OpenFE++: Efficient Automated Feature Generation via Feature Interaction

Lei Wang* Yu Shi* Yifei Jin† Jian Li*

Abstract

Automated feature generation can greatly enhance the performance of machine learning models in many tabular and time-series prediction problems. The current state-of-the-art method, OpenFE, follows the "expand-and-reduce" framework, which generates a candidate feature set and subsequently extracts the most effective features from this set. Nevertheless, with the increase in the number of features and the length of time series, feature generation algorithms based on expand-and-reduce often produce an overwhelmingly large pool of candidate features, rendering the identification of effective features quite time-consuming and more prone to overfitting. To resolve this issue, we propose OpenFE++, which leverages the feature interactions from both the feature and temporal dimensions to construct a substantially reduced candidate feature set, thereby enhancing efficiency and effectiveness in feature generation. In the feature dimension, OpenFE++ utilizes locally interacted features to generate meaningful candidate features without exhaustively enumerating all possibilities. In the temporal dimension, it evaluates the lagged effects among different features and generates temporally meaningful features via the representative lagged periods, eliminating the need for enumeration in sequence length. Thus, OpenFE++ can efficiently generate effective, generalizable and interpretable features to boost the forecasting performance of machine learning models. We conduct extensive experiments on fourteen widely used benchmark datasets (ten benchmarks for tabular tasks and four benchmarks for time-series tasks) to demonstrate that OpenFE++ outperforms other baseline models in both efficiency and effectiveness.

Key words: Feature Generation, Feature Interaction, Tabular Data, Time-series Data

1 Introduction

Feature generation [1, 2] is a crucial and challenging task, recognized for its ability to improve the performance of machine learning models in both tabular prediction tasks [3, 4, 5, 6] and time-series forecasting tasks [7, 8, 9, 10]. The conventional approach to feature generation require data scientists to leverage domain-specific knowledge to create a set of features tailored to a specific scenario. However, due to its time-consuming nature and the requirement for specialized expertise, researchers are progressively transitioning from manual design to automated feature generation methods.

Automated feature generation refers to creating new features from existing data in an automated manner [11]. This process typically follows an "expand-and-reduce" framework, such as OpenFE [3], AutoFeat [12] and SAFE [13]. Initially, a large pool of candidate features is constructed by extracting characteristics or combining existing features using pre-defined operators. Subsequently, specific feature selection methods are employed to extract the most effective features. The primary challenge in this task lies in the substantial size of the candidate feature set, as the sheer volume of candidate features can lead to computational inefficiencies and hinder the identification of the most relevant and effective features. For this challenge, Autocross [5] employs beam search to constrain the growth of the search space for high-order features in the expansion phase. OpenFE [3] has designed a two-stage pruning algorithm to efficiently filter features in the reduction phase. Nevertheless, for a large number of features, existing algorithms in the expansion phase still results in a substantial volume of candidate features, making the identification of effective features quite hard and extreme time-consuming.

For this issue, we consider the feasibility of evaluating the potential performance of the feature combinations before their actual calculation. We explore how machine learning models combine different features and extract the interacted features from the model to serve as the candidate feature combinations, as ML models are recognized for their capability to capture complex interactions. Though they are effective in combining features, the well-known overfitting issue can lead to spurious interactions that impede the model's generalization. To demonstrate this problem, we assess the feature interaction strength [14, 15] in a trained model in Fig. 1. We observe that the genuine interacted features may be concealed by spurious interactions induced by important features. Hence, extracting feature interactions should be done in localized combination spaces (where

*Tsinghua University. Email: wanglei20@mails.tsinghua.edu.cn, shi-y23@mails.tsinghua.edu.cn, lijian83@mail.tsinghua.edu.cn

†X-technology (Beijing) Co. Ltd. Email: yfjin1990@gmail.com

features share similar characteristics to exclude other interfering factors). Furthermore, the interaction among different time-series variables frequently displays lagged effects rather than immediate impacts. Consequently, measuring the representative lagged periods and constructing temporal features based on them can generate meaningful and potentially effective candidate features.

Based on the above motivations, we propose OpenFE++ to prune the candidate feature set in both the feature and temporal dimensions. For the feature dimension, our algorithm preserves feature combinations demonstrating stronger interaction strength within the localized spaces. For the temporal dimension, we measure the interaction with lagged effects among temporal variables and generate the temporal features across the representative lagged periods. This design significantly reduces the size of the candidate feature set, amplifying the efficiency of feature generation, and it enables automated feature generation with a large number of features and long sequence length. Notably, extensive experiments have demonstrated that the removal of superfluous candidate features leads to enhanced downstream performance. This indicates that the carefully curated candidate feature set can not only accelerate the generation process but also reduce the risk of overfitting. The contributions of our paper are summarized as follows:

- We tackle the substantial size issue of the candidate feature set in automated feature generation. To facilitate the effective pruning of candidate features before their actual calculation, we conduct an in-depth investigation of the effects of feature interactions from both the feature and temporal dimensions.
- We propose OpenFE++, which utilizes the locally interacted features and representative lagged periods to construct an effective candidate feature set with significantly smaller size.
- We conduct extensive experiments on 14 datasets, and our experimental results indicate that OpenFE++ significantly enhances the efficiency and effectiveness of automated feature generation. As an example, for third-order candidate features (typically totaling around 10^8), OpenFE++ preserves only 1%% of the candidates and efficiently extracts effective features in 30 minutes, while previous methods either fail to find effective features or fail to complete within a day.

2 Related Work

Automated Feature Generation "Expand-and-reduce" [3] is a widely used framework in the automated feature generation. Within this framework, there are two main research threads: the first focuses on expanding the candidate feature set more effectively [5, 13, 16, 17], and the second focuses on feature selection using different criteria [3, 12, 13, 18]. In the expansion phase, the most significant challenge is the enormous size of the candidate feature set. For this issue, AutoCross [5] deploys the beam search method to limit the growth of the expansion space, FCTree [16] and SAFE [13] determine the operators and ingredient features by fitting tree models on the training data. Besides constraining space, [17, 18, 19, 20] develop policy networks to directly generate valuable candidate features. In the reduction phase, the main challenge lies in the computation of numerous features and the assessment of their incremental performance. To resolve this issue, AutoFeat [12] uses regularized method and SAFE [13] uses information value to filter out uninformative features. OpenFE [3] and AutoCross [5] employ a successive approach to allocate computational resources and use a simple and effcient learning model to measure the incremental performance. AlphaGen [18] considers the correlation of different features, maintaining a high-predictive and low-correlation feature set.

Feature Interaction Feature interaction [14] refers to the integrated effects of two or more features in statistical or machine learning models, and understanding interaction helps in building reliable models. For interaction, Two-way ANOVA [14] and FAST [21] conduct hypothesis tests to check whether there exists interaction between two variables. Another scheme is to list all the interaction forms of interest and then extract important interaction effects by the Lasso methods [22, 23]. With the development of machine learning, it is widely acknowledged that ML can effectively capture complex statistical interactions [24, 25], leading to a growing trend in detecting feature interactions within trained models. For instances, the appearance in the same tree path for tree-based models [15], second-order partial derivative [26] and Neural Interaction Detection (NID) [27] for neural networks. Though effective, these methods may identify spurious feature interactions, potentially overlooking effective feature combinations (Sec. 4.1).

3 Problem Setting

Generate Candidate Features For the tabular data, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{1\le i\le N}$ contains a set of features $\mathcal{J}$ ($|\mathcal{J}| = d$ and $\boldsymbol{x}_i \in \mathbb{R}^d$). We utilize the predefined operator set $\mathcal{A}$ (*e.g.*, $\{\times, \max\}$) to generate the candidate feature set $\mathcal{A}(\mathcal{J})$ via enumeration. For any feature pairs, we iterate over the operator set $\mathcal{A}$ and apply all the feasible operators to generate new features. Take the second-order combinations for example, each feature pair $\{(x^{(j_1)}, x^{(j_2)})\}$ iterates over all feasible operators to generate a list of second-order features $\{x^{(j_1)} \times x^{(j_2)}, \max(x^{(j_1)}, x^{(j_2)})\}$. Specifically for k-order features, it involves k features $\{(x^{(j_1)}, ..., x^{(j_k)})\}$

that are combined with $k-1$ feasible operators, and its size $|\mathcal{A}(\mathcal{J})| \propto d^k$ increases exponentially with order k.

For the univarite time-series forecasting task, $\mathcal{D} = \{(\boldsymbol{x}_{t-L+1:t}, y_{t+1:t+H})\}_{L \le t \le N-H}$ is a sequence of data points with a set of correlated variates ($\boldsymbol{x}_{t-L+1:t} \in \mathbb{R}^{d \times L}$), where L, H denote the lookback window and the forecasting horizon respectively. Besides the operators mentioned above, there exist some temporal operators (*e.g.*, ts_max($\boldsymbol{x}^{(j_1)}, T$)), which require an additional variable $T \in [1, L]$ to compute the features. After generating d' features, we concatenate them with the original features, $\boldsymbol{x}'_{t-L+1:t} \in \mathbb{R}^{(d+d') \times L}$, and then proceed to evaluation. The operators are detailed in Appendix.

Automated Feature Generation Automated feature generation focuses on automatically generating new features based on existing data to enhance the performance of models in downstream tasks. Assume the data $\mathcal{D}$ ($|\mathcal{D}| = N$) can be divided into training, validation and testing sets (*i.e.*, $\mathcal{D}_{tr}, \mathcal{D}_{val}, \mathcal{D}_{te}$), and $\mathcal{D}$ consists of a set of base features $\mathcal{J}$ ($|\mathcal{J}| = d$). Given the pre-defined operators $\mathcal{A}$, the base features $\mathcal{J}$ can generate a large set of candidate features by enumeration, denoted as $\mathcal{A}(\mathcal{J})$. The automated feature generation method will select a small set of features $\mathcal{S} \subseteq \mathcal{A}(\mathcal{J})$, deploy a learning algorithm $\mathcal{L}$ to build a model $\mathcal{L}(\mathcal{D}_{tr}, \mathcal{J}+\mathcal{S})$ based on the features $\mathcal{J}+\mathcal{S}$, and evaluate the performance on the validation set $\mathcal{D}_{val}$ with $\mathcal{J}+\mathcal{S}$:

$$\max_{\mathcal{S} \subseteq \mathcal{A}(\mathcal{J})} \mathcal{E}\left(\mathcal{L}\left(\mathcal{D}_{tr}, \mathcal{J}+\mathcal{S}\right), \mathcal{D}_{val}, \mathcal{J}+\mathcal{S}\right), \tag{3.1}$$

where $\mathcal{E}$ represents the evaluation metric. Then, the higher evaluation score on the testing set $\mathcal{D}_{te}$ (*i.e.*, $\mathcal{E}(\mathcal{L}\left(\mathcal{D}_{tr}, \mathcal{J}+\mathcal{S}\right), \mathcal{D}_{te}, \mathcal{J}+\mathcal{S})$ indicates the better performance of generated features $\mathcal{S} \subseteq \mathcal{A}(\mathcal{J})$.

4 Methods

In this section, we first discuss the necessity to prune the candidate feature set $\mathcal{A}(\mathcal{J})$ and the intuition of our reduction methods. Subsequently, we elaborate on OpenFE++ in constructing candidate feature set via locally interacted features and generating temporal features via representative lagged periods.

4.1 Motivation

High Complexity The complexity of the candidate feature set $\mathcal{A}(\mathcal{J})$ can be extremely high. For the k-order feature combinations, d base features can generate $\mathcal{O}(q^{k-1}d^k)$ candidate features, where q represents the number of binary operators (e.g., $\{+, -, \times, \div\}$). In addition, when dealing with time-series data, the candidate feature set will expand to $\mathcal{O}((qL)^{k-1}d^k)$, where L denotes the lookback horizon. The exponential growth in complexity makes it impractical to calculate these features and apply automated feature generation to large-scale datasets. On the other hand, the sheer volume hinders the identification of the most effective features, consistently increasing the risk of overfitting. For this issue, we aim to prune the candidate feature set prior to feature calculation, incorporating the feature interactions from both the feature and temporal dimensions.

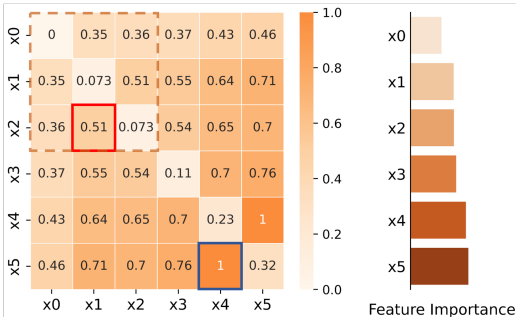


Figure 1: Feature interaction strength (left) and feature importance (right) extracted from a trained model. Important features often induce spurious interactions $\{x_4, x_5\}$, and genuine interaction $\{x_1, x_2\}$ can be prominently highlighted in localized spaces.

Feature Interaction Feature interaction refers to the combined effects of two or more features that jointly impact predictions (useful as feature combinations). Given a simple example, we consider the function $f(\boldsymbol{x}) = x^{(0)} + \exp(x^{(1)} + x^{(2)}) + \ln|x^{(3)}| + |x^{(4)}|^{-1} + |x^{(5)}|^{-1}$, where $\{x^{(i)}\}$ are *i.i.d.* uniformly sampled from $(10^{-4}, 1]$. It is worth noting that $x^{(3-5)}$ are important features, and $(x^{(1)}, x^{(2)})$ are features with interaction as their individual contributions to the outcome depend on the values of each other. In order to assess feature interactions, in tree models, co-occurrence [13] is typically considered as an interaction effect, while in neural networks, analyzing network weights [27] is used to evaluate interaction effects (similar to second-order derivatives). For efficiency, we employ tree-based models to extract interaction effects and the detailed explanations can be seen in Appendix.

We use LightGBM [28] to fit $f(\boldsymbol{x})$ and regard the frequency of two features simultaneously appear in a tree path in the tree-based model as a measure of interaction strength [13]. In Fig. 1, we observe that the important features often induce spurious interactions (large interaction strength but not interacting in the function), hindering us from extracting useful candidate feature combinations. However, the effective interaction $(x^{(1)}, x^{(2)})$ still stands out prominently in the sub-space $\{\mathcal{I} \times \mathcal{I} : \mathcal{I} = \{x^{(i)}\}_{0 \le i \le 2}\}$. Locally, with minor variation in feature importance, the higher interaction strength implies the higher likelihood of representing the effective interaction. This motivates us to partition the huge feature combination space into small spaces, preserving the most impactful feature combinations in these spaces.

4.2 Reduction in the Feature Dimension via Locally Interacted Features

Space Allocation For a dataset, $\{\boldsymbol{x}, y\}$, with d features ($\boldsymbol{x} \in \mathbb{R}^d$) and corresponding target y, we deploy the tree-based model [28] to fit a base model $h : \boldsymbol{x} \to y$ to obtain the basic characteristics of the given d features. With the model h, we can obtain the feature importance vector $F \in \mathbb{R}^d$ and the interaction strength matrix $I \in \mathbb{R}^{d\times d}$ (second-order here for simplicity, higher-order is also applicable). The term F_i represents the frequency that $x^{(i)}$ is used as a splitting node, and $I_{i,j}$ indicates the frequency that two features $(x^{(i)}, x^{(j)})$ appear on the same tree path (similar definition for triple features). Let $\mathcal{I}$ denote the sorted indices of the feature importance vector F, *i.e.*, $F_{\mathcal{I}[1]} \le F_{\mathcal{I}[2]} \cdots \le F_{\mathcal{I}[d]}$. Subsequently, we segment the sorted features into τ equally sized intervals based on the sorted indices $\mathcal{I}$:

$$\{\mathcal{I}_i : |\mathcal{I}_i| = \lceil d/\tau \rceil\}_{i\in[1,\tau]}. \tag{4.2}$$

Then the features falling within the same interval have a relatively small variation in their feature importance. With these intervals, we divide the second-order feature combination space into τ^2 small spaces (*i.e.*, $\{R_{i,j} = \mathcal{I}_i \times \mathcal{I}_j\}_{1\le i,j\le\tau}$). Consequently, features exhibiting significant interaction strength within each space can be extracted to form the candidate feature set.

Candidate Selection In these spaces $R_{i,j}$, the feature pairs exhibiting stronger interaction strength are more likely to generate the effective feature combinations. Hence, within each space $R_{i,j}$, we select $M_{i,j}$ feature pairs (each pair denoted as C) to generate new feature combinations based on their interaction strength, arranging them from the highest to the lowest. Meanwhile, we generate more feature combinations in the space $R_{i,j}$ if its overall interaction strength is strong. We use $S_{i,j}$ to denote the overall interaction strength, $S_{i,j} = \sum_{k_1,k_2} I_{k_1,k_2}\ \forall k_1 \in \mathcal{I}_i, \forall k_2 \in \mathcal{I}_j$, where I_{k_1,k_2} denotes interaction strength in the space $\{R_{i,j} = \mathcal{I}_i \times \mathcal{I}_j\}$. Then $M_{i,j}$ is calculated as follows:

$$M_{i,j} = \left\lfloor \frac{S_{i,j}}{\sum_{i,j} S_{i,j}} \cdot d^2 \cdot \eta \right\rfloor, \tag{4.3}$$

where η denotes the reduction ratio of whole feature combination space size d^2. For small datasets, we keep η around 5%, while for large datasets, we maintain η between 0.01% and 0.5% (such as the Traffic dataset with 862 variates). Upon the selected feature pairs set $\mathcal{C}$, whose element $C \in \mathcal{C}$ denotes a tuple of locally interacted features, we can generate the reduced candidate feature set via $\bigcup_{C\in\mathcal{C}} \mathcal{A}(C)$, with size reduced from $\mathcal{O}(qd^2)$ to $\mathcal{O}(\eta \cdot qd^2)$.

Discussion With such a partition and allocation strategy, the generated candidate features contain not only feature pairs with strong individual importance but also those with weaker individual importance yet significant interaction strength. As a result, our approach effectively reduces the search space while ensuring potential effectiveness and feature diversity, thereby enhancing downstream performance.

Algorithm 1 OpenFE++

Input: Training and validation data $\mathcal{D}_{tr}, \mathcal{D}_{val}$. Data dimension d, num of intervals τ, max order k, reduction ratio for each order $[\eta_1, ..., \eta_k]$.
Output: Generated feature set $\mathcal{S}$.

```
Init candidate feature set S = ∅.
Obtain T_lag or T_period according to Eq. (4.6) and Eq. (4.7).
Build model h : x → y on D_tr and extract F ∈ R^d from h.
for o ∈ {1, ··· , k} do
    Extract I ∈ R^{d^o} and allocate τ^o small spaces.
    Select feature combinations C ∈ C according to Eq. (4.3)
    with the reduction ratio η_o.
    S = S ∪ ⋃_{C∈C} A(C) ; /* Apply T_lag/T_period for TS */
Return: S = OpenFE(D_tr, D_val, S).
```

4.3 Reduction in the Temporal Dimension via Representative Lagged Periods

For time-series data, the temporal features capture characteristics over a specific period of time T. However, the brute-force enumeration of sequence length further increases the algorithm's complexity and overlooks the temporal nature of time-series data, potentially hurting the interpretability and generalization capability of the generated features. Hence, we advocate for the adoption of lagged periods to generate effective and interpretable features while reducing complexity.

Lagged Periods For time-series data, the interaction between features plays a crucial role, often demonstrating a lagged influence rather than an immediate effect. The lagged effect of two variables with a lagged period T is defined as follows, where $\sigma(\cdot)$ denotes the standard deviation:

$$Q_t^{(i,j)}(T) = \frac{\mathrm{Cov}\left(\boldsymbol{x}^{(j)}_{t-L+1-T:t-T}, \boldsymbol{x}^{(i)}_{t-L+1:t}\right)}{\sigma\left(\boldsymbol{x}^{(j)}_{t-L+1-T:t-T}\right)\cdot\sigma\left(\boldsymbol{x}^{(i)}_{t-L+1:t}\right)} \tag{4.4}$$

Consequently, taking into account the representative lagged characteristics also yields effective candidate time windows $\mathcal{T}_{lag}$ for feature generation. For the lagged effects, we utilize an extension of Wiener-Khinchin theorem [29, 30] to estimate the lagged effects between two sequences quickly:

$$\left\{Q_t^{(i,j)}(T)\right\}_{T=0}^{L-1} = \frac{1}{L}\mathcal{F}^{-1}\left(\mathcal{F}\left(\boldsymbol{x}^{(j)}_{t-L+1:t}\right)\odot\overline{\mathcal{F}\left(\boldsymbol{x}^{(i)}_{t-L+1:t}\right)}\right) \tag{4.5}$$

where $\mathcal{F}$ denotes the Fast Fourier Transform, $\mathcal{F}^{-1}$ denotes its inverse, $\odot$ represents the element-wise product, and the $\bar{\cdot}$ denotes the conjugate operation. Based on Eq. (4.5), we can calculate all the lagged

Table 1: Results on tabular task. Best results are in **bold**, and underline denotes the second best results. We repeat each experiment 10 times. '/' and '-' signifies not applicable to corresponding metric or dataset, and time represents the minutes taken to generate features. ✗ denotes exceeding the time limit (24hours).

Dataset	CA		MI		ME		TE		BR		DI		NO		VE		JA		CO		Rankings
Metric	RMSE ↓	Time ↓	RMSE ↓	Time ↓	RMSE ↓	Time ↓	AUC ↑	Time ↓	AUC ↑	Time ↓	AUC ↑	Time ↓	AUC ↑	Time ↓	AUC ↑	Time ↓	Acc. ↑	Time ↓	Acc. ↑	Time ↓	1^{st} count ↑
Base	0.432	/	0.744	/	1128.4	/	0.671	/	0.756	/	0.731	/	0.996	/	0.925	/	0.721	/	0.969	/	0
FCTree	0.440	2.3	0.744	1378	1096.4	6.9	0.670	5.7	0.751	330	0.731	6.8	0.996	11	0.927	74	0.721	40	0.971	160	0
SAFE	-	-	-	-	-	-	0.674	5	0.749	6.0	0.729	0.9	0.996	1.3	0.926	10	-	-	-	-	0
AutoFeat	0.444	0.2	0.744	23	7171.9	20.6	0.672	32.1	0.747	573	0.732	37	0.994	49	0.925	535	0.721	354	0.968	1284	0
AutoCross	-	-	-	-	-	-	0.667	101	0.765	1078	0.727	169	0.994	148	0.921	146	-	-	-	-	0
FETCH	0.430	98.1	✗		1130.6	1202	0.673	150	✗		0.731	241	0.996	325	0.927	1417	0.720	528	✗		0
OpenFE	0.421	0.1	0.744	47	982.0	0.4	0.680	1.3	0.786	28	0.888	1.9	**0.997**	5.8	**0.928**	6.8	0.729	5.1	0.974	82.2	2
Ours (k=2)	0.412	0.2	0.742	8.5	**979.7**	0.5	0.682	0.5	**0.803**	3.1	**0.892**	0.7	**0.997**	0.7	0.927	1.1	**0.730**	1.4	0.977	13.8	5
Ours (k=3)	**0.410**	0.3	**0.739**	35	1119.6	2.3	**0.683**	0.5	**0.803**	41	**0.892**	1.0	0.996	2.8	0.927	1.2	0.728	1.6	**0.978**	23.8	**6**

coefficients with the delay value $T \in [0, L-1]$ in $\mathcal{O}(L \log L)$. With $Q_t^{(i,j)}(T)$, we take summation across all variables (i, j) and time t, and get the K most significant lagged periods $\mathcal{T}_{lag}$:

$$\mathcal{T}_{lag} = \underset{0<T<L}{\text{argTopK}} \sum_{i,j,t} Q_t^{(i,j)}(T). \tag{4.6}$$

With $|\mathcal{T}_{lag}| = K$, we can reduce the size of second-order features from $\mathcal{O}(qLd^2)$ to $\mathcal{O}(\eta \cdot qKd^2)$. In addition, the lagged effects can naturally capture the influence of the past observations on the current state and capture the seasonality patterns. These characteristics directly provide a comprehensive understanding of the underlying temporal dynamics, thereby aiding us in efficiently generating effective, generalizable and interpretable features. If there is no clear lagged effects, we provide an alternative reduction method based on periodicity.

Periodic Properties Periodicity is a crucial characteristic of time series data, with short and long periods representing the micro and macro features of the time series. Motivated by this, we employ Fast Fourier Transform ($\mathcal{F}$) to transform $\boldsymbol{x}_{t-L+1:t}$ into the frequency domain, extract the K most dominant frequencies based on the average amplitude (Amp) and translate into the periods $\mathcal{T}_{period} = \{\lceil L/f_i \rceil\}_{1 \le i \le K}$:

$$\mathbf{A} = \text{Avg}\left(\text{Amp}\left(\mathcal{F}(\boldsymbol{x}_{t-L+1:t})\right)\right), \{f_1, ..., f_K\} = \underset{f_* \in \{1,\ldots,\lfloor \frac{L}{2} \rfloor\}}{\text{argTopK}} (\mathbf{A}). \tag{4.7}$$

By leveraging the meaningful $\mathcal{T}_{period}$, we can conduct feature generation across different scales, covering different scopes and improving the interpretability of features.

The selection of these two sets of time windows $\mathcal{T}_{lag}$ and $\mathcal{T}_{period}$ holds different practical significance. When dealing with data that exhibits clear lagged effects, using $\mathcal{T}_{lag}$ yields better performance as it reveals the underlying temporal dynamics. For data lacking distinct lagged characteristics, using $\mathcal{T}_{period}$ is more advantageous as it extracts information across various scales.

4.4 Implementation Our algorithm can efficiently identify the effective feature pairs in the expansion phase, and then utilizes pre-defined operators (see Appendix) to generate candidate features. Specifically, we employ LightGBM [28] to construct the tree-based model h for the given target. The frequency that a feature is used to make a split serves as the importance metric and the frequency that features simultaneously appear in a tree path as the interaction strength (discussion of feature interaction extracted by neural networks can be seen in Appendix). For high-order feature combinations ($k \ge 3$), we can construct the interaction strength $I \in \mathbb{R}^{d^k}$ in a similar way. During the reduction phase, we employ OpenFE [3] to extract effective features. Pseudocodes of OpenFE++ are shown in Algorithm 1.

5 Experiments

To demonstrate the performance of OpenFE++, we conduct extensive experiments across 14 commonly used datasets, including 10 popular tabular datasets and 4 well-known time-series datasets.

Datasets Our experiments are conducted across various domains of datasets, primarily divided into tabular tasks and time-series tasks. Specifically, for tabular prediction, we adopt 3 regression tasks and 7 classification tasks (details in Appendix), following the previous literature [3]. For time-series forecasting, we adopt the National Illness (ILI) [31], Weather [32], Electricity (ECL) [32] and Traffic [32] datasets, following previous long time-series forecasting works [33, 34, 35].

Benchmarks We compare with the following well-known automated feature generation benchmarks: (1) For tabular feature generation: **FCTree** [16], **SAFE** [13], **AutoFeat** [12], **AutoCross** [5], **FETCH** [17] and **OpenFE** [3]. (2) For temporal feature generation: **GPlearn** [10]. Furthermore, we adopt **LightGBM** [28], **TimesNet** [33], **PatchTST** [34], **iTransformer** [35], and **NLinear** [36] to assess the performance of the generated features in the downstream tasks.

5.1 Main Results

Tabular Task Table 1 presents the performance of OpenFE++ in comparison to other automated feature generation baselines across ten datasets. The number of generated features for different methods is kept consis-

Table 2: Results on time-series task. The background color reflects the degree that OpenFE++ ($\mathcal{T}_{lag}$) improves performance, and the best results are shown in **bold**. Imp Count represents the number of enhanced experiments.

Models		PatchTST		TimesNet		+ OpenFE++		iTransformer		+ OpenFE++		NLinear		+ OpenFE++	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ILI	24	0.6601	0.573	0.7471	0.624	0.6963	0.619	0.6531	0.561	0.6917	0.598	0.6601	0.551	**0.4664**	**0.442**
	36	0.6517	0.614	0.5260	0.553	0.4975	0.542	**0.4781**	0.527	0.5076	0.546	0.7204	0.611	0.5408	**0.505**
	48	0.6922	0.661	0.6729	0.651	0.7138	0.653	0.5789	0.599	0.5862	0.595	0.7069	0.644	**0.5617**	**0.551**
	60	0.7337	0.712	0.8996	0.804	0.8256	0.723	0.6729	0.659	0.6978	0.664	0.6488	0.625	**0.5961**	**0.568**
ECL	96	0.3173	0.398	0.3268	0.426	0.3476	0.442	0.2908	0.395	0.2781	0.390	0.2988	0.409	**0.2654**	**0.376**
	192	0.3456	0.414	0.4094	0.475	0.3697	0.453	0.3359	0.423	0.3150	0.412	0.3086	0.415	**0.3032**	**0.410**
	336	0.4157	0.450	0.4430	0.496	0.4375	0.488	0.3689	0.443	0.3568	0.437	0.3357	0.438	**0.3242**	**0.420**
	720	0.5038	0.516	0.4436	0.497	0.4605	0.503	0.4966	0.524	0.4642	0.507	0.3268	0.427	**0.2904**	**0.402**
Traffic	96	0.1749	0.253	0.1545	0.250	0.1542	0.245	0.1503	0.231	**0.1408**	**0.223**	0.2554	0.340	0.1591	0.248
	192	0.1643	0.246	0.1673	0.261	0.1553	0.244	0.1458	0.228	**0.1399**	**0.221**	0.2237	0.309	0.1592	0.245
	336	0.1686	0.254	0.1684	0.260	0.1576	0.252	0.1440	0.232	**0.1422**	**0.229**	0.2122	0.301	0.1563	0.247
	720	0.2008	0.275	0.1951	0.289	0.1806	0.277	0.1606	0.250	**0.1598**	**0.247**	0.2330	0.320	0.1740	0.263
Weather	96	**1.27e-3**	**2.60e-2**	1.48e-3	2.84e-2	1.48e-3	2.84e-2	1.34e-3	2.69e-2	1.29e-3	**2.60e-2**	1.30e-3	2.72e-2	1.32e-3	2.65e-2
	192	1.60e-3	2.99e-2	1.76e-3	3.11e-2	1.64e-3	3.03e-2	1.60e-3	**2.98e-2**	**1.59e-3**	**2.98e-2**	1.97e-3	3.41e-2	1.61e-3	3.04e-2
	336	1.79e-3	3.19e-2	1.80e-3	3.19e-2	1.76e-3	**3.03e-2**	**1.72e-3**	3.12e-2	**1.72e-3**	3.12e-2	1.91e-3	3.31e-2	1.87e-3	3.28e-2
	720	2.22e-3	3.58e-2	2.26e-3	3.60e-2	2.19e-3	3.53e-2	2.19e-3	3.55e-2	2.16e-3	**3.51e-2**	2.27e-3	3.61e-2	**2.12e-3**	**3.51e-2**
Imp Count		-	-	-	-	12	12	-	-	11	11	-	-	15	16

tent to ensure a fair comparison (details in Appendix) and we use LightGBM as the base model for evaluation. From the table, it is evident that OpenFE++ achieves superior performance compared to other automated feature generation methods. Specifically, it demonstrates a significant reduction in running time and an average improvement of 5.1% in metrics over the base model. We credit this to the effectiveness of OpenFE++ in pruning a substantial size of interfering ineffective candidate features, thereby significantly accelerating the evaluation process and mitigating the risk of overfitting. For higher-order features ($k = 3$), previous works usually do not yield additional benefits and even result in intolerable evaluation time [3]. However, OpenFE++ enables us to achieve improved results in certain datasets without significantly increasing the time consumption.

Time-series Task Table 2 showcases the incremental performance of OpenFE++ on several representative long-term forecasting models with various prediction lengths. For ILI, we select 104 (i.e., 2 years) as the lookback length L with the prediction horizons $H \in \{24, 36, 48, 60\}$. For other datasets, we set the lookback length $L = 96$ with the prediction horizons $H \in \{96, 192, 336, 720\}$ [33]. In this experiment, we employ multivariate time-series to predict specified univariate sequences, and OpenFE++ provides additional features to enhance the model performance (with the exception of PatchTST [34], as it is channel independent). For *NLinear*, we incorporate an additional linear layer to integrate information from different channels.

The results demonstrate that the locally interacted features extracted from a tree-based model can also effectively enhance the performance of the neural networks in most experimental setups. Specifically, the generated features significantly improve the performance of *NLinear*, enabling it to approach the performance of SOTA transformer-based models and even achieve optimal results in certain instances. For efficiency, OpenFE++ allows automated feature generation to efficiently extract effective features from large-scale datasets (862 features in Traffic), which was previously intolerably time-consuming. Additionally, our generated features can be seamlessly integrated into the original model structure, with minor impact on the training and inference efficiency of the model.

5.2 Method Analysis

Reduction To evaluate the impact of the reduced candidate feature set via locally interacted features, we present experimental results in Fig. 2. In this experiment, we consider second-order features ($k = 2$) and divide the features into five intervals ($\tau = 5$) according to Eq. (4.2). Besides the reduction method **Interaction** shown in our Eq. (4.3), we compare it with two commonly used reduction methods: **Uniform**, which randomly selects the candidate feature combinations, and **Importance**, which retains important features for feature combinations.

As depicted in Fig. 2, a larger set of candidate features does not necessarily guarantees superior results, instead, a smaller and carefully curated candidate feature set (approximately 5%) even yields better performance. For instance, in the BR dataset, a reduction ratio of 1% achieves the best performance, but increasing candidate feature set will increase the difficulty of feature selection and raise the risk of overfitting. This

Table 3: Forecasting performance with additional temporal features generated by different methods. The results are averaged on four different forecasting horizons and the best are shown in **bold**.

Method	ILI (7)		ECL (321)		Traffic (862)		Weather (21)		Improvement	
Metrics	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
NLinear	0.6841	0.608	0.3172	0.422	0.2311	0.318	1.86e-3	3.26e-2	-	-
+ GPlearn	0.7639	0.662	0.3023	0.404	0.2015	0.288	2.03e-3	3.24e-2	-0.7%	+1.3%
+ Ours ($\mathcal{T}_{period}$)	0.5519	0.526	0.2967	0.404	0.1799	0.272	**1.65e-3**	**3.02e-2**	+14.8%	+10.0%
+ Ours ($\mathcal{T}_{lag}$)	**0.5413**	**0.517**	**0.2958**	**0.402**	0.1622	0.251	1.73e-3	3.12e-2	**+16.2%**	**+11.3%**
iTransformer	0.5958	0.586	0.3731	0.446	0.1502	0.235	1.71e-3	3.09e-2	-	-
+ GPlearn	0.6345	0.619	0.3749	0.448	0.1495	0.235	1.70e-3	3.08e-2	-1.5%	-1.4%
+ Ours ($\mathcal{T}_{period}$)	0.5999	0.587	0.3598	0.439	0.1514	0.237	1.69e-3	3.06e-2	+0.8%	+0.4%
+ Ours ($\mathcal{T}_{lag}$)	0.6208	0.601	0.3535	0.436	**0.1457**	**0.230**	1.69e-3	3.05e-2	**+1.3%**	**+0.8%**

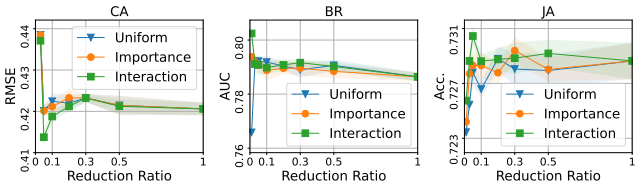


Figure 2: Evaluation of different reduction ratios (η) across three representative datasets (CA, BR, JA). Each dataset involves three different allocation methods (Uniform, Importance and Interaction). The background color corresponds to the interval of standard deviation.

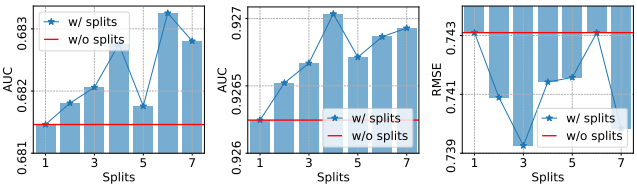


Figure 3: Evaluation of different splits count (τ) across three representative datasets (TE, VE, MI).

implies that an excessively large candidate set may conceal useful features, but OpenFE++ can effectively extract these features, leading to enhancements in both efficiency and effectiveness. Moreover, our reduction method demonstrates better performance compared to using a small number of important features for feature combination (Importance), which indicates that the combinations of important features may not necessarily yield stronger incremental performance.

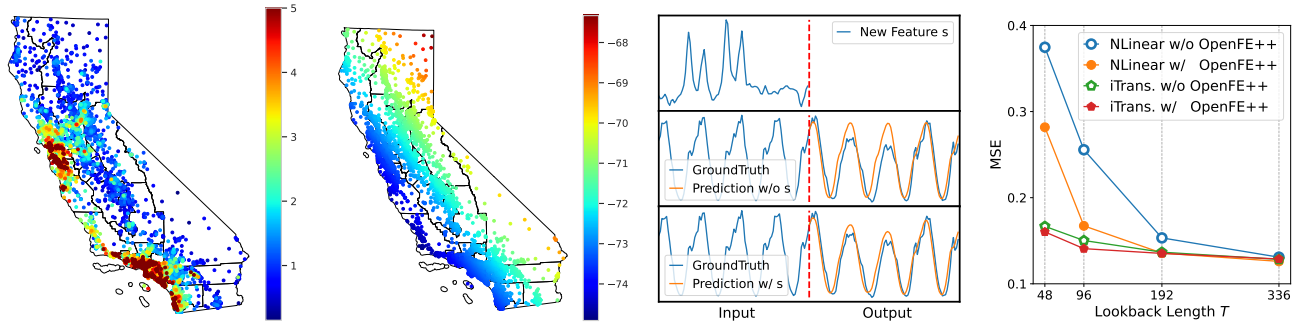
Space Allocation To demonstrate the effect of allocated small spaces in extracting effective feature combinations, we adjust the number of intervals (τ) to assess the forecasting performance of the generated features. We conduct experiments with $k = 3$ and select hyper-parameter η on the validation set. The results depicted in Fig. 3 reveal that forecasting performance exhibits a gradual improvement as τ increases. Here, $\tau = 1$ indicates the absence of allocating small spaces and the top interacted features are selected to generate candidate features. The enhanced performance as τ grows, with less variation in feature importance in smaller spaces, signifies its effectiveness in identifying effective interacted features within the localized spaces.

Temporal Feature Generation To evaluate the forecasting performance of the generated temporal features with our reduced temporal windows, we conduct experiments to generate temporal features with $\mathcal{T}_{lag}$ or $\mathcal{T}_{period}$ and compare them with GPlearn [10]. For GPlearn, the temporal window is randomly sampled from a maximum lookback window L. Table 3 shows that OpenFE++ with $\mathcal{T}_{lag}$ or $\mathcal{T}_{period}$ can both improve the forecasting performance and significantly outperforms GPlearn. Moreover, OpenFE++ achieves an average 13.8% reduction in forecasting errors for the linear model and 1% for iTransformer [35], indicating that the feature generation continues to provide benefits for complex deep learning models. Full results can be found in Appendix.

5.3 Experimental Analysis

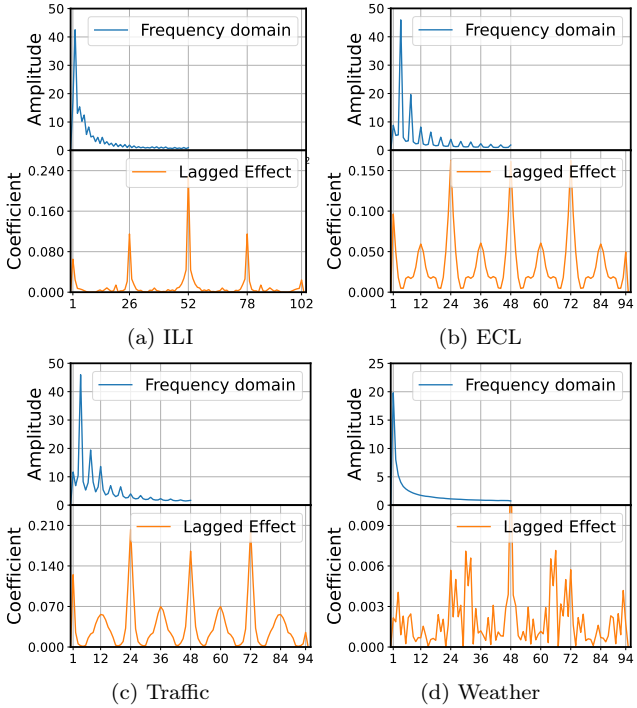
Case of Generated Feature To obtain a comprehensive understanding of the generated features, we give the visualization of the target label (median house value) of CA dataset in Fig. 4(a) (the color for each data point corresponds to the value at its specific geographical location) and corresponding top-1 ranked generated feature ($\sqrt{\text{longitude}} + \text{longitude} + \text{latitude}$) in Fig. 4(b). It is evident that the price of a house is closely correlated with its geographical location, with values being higher when the it is situated closer to the coastline. OpenFE++ identifies the interaction between latitude and longitude, and the generated feature exhibits distinct values based on distance to the coastline.

For the time-series forecasting task, we give the visualization of the generated feature and forecasting results of the Traffic dataset in Fig. 4(c). This dataset consists of traffic flow from 862 sites (from var_0 to var_{861}), and OpenFE++ has identified an effective feature s (ts_delta($var_{267}, 72$)) that significantly enhances predictive performance. With this feature, it is evi-



(a) CA Label (b) CA Top-1 Feature (c) Traffic Prediction (d) Traffic Performance

Figure 4: Analysis of the generated features. (a) The distribution of the target label across various geographical locations in the CA dataset. (b) The distribution of the top-1 generated feature ($k = 3$) in the CA dataset. (c) Visualization of the generated feature s, along with the corresponding forecasting results using the NLinear model without (w/o) or with (w/) the generated features s. (d) Performance of different models with (w/) or without (w/o) OpenFE++ under various lookback length L.



(a) ILI (b) ECL (c) Traffic (d) Weather

Figure 5: Visualization of the frequency domain and the lagged effects across four datasets.

dent that a linear model (NLinear [36]) also performs remarkably well. As depicted in Fig. 4(d), the generated features effectively enhance predictive performance within a limited lookback window. In this scenario, the cost of incorporating additional features is lower than directly extending the lookback window, as the latter will moderately increase the model's parameter count.

Selection of $\mathcal{T}_{lag}$ or $\mathcal{T}_{period}$ Table 3 demonstrates that the features generated by $\mathcal{T}_{lag}$ perform better on ILI, ECL and Traffic datasets, whereas those generated by $\mathcal{T}_{period}$ perform better on the Weather dataset. To illustrate the disparities between the two approaches and to demonstrate how to select the most suitable one, we present visualizations of the frequency domain and the coefficients representing the lagged effects across four datasets in Fig. 5. The spectrograms of the first three datasets often depict a jagged pattern, and there are pronounced lagged effects that serve as crucial characteristics. Employing $\mathcal{T}_{lag}$ for feature generation in such instances consistently yields significantly improved results. However, in the case of the Weather dataset, no apparent lagged effect is observed, with the strongest coefficient hovering around 0.01. Consequently, utilizing $\mathcal{T}_{lag}$ in this scenario fails to sufficiently extract effective features, whereas employing $\mathcal{T}_{period}$ facilitates the extraction of diverse information across various scales, ultimately resulting in superior performance.

6 Conclusion & Limitation

In this paper, we propose a novel method called OpenFE++ designed to efficiently generate effective features for both tabular and time-series tasks. Within OpenFE++, we construct the candidate feature set using the locally interacted features to significantly reduce the search space, and generate temporal features via representative lagged periods to enhance their effectiveness and interpretability. The experimental results have demonstrated our superior performance over other automated feature generation methods in both efficiency and effectiveness. OpenFE++ acquires the locally interacted features from a trained model, thereby enhancing effectiveness while improving efficiency. However, there are exceptional cases where some feature interactions may not be captured by the trained model, then a larger feature set is required to encompass these interactions.

Acknowledgements

The authors are supported in part by the National Natural Science Foundation of China Grant 62161146004.

References

[1] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: an overview from a database perspective. *IEEE Transactions on Knowledge and data Engineering*, 8(6):866–883, 1996.

[2] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[3] Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. Openfe: Automated feature generation with expert-level performance. In *International Conference on Machine Learning*, pages 41880–41901. PMLR, 2023.

[4] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1161–1170, 2019.

[5] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1936–1945, 2019.

[6] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7. 2017.

[7] Alex Daniel Reneau, Jerry Yao-Chieh Hu, Ammar Gilani, and Han Liu. Feature programming for multivariate time series prediction. In *International Conference on Machine Learning*, pages 29009–29029. PMLR, 2023.

[8] Stefan Meisenbacher, Marian Turowski, Kaleb Phipps, Martin Rätz, Dirk Müller, Veit Hagenmeyer, and Ralf Mikut. Review of automated time series forecasting pipelines. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(6):e1475, 2022.

[9] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. Tsfel: Time series feature extraction library. *SoftwareX*, 11:100456, 2020.

[10] Ryan J. McKay, Trevor Stephens, and Lee Spector. gplearn: Genetic programming for symbolic regression. `https://github.com/trevorstephens/gplearn`, 2018.

[11] Sohail Asghar and Khalid Iqbal. Automated data mining techniques: A critical literature review. In *2009 international conference on information management and engineering*, pages 75–79. IEEE, 2009.

[12] Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 111–120. Springer, 2020.

[13] Qitao Shi, Ya-Lin Zhang, Longfei Li, Xinxing Yang, Meng Li, and Jun Zhou. Safe: Scalable automatic feature engineering framework for industrial tasks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1645–1656. IEEE, 2020.

[14] Ronald Aylmer Fisher. Statistical methods for research workers. In *Breakthroughs in statistics: Methodology and distribution*, pages 66–70. Springer, 1970.

[15] Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

[16] Wei Fan, Erheng Zhong, Jing Peng, Olivier Verscheure, Kun Zhang, Jiangtao Ren, Rong Yan, and Qiang Yang. Generalized and heuristic-free feature construction for improved accuracy. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 629–640. SIAM, 2010.

[17] Liyao Li, Haobo Wang, Liangyu Zha, Qingyi Huang, Sai Wu, Gang Chen, and Junbo Zhao. Learning a data-driven policy network for pre-training automated feature engineering. In *The Eleventh International Conference on Learning Representations*, 2022.

[18] Shuo Yu, Hongyan Xue, Xiang Ao, Feiyang Pan, Jia He, Dandan Tu, and Qing He. Generating synergistic formulaic alpha collections via reinforcement learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

[19] Dongjie Wang, Yanjie Fu, Kunpeng Liu, Xiaolin Li, and Yan Solihin. Group-wise reinforcement feature generation for optimal and explainable representation space reconstruction. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1826–1834, 2022.

[20] Dongjie Wang, Meng Xiao, Min Wu, Yuanchun Zhou, Yanjie Fu, et al. Reinforcement-enhanced autoregressive feature transformation: Gradient-steered search in continuous space for postfix expressions. *Advances in Neural Information Processing Systems*, 36, 2024.

[21] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631, 2013.

[22] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

[23] Sanjay Purushotham, Martin Renqiang Min, C-C Jay Kuo, and Rachel Ostroff. Factorized sparse learning models with interpretable high order feature interactions. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 552–561, 2014.

[24] David R Cox. Interaction. *International Statistical Review/Revue Internationale de Statistique*, pages 1–24,

1984.
[25] Michael G Moore. Three types of interaction, 1989.
[26] Michael Tsang, Dehua Cheng, Hanpeng Liu, Xue Feng, Eric Zhou, and Yan Liu. Feature interaction interpretability: A case for explaining ad-recommendation systems via neural interaction detection. *arXiv preprint arXiv:2006.10966*, 2020.
[27] Michael Tsang, Dehua Cheng, and Yan Liu. Detecting statistical interactions from neural network weights. *arXiv preprint arXiv:1705.04977*, 2017.
[28] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.
[29] Norbert Wiener. Generalized harmonic analysis. *Acta mathematica*, 55(1):117–258, 1930.
[30] Lifan Zhao and Yanyan Shen. Rethinking channel dependence for multivariate time series forecasting: Learning from leading indicators. *arXiv preprint arXiv:2401.17548*, 2024.
[31] Centers for Disease Control and Prevention. National, regional, and state level outpatient illness and viral surveillance. `https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html`, 2020. Accessed: 2020-06-30.
[32] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in neural information processing systems*, 34:22419–22430, 2021.
[33] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The eleventh international conference on learning representations*, 2022.
[34] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.
[35] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*, 2023.
[36] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023.