SMiLer: A Semi-Lazy Time Series Prediction System for Sensors

Jingbo Zhou
Interactive Digital Media Institute
National University of Singapore
jzhou@u.nus.edu

Anthony K. H. Tung
School of Computing
National University of Singapore
atung@comp.nus.edu.sg

ABSTRACT

It is useful to predict future values in time series data, for example when there are many sensors monitoring environments such as urban space. The Gaussian Process (GP) model is considered as a promising technique for this setting. However, the GP model requires too high a training cost to be tractable for large data. Though approximate methods have been proposed to improve GP's scalability, they usually can only capture global trends in the data and fail to preserve small-scale patterns, resulting in unsatisfactory performance.

We propose a new method to apply the GP for sensor time series prediction. Instead of (eagerly) training GPs on entire datasets, we custom-build query-dependent GPs on small fractions of the data for each prediction request.

Implementing this idea in practice at scale requires us to overcome two obstacles. On the one hand, a central challenge with such a semi-lazy learning model is the substantial model-building effort at kNN query time, which could lead to unacceptable latency. We propose a novel two-level inverted-like index to support kNN search using the DTW on the GPU, making such "just-in-time" query-dependent model construction feasible for real-time applications.

On the other hand, several parameters should be tuned for each time series individually since different sensors have different data generating processes in diverse environments. Manually configuring the parameters is usually not feasible due to the large number of sensors. To address this, we devise an adaptive auto-tuning mechanism to automatically determine and dynamically adjust the parameters for each time series with little human assistance.

Our method has the following strong points: (a) it can make prediction in real time without a training phase; (b) it can yield superior prediction accuracy; and (c) it can effectively estimate the analytical predictive uncertainty.

To illustrate our points, we present SMiLer, a semi-lazy time series prediction system for sensors. Extensive experiments on real-world datasets demonstrate its effectiveness and efficiency. In particular, by devising a two-level inverted-like index on the GPU with an enhanced lower bound of the DTW, SMiLer accelerates the efficiency of kNN search by one order of magnitude over its baselines. The prediction accuracy of SMiLer is better than the state-of-the-art competitors (up to 10 competitors) with better estimation of predictive uncertainty.

Categories and Subject Descriptors: H.2.8 [Database management]: Database applications-Data mining

Keywords: Semi-lazy learning; Sensors; Time series; Predictive analysis; Gaussian Process; DTW; GPU

1. INTRODUCTION

With sensor data becoming prevalent, time series prediction of sensors is valuable in many applications including event prediction, air pollution forecasting, manufacturing condition monitoring and medical diagnoses. Since statistical regression methods are not powerful enough to handle large varieties of time series in realistic settings, machine learning methods have drawn much attention and are becoming popular for time series prediction.

Among the models, the Gaussian Process (GP) model have received significant interests for time series prediction [70, 34, 50, 16, 20, 35, 72, 23, 40]. This is due to its nonparametric nonlinear property and excellent modeling capability for a wide variety of behavior. In addition, the GP model can easily estimate analytical predictive uncertainty with conceptually simple closed-form expression.

A major limitation of the GP is its poor scalability that scales as $O(n^3)$ where n is the size of training data. To overcome this limitation, some approximation methods are usually adopted to find approximated representation of the whole dataset. The consequence is that the constructed GP models are more influenced by global distribution of the whole time series data, while local behavior and small-scale patterns are not captured. We refer this as the "*information loss*" problem. In addition, the GP models, as well as many other non-linear machine learning models, may also suffer from "*concept drift*" problem. In the case of sensors monitoring dynamic environments such as urban space, since the underlying model generating the data might gradually change, the constructed global GP models might be outdated by the time when sufficient historical data is collected to build the models. In this case, paying a high computational cost to construct a large, global model that fits the whole of the sensor time series may be wasteful.

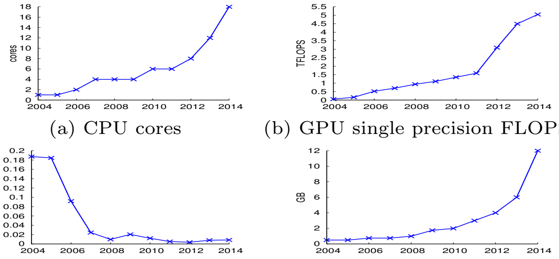
EXAMPLE 1.1 (TRAFFIC SENSOR PREDICTION). *Time series prediction for traffic sensors can be very useful for many*

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2749429.

smart city applications such as abnormal event detection, traffic jam prediction and flow speed prediction [42]. While the standard GP model is not tractable for large data, the approximated GP models have relative low prediction accuracy, and also high training time cost. These observations are supported by our experiment evaluation for traffic sensors prediction (see Fig. 13(a) and Section 6.4.2).

In this paper, we propose a new approach to employing the GP model in sensor time series prediction. Instead of attempting to eagerly train a global GP model on the entire dataset which may suffer from information loss and is not adaptive to concept drift, we propose to build a query-dependent GP model for each prediction request. In general, the methodology of our solution can be considered as a semi-lazy learning approach, which is a hybrid of the eager learning approach (e.g. GPs and SVMs) and the lazy learning approach (e.g. kNN regression). Its framework essentially follows the lazy learning paradigm until the last step, where sophisticated models, i.e. GPs in this paper, are applied on the kNN results of the submitted prediction request.

Fig. 2 shows an illustration of the semi-lazy learning GP prediction idea. We use the time series of a sensor in the last few time steps as the input request to retrieve a set of k-Nearest Neighbor (kNN) time series segments from historical data. The kNNs are then utilized to construct GPs for predicting the future value of the sensor.



(a) CPU cores

(b) GPU single precision FLOPS

(c) CPU memory price

(d) GPU memory size

Figure 1: Trends of computing performance over time.[1]

Fig. 1 illustrates the growth in computing capability over recent years. We believe that such growths in modern hardware will increasingly make semi-lazy learning feasible: cheap and fast storage can support high speed similarity search (e.g. kNN) and powerful computing devices can support efficient online model construction (e.g. GP). The work in this paper is part of our "*Genie and Lamp*" project [1] which aims to provide a systematic investigation into the use of semi-lazy learning for predictive analytics.

Two main technical obstacles must be overcome to make the semi-lazy time series prediction scalable in practice. First, to build the query-dependent GP model, substantial model-building effort at kNN query time could lead to unacceptable latency. As will be explained in Section 4, we define a "*(Continuous) Suffix kNN Search*" problem which tries to identify small fractions of data for each prediction request. A property of the Suffix kNN Search is that each prediction request invokes a set of kNN queries sharing common suffix. To make such kNN search feasible in real-time applications, we develop an efficient search method using the popular Dynamic Time Warping (DTW) distance on the Graphic Processing Unit (GPU). Specifically, we design a novel two-level inverted-like index on the GPU and propose a new enhanced DTW lower bound to accelerate the search process.

Second, several parameters have to be configured. The underlying generating process of time series data may be diverse and continuously changing. As a result, we have to set different parameters for different sensors and even for the same sensor at different times. It is infeasible to manually set parameters for each sensor in a sensor network. To overcome this, we propose an adaptive auto-tuning mechanism to dynamically adjust the parameters for each sensor during continuous prediction without requiring user intervention.

Our method has several appealing advantages:

- It can make time series prediction of sensors in real time without a training phase in contrast to non-linear machine learning models like GPs and SVMs.
- It can yield superior prediction accuracy over several eager learning competitors. Since historical time series data is kept until prediction time in semi-lazy learning approach, a very rich set of models is preserved as part of the data. The query-dependent GP model caters to specifically making prediction for a submitted query without the need to build a generalized model that caters to the whole dataset.
- It can effectively estimate the analytical predictive uncertainty. Compared to traditional kNN regression, our method not only has higher prediction accuracy, but also can provide a closed-formed analytical expression to measure the prediction confidence.

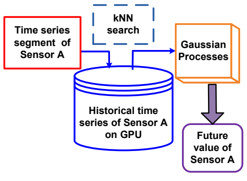

Time series segment of Sensor A
kNN search
Gaussian Processes
Historical time series of Sensor A on GPU
Future value of Sensor A

Figure 2: An illustration for the semi-lazy (query-dependent) GP to time series prediction.

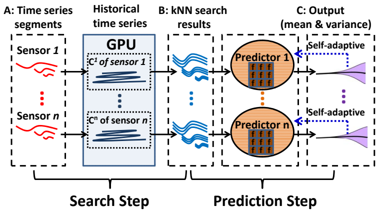


Figure 3: Overview framework of SMiLer.

To make our method feasible, we present **SMiLer**, *a SeMi-Lazy time series prediction system for sensors*. Fig. 3 shows an overall framework of SMiLer comprising of two main steps which overcome two mentioned obstacles respectively. More details about it will be clarified in Section 3.4. In summary, our key contributions are listed as follows.

- We propose a new method to enable the Gaussian Process model for time series prediction of sensors. It brings up a new insight for time series prediction from the semi-lazy learning perspective.
- We contrive a novel two-level inverted-like index on the GPU with an enhanced lower bound of the DTW, which can accelerate the proposed Suffix kNN Search problem to quickly identify small fractions of data for building query-dependent GP models.
- We devise an adaptive auto-tuning mechanism which can automatically learn and adjust the parameters of models over time for each sensor.
- We conduct extensive experiment study on several real-life datasets. SMiLer accelerates the efficiency of kNN search by one order of magnitude over its competitors. The prediction accuracy of SMiLer is better than several state-of-the-art competitors (up to 10 competitors

[1] References of data source are presented in Appendix A.

including GPs, SVMs, kNN and linear models with SGD) with better estimation of predictive uncertainty.

The rest of this paper is organized as follows. Next, we discuss the related work in Section 2, followed by an overview of the framework of SMiLer in Section 3. Then we address the search problem in Section 4 and the model construction problem in Section 5. Finally, we evaluate our system in Section 6, and conclude the paper in Section 7.

2. RELATED WORK

In thi section, first, we review existing methods for time series prediction. Next, we give a discussion about GPs. Then, we investigate literatures about the semi-lazy approach. Finally, we briefly talk about kNN search under DTW distance.

2.1 Time series prediction methods

Existing time series prediction methods can be separated into two categories, i.e. statistical regression methods and machine learning methods. Statistical regression methods focus on finding parameterized functions, including linear models (e.g. ARIMA [15] and robust regression [59]) and nonlinear models (e.g. exponential smoothing [71, 38] and ((G)ARCH) models [31, 13]), which can predict the behavior of time series. Their major drawback is that they usually impose rigid assumptions on the time series data which may not be true in real-life applications.

Machine learning methods can be further grouped into two classes: the eager learning approach and the lazy learning approach. The eager learning approach usually has a pre-processing stage to train models for prediction such as Artificial Neural Networks (ANNs) [2, 68], Support Vector Machines (SVMs) [58, 46] and Gaussian Processes (GPs) [34, 50, 17, 72]. The common problem is that they usually require high computational cost. Although some approximate methods have been proposed, they are more influenced by the global distribution of the whole dataset, resulting in high potential of overfitting or underfitting [61]. Whereas, though some linear models with Stochastic Gradient Descent (SGD) can be efficiently trained, their prediction performance is not as good as the non-linear models in some applications.

The lazy learning approach typically is done by finding a set of k similar time series of the sensor and doing some simple computation (e.g. average) over these query results [44, 18, 11, 8]. These methods however cannot estimate the analytical predictive uncertainty directly. Bootstrap can partially remedy this drawback but requires high time cost. Furthermore, bootstrap cannot work well in high dimensional space when the time series segment is long. Besides, the accuracy of the semi-lazy learning approach still needs to be further improved.

2.2 GP for time series prediction

The Gaussian Process (GP) has received extensive attention for time series prediction [70, 34, 50, 16, 20, 35, 72, 23, 40]. GP has powerful and flexible model capability since it can be derived from both perspectives of neural networks and Bayesian nonparametric regression [49, 47]. The evaluation study in [55] further demonstrates that the GP can consistently outperform (or at least be comparable to) other modeling approaches like neural networks or local learning methods. In contrast to SVMs, GPs can innately predict analytical model uncertainty.

Unfortunately, the GP has a poor scalability (scaling as $O(n^3)$). Low-rank approximation is a popular method to improve its scalability [25, 56, 65, 43], but can only capture the global patterns of the data rather than the local behavior and patterns. There are also a few of papers about the localized models [33, 51, 48], which try to divide the data into different parts and build GP model for each part. Nevertheless, how to partition the data becomes a serious problem in high dimensional space. For this reason, the applications of this method are suitable for low dimensional spatial data analysis [33, 51, 48], but not for time series prediction.

We propose a new method to improve the scalability of GPs. The general idea is to build local GPs on the kNNs of prediction requests. This method can preserve the strength but not the weakness of the eager learning and the lazy learning approach. Using many local models to form an implicit global approximation, it can commit to a much richer hypothesis [73] but avoid an intractable training phase.

2.3 Semi-lazy prediction

To our best knowledge, SMiLer is the first work to exploit the semi-lazy learning approach to time series prediction. It is still desirable to discuss existing works about semi-lazy learning approach. There are indeed some works [74, 12] employing SVMs on kNN results for image classification. These methods are customized for image features (like tangle distance), and cannot be directly extended to support time series prediction. The authors in [67] propose a kNN based Kalman Filter GP regression. However, it still needs an offline processing to learn the hyperparameters, which makes it essentially still be an eager learning method.

Compared with the semi-lazy trajectory prediction[77, 76], SMiLer is a general framework for time series prediction, while the method in [77] is only applicable to object path prediction because its methodology is specially catered to the scenario where there are many moving objects in a dynamic environment. Furthermore, the method in [77] requires several parameters which have to be tuned manually; on the contrary SMiLer can minimize the user assistance by devising an adaptive auto-tuning mechanism. Additionally, except the semi-lazy GP idea, we also devote an extensive study to kNN search under the DTW distance leveraging the GPU and devise an adaptive mechanism for auto-tuning parameters, both of which are not touched by previous semi-lazy learning methods.

2.4 kNN search on Dynamic Time Warping

There have been many efforts to speed up kNN search under the DTW [41, 78, 30, 6, 54]. Various indexing methods in memory [6] and disk [37, 5, 36] are also studied in recent years. In [60], the authors compared the performance of GPUs and FPGAs with scanning method for DTW computation. We refer interested readers to [30]. As far as we know, our paper is the first study to design index on the GPU to accelerate kNN search. We also propose an enhanced lower bound of the DTW suitable for GPU computation.

3. OVERVIEW

In this section, we give a formal description of our semi-lazy method for time series prediction, and discuss the challenges of our method. Table 1 lists the basic notations used throughout this paper.

Table 1: Table of Basic Notations.

C^i	time series of sensor i	c^i_t	value of C^i at t
$x^i_{j,d}$	d-length segment of C^i	$X^i_{k,d}$	data set $\{x^i_{j,d}\}^k_{j=1}$
$y^i_{j,h}$	h-step ahead value of $x^i_{j,d}$	Y^i_h	a vector of $y^i_{j,h}$
EKV	Ensemble kNN Vector	k	number of kNNs
ELV	Ensemble Length Vector	d	length of segment

3.1 Preliminaries

A time series C^i is a collection of observations made sequentially in time from a sensor i (or more generally, an unknown system), i.e., $C^i = \{c^i_0, c^i_1, .., c^i_j, ...\}$, where c^i_j is the value of C^i at timestamp j. We assume the sample rate of one sensor is always fixed[1], therefore, a time series is only a sequence of data points. $|C^i|$ denotes the length of C^i. A set of contiguous observations of C^i between two points c^i_t and c^i_{t+d} is called a *segment* and is denoted by $C^i_{t,d}$. We also call a segment with length d as a d-length segment.

At time t_0, the h-step ahead prediction is to predict the value of the sensor at time t_0+h. Taking a d-length segment $x^i_{0,d} = C^i_{t_0-d+1,d} = \{c^i_{t_0-d+1}, .., c^i_{t_0}\}$ as model input and denoting the h-step ahead value of $x^i_{0,d}$ by $y^i_{0,h} = c^i_{t_0+h}$, the h-step ahead prediction model is a mapping $f(\cdot)$ between $x^i_{0,d}$ and $y^i_{0,h}$, i.e. $y^i_{0,h} = f(x^i_{0,d})$.

3.2 Semi-lazy time series prediction

In this section, we first present the general framework for semi-lazy time series prediction, and then introduce the adaptive auto-tuning mechanism.

3.2.1 *Abstract semi-lazy time series predictor*

Given a time series segment $x^i_{0,d} = \{c^i_{t_0-d+1}, .., c^i_{t_0}\}$ ending at time t_0, we can retrieve k nearest neighbor segments from time series C^i, i.e. $X^i_{k,d} = \{x^i_{j,d}\}^k_{j=1} = [x^i_{j,d}, ..., x^i_{k,d}]$ (where $x^i_{j,d}$ is a segment of C^i ending at time t_j, i.e. $x^i_{j,d} = \{c^i_{t_j-d+1}, .., c^i_{t_j}\}$). For each segment $x^i_{j,d}$, its h-step ahead value is $y^i_{j,h} = c^i_{t_j+h}$. We denote the h-step ahead values of every $x^i_{j,d}$ in $X^i_{k,d}$ by a vector $Y^i_h = [y^i_{1,h}, y^i_{2,h}, ..., y^i_{k,h}]^\top = [c^i_{t_1+h}, c^i_{t_2+h}, ..., c^i_{t_k+h}]^\top$. Now given $(X^i_{k,d}, Y^i_h)$, we formally define the semi-lazy time series predictor as below.

Definition 3.1 (Semi-Lazy Time Series Predictor). *Given a d-length time series segment $x^i_{0,d} = \{c^i_{t_0-d+1}, .., c^i_{t_0}\}$ of C^i ending at time t_0, the semi-lazy time series predictor is a model which can use the kNN data $(X^i_{k,d}, Y^i_h) = \{x^i_{j,d}, y^i_{j,h}\}^k_{j=1}$ and test input $x^i_{0,d}$ to obtain the posterior distribution of the h-step ahead observation $y^i_{0,h}$ (i.e. $c^i_{t_0+h}$):*

$$y^i_{0,h} = f(x^i_{0,d}, X^i_{k,d}, Y^i_h) \sim \mathcal{N}(u, \sigma^2) \tag{1}$$

where $f(\cdot)$ is an abstract predictor which can be instantiated with suitable probabilistic prediction model.

The semi-lazy time series predictor is built independently for each sensor, but multiple sensors can be processed in the same way. Hereafter, unless otherwise stated, we focus on the kNN search and prediction for one sensor.

The superscript "i" (e.g. $y^i_{0,h}$ and $x^i_{j,d}$) indicates that the variables are from sensor i. Hereafter, when we focus on one sensor, we omit the superscript i for convenience.

3.2.2 *Auto-tuning mechanism with ensemble method*

We design an auto-tuning mechanism to eliminate the parameters of the semi-lazy prediction model as well as to improve the prediction accuracy. In Definition 3.1, for each predictor, there are two parameters: (1) k: the number of nearest neighbors and (2) d: the length of time series segment. For different sensors with different intrinsic properties, the semi-lazy predictors may desire different k and d. With the changing of the data generating process, the values of k and d also have to be adjusted.

We introduce an ensemble method to the semi-lazy model, which forms a matrix of abstract predictors $f_{i,j}$ for a sensor with different k and d. The final predictor is the mixture of all the $f_{i,j}$ predictors. Let us define an ensemble matrix λ:

$$\lambda = \begin{bmatrix} (k_0, d_0) & ... & (k_0, d_{n-1}) \\ ... & (k_i, d_j) & ... \\ (k_{m-1}, d_0) & ... & (k_{m-1}, d_{n-1}) \end{bmatrix} \tag{2}$$

where k_i is the number of nearest neighbors and d_j is the length of time series segment for predictor $f_{i,j}$. We group different k_i in the "*Ensemble kNN Vector*" denoted by $EKV = [k_0, ..., k_{m-1}]$, and group different d_j in the "*Ensemble Length Vector*" denoted by $ELV = [d_0, ..., d_{n-1}]$. In the ensemble matrix λ, each element $\lambda_{i,j}$ also indicates the weight of $f_{i,j}$ contributed to the final predictor. Hence, the ensemble prediction model for one sensor is formally defined as:

$$f_{em} = \frac{1}{\mathcal{C}_\lambda} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \lambda_{i,j} f_{i,j} \tag{3}$$

where $\mathcal{C}_\lambda$ is the normalization constant by summing the weight of every element in λ, i.e. $\mathcal{C}_\lambda = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \lambda_{i,j}$. Later we further propose an intelligent self-adaptive method to adjust λ during continuous prediction (see Section 5.1).

3.3 Challenges of semi-lazy prediction

Now we can summarize the challenges for semi-lazy time series prediction. First, we need to quickly identify the kNNs for each semi-lazy predictor. To overcome the unacceptable latency for kNN search, we resort to the help of the GPU to accelerate the search process in Section 4.

Specifically, we formalize the "*(Continuous) Suffix kNN Search*" problem in Section 4. With the ensemble method, we need to invoke a set of kNN queries for one sensor per prediction (see Eqn. (2)). We can see that, if $d_i < d_j$, x_{0,d_i} is just a suffix of x_{0,d_j} since both of them end at time t_0. Moreover, we usually need to continuously predict the future value of sensors. During the continuous prediction, some computation can also be reused. Based on the suffix property and the continuous prediction, we further propose a two-level inverted-like index on the GPU in Section 4.

Second, a proper model selection for the abstract predictor $f(\cdot)$ and an auto-tuning mechanism are required. We propose to use GPs as predictors and give a complete solution for the semi-lazy time series prediction in Section 5.2. We also fully depict the adaptive auto-tuning mechanism for dynamic adjusting the ensemble matrix λ in Section 5.1.

3.4 Framework of SMiLer

SMiLer is designed to make the semi-lazy time series prediction model feasible. Fig. 3 shows an overview framework of SMiLer to satisfy the above objectives.

In the first step, called Search Step, with the input of the last few time steps (Rectangle A in Fig. 3) of target sensors,

[1] This is not a real limitation since the user can easily reinterpolate data if the sample rate is changed.

we invoke the Continuous Suffix kNN Search with multiple k and d. The queries of one sensor are parallel processed on the data of itself by the the GPU. (see Section 4)

Next, in the Prediction Step, the kNN results (Rectangle B in Fig. 3) are input into the semi-lazy GP models to predict future value (with mean and variance) of each sensor (Rectangle C in Fig. 3). The adaptive auto-tuning mechanism is utilized to improve the prediction performance as well as to minimize users' assistance. (see Section 5)

4. DTW KNN SEARCH WITH THE GPU

We use DTW distance to find kNN segments on the GPU. There have been several similarity measures for time series, such as Euclidean distance [32], DTW [10], LCSS [66], ERP [21], EDR [22] and SpADe [24]. Euclidean distance is simple but sensitive to noise (e.g. shifting and scaling) problem which usually appears in time series. Among these measures, DTW is a simple but effective one which is robust to noise (e.g. shifting and scaling). Other distance measures can also handle time series similarity search, but they usually need a sophisticated index structure, which cannot be easily implemented on the GPU. Besides, there are some evidences showing that DTW is the best measures for time series data mining problems [30, 60, 54].

4.1 Problem formulation

The SMiLer Index is designed to identify a set of kNNs for the ensemble prediction model (refer to Section 3.2.2). The kNN search with different k in $EKV = [k_0, k_2, ..., k_{m-1}]$ (see Eqn. (2)) can be solved by invoking the NN search with maximum value k_n, and then selecting subsets of the result according to DTW distance order.

However, it is not trivial for kNN search with different query lengths but sharing common suffix. Suppose at time t_0 the set of the queries is $\{x_{0,d_0}, ..., x_{0,d_{n-1}}\}$ where $x_{0,d_i} = \{c_{t_0-d_i+1}, ..., c_{t_0}\}$. We can see that, if $d_i < d_j$, x_{0,d_i} is a suffix of x_{0,d_j} since both of them end at time t_0. An example of x_{0,d_0} and x_{0,d_1} is shown in Fig. 5. It is desirable to reduce the computation cost based on the suffix property.

Another opportunity arises from the continuous kNN search. Since the query input is gradually changing over time, we should also consider this property to improve efficiency.

Let us define some simplified notations. For a target sensor, "*Master Query*" denotes the longest query segment $MQ = x_{d_{n-1}} = \{q_0, ..., q_{d_{n-1}}\} = \{c_{t_0-d_{n-1}+1}, ..., c_{t_0}\}$. Since every x_{d_i} is a suffix of $x_{0,d_{n-1}}$, we also denote each query segment as "*Item Query*" $IQ_i = x_{0,d_i} = \{q_{d_n-d_i}, ..., q_{d_{n-1}}\} = \{c_{t_0-d_i+1}, ..., c_{t_0}\}$. Fig. 5(b) shows an example of MQ and its item queries IQ_0 and IQ_1. The "*Continuous Suffix kNN Search*" problem can be formally defined as:

Definition 4.1 ((Continuous) Suffix kNN Search). *Given a master query MQ of a sensor with $ELV = [d_0, ..., d_{n-1}]$, we can generate a set of item queries $\{IQ_0, ..., IQ_{n-1}\}$ where $|IQ_i| = d_i$ and IQ_i is a suffix of IQ_j if $d_i < d_j$. The Suffix kNN Search is to find kNN segments C_{t,d_i} for each item query IQ_i under DTW distance on time series C of the sensor. From time t to time $t+1$, we may continuously do Suffix kNN search with appending the latest observed point into MQ and delete the oldest point from MQ.*

In the following sections, we will exploit the SMiLer Index for the Continuous Suffix kNN Search on the GPU. Readers may refer to Appendix B.1 for a brief review about DTW. Recall that in this paper we only consider the DTW under Sakoe-Chiba band constraint with warping width ρ. We also provide a short introduction of the GPU in Appendix B.2.

4.2 Enhanced lower bound for DTW

We propose an enhanced lower bound for DTW, denoted by LB_{en}, which is derived from the existing lower bound LB_keogh [41] (refer to Appendix B.1).

Depending on which envelope is used, for convenience, we simply denote $LB_{EQ}(Q, C) = LB\_keogh(\mathrm{E}(Q), C)$ and $LB_{EC}(Q, C) = LB\_keogh(E(C), Q)$. Examples of envelope, $LB_{EQ}(Q, C)$ and $LB_{EC}(Q, C)$ are shown in Fig. 5 (a)(b).

Based on LB_keogh, we propose an enhanced lower bound of DTW, denoted by LB_{en}, which are defined as:

$$LB_{en}(Q, C) = \max\{LB_{EQ}(Q, C), LB_{EC}(Q, C)\}$$

Theorem 4.1. *$LB_{en}(Q, C)$ is a lower bound of $DTW(Q, C)$.*

Proof. Since $LB_{EC}(Q, C) \le DTW(Q, C)$ and $LB_{EQ}(Q, C) \le DTW(Q, C)$, we have $LB_{en}(Q, C) \le DTW(Q, C)$. □

4.3 SMiLer Index: a two-level inverted-like index

In this section, we present the SMiLer Index on the GPU memory as well as how to use this index to support the Continuous Suffix kNN search. The novel point of the SMiLer Index is that we can reuse the intermediate results to accelerate the computation of the DTW lower bound.

As shown in Fig. 4, indeed the SMiLer Index is a two-level inverted-like index which contains a "*window level index*" and a "*group level index*". This inverted index-like structure can enjoy the parallel capability of the GPU by using one block to process one posting list.

4.3.1 Window level index of the SMiLer Index

Following the DualMatch framework [45, 36], we divide the time series C into disjoint windows DW and divide the master query MQ into sliding windows SW where $\omega = |DW| = |SW|$. Examples of DW and SW are illustrated in Fig. 5(a)(b). Note that we divide sliding windows in time-reserved order (from right to left in Fig. 5(b)).

In the *window level index*, a keyword (index term) is a sliding window (of the master query) whose posting list stores lower bounds between the sliding window and disjoint windows of time series C. The window level inverted index can be constructed efficiently by the GPU. We use one block to treat one sliding window to parallel compute lower bounds (LB_{EQ} and LB_{EC}) between all SWs and DWs. Illustration of the window level index is shown in Fig. 4 and Fig. 5(c).

Remark 1: reuse based on the continuous query.

During continuous prediction, we can reuse the computed result on the window level index to avoid building the SMiLer Index from scratch. Suppose that at time $t_0 - 1$ there is a master query MQ'. Then at time t_0, the new master query MQ is constructed by adding one point to the head of MQ' and removing the last point of MQ'. Consequently, we only need to add a new sliding window to MQ and remove the last sliding window of MQ'.

Fig. 6 illustrates how to update the window level index during the continuous prediction. For a new master query at time t_0, we first clear the posting list of the last sliding window SW_n, and then place the posting list of the new sliding window SW' in the memory space of SW_n (see Fig.

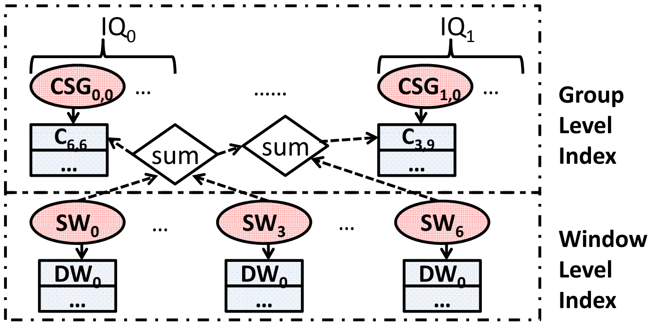
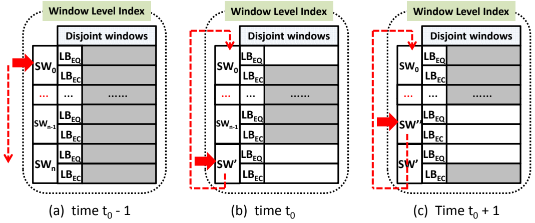


Figure 4: SMiLer Index: a two-level inverted-like index.

6(b)). The starting cursor (the red vertical arrow) of the window level index moves from SW_0 to SW'. Then at time t_0+1 (see Fig. 6 (c)), the new sliding window SW'' replaces the memory space of SW_{n-1} and the starting cursor moves to SW''. In addition, after adding a new point, the envelopes of previous ρ (i.e. warping width) sliding windows are changed. As a result, we need to re-calculate LB_{EQ} in the posting lists for these affected sliding windows. For example, if $\rho = 1$, LB_{EQ} of SW_0 is re-calculated in Fig. 6 (b), and LB_{EQ} of SW' is re-calculated in Fig. 6(c).



(a) time t_0 - 1 (b) time t_0 (c) Time t_0 + 1

Figure 6: Reuse window level index for continuous prediction.

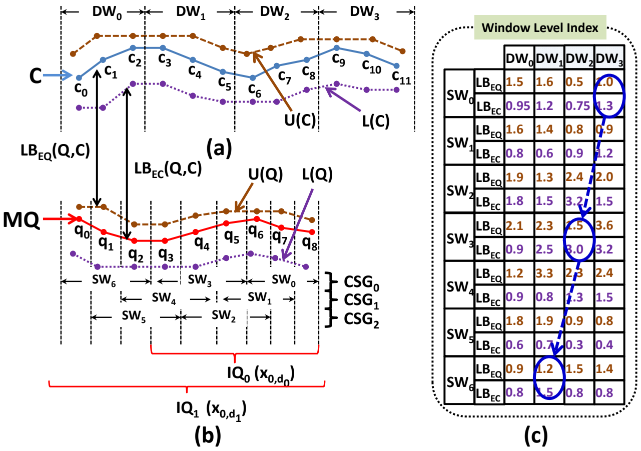
4.3.2 *Group level index of the SMiLer Index*

In the *group level index*, a keyword is a *Catenated Sliding Window Group (CSG)* (will be explained later) of item query IQ_i. The posting list of the CSG of IQ_i stores the lower bounds between IQ_i and a set of segments of C. The general idea for constructing group level index is that partial sum of posting lists in window level index are just the lower bounds between item queries and candidate segments. We first define the keyword in the group level index, i.e. the Catenated Sliding Window Group (CSG) which is inspired by the concept of "equivalence class" in [36].

Definition 4.2 (Catenated Sliding Window Group). *A Catenated Sliding Window Group (CSG) of query Q contains maximum number of sliding windows without overlap.*

$CSG_{i,b}$ denotes a CSG of item query IQ_i, where subscript i is identifier of IQ_i and subscript b is identifier of the first sliding window (from right to left) SW_b in the $CSG_{i,b}$. For a master query MQ without subscript, we denote it as CSG_b. Note that $CSG_{i,b}$ is always the prefix of CSG_b.

Example 4.1. *In Fig. 5(b), master query MQ has three CSGs which are $CSG_0 = \{SW_0, SW_3, SW_6\}$, $CSG_1 = \{SW_1, SW_4\}$ and $CSG_2 = \{SW_2, SW_5\}$. The CSGs of IQ_0 are $CSG_{0,0} = \{SW_0, SW_3\}$, $CSG_{0,1} = \{SW_1\}$ and $CSG_{0,2} = \{SW_2\}$. $CSG_{0,b}$ of IQ_0 is always a prefix of CSG_b of MQ.*



		DW$_0$	DW$_1$	DW$_2$	DW$_3$
SW$_0$	LB$_{EQ}$	1.5	1.6	0.5	1.0
	LB$_{EC}$	0.95	1.2	0.75	1.3
SW$_1$	LB$_{EQ}$	1.6	1.4	0.8	0.9
	LB$_{EC}$	0.8	0.6	0.9	1.2
SW$_2$	LB$_{EQ}$	1.9	1.3	2.4	2.0
	LB$_{EC}$	1.8	1.5	3.2	1.5
SW$_3$	LB$_{EQ}$	2.1	2.3	1.5	3.6
	LB$_{EC}$	0.9	2.5	3.0	3.2
SW$_4$	LB$_{EQ}$	1.2	3.3	2.3	2.4
	LB$_{EC}$	0.9	0.8	1.3	1.5
SW$_5$	LB$_{EQ}$	1.8	1.9	0.9	0.8
	LB$_{EC}$	0.6	0.7	0.3	0.4
SW$_6$	LB$_{EQ}$	0.9	1.2	1.5	1.4
	LB$_{EC}$	0.8	1.5	0.8	0.8

Figure 5: An illustration for SMiLer Index.

Theorems of the CSG for index construction.

We introduce two important theorems for constructing the group level index. Before that, we first clarify the meaning of the alignment between a CSG and disjoint windows.

Lemma 4.1. *Suppose that item query IQ_i has a $CSG_{i,b}$ whose sliding windows are aligned with a set of contiguous disjoint windows (from right to left) i.e. $\{DW_r, DW_{r-1}, ...\}$. Then this alignment indicates an alignment between IQ_i and a segment C_{t,d_i} where the value of subscript t of C_{t,d_i} is:*

$$t = (r - |CSG_{i,b}| + 1) * \omega - (d_i - b)\%\omega \tag{4}$$

where $|CSG_{i,b}|$ is the number of windows in $CSG_{i,b}$.

Proof. See Appendix C.1. □

Theorem 4.2. *For each pair of IQ_i and C_{t,d_i}, there is one and only one alignment between $CSG_{i,b}$ and corresponding disjoint windows.*

Proof. See Appendix C.2. □

Then we can deduce the window enhanced lower bound LB_w, between IQ_i and C_{t,d_i} from the window level index.

Theorem 4.3. *Given a $CSG_{i,b} = \{SW_b, SW_{b+w}, ...\}$ of IQ_i and a disjoint window DW_r, we can define the window enhanced lower bound LB_w between IQ_i and C_{t,d_i} as:*

$$LB_w(IQ_i, C_{t,d_i}) = \max \begin{cases} \sum_{j=0}^{m-1} LB_{EQ}(SW_{b+j*\omega}, DW_{r-j}) \\ \sum_{j=0}^{m-1} LB_{EC}(SW_{b+j*\omega}, DW_{r-j}) \end{cases} \tag{5}$$

where $m = |CSG_{i,b}|$ and $t = (r - m + 1) * \omega - (d_i - b)\%\omega$. The following inequality always holds:

$$LB_w(IQ_i, C_{t,d_i}) \leq DTW(IQ_i, C_{t,d_i})$$

Proof. See Appendix C.3. □

Group level index construction.

The construction of the group level index is based on the property of the Suffix kNN search and Theorem 4.3. Since IQ_i is always a suffix of IQ_j if $d_i < d_j$ for the Suffix kNN search, from Eqn. (5) we can see that the intermediate results of $LB_w(IQ_j, C_{t,d_i})$ just contain the lower bound between IQ_i and a candidate segment. We use this intuition to design the algorithm for building group level index.

For each item query IQ, we spin off a set of CSGs as the keywords to construct the group level index, where the posting lists of CSGs store the enhanced lower bounds between IQ and a set of time segments. Specifically, Theorem 4.2

tells us that we can get the lower bounds between IQ and every candidate time series segments in this way.

Eqn. (5) indicates that, by shift summing the posting lists of sliding windows within the same CSG of MQ, we can sequentially get all the lower bounds of IQ_i $(0 \le i \le n-1)$ of MQ (see Example 4.2). In other words, based on the suffix property, by sequentially summing the posting lists of sliding windows within the same CSG of a master query, we can obtain the posting lists of the CSG for each item query.

This computation can also be efficiently parallel processed by the GPU. The idea is to use one block of the GPU to process one CSG and use each thread of the block to handle several disjoint windows (elements in posting lists). In Algorithm 1 (Appendix D), we show the pseudo code to compute the posting lists of CSGs on the GPU.

EXAMPLE 4.2. *We use a block of the GPU to scan posting lists sequentially in order of $SW_0 \to SW_3 \to SW_6$ (see Fig. 4). In the block, there is a thread (blue dashed arrow) visiting the elements of the posting lists in order of: $(SW_0, DW_3) \to (SW_3, DW_2) \to (SW_6, DW_1)$ (see Fig. 5(c)). To sum the first two elements, we get lower bound between IQ_0 and $C_{6,6}$, $LB_{EC}(IQ_0, C_{6,6}) = LB_{EC}(SW_0, DW_3) + LB_{EC}(SW_3, DW_2)$ (refer to Eqn. 5). With summing the third element, we have $LB_{EC}(IQ_1, C_{3,9}) = LB_{EC}(IQ_0, C_{6,6}) + LB_{EC}(SW_6, DW_1)$, which is an element of posting list of $CSG_{1,0}$ (see Fig. 4).*

Remark 2: reuse based on the Suffix kNN Search.

The reuse of the computed result on the group level index can be seen from two points. First, for a set of item queries sharing suffix, by sequentially summing the posting lists of sliding windows in suffix order, we can compute the lower bounds between all item queries and time series segments in one-pass scan (see Example 4.2), which can avoid scanning the GPU memory multiple times. Second, the elements in the posting lists of window level index are computed once but are reused multiple times for computing the lower bound of every item query.

4.3.3 kNNs: filtering, verification and selection

After building the SMiLer Index, we follow the filtering-verification framework to retrieve the kNN results. By scanning the posting lists of CSGs in the group level index, we can get the DTW lower bound between item query IQ_i and all candidates segments. We filter the segments whose lower bounds are larger than a threshold τ_i (*Filtering*), and then compute the real DTW distance between IQ_i and unfiltered segments C_{t,d_i} (*Verification*). Finally, we select the kNNs of each IQ_i from all unfiltered candidates (*Selection*).

Filtering. The method is to discard candidates whose lower bounds are larger than threshold τ_i during scanning the post lists in group level index. There are two methods to determine threshold τ_i for IQ_i. The first one is to select the segment with the k-th smallest lower bound, and then set τ_i as the DTW distance between the segment and IQ_i. The second method is to reuse the kNN results during continuous prediction. Suppose that at time $t_0 - 1$ there is a query item IQ'_i. Since the difference between IQ_i and IQ'_i should be slight, we can take the distance between the k-th NN segment of IQ'_i and IQ_i as threshold τ_i. In SMiLer, we use the first method to determine the τ_i in initial queries, and then use the second one for the following continuous queries.

Verification. In Algorithm 2 (Appendix E), we show the pseudo code to compute the DTW distance (with Sakoe-Chiba band) between item queries and un-filtered candidates. The novelty of Algorithm 2 lies in the use of a compressed warping matrix. The shared memory, which is much faster than the global memory, is quite small (up to 64KB). To store the warping matrix in the shared memory, we design a compressed warping matrix with size of $2 \times (2*\rho+2)$ where ρ is the warping width. The essential idea is to temporarily store the matrix elements along the warp path, while the modulus operation (%) is employed to reuse the memory space. Please refer to Appendix E for more details.

Selection. We use a GPU k-selection algorithm to select kNNs with the smallest DTW distance from all unfiltered candidates. The main technique is distributive partitioning for k-selection on the GPU [3]. We adopt the existing work for GPU k selection [3] but with two incremental improvements: (1) we use one block to handle one k-selection for one query to support multiple k-selections; (2) we return all k smallest segments instead of only the k-th one.

4.4 Utility of the GPU

We summarize the utility of the GPU in the SMiLer Index from several perspectives. First of all, the SMiLer Index is essentially a two-level inverted-like index which can be parallel processed in as fine-grained manner as possible to fully utilize the GPU parallel computation capability. For example, we use one block to treat one sliding window to construct window level index; and we use one block to treat one CSG to construct group level index.

Second, based on properties of the *Continuous Suffix kNN Search*, the SMiLer Index can reuse intermediate results to improve efficiency (see Section 4.3.1, Remark 1: reuse based on the continuous query and Remark 2:reuse based on the Suffix kNN Search).

Third, the proposed enhanced lower bounds (LB_{en}) are innately applicable to GPU computation, which is not adopted by existing CPU based method like [45, 41, 36, 54]. Owing to the powerful parallel processing ability, we can obtain a tighter lower bound by computing both LB_{EC} and LB_{EQ} without increasing the response time.

Fourth, we try to ensure that processing in each thread block is as homogenous as possible. We use a two-phase scheme to filter and verify candidates instead of having them in one phase. The reason is that, due to the property of the SIMD architecture, the GPU hardware serializes different execution paths. If we mixed the filtering and verification, threads doing different processing need to wait for each other before continuing their processing which sacrifices efficiency.

Last, the SMiLer Index can easily scale up with multiple sensors, where we only need to create multiple SMiLer Indexes and invoke more blocks.

5. TIME SERIES PREDICTION VIA SEMI-LAZY LEARNING

Following by Section 3.2, we continue the discussion about our semi-lazy model. We first present the adaptive auto-tuning mechanism. Next, we introduce the instantiation of the *abstract predictor* with the Gaussian Process.

5.1 Adaptive auto-tuning mechanism with continuous prediction

The adaptive auto-tuning mechanism will dynamically adjust the ensemble matrix λ during continuous prediction

(Section 5.1.1). We also devise a sleep and recovery strategy (Section 5.1.2) to reduce the computational cost.

5.1.1 Auto-tuning with self-adaptive prediction

During the continuous prediction, we can self-adaptively learn to adjust the weight of each predictor in the ensemble matrix. The trick is that, after acquiring the true value of the sensor, we can evaluate each predictor by comparing the true value with the predicted one. Then we can increase the weight of predictors making good prediction.

Taking an abstract predictor $f_{i,j}$ as an example, we denote the true value of the sensor at time t as $y(t)$, and denote the predicted mean and variance as $u_{i,j}(t)$ and $\sigma^2_{i,j}(t)$. The likelihood function of $f_{i,j}$ after observing $y(t)$ is:

$$l_{i,j}(t) = l(y(t), u_{i,j}(t), \sigma^2_{i,j}(t)) \tag{6}$$

$$= \frac{1}{\sqrt{2\pi\sigma^2_{i,j}(t)}} exp(-\frac{(y(t) - u_{i,j}(t))^2}{2\sigma^2_{i,j}(t)}) \tag{7}$$

It is clear that the larger the likelihood $l_{i,j}(t)$ is, the better the predictor is. Then the weight of $f_{i,j}$ in the ensemble matrix at time t is adjusted as follows:

$$\bar{\lambda}_{i,j}(t) = \lambda_{i,j}(t-1) + \frac{l_{i,j}(t)}{\sum_i \sum_j l_{i,j}(t)} \tag{8}$$

After Eqn. (8), we need to further re-normalized $\bar{\lambda}_{i,j}(t)$ to get the final weight of the predictor $f_{i,j}$, i.e.:

$$\lambda_{i,j}(t) = \frac{\bar{\lambda}_{i,j}(t)}{\sum_i \sum_j \bar{\lambda}_{i,j}(t)} \tag{9}$$

In fact, combining Eqn. (8) and Eqn. (9), $\lambda_{i,j}(t)$ is an effectively exponential smoothing average of the posterior probability of the predictor $f_{i,j}$ over time.

5.1.2 Sleep and recovery

We further devise a strategy to control the sleep and recovery of every predictor. If $\lambda_{i,j}(t)$ is smaller than a threshold, we can temporarily make $f_{i,j}$ sleep to reduce the computational cost. The predictor would be recovered later.

The strategy is briefly presented here. In SMiLer, each predictor $f_{i,j}$ has a sleep counter $\varsigma_{i,j}$ specified how many steps it would sleep. If the weight $\lambda_{i,j}$ is smaller than threshold $\eta = \frac{1}{2*n*m}$ ($n*m$ is the number of elements of the ensemble matrix), we make predictor $f_{i,j}$ sleep, who will recover when the number of subsequent prediction steps exceeds $\varsigma_{i,j}$. If there are κ predictors recovered, the new weight of every recovered predictor is $\eta/(1-\kappa*\eta)$. After normalization, the weight of recovered predictors are equal to η.

Aiming to make the "weaker" predictor sleep longer, the sleep counter $\varsigma_{i,j}$ is also self-adaptive during the continuous prediction. $\varsigma_{i,j}$ is first initialized as 1, which means the predictor would only sleep for one step. If after recovery the predictor $f_{i,j}$ goes to sleep immediately in the next step, we will double the value of $\varsigma_{i,j}$. Otherwise, if the predictor successfully avoids the sleep trap, we would continuously halve the value of $\varsigma_{i,j}$ at very prediction step until $\varsigma_{i,j} = 1$.

5.2 Instantiation of the abstract predictor

5.2.1 A simple aggregation predictor

One simple predictor is a function to aggregate all the h-step ahead values of the kNN data. We define an Aggregation Regression (AR) function with pseudo-mean $\tilde{u}_0$ and pseudo-variance $\tilde{\sigma}^2_0$:

$$\hat{y}(t_0 + h) = f(x_{0,d}, X_{k,d}, Y_h) \tag{10}$$

$$= AR(x_{0,d}, X_{k,d}, Y_h) \sim \mathcal{N}(\tilde{u}_0, \tilde{\sigma}^2_0) \tag{11}$$

$$\tilde{u}_0 = \frac{\sum_{a=1}^{k} y_{a,h}}{k} \tag{12}$$

$$\tilde{\sigma}^2_0 = \frac{\sum_{a=1}^{k} (y_{a,h} - \tilde{u}_0)^2}{k} \tag{13}$$

AR predictor is simple and can be effectively computed, but its drawback is that the true value of $y(t_0 + h)$ may not follow the normal distribution defined by $\tilde{u}_0$ and $\tilde{\sigma}^2_0$.

5.2.2 Gaussian Process predictor

In this section, we introduce the Gaussian Process (GP) predictor, which has better prediction accuracy and good ability to estimate the predictive uncertainty. In Appendix B.3, we briefly recall some fundamentals of the GP.

For the GP predictor, given an input test segment $x_{0,d}$, the predictive distribution of $\hat{y}_{0,h}$ is obtained through conditioning on the kNN data $(X_{k,d}, Y_h)$ (recall Section 3.2.1). The predictive distribution is also a Gaussian distribution with mean u_0 and variance σ^2_0 as follows:

$$\hat{y}(t_0 + h) = f(x_{0,d}, X_{k,d}, Y_h) \tag{14}$$

$$= GP(x_{0,d}, X_{k,d}, Y_h) \sim \mathcal{N}(u_0, \sigma^2_0) \tag{15}$$

$$u_0 = c_0^\top C^{-1} Y_h \tag{16}$$

$$\sigma^2_0 = c(x_{0,d}, x_{0,d}) - c_0^\top C^{-1} c_0 \tag{17}$$

where C, c and c_0 are specified by the covariance function:

$$c(x_a, x_b) = \theta_0^2 exp\left(-\frac{1}{2}\frac{\|x_a - x_b\|^2}{\theta_1^2}\right) + \delta_{ab}\theta_2^2 \tag{18}$$

More details about Eqn. (16), Eqn. (17) and covariance function can be found in Appendix B.3. However, before making prediction, an important point is to determine the hyperparameters $\Theta = \{\theta_0, \theta_1, \theta_2\}$.

Online training for model optimization.

We use an online training method to determine the hyperparameters $\Theta = \{\theta_0, \theta_1, \theta_2\}$. For the eager learning approach, a heavy training process is employed to learn the optimal hyperparameters of GP in a pre-processing stage. In contrast, with the semi-lazy learning approach, we can afford the time to invoke an online training process to determine the hyperparameters because there are only a small number of training points (i.e. the kNN data $(X_{k,d}, Y_h)$). The advantage of this method is that the hyperparameters are specially trained for the test input $x_{0,d}$ (and its neighbors). In this way, we can avoid the underfitting or overfitting problems of the eager learning approach.

Now we explain how to train the GP predictor (to determine the hyperparameters) on the kNN dataset $(X_{k,d}, Y_h)$. For the GP, the predictive log probability when leaving out a training item $(x_{a,d}, y_a)$ is [56]:

$$logp(y_a | X_{k,d}, Y_{-a,h}, \Theta) = -\frac{1}{2} log\sigma^2_a - \frac{(y_a - u_a)^2}{2\sigma^2_a} - \frac{1}{2} log2\pi \tag{19}$$

where notation $Y_{-a,h}$ means all h-step ahead values in Y_h except y_a. The u_a and σ^2_a are computed according to Eqn. (16) and Eqn. (17) respectively. Thus, the leave-one-out

(LOO) log likelihood function on the whole kNN data is:

$$L(X_{k,d}, Y_h, \Theta) = \sum_{a=1}^{k} logp(y_a | X_{k,d}, Y_{-a,h}, \Theta) \tag{20}$$

The objective is to maximize the LOO log likelihood function (i.e. Eqn. (20)). To achieve this goal, we can compute its partial derivatives w.r.t. the hyperparameters and use the Conjugate Gradient (CG) optimization.

Since the expressions in Eqn. (16) and Eqn. (17) are almost identical for different points (only one column and one row removed in turn), the computation cost to optimize Eqn. (20) can be significantly reduced by the inversion of the partitioned matrix. An efficient approach to such training process can be found in [64].

Online training in continuous prediction.

In the continuous prediction, we can use an online optimization method to train the GP model. The intuition for the online training is that the hidden model generating the time series should change gradually. Consequently, the fixed steps pursuit training method is enough to find near-optimal value of the hyperparameters. Based on this point, in SMiLer, we only use the fixed five-step gradient descent to update the hyperparameters for the subsequential predictions. For a time series predictor GP, let $\theta_r(t)$ denote a hyperparameter at time t. We can use $\theta_r(t)$ as the inial seed value (instead of random seed values) to deduce the the parameter $\theta_r(t+1)$ by one step gradient descent. After the initial deducing, we further employ the CG optimization method (with fixed steps of descent) to obtain a near-optimal value of θ_r. By this method, the energy paid for the training process in previous steps is partially preserved.

6. EXPERIMENTS

6.1 Settings

Table 2: Default parameter for experiment.

Parameter	Description	value
ρ	warping width	8
ω	window length	16
ELV	Ensemble Length Vector	$\{32, 64, 96\}$
EKV	Ensemble kNN Vector	$\{8, 16, 32\}$

6.1.1 Environment and parameters

Experiments were conducted on a CPU-GPU platform. The GPU is a GeForce GTX TITAN with 6 GB memory. We implemented the GPU code using CUDA 6. The other program was implemented in C++ running on CentOS 6 with an Intel Core i7-3820 CPU server and 64 GB RAM.

Table 2 lists the default parameters in the experiment. For the DTW computation (SMiLer and its competitors), we set the warping width as $\rho = 8$. The window size ω is set as 16. Unless otherwise stated, in SMiLer we used a 3×3 ensemble matrix for prediction (see Section 3.2.2) where k and d are indicated in EKV and ELV respectively.

6.1.2 Datasets

We used three real-life time series datasets to evaluate our system. Two datasets are publicly available which can ensure the repeatability, and the remaining one is provided by our collaborators. We used z-normalization to normalize the time series of each sensor.

Table 3: Effect of the enhanced lower bound LB_{en}. The "time" (in seconds) is the total time for verifying un-filtered candidates of all sensors, and the "number" indicates the number of unfiltered candidates per query per sensor.

data	ROAD		MALL		NET	
	time	number	time	number	time	number
LB_{EQ}	2.30	12558	1.12	6632	0.11	753
LB_{EC}	1.55	9206	0.94	5707	0.11	725
LB_{en}	1.11	6739	0.63	3677	0.079	516

[**ROAD**] This dataset [27] consists of times series of 963 road traffic sensors of San Francisco bay area freeways in PEMS website [53]. Each sensor measured the occupancy rate of a road in the city for 15 months with 10-minute sample interval. In total, there are 61.0 million data points. The data set is available in the UCI Machine Learning Repository [7] and can be download freely [28].

[**MALL**] This dataset consists of time series of available car park lots in main shopping malls in Singapore. There are a total of 26 car parks with a record in every 10 minutes for 12 months. We duplicated every time series 40 times. In total, there are 1040 (26×40) sensor time series and 53.9 million data points (after duplication). The data is crawled from dataMall website [63] from Sept. 2013 to Sept. 2014.

[**NET**] This dataset is a time series of internet traffic data of a network backbone. It was collected for 3 months with 5-minute sample interval. We duplicated this time series 1024 times. In total, there are 1024 (1×1024) sensor time series and 20.4 million data points (after duplication). The data set can be download freely from DataMarket [29].

6.2 Search Step: Suffix kNN search on DTW

6.2.1 Competitors and experiment settings

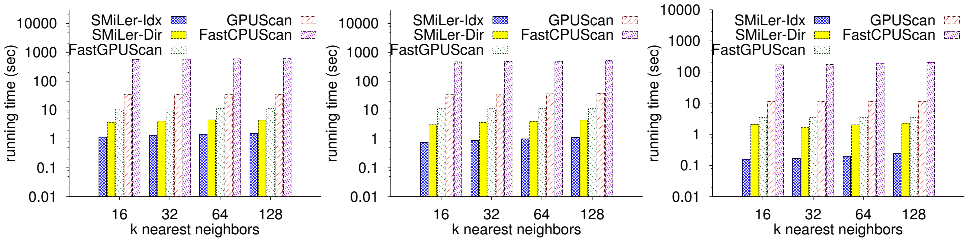
We compared our DTW kNN search method of SMiLer, denoted as "*SMiLer-Idx*", with three kNN search competitors, namely FastGPUScan, GPUScan [60] and FastCPUScan. FastGPUScan first computes the DTW between the queries and all segments with the Sakeo-Chiba constraint, and then uses the GPU fast selection method to obtain the kNNs. GPUScan [60] is similar to FastGPUScan but without the Sakeo-Chiba constraint. FastCPUScan computes the DTW under the Sakeo-Chiba constraint with suitable pruning criteria studied in [41, 54]. For all datasets, we randomly selected a master query for each sensor to do the Suffix kNN search with 100 steps continuous query. The running time is the total time of all sensors per query step.

6.2.2 Evaluation with running time

Fig. 7 (y-axis is log-scaled) shows that SMiLer-Idx is one order of magnitude faster for the Suffix kNN Search compared with the best competitor FastGPUScan. SMiLer-Idx needs about 1 second to finish the search on all sensors, while FastGPUScan needs 10 seconds and FastCPUScan needs about 500 seconds. The time cost of SMiLer-Idx is quite stable with different numbers of nearest neighbors k.

Table 3 exhibits the utility of the lower bound LB_{en}. Both the number of unfiltered candidates and the time cost for verifying the candidates of LB_{en} are only about a half of the ones of LB_{EQ} and two-thirds of the ones of LB_{EC}.

Fig. 8 further demonstrates the effectiveness of the two-level inverted-like index of SMiLer. In Fig. 8, we use "*SMiLer-Dir*" to denote the method to compute the LB_{en} directly for



(a) ROAD (b) MALL (c) NET

Figure 7: Time cost (log-scaled) of the Suffix kNN Search on all sensors with varying k.

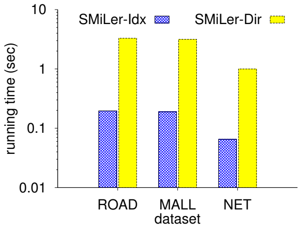


Figure 8: Time cost (log-scaled) for computing LB_{en} for all sensors.

each item query without building index at the window level (i.e. simply scanning the data to compute the lower bound). We can see that the SMiLer-Idx can significantly reduce the running time for computing LB_{en} by much more than one order of magnitude over SMiLer-Dir.

6.3 Prediction Step: effectiveness of SMiLer

6.3.1 *Competitors and experiment settings*

SMiLer-AR and **SMiLer-GP** denote our semi-lazy prediction models with different instantiated predictors. We category competitors into two groups: offline learning models and online learning models. The offline learning models, which are examples of eager learning, have a training phase; while the online learning models build the model on the fly with the incoming data. The offline learning models are:

- **PSGP**, i.e. Projected Sparse Gaussian Process, is an approximation to the Gaussian Process [56] by projecting all information onto a set of "active points" [25].We set the number of "active points" to 32, whose effect is also investigated in Section 6.4.2. We used an open-source project of PSGP [9].
- **VLGP**, i.e. Variational Learning Gaussian Process, is another state-of-the-art scalable Gaussian Process approximation model [65]. We set the number of inducing inputs as 32 whose effect is similar to the active points of PSGP. We use its implementation in [62].
- **NysSVR**, is a low-rank approximation of RBF kernel Support Vector Regression (SVR) model with Nyström method [69]. We set the reduced rank as 128.
- **SgdSVR**, is a linear SVR model with the Stochastic Gradient Descent (SGD) optimization method [75].
- **SgdRR**, is a robust regression [59] model with the SGD optimization method.

For NysSVR, SgdSVR and SgdRR, we used their implementation in Scikit-learn [52]. The grid search method by 10-fold cross validation is used to find their optimal parameters. We adopted the method in libSVM [19] to estimate the prediction confidence of SVR. The online learning models are:

- **LazyKNN**, is a lazy learning prediction method for time series prediction [4], where the predicted value is an average of the kNNs weighted by the inverse of DTW distance. We used the variance of the kNNs as the predicted variance.
- HoltWinters, which has two sub-methods: "**FullHW**" and "**SegHW**", is a popular statistical regression model for time series with periodical patterns [71, 38]. We used its implementation in "forecast" package of R [39]. We set the period as one day, and parameters were determined by minimizing the squared error. For "FullHW", we used all the available data to construct the model for each prediction, and for "SegHW", we used the last 10 days data to construct the model.
- **OnlineSVR** and **OnlineRR**, are similar with SgdSVR and SgdRR, but are trained in a one-pass online fashion [14]. We used the first one-third of the data to determine their parameters with grid search by 10-fold cross validation, and then used the following data to sequentially update the model with SGD.

Though some competitors are in R or Python, their core algorithms (except VLGP) are still implemented by C++ or Cython. Therefore, their running time is still comparable.

We evaluate the prediction performance by two measures: mean absolute error (MAE), which is an average of the absolute errors between the predicted value and the true value; and mean negative log predictive density (MNLPD), which is an average of negative log of the density of the true value under the predicted normal distribution. The MAE can evaluate the accuracy of the predicted result; while the MNLPD can assess the quality of the predictive uncertainty. For both measures, the smaller the value is, the better the method is.

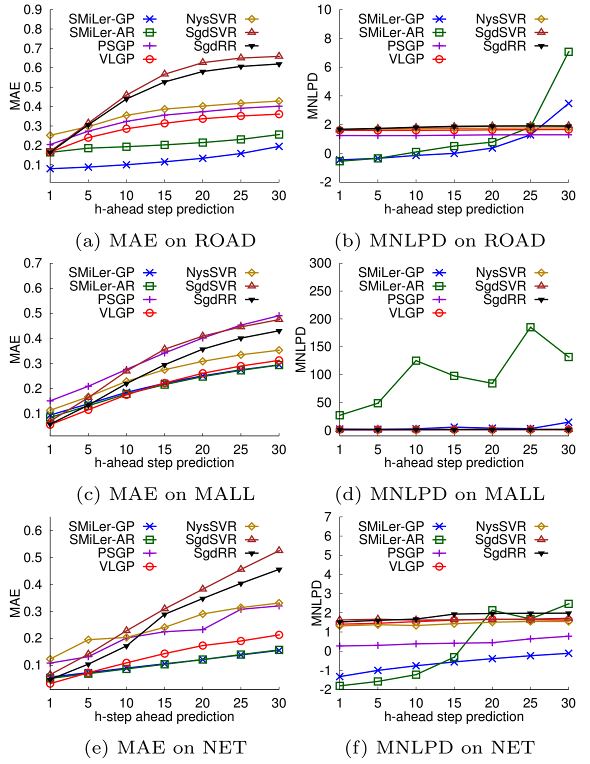
For the ROAD dataset, we cut off a segment (i.e. leave-out testing) with 1000 points at the end of every time series. For MALL and NET datasets, since there is duplication, we randomly cut off a segment (leave-out testing) with 1000 points from every time series. Then we made 200-step continuous prediction along the segment. For PSGP, VLGP and NysSVR, we only tested on 50 randomly selected time series, since their training time costs for all sensors are too high to be acceptable (see Section 6.4.2 for more explanation).

Table 4: Running time comparison. The training time cost ("trn", in hours) is the total time for training the models for all sensors for one prediction step in the experiment. The prediction time ("prd", in milliseconds) is the average prediction time per sensor per query.

data	ROAD		MALL		NET	
	trn(h)	prd(ms)	trn(h)	prd(ms)	trn(h)	prd(ms)
SMiLer-GP	-	27.59	-	24.46	-	25.32
SMiLer-AR	-	1.48	-	0.95	-	0.25
FullHW	-	724.87	-	770.73	-	188.05
SegHW	-	58.52	-	64.23	-	83.32
LazyKNN	-	0.63	-	0.46	-	0.11
PSGP	1.8e3	0.037	1.6e3	0.031	126.3	0.024
VLGP	198.4	0.0068	274.3	0.011	57.5	0.0068
NysSVR	95.3	0.0085	86.2	0.0087	28.9	0.0088
SgdSVR	2.2	2.1e-4	2.3	2.2e-4	0.6	2.1e-4
SgdRR	13.5	2.7e-4	12.7	2.4e-4	4.0	2.5e-4
OnlineSVR	0.6	2.4e-4	0.5	2.2e-4	0.19	2.2e-4
OnlineRR	2.4	2.7e-4	2.3	2.5e-4	0.72	2.4e-4

6.3.2 *Evaluation with MAE and MNLPD*

Fig. 9 and Fig. 10 show the prediction performance of SMiLer compared to offline learning and online learning mod-

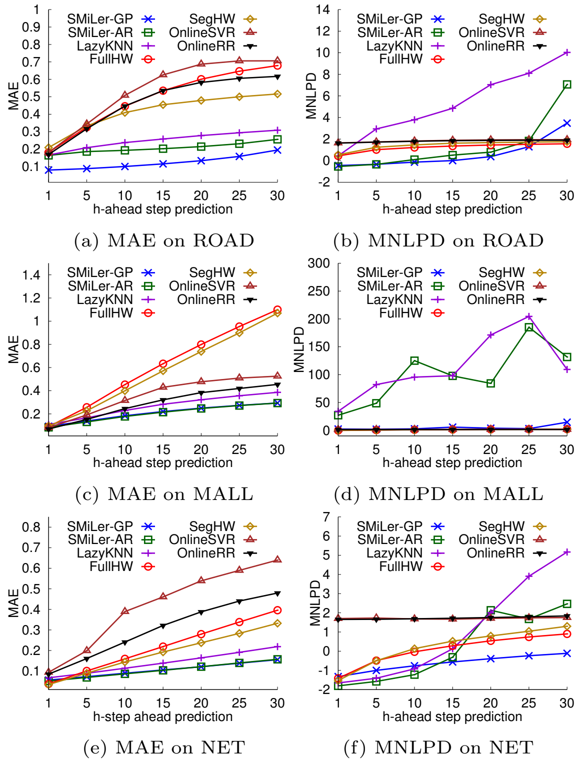


(a) MAE on ROAD (b) MNLPD on ROAD

(c) MAE on MALL (d) MNLPD on MALL

(e) MAE on NET (f) MNLPD on NET

Figure 9: MAE and MNLPD of offline learning models with varying h-step ahead prediction.



(a) MAE on ROAD (b) MNLPD on ROAD

(c) MAE on MALL (d) MNLPD on MALL

(e) MAE on NET (f) MNLPD on NET

Figure 10: MAE and MNLPD of online learning models with varying h-step ahead prediction.

els with varying look-ahead step h. From Fig. 9 (a)(c)(e) and Fig. 10 (a)(c)(e), we can see that SMiLer-GP always has a smaller MAE than all competitors. The reason is that, as a query-dependent GP model, SMiLer-GP caters to specifically making prediction for the submitted query without the need to cater for all parts of the data. Whereas the offline learning methods may result in over generalized global models, and the online learning methods may not have powerful enough modeling capability. An interesting point is that the MAE of SMiLer-AR is about 2 times larger than SMiLer-GP on the ROAD dataset (see Fig. 10(a)), while their MAE is almost identical on the MALL and NET datasets (see Fig. 10(c)(e)). It is because the ROAD dataset contains more dynamic traffic information while the MALL and NET datasets have some seasonal patterns. Therefore, SMiLer-GP, with strong GP predictors, has an innate advantage over SMiLer-AR on such complex data. Moreover, SMiLer-AP is worse than SMiLer-GP under MNLPD measure.

Fig. 9(b)(d)(f) and Fig. 10(b)(d)(f) demonstrate that SMiLer-GP is better than (or at least is comparable to) all competitors under MNLPD measure. In particular, SMiLer-GP is significantly better than SMiLer-AR and LazyKNN under the MNLPD measure. SMiLer-GP also has smaller MNLPD than the other competitors except after step 30.

6.3.3 *Effect of the adaptive auto-tuning mechanism*

Fig. 11 reveals the effect of the adaptive auto-tuning mechanism of SMiLer. SMiLerNE-GP and SMiLerNE-AR denote the experimental result of SMiLer without the ensemble method (i.e. only one predictor instead of a matrix of predictors). For SMiLerNE, we fixed the query segment length as $d = 64$ and the number of nearest neighbors as $k = 32$. SMiLerNW-GP and SMiLerNW-AR denote the SMiLer with the ensemble prediction but without the self-adaptive prediction. We can see that SMiLer-GP always has a better performance than SMiLerNW-GP and SMiLerNE-GP under both measures (MAE and MNLPD). But SMiLer-AR only holds similar conclusion on the MAE measure since it lacks ability to estimate predictive uncertainty.

6.4 Practicality of SMiLer

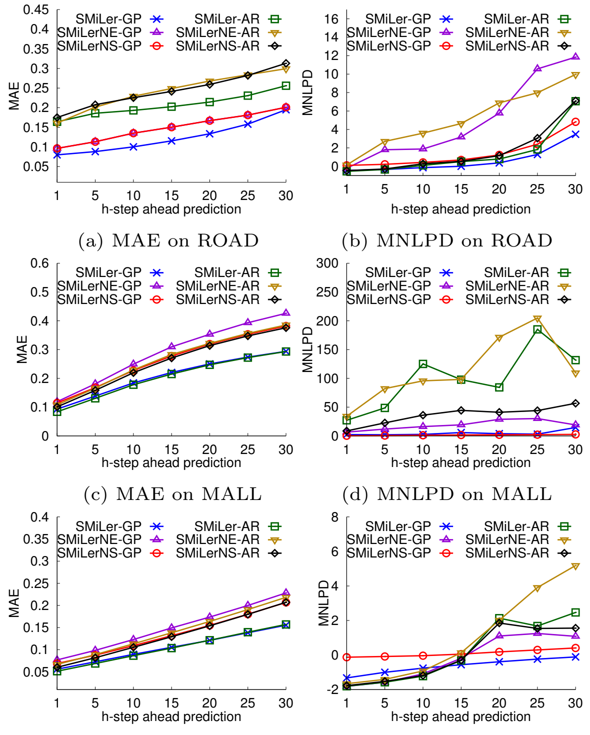
6.4.1 *Scalability of SMiLer*

Table 4 shows running time of SMiLer and its competitors. Except for FullHW and SegHW, SMiLer-GP requires a larger prediction time. However, we should note that SMiLer-GP does not have a training phase, but has the best prediction performance under both MAE and MNLPD. This shows a trade-off between the time and the accuracy.

Fig. 12 (a)(b) show the total time cost of SMiLer on all the sensors per prediction. The paybacks of the higher time cost of SMiLer-GP are more accurate prediction results (lower MAE) and better estimation of predictive uncertainty (lower MNLPD). If the predictive uncertainty is not a concern, the SMiLer-AR may still be a choice.

Note that, since the sample time interval is 5-10 minutes in the datasets, both SMiLer-AR and SMiLer-GP can perform prediction for all sensors in real time. Besides, the running time of SMiLer-GP can be further reduced by multithreading on multi-core architecture.

SMiLer scales well with the number of sensors in real-life applications. Fig. 12(c) shows the maximum number of sensors supported by one GPU. The space of SMiLer Index is $O(nM)$ where n is number of sensors and M is size of time series data per sensor. In our system, 6GB memory is large enough for all sensors (about 1000 sensors of ROAD data

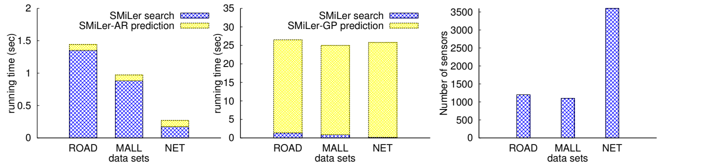


(a) MAE on ROAD (b) MNLPD on ROAD

(c) MAE on MALL (d) MNLPD on MALL

(e) MAE on NET (f) MNLPD on NET

Figure 11: Effect of the adaptive auto-tuning mechanism.



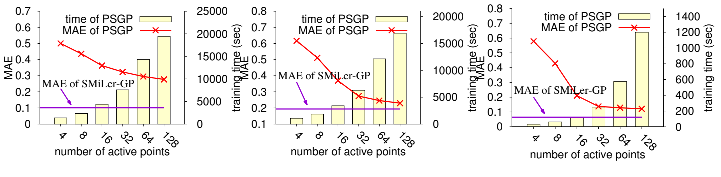
(a) SMiLer-AR (b) SMiLer-GP (c) capability of SMiLer

Figure 12: Scalability of SMiLer: (a) and (b) show total time cost of all sensors to make prediction, (c) shows maximum number of sensors per GPU on the data.

from a city) while each sensor has one year data. To support more sensors, there are two options. First, we can simply use multiple-GPU system. We also believe that the memory of the GPU will be continuous increasing while the number of sensors in a city is quite stable over years. Second, we can reduce M (the size of one sensor data). For example, if we take a sample of ten percent of ROAD dataset into the GPU, one GPU can support more than ten thousands of sensors. But its prediction performance may be degenerate. This is a trade-off between the space and the prediction performance.

6.4.2 *Comparison of PSGP and SMiLer*

We claim that SMiLer is more practical than its competitors. It seems that the low-rank approximation models, such as PSGP and VLGP, have a potential to achieve better prediction performance since their accuracy can be improved by increasing the "rank". We take PSGP as an example to reveal the impracticality of this solution.



(a) ROAD (b) MALL (c) NET

Figure 13: Comparison of PSGP and SMiLer-GP: average training time per sensor of PSGP and their MAE.

Fig. 13 indicates why SMiLer-GP is more practical than PSGP. For each dataset, we randomly selected 50 sensors to make prediction using PSGP and averaged their training time and MAE. In Fig. 13, when we vary the number of active points, the left y-axis shows the MAE of PSGP and the right y-axis shows the average training time of PSGP for each sensor. The MAE of SMiLer-GP averaged on such 50 sensors is also illustrated with violet solid line. We can see that, after the number of active points is larger than a threshold (e.g. 32), the marginal improvement of MAE is small, but the increase of the time is exponential. For example, on the ROAD dataset, the total training time for its 963 sensors with 128 active points should be about 200 days (18000*963=17334000 seconds). However, SMiLer-GP still has lower MAE than PSGP on all the datasets even if we allow such an expensive training process for PSGP.

The experiment in this section and Fig. 13 (a) also experimentally demonstrates that SMiLer can overcome the critical challenges in the "*traffic sensor prediction*" problem introduced in Example 1.1 (see Section 1). Without having a training phase, SMiLer can still achieve much better prediction performance than the approximated low-rank GP model which requires high training time cost.

7. CONCLUSIONS

We present SMiLer, a semi-lazy time series prediction system for sensors. The core idea is to employ the semi-lazy learning approach to enable GPs for time series prediction. Two challenging problems are solved, which are a suffix kNN search method under DTW and a semi-lazy GP prediction model. For the former problem, we depicted a GPU-based two-level inverted-like index for fast Suffix kNN search. For the latter one, we devised an adaptive auto-tuning mechanism integrating the ensemble and continuous prediction.

Extensive experiment study demonstrates that SMiLer performs the prediction with good accuracy, and can also scale well with carrying out prediction in real time.

8. ACKNOWLEDGMENTS

We would like to gratefully acknowledge Prof. H. V. Jagadish who provided insightful comments, inspiring advice and kind help on this paper. We would also like to give many thanks to Chang Fanxi Francis who generously shared MALL dataset to us for the experiment study. The work by Anthony K.H. Tung was partially supported by NUS FR-C Grant R-252-000-370-112. This research was carried out at the NUS-ZJU Sensor-Enhanced social Media (SeSaMe) Centre. It is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the Interactive Digital Media Programme Office.

9. REFERENCES

[1] Genie and lamp. `http://www.comp.nus.edu.sg/~atung/gl/`, 2014.

[2] R. Adhikari and R. Agrawal. A novel weighted ensemble technique for time series forecasting. In *PAKDD*, pages 38–49, 2012.

[3] T. Alabi, J. D. Blanchard, B. Gordon, and R. Steinbach. Fast k-selection algorithms for graphics processing units. *Journal of Experimental Algorithmics*, 17:4–2, 2012.

[4] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[5] I. Assent, R. Krieger, F. Afschari, and T. Seidl. The ts-tree: efficient time series search and retrieval. In *EDBT*, pages 252–263, 2008.

[6] V. Athitsos, P. Papapetrou, M. Potamias, G. Kollios, and D. Gunopulos. Approximate embedding-based subsequence matching of time series. In *SIGMOD*, pages 365–378, 2008.

[7] K. Bache and M. Lichman. UCI machine learning repository. `https://archive.ics.uci.edu/ml`, 2013.

[8] T. Ban, R. Zhang, S. Pang, A. Sarrafzadeh, and D. Inoue. Referential knn regression for financial time series forecasting. In *ICONIP*, pages 601–608, 2013.

[9] R. Barillec, B. Ingram, D. Cornford, and L. Csató. Projected sequential gaussian processes: A c++ tool for interpolation of large datasets with heterogeneous noise. *Computers&Geosciences*, 37(3):295–309, 2011.

[10] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI94 workshop on knowledge discovery in databases*, pages 359–370, 1994.

[11] G. Biau, K. Bleakley, L. Györfi, and G. Ottucsák. Non-parametric sequential prediction of time series. *Journal of Nonparametric Statistics*, 22(3):297–317, 2010.

[12] E. Blanzieri and F. Melgani. Nearest neighbor classification of remote sensing images with the maximal margin principle. *IEEE Transactions on Geoscience and Remote Sensing*, 46(6):1804–1811, 2008.

[13] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327, 1986.

[14] L. Bottou. On-line learning and stochastic approximations. In *On-line learning in neural networks*, pages 9–42. Cambridge University Press, 1999.

[15] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, 4th edition, 2008.

[16] P. Boyle and M. Frean. Dependent gaussian processes. In *NIPS*, pages 217–224, 2004.

[17] S. Brahim-Belhouari and A. Bermak. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis*, 47(4):705–712, 2004.

[18] D. Chakrabarti and C. Faloutsos. F4: large-scale automated forecasting using fractals. In *CIKM*, pages 2–9, 2002.

[19] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM TIST*, 2(3):27, 2011.

[20] N. Chapados and Y. Bengio. Augmented functional time series representation and forecasting with gaussian processes. In *NIPS*, pages 457–464, 2007.

[21] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[22] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[23] T. Chen and J. Ren. Bagging for gaussian process regression. *Neurocomputing*, 72(7):1605–1610, 2009.

[24] Y. Chen, M. Nascimento, B. Ooi, and A. Tung. Spade: On shape-based pattern detection in streaming time series. In *ICDE*, pages 786–795, 2007.

[25] L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668, 2002.

[26] C. Cuda. Programming guide. *NVIDIA Corporation*, 2014.

[27] M. Cuturi. Fast global alignment kernels. In *ICML*, pages 929–936, 2011.

[28] M. Cuturi. Pems-sf data set. `https://archive.ics.uci.edu/ml/datasets/PEMS-SF`, 2014.

[29] DataMarket. Internet traffic data. `http://data.is/19Cbyed`, 2014.

[30] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008.

[31] R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, pages 987–1007, 1982.

[32] C. Faloutsos and R. T. Snodgrass. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[33] R. Furrer, M. G. Genton, and D. Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3), 2006.

[34] A. Girard, C. E. Rasmussen, J. Quinonero-Candela, and R. Murray-Smith. Gaussian process priors with uncertain inputs-application to multiple-step ahead time series forecasting. In *NIPS*, pages 545–552, 2002.

[35] T. Hachino and V. Kadirkamanathan. Time series forecasting using multiple gaussian process prior model. In *CIDM*, pages 604–609, 2007.

[36] W.-S. Han, J. Lee, Y.-S. Moon, S.-W. Hwang, and H. Yu. A new approach for processing ranked subsequence matching based on ranked union. In *SIGMOD*, pages 457–468, 2011.

[37] W.-S. Han, J. Lee, Y.-S. Moon, and H. Jiang. Ranked subsequence matching in time-series databases. In *VLDB*, pages 423–434, 2007.

[38] C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.

[39] R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(i03), 2008.

[40] J.-T. Jeng, C.-C. Chuang, and C.-W. Tao. Hybrid svmr-gpr for modeling of chaotic time series systems with noise and outliers. *Neurocomputing*, 73(10):1686–1693, 2010.

[41] E. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[42] Y. Liu, A. Choudhary, J. Zhou, and A. Khokhar. A scalable distributed stream mining system for highway traffic data. In *PKDD*, pages 309–321. 2006.

[43] K. H. Low, J. Yu, J. Chen, and P. Jaillet. Parallel gaussian process regression for big data: low-rank representation meets markov approximation. In *AAAI*, 2015.

[44] J. McNames. A nearest trajectory strategy for time series prediction. In *Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*, pages 112–128, 1998.

[45] Y.-S. Moon, K.-Y. Whang, and W.-K. Loh. Duality-based subsequence matching in time-series databases. In *ICDE*, pages 263–272, 2001.

[46] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *ICANN*, pages 999–1004, 1997.

[47] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[48] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[49] A. O'Hagan and J. Kingman. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–42, 1978.

[50] C. J. Paciorek and M. J. Schervish. Nonstationary covariance functions for gaussian process regression. In *NIPS*, pages 273–280, 2003.

[51] C. Park, J. Z. Huang, and Y. Ding. Domain decomposition approach for fast gaussian process regression of large spatial data sets. *JMLR*, 12:1697–1728, 2011.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12:2825–2830, 2011.

[53] PeMS. Freeway performance measurement system. `http://pems.dot.ca.gov/`, 2014.

[54] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, pages 262–270, 2012.

[55] C. E. Rasmussen. *Evaluation of Gaussian Process and Other Methods for Non-linear Regression.* PhD thesis, University of Toronto, 1996.

[56] C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning.* MIT Press, 2006.

[57] C. A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *SDM*, pages 506–510, 2005.

[58] G. Ristanoski, W. Liu, and J. Bailey. Time series forecasting using distribution enhanced linear regression. In *PAKDD*, pages 484–495. 2013.

[59] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*, volume 589. John Wiley & Sons, 2005.

[60] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul. Accelerating dynamic time warping subsequence search with gpus and fpgas. In *ICDM*, pages 1001–1006, 2010.

[61] N. Segata and E. Blanzieri. Fast and scalable local kernel machines. *JMLR*, 11:1883–1926, 2010.

[62] SheffieldML. Gpy gaussian process toolkit. `http://sheffieldml.github.io/GPy/`, 2014.

[63] L. T. A. Singapore. Carpark lots availability. `https://www.mytransport.sg/content/mytransport/home/dataMall.html`, 2014.

[64] S. Sundararajan and S. S. Keerthi. Predictive approaches for choosing hyperparameters in gaussian processes. *Neural Computation*, 13(5):1103–1118, 2001.

[65] M. K. Titsias. Variational learning of inducing variables in sparse gaussian processes. In *AISTATS*, pages 567–574, 2009.

[66] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[67] Y. Wang and B. Chaib-Draa. A knn based kalman filter gaussian process regression. In *IJCAI*, pages 1771–1777, 2013.

[68] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[69] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, pages 682–688, 2001.

[70] C. K. Williams and C. E. Rasmussen. Gaussian processes for regression. In *NIPS*, pages 514–520, 1996.

[71] P. R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342, 1960.

[72] W. Yan, H. Qiu, and Y. Xue. Gaussian process for long-term time-series forecasting. In *IJCNN*, pages 3420–3427, 2009.

[73] A. Zakai and Y. Ritov. Consistency and localizability. *JMLR*, 10:827–856, 2009.

[74] H. Zhang, A. C. Berg, M. Maire, and J. Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR*, pages 2126–2136, 2006.

[75] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML*, pages 116–123, 2004.

[76] J. Zhou, A. K. Tung, W. Wu, and W. S. Ng. R2-d2: a system to support probabilistic path prediction in dynamic environments via “semi-lazy” learning. *PVLDB*, 6(12):1366–1369, 2013.

[77] J. Zhou, A. K. Tung, W. Wu, and W. S. Ng. A “semi-lazy” approach to probabilistic path prediction in dynamic environments. In *KDD*, pages 748–756, 2013.

[78] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *SIGMOD*, pages 181–192, 2003.

APPENDIX

A. DATA SOURCE OF FIG. 1

The data of Fig. 1(a) is selected from `http://ark.intel.com/` according to the family of the Intel Xeon Processor E5 and 5000. Fig. 1(b) is depicted from data of Dr. Michael Galloy in `https://github.com/mgalloy/cpu-vs-gpu`. Fig. 1(c) is drawn according to a list maintained by Prof. John C. McCallum in `http://www.jcmit.com/memoryprice.htm` whose major source is `NewEgg.com`. The data of Fig. 1(d) is collected from `http://www.techpowerup.com/gpudb/` according to the family of the NVIDIA GeForce GPU.

B. CONCEPT AND BACKGROUND

B.1 Dynamic Time Wrapping

We give a short explanation of Dynamic Time Warping (DTW) here. For more details, please refer to [41]. Without losing the generality, we assume that all time series are of equal length (see [57]). We have two time series $Q = [q_1, q_2, ..., q_d]$ and $C = [c_1, c_2, .., c_d]$. The DTW computes the best possible alignment between Q and C with respect to the overall warping cost. In Fig. 14(a), a warping matrix γ is constructed where element (i, j) corresponds to the alignment between $q_i \in Q$ and $c_j \in C$. Indeed, computing DTW is to find a contiguous set of matrix elements, called warping path, which defines the optimal alignment between Q and C (see Fig. 14). Typically, the warping path is restricted to not more than ρ cells from the diagonal [41, 54, 5]. This constraint is called Sakoe-Chiba band where ρ is called the warping width. This band constraint not only reduces the computation cost of DTW, but also avoids the degenerated matchings (e.g. most of elements of a time series are matched to several elements of the other) [5]. In this paper, we only consider the DTW with Sakoe-Chiba band constraint. The definition of DTW with ρ warping path is:

$$\gamma(i,j) = dist(q_i, c_j) + \min \begin{cases} \gamma(i-1,j) \\ \gamma(i,j-1) \\ \gamma(i-1,j-1) \end{cases} \tag{21}$$

$$\gamma(0,0) = 0, \gamma(i,0) = \gamma(0,j) = \infty, \tag{22}$$

$$dist(q_i, c_j) = \infty \ if|i-j| > \rho \tag{23}$$

$$D(Q,C) = \gamma(d,d) \tag{24}$$

where $dist(\cdot)$ is the distance between observations.

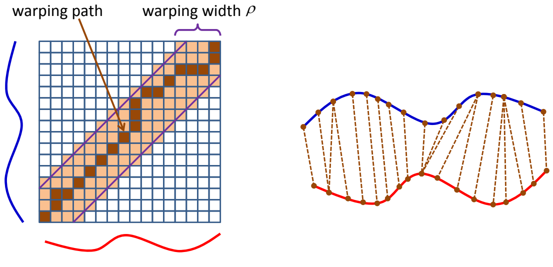
Now we give an introduction about the popular lower bound of DTW – *LB_keogh* [41]. We first define the “envelope” of time series.

Definition B.1 (Time Series Envelope). *Given a time series C and a warping width ρ, the envelope* $\mathrm{E}(C)$ *contains two sequences: upper envelope* $\mathrm{U}(C)$ *and lower envelope* $\mathrm{L}(C)$*, whose i-th elements are defined as:*

$$U_i^c = \max_{-\rho \le r \le \rho} (c_{i+r}), \quad L_i^c = \min_{-\rho \le r \le \rho} (c_{i+r}) \tag{25}$$

LB_keogh is the distance between E(C) and the query Q:

$$LB\_keogh(\mathrm{E}(C), Q) = \sum \begin{cases} dist(U_i^c, q_i) & q_i > U_i^c \\ dist(L_i^c, q_i) & q_i < L_i^c \\ 0 & otherwise \end{cases} \quad (26)$$



(a) (b)

Figure 14: (a) An illustration for warping matrix with warping width and warping path; (b) Result alignment according to warping path.[1]

B.2 Graphics Processing Unit

The Graphics Processing Unit (GPU) is a computing device that provides a massively parallel execution environment for many threads. With all of the threads running on multiple processing cores, and executing the same program on separate data, the GPU shares many aspects of Single-Instruction-Multiple-Data (SIMD) architecture.

We implemented our algorithm on a NVIDIA GPU using the Compute Unified Device Architecture (CUDA) toolkit [26]. Each CUDA function is executed by an array of threads. A small batch (e.g. 256) of threads are organized as one block that possesses an associated pool of "shared memory" for the cooperation of the threads. Note that the shared memory is much faster than the global memory of the GPU.

B.3 Gaussian Process

We give a brief review of Gaussian Processes (GPs) here. Please refer to [56] for a comprehensive introduction. A Gaussian Process is a collection of random variables, any subset of which has a joint normal distribution. Suppose that a set of data pairs $(X, Y) = \{x_i, y_i\}_{i=1}^k$ are random variables, where x_i is a d-dimensional vector and y_i is the predicted value. We can assume that there is an underlying prediction function $f(\cdot)$ such that $\hat{y}_i = f(x_i)$ is based on the GP, which is fully specified by the mean function $m(x)$ and the covariance function $c(x_i, x_j)$. We usually further assume that the mean function is set to be zero, i.e.:

$$[y_1, y_2, ..., y_n]^\top \sim \mathcal{GP}(0, \Sigma) \quad (27)$$

where $\Sigma_{ij} = cov(y_i, y_j) = cov(f(x_i), f(x_j)) = c(x_i, x_j)$, which specifies the covariance between pairs of random variables. A widely-used covariance function is the squared exponential (SE) covariance function, i.e.,

$$\begin{aligned} cov(y_i, y_j) &= cov(f(x_i), f(x_j)) = c(x_i, x_j) \\ &= \theta_0^2 exp\left(-\frac{1}{2}\frac{\|x_i - x_j\|^2}{\theta_1^2}\right) + \delta_{ij}\theta_2^2 \end{aligned}$$

where δ_{ij} is a Kronecker delta. $\delta_{ij} = 1$ if and only if $i = j$ and $\delta_{ij} = 0$ otherwise. The vector $\Theta = \{\theta_0, \theta_1, \theta_2\}$ is a set of hyperparameters. In particular, θ_1 is called a *characteristic length-scale*, which determines how relevant an input is: if the length-scale has a very large value, the covariance becomes almost independent of that input, effectively removing it from the inference.

Now given data $(X, Y) = \{x_i, y_i\}_{i=1}^n$, and a test input vector x_0, we want to estimate the predictive distribution of the value y_0 corresponding to the input x_0. Given the Gaussian Process prior and Bayesian rules, the joint distribution of the observed values and the predicted value y_0 is given by

$$\begin{bmatrix} Y \\ y_0 \end{bmatrix} \sim \mathcal{GP}\left(0, \begin{bmatrix} C(X, X) & C(X, x_0) \\ C(x_0, X) & c(x_0, x_0) \end{bmatrix}\right) \quad (28)$$

where $C(X, x_0) = [c(x_1, x_0), ..., c(x_n, x_0)]^\top$ is the $n \times 1$ vector of variances between the test vector and training vectors (similar for the other entries $C(X, X), C(x_0, x_0)$ and $C(x_0, X)$). For simplicity, we use a compact notation as $C = C(X, X)$, $c_0 = C(X, x_0) = C^\top(x_0, X)$.

In the Gaussian Process model, for a test input x_0, the predictive distributive is simply obtained through conditioning on the training data. The joint distribution of the variables being Gaussian, the posterior distribution for the input test data $p(\hat{y}_0|x_0, X, Y)$ is also a Gaussian distribution, with the following mean and variance:

$$\hat{y}_0 = f(x_0, X, Y) \sim \mathcal{N}(u_0, \sigma_0^2) \quad (29)$$

$$u_0 = E(y_0) = c_0^\top C^{-1} Y \quad (30)$$

$$\sigma_0^2 = cov(y_0) = c(x_0, x_0) - c_0^\top C^{-1} c_0 \quad (31)$$

C. PROOF

C.1 Proof of Lemma 4.1

Proof. We have two observations: (O1) For the item query IQ_i with $CSG_{i,b}$, the number of points in the right side of SW_b (exclusive) is b. Then, the number of points in the left side of SW_b adding the number of points of SW_b is $d_i - b$. According to the definition of CSG, in the left side of SW_b, only $(d_i - b)\%\omega$ points are not included in $CSG_{i,b}$.

(O2) If the rightmost aligned disjoint window is DW_r, then the leftmost aligned disjoint window is $DW_{r-|CSG_{i,b}|+1}$. The starting position (from left to right) of the point of $DW_{r-|CSG_{i,b}|+1}$ is $(r - |CSG_{i,b}| + 1) * \omega$.

Then, based on (O1) and (O2), in order to match IQ_i with a segment, we only need $(d_i - b)\%\omega$ number of points from the starting position of $DW_{r-|CSG_{i,b}|+1}$ to left forward. Therefore, the starting position of the aligned segment C_{t,d_i} is $t = (r - |CSG_{i,b}| + 1) * \omega - (d_i - b)\%w$. □

C.2 Proof of Theorem 4.2

Proof. Suppose that the disjoint windows covered by segment C_{t,d_i} are $\{DW_r, DW_{r-1}, ..., DW_{r-m+1}\}$ where m is the number of the disjoint windows and r is the identifier of the rightmost disjoint window. We have $m = \lfloor\frac{(r+1)*\omega-t}{\omega}\rfloor$. We prove the corollary in two directions.

(I) There is one alignment. For the segment C_{t,d_i}, we can find a $CSG_{i,b}$ of IQ_i, where $b = t + d_i - (r+1) * w$ (b is the number of points in the right side of SW_b). The number of sliding windows in $CSG_{i,b}$ is $\lfloor\frac{d_i-b}{\omega}\rfloor$. By replacing b with $b = t + d_i - (r+1) * w$, it is computed that $m = \lfloor\frac{d_i-b}{\omega}\rfloor = \lfloor\frac{(r+1)*\omega-t}{\omega}\rfloor$. Therefore, $CSG_{i,b}$ can be aligned with the disjoint windows covered by C_{t,d_i}.

[1] The figure is reproduced and modified from [41].

(II) There is only one alignment. We prove it by contradiction. Suppose that for a segment C_{t,d_i} there are at least two CSGs, denoted by $CSG_{i,b'}$ and $CSG_{i,b''}$, which are aligned with the disjoint windows $\{DW_r, DW_{r-1}, ..., DW_{r-m+1}\}$. The number of points of C_{t,d_i} in the left side of DW_{r-m+1} is $b_l = (r-m+1)*\omega - t$ (b_l is the number of points from t to the starting point (exclusive) of the disjoint window DW_{r-m+1}). Then the total length of C_{t,d_i} is $d'_i = b_l + m*\omega + b'$ and $d''_i = b_l + m*\omega + b''$. Since $b' \neq b''$, it is obvious that $d'_i \neq d''_i$. But C_{t,d_i} have only one length, i.e. $d_i = d'_i = d''_i$. There is a contradiction. Therefore, there is only one alignment between the CSG and the disjoint windows.

Based on (I) and (II), we prove the claim. $\square$

C.3 Proof of Theorem 4.3

Proof. Suppose that the disjoint windows covered by segment C_{t,d_i} are $\{DW_r, DW_{r-1}, ..., DW_{r-m+1}\}$ where r is the identifier of the rightmost disjoint window and m is the number of disjoint windows such that $m = \lfloor \frac{(r+1)*\omega - t}{\omega} \rfloor$. Then, the number of points of C_{t,d_i} in the left side of DW_{r-m+1} is $b_l = (r-m+1)*\omega - t$ and the number of points of C_{t,d_i} in the right side of DW_r is $b_r = t + d_i - (r+1)*w$ (The calculation of b_l and b_r can be found in Appendix C.2).

For simplicity, let the distance between the point q_j and the envelop of c_i be $LB(E(c_i), q_j)$, i.e.

$$LB(E(c_i), q_j) = \begin{cases} dist(U_i^c, q_j) & q_j > U_i^c \\ dist(L_i^c, q_j) & q_j < L_i^c \\ 0 & otherwise \end{cases}$$

For the lower bound $LB_{EC}(IQ_i, C_{t,d_i})$, we have:

$$LB_{EC}(IQ_i, C_{t,d_i}) = \sum_{j=0}^{d_i-1} LB(E(c_{t+j}), q_j)$$

$$= \sum_{j=0}^{b_l-1} LB(E(c_{t+j}), q_j) + \sum_{j=d_i-b_r+1}^{d_i} LB(E(c_{t+j}), q_j)$$

$$+ \sum_{j=b_l}^{d_i-b_l-b_r} LB(E(c_{t+j}), q_j)$$

$$\geq \sum_{j=b_l}^{d_i-b_l-b_r} LB(E(c_{t+j}), q_j) = \sum_{a=0}^{m-1} LB_{EC}(SW_{b+a*\omega}, DW_{r-a})$$

In the same way, we can also get $LB_{EQ}(IQ_i, C_{t,d_i}) \geq \sum_{a=0}^{m-1} LB_{EQ}(SW_{b+a*\omega}, DW_{r-a})$. Combining these two inequalities, we have $LB_w(IQ_i, C_{t,d_i}) \leq LB_{en}(IQ_i, C_{t,d_i})$. According to Theorem 4.1, for the enhanced lower bound we also have $LB_{en}(IQ_i, C_{t,d_i}) \leq DTW(IQ_i, C_{t,d_i})$. Finally, we get $LB_w(IQ_i, C_{t,d_i}) \leq DTW(IQ_i, C_{t,d_i})$. $\square$

D. ALGORITHM FOR COMPUTING LOWER BOUND IN GROUP LEVEL INDEX

Algorithm 1 shows the pseudo code for constructing the posting list of CSG in the group level index of the SMiLer Index. We use one thread to scan the elements of posting lists of sliding windows in the sliding window index to compute the lower bound of item queries. There are two key points. The first one is to do shift sum (recall Eqn. (5)) to compute the lower bound LB_{EQ} and LB_{EC} in Line 7 and Line 8 by sequentially accessing the window level index. The second one is to compute LB_w (recall Theorem 4.3) for each item query in Line 11. The computed LB_w is just the element of the posting list in the group level index.

Algorithm 1: Group level index construction: compute enhanced lower bound of the CSG.

```
   // One block of the GPU takes one CSG
1  for each $CSG_b$ of master query $MQ$ do
2      for each disjoint window $DW_r$ of $C$ do
3          $j \leftarrow 0$ // count window number
4          $i \leftarrow 0$ // count item query number
5          $d \leftarrow b + \omega$ // $\omega$ is window length
           // $n$ is number of item queries per $MQ$
6          while $i < n$ do
               // access window level index
7              $LB_q \leftarrow LB_{EQ}(SW_{b+j*\omega}, DW_{r-j})$
8              $LB_c \leftarrow LB_{EC}(SW_{b+j*\omega}, DW_{r-j})$
9              if $d + \omega > |IQ_i|$ and $d \leq |IQ_i|$ then
10                 $t \leftarrow (r-j)*\omega - (d-b)\%\omega$
11                 $LB_w(IQ_i, C_{t,d}) \leftarrow \max\{LB_q, LB_c\}$
12                 Store $LB_w(IQ_i, C_{t,d})$ in posting list of
                   $CSG_{i,b}$ of $IQ_i$
13                 $i \leftarrow i + 1$ // for next item query
14             $j \leftarrow j + 1$, $d \leftarrow d + \omega$
```

E. ALGORITHM FOR COMPUTING DTW

Algorithm 2: Compute DTW distance

```
   input : a query $Q$ and a time series $C$ ($|Q| = |C| = d$)
   output: The DTW distance between $Q$ and $C$
   // This is pseudo-code for one thread.
1  $m = 2*\rho + 2$;// $\rho$ is warping width
2  $\gamma[m][2]$ // $\gamma$ is allocated in shared memory
3  for $i \leftarrow 1$ to $m - 1$ do
4      $\gamma(i, 0) = \infty$
5  $\gamma(0, 1) = \infty$
6  for $j \leftarrow 1$ to $d$ do
7      $\gamma((j - \rho - 1)\%m, j\%2) = \infty$
8      $\gamma((j + \rho)\%m, (j - 1)\%2) = \infty$
9      for $i \leftarrow (j - \rho)$ to $(j + \rho)$ do
10         $\gamma(i\%m, j\%2) =$
           $dist(q_i, c_j) + \min \begin{cases} \gamma((i-1)\%m, j\%2) \\ \gamma(i\%m, (j-1)\%2) \\ \gamma((i-1)\%m, (j-1)\%2) \end{cases}$
11 return $\gamma(d\%m, d\%m)$// $|Q| = |C| = d$
```

In Algorithm 2, we show the pseudo code to compute the DTW distance (with Sakoe-Chiba band constraint) between a query Q and a time series C. It is worth noting that, in order to reduce the number of accesses to the global memory, the query Q should reside in the shared memory; and furthermore, the query Q must be placed in the inner loop of Algorithm 2 (i.e. Q should be placed in line 9 instead of line 6) [60].