

OneShotSTL: One-Shot Seasonal-Trend Decomposition For Online Time Series Anomaly Detection And Forecasting

Xiao He
Alibaba Group
xiao.hx@alibaba-inc.com

Ye Li
Alibaba Group
liye.li@alibaba-inc.com

Jian Tan
Alibaba Group
j.tan@alibaba-inc.com

Bin Wu
Alibaba Group
binwu.wb@alibaba-inc.com

Feifei Li
Alibaba Group
lifeifei@alibaba-inc.com

ABSTRACT

Seasonal-trend decomposition is one of the most fundamental concepts in time series analysis that supports various downstream tasks, including time series anomaly detection and forecasting. However, existing decomposition methods rely on batch processing with a time complexity of $O(W)$, where W is the number of data points within a time window. Therefore, they cannot always efficiently support real-time analysis that demands low processing delay. To address this challenge, we propose OneShotSTL, an efficient and accurate algorithm that can decompose time series online with an update time complexity of $O(1)$. OneShotSTL is more than 1,000 times faster than the batch methods, with accuracy comparable to the best counterparts. Extensive experiments on real-world benchmark datasets for downstream time series anomaly detection and forecasting tasks demonstrate that OneShotSTL is from 10 to over 1,000 times faster than the state-of-the-art methods, while still providing comparable or even better accuracy.

PVLDB Reference Format:

Xiao He, Ye Li, Jian Tan, Bin Wu, Feifei Li. OneShotSTL: One-Shot Seasonal-Trend Decomposition For Online Time Series Anomaly Detection And Forecasting. PVLDB, 16(6): 1399 - 1412, 2023.
doi:10.14778/3583140.3583155

1 INTRODUCTION

With the rapid advancement in data collection and storage techniques, time series are witnessing an enormous increase in applications, especially in areas such as Internet-of-Things (IoT [21]) and IT Operations (AIOps [4]). For instance, cloud database systems collect large sets of metrics, including CPU usage and request rate, to facilitate the monitoring of complex systems. Driven by this trend, cloud companies (e.g., AWS Cloudwatch [2] and Alibaba CloudMonitor [1]) have provided online services to improve the observability of servers and applications for their users.

From an algorithmic perspective, real-time analysis tasks on Time Series fall into either Anomaly Detection (TSAD) or Forecasting (TSF). For a time series $\{y_1, y_2, \dots, y_t, \dots\}$, at each time point t ,

an online TSAD method outputs an anomaly score for the current data y_t , while an online TSF method predicts the future value of data \hat{y}_{t+i} at time $t+i$ for $i > 0$. The former is used to identify performance issues of the system based on historical observations (e.g., through alerting or diagnosis), while the latter is to predict the future to prevent such issues by taking certain actions in advance (e.g., by automatically scaling up the computation resources). The significant volume of data generated from large-scale systems, together with the requirement of low service delays, impose great challenges to TSAD and TSF, due to the online nature of the processing.

There have been a large number of TSAD [12, 14, 27, 38, 44] and TSF [11, 29, 34, 35, 45, 47] methods proposed in the literature. In this paper, we focus on an important class of algorithms based on Seasonal-Trend Decomposition (STD). STD is one of the most fundamental operations in time series analysis, which has been extensively studied and used in economics and finance [16, 19]. Many studies have shown that STD methods are effective for TSAD [19, 40, 42] and TSF [18, 29], since very often practical time series data exhibit trends and seasonal patterns [40]. However, most of the existing STD methods, e.g., STL [16] and RobustSTL [40], are based on batch processing, and thus essentially offline by nature, which does not support incremental updates. The only exception is a recent method OnlineSTL [28], which decomposes time series in an online fashion by effectively utilizing the fast computation from explicit mathematical closed-form expressions. Remarkably, OnlineSTL can process more than 100 times faster than traditional STD methods. However, only simple trend and seasonal filtering methods (e.g., tri-cube and exponential smoothing) are supported. Therefore, the decomposition quality of OnlineSTL is worse than its full-fledged batch counterparts, e.g., RobustSTL [40], which introduces l_1 -norm regularizer. Moreover, although OnlineSTL is fast, its single update complexity is still $O(T)$ with T being the length of the seasonal period. Thus, its performance degrades with the increase of T .

To address the aforementioned challenges, in this paper, we propose a new online STD algorithm, termed OneShotSTL, that significantly accelerates the decomposition with an update complexity of $O(1)$. The needed information from the past observations to update the effect by observing a newly arrived data point is characterized by a fixed number of state parameters of OneShotSTL, whose values are carried forward as the online processing keeps going. Thus, OneShotSTL processes each data point only once. To achieve high accuracy, we adopt a l_1 -norm regularizer and adaptively adjust the seasonal component in our online computation. Notably, OneShotSTL can yield accurate decomposition results

Xiao He is the corresponding author.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 16, No. 6 ISSN 2150-8097.
doi:10.14778/3583140.3583155

Table 1: Comparison of different STD algorithms

| Algorithm | Batch or Online | Trend Change | Seasonality Shifts | Online Complexity |
|---------------------|-----------------|--------------|--------------------|--------------------------|
| STL [16] | Batch | No | No | - |
| TATBS [18] | Batch | No | No | - |
| STR [19] | Batch | No | No | - |
| RobustSTL [40] | Batch | Yes | Yes | - |
| FastRobustSTL [42] | Batch | Yes | Yes | - |
| OnlineRobustSTL [7] | Online | Yes | Yes | $O(T)$ |
| OnlineSTL [28] | Online | No | No | $O(T)$ |
| OneShotSTL | Online | Yes | Yes | $O(1)$ |

comparable to RobustSTL [40] (the best batch method that we are aware of in the literature) and in the meanwhile can greatly reduce the processing time. Extensive experiments on both TSAD and TSF tasks demonstrate the superior performance of OneShotSTL.

Contributions. This work makes the following contributions:

- To the best of our knowledge, the proposed OneShotSTL is the first online STD method with an update complexity of $O(1)$, which significantly improves the existing ones ($O(T)$).
- OneShotSTL is ultra-fast in practice. It takes around $20\mu\text{s}$ to process each data point on a typical commodity server using a single CPU core, which is more than 1,000 times faster than the batch STD methods and 20 times faster than OnlineSTL on datasets with a seasonal period of 12,800.
- OneShotSTL is also accurate. Extensive experiments on benchmark datasets for TSAD and TSF show that OneShotSTL is from 10 to more than 1,000 times faster than the state-of-the-art methods with comparable or even better accuracy.

2 PROBLEM SETTING AND RELATED WORKS

Since our online STD framework is derived from the batch STD, we formally introduce both the batch and the online STD problems. We also review the related existing works in this area. Throughout the paper, we use bold lowercase/capital letters to represent vectors/matrices, respectively.

2.1 Batch STD

For batch processing, consider the following model with trend and seasonality for a time series $\mathbf{y} \in \mathbb{R}^N$ of a finite batch size N :

$$y_i = \tau_i + s_i + r_i, i = 1, 2, \dots, N, \quad (1)$$

where y_i , τ_i , s_i and r_i denote the observation, the decomposed trend, seasonal and residual components at time point i , respectively. We only consider the decomposition of data with a single seasonal cycle of length $T < N$, which can be readily estimated by period detection methods [41]. This can be easily extended to the case with multiple seasons as shown in previous studies [40, 42]. In this paper, we treat data with multiple seasons as a single seasonal sequence with T being the longest seasonal period.

2.2 Online STD

For online processing, data points arrive in a streaming fashion, denoted by $\{y_1, y_2, \dots, y_t, \dots\}$ where the sequence can be even unbounded. At time t , based on all the past observations $\{y_1, \dots, y_{t-1}\}$ and the latest one y_t , online methods only decompose y_t into trend τ_t , seasonal s_t and residual r_t with $y_t = \tau_t + s_t + r_t$. Similarly to

OnlineSTL [28], we divide the online processing into two phases, a single offline initialization phase and a lasting online phase. Formally, at time t we observe $\{y_1, y_2, \dots, y_{t_0}, y_{t_0+1}, \dots, y_t\}$. The first t_0 points $\{y_1, \dots, y_{t_0}\}$ are used for initialization. The online processing starts from time $t_0 + 1$ and keeps computing τ_t , s_t and r_t for $t > t_0$ by using only data points observed before t . The seasonal cycle T , as a fixed parameter, can be estimated from $\{y_1, \dots, y_{t_0}\}$ using a period detection method, e.g., [39, 41]. During the online computation phase, as data points keep arriving, we observe new seasonal periods, and each has its own cycle length T_c . These cycle lengths are not necessarily equal. We assume that each T_c of the underlying cycle lengths from the streaming data sequence, although unknown, only slightly deviates from T . Specifically, T_c can be an arbitrary integer value within a small interval decided by a hyper-parameter H , i.e., $T_c \in [T - H, T + H]$. This assumption prevents cycle lengths from varying drastically, e.g., T_c gradually changing from 1000 to 2000 across multiple periods. A more general treatment of the varying cycle lengths deserves a dedicated study given the vast existing literature on this topic [33, 36, 39, 41].

2.3 Existing STD Methods

Batch STD has been extensively studied in economics and finance for decades, with a variety of algorithms proposed in the literature [16, 18, 19, 28, 40]. Among them, the most popular one is STL [16], which adopts an alternating algorithm to estimate the trend and seasonal components using LOESS (Local regrESSion) smoothing. Later, TATBS [18] and STR [19] are proposed with confidence intervals of the decomposition results. However, they need extensive computations, especially for data with long seasonality. In addition, these methods may encounter problems in handling complex patterns, e.g., with abrupt changes of trend (see Figure 4) and seasonality shift (see Figure 3), which are common in metrics for monitoring in AIOps. As a way to improve these issues, RobustSTL [40] and its accelerated version FastRobustSTL [42] are proposed using l_1 -norm trend filtering and non-local seasonal filtering.

All these batch methods can be used for the online setting on a sliding window of length $W > T$ that contains the most recent data points. A common approach is to set W equal to the length of a few seasonal cycles, e.g., $W = 4T$. Thus, the time complexity of these batch methods is at least $O(T)$. However, as pointed out in the recent paper OnlineSTL [28], most of these methods cannot always be directly and efficiently utilized in real-time applications due to long processing delays. OnlineSTL is the first online decomposition algorithm that alternatively applies tri-cube (trend) and exponential smoothing (seasonal) filtering, which utilizes fast computation from explicit mathematical closed-form expressions. OnlineSTL focuses on speed and can process 10,000 time series data points per second using a single CPU core on a typical commodity server. However, it sacrifices quality/accuracy by using simple trend and seasonal filtering and cannot deal with complex patterns. More importantly, the online update complexity of OnlineSTL and other existing online algorithms, e.g., online RobustSTL [7, 42] (abbreviated as OnlineRobustSTL), is still $O(T)$. Therefore, the performance degrades as the length of the seasonality cycle increases.

We summarize the comparisons of batch and online STD methods in Table 1, including our proposed OneShotSTL. Note that OneShotSTL is the only work with $O(1)$ update complexity.

For the online setting, the whole data processing procedure consists of an initialization phase and an online updating phase. Specifically, for a data sequence $\{y_1, y_2, \dots, y_{t_0}, y_{t_0+1}, \dots, y_t\}$, the first t_0 data points $\{y_1, y_2, \dots, y_{t_0}\}$ are used for initialization, $t_0 \geq W$, e.g., $W = 4T$ with T being the longest seasonality period as in OnlineSTL. We can apply a batch model, e.g., STL, processing on $\{y_1, y_2, \dots, y_{t_0}\}$ and obtain $\{\tau_1, \tau_2, \dots, \tau_{t_0}\}$ and $\{s_1, s_2, \dots, s_{t_0}\}$. Therefore, when decomposing the first data point $t_0 + 1$ in the online phase, we already have the seasonal component of the previous data points. Further, after each online decomposition, we can always keep the most recent period of the seasonal component in a sliding window. We name the seasonal buffer as $\mathbf{v} \in \mathbb{R}^T$ and initialize it with $\mathbf{v} = [s_{t_0-T}, \dots, s_{t_0}]^T$. Then we modify the second term of Problem (2) on seasonal component and rewrite the problem as:

$$\min_{\tau, s} \left(\sum_{j=t_0+1}^{t_0+M} (\tau_j + s_j - y_j)^2 + \sum_{j=t_0+1}^{t_0+M} (s_j - v_{j\%T})^2 \right. \\ \left. + \lambda_1 \sum_{j=t_0+1}^{t_0+M} |\tau_j - \tau_{j-1}| + \lambda_2 \sum_{j=t_0+1}^{t_0+M} |\tau_j - 2\tau_{j-1} + \tau_{j-2}| \right) \quad (7)$$

where for each newly arrived data point y_t at timestamp $t = t_0 + M$, we decompose all the y_j into τ_j and s_j for $j \in [t_0 + 1, t_0 + M]$ but only output the last decomposition result τ_{t_0+M} and s_{t_0+M} .

Similarly, we can introduce auxiliary variables $\mathbf{p} \in \mathbb{R}^M$ and $\mathbf{q} \in \mathbb{R}^M$ and use IRLS [22] for solving Problem (7). The update rules of p_t and q_t are the same as Eq. (4) and (5). But when \mathbf{p} and \mathbf{q} are fixed, we solve a different linear system to derive $\boldsymbol{\tau}$ and \mathbf{s} . Specifically, we concatenate $\boldsymbol{\tau}$ and \mathbf{s} to form $\mathbf{x} = \{\tau_{t_0+1}, s_{t_0+1}, \dots, \tau_{t_0+M}, s_{t_0+M}\}$. The optimal \mathbf{x} is the solution of a linear system $\mathbf{Ax} = \mathbf{b}$, satisfying

$$\mathbf{A} = \mathbf{C}_1^T \mathbf{C}_1 + \mathbf{C}_2^T \mathbf{C}_2 + \lambda_1 (\mathbf{p} \cdot \mathbf{C}_3)^T \mathbf{C}_3 + \lambda_2 (\mathbf{q} \cdot \mathbf{C}_4)^T \mathbf{C}_4 \\ \mathbf{b} = \mathbf{C}_1^T \mathbf{y} + \mathbf{C}_2^T \mathbf{u}, \text{ where } \mathbf{u} = [v^T, \dots, v^T]^T \in \mathbb{R}^M$$

$$\mathbf{C}_1 = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & 1 \\ & & & & 1 \end{bmatrix} \in \mathbb{R}^{M \times 2M} \\ \mathbf{C}_2 = \begin{bmatrix} 0 & 1 & & & \\ & & \ddots & & \\ & & & 0 & 1 \end{bmatrix} \in \mathbb{R}^{M \times 2M} \quad (8) \\ \mathbf{C}_3 = \begin{bmatrix} 1 & 0 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{(M-1) \times 2M} \\ \mathbf{C}_4 = \begin{bmatrix} 1 & 0 & -2 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & 0 & -2 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{(M-2) \times 2M}$$

However, the above method still solves a batch problem. Moreover, it is difficult to derive an efficient online algorithm based on it. Note that, in the online setting, we target the decomposition of the newly arrived data point. Thus, we modify JointSTL for the online setting that can well approximate the last decomposition, as shown in Algorithm 2.

During the initialization phase of Algorithm 2, we use STL or JointSTL on $\{y_1, \dots, y_{t_0}\}$ and set \mathbf{v} as introduced before. Then we

Algorithm 2: Modified Batch JointSTL for online setting

```
//Initialization phase
Input:  $y_1, \dots, y_{t_0}, T, \lambda$ , maximum iterations  $I$ 
1 Obtain  $\boldsymbol{\tau}, \mathbf{s}, \mathbf{r}$  by STL or JointSTL; Set  $\mathbf{v} = [s_{t_0-T}, \dots, s_{t_0}]$ ;
2  $\mathbf{p}^0 = \mathbf{q}^0 = [1]$ ;  $\mathbf{p}^i = \mathbf{q}^i = []$ , for  $i \in [1, I]$ ;

//Online phase
Input:  $\mathbf{y} = \{y_1, y_2, \dots, y_t\}, T, \lambda, I$ 
Output:  $\tau_t, s_t, r_t$ 
3 for ( $i = 0$ ;  $i < I$ ;  $i++$ )
4   Construct  $\mathbf{A}$  and  $\mathbf{b}$  based on Eq. (8) using  $\mathbf{p}^i, \mathbf{q}^i$ ;
5   Obtain  $\mathbf{x}$  by solving the linear system  $\mathbf{Ax} = \mathbf{b}$ ;
6    $\tau_t = x_{-2}$ ;  $s_t = x_{-1}$ ;  $r_t = y_t - \tau_t - s_t$ ;
7   Update  $p_t$  and  $q_t$  based on Eq. (4) and (5);
8    $\mathbf{p}^{i+1} = [p^{i+1}, p_t]$ ;  $\mathbf{q}^{i+1} = [q^{i+1}, q_t]$ ;
9    $\mathbf{p}^0 = [p^0, 1]$ ;  $\mathbf{q}^0 = [q^0, 1]$ ;
10  $v_{t\%T} = s_t$ ;
```

set $\mathbf{p} = \mathbf{p}^i$ and $\mathbf{q} = \mathbf{q}^i$ for the i -th iteration, which will be used for constructing \mathbf{A} and \mathbf{b} in Eq. (8). Because, during the online phase, we have different vectors of \mathbf{p}^i and \mathbf{q}^i updated using Eq. (4) and (5) for each iteration i . Unlike JointSTL which updates the whole vector of \mathbf{p} and \mathbf{q} , while decomposing y_t , we only append \mathbf{p}^i and \mathbf{q}^i with the latest p_t and q_t calculated from the decomposition of τ_t and s_t for iteration i . Finally, we output the decomposition results from the last iteration. In practice (see experiments in Section 5), we find that this approximation provides sufficiently good decomposition.

However, Algorithm 2 is still unrealistic since for each time point $t = t_0 + M$, one needs to solve a linear system $\mathbf{Ax} = \mathbf{b}$ in each iteration of Algorithm 2 (lines 4-8), where $\mathbf{A} \in \mathbb{R}^{2M \times 2M}$ and its size is increasing while processing more data points online. Recall that in the online setting, at time point t , one only needs to partially solve $\mathbf{Ax} = \mathbf{b}$ and outputs the latest decomposition τ_t and s_t . Moreover, from Eq. (8) we can see that \mathbf{A} is a symmetric banded matrix with constant bandwidth of 9 in regardless of the growing of M and the values of \mathbf{p} and \mathbf{q} , since $\mathbf{C}_1^T \mathbf{C}_1$, $\mathbf{C}_2^T \mathbf{C}_2$, $\mathbf{C}_3^T \mathbf{C}_3$ and $\mathbf{C}_4^T \mathbf{C}_4$ are all banded matrices with constant bandwidth of 3, 3, 5 and 9, respectively, for any $M \geq 5$. Therefore, the linear system $\mathbf{Ax} = \mathbf{b}$ can be efficiently solved using Cholesky/Doolittle factorization and Gaussian elimination. The crux of the matter is that the Cholesky/Doolittle factorization and the forward substitution of Gaussian elimination can be made online. Specifically, at each time point and for each iteration of Algorithm 2, we only need to conduct the first few steps of the backward substitution and output the last two elements of \mathbf{x} , i.e., τ_t and s_t . Following this idea, we propose OneShotSTL detailed in the next section.

3.3 OneShotSTL Aglorithm

This section presents the online symmetric Doolittle factorization algorithm with $O(1)$ update complexity to accelerate Algorithm 2, which leads to the full OneShotSTL algorithm.

3.3.1 Online Symmetric Doolittle Factorization. For each time point t , Algorithm 2 needs to sequentially solve I different linear systems of $\mathbf{Ax} = \mathbf{b}$, where I is a configured parameter on the number

of iterations. Note that this number I is prefixed and independent of the input data size, e.g., $I = 8$. The reason is that the optimization of Problem (7) using Algorithm 2 based on IRLS usually converges very fast [22]. Therefore, we only need a few iterations to obtain satisfactory results (see experiments in Section 5). This algorithm has two steps. The first step is to initialize the first iteration, and the second step is to repeat the rest $I - 1$ iterations based on the result from the previous iteration.

Algorithm 3: Symmetric Doolittle Factorization

Input: $A \in \mathbb{R}^{M \times M}$
Output: $L \in \mathbb{R}^{M \times M}$, $D \in \mathbb{R}^{M \times M}$

- 1 **for** ($k = 1; k < M; k ++$)
- 2 $L_{kk} = 1;$
- 3 $D_{kk} = A_{kk} - \sum_{i=1}^{k-1} D_{ii} \cdot L_{ki}^2;$
- 4 **for** ($j = k + 1; j < M; j ++$)
- 5 $L_{kj} = 0;$
- 6 $L_{jk} = (A_{jk} - \sum_{i=1}^{k-1} L_{ji} \cdot D_{ii} \cdot L_{ki}) / D_{kk};$

At time point $t = t_0 + M$, we can construct $A^t \in \mathbb{R}^{2M \times 2M}$ using Eq. (8) with $p = q = 1$ for the first iteration of Algorithm 2. Similarly, for the next time point $t + 1$, we can construct A^{t+1} . Interestingly, some sub-matrices of A^t and A^{t+1} are identical. Figure 2 shows two examples of A^t and A^{t+1} with $M = 5$ and $M = 6$, respectively. As can be seen from the figure, we can split each matrix into 4 parts according to dashed lines. The top-left sub-matrices of A^t and A^{t+1} are identical, and the top-right/bottom-left sub-matrices of A^t and A^{t+1} are also the same after appending rows and columns of zeros. The only differences lie in the bottom-right sub-matrix in the red box, where $A^o \in \mathbb{R}^{4 \times 4}$ of A^t is replaced by $A^* \in \mathbb{R}^{6 \times 6}$ of A^{t+1} . One can easily verify that the same situation occurs for any $M \geq 5$.

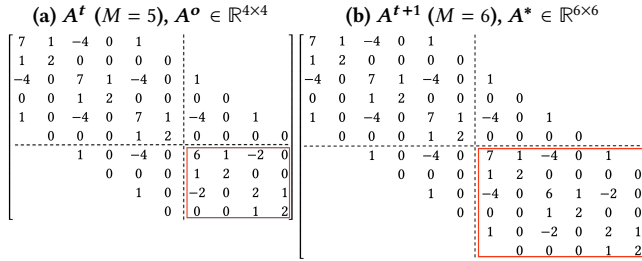


Figure 2: Example of A with $p=q=1$ and $\lambda_1 = \lambda_2 = 1$.

Next, we show that the common parts of A^t and A^{t+1} from adjacent timestamps can be utilized for online processing. Firstly, we can adopt Symmetric Doolittle Factorization (SDF) shown in Algorithm 3 to factorize $A = LDL^T$, where D is a diagonal matrix and L is a lower triangular matrix. Then, while factorizing $A^t = L^t D^t L^{tT}$ using SDF, we can buffer the latest L^t and D^t before processing A^o (Figure 2(a)). Further, we can continue to process A^* (Figure 2(b)) for time point $t + 1$ while factorizing A^{t+1} since the top-left parts of L^{t+1} and D^{t+1} are the same with L^t and D^t . Lastly, thanks to the fact that A is a banded matrix of length 9, we only need to buffer sub-matrices of L and D with a size of $\mathbb{R}^{10 \times 10}$ since the others are either 0 or not used in the computation.

The complete OnlineDoolittle algorithm for partially solving $Ax = b$ online is presented in Algorithm 4, including the forward

Algorithm 4: OnlineDoolittle

Input: $A^* \in \mathbb{R}^{6 \times 6}$, $b^* \in \mathbb{R}^6$, ($L^o \in \mathbb{R}^{8 \times 4}$, $D^o \in \mathbb{R}^{4 \times 4}$ and $b^o \in \mathbb{R}^4$ are needed for the first call)
Output: τ_t, s_t, r_t

- 1 L^o, D^o and b^o are initialized for the first call and updated online;
- 2 $\bar{L} = \text{zeros}(10, 10); \bar{L}_{1:8, 1:4} = L^o;$
- 3 $\bar{D} = \text{zeros}(10, 10); \bar{D}_{1:4, 1:4} = D^o;$
- 4 $\bar{b} = [b^o; b^*];$
- 5 **for** ($k = 5; k \leq 10; k ++$)
- 6 $\bar{L}_{kk} = 1;$
- 7 $\bar{D}_{kk} = A_{k-4, k-4}^* - \sum_{i=k-4}^k \bar{D}_{ii} \cdot \bar{L}_{ki}^2;$
- 8 $\bar{b}_k = -\sum_{i=k-4}^k \bar{L}_{ki} \cdot \bar{b}_i; //$ Forward substitution part1
- 9 **for** ($j = k + 1; j \leq 10; j ++$)
- 10 $\bar{L}_{jk} = (A_{j-4, k-4}^* - \sum_{i=k-4}^k \bar{L}_{ji} \cdot \bar{D}_{ii} \cdot \bar{L}_{ki}) / \bar{D}_{kk};$
- 11 **for** ($j = 5; j \leq 10; j ++$)
- 12 **for** ($i = j + 1; i < j + 5; i ++$)
- 13 $\bar{b}_i = \bar{L}_{ij} \cdot \bar{b}_j; //$ Forward substitution part2
- 14 **for** ($j = 10; j \geq 5; j --$)
- 15 **for** ($i = j + 1; i \leq 10; i ++$)
- 16 $\bar{b}_i = \bar{D}_{ii} \cdot \bar{L}_{ji} \cdot \bar{b}_j; //$ Backward substitution
- 17 $L^o = \bar{L}_{3:10, 3:6}; D^o = \bar{D}_{3:6, 3:6}; b^o = \bar{b}_{3:6};$
- 18 $\tau_t = \bar{b}_9; s_t = \bar{b}_{10}; r_t = y_t - \tau_t - s_t;$

and backward substitutions of Gaussian elimination that can be modified with the same principle. Note that OnlineDoolittle is almost the same as the original SDF (Algorithm 3), where we keep updating the most recent L^o, D^o and b^o with the factorization and the forward/backward substitutions conducted online. Finally, it outputs the decomposition result of τ_t and s_t for each timestamp t . The number of calculations needed in Algorithm 4 is always a constant, and thus its runtime complexity is $O(1)$.

Next, we show that OnlineDoolittle algorithm can also be used for partially solving $Ax = b$ for the i -th iteration of Algorithm 2. To construct A and b using Eq. (8) for iteration i , we need p^{i-1} and q^{i-1} updated from the previous iteration. Note that we only append p_t and q_t at each time point t calculated in iteration $i - 1$ to p^{i-1} and q^{i-1} . Therefore, the values of $p_{1:t-1}$ and $q_{1:t-1}$ for all the previous timestamps are fixed while processing time point t at iteration i . Thus, the A^t and A^{t+1} from adjacent timestamps for iteration i also share the same property, as shown in Figure 2. Then, we can build a linear system with growing A and b for each iteration, whose construction depends on the results of the previous iteration. Eventually, they can be partially solved online using the OnlineDoolittle algorithm to obtain the final decomposition results.

3.3.2 OneShotSTL algorithm details. Combining all the ingredients introduced before, we have the complete OneShotSTL algorithm, shown in Algorithm 5. It solves the same problem as Algorithm 2 in an online fashion. During the initialization phase, we conduct STL or batch JointSTL on the existing data to obtain one period of seasonal component v . During the online phase, at time point t OneShotSTL processes a new observation y_t and outputs the decomposition results τ_t and s_t . Specifically, for each iteration i , it sequentially updates p_t and q_t using τ_t and s_t from previous iterations and constructs A^* and b^* as in Figure 2 using Eq. (8). Then, each iteration incrementally applies the symmetric Doolittle

Algorithm 5: OneShotSTL

```
//Initialization phase
Input:  $y_1, \dots, y_{t_0}, T, \lambda$ , maximum iterations  $I$ 
1  $M = 0$ ; Obtain  $\tau, s, r$  by STL or JointSTL; Set  $\mathbf{v} = [s_{t_0-T}, \dots, s_{t_0}]$ ;
2  $\mathbf{p}^0 = \mathbf{q}^0 = [1]$ ;  $\mathbf{p}^i = \mathbf{q}^i = []$ , for  $i \in [1, I]$ ;

//Online phase
Input:  $y_t, t, n, \lambda, I$ 
Output:  $\tau_t, s_t, r_t$ 
3 for ( $i = 0; i < I; i++$ )
4   Construct  $\mathbf{A}^*$  and  $\mathbf{b}^*$  as in Figure 2 using Eq. (8);
5    $\tau_t, s_t, r_t = \text{OnlineDoolittle}_i(\mathbf{A}^*, \mathbf{b}^*)$ ;
6   Update  $\mathbf{p}_t$  and  $\mathbf{q}_t$  using Eq. (4) and (5);
7    $\mathbf{p}^{i+1} = [\mathbf{p}^{i+1}, \mathbf{p}_t]$ ;  $\mathbf{q}^{i+1} = [\mathbf{q}^{i+1}, \mathbf{q}_t]$ ;
8  $\mathbf{p}^0 = [\mathbf{p}^0, 1]$ ;  $\mathbf{q}^0 = [\mathbf{q}^0, 1]$ ;
9  $v_{t\%T} = s_t$ ;
```

factorization using \mathbf{A}^* and \mathbf{b}^* with OnlineDoolittle in Algorithm 4, denoted as OnlineDoolittle_i for iteration i . Finally, we output the decomposition result τ_t and s_t from the last iteration and update the corresponding value in \mathbf{v} with the decomposed seasonal s_t .

The time complexity of OnlineDoolittle (Algorithm 4) is $O(1)$. Thus, the worst-case online update complexity of OneShotSTL is $O(I)$, where I is the maximum number of iterations. Note that I is a fixed constant, independent of the problem size, implying that the online update complexity of OneShotSTL is $O(1)$. In practice, it is the main factor that affects the speed of OneShotSTL. But, a small value, e.g., $I = 8$, is sufficient to produce satisfactory results; see experiments in Section 5).

3.4 Handle Seasonality Shift

In this section, we handle the case of seasonality shift when the underlying T_c can be any integer in $[T - H, T + H]$, $H \geq 1$.

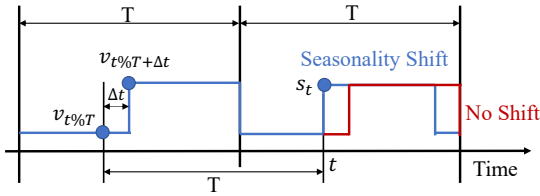


Figure 3: Seasonality Shift of Δt time points.

Figure 3 depicts an example of seasonality shift of Δt time points, which happens commonly in the metric monitoring applications of AIOps. The blue line represents the ground truth of 2 periods of seasonal component of a time series data. From time point t there is a shift of Δt time points. The red dashed line represents the imaginary data if there is no shift. For simplicity, until time point t we assume \mathbf{v} equals the first seasonal period in Figure 3. Clearly, OneShotSTL in Algorithm 5 will not decompose it well after time point t , since $v_{t\%T}$ is significantly different from the ground truth s_t . If we know Δt , we can replace $(s_t - v_{t\%T})$ in Problem (7) with $(s_t - v_{t\%T + \Delta t})$ to tackle the seasonality shift issue. However, we do not know the correct Δt for each time point t in practice.

To solve this problem, we first use the hyper-parameter H to form neighborhoods \mathcal{E} with all possible Δt within H , e.g., $\mathcal{E} = \{0\}$ for

Algorithm 6: NSigma

```
Input:  $r_t, n$ 
Output: isAnomaly, score
1 isAnomaly = False;
2 if not initialized then
3   sum = sumSquared = count = 0;
4 if count > 0 then
5   mean = sum / count; std = sqrt(sumSquared / count - mean2);
6   score = abs( $r_t$  - mean) / std;
7   if score > n then
8     isAnomaly = True;
9 count += 1; sum +=  $r_t$ ; sumSquared +=  $r_t^2$ ;
```

$H = 0$ and $\mathcal{E} = \{-1, 0, 1\}$ for $H = 1$. Further, we observe that when a seasonality shift happens, there will be a spike anomaly point in the decomposed residual r_t . Therefore, we monitor the residual r_t and use the streaming NSigma algorithm shown in Algorithm 6 to detect whether there is an anomaly in r_t for each time point t . In Algorithm 6, n is an input parameter indicating the level of abnormal, e.g., $n = 5$. Once we found an anomaly, we repeat lines 3 – 8 of Algorithm 5 for each $\Delta t \in \mathcal{E}$ that is used to construct \mathbf{A}^* and \mathbf{b}^* (line 4). Then we pick Δt with the smallest absolute value of r_t and output the corresponding τ_t and s_t . This method gives the most accurate decomposition results in our simulations.

With the above method to handle seasonality shift, the computation complexity of OneShotSTL is $O(I \cdot H)$, where H is the prefixed parameter that indicates the maximum variations allowed for the cycle length shift, e.g., $H = 20$. Since both I and H are fixed constant, independent of each period length T_c , implying that the online update complexity of OneShotSTL is still $O(1)$. Moreover, it also does not affect the speed of OneShotSTL much in practice since anomalies (e.g., with $n > 5$) in the residual signal are rare (e.g., usually less than 1%).

4 EXTENSION FOR DOWNSTREAM TASKS

OneShotSTL and other online STD methods can be easily extended for downstream tasks of online TSAD and TSF as follows:

- (1) **Univariate TSAD:** For an unbounded time series, at each time point t , an online TSAD method observes all the historical data $\{y_1, y_2, \dots, y_t\}$ and outputs an anomaly score for y_t . OnlineSTL and OneShotSTL can be extended to online TSAD methods by combining with streaming NSigma shown in Algorithm 6. Besides the decomposition results, they can output anomaly scores using NSigma on the decomposed residual signals.
- (2) **Univariate TSF:** Similarly, an online TSF method observes $\{y_1, \dots, y_t\}$ at time point t and makes an i -step prediction for the future value of data \hat{y}_{t+i} at time point $t + i$. Seasonal signals are periodical, thus we can use them for long-sequence forecasting. We adopt a simple method where we buffer the most recent decomposed trend (τ_{t-1}) and one period of seasonal signals (\mathbf{v}) during online decomposition. Then, to make an i -step prediction at timestamp t , we simply use $\hat{y}_{t+i} = \tau_{t-1} + v_{t\%T+i}$ where T is the period length. The same method applies to OnlineSTL.

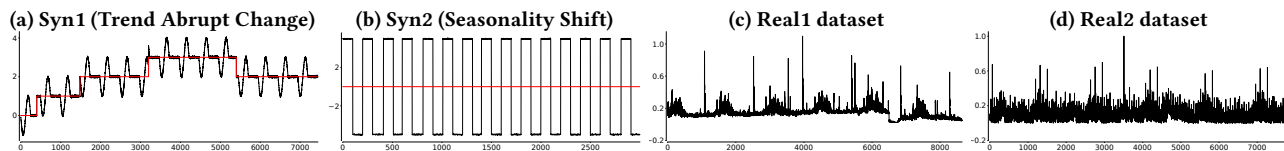


Figure 4: Synthetic and Real-world datasets for the evaluation of STD quality.

5 EXPERIMENTS

We conduct comprehensive experiments to evaluate OneShotSTL in comparison with other methods. After introducing the experimental setting, we evaluate the decomposition accuracy and the scalability, followed by comparison on downstream TSAD and TSF tasks.

5.1 Experimental Settings

5.1.1 Datasets. Three sets of datasets are used for evaluation.

1. Synthetic and Real-world STD datasets. We generate two synthetic datasets (**Syn1** and **Syn2**) and use two real-world datasets (**Real1** and **Real2**) as shown in Figure 4. These two real-world datasets are on the request rates of some internal APIs on Alibaba Cloud Database systems. The red lines in Figure 4 (a) and (b) show the ground truth of the trend signals of the synthetic datasets. **Syn1** and **Syn2** are designed for cases with abrupt trend changes and seasonality shifts, respectively. In **Syn2** there are 4 periods shifted by 10 data points which are not visually distinguishable. Due to the limited space, we present the details of generating the synthetic datasets in the supplementary material [10]. For the two real-world datasets, **Real1** exhibits an abrupt change of the trend and **Real2** contains observable noises with weak seasonality.

2. Univariate TSAD datasets. We evaluate the univariate TSAD task using the TSB-UAD benchmark [31]. There are over 2000 real-world time series in the TSB-UAD benchmark with anomalies labeled by domain experts. Among them, the most famous one is the KDD CUP 2021 (KDD21) TSAD competition dataset [17] consists of 250 univariate time series. For details of these datasets please refer to [31]. For KDD21, IOPS, NASA-MSL and NASA-SMAP datasets, we use the train part of the data for initialization. For the rest of the datasets, we use the first 3000 data points for initialization.

3. Univariate TSF datasets. We evaluate the TSF task on six real-world datasets (ETT, Electricity, Exchange, Traffic, Weather, and Illness) from Informer [45] which have been extensively used to benchmark long sequence TSF by recent works, e.g., FEDformer [47] and FiLM [46]. We follow the same experimental settings as these methods with forecasting lengths of {24, 36, 48, 60} for Illness and {96, 192, 336, 720} for the rest. The train, validation, and test splitting details can be found in [45] and are publicly available [5].

5.1.2 Comparison Methods. We compare OneShotSTL with the state-of-the-art STD, TSAD, and TSF baseline methods.

1. STD Baselines. We compare OneShotSTL with two batch STD methods (STL [16] and RobustSTL [40, 42]) and two online methods (OnlineSTL [28] and OnlineRobustSTL [7, 42]). For the online setting, STL and RobustSTL can also be used on the sliding window of the most recent data points, which are named WindowSTL and Window-RobustSTL. We use the public Java implementation of STL [6] and implement OnlineSTL and OneShotSTL in Java. For RobustSTL we use the Python implementation that is publicly available in package SREWorks [7].

2. TSAD Baselines. We compare with matrix profile-based methods NormA [14], STOMPI [44], SAND [15], and DAMP [27], which show state-of-the-art performance for univariate time series anomaly detection on the TSB-UAD benchmark. NormA is a batch TSAD method, while STOMPI, SAND, and DAMP are online TSAD methods. A batch method reads all the data of a time series and output an anomaly score for each data point, while an online method uses the training data for initialization and update the model during online detection. Further, we compare with deep learning-based TSAD methods including, LSTM [32], USAD [12] and TranAD [38]. USAD and TranAD are recent methods designed for fast training. We use the Python implementation of LSTM, NormA, and SAND from the TSB-UAD benchmark [31]. For USAD and TranAD we use the publicly available Python code provided by the authors from [20] and [37] respectively. For STOMPI we use the code from *stumpy* [26] Python package. Damp is implemented in Matlab and is publicly available [43].

3. TSF Baselines. We have selected six TSF methods for comparison, including ARIMA [11], DeepAR [34], NBEATS [29], Informer [45], FEDformer [47], and FiLM [46]. ARIMA is a traditional statistical method and the last five are deep learning-based methods, including the most recent ones (e.g., FEDformer and FiLM) with state-of-the-art forecasting accuracies on the evaluated datasets. For ARIMA we use the implementation from *statsforecast* [9] Python package. For DeepAR and NBEATS, we use the code from *pytorch-forecasting* [8]. For Informer [45], FEDformer [47] and FiLM [46], we use the Python codes from the authors, which are publicly available.

5.1.3 Evaluation Metrics. We also have multiple evaluation metrics for different time series analysis tasks.

1. Decomposition Metric. We adopt the Mean Absolute Error (MAE) between the decomposed and the ground truth signals for synthetic datasets to evaluate the decomposition quality. Further, we show the visualization of the decomposed signals for both synthetic and real-world STD datasets.

2. Univariate TSAD Metric. Recently, there are studies arguing that the traditional evaluation metrics (e.g., Precision, Recall, and AUC) of TSAD methods are too sensitive to the noisy labels [24, 30]. Therefore, we use a recently proposed evaluation method VUS-ROC [30] (also from the authors of TSB-UAD) for evaluations. VUS-ROC has been shown to be more robust against noisy labels than other TSAD evaluation metrics [30]. Moreover, on KDD21 dataset [17] we use the score function from the KDD CUP 2021 TSAD competition for evaluation. Since there is only one anomaly event for each time series in KDD21 datasets, it just checks whether the top-ranked anomaly point is among the neighborhood of the labeled anomaly event.

3. Univariate TSF Metric. We adopt the MAE between the real and the predicted values in the test set for the evaluation of the forecasting task.

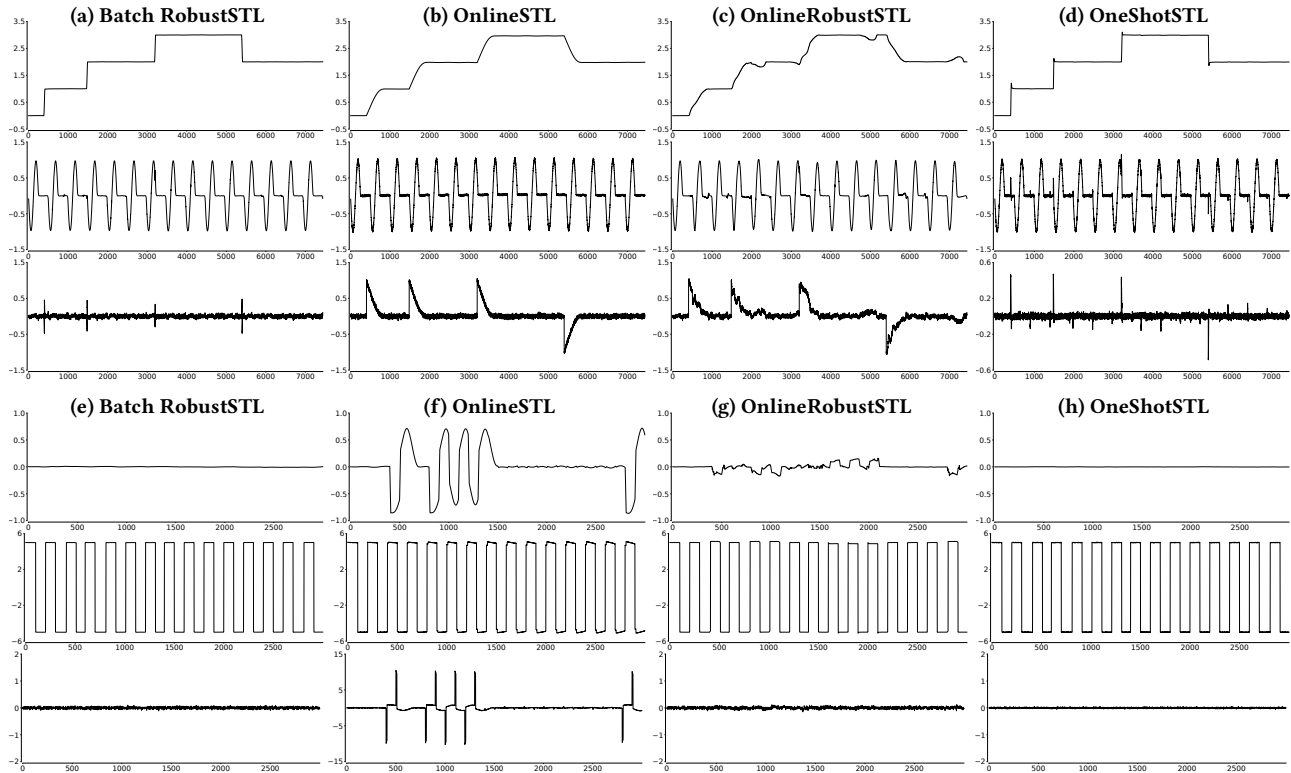


Figure 5: Comparison of decomposition results on Syn1 ((a)-(d)) and Syn2 ((e)-(h)).

5.1.4 Hyper-parameters setting. All STD and TSAD methods (except the deep learning methods) need the period length T as the input. For synthetic datasets, we use the ground truth value $T = 500$ for **Syn1** and $T = 250$ for **Syn2**. For real-world datasets, we use the `find_length` function from TSB-UAD package [3] to detect T , which is based on auto-correlation.

1. STD Methods Setting. STL does not need any other hyper-parameters. RobustSTL needs a dozen of them and we use the authors-recommended default values [7]. We set $\alpha = 0.7$ for OnlineSTL as the authors suggested [28]. The key hyper-parameters for OneShotSTL are the λ_1 and λ_2 which control the smoothness of the decomposed trend. We always set them the same with $\lambda = \lambda_1 = \lambda_2$ and tune λ as follows. On the training data, we perform STL and OneShotSTL with $\lambda \in \{10^0, 10^1, \dots, 10^4\}$. Then we pick up the one that is the closest (the smallest MAE) to the results from STL. We set the seasonality shift window $H = 20$, the maximum iteration $I = 8$ and $n = 5$ for NSigma, if not otherwise specified.

2. TSAD Methods Setting. We use the hyper-parameters suggested by the authors, which usually are the default ones. For methods implemented in TSB-UAD, their default hyper-parameters have already been used for the evaluation in the TSB-UAD benchmark paper [31].

3. TSF Methods Setting. For Informer, FEDformer, and FiLM, we use the hyper-parameters that have already been extensively tuned on the evaluated datasets by the authors. For Arima, we use the parameter-free AutoArima from `statsforecast` [9] package. For DeepAR and NBEATS, hyper-parameters of the initial learning rate and hidden sizes are tuned on the validation sets by following the

instructions [8]. For OnlineSTL and OneShotSTL, we tune α and λ_1 and λ_2 using the validation sets respectively.

5.1.5 Hardware and setting. All experiments of NSigma, DAMP, OnlineSTL, and OneShotSTL are performed on a Macbook Pro 2018 with 4 CPU cores (Intel Core i5) and 16 GiB memories. Experiments for the other methods are conducted on Alibaba cloud where an ECS (ecs.gn6e, 48 CPU cores, 368 GiB memories, NVIDIA V100×4 GPU cards) is used. For all the deep learning-based methods, an NVIDIA V100 GPU card is used for efficient training and testing.

5.2 How accurate is OneShotSTL?

We first compare the quality of the decomposition from OneShotSTL with the previous STD methods. Figure 5 depicts the decomposition results on **Syn1** (from (a) to (d)) and **Syn2** (from (e) to (h)) respectively. As can be seen from sub-figures (a) to (d), OnlineSTL and OnlineRobustSTL output too smooth trend on **Syn1** and fail to recover the ground truth signal of trend, while OneShotSTL successfully handles such cases. Further, OnlineSTL fails to handle seasonality shift on **Syn2** and outputs significantly different trend and residual signals. In comparison, OneShotSTL can deal with such cases perfectly as the batch method RobustSTL. Moreover, Table 2 depicts the MAE between the ground truth signals and the decomposed signals from both batch and online methods. As can be seen, RobustSTL and OneShotSTL (shown in bold) are the best-performing batch and online methods respectively on both synthetic datasets. Their MAE scores are quite close, which are significantly better than all the other comparisons.

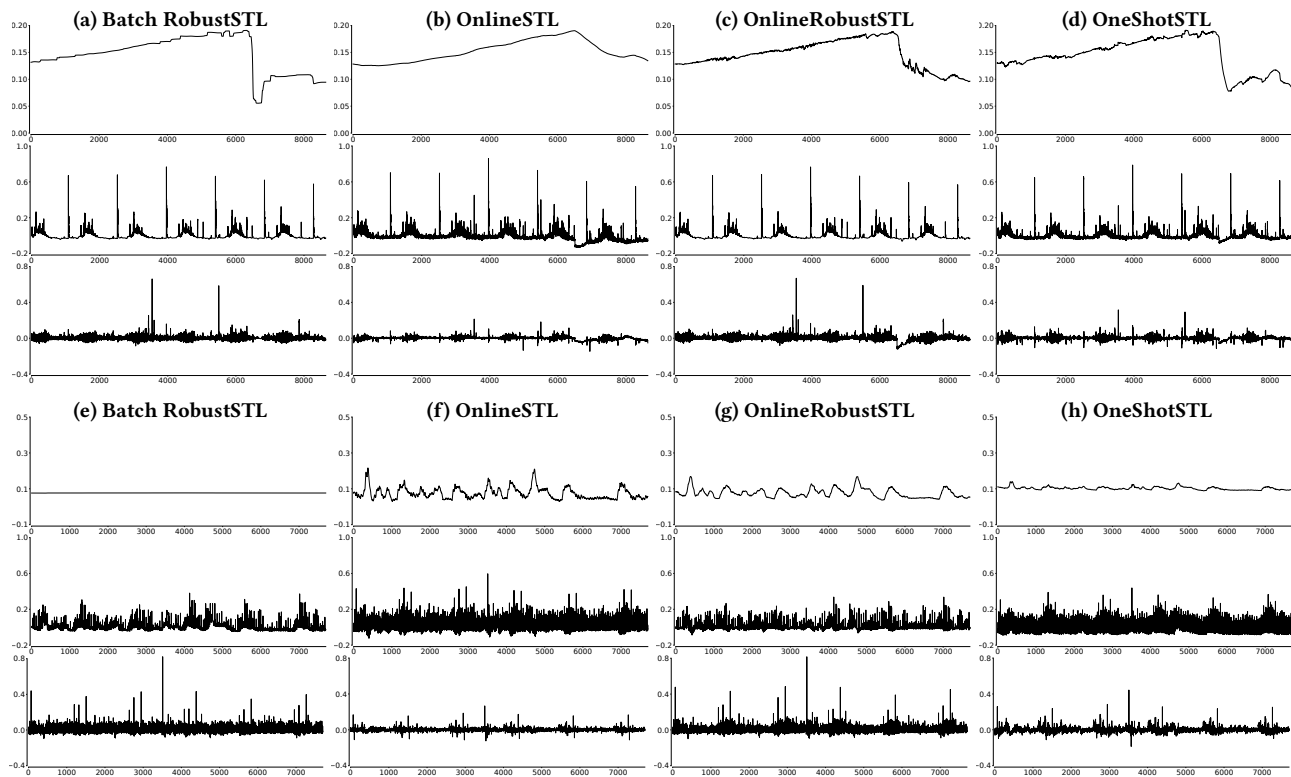


Figure 6: Visualization of decomposition results on Real1 ((a)-(d)) and Real2 ((e)-(h)) datasets.

Table 2: Decomposition comparison on synthetic datasets

| Data | Type | Algorithm | Trend (MAE) | Seasonal (MAE) | Residual (MAE) |
|------|-------------------|------------------|--------------|----------------|----------------|
| Syn1 | Batch | STL | 0.134 | 0.015 | 0.144 |
| | | RobustSTL | 0.004 | 0.013 | 0.016 |
| | Online | Window-STL | 0.134 | 0.092 | 0.174 |
| | | OnlineSTL | 0.104 | 0.023 | 0.093 |
| | | Window-RobustSTL | 0.045 | 0.018 | 0.046 |
| | | OnlineRobustSTL | 0.131 | 0.033 | 0.123 |
| | OneShotSTL | 0.007 | 0.014 | 0.019 | |
| Syn2 | Batch | STL | 0.084 | 0.433 | 0.505 |
| | | RobustSTL | 0.004 | 0.004 | 0.004 |
| | Online | Window-STL | 0.084 | 0.313 | 0.313 |
| | | OnlineSTL | 0.225 | 0.374 | 0.571 |
| | | Window-RobustSTL | 0.032 | 0.031 | 0.006 |
| | | OnlineRobustSTL | 0.037 | 0.031 | 0.013 |
| | OneShotSTL | 0.004 | 0.013 | 0.013 | |

Next, we visualize the decomposed signals of real-world datasets **Real1** and **Real2** in Figure 6 (a) to (d) and (e) to (h) respectively. As can be seen from the sub-figures (a) to (d) of Figure 6, the decomposed trend signal of **Real1** from OneShotSTL is closer to that from RobustSTL, which can better discover the abrupt change of the trend than OnlineSTL and OnlineRobustSTL. Further, from the results of **Real2** in sub-figures (e) to (h) of Figure 6 we can see that there are strong variations of the decomposed trend signals from OnlineSTL and OnlineRobustSTL. These may increase the possibility of false alarms for anomaly detection on the decomposed trend signal. In comparison, OneShotSTL can handle it better.

5.3 How Fast is OneShotSTL?

Next, we compare the efficiency of different STD methods. Remember that OneShotSTL is the only online method with $O(1)$ update complexity. All the other methods have an update complexity of $O(T)$. Specifically, we build a new synthetic dataset of 200,000 points by repeating **Syn1**. Please note that the runtime of all the STD methods only depends on the size and period length T of the input time series. Then we evaluate the decomposition methods with different input $T \in \{100, 200, 400, 800, 1600, 3200, 6400, 12800\}$ and report the average latency of processing one data point. We evaluate this range of T because, in the context of real-time metric monitoring scenarios in AIops, there is usually a daily or weekly seasonality pattern. Then for minute data, the period length is 1,440 and 10,080 for the daily and weekly seasonality respectively.

The results are shown in Figure 7. All methods except OneShotSTL scale linearly with the increase of T . OneShotSTL is the only method with a constant latency (around $20\mu s$) to process one data point, which agrees with the theoretical analysis in Section 3. Approximately, it can process up to 50,000 time series per second using a single CPU core. OnlineSTL performs better when $T < 800$, while OneShotSTL is faster when $T > 800$. When $T = 12,800$, OnlineSTL needs about $450\mu s$ to decompose one data point, which is more than 20 times slower than OneShotSTL. The other three methods Window-STL, Window-RobustSTL, and OnlineRobustSTL are at least two orders of magnitude slower than OnlineSTL and OneShotSTL. Therefore, in the following evaluation of the TSAD and TSF tasks, we only include OnlineSTL and OneShotSTL.

Table 3: Univariate TSAD Results (VUS-ROC) on sixteen datasets from TSB-UAD benchmark. A higher VUS-ROC indicates better performance. The ranking of the method is shown in the bracket and the best results are shown in bold.

| Method | LSTM | USAD | TranAD | NormA | SAND | STOMPI | DAMP | NSigma | OnlineSTL | OneShotSTL |
|-----------------|-------------|------------------|------------|------------------|--------------|------------|------------------|------------------|------------------|------------------|
| Daphnet | 0.476 (7) | 0.653 (3) | 0.548 (6) | 0.462 (10) | 0.594 (5) | 0.465 (7) | 0.475 (8) | 0.605 (4) | 0.696 (2) | 0.735 (1) |
| Dodgers | 0.743 (9) | 0.872 (1) | 0.768 (7) | 0.815 (4) | 0.788 (6) | 0.794 (5) | 0.630 (10) | 0.825 (2) | 0.756 (8) | 0.819 (3) |
| ECG | 0.745 (9) | 0.936 (1) | 0.759 (7) | 0.935 (2) | 0.884 (4) | 0.891 (3) | 0.864 (5) | 0.715 (10) | 0.754 (8) | 0.767 (6) |
| Genesis | 0.507 (10) | 0.827 (1) | 0.742 (4) | 0.669 (5) | 0.507 (9) | 0.530 (8) | 0.644 (7) | 0.823 (2) | 0.666 (6) | 0.762 (3) |
| GHL | 0.627 (8) | 0.913 (1) | 0.831 (4) | 0.759 (5) | 0.727 (7) | 0.603 (9) | 0.553 (10) | 0.732 (6) | 0.866 (3) | 0.872 (2) |
| IOPS | 0.743 (3) | 0.740 (5) | 0.690 (6) | 0.503 (9) | 0.487 (10) | 0.550 (8) | 0.616 (7) | 0.774 (1) | 0.741 (4) | 0.749 (2) |
| MGAB | 0.683 (2) | 0.628 (10) | 0.647 (4) | 0.628 (9) | 0.637 (5) | 0.628 (8) | 0.731 (1) | 0.665 (3) | 0.634 (6) | 0.628 (7) |
| MITDB | 0.613 (10) | 0.745 (5) | 0.667 (8) | 0.824 (1) | 0.792 (2) | 0.746 (4) | 0.773 (3) | 0.671 (7) | 0.664 (9) | 0.685 (6) |
| NAB | 0.510 (10) | 0.631 (1) | 0.551 (6) | 0.580 (4) | 0.519 (9) | 0.520 (8) | 0.540 (7) | 0.570 (5) | 0.584 (3) | 0.590 (2) |
| NASA-MSL | 0.613 (8) | 0.699 (1) | 0.650 (6) | 0.645 (7) | 0.695 (3) | 0.606 (10) | 0.609 (9) | 0.694 (5) | 0.696 (2) | 0.688 (5) |
| NASA-SMAP | 0.642 (8) | 0.711 (4) | 0.551 (10) | 0.818 (1) | 0.800 (2) | 0.711 (3) | 0.665 (6) | 0.642 (7) | 0.601 (9) | 0.666 (5) |
| Occupancy | 0.532 (6) | 0.421 (7) | 0.845 (2) | 0.626 (5) | 0.254 (10) | 0.399 (9) | 0.408 (8) | 0.770 (4) | 0.855 (1) | 0.840 (3) |
| Opportunity | 0.564 (6) | 0.481 (7) | 0.454 (8) | 0.808 (1) | 0.777 (2) | 0.767 (3) | 0.707 (4) | 0.586 (5) | 0.418 (9) | 0.404 (10) |
| SensorScope | 0.546 (10) | 0.555 (8) | 0.573 (7) | 0.632 (4) | 0.551 (9) | 0.587 (6) | 0.609 (5) | 0.668 (2) | 0.646 (3) | 0.672 (1) |
| SMD | 0.606 (6) | 0.697 (4) | 0.699 (3) | 0.612 (9) | 0.580 (10) | 0.622 (8) | 0.635 (7) | 0.680 (5) | 0.732 (2) | 0.749 (1) |
| SVDB | 0.650 (7) | 0.731 (5) | 0.647 (10) | 0.924 (1) | 0.896 (2) | 0.830 (3) | 0.815 (4) | 0.649 (8) | 0.647 (9) | 0.690 (6) |
| YAHOO | 0.768 (6) | 0.626 (9) | 0.677 (8) | 0.915 (1) | 0.895 (2) | 0.544 (10) | 0.810 (4) | 0.761 (7) | 0.833 (3) | 0.796 (5) |
| Avg. VUS-ROC | 0.624 (10) | 0.698 (3) | 0.664 (7) | 0.713 (1) | 0.669 (6) | 0.634 (9) | 0.652 (8) | 0.695 (4) | 0.693 (5) | 0.713 (1) |
| Avg. Rank | 7.35 (10) | 4.29 (2) | 6.23 (8) | 4.58 (3) | 5.70 (6) | 6.70 (9) | 6.17 (7) | 4.82 (4) | 5.11 (5) | 4.00 (1) |
| Avg. Time (sec) | 13995 s (9) | 7465 s (8) | 545 s (4) | 1164 s (5) | 23601 s (10) | 3689 s (6) | 5298 s (7) | 2 s (1) | 61 s (2) | 160 s (3) |

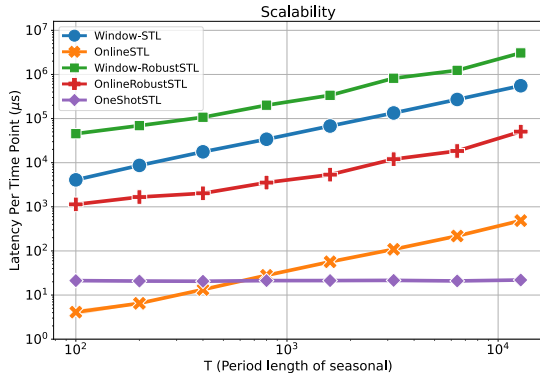


Figure 7: Comparison of average latency per data point with different input of $T \in \{100, 200, 400, 800, 1600, 3200, 6400, 12800\}$.

5.4 How do the methods compare in TSAD?

In this subsection, we apply STD methods to the TSAD task and compare them with the corresponding state-of-the-art methods.

Firstly, we evaluate seventeen datasets from the TSB-UAD benchmark with results shown in Table 3. The average VUS-ROC score from a dataset is displayed in each row of the table, along with the method’s ranking, which is indicated in brackets. The average VUS-ROC, ranking, and execution time for all seventeen datasets are shown in the table’s last three rows. The table shows that OneShotSTL surpasses majority of the comparison methods with an average VUS-ROC of 0.713 and an average ranking of 4.00. The table also demonstrates that no particular approach consistently outperforms the others. Specifically, USAD, NormA and OneShotSTL perform the best on 6, 5, and 3 datasets, respectively. Notably, we also observe that these three approaches are low-ranked algorithms on other datasets. OneShotSTL is higher in both the average ranking and the VUS-ROC score than the other methods because it performs consistently across various datasets. Surprisingly, simple NSigma

Table 4: Univariate TSAD Results on KDD21 datasets

| Data | Type | Method | Score | Time |
|-----------------|--------|----------------|--------------|-----------|
| KDD21 | Deep | LSTM | 0.460 | 1842 min |
| | | USAD | 0.168 | 54 min |
| | | TranAD | 0.196 | 54 min |
| | MP | NormA | 0.500 | 24 min |
| | | STOMPI | 0.360 | 68 min |
| | | SAND | 0.388 | 13534 min |
| | | DAMP | 0.512 | 384 min |
| | STD | NSigma | 0.132 | < 0.1 min |
| | | OnlineSTL | 0.268 | 4.3 min |
| | | OneShotSTL | 0.288 | 7.5 min |
| | STD+MP | NSigma+DAMP | 0.324 | 4.7 min |
| | | OnlineSTL+DAMP | 0.408 | 7.2 min |
| OneShotSTL+DAMP | | 0.508 | 9.5 min | |

provides competitive results with an average VUS-ROC score of 0.695, which outperforms many sophisticated methods like SAND (0.669) and DAMP (0.652). Furthermore, NSigma is also the fastest method with an average runtime of only 2 seconds followed by OnlineSTL (61 seconds) and OneShotSTL (160 seconds). The other methods take thousands or even tens of thousands of seconds to complete.

Next, we evaluate KDD21 dataset. The average accuracy across 250 time series and total time for training/testing are shown in Table 4. LSTM provides good accuracy of 0.460, but its runtime (1842 minutes) is significantly longer than the other methods except SAND (13534 minutes). The other two deep learning-based methods are faster, but they do not perform well on KDD21 datasets in terms of accuracy. Online TSAD method DAMP performs the best with an accuracy of 0.512. However, DAMP takes 384 minutes (around 6 hours) to finish detecting all the datasets. Further, OnlineSTL and OneShotSTL can significantly improve the results of NSigma (0.132) to 0.268 and 0.288 respectively. However, their accuracies are still significantly worse than DAMP. This is because STD methods can

Table 5: Univariate TSF results (MAE) on six datasets with forecasting lengths {96, 192, 336, 720} ({24, 36, 48, 60} for Illness). The ranking of the method is shown in the bracket and the best results are highlighted in bold.

| Method | | FiLM | FEDformer | Informer | NBEATS | DeepAR | AutoArima | OnlineSTL | OneShotSTL |
|-----------------|-----|------------------|------------------|------------|------------------|------------|------------------|------------------|------------------|
| ETTm2 | 96 | 0.189 (1) | 0.189 (1) | 0.225 (5) | 0.201 (3) | 0.942 (8) | 0.379 (7) | 0.375 (6) | 0.211 (4) |
| | 192 | 0.233 (1) | 0.245 (3) | 0.283 (5) | 0.248 (4) | 0.993 (8) | 0.419 (7) | 0.391 (6) | 0.244 (2) |
| | 336 | 0.274 (2) | 0.279 (4) | 0.336 (5) | 0.274 (2) | 0.442 (8) | 0.437 (7) | 0.419 (6) | 0.273 (1) |
| | 720 | 0.323 (2) | 0.325 (3) | 0.435 (4) | 0.468 (6) | 1.028 (8) | 0.475 (7) | 0.452 (5) | 0.321 (1) |
| Electricity | 96 | 0.247 (1) | 0.370 (3) | 0.538 (5) | 0.387 (4) | 0.641 (6) | 0.898 (7) | 1.031 (8) | 0.331 (2) |
| | 192 | 0.258 (1) | 0.386 (3) | 0.558 (5) | 0.458 (4) | 0.698 (6) | 0.925 (7) | 1.013 (8) | 0.355 (2) |
| | 336 | 0.283 (1) | 0.431 (3) | 0.613 (5) | 0.494 (4) | 0.747 (6) | 1.013 (7) | 1.037 (8) | 0.389 (2) |
| | 720 | 0.341 (1) | 0.484 (4) | 0.682 (5) | 0.483 (3) | 0.796 (6) | 1.096 (8) | 1.070 (7) | 0.444 (2) |
| Exchange | 96 | 0.259 (4) | 0.284 (5) | 0.615 (8) | 0.358 (6) | 0.597 (7) | 0.241 (1) | 0.241 (1) | 0.252 (3) |
| | 192 | 0.352 (4) | 0.420 (7) | 0.912 (8) | 0.387 (5) | 0.419 (6) | 0.321 (1) | 0.327 (2) | 0.334 (3) |
| | 336 | 0.461 (4) | 0.511 (5) | 0.984 (8) | 0.652 (6) | 0.920 (7) | 0.432 (1) | 0.433 (2) | 0.436 (3) |
| | 720 | 0.708 (4) | 0.832 (6) | 1.072 (7) | 1.114 (8) | 0.710 (5) | 0.681 (3) | 0.626 (2) | 0.624 (1) |
| Traffic | 96 | 0.215 (3) | 0.263 (4) | 0.353 (5) | 0.206 (2) | 0.614 (6) | 1.036 (7) | 1.465 (8) | 0.181 (1) |
| | 192 | 0.199 (2) | 0.265 (4) | 0.376 (5) | 0.211 (3) | 0.992 (6) | 1.101 (7) | 1.461 (8) | 0.181 (1) |
| | 336 | 0.212 (3) | 0.266 (4) | 0.387 (5) | 0.209 (2) | 1.105 (6) | 1.138 (7) | 1.470 (8) | 0.182 (1) |
| | 720 | 0.252 (3) | 0.286 (4) | 0.436 (5) | 0.243 (2) | 1.166 (6) | 1.213 (7) | 1.461 (8) | 0.199 (1) |
| Weather | 96 | 0.026 (4) | 0.046 (7) | 0.044 (6) | 0.022 (1) | 0.059 (8) | 0.025 (3) | 0.031 (5) | 0.024 (2) |
| | 192 | 0.029 (4) | 0.059 (8) | 0.040 (6) | 0.026 (2) | 0.047 (7) | 0.028 (3) | 0.032 (5) | 0.025 (1) |
| | 336 | 0.030 (2) | 0.050 (8) | 0.049 (7) | 0.031 (3) | 0.035 (6) | 0.031 (3) | 0.033 (5) | 0.026 (1) |
| | 720 | 0.037 (5) | 0.091 (8) | 0.042 (7) | 0.033 (2) | 0.033 (2) | 0.035 (4) | 0.037 (6) | 0.030 (1) |
| Illness | 24 | 0.538 (2) | 0.629 (3) | 2.050 (8) | 0.467 (1) | 0.726 (5) | 0.821 (6) | 0.917 (7) | 0.682 (4) |
| | 36 | 0.481 (1) | 0.604 (3) | 1.916 (8) | 0.519 (2) | 0.810 (5) | 0.928 (7) | 0.840 (6) | 0.757 (4) |
| | 48 | 0.584 (1) | 0.699 (3) | 1.846 (8) | 0.678 (2) | 0.861 (6) | 0.941 (7) | 0.843 (5) | 0.773 (4) |
| | 60 | 0.644 (1) | 0.828 (4) | 2.057 (8) | 0.773 (2) | 0.873 (5) | 0.913 (6) | 0.963 (7) | 0.826 (3) |
| Avg. MAE | | 0.308 (1) | 0.368 (3) | 0.702 (7) | 0.373 (4) | 0.677 (6) | 0.647 (5) | 0.707 (8) | 0.337 (2) |
| Avg. Ranking | | 2.37 (2) | 4.45 (4) | 6.16 (7) | 3.29 (3) | 6.20 (8) | 5.41 (5) | 5.79 (6) | 2.08 (1) |
| Avg. Time (sec) | | 7860 s (7) | 2110 s (5) | 2003 s (4) | 790 s (3) | 4709 s (6) | 8779 s (8) | 0.3 s (1) | 0.3 s (1) |

only handle seasonal patterns but the KDD21 dataset contains non-seasonal patterns. But OnlineSTL and OneShotSTL are much faster than DAMP. Therefore, we further integrate them with DAMP by using them as a pre-filtering step. For each time series, we filter the top-ranked 1% of testing points by the STD methods and give them to DAMP for detection. By doing this, OneShotSTL can reduce the computation time of DAMP from 384 minutes to 9.5 minutes (more than 40 times faster) with negligible loss of accuracy (from 0.512 to 0.508). Meanwhile, we can observe that OneShotSTL outperforms OnlineSTL (0.408) while being used as filtering for DAMP.

Based on the above results, we conclude that OneShotSTL outperforms OnlineSTL for TSAD tasks by providing better detection accuracies in most cases. Moreover, OneShotSTL and matrix profile-based methods perform well on different datasets of the TSB-UAD benchmark, but OneShotSTL is from 10 to 100 times faster.

5.5 How do the methods compare in TSF?

In this subsection, we apply STD methods to the TSF task and compare them with the corresponding state-of-the-art methods.

Table 5 depicts the performance of 8 comparison methods on 6 benchmark datasets with forecasting length {96, 192, 336, 720} ({24, 36, 48, 60} for Illness). Each row of the table shows the prediction errors (measured by MAE) and the ranking of each method for each setting is shown in the bracket with the best results highlighted in bold. The last 3 rows of the table represent the average MAE, ranking and execution time (including both training and testing time) across all the settings. First, we can see from the table

that, regarding prediction error, FiLM and OneShotSTL are the top-two performers with average MAE of 0.308 and 0.337 respectively, followed by FEDformer (0.368) and NBEATS (0.373). In terms of average ranking, OneShotSTL is the best method with an average ranking of 2.08, followed by FiLM (2.37), NBEATS (3.29), and FEDformer (4.45). Specifically, OneShotSTL performs better on datasets with strong seasonality (ETTm2, Electricity, Traffic, and Weather) and predicts relatively worse on Exchange and Illness with weak or no seasonality. Furthermore, STD methods have great advantages in speed over comparison TSF methods. The average runtime of OnlineSTL and OneShotSTL is less than 1 second using a single CPU core on a commodity machine, while the processing time for deep learning-based methods is from hundreds to thousands of seconds using an NVIDIA V100 GPU (more than 1000 times slower).

In summary, the forecasting accuracies of OneShotSTL are comparable to the state-of-the-art deep learning method FiLM on the evaluated datasets. Meanwhile, its runtime is more than 1000 times faster than the non-STD methods.

5.6 Ablation Study of OneShotSTL

In this subsection, we make ablation studies on three input parameters of OneShotSTL: periodicity length T , maximum seasonality shift length H , and number of iterations I .

Firstly, we investigate how the errors in estimating T can impact the final performance. Remember that T is automatically discovered during the initialization phase of OneShotSTL. Thus, we use different values of $\Delta T = \{0, 5, 10, 15, 20\}$ and apply $T \leftarrow T + \Delta T$ as

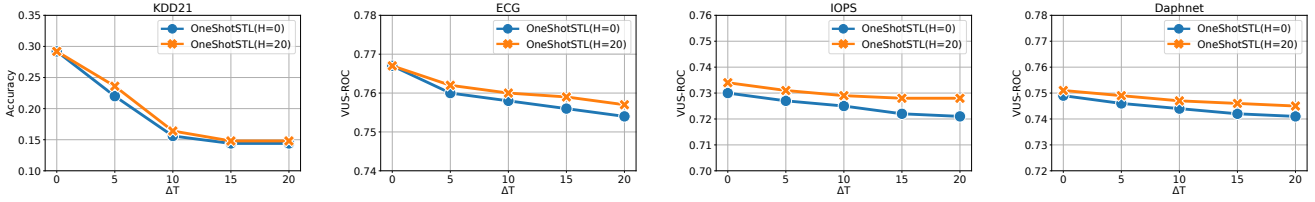


Figure 8: Ablation Study of hyper-parameters T and H of OneShotSTL on TSAD task. We add noise to T by $\Delta T \in \{0, 5, 10, 15, 20\}$. Y-axis represents the accuracy and VUS-ROC scores (higher the better).

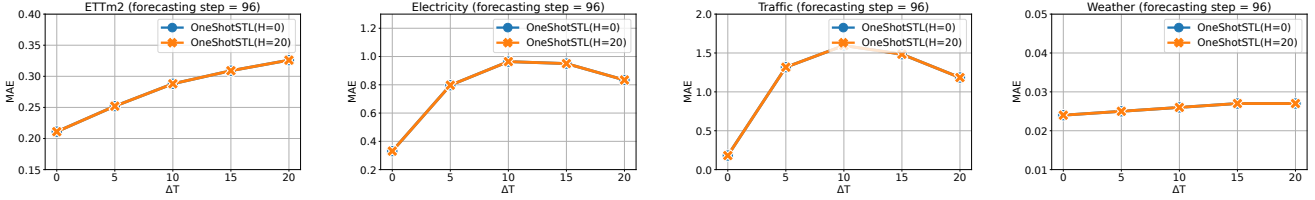


Figure 9: Ablation Study of hyper-parameters T and H of OneShotSTL on TSF task. We add noise to T by $\Delta T \in \{0, 5, 10, 15, 20\}$. Y-axis represents the MAE (lower the better).

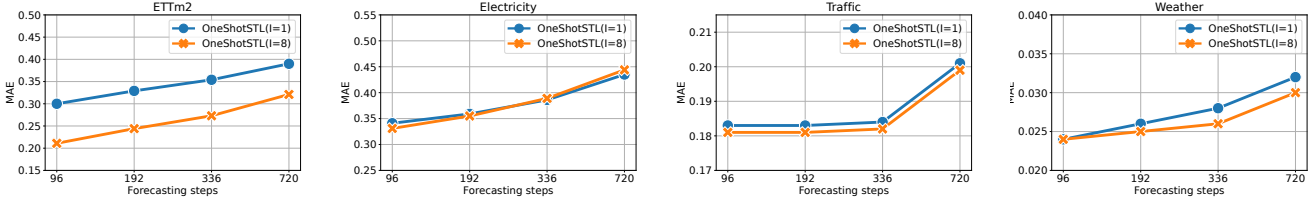


Figure 10: Ablation Study of hyper-parameter I of OneShotSTL on TSF task. Y-axis represents the MAE (lower the better).

the periodicity length parameter. Meanwhile, for each dataset, we compare OneShotSTL with a seasonality shift window of $H = 0$ and $H = 20$ respectively. Figure 8 shows the performance of OneShotSTL with different ΔT on four TSAD datasets. As can be seen from Figure 8, in all cases, OneShotSTL performs better with $H = 20$. Moreover, the performance of OneShotSTL degrades with the increase of ΔT for all four datasets. However, on KDD21 the score of OneShotSTL drops quickly immediately after $\Delta T = 5$, while on the other three datasets the performance of OneShotSTL drops slowly. Further, we evaluate the impact of T on TSF tasks in the same way, with the results shown in Figure 9. From the figure, we can see that the prediction error of OneShotSTL significantly increases on all datasets for both cases with $H = 0$ and $H = 20$. This is probably because we can only correct T_c by searching for the optimal value within $[T - H, T + H]$, which works for TSAD tasks since we only target on the newly arrived data point. However, for TSF tasks, since we cannot correct the future T_c , and thus just use T directly to forecast the future data ($\hat{y}_{t+i} = \tau_{t-1} + v_{t\%T+i}$ in Section 4).

Next, we evaluate the impact of input parameter I for OneShotSTL. Figure 10 shows the MAE of OneShotSTL with $I = 8$ and $I = 1$ using different forecasting steps with the same $H = 20$. Due to limited space, we omit the results of the ablation study of I on TSAD tasks, where we get similar observations. From Figure 10, we can see that with $I = 8$ OneShotSTL more often produces lower prediction errors than that with $I = 1$. Specifically, on ETTm2 datasets, the margin is large, while on the other three datasets we observe minor improvements with $I = 8$. In general, setting a larger I will improve the quality of decomposition, but sacrifice the speed.

6 CONCLUSION

In this work, we propose an accurate and efficient online STD algorithm OneShotSTL that can decompose time series online with an update complexity of $O(1)$. It can significantly reduce the processing time complexity of the existing methods that often require $O(T)$, e.g., OnlineSTL [28]. Resultantly, the online update time of a single time point using OneShotSTL is more than 1,000 times faster than the batch STD methods, with accuracy comparable with the best counterparts, e.g., RobustSTL [40]. Extensive experiments on real-world benchmark datasets for downstream tasks, e.g., TSAD and TSF problems, demonstrate that OneShotSTL can be ranging from 10 to more than 1000 times faster than the state-of-the-art TSAD and TSF methods while still providing comparable or even better results.

The method has its limitations. The proposed OneShotSTL assumes the underlying $T_c \in [T - H, T + H]$ for two constants T and H . This may not hold especially when T_c drastically varies across multiple cycles. Adaptively detecting the seasonal cycle T_c online would be of great importance for the online STD problem. In addition, handling missing points or irregularly sampled time series is crucial in practice, which cannot be handled by most of the current STD methods. In the future, we plan to explore the potentials of OneShotSTL for TSF tasks since decomposed trend and seasonal signals could help long sequence forecasting when combined with other techniques. Moreover, integrating OneShotSTL with matrix profile-based methods for online TSAD tasks is also an interesting direction. All these lead to the future research directions of the online STD problem.

REFERENCES

- [1] [n.d.]. Alibaba Cloudmonitor. <https://www.alibabacloud.com/product/cloud-monitor>. Last accessed 2022-09-21.
- [2] [n.d.]. AWS Cloudwatch. <https://aws.amazon.com/cloudwatch/>. Last accessed 2022-09-21.
- [3] [n.d.]. Find length function of TSB-UAD. https://github.com/TheDatumOrg/TSB-UAD/blob/main/TSB_UAD/utlils/slidingWindows.py. Last accessed 2022-12-05.
- [4] [n.d.]. How to Get Started With AIOps. <https://www.gartner.com/smarterwithgartner/how-to-get-started-with-aiops>. Last accessed 2022-09-21.
- [5] [n.d.]. Informer Datasets. <https://github.com/zhouhaoyi/Informer2020>. Last accessed 2022-12-05.
- [6] [n.d.]. Java implementation of STL in stl-decomp-4j package. <https://github.com/ServiceNow/stl-decomp-4j>. Last accessed 2022-09-21.
- [7] [n.d.]. Python implementation of FastRobustSTL in SREWorks. <https://github.com/alibaba/SREWorks/>. Last accessed 2022-09-21.
- [8] [n.d.]. Pytorch-forecasting package. <https://pytorch-forecasting.readthedocs.io/en/stable/index.html>. Last accessed 2022-09-21.
- [9] [n.d.]. Statsforecast package. <https://nixtla.github.io/statsforecast/>. Last accessed 2022-09-21.
- [10] [n.d.]. Supplementary material. <https://github.com/xiao-he/OneShotSTL/>. Last accessed 2023-02-17.
- [11] Adebisi A. Ariyo, Adewumi O. Adewumi, and Charles K. Ayo. 2014. Stock Price Prediction Using the ARIMA Model. In *Proceedings of the 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation (UKSIM '14)*. IEEE Computer Society, USA, 106–112. <https://doi.org/10.1109/UKSim.2014.67>
- [12] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. 2020. USAD: UnSupervised Anomaly Detection on Multivariate Time Series. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 3395–3404. <https://doi.org/10.1145/3394486.3403392>
- [13] Amir Beck. 2015. On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes. *SIAM Journal on Optimization* 25, 1 (2015), 185–209.
- [14] Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. 2021. Unsupervised and scalable subsequence anomaly detection in large data series. *VLDB J.* 30, 6 (2021), 909–931. <https://doi.org/10.1007/s00778-021-00655-8>
- [15] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. 2021. SAND: Streaming Subsequence Anomaly Detection. *Proc. VLDB Endow.* 14, 10 (2021), 1717–1729. <https://doi.org/10.14778/3467861.3467863>
- [16] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. 1990. STL: A Seasonal-Trend Decomposition Procedure Based on Loess (with Discussion). *Journal of Official Statistics* 6 (1990), 3–73.
- [17] Dau, Hoang Anh and Keogh, Eamonn and Kamgar, Kaveh and Yeh, Chin-Chia Michael and Zhu, Yan and Gharghabi, Shaghayegh and Ratanamahatana, Chotirat Ann and Yanping and Hu, Bing and Begum, Nurjahan and Bagnall, Anthony and Mueen, Abdullah and Batista, Gustavo, and Hexagon-ML. 2018. The UCR Time Series Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.
- [18] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. 2011. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American statistical association* 106, 496 (2011), 1513–1527.
- [19] Alexander Dokumentov, Rob J Hyndman, et al. 2015. STR: A seasonal-trend decomposition procedure based on regression. *Monash University: Melbourne, Australia* (2015), 23.
- [20] Francesco Galati, Julien Audibert, and Maria A. Zuluaga. [n.d.]. Python implementation of USAD. <https://github.com/manigalati/usad>. Last accessed 2022-09-21.
- [21] Sam George. 2019. IoT Signals report: IoT's promise will be unlocked by addressing skills shortage, complexity and security. <https://blogs.microsoft.com/blog/2019/07/30/>. Last accessed 2022-09-21.
- [22] P. J. Green. 1984. Iteratively Reweighted Least Squares for Maximum Likelihood Estimation, and some Robust and Resistant Alternatives. *Journal of the Royal Statistical Society. Series B (Methodological)* 46, 2 (1984), 149–192. <http://www.jstor.org/stable/2345503>
- [23] Xiao He, Francesco Alesiani, and Ammar Shaker. 2019. Efficient and Scalable Multi-Task Regression on Massive Number of Tasks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 3763–3770. <https://doi.org/10.1609/aaai.v33i01.33013763>
- [24] Siwon Kim, Kukjin Choi, Hyun-Soo Choi, Byunghan Lee, and Sungroh Yoon. 2022. Towards a Rigorous Evaluation of Time-Series Anomaly Detection. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelfth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 7194–7201. <https://ojs.aaai.org/index.php/AAAI/article/view/20680>
- [25] Seung-Jean Kim, Kwangmoo Koh, Stephen Boyd, and Dmitry Gorinevsky. 2009. ℓ_1 trend filtering. *SIAM review* 51, 2 (2009), 339–360.
- [26] Sean M. Law. 2019. STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining. *The Journal of Open Source Software* 4, 39 (2019), 1504.
- [27] Yue Lu, Renjie Wu, Abdullah Mueen, Maria A. Zuluaga, and Eamonn J. Keogh. 2022. Matrix Profile XXIV: Scaling Time Series Anomaly Detection to Trillions of Datapoints and Ultra-fast Arriving Data Streams. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, Aidong Zhang and Huzefa Rangwala (Eds.). ACM, 1173–1182. <https://doi.org/10.1145/3534678.3539271>
- [28] Abhinav Mishra, Ram Sriharsha, and Sichen Zhong. 2022. OnlineSTL: Scaling Time Series Decomposition by 100x. *Proc. VLDB Endow.* 15, 7 (2022), 1417–1425. <https://www.vldb.org/pvldb/vol15/p1417-mishra.pdf>
- [29] Boris N. Oreshkin, Dmitri Carпов, Nicolas Chapados, and Yoshua Bengio. 2020. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. <https://openreview.net/forum?id=r1ecqn4YwB>
- [30] John Paparrizos, Paul Boniol, Themis Palpanas, Ruy S Tsay, Aaron Elmore, and Michael J Franklin. 2022. Volume Under the Surface: A New Accuracy Evaluation Measure for Time-Series Anomaly Detection. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2774–2787.
- [31] John Paparrizos, Yuhao Kang, Paul Boniol, Ruy S. Tsay, Themis Palpanas, and Michael J. Franklin. 2022. TSB-UAD: An End-to-End Benchmark Suite for Univariate Time-Series Anomaly Detection. *Proc. VLDB Endow.* 15, 8 (2022), 1697–1711. <https://www.vldb.org/pvldb/vol15/p1697-paparrizos.pdf>
- [32] Daehyung Park, Yuuna Hoshi, and Charles C. Kemp. 2018. A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder. *IEEE Robotics Autom. Lett.* 3, 2 (2018), 1544–1551. <https://doi.org/10.1109/LRA.2018.2801475>
- [33] Tom Puech, Matthieu Bousard, Anthony D'Amato, and Gaëtan Millerand. 2019. A Fully Automated Periodicity Detection in Time Series. In *Advanced Analytics and Learning on Temporal Data - 4th ECML PKDD Workshop, AALTD 2019, Würzburg, Germany, September 20, 2019, Revised Selected Papers (Lecture Notes in Computer Science)*, Vincent Lemaire, Simon Malinowski, Anthony J. Bagnall, Alexis Bondu, Thomas Guyet, and Romain Tavenard (Eds.), Vol. 11986. Springer, 43–54. https://doi.org/10.1007/978-3-030-39098-3_4
- [34] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191.
- [35] Sean J. Taylor and Benjamin Letham. 2017. Forecasting at Scale. *PeerJ Prepr.* 5 (2017), e3190. <https://doi.org/10.7287/peerj.preprints.3190v1>
- [36] Maximilian Toller, Tiago Santos, and Roman Kern. 2019. SAZED: parameter-free domain-agnostic season length estimation in time series data. *Data Min. Knowl. Discov.* 33, 6 (2019), 1775–1798. <https://doi.org/10.1007/s10618-019-00645-z>
- [37] Shreshth Tuli. [n.d.]. Python implementation of TranAD. <https://github.com/imperial-qore/TranAD>. Last accessed 2022-09-21.
- [38] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. 2022. TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. *Proc. VLDB Endow.* 15, 6 (2022), 1201–1214. <https://www.vldb.org/pvldb/vol15/p1201-tuli.pdf>
- [39] Michail Vlachos, Philip S. Yu, and Vittorio Castelli. 2005. On Periodicity Detection and Structural Periodic Similarity. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, Newport Beach, CA, USA, April 21-23, 2005*, Hillo Kargupta, Jaideep Srivastava, Chandrika Kamath, and Arnold Goodman (Eds.). SIAM, 449–460. <https://doi.org/10.1137/1.9781611972757.40>
- [40] Qingsong Wen, Jingkun Gao, Xiaomin Song, Liang Sun, Huan Xu, and Shenghuo Zhu. 2018. RobustSTL: A Robust Seasonal-Trend Decomposition Algorithm for Long Time Series. *CoRR abs/1812.01767* (2018). arXiv:1812.01767 <http://arxiv.org/abs/1812.01767>
- [41] Qingsong Wen, Kai He, Liang Sun, Yingying Zhang, Min Ke, and Huan Xu. 2021. RobustPeriod: Robust Time-Frequency Mining for Multiple Periodicity Detection. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2328–2337. <https://doi.org/10.1145/3448016.3452779>
- [42] Qingsong Wen, Zhe Zhang, Yan Li, and Liang Sun. 2020. Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 2203–2213. <https://doi.org/10.1145/3394486.3403271>
- [43] Renjie Wu, Abdullah Mueen, Maria A. Zuluaga, and Eamonn J. Keogh. [n.d.]. Matlab implementation of DAMP. <https://sites.google.com/view/discord-aware-matrix-profile>. Last accessed 2022-09-21.
- [44] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Zachary Zimmerman, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. 2018. Time series joins, motifs, discords and shapelets:

- a unifying view that exploits the matrix profile. *Data Min. Knowl. Discov.* 32, 1 (2018), 83–123. <https://doi.org/10.1007/s10618-017-0519-9>
- [45] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11106–11115.
- [46] Tian Zhou, Ziqing Ma, Xue Wang, Qingsong Wen, Liang Sun, Tao Yao, Wotao Yin, and Rong Jin. 2022. FiLM: Frequency improved Legendre Memory Model for Long-term Time Series Forecasting. *CoRR* abs/2205.08897 (2022). <https://doi.org/10.48550/arXiv.2205.08897> arXiv:2205.08897
- [47] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. 2022. FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.), Vol. 162. PMLR, 27268–27286.