrophic forgetting in deep LSTM networks. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*, pages 714–728. Springer, 2019.

Sofia Serrano and Noah A. Smith. Is attention interpretable? In *Annual Meeting of the Association for Computational Linguistics*, 2019.

Sindhu Tipirneni and Chandan K. Reddy. Self-supervised transformer for sparse and irregularly sampled multivariate clinical time-series. *ACM Transactions on Knowledge Discovery from Data*, 16(6):1–17, 2022.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, 2021.

Cao Xiao, Edward Choi, and Jimeng Sun. Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review. *Journal of the American Medical Informatics Association*, 25(10):1419–1428, 2018.

Zhichao Yang, Avijit Mitra, Weisong Liu, Dan Berlowitz, and Hong Yu. TransformEHR: transformer-based encoder-decoder generative model to enhance prediction of disease outcomes using electronic health records. *Nature Communications*, 14(7857), 2023.

Jiexia Ye, Weiqi Zhang, Ke Yi, Yongzi Yu, Ziyue Li, Jia Li, and Fugee Tsung. A survey of time series foundation models: Generalizing time series representation with large language model. *arXiv preprint arXiv:2405.02358*, 2024.

Tian Zhou, Peisong Niu, xue wang, Liang Sun, and Rong Jin. One fits all: Power general time series analysis by pretrained lm. In *Advances in Neural Information Processing Systems*, 2023.

Appendix A. Implementation Details

Experiments were run with NVIDIA RTX A6000, Python 3.11.5, Pytorch 2.4.0, Huggingface-hub 0.24.0, and MOMENT-1-large.

All experiments were run with the following hyperparameters:

1. Number of epochs: 10
2. Scheduler: OneCycleLR
3. Optimizer: AdamW
4. Learning rate: 5×10^{-5}
5. Max learning rate: 0.01
6. Weight decay: 0.05
7. Loss function: MSE for forecasting, binary cross-entropy for classification
8. Prompt size K: 16 for forecasting dataset, 4 for classification
9. LoRA hyperparameters:
    - Attention dimension: 1 (attention dimension was chosen to make the number of trainable parameters of LoRA close to that of Gen-P-Tuning.)
    - Alpha: 16
    - Dropout: 0.1

Other parameters are taken from the source code or the paper of MOMENT. We did not perform any hyperparameter tuning.

Appendix B. Aggregating Across Patches in the Prompt Module

The input to the Prompt Module has a size that scales with the number of patches whereas the output of the

Table 8: MIMIC Mortality test set scores (mean ± std. dev. over 5 random seeds). For each univariate foundation model, per column we bold whichever score is highest.

Model	Aggregation Method	Raw Accuracy	AUROC	F1	AUPRC
MOMENT	MLP	**0.882** ± 0.010	0.750 ± 0.025	0.553 ± 0.044	0.280 ± 0.017
	RNN	0.881 ± 0.010	**0.755** ± 0.027	0.584 ± 0.029	**0.304** ± 0.029
	Transformer	0.881 ± 0.005	0.754 ± 0.021	**0.591** ± 0.031	0.292 ± 0.026
GPT4TS	MLP	**0.892** ± 0.014	0.685 ± 0.066	0.479 ± 0.017	0.226 ± 0.051
	RNN	0.890 ± 0.011	**0.716** ± 0.030	0.489 ± 0.032	0.253 ± 0.026
	Transformer	0.887 ± 0.016	0.708 ± 0.025	**0.499** ± 0.033	**0.255** ± 0.038

Table 9: MIMIC Phenotyping test set scores (mean ± std. dev. over 5 random seeds). For each univariate foundation model, per column we bold whichever score is highest.

Model	Aggregation Method	Raw Accuracy	AUROC	F1	AUPRC
MOMENT	MLP	0.830 ± 0.006	0.640 ± 0.018	0.079 ± 0.030	0.270 ± 0.015
	RNN	**0.837** ± 0.004	**0.669** ± 0.013	0.129 ± 0.024	**0.297** ± 0.013
	Transformer	0.835 ± 0.004	0.666 ± 0.015	**0.135** ± 0.017	0.294 ± 0.012
GPT4TS	MLP	**0.832** ± 0.005	0.581 ± 0.018	0.026 ± 0.024	0.230 ± 0.016
	RNN	0.831 ± 0.006	0.592 ± 0.008	**0.037** ± 0.013	0.229 ± 0.007
	Transformer	**0.832** ± 0.003	**0.599** ± 0.010	0.020 ± 0.009	**0.231** ± 0.010

Table 10: Influenza-like illness forecasting test set scores (mean ± std. dev. over 5 random seeds). For each univariate foundation model, per column we bold whichever score is best.

Model	Aggregation Method	MSE	MAE
MOMENT	MLP	**2.718** ± 0.049	**1.154** ± 0.014
	RNN	3.044 ± 0.081	1.210 ± 0.017
	Transformer	3.083 ± 0.080	1.200 ± 0.016
GPT4TS	MLP	3.196 ± 0.556	1.253 ± 0.104
	RNN	**2.721** ± 0.678	**1.140** ± 0.145
	Transformer	2.939 ± 0.378	1.209 ± 0.092

Prompt Module does not depend on the number of patches. As a reminder (Section 3.2's step 1), the Prompt Module first embeds $\mathbf{E}$ (using an transformer) to obtain the array $\mathbf{P'} \in \mathbb{R}^{P \times C \times D}$. At this point, we want to summarize $\mathbf{P'}$ into an array that does not depend on P. There are many ways of doing this summarization that is fundamentally about aggregating information from $\mathbf{P'}$ across patches. Some examples include transformers, recurrent neural networks (RNN), and multi-layer perceptions (MLP). Transformers and RNNs are able to process sequences that vary in length. MLPs could also be applicable since many time series foundation models pad or truncate input sequences so that they are all the same length.

In more detail, we could aggregate across patches using the following three different approaches:

1. Transformer: A transformer strategy has already been presented in Section 3.2's step 2.
2. RNN: We use an RNN to map $\mathbf{P'}$ from $\mathbb{R}^{P \times C \times D}$ to $\mathbb{R}^{P \times C \times (K \cdot D)}$. We use the output at the last timestamp, and then a reshape operation to map it from $\mathbb{R}^{C \times (K \cdot D)}$ to $\mathbb{R}^{(K \cdot C) \times D}$.
3. MLP: We use a MLP to map $\mathbf{P'}$ from $\mathbb{R}^{P \times C \times D}$ to $\mathbb{R}^{K \times C \times D}$, then followed by a flatten operation to map it from $\mathbb{R}^{K \times C \times D}$ to $\mathbb{R}^{(K \cdot C) \times D}$.

We present the test performance of MIMIC Mortality (Table 8), MIMIC Phenotyping (Table 9), and forecasting (Table 10) using transformer, RNN, and MLP aggregation strategies.