Appendix of "HyperTime: Hyperparameter Optimization for Combating Temporal Distribution Shifts"

This appendix is organized into several sections that provide additional details and information related to the main paper:

(1) Section A presents the details of the hyperparameter optimization algorithm in our method.
(2) Section B includes a theoretical analysis in this paper.
(3) Section C shows detailed information on the search space used in the paper.
(4) Section D provides additional empirical evaluation results.
(5) Section E provides detailed information on the datasets used in the paper.

A DETAILS OF LEXIFLOW

LexiFlow is a randomized direct search based HPO algorithm, which is able to direct the search to the optimum based on lexicographic comparisons over pairs of configuration. It start from a initial hyperparameter configuration and gradually move to the optimal point by making comparisons with nearby configurations in the search space. More details about LexiFlow could be found in the paper [60].

Algorithm 1 LexiFlow

```
Input: Objectives $\mathbf{L}(\cdot)$, tolerances $K$ (optional).
1  Initialization: Initial configuration $c_0$, $t' = r = s = 0$, $\delta = \delta_{init}$;
2  Obtain $\mathbf{L}(c_0)$, and $c^* \leftarrow c_0$, $\mathcal{H} \leftarrow \{c_0\}$, $Z_{\mathcal{H}} \leftarrow \mathbf{L}(c_0)$
3  while $t = 0, 1, \ldots$ do
4      Sample $\mathbf{u}$ uniformly from unit sphere $\mathbb{S}$
5      if Update($\mathbf{L}(c_t + \mu\mathbf{u}), L(c_t), Z_{\mathcal{H}}$) then $c_{t+1} \leftarrow c_t + \mu\mathbf{u}$, $t' \leftarrow t$;
6      else if Update($\mathbf{L}(c_t - \mu\mathbf{u}), L(c_t), Z_{\mathcal{H}}$) then $c_{t+1} \leftarrow c_t - \mu\mathbf{u}$, $t' \leftarrow t$ ;
7      else $c_{t+1} \leftarrow c_t$, $s \leftarrow s + 1$ ;
8      $\mathcal{H} \leftarrow \mathcal{H} \cup \{c_{t+1}\}$, and update $Z_{\mathcal{H}}$ according to (9)
9      if $s = 2^{d-1}$ then $s \leftarrow 0$, $\delta \leftarrow \delta\sqrt{(t'+1)/(t+1)}$ ;
10     if $\delta < \delta_{lower}$ then
           // Random Restart
11         $r \leftarrow r + 1$, $c_{t+1} \leftarrow N(c_0, I)$, $\delta \leftarrow \delta_{init} + r$
12 Procedure Update($\mathbf{L}(c'), \mathbf{L}(c), Z_{\mathcal{H}}$):
13 if $\mathbf{L}(c') \prec_{(Z_{\mathcal{H}})} \mathbf{L}(c)$ Or ($\mathbf{L}(c') =_{(Z_{\mathcal{H}})} \mathbf{L}(c)$ and $\mathbf{L}(c') \prec_l \mathbf{L}(c)$) then
14     if $\mathbf{L}(c') \prec_{(Z_{\mathcal{H}})} \mathbf{L}(c^*)$ Or ($\mathbf{L}(c') =_{(Z_{\mathcal{H}})} \mathbf{L}(c^*)$ and $\mathbf{L}(c') \prec_l \mathbf{L}(c^*)$)
        then
15         $c^* \leftarrow c'$
16     Return True
17 else
18     Return False
19 Output: A lexi-optimal configuration $c^*$
```

[1]We adjust LexiFlow and make such changes: 1. Remove the optional input targets 2. Adjust tolerance from an absolute value to a relative value in percentage.

Given any two hyperparameter c' and c, the targeted lexicographic relations $=_{(Z)}$, $\prec_{(Z)}$ and $\preceq_{(Z)}$ in Algorithm 1 are defined as:

$$\mathbf{L}(c') =_{(Z)} \mathbf{L}(c) \Leftrightarrow L^{(i)}(c') = L^{(i)}(c) \vee \tag{6}$$
$$(L^{(i)}(c') \leq z^{(i)} \wedge L^{(i)}(c) \leq z^{(i)})\ \forall i \in [1, ..., I],$$
$$\mathbf{L}(c') \prec_{(Z)} \mathbf{L}(c) \Leftrightarrow \exists i \in [I] : L^{(i)}(c') < L^{(i)}(c) \wedge \tag{7}$$
$$L^{(i)}(c) > z^{(i)} \wedge L_{i-1}(c) =_{(Z)} L_{i-1}(c'),$$
$$\mathbf{L}(c') \preceq_{(Z)} \mathbf{L}(c) \Leftrightarrow \mathbf{L}(c') \prec_{(Z)} \mathbf{L}(c) \vee \mathbf{L}(c') =_{(Z)} \mathbf{L}(c), \tag{8}$$

Where $L_{i-1}(c)$ denotes the a vector with the first $i-1$ dimensions of $\mathbf{L}(c)$, i.e., $L_{i-1}(c) = [L^{(1)}(c), ..., L^{(i-1)}(c)]$. $\forall i \in [1, ..., I]$, $z^{(i)}$ are computed based on historically evaluated points $\mathcal{H}$. $C^0_{\mathcal{H}} = \mathcal{H}$, $\forall i \in [1, ..., I]$:

$$z^{(i)} = L^{(i)}_{\mathcal{H}} * (1 + \kappa^{(i)}), C^i_{\mathcal{H}} := \{c \in C^{i-1}_{\mathcal{H}} | L^{(i)}(c) \leq z^{(i)}\},$$
$$L^{(i)}_{\mathcal{H}} := \min_{c \in C^{i-1}_{\mathcal{H}}} L^{(i)}(c). \tag{9}$$

B THEORETICAL ANALYSIS

(1) We denote the k-th validation set as: $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_K$.
(2) We use d to denote the a data instance pair $(\mathbf{x}, y)$ in a particular validation set $\mathcal{D}$ in general.
(3) We use $l_c(d)$ to denote the loss of a particular ML model configured c on data instance d.
(4) We use $\mathbb{P}$ to denote the test data distribution.

Proof of Lemma 1. We denote by $\mathbb{P}$ the data distribution on which $\mathcal{D}_{\text{test}}$ and $\mathcal{D}_{\text{val}}$ is drawn from. Without loss of generality, we assume the loss function is Mean squared Error, i.e., for any validation set $\mathcal{D}$, $Loss(f_c, \mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} l_c(d) = \frac{1}{|\mathcal{D}|}\sum_{(\mathbf{x},y)\in\mathcal{D}}(f_c(\mathbf{x}) - y)^2$. We further assume a bounded loss: $\forall d \sim \mathbb{P}$, $l_c(d) < \beta$. We have:

$$|\mathtt{Loss}(f_c, \mathcal{D}_{\text{val}}) - \mathbb{E}[\mathtt{Loss}(f_c, \mathcal{D}_{\text{test}})]| = |\frac{1}{|\mathcal{D}_{\text{val}}|}\sum_{i=1}^{|\mathcal{D}_{\text{val}}|} l_c(d_i) - \mathbb{E}_{d\sim\mathbb{P}}[l_c(d)]|, \tag{10}$$

where d_i is the i-th data instance in $\mathcal{D}_{\text{val}}$ and thus $d_i \sim \mathbb{P}$. According to Hoeffding's inequality [22], we have:

$$Pr(|\mathtt{Loss}(f_c, \mathcal{D}_{\text{val}}) - \mathbb{E}[\mathtt{Loss}(f_c, \mathcal{D}_{\text{test}})]| > \epsilon) = \tag{11}$$
$$|\frac{1}{|\mathcal{D}_{\text{val}}|}\sum_{i=1}^{|\mathcal{D}_{\text{val}}|} l_c(d_i) - \mathbb{E}_{d\sim\mathbb{P}}[l_c(d)]| \leq$$
$$2\exp\frac{-2|\mathcal{D}_{\text{val}}|\epsilon^2}{\frac{1}{|\mathcal{D}_{\text{val}}|}\sum_{i=1}^{|\mathcal{D}_{\text{val}}|}\beta} = 2\exp\frac{-2|\mathcal{D}_{\text{val}}|\epsilon^2}{\beta}.$$

By letting $2\exp\frac{-2|\mathcal{D}_{\text{val}}|\epsilon^2}{\beta} = \epsilon$, we have with probability at least $1-\epsilon$.

$$|\mathtt{Loss}(f_c, \mathcal{D}_{\text{val}}) - \mathbb{E}[\mathtt{Loss}(f_c, \mathcal{D}_{\text{test}})]| \leq \sqrt{\frac{\beta\ln(2/\epsilon)}{2|\mathcal{D}_{\text{val}}|}}.$$

Which completes the proof. □

Proof of Theorem 2. **Proof sketch.** We consider the following two cases: (I) $L_{k^*}(\hat{c}) \leq L_{avg}(\hat{c})$; (II) $L_{k^*}(\hat{c}) > L_{avg}(\hat{c})$.

It is easy to prove that under case (I), we have $L_{k^*}(\hat{c}) \leq L_{\text{avg}}(\hat{c}) \leq (1+\kappa)L^*_{\text{avg}} \leq (1+\kappa)L_{\text{avg}}(c^*_{k^*})$, in which the last inequality is based on the definition of L^*_{avg}.

Under case (II), when $\kappa \geq \frac{L_{\text{avg}}(c^*_{k^*})}{L^*_{\text{avg}}} - 1$, we have $c^*_{k^*} \in C^{(0)}_*$, i.e., $c^*_{k^*}$ is within the $\kappa^{(0)}$-tolerance from the best average validation loss, and thus

$$L_{k^*}(\hat{c}) \leq L_{\text{worst}}(\hat{c}) \leq L_{\text{worst}}(c^*_{k^*}),$$

in which the last inequality is based on the fact that $c^*_{k^*}$ is within the $\kappa^{(0)}$-tolerance from the best average validation loss. We choose the best configuration according to the performance on L_{worst}, and thus $L_{\text{worst}}(\hat{c}) < L_{\text{worst}}(\tilde{c})$ for all $\tilde{c} \in C^{(0)}_*$.

Combining the conclusions under both cases and high probability concentration of $L_{k^*}(\hat{c})$ to $\mathbb{E}[\texttt{Loss}(f_{\hat{c}}, \mathcal{D}_{\text{test}})]$ finishes the proof. We provide a more rigorous proof below.

Case 1: $L_{k^*}(\hat{c}) \leq L_{avg}(\hat{c})$.

In this case, we have,

$$L_{k^*}(\hat{c}) \leq L_{\text{avg}}(\hat{c}) \leq (1+\kappa)L^*_{\text{avg}} \leq (1+\kappa)L_{\text{avg}}(c^*_{k^*}). \tag{12}$$

Case 2: $L_{k^*}(\hat{c}) > L_{avg}(\hat{c})$:

When $\kappa \geq \frac{L_{\text{avg}}(c^*_{k^*})}{L^*_{\text{avg}}} - 1$, we have $c^*_{k^*} \in C^1_*$, and thus

$$L_{k^*}(\hat{c}) \leq L_{\text{worst}}(\hat{c}) \leq L_{\text{worst}}(c^*_{k^*}), \tag{13}$$

in which the last inequality is based on the fact that when $c^*_{k^*} \in C^1_*$, we choose the best configuration according to the performance on L_{worst}, and thus $L_{\text{worst}}(\hat{c}) < L_{\text{worst}}(\tilde{c})$ for all $\tilde{c} \in C^1_*$.

Combining **Case 1** and **Case 2**, we have,

$$L_{k^*}(\hat{c}) \leq \begin{cases} (1+\kappa)L_{avg}(c^*_{k^*}), \text{ if } L_{k^*}(\hat{c}) \leq L_{avg}(\hat{c}) \\ L_{\text{worst}}(c^*_{k^*}), \text{ Otherwise} \end{cases} \tag{14}$$

According to the conclusion from Lemma 1, we have,

$$\mathbb{E}[\texttt{Loss}_{\mathcal{D}_{\text{test}}}(\hat{c})] \leq L_{k^*}(\hat{c}) + \sqrt{\frac{\beta \ln(1/\delta)}{2|\mathcal{D}_{val}|}}. \tag{15}$$

Combining Eq. (14) and Eq. (15) finishes the proof. □

C SEARCH SPACE

C.1 Search Space of gradient-boosting tree

Table 7: Hyperparameters tuned in XGboost.

hyperparameter	type	range
estimators number	int	[4, min(32768, train_datasize)]
max leaves	int	[4, min(32768, train_datasize)]
max depth	int	[0, 6, 12]
min child weight	float	[0.001, 128]
learning rate	float	[1/1024, 1.0]
subsample	float	[0.1, 1.0]
colsample by tree	float	[0.01, 1.0]
colsample by level	float	[0.01, 1.0]
reg alpha	float	[1/1024, 1024]
reg lambda	float	[1/1024, 1024]

Table 8: Hyperparameters tuned in LGBM.

hyperparameter	type	range
estimators number	int	[4, min(32768, train_datasize)]
leaves number	int	[4, min(32768, train_datasize)]
min child sample	int	[2, 129]
learning rate	float	[1/1024, 1.0]
log_max_bin	int	[3, 11]
colsample by tree	float	[0.01, 1.0]
reg alpha	float	[1/1024, 1024]
reg lambda	float	[1/1024, 1024]

C.2 Search Space of neural network

We use the same neural network backbone as the Wild-Time [55] benchmark for different datasets based on its source code https://github.com/huaxiuyao/Wild-Time. We list the detailed search space used in different datasets in Table 9.

Table 9: Hyperparameters tuned in neural networks.

Dataset	Hyparameter	type	Range
Yearbook	Training iteration	int	[3000, 5000]
	learning rate	float	[1e-4,1e-1]
	batch size	int	{32, 64, 128, 256}
	n_conv_channels	int	[16, 512]
	kernel_size	int	{2, 3, 4, 5}
	has_max_pool	bool	True or Flase
FMoW-Time	Training iteration	int	[3000, 6000]
	learning rate	float	[1.5e-5,3e-4]
	batch size	int	{32, 64, 128, 256}
	weight_decay	float	[0, 0.03]
MIMIC-IV	Training iteration	int	[3000, 5000]
	learning rate	float	[5e-4,5e-2]
	n_head	int	{2, 3, 4, 5}
	n_layer	int	{2, 3, 4, 5}
	hidden_size	int	{64, 128, 256, 512}
Huffpost	Training iteration	int	[6000, 8000]
	learning rate	float	[1e-5,1e-4]
	weight_decay	float	[0.01, 0.03]
arXiv	Training iteration	int	[6000, 8000]
	learning rate	float	[1e-5,1e-4]
	weight_decay	float	[0.01, 0.03]

Table 10: The number of validation folds and test folds for each dataset of the Wild-Time benchmark in our experiments.

	Yearbook	FMoW-Time	MIMIC-IV	Huffpost	arXiv
Val. num	8	11	2	4	10
Tes. num	9	5	2	3	6

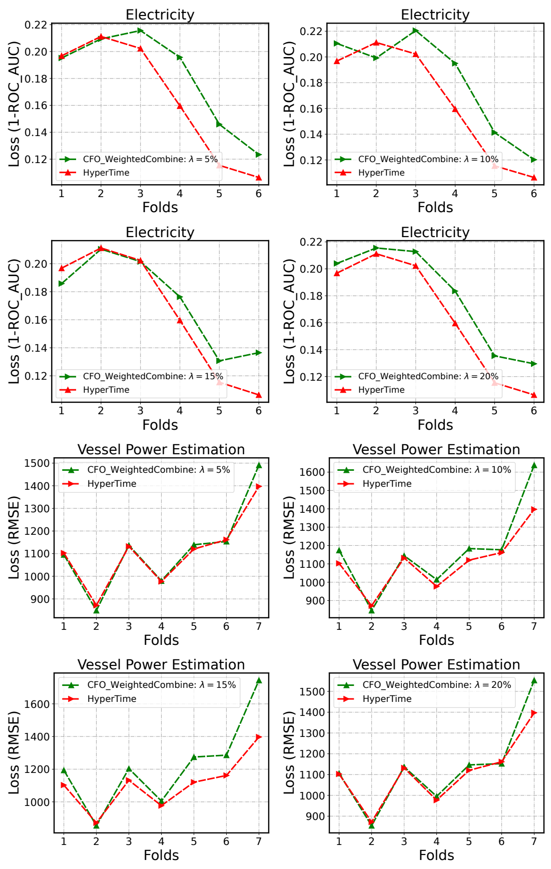

Figure 6: Per fold test loss (lower the better) for Hypertime and CFO_WeightedCombine on Electricity and Vessel Power Estimation datasets with different weight settings. The results are averaged over five random seeds.

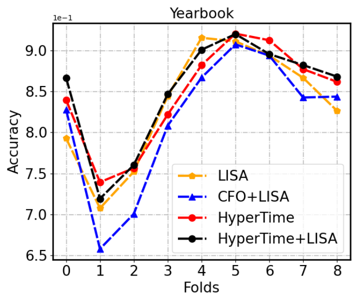

Figure 7: Per fold test accuracy for a state-of-the-art robust training method LISA [56], our method HyperTime, and the methods combining LISA and CFO and HyperTime respectively. The results are from the same set of experiments with that in Table 6. All the numbers are the higher the better.

D ADDITIONAL EMPIRICAL RESULTS

Per fold performance in Table 6. In Figure 7, we present the per-fold test performance for different methods in Table 6. Our observations indicate that the combination of HyperTime and Lisa achieves the best performance compared to other methods. It demonstrates that the combination of HyperTime and other non-HPO solutions overall further boost the model performance.

Supplementary results of CFO_WeightedCombine. In this section, we conduct additional experiments to compare HyperTime with CFO_WeightedCombine, which set the optimization objectives as a weighted combination of average validation loss and the worst fold validation loss in CFO. To represent the weights assigned to the worst fold validation loss, we use the symbol λ. Consequently, we set the weight for the average validation loss as $1 - \lambda$. We use four different λ settings for CFO_WeightedCombine: 5%, 10%, 15%, and 20%. Figure 6 shows the per-fold test loss of these two methods. We observe that HyperTime outperforms CFO_WeightedCombine under all four weight settings. This further demonstrates the importance of formulating the optimization of these two objectives as a lexicographic optimization problem.

E DATASET DETAILS

E.1 More details about Temperature prediction dataset

Temperature prediction is a synthetic data for urban climate research, which includes 75 years of urban climate condition information in specific areas. It has distribution shifts as mentioned in existing urban climate research works [28, 40]. Here we select 16 gridcells of data according to [65], with the latidude of 35.34, 36.28, 37.23, 38.17 and longitude of 115.0, 116.2, 117.5, 118.8. It includes ten features including near-surface humidity, eastward near-surface wind, precipitation, etc. In our experiment, we predict the urban daily maximum of average 2-m temperature which could be regarded as a regression task. More information about this data is available at [65].

E.2 More information about fold splitting

To ensure fair and consistent comparisons, we use the same validation/test folds splitting setting as the Wild-Time benchmark. We list the number of training and test folds in Table 10. More information can be found in [55].