## Appendix 1. The mechanism of SDK.

**Response:** "Unfold" operation in formula (3) and (4) is illustrated in a new way, which could help readers better understand the meaning of this operation. Besides, theoretical analysis between SDK and dynamic convolution is proposed to illustrate the advantages of SDK, and a case study is conducted to further verify the performance of SDK.

**The description of SDK.** Earlier studies overlook the unique characteristics of sales time series, as sales time series is a dynamic process, and will be influenced by many factors apart from the timeline, so different local spatiotemporal areas may contain different dependent correlations. Our main idea is that dynamic kernel functions should be designed for different local areas. In addition, different with existing researches related with dynamic kernel, which is mainly applied in computer vision (CV) domain, a novel SDK is designed for solving MTS predictions and focuses on assigning specific weight for each time point of input $X$.

For the input multivariate time series (MTS) $X_{1\sim t}^{1\sim N}$, assume $L$ CNN filters are assigned, for the $l^{th}$ filter, the kernel size is $K * N$. We adopted two convolutional layers with kernel size $1 * 1$ to provide a dynamic weight for each "cell" $X_j^i$ in matrix $X_{1\sim t}^{1\sim N}$. The design of specific weight for each "cell" is shown in formula (1):

$$\begin{aligned} W_{1\sim t+(2k-1)//2}^{1\sim N} =& conv2(conv1(X_{1\sim t}^{1\sim N}, ks=1)), \\ & ks=1, ph=K//2, str=1) \end{aligned} \tag{1}$$

where $W_{1\sim t+(2k-1)//2}^{1\sim N}$ is a weight matrix with $1 \sim N$ columns and $1 \sim t+(2k-1)//2$ rows, it extends both columns and rows of $X_{1\sim t}^{1\sim N}$ by $(2k-1)//2$ based on the $ph$ = $padding\_height$ and $str$ = stride. The purpose of assigning $(2K-1)//2$ for columns is to ensure the height of SDK convolutional output is fixed as $t$. conv1 and conv2 are two convolutional layers to assign weight for each "cell" and enhance non-linear representations. In the next step, $W_{1\sim t+(2k-1)//2}^{1\sim N}$ and $X_{1\sim t}^{1\sim N}$ will be divided into $t$ $K \times N$ matrices $W^{(K\times N)\times t\times 1}$ and $X^{(K\times N)\times t\times 1}$ separately by assigning kernel size as $K \times N$. The expression could be seen as below:

$$\begin{aligned} W &= W^{(K\times N)\times t\times 1} \\ &= divW(W_{1\sim t+(2k-1)//2}^{1\sim N}, K\times N, 0, 1) \end{aligned} \tag{2}$$

$$\begin{aligned} X &= X^{(K\times N)\times t\times 1} \\ &= divX(X_{1\sim t}^{1\sim N}, K\times N, (2K-1)//2, 1) \end{aligned} \tag{3}$$

where $divW$ is a function to divide $W_{1\sim t+(2k-1)//2}^{1\sim N}$ into $t$ $K \times N$ matrices $W^{(K\times N)\times t\times 1}$ by assigning kernel size as $K \times N$, padding as 0 and stride as 1. $divX$ is a function to divide $X_{1\sim t}^{1\sim N}$ into $t$ $K \times N$ matrices $X^{(K\times N)\times t\times 1}$ by assigning kernel size as $K \times N$, padding as $(2K-1)//2$ and stride as 1. The convolution operation with kernel $K \times N$ could be seen in formula (4):

$$O_{1\sim t}^l = [\sum_{k=1}^{K\times N} [W \odot X](k,i,j) \; for \; i \; in \; t \; and \; j=1] \tag{4}$$

where $O_{1\sim t}^l \in R^{t\times 1}$ is the output of $l^{th}$ filter with kernel size as $K \times N$, $\odot$ is dot product between $W(k,i,j)$ and $X(k,i,j)$. In formula (5), $O_{1\sim t}^L \in R^{t\times L}$is the stack of all $O_{1\sim t}^l$ to form a new matrix with row as $t$ and column as $L$.

$$O_{1\sim t}^L = Stack_{l=0}^L(O_{1\sim t}^l) \tag{5}$$

Formula $(1) \sim (5)$ could be defined as a mapping function: $O_{1\sim t}^L = F(X_{1\sim t}^{1\sim N})$. We adopt 4-layers mapping functions $F$ to model future-vision based local correlations among different time series at time $t+1$, $t+2$, $t+3$ and $t+4$, which could be seen in formula (6):

$$\begin{aligned} (O_{1\sim t}^L)_{t+1} &= F_0(X_{1\sim t}^{1\sim N}) \\ (O_{1\sim t}^L)_{t+2} &= F_1((O_{1\sim t}^L)_{t+1}) \\ (O_{1\sim t}^L)_{t+3} &= F_2((O_{1\sim t}^L)_{t+2}) \\ (O_{1\sim t}^L)_{t+4} &= F_3((O_{1\sim t}^L)_{t+3}) \end{aligned} \tag{6}$$

where $(O_{1\sim t}^L)_{t+1}$,$(O_{1\sim t}^L)_{t+2}$,$(O_{1\sim t}^L)_{t+3}$ and $(O_{1\sim t}^L)_{t+4}$ are construal of local dependent correlations of all time series variables at time $t+1$, $t+2$ ,$t+3$ and $t+4$. $F_0$, $F_1$,$F_2$ and $F_3$ are 4-layers mapping functions based on SDK component.

**Compared with Dynamic Convolution**. For a MTS $X_{1\sim t}^{1\sim N}$, one of its $K \times N$ matrix is $X_{i\sim i+K}^{1\sim N}$ $(i \in t)$.

Dynamic convolution assigns $D$ kernels with size as $K \times N$, in which the $dth$ kernel is $W_d(X_{1\sim t}^{1\sim N})$, and attention weight of the $L \times N$ matrix is $\pi_d(X_{i\sim i+K}^{1\sim N})$. Then convolutional operation of $X_{i\sim i+K}^{1\sim N}$ could be expressed as:

$$O_i = \pi_d \times \sum W_d(X_{1\sim t}^{1\sim N}) \odot X_{i\sim i+K}^{1\sim N} \tag{7}$$

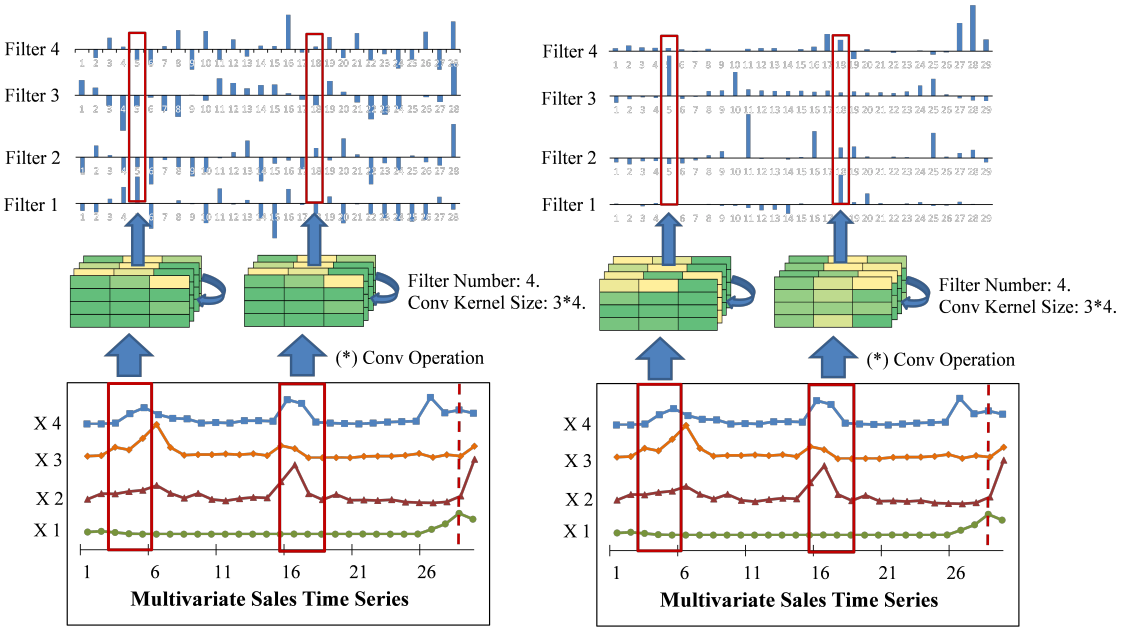
SDK assigns specific weight for each time point, and make convolution operations based on aggregation of all time points in the $K \times N$ matrix. Assume the weight-specific matrix is $W(X_{i\sim i+K}^{1\sim N})$, which could be derived from formula (1), then SDK convolution operation could be expressed as:

$$O_i = \sum W(X_{i\sim i+K}^{1\sim N}) \odot X_{i\sim i+K}^{1\sim N} \tag{8}$$

By comparing formula (7) and (8), we can find that SDK focuses on more microscopic level. It directly calculates weight for each time point, but dynamic convolution mainly focuses on relationships between global and local matrix, because it is mainly applied in computer vision domain. In the task of sales prediction, a time point is very important because it represents a specific product sales at that time. **Case Study of SDK**. A case study of comparing SDK and CNN can be seen in Figure 1. The top part of Figure 1 (a) and (b) represents the filter output after CNN and SDK convolutional operations. Compared with filter output based on dynamic convolution and traditional CNN, it is obvious that the output of SDK has a significant correlation with the changing patterns of original time series.

## Appendix 2. Algorithm description of SR

The description of SR can be seen in Algorithm 1. The main idea is to make co-integration test between a time series




(a) CNN (b) SDPANet

Figure 1: Running efficiency

with its nearest time series instead of traversing all time series. Line 3 defines status vector $f[N]$ and $pre[N]$. Lines 5 to 11 design a strategy to make co-integration between a time series with its nearest time series instead of traversing all time series. Line 10 defines status transfer function. The main strategy is that status vector $f[i]$ stores cumulative information for each subset with the increasing of $i$. $pre[i]$ stores the information of previous time series, which passes co-integration test with current time series $i$.

Line 12 to 20 describe the strategy to find all subsets in one loop. For the element at $k = X.size() - 1$, status vector $f[k]$ stores the number of elements in the subset, status vector $pre[k]$ stores the previous element in the subset. Thus backward tracking through $pre$ can obtain all elements in a subset.

## Appendix 3. The description of DC loss

**Response**: We simplify the description of DC loss, and further emphasize its value and our innovation in this research.

The DC loss can adjust the prediction results based on captured dynamic changes, and is mainly based on Dynamic Time Warping (DTW). Different with Mean Average Error (MAE), DTW mainly focuses on finding similar patterns between two time series, and is sensitive to dynamic changes. Assume at time $t$, the true and predicted ith time series are $Y_{t+1\sim t+4}$ and $\widetilde{Y}_{t+1\sim t+4}$. The two time series could generate a distance matrix $D^i_{t+1\sim t+4} \in R^{4\times 4}$ based on DTW loss. DTW can find the minimal distance path from (1 ,1) to (4, 4) of $D^i_{t+1\sim t+4}$ by constructing optimization association matrix $A$.

Matrix $A$ is mainly used to detect dynamic changes. In order to realize dynamic change alignment, we introduce time penalty matrix $\Omega \in R^{4\times 4}$, where $\Omega[m,n] = (m-n)^2/(4\times 4)$. The purpose of designing $\Omega$ is for a pair $(m,n)$ with high similarity in $A$, but if their time distance $\Omega[m,n]$ is high, which means there exists dislocation between them in the correlation of sequence shape, then a penalty will be added to enlarge the current loss for aligning time between $Y^i_{t+1\sim t+4}$ and $\widetilde{Y}^i_{t+1\sim t+4}$. Finally, the loss of Dynamic Change Detection and Alignment $L_{DC}$ is defined as:

$$L_{DC} = Sum(Mean(A \times \Omega, dim = 0))/(4 \times 4) \quad (9)$$

where $Mean(A \times \Omega, dim = 0)$ is to average each column of matrix $A \times \Omega$, and the output is $1 \times 4$ dimension vector. $Sum$ means to sum each element of the output vector.

## Appendix 4. Relations of the four components

The main function of SDK is to capture dynamic non-linear correlations among MTS. As seen in Figure 1 in supplementary materials, CNN weight-shared strategies and dynamic convolution (2021) fail to capture the changing patterns. Ablation test in Figure 5 shows that SDK outperformances CNN and dynamic convolution significantly, and the performance of SDK is more stable according to RSE and CORR.

The main function of SR is to capture dynamic linear correlations among MTS. Ablation test in Figure 5 also shows that removing SR (SDPANet-SR) will cause a poor performance, which indicates SR is an essential important component. Compared with AR (SDPANet-SR+AR), the improvement of SR is also significant. Because AR does not take into account linear correlations among different time series.

Algorithm 1: Co-Integration based Dynamic Programming

**INPUT:** Short-term Multivariate time series $X_{t-4\sim t}^{1\sim N}$.
**OUTPUT:** The Set of subgroups: $S = [S_1, S_2, \ldots, S_C]$.

$S$ = **Function** getSubset($X_{t-4\sim t}^{1\sim N}$):
  **Let** $X = X_{t-4\sim t}^{1\sim N}$;
  **Define** array $f[N]$ and $pre[N]$;
  # $f[N]$ **and** $pre[N]$ **are Status Vectors**.
  **For** $(i = 1; i < N; i++)$ :
    **Assign** $f[i] = 1$ and $pre[i] = 1$;
    **For** $(j = i - 1; j > 0; j - -)$ :
    # $j$ **Traverses from the nearest Location of** $i$.
      **If** (ci_ test $(X_{t-4\sim t}^{i}, X_{t-4\sim t}^{j})$ == true):
        $f[i] = f[j] + 1$;$pre[i] = j$;
        break;
  **While** ($X$ is not empty):
    **Define** $S_i$ as an empty list;
    $k$ = $X$.size() - 1;
    $Si.append(X_{t-4\sim t}^{k})$;
    **If** $(f[i] > 1)$:
      **While** $(k! = pre[k])$:
      **# Track all the members of** $S_i$ **from** $pre$**.**
        $k = pre[k]; Si.append(X_{t-4\sim t}^{k})$;
      $S$.append($S_i$);
      $X$.remove($S_i$);
    **Return** $S$;

$S$ = **getSubset**($X_{t-4\sim t}^{1\sim N}$);
**OUTPUT** $S$;

The relations between all components are based on a general framework (MLCNN, 2020). The four components have complementary advantages.

## Appendix 5. Performance evaluation

Seven state-of-the-art MTS predictions models are selected as baselines. The descriptions of each baseline could be seen as below:

- RNN-LSTM: RNN-LSTM is widely used in time series research because its design makes it suitable for processing and predicting important events with long intervals and delays in time series.
- WaveNet[1]: A sequence generation model proposed by DeepMind. It is mainly based on an extended causal convolutional layer, which allows it to deal with temporal sequences and long-term dependencies without a marked increase in model complexity.
- LSTNet[2]: Long and Short-term Time-series Network adopts a CNN-LSTM framework to extract local dependency patterns. It uses convolutional layer and recurrent layer to capture long-term dependency patterns. More importantly, it proposes a novel recurrent-skip layer to capture periodic properties in the input data for forecasting.
- MLCNN[3]: Multi-Level Construal Neural Network is a multi-task deep learning framework that improves predictive performance by fusing forecasting information from different future times. The framework uses a Convolutional Neural Network to extract multi-level abstract representations of the raw data, then models interactions between multiple predictive tasks and fuses their future visions through an Encoder-Decoder framework.
- DARNN[4]: The Dual-stage Attention-based Recurrent Neural Network uses a two-stage attention mechanism for multivariate time series predictions. The first stage learns the weights of input variables through a spatial perspective, and the second stage uses a temporal perspective to learn the weights of hidden states across all time steps.
- MTNet[5]: Memory Time series network (MTNet) is jointly trained by a large memory component, three separate encoders, and an autoregressive component. The memory component and attention mechanism store long-term correlation patterns through modelling historical data and make prediction results explainable.
- StemGNN [6]: It combines Graph Fourier Transform (GFT) which models inter-series correlations and Discrete Fourier Transform (DFT) which models temporal dependencies in an end-to-end framework. After passing through GFT and DFT, the spectral representations hold clear patterns and can be predicted effectively by convolution and sequential learning modules. Moreover, StemGNN learns inter-series correlations automatically from the data without using pre-defined priors.

Compared with StemGNN, SDPANet also obtains better results. Although StemGNN is very competitive compared with other baselines. Especially the performance of StemGNN in terms of CORR is very high, which indicates that StemGNN can accurately predict the trend of time series from the perspective of future vision. However, StemGNN seldom considers making optimization designs for improving the model capability in capturing dynamic changing patterns. Thus the variance in terms of RMAE and RSE is a little large. As introduced in StemGNN, latent correlation layer uses self-attention to generate the initial correlations between different time series, seldom consider the correlations between different time points.

## Appendix 6. Running efficiency

Running efficiency of each component is conducted in this section. SDPANet and its 4 variants are selected, and the description of all selected variants can be seen as below:
**SDPANet-SDK**: We remove the SDK module and replace it with the CNN module.
**SDPANet-HA**: We replace the Hierarchical Attention Component with shared-main LSTM of MLCNN.

[1]https://github.com/ibab/tensorflow-wavenet
[2]https://github.com/laiguokun/LSTNet
[3]https://github.com/smallGum/MLCNN
[4]https://github.com/fanyun-sun/DARNN
[5]https://github.com/Maple728/MTNet
[6]https://github.com/microsoft/StemGNN

Table 1: Performance of the proposed SDPANet model on Galanz and Cainiao dataset

| | | Galanz | | Cainiao | |
|---|---|---|---|---|---|
| Metrics | Method | GW1∼GW10 | GWN | CW1∼CW5 | CWN |
| R-MAE | LSTM | 0.708 ± 0.077 | 1.113 | 1.183 ± 0.086 | 1.169 |
| | WaveNet | 1.025 ± 0.262 | 1.103 | 1.367 ± 0.076 | 1.342 |
| | MLCNN | 0.708 ± 0.065 | 1.110 | 1.231 ± 0.219 | 1.232 |
| | LSTNet | 0.656 ± 0.151 | 1.087 | 1.229 ± 0.094 | 1.21 |
| | MTNet | 0.699 ± 0.125 | 1.143 | 1.492 ± 0.196 | 1.497 |
| | DARNN | 0.628 ± 0.151 | 1.156 | 1.430 ± 0.079 | 1.453 |
| | StemGNN | 0.735 ± 0.32 | 1.10 | 1.337 ± 0.251 | 1.301 |
| | SDPANet | **0.389 ± 0.035** | **0.522** | **0.904 ± 0.058** | **0.912** |
| RSE | LSTM | 2.534 ± 0.938 | 1.034 | 1.083 ± 0.074 | 1.012 |
| | WaveNet | 1.853 ± 0.554 | 1.012 | 2.401 ± 0.332 | 1.843 |
| | MLCNN | 0.914 ± 0.91 | 1.241 | 1.563 ± 0.074 | 1.659 |
| | LSTNet | 1.071 ± 0.11 | 1.034 | 1.255 ± 0.101 | 0.979 |
| | MTNet | 1.224 ± 0.233 | 1.025 | 1.224 ± 0.039 | 1.154 |
| | DARNN | 1.709 ± 0.85 | 0.688 | 0.992 ± 0.025 | 0.979 |
| | SteamGNN | 1.084 ± 0.062 | 1.29 | 1.819 ± 0.683 | 0.888 |
| | SDPANet | **0.602 ± 0.356** | **0.58** | **0.862 ± 0.073** | **0.807** |
| CORR | LSTM | 0.827 ± 0.033 | 0.710 | 0.316 ± 0.001 | 0.287 |
| | WaveNet | 0.372 ± 0.039 | 0.287 | 0.233 ± 0.008 | 0.166 |
| | MLCNN | 0.926 ± 0.005 | 0.91 | 0.613 ± 0.019 | 0.589 |
| | LSTNet | 0.123 ± 0.016 | 0.110 | 0.255 ± 0.003 | 0.228 |
| | MTNet | 0.091 ± 0.006 | 0.125 | 0.115 ± 0.010 | 0.150 |
| | DARNN | 0.783 ± 0.057 | 0.741 | 0.516 ± 0.026 | 0.452 |
| | StemGNN | 0.756 ± 0.201 | 0.54 | 0.751 ± 0.055 | 0.639 |
| | SDPANet | **0.923 ± 0.006** | **0.931** | **0.796 ± 0.003** | **0.805** |

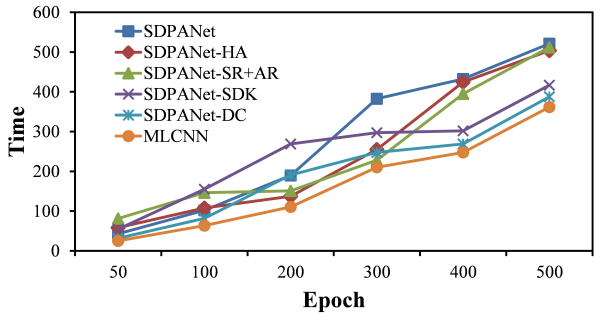



Figure 2: Running efficiency

**SDPANet-DC**: We remove the dynamic change loss function and replace it with the L1 loss function.
**SDPANet-SR**: We remove the Simultaneous Regression component.
**SDPANet-SR+AR**: We replace the Simultaneous Regression (SR) with Auto Regression (AR) component.

Experimental results can be seen in Figure 2. MLCNN is taken as a benchmark. When Epoch reaches 500, SDPANet is about 200 seconds slower than MLCNN, and SDK and DC loss are the main time-consuming components. Besides, the running efficiency of SDPANet and SDPANet-SR+AR is very close, which indicates that using dynamic programming to optimize SR could obtain a significant improvement in running efficiency.