Technical Appendix

An Efficient and Explainable Transformer-Based Few-Shot Learning for Modeling Electricity Consumption Profiles Across Thousands of Domains

August 9, 2024

Contents

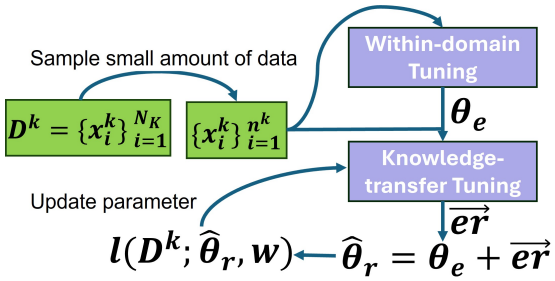

Figure 1: Training/inference process of one domain. $\hat{\theta_r} = \{\mu_j, \sigma_j\}_{j=1}^{J}$ represents the predicted parameter of GMMs, $\mathbf{w}$ is the weights of components, $l(\cdot)$ is the loss function. In the training process, $\mathcal{D}^k \in \mathcal{S}$. In the inference process, $\mathcal{D}^k \in \mathcal{T}$, and only $\hat{\theta}_r$ is predicted without loss computation and parameter updating for the Transformer.

1 Loss Design

In this section, we detail the motivation behind our loss design.

A naive approach to designing the loss function would involve directly applying the EM algorithm to the complete domain data to estimate the parameter θ_r. However, this approach has two major shortcomings 1) the θ_r estimate obtained through the EM algorithm is actually not globally optimal due to the inherent limitations of the EM method, and 2) Implementing the EM algorithm for every sample in each iteration significantly slows down the training process. Therefore, we utilize the negative log-likelihood as our loss function to bypass these issues, which is given by

$$\log \mathcal{L}(\mathcal{D}^k \mid \hat{\theta}_r, \mathbf{w}) = -\sum_{i=1}^{N} \log \left(\sum_{j=1}^{J} w_j \mathcal{N}(\mathbf{x}_i^k \mid \mu_j, \sigma_j) \right), \tag{1}$$

where $w_j \in \mathbf{w}$ is the fixed weight of the j-th Gaussian component, N is the number of data points in the domain, J is the number of Gaussian components, $\mathcal{N}(\mathbf{x}_i^k \mid \mu_j, \sigma_j)$ is the Gaussian probability density function for the j-th component with mean μ_j and covariance σ_j.

2 Training and inference

In this Section, we detailed the training and inference process which are summarized in Fig 1. In the main paper, we propose the following equation to determine z in Within-domain Tuning

$$z = \text{int}\left(e^{\beta n}\right), \tag{2}$$

where n is the number of ECP samples used in Within-domain Tuning, β is a parameter set to 0.015, which is the result of empirical testing in $\mathcal{S}$, int$(\cdot)$ means

the integer part of a value. With loss function in (1) and equation in (2), we can train the model using the algorithm shown in Algorithm 1. Similarly, for inference, we can use the algorithm shown in Algorithm 2.

Algorithm 1 Training

Require: Source domain collection $\mathcal{S}$, initial parameters θ_o, fixed weights $\mathbf{w}$, batch size L_s, the highest and lowest learning rates l_h, l_w.

Repeat
  Initialize parameters of GMMs $\theta = \theta_o$
  Sample L_s source domains from $\mathcal{S}$ to form the dataset $\{\mathcal{D}^k\}_{k=1}^{L_s}$
  From each source domain in $\{\mathcal{D}^k\}_{k=1}^{L_s}$, randomly sample n^{k_s} ECP samples, where $4 \leq n^{k_s} \leq 25$
  Compute the corresponding z using (2) for each sampled source domain
  Perform Within-domain Tuning to obtain θ_e for each source domain
  Perform Knowledge-transfer Tuning to obtain $\vec{er}$ for each source domain
  Compute the predicted $\hat{\theta}_r$ for each source domain
  Compute the loss in (1) and update the model parameters
Until convergence

Algorithm 2 Inference

Require: One target domain $\mathcal{D} = \{\mathbf{x_i}\}_{i=1}^{N}$, initial parameters θ_o, fixed weights $\mathbf{w}$, pre-trained Transformer encoder.

  Initialize parameters of GMMs $\theta = \theta_o$
  Compute the corresponding z using (2) and ECP samples in $\mathcal{D}$
  Perform Within-domain Tuning to obtain θ_e for the domain
  Perform Knowledge-transfer Tuning to obtain $\vec{er}$ for the domain
  Compute the predicted $\hat{\theta}_r$ for the domain

3 Implementation Details of Our Method

3.1 LogSumExp Trick

It is highly likely to encounter numerical stability issues during training using the loss function (1) and solving EM in batch (Within-domain Tuning). Therefore, we implement the LogSumExp trick to prevent instability. The LogSumExp trick ensures numerical stability by preventing overflow and underflow in the exponential function computations. The loss function (1) has format like

$$\text{LogSumExp}(x_1, x_2, \ldots, x_n) = \log\left(\sum_{i=1}^{n} \exp(x_i)\right). \tag{3}$$

However, for numerical stability, we implement it as

$$\text{LogSumExp}(x_1, x_2, \ldots, x_n) = m + \log\left(\sum_{i=1}^{n} \exp(x_i - m)\right), \tag{4}$$

where $m = \max(x_1, x_2, \ldots, x_n)$. By using this trick, we ensure that our computations remain stable, even when dealing with extreme values, thereby improving the robustness of our training process.

3.2 Other Details

The proposed method is order-agnostic, meaning that swapping the order of input ECP samples (or ECP token order) does not affect the output. Therefore, operations such as batch normalization should not be included in the model, as they would compromise this characteristic.

4 Implementation of TimesNet

In the main paper, we present the evaluation of TimesNet using two distinct loss functions. We adopt the time series imputation model from the original paper in `https://github.com/thuml/Time-Series-Library.git`. However, the imputation task of TimesNet is originally trained with MSE loss which is defined as

$$\mathcal{L}_{\text{MSE}} = \sum_{i=1}^{L} (y_i - \hat{y}_i)^2, y_i \in \mathbb{R}^{N^k \times T}, \hat{y}_i \in \mathbb{R}^{N^k \times T}, \tag{5}$$

where L is the batch size, y_i is the i-th original time series data and $\hat{y}_i$ is the i-th imputed time series data. N^k is the number of ECP samples in the k-th domain. Our primary objective in this paper is to model the distribution of ECP data rather than to accurately impute the energy consumption time series. The MSE loss implicitly takes into account the order of ECP samples, which may impose unnecessary constraints on the model. To mitigate this issue, we employ the MMD as the loss function to train the model. MMD is defined as

$$\begin{aligned}\mathcal{L}^2_{\text{MMD}} =& \frac{1}{N^k(N^k-1)} \sum_i \sum_{j \neq i} k(y_i, y_j) - \frac{2}{N^k N^k} \sum_i \sum_j k(y_i, \hat{y}_j) \\ &+ \frac{1}{N^k(N^k-1)} \sum_i \sum_{j \neq i} k(\hat{y}_i, \hat{y}_j),\end{aligned} \tag{6}$$

where $k(\cdot)$ is the kernel function applied to pairs of data points. In this paper, we adopt the Radial Basis Function kernel, which is commonly used

$$k(\hat{y}_i, y_i) = \exp\left(-\frac{\|\hat{y}_i - y_i\|^2}{2\sigma^2}\right), \tag{7}$$

where σ is a hyper-parameter to set.

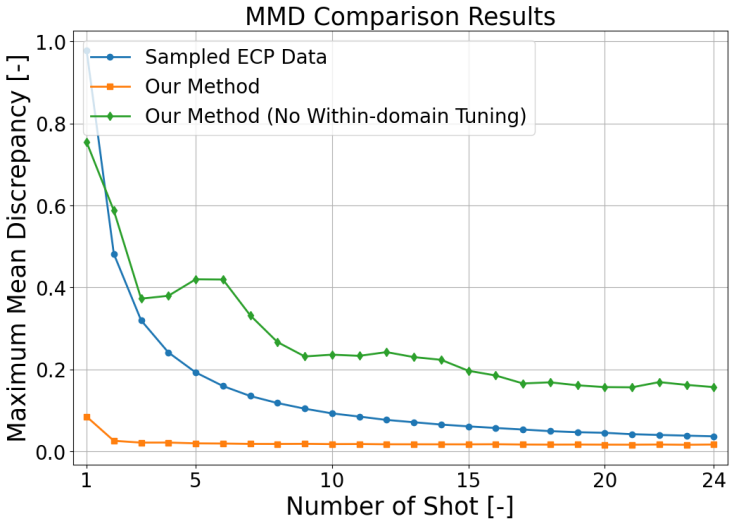

Figure 2: Comparison of MMD values. For each shot, we sample 100 target domains from $\mathcal{T}$ and compute the average MMD values of n^{k_t} (number of shots) from 1 to 24.

5 Ablation Study

In this section, we present a detailed evaluation of the impact of Within-domain Tuning. To assess its effectiveness, we conducted experiments where this component was omitted, and only the Transformer encoder was employed to directly predict $\vec{or}$, subsequently calculating $\hat{\theta}_r = \vec{or} + \theta_o$. Fig 2 shows the MMD value of n^{k_t} range from 1 to 24, and compares the performance of our method with and without Within-domain Tuning. The results clearly indicate that, without Within-domain Tuning, the model's performance deteriorates. The reason behind this is, as mentioned in the main paper, the distance $||\theta_o - \theta_r||$ in the parameter space of GMMs can be very large, which can adversely affect the model's accuracy. Figure 3 presents the 24-shot generation results of the GMMs predicted with and without Within-domain Tuning.

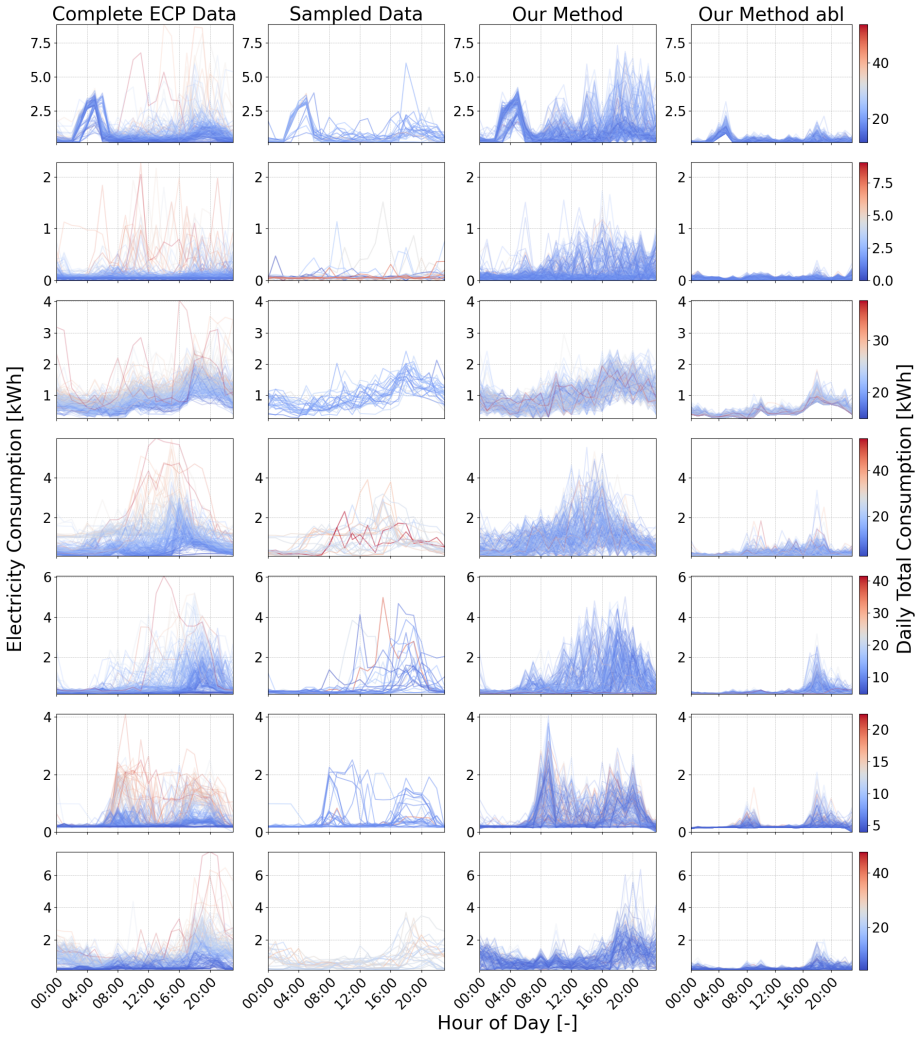

Figure 3: In each subfigure, every row represents the experimental results for a specific target domain. Each row, from left to right, includes (1) the complete ECP data of the domain, (2) the sampled ECP data used as input for models, (3) the ECP data generated from GMMs whose parameters are predicted by our method, and (4) the ECP data generated from GMMs whose parameters are predicted by our method without using Within-domain Tuning.