Learning Autoregressive Model in LSM-Tree based Store

Yunxiang Su
Tsinghua University
Beijing, China
suyx21@mails.tsinghua.edu.cn

Wenxuan Ma
Tsinghua University
Beijing, China
mwx22@mails.tsinghua.edu.cn

Shaoxu Song
BNRist, Tsinghua University
Beijing, China
sxsong@tsinghua.edu.cn

ABSTRACT

Database-native machine learning operators are highly desired for efficient I/O and computation costs. While most existing machine learning algorithms assume the time series data fully available and readily ordered by timestamps, it is not the case in practice. Commodity time series databases store the data in pages with possibly overlapping time ranges, known as LSM-Tree based storage. Data points in a page could be incomplete, owing to either missing values or out-of-order arrivals, which may be inserted by the imputed or delayed points in the following pages. Likewise, data points in a page could also be updated by others in another page, for dirty data repairing or re-transmission. A straightforward idea is thus to first merge and order the data points by timestamps, and then apply the existing learning algorithms. It is not only costly in I/O but also prevents pre-computation of model learning. In this paper, we propose to offline learn the AR models locally in each page on incomplete data, and online aggregate the stored models in different pages with the consideration of the aforesaid inserted and updated data points. Remarkably, the proposed method has been deployed and included as a function in an open source time series database, Apache IoTDB. Extensive experiments in the system demonstrate that our proposal LSMAR shows up to one order-of-magnitude improvement in learning time cost. It needs only about 10s of milliseconds for learning over 1 million data points.

CCS CONCEPTS

• **Information systems** → **Query operators**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Autoregressive models, LSM store, in-database operators

ACM Reference Format:
Yunxiang Su, Wenxuan Ma, and Shaoxu Song. 2023. Learning Autoregressive Model in LSM-Tree based Store. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3580305.3599405

1 INTRODUCTION

IoT data are often collected in a preset frequency, e.g., in every second, leading to time series with regular intervals. However, the data arrivals are often out-of-order, owing to various network delays in the IoT scenarios [11, 16]. Moreover, some points could be missing and inserted back by data imputation program [21, 25]. Furthermore, dirty values are also detected and updated later by requesting point re-transmission or data repairing program [26].

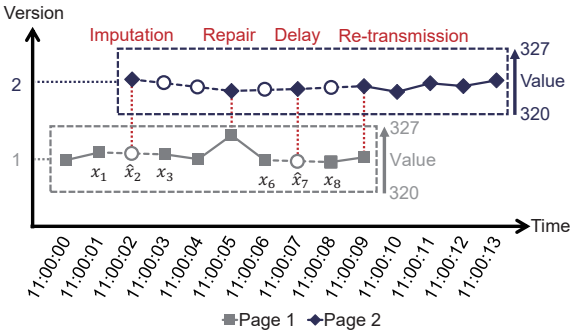


Figure 1: Example of imputed, repaired, delayed and re-transmitted data in LSM-Tree store for learning AR models

To handle the aforesaid out-of-order arrival, point insertion for imputation, value update for repairing and so on [18], most commodity time series databases, such Apache IoTDB [3], employ a Log-Structured Merge-Tree (LSM-Tree) [20] based storage. As illustrated in Figure 1, data points are batched in disk pages referring to their arrivals. Some points such as the one at time 11:00:07 are delayed and batched with other data points in page 2. The corresponding position in page 1 is thus denoted by a hollow circle. Likewise, the missing point at time 11:00:02 is imputed, again in page 2. Moreover, the point at time 11:00:05 with dirty value in page 1 is repaired later by the one at the same time in page 2, while the point at time 11:00:09 is updated by data re-transmission.

To learn models over the time series scattered in different pages with possibly overlapping time ranges, a straightforward idea is to first load the data in disk pages to memory. Then, they are merged by inserting the delayed or imputed points, and updating the re-transmitted or repaired points. Existing learning algorithms [8] can thus be applied over the merged time series ordered by time. It is obviously inefficient in I/O cost to read many stale data points and merge them online. Moreover, the insertion and update of data points also prevent directly utilizing the pre-computed models in individual page when it is written to disk.

In this paper, we propose to design efficient schemes for online aggregating the pre-computed models locally in each page. The autoregressive (AR) is considered, since it is simple enough to learn during the flush of the corresponding page to disk in database ingestion. As illustrated in Figure 1, the model is pre-trained over the incomplete time series in page 1 and stored. Intuitively, when

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599405

Table 1: Notations

Symbol	Description
$\mathbf{x}$	time series of $n = \|\mathbf{x}\|$ data points
(t_h, x_h)	time and value of h-th data point in $\mathbf{x}$
p	the autoregressive model order
γ_i	the i-th order auto-covariances validity of time series $\mathbf{x}$
ϕ_i	the i-th autoregressive model coefficient

new data points appear, e.g., in page 2, the pre-computed models could be fine-tuned rather than learning from scratch.

Our major contributions in this study are as follows.

(1) We devise the learning process of models in a single page in Section 3, including learning models with imputation, and updating models with modified points.

(2) We propose the efficient aggregation of models learned in different pages, with the consideration of inserted and updated values, in Section 4. For pages with various cases, i.e., adjacent, disjoint and overlapped in time ranges, we derive the corresponding model aggregation strategies, respectively. The theoretical results in Propositions 4.1, 4.3, 4.5 and 4.7 guarantee the correctness of model aggregation, i.e., equivalent to the baseline of learning over the merged time series.

(3) We present the algorithm for learning models on time series scattered in different pages in Section 5. The complexity analysis illustrates that our proposal is more efficient than the baseline of learning from scratch over the online merged time series. Moreover, we provide the details about system deployment, as a function in Apache IoTDB [3], an open-source time series database. The document is available in the product website [4], and the code is included in the product repository by system developers [1].

(4) We conduct extensive experiments for evaluation in Section 6. Our proposal LSMAR shows up to one order-of-magnitude improvement in learning time cost, compared to the aforesaid baseline of online merging data and learning from scratch. It needs only about tens of milliseconds for learning over 1 million data points, while the baseline takes hundreds. The experiment code and public data are available anonymously in [2] for reproducibility.

Finally, we discuss related work in Section 7 and conclude the paper in Section 8. Table 1 lists the notations used in the paper.

2 PRELIMINARY

For a better comprehension of our proposal, we first introduce the baseline autoregressive model in Section 2.1, and we present the structure about LSM-Tree based storage with an example in Section 2.2. Section 2.3 introduces the problem of learning autoregressive models in LSM-Tree based storage.

2.1 Autoregressive Model

Autoregressive model fits a point x_l by utilizing its past few points, i.e., $x_{l-1}, x_{l-2}, \ldots, x_{l-p}$, where parameter p determines the number of the past points used for fitting autoregressive models.

Definition 2.1 (Autoregressive Model [8]). An autoregressive model of order p, denoted as $AR(p)$, is defined as

$$\hat{x}_l = \sum_{i=1}^{p} \phi_i x_{l-i} + \varepsilon_l,$$

where $\phi_1, \cdots, \phi_p$ are the parameters of models, and ε_l denotes the white noise $N(0, \sigma^2)$ at timestamp l.

Given a zero-mean time series $\mathbf{x}[1:n]$, by calculating the auto-covariances of order $0, 1, 2, \ldots, p$, denoted by $\gamma_0, \gamma_1, \gamma_2, \ldots, \gamma_p$, i.e.,

$$\gamma_i = \frac{1}{n-i} \sum_{l=1}^{n-i} x_l x_{l+i},\ i = 0, 1, \ldots, p, \tag{1}$$

the coefficients of AR models could be estimated by solving Yule-Walker Equation [10], which has the following form.

$$\begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_p \end{pmatrix} = \begin{pmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_{p-1} \\ \gamma_1 & \gamma_0 & \gamma_1 & \cdots & \gamma_{p-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma_{p-1} & \gamma_{p-2} & \gamma_{p-3} & \cdots & \gamma_0 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_p \end{pmatrix} \tag{2}$$

It is worth noting that not all time series can be modeled by autoregressive models. Generally, autoregressive models are applied on weak stationary time series.

Definition 2.2 (Weak Stationary Time Series [15]). Given a time series $\mathbf{x}$, by considering the value x_l at timestamp l as a continuous random variable, $\mathbf{x}$ is called a weak stationary time series, if

$$E(x_l) = \mu, \quad \forall l,$$
$$E(x_{l+s} x_l) = \sigma_s^2, \quad \forall l, s,$$

where μ is a constant and σ_s^2 only depends upon the lag s.

For simplicity, we consider all time series mentioned in the article as zero-mean time series. For time series with mean value not equal to 0, we replace the value x_l at timestamp l with $x_l - \mu$.

2.2 LSM-Tree Storage

To support frequent and extensive writing and reading of time series data, the Log-Structured Merge-Tree (LSM-Tree) [20] is often employed in time series database. We follow the convention of Apache IoTDB, an LSM-Tree based time series database.

Figure 2 presents an overview of the LSM-Tree storage structure, where a time series is stored into multiple pages, i.e., P_1 to P_6, with possibly overlapped time intervals. Each page consists of PageHeader (denoted by blue rectangles), recording metadata, e.g., StartTime and EndTime, and PageData (denoted by red rectangles), storing the batched data received in a time period. Note that each page is associated with a version, and the higher the version is, the later the batched data points are received. If pages have overlapped time intervals, e.g., P_1 and P_2, the page with the higher version would overwrite the page with the lower version.

Example 2.3. Figure 1 presents a time series stored in two pages, page 1 and page 2, recording the oil temperatures in the tank of a sailing ship. The vertical axis denotes the page version, and the value of each point is suggested by the relative height in the dotted rectangle with range from 320 to 327. Due to transmission issues,

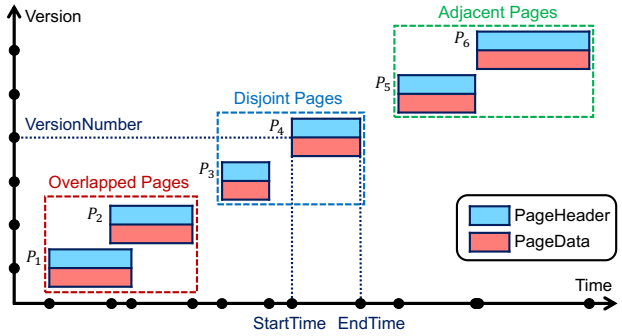


Figure 2: Aggregating models of pages in different cases

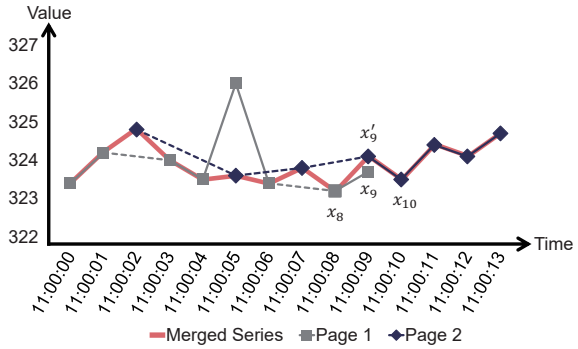


Figure 3: Merging two pages in Figure 1 as one series

some points may be missing or delayed. For instance, the point at 11:00:02 in page 1 is not received on time, while the delayed point arrives after the data points in page 1 are flushed in disk. Thus, the delayed point is batched with other points in a page with a higher version 2, leading to overwriting. Besides, the point at 11:00:07 in page 1 is a missing point (denoted by hollow circles), imputed by linear interpolation for model learning when flushing to the disk, referring to Section 3.1.

Moreover, from the perspective of LSM-Tree storage structure, the pages in Figure 1 correspond to the overlapped pages P_1 and P_2 in Figure 2. In addition to the case where pages have overlapped time intervals, pages may also be disjoint (e.g., the pages P_3 and P_4 in Figure 2) or adjacent (e.g., the pages P_5 and P_6 in Figure 2), which will be further introduced in Section 4 in detail.

2.3 Learning Models in LSM-Tree based Storage

We are now ready to introduce the model learning process in LSM-Tree based storage. With considering overwriting mentioned in Section 2.2, our aim is to efficiently aggregate the models on multiple pages by utilizing the page metadata.

To address the overlapped pages and out-of-order data points, a straightforward method is to merge all pages into one series, and then learn model coefficients from scratch. However, by utilizing the property of LSM-Tree storage, a more efficient method is to learn the aggregated coefficients from the metadata of each page.

Example 2.4 (Example 2.3 continued). Figure 3 shows the series stored in two pages, the same as illustrated in Figure 1, while with the vertical axis denoting the value. The red thick line denotes the merged series, considering that some points in page 2 may overwrite that in page 1. Notably, the overwriting only occurs from 11:00:02 to 11:00:09, that is, there is an efficient way to update the auto-covariances by only considering the influence of the points from 11:00:02 to 11:00:09. Specifically, given model order $p = 1$, we only need to update the auto-covariances of the segment from 11:00:01 to 11:00:13, referring to Formula (1) in Section 2.1. However, for the baseline method, it merges online the pages, and then learns the model on the merged series from scratch, which could be costly when the page size is large.

3 LEARNING MODELS IN A PAGE

In this section, we consider two possible cases when learning models in a page. Section 3.1 elaborates the learning process with missing points. Section 3.2 introduces the updating process when some points are modified, which is further utilized to re-calculate the auto-covariances of updated segments in Section 4.3.

3.1 Learning Models with Imputation

Due to the harsh environment in which the sensors operate, the sensor sometimes goes off-line, leading to the missing values in the sensor data stream [17]. It may affect the learning processing of models, since timestamps are not consecutive. Moreover, transmission failures may also cause missing values. We follow the similar convention in [12] to learn model coefficients with imputation in a single page, that is, utilizing a simple imputation method, linear interpolation, to fill in missing values for each missing value in the page. Again, too complicated imputation methods may not be affordable, since it is done during database ingestion.

With imputation, the timestamps of data points in a page are consecutive, and we could follow the preliminary in Section 2.1 to learn models. That is, we calculate the auto-covariances of data points in a page, and store the auto-covariances in the metadata, e.g., PageHeader in Figure 2. Then we solve Equation (2) to obtain the model coefficients.

Example 3.1 (Example 2.3 continued). Consider again the example in Figure 1. For simplicity, we use x_l to denote the value at timestamp 11:00:l hereinafter, e.g., x_3 denotes the value at timestamp 11:00:03. During the learning process of each individual page, the missing points at 11:00:02 and 11:00:07 in page 1 are linearly interpolated by their temporally nearest points, i.e., $\hat{x}_2 = \frac{x_1+x_3}{2}$ and $\hat{x}_7 = \frac{x_6+x_8}{2}$. Then the model is learnt on the imputed complete series in page 1, and the same process for page 2.

3.2 Updating Models with Modified Points

Apart from the missing values, due to sensor data re-transmission or low quality data repairing [26], some points may be modified by other points with the same timestamps. With one point modified, the auto-covariances γ_i would change accordingly referring to the following proposition.

Proposition 3.2. *Consider a point h with value x_h in segment $\mathbf{x}[1:n]$, with the auto-covariances of the segment $\gamma_0, \gamma_1, \ldots, \gamma_p$ obtained. If the value of the point is modified to x'_h, the auto-covariances of the segment could be updated by*

$$\gamma'_i = \gamma_i + \frac{1}{n-i}(x'_h - x_h)(x_{h-i} + x_{h+i}),$$

where x_{h-i} would be set to 0 if $h - i < 1$, and x_{h+i} would be set to 0 if $h + i > n$.

Thereby, if some points are modified, the auto-covariances could be updated individually by applying Proposition 3.2 on each modified point, instead of from scratch.

Example 3.3. Consider again the merged series in Figure 3. The point at 11:00:09, is modified from $x_9 = 323.7$ to $x'_9 = 324.1$, owing to the re-transmission as aforesaid. If we only consider the influence of modifying this point, the first order auto-covariance can be updated by $\gamma'_1 = \gamma_1 + (x'_9 - x_9)/(14-1)(x_8 + x_{10}) = 0.197 + 0.4 \times (-0.3)/(14-1) = 0.188$.

4 AGGREGATING MODELS OF TWO PAGES

In this section, we consider three different cases for aggregating models of two pages $\mathbf{x}[1:h]$ and $\mathbf{x}[g:n]$, i.e., adjacent pages in Section 4.1, disjoint pages in Section 4.2 and overlapped pages in Section 4.3. Overlapped pages refer to two pages with overlapped time intervals, i.e., the start time of the high-version page is earlier than the end time of the low-version page. Moreover, if the start time of the high-version page is much later than the end time of the low-version page (more than 1 sampling interval), such two pages are considered as disjoint pages. Otherwise, if the gap between start time of the high-version page and the end time of the low-version page is exactly 1 sampling interval, such two pages are adjacent pages. For instance, Figure 2 presents 6 pages P_1 to P_6 stored in the database, covering the three situations mentioned above. Pages P_1 and P_2 are overlapped pages with overlapped time intervals, pages P_3 and P_4 are disjoint pages, and pages P_5 and P_6 are adjacent pages.

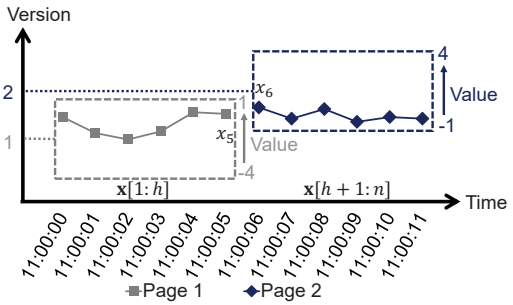
Note that once the auto-covariances are obtained, the model coefficients could be estimated by solving Equation (2) in $O(p^2)$ time [10]. Thereby, we focus on the time-consuming part, i.e., the calculation of auto-covariances γ_i in Formula (1). We show below how γ_i can be efficiently updated in various scenarios of pages, in Propositions 4.1, 4.3, 4.8 and so on.

4.1 Aggregation of Adjacent Pages

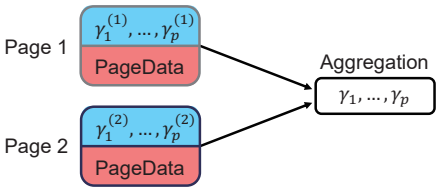
Two adjacent non-overlapped segments could be aggregated easily, by calculating the weighted sum of two segments and the auto-covariances between the tail of the former page and the head of the latter page. Proposition 4.1 below gives the expression of the aggregated auto-covariances.

Proposition 4.1. *For two adjacent non-overlapped segments $\mathbf{x}[1:h], \mathbf{x}[h+1:n]$, the i-th order auto-covariances of $\mathbf{x}[1:n]$ can be obtained by*

$$\gamma_i = \frac{h-i}{n-i}\gamma_i^{(1)} + \frac{n-h-i}{n-i}\gamma_i^{(2)} + \frac{1}{n-i}\sum_{l=1}^{i} x_{h+l}x_{h+l-i},$$



(a) Time series in adjacent pages



(b) Aggregation process of two adjacent pages

Figure 4: Example for aggregation of adjacent pages

where $\gamma_i^{(1)}$ denotes the i-th order auto-covariance of $\mathbf{x}[1:h]$, and $\gamma_i^{(2)}$ denotes the i-th order auto-covariance of $\mathbf{x}[h+1:n]$.

Example 4.2. Consider the adjacent pages in Figure 4(a) with the aggregation process present in Figure 4(b). The vertical axis in Figure 4(a) illustrates the version number of each page. Each page is surrounded by a dotted rectangle, gray for page 1 and blue for page 2, The arrow on the right side of the rectangle denotes the value range, and the relative height of each point in the dotted rectangle illustrates the value of each point. For instance, the point at 11:00:09 has value -0.3 close to the lower bound of the blue rectangle denoting -1.

With pre-computed metadata, the first order auto-covariances of both pages could be directly obtained, i.e., $\gamma_1^{(1)} = 0.49$ and $\gamma_1^{(2)} = -0.052$. Since the point at 11:00:05 has value $x_5 = 0.2$ and the point at 11:00:06 has value $x_6 = 0.6$, referring to Proposition 4.1, the aggregated γ_1 is equal to $(5\gamma_1^{(1)} + 5\gamma_1^{(2)} + x_5x_6)/(12-1) = 0.21$.

4.2 Aggregation of Disjoint Pages

In practice, not all pages are adjacent and could be aggregated by Proposition 4.1 in Section 4.1. Generally, due to the machines or sensors going-offline in a short period, there may be a piece of missing values between pages, i.e., pages are disjoint. In this section, we thus consider the aggregation of two disjoint pages under different conditions: large disjoint length in Section 4.2.1, and small disjoint length in Section 4.2.2.

The aggregation result of two disjoint pages $\mathbf{x}[1:h], \mathbf{x}[g:n]$ actually depends upon the imputation methods. For general imputation methods, it takes $O(pd)$ extra time for aggregation, in addition to the imputation time, where d denotes the disjoint length, i.e., $d = g - h$. However, if we use linear interpolation for imputation, the aggregation process could be accelerated by utilizing the properties of linear interpolation.

4.2.1 Large Disjoint Length. Consider two disjoint pages with disjoint length d larger than $i+1$, where i is the order of the auto-covariance, ranging from 1 to p. In such a case, the disjoint length is large enough to ensure that the tail of $\mathbf{x}[1:h]$ and the head of $\mathbf{x}[g:n]$ will not influence each other. Thus, we only need to calculate the i-th auto-covariance of the imputed series, and aggregate $\mathbf{x}[1:h], \mathbf{x}[g:n]$ and the imputed series as adjacent pages. The following proposition formalizes the aggregation process and utilizes the property of linear interpolation for acceleration.

Proposition 4.3. *For two disjoint segments $\mathbf{x}[1:h], \mathbf{x}[g:n]$, denoting the disjoint length by $d = g - h$, if $d - i - 1 > 0$, then the i-th order auto-covariances of $\mathbf{x}[1:n]$ can be obtained by*

$$(h+n+d-g-i)\gamma_i = (h-i)\gamma_i^{(1)} + (n-g-i+1)\gamma_i^{(2)} + (d-i-1)\gamma_i' + \sum_{l=0}^{i-1} x_{h-l}\left[x_h + (i-l)\frac{\Delta x}{d}\right] + \sum_{l=0}^{i-1} x_{g+l}\left[x_h + (d-i+l)\frac{\Delta x}{d}\right],$$

and γ_i' has the following form,

$$\gamma_i' = x_h x_g + \frac{(d-i)i}{2}\left(\frac{\Delta x}{d}\right)^2 + \frac{(d-i)(2d-2i-1)}{6}\left(\frac{\Delta x}{d}\right)^2$$

where γ_i' denotes the i-order auto-covariances of the imputed series, and $\Delta x = x_g - x_h$ denotes the difference between x_g and x_h.

Example 4.4. Consider the disjoint pages in Figure 5, with linear interpolation applied on the missing points between page 1 and page 2 in Figure 5(a). The imputed points are denoted by red hollow circles, and for simplicity, we denote the imputed point at 11:00:l by $\hat{x}_l$. The disjoint length between two pages is $d = 7 - 3 = 4$.

Given $i = 1$, the formula $d - i - 1 = 4 - 1 - 1 > 0$ satisfies the condition in Proposition 4.3, that is, the disjoint length is large enough to ensure that two pages will not influence each other. Since $x_3 = -0.9$ and $x_7 = -0.1$, the first auto-covariances of the imputed series is $\gamma' = x_3 x_7 + \frac{3\times1\times0.2^2}{2} + \frac{3\times5\times0.2^2}{6} = 0.25$. With the pre-computed $\gamma_1^{(1)} = 0.887, \gamma_1^{(2)} = -0.05$, the aggregated γ_1 is $\gamma_1 = (3\gamma_1^{(1)} + 4\gamma_1^{(2)} + 2\gamma' + x_3\hat{x}_4 + \hat{x}_6 x_7)/(12-1) = 0.329$.

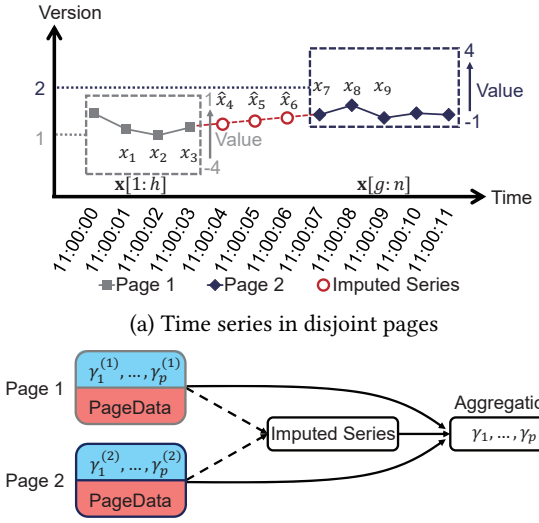
4.2.2 Small Disjoint Length. Proposition 4.3 considers two disjoint pages with a relatively large disjoint length, i.e., $d > i+1$, while Proposition 4.5 considers its complement, i.e., $d \le i+1$. In such case, the disjoint length is so small that the i-th auto-covariance of the imputed series is 0. Besides, the tail of $\mathbf{x}[1:h]$ and the head of $\mathbf{x}[g:n]$ will influence each other, unfortunately. Proposition 4.5 aggregates disjoint pages by considering the effect between the tail of $\mathbf{x}[1:h]$, the imputed series and the head of $\mathbf{x}[g:n]$

Proposition 4.5. *For two disjoint segments $\mathbf{x}[1:h], \mathbf{x}[g:n]$, denoting the disjoint length by $d = g - h$, if $d - i - 1 \le 0$, then the i-th order auto-covariances of $\mathbf{x}[1:n]$ can be obtained by*

$$(h+n+d-g-i)\gamma_i = (h-i)\gamma_i^{(1)} + (n-g-i-1)\gamma_i^{(2)} + \sum_{l=1}^{d-1}\left(x_h + l\frac{\Delta x}{d}\right)x_{h-i+l} + \sum_{l=1}^{i} x_{g+l-1}\left\{x_{h+l+d-i-1}[1-I(l)] + \left[x_h + (l-i+d-1)\frac{\Delta x}{d}\right]I(l)\right\},$$

where $I(l)$ is the characteristic function with the following form,

$$I(l) = \begin{cases} 0, & \text{if } \ l \le i-d+1 \\ 1, & \text{otherwise} \end{cases}.$$



(a) Time series in disjoint pages

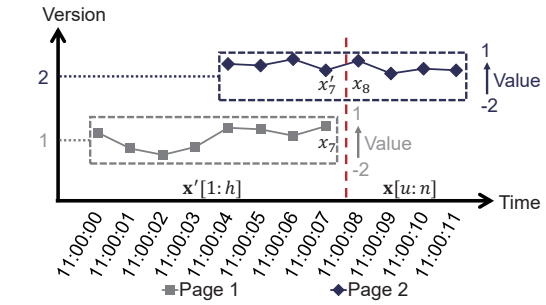
(b) Aggregation process of two disjoint pages

Figure 5: Example for aggregation of disjoint pages

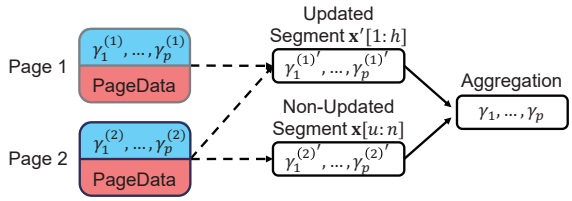
Example 4.6. Consider the disjoint pages in Figure 5 again with disjoint length $d = 4$, while given $i = 3$. Since $d - i - 1 = 4 - 3 - 1 = 0$ satisfies the condition in Proposition 4.5, the tail of page 1 and the head of page 2 will influence each other. Given the pre-computed $\gamma_3^{(1)} = 0, \gamma_3^{(2)} = -0.025$, the aggregated γ_3 is $\gamma_3 = (\gamma_3^{(1)} + 2\gamma_3^{(2)} + \hat{x}_6 x_3 + \hat{x}_5 x_2 + \hat{x}_4 x_1 + x_7\hat{x}_4 + x_8\hat{x}_5 + x_9\hat{x}_6)/(12-3) = 0.17$.

4.3 Aggregation of Overlapped Pages

We next consider the aggregation of overlapped segments. For a time series $\mathbf{x}[1:n]$ stored in two segments $\mathbf{x}[1:h], \mathbf{x}[g:n]$ with overlapping time intervals, we denote the updated segments by $\mathbf{x}'$ whose points are merged from multiple original segments. For the non-updated segments, whose points can be directly obtained from one original segment, we denote such segments by $\mathbf{x}$ with different ending indexes from the original segments. Intuitively, the overlapped segments $\mathbf{x}[1:h], \mathbf{x}[g:n]$ can be split into two non-overlapped segments $\mathbf{x}'[1:h], \mathbf{x}[u:n]$. Then the auto-covariances of $\mathbf{x}[1:n]$ can be derived by aggregating the non-overlapped segments $\mathbf{x}'[1:h], \mathbf{x}[u:n]$, referring to the following proposition.



(a) Time series in overlapped pages



(b) Aggregation process of two overlapped pages

Figure 6: Example for aggregation of overlapped pages

Proposition 4.7. *For two overlapped segments* $\mathbf{x}[1:h], \mathbf{x}[g:n]$ *with* $h \geq g$*, the i-th order auto-covariance of* $\mathbf{x}[1:n]$ *is equal to the aggregated auto-covariance of non-overlapped segments* $\mathbf{x}'[1:h], \mathbf{x}[u:n]$*, where* u *is the earliest point in* $[g:n]$ *with* $u > h$.

In real unordered scenarios, only few points are involved in the overlap and modified. Thus, we can utilize Proposition 3.2 to calculate the auto-covariances of the updated segment $\mathbf{x}'[1:h]$, by considering each modified point in turn.

Moreover, since sensors usually re-transmit values in a short period, the overlap length is small in general. Therefore, there is an intuitive way for efficiently calculating the auto-covariances of the split non-updated segments, that is, eliminating the influence of the discarding points, referring to the following proposition.

Proposition 4.8. *For a non-updated segment* $\mathbf{x}[u:v] \subset \mathbf{x}[g:n]$*, with* $g \leq u \leq v \leq n$*, its auto-covariances can be obtained by eliminating the influence of the discarding points, i.e.,*

$$(v-u+1)\gamma'_i = (n-g+1)\gamma_i - \sum_{l=g}^{u-1} x_l x_{l+i} - \sum_{l=v+1}^{n} x_{l-i}x_l.$$

Example 4.9. Consider the overlapped pages in Figure 6, where the points from 11:00:04 to 11:00:07 in page 2 overwrite those in page 1. As aforesaid, the overlapped pages could be split into two non-overlapped segments, i.e., updated segment $\mathbf{x}'[1:h]$, and non-updated segment $\mathbf{x}[u:n]$, split by two dotted red lines in the figure. For the updated segment $\mathbf{x}'[1:h]$, we only need to consider the influence of the points from 11:00:04 to 11:00:07 by Proposition 3.2, and calculate the first order auto-covariance $\gamma_1^{(1)'} = 0.359$. For non-updated segment $\mathbf{x}[u:n]$, we only need to eliminate the influence of the points from 11:00:04 to 11:00:07 in page 2, referring to Proposition 4.8, and we have $\gamma_1^{(2)'} = -0.05$ for segment $\mathbf{x}[u:n]$.

Algorithm 1: LSMAR, learning autoregressive model on multi-segment time series

Input: a set $\mathcal{X}$ of the segments of time series $\mathbf{x}$, AR model order p, sampling interval τ
Output: AR model coefficients Φ

1 order the segments in $\mathcal{X}$ by the start times
2 **foreach** *pair of consecutive segments* $\mathbf{x}[m:h], \mathbf{x}[g:n]$ **do**
3 **if** $h \geq g$ **then** `// overlapped cases`
4 split $\mathbf{x}[m:h], \mathbf{x}[g:n]$ into $\mathbf{x}'[m:h], \mathbf{x}[u:n]$ by Proposition 4.7
5 **foreach** $i := 0$ *to* p **do**
6 calculate $\gamma_i^{(1)'}$ of $\mathbf{x}'[m:h]$ by Proposition 3.2
7 calculate $\gamma_i^{(2)'}$ of $\mathbf{x}[u:n]$ by Proposition 4.8
8 $\mathcal{X} := \mathcal{X} \backslash \{\mathbf{x}[m:h], \mathbf{x}[g:n]\}$
9 $\mathcal{X} := \mathcal{X} \cup \{\mathbf{x}'[m:h], \mathbf{x}[u:n]\}$
10 **foreach** *pair of consecutive segments* $\mathbf{x}[m:h], \mathbf{x}[g:n]$ **do**
11 **if** $g > h+1$ **then** `// disjoint cases`
12 $d := g - h$
13 aggregate $\gamma_0, \ldots, \gamma_p$ by Proposition 4.3 and 4.5
14 **else** `// adjacent cases`
15 aggregate $\gamma_0, \ldots, \gamma_p$ by Proposition 4.1
16 $\Phi(\phi_0, \ldots, \phi_p) :=$ Yule-Walker$(\gamma_0, ..., \gamma_p)$
17 return Φ

Then by Proposition 4.1, we can aggregate the overlapped pages by $\gamma' = (7\gamma_1^{(1)'} + 3\gamma_1^{(2)'} + x'_7 x_8)/(12-1) = 0.21$.

5 IMPLEMENTATION IN LSM-TREE STORE

In this section, we focus on the implementation of our proposal in LSM-Tree based database. Section 5.1 proposes the algorithm for learning models on multi-segment time series. The corresponding complexity analysis of the algorithm is given in Section 5.2. Besides, we introduce the system deployment in Section 5.3.

5.1 Learning Algorithm

Algorithm 1 presents the pseudo-code of our proposed learning method LSMAR. The segments are first sorted by their start times in Line 1. For each consecutive segments $\mathbf{x}[m:h], \mathbf{x}[g:n]$, if the segments are overlapped, i.e., $h \geq g$, they will be split into two non-overlapped segments by Proposition 4.7 in Line 4. Then the auto-covariances of the updated segments are calculated in Line 6, and the auto-covariances of the non-updated segments are calculated by Proposition 4.8 in Line 7. Lines 8 to 9 replace original segments $\mathbf{x}[m:h], \mathbf{x}[g:n]$ with split segments $\mathbf{x}'[m:h], \mathbf{x}[u:n]$. After splitting all overlapped segments, the algorithm aggregates the auto-covariances of each pair of consecutive disjoint or adjacent segments. Lines 12 to 13 aggregate disjoint segments by Propositions 4.3 and 4.5, and Line 15 aggregates adjacent segments by Proposition 4.1. At last, the model coefficients are obtained by solving Yule-Walker Equation, i.e., Equation (2), in Line 16.

Table 2: Dataset statistics and learning settings

Name	# Data points	Model order p
CRRC-Train	1m	2
CSSC-Ship	1.3m	8
WH-Chemistry	1m	7
KAGGLE-Bitcoin	1.5m	2
UCI-Gas	900k	4
UCI-PAMAP2	370k	7

5.2 Performance Analysis

Consider M segments with N data points in each segment. Among M segments, suppose there are Q segments overlapped with other segments with average overlapped length L_1, and R segments disjoint from other segments with average disjoint length L_2.

For each pair of overlapped segments, our proposal takes $O(pL_1)$ for aggregation in average. For each pair of disjoint or adjacent segments, the i-order auto-covariance γ_i can be aggregated in $O(i)$ time. Thus, aggregating all auto-covariances takes $O(p^2)$ time. Moreover, it takes $O(p^2)$ time to solve Yule-Walker Equation by utilizing the Levinson-Durbin algorithm. Thereby, the overall time cost is $O(pL_1Q + p^2(M-Q) + p^2) = O(pL_1Q + p^2M)$.

The baseline learning process takes $O(RL_2)$ time for imputation, and $O(p(MN + RL_2))$ for learning from scratch. The overall time cost of the baseline is thus $O(RL_2 + p(MN + RL_2)) = O(pMN + pRL_2)$.

5.3 System Deployment

The autoregressive model learning measure has been deployed and included as a function in Apache IoTDB [3], an open-source time series database management system. The document is available in the website [4]. By executing the following SQL statement, users can obtain the learned model coefficients of time series root.test.d0.s0.

```
select AR(s0, 'p'='3')
from root.test.d0
where time >= 2020-01-16T00:00:00
  and time <= 2020-01-16T23:59:59
```

It learns the model coefficients by utilizing the data points in the time range of [2020-01-16T00:00:00, 2020-01-16T23:59:59] of the time series root.test.d0.s0, with the user-specified model order $p = 3$. The query returns a list of model coefficients with length $p = 3$, which can be further utilized for downstream tasks, such as outlier detection, time series classification as stated in Section 7.1.

6 EXPERIMENTS

In this section, we conduct extensive experiments for evaluating the efficiency of our proposal, including (1) scalability in data sizes in Section 6.2, (2) evaluation with different model orders, i.e., query parameters p in Section 6.3, (3) evaluation under various data loads, including different page sizes in Section 6.4, different disjoint length in Section 6.5, and different overlap length in Section 6.6.

6.1 Setup

We implement the baseline autoregressive model for comparison. The baseline loads all possibly overlapped pages, merges data online, and then learns model parameters from scratch, i.e., not utilizing the pre-computed statistics. Since the result of our proposed LSMAR is exactly the same as the result of the baseline, the evaluation mainly focuses on the learning efficiency.

Table 2 lists six datasets used in the evaluation, with the first three private datasets collected from our industrial partners, and the last three public datasets. The default model order p is also listed in Table 2, which is determined by the pattern of the dataset. The default page size is set to 1024 for the experiments in Section 6.2 and 6.3, and 10240 for the experiments in Section 6.5 and 6.6.

All the experiments run on a machine with Intel Core 8 CPU (2.3 GHz) and 16 GB of memory, with Apache IoTDB v0.13.3 installed. The algorithm code has been included in the system repository of Apache IoTDB [1]. The experiment related code is available in [2].

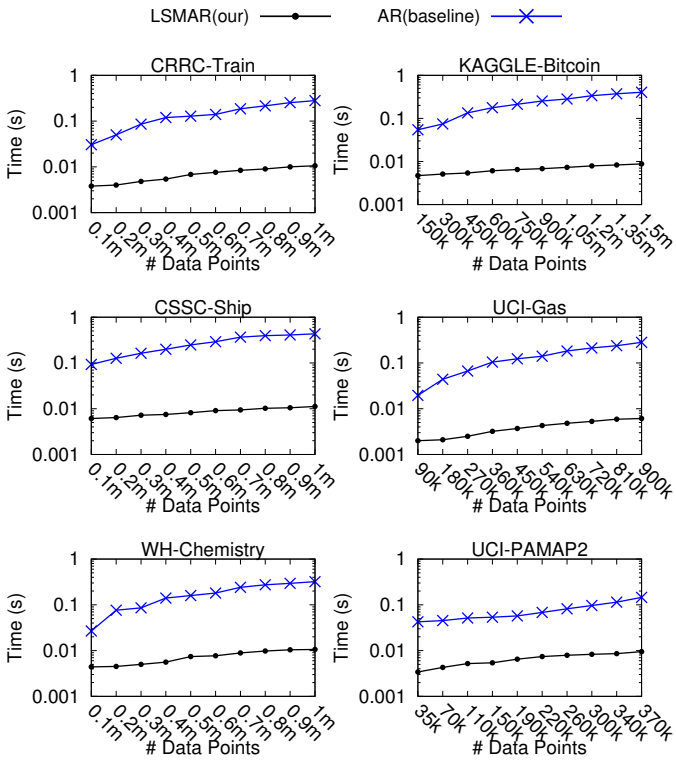


Figure 7: Time costs under different data sizes n

6.2 Scalability in Data Sizes

Figure 7 reports the time cost under different data sizes. For each dataset, we linearly vary the number of data points involved in queries and measure the corresponding time costs. When the number of data points increases, the baseline needs to learn models from more data points, leading to higher time cost. However, LSMAR only needs to aggregate more pages. Since the number of pages is much smaller than the number of data points (1 page contains 1024 data points, as stated in Section 6.1), the time cost of LSMAR increases much more slowly than the baseline, with the increase of data size. Our proposal shows great efficiency compared to the baseline, with 1-2 order of magnitude improvement.

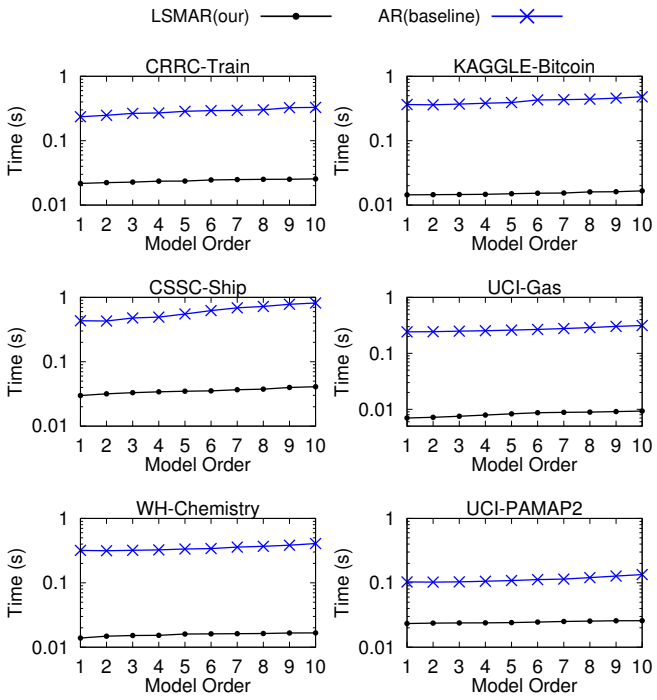


Figure 8: Time costs under different model orders p

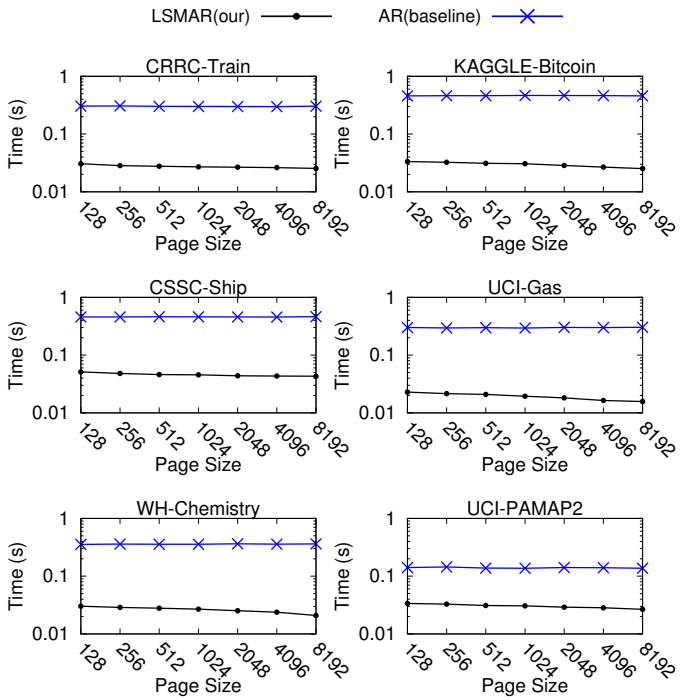


Figure 9: Time costs under different page sizes

6.3 Evaluation with Different Model Orders

Figure 8 evaluates the algorithm performance under different model orders in the query. When the model order p increases, the baseline takes more time to calculate the auto-covariances $\gamma_0, \gamma_1, \ldots, \gamma_p$, each with $O(n)$ time. Thereby, the time cost of the baseline increases with the model order p increasing. The time cost of our proposal slightly increases with the model order p increasing, while much lower than that of the baseline. This is because LSMAR just needs to aggregate each pair of consecutive pages in $O(p^2)$ time, and the number of pages M is far smaller than that of data points N, which is consistent with the analysis in Section 5.2.

6.4 Evaluation with Different Page Sizes

Figure 9 evaluates the performance under different page sizes. Page size is a parameter for LSM-Tree based storage configuration, which determines the number of data points in a page. Note that if the data size is fixed, the larger the page size is, the smaller the number of pages is. Therefore, with the increase of page size, the number of pages M decreases, and the time cost of LSMAR thus decreases. However, for the baseline method, it merges data online and learns models from scratch regardless of the configuration. Thereby, its time complexity is not affected by the page sizes, and the time cost of the baseline method in Figure 9 keeps constant.

6.5 Evaluation with Different Disjoint Length

Figure 10 varies the disjoint length between two consecutive pages. The baseline method needs to first impute all the missing values, and then learn on the imputed time series. However, our proposed LSMAR can aggregate the disjoint pages directly by Propositions 4.3 and 4.5. It only takes $O(p^2)$ time to aggregate each pair of consecutive disjoint pages, referring to the analysis in Section 5.2 Thus, the time cost of the baseline increases linearly with disjoint length increasing, while the time cost of LSMAR keeps constant.

6.6 Evaluation with Different Overlap Length

Figure 11 varies the overlap length between two consecutive pages. When the overlap length increases, the time cost of LSMAR increases, since it takes more time to calculate the auto-covariances of the updated segments and non-updated segments, as stated in Section 4.3. However, for the baseline, it always merges all data points into one series regardless of the overlap length, leading to constant time cost in Figure 11. Remarkably, though LSMAR takes more time to handle overlapped pages with the increase of overlap length, it is still much more efficient than the baseline method due to the utilization of the metadata.

7 RELATED WORK

7.1 Autoregressive Model

Autoregressive models [8] are widely applied on forecasting and detection. There are a lot of extensions and variations based on autoregressive models, including ARX [13], IMR [26], ARIMA [8], SARIMA [8], ARFIMA [14] and vector ARIMA [24]. On the base of AR model, ARX [13] further utilizes exogenous inputs to improve the performance. ARIMA [8] combines both the autoregressive process and the moving average process with integration. SARIMA [8] further considers the seasonal effect. ARFIMA [14] extends the integration order in ARIMA from integer to fraction, for a better

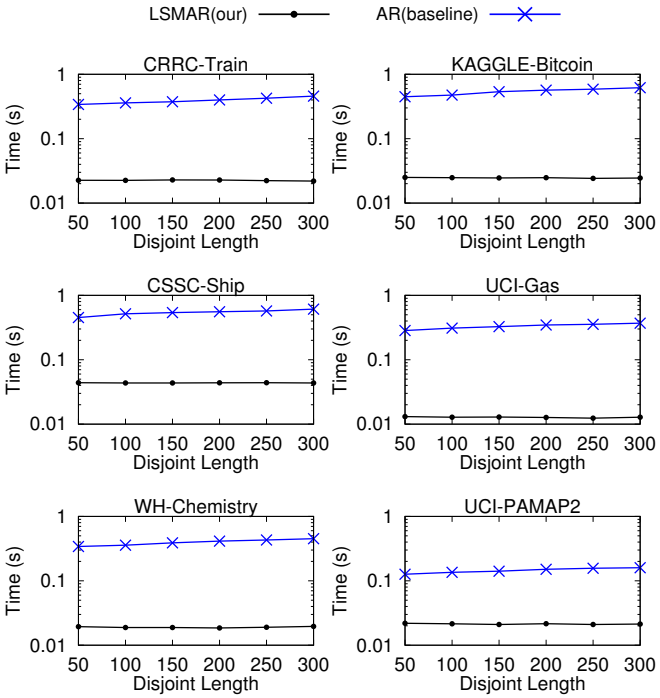


Figure 10: Time costs under different disjoint length

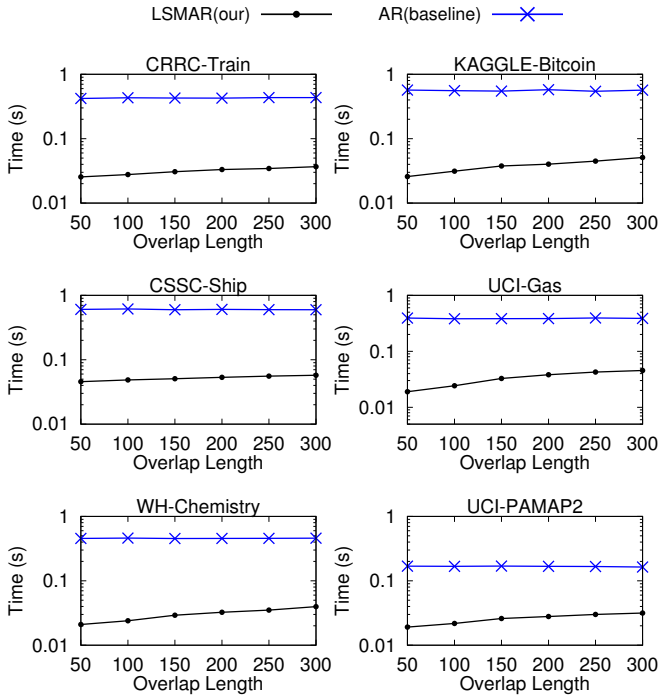


Figure 11: Time costs under different overlap length

forecasting performance on time series with long-range dependency. Vector ARIMA [24] can detect additive outlier, innovational outlier, level shift, and temporary change from multivariate time series. As a basic method in machine learning, AR is simply enough to learn when flushing batched data to disk, and we thus implement the autoregressive model in Apache IoTDB as a database-native machine learning operator. We do not implement ARX and IMR for evaluation, since they both require labeled data, difficult to maintain in the LSM-Tree based store.

The learned coefficients of autoregressive models can be utilized for a variety of downstream tasks, such as forecasting, detection and clustering. STIFF [19] utilizes autoregressive model for local model construction. Chakraborty et al. [9] consider the extracted real-world events on ARIMA for forecasting. Toledano et al. [23] utilize ARIMA for anomaly detection in their anomaly detection system, Anodot. Bagnall and Janacek [7] propose to cluster time series based on the coefficients learned from ARIMA.

7.2 LSM-Tree based Storage

The Log-Structured Merge-Tree (LSM-Tree) [20] can handle extensive writing workloads for time series, especially for IoT data. Thus it is often employed in the time series database, such as InfluxDB [5], Apache IoTDB [3]. Database-native machine learning operators on LSM-Tree based storage aim to accelerate the learning process by utilizing the properties of LSM-Tree. Absalyamov et al. [6] propose a novel lightweight approach for data synopses, including histograms and wavelets. LDI [22] learns the data distribution in LSM storage to improve insertion performance. However, machine learning operators for forecast and detection are rarely investigated in the previous studies, and our work thus focuses on the basic machine learning operator, i.e., autoregressive model, which can be applied both on forecast and detection.

8 CONCLUSION

In this paper, we propose an efficient method LSMAR for learning AR models in LSM-Tree based store. Data points are batched into different pages in such storage, and the later received points may overwrite the previously received points, owing to the out-of-order transmission, missing value imputation, data repairing and so on. Thereby, the straight-forward method is to online merge the scattered data and learn from scratch, while our LSMAR proposes to utilize the pre-computed information in the LSM-Tree based store. We derive several propositions to ensure the aggregation of pre-learned models in different pages. Remarkably, the algorithm for learning models on multi-segment time series has been deployed and become a function of Apache IoTDB, an LSM-Tree based time series database. We conduct extensive experiments in the system, where our proposal shows high efficiency in every evaluation of different aspects compared to the baseline.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (62232005, 62072265, 62021002), the National Key Research and Development Plan (2021YFB3300500), Beijing National Research Center for Information Science and Technology (BNR2022RC01011), and Alibaba Group through Alibaba Innovative Research (AIR) Program. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

REFERENCES

[1] [n. d.]. https://github.com/apache/iotdb/tree/research/autoregressive.
[2] [n. d.]. https://github.com/thssdb/LSM-AR-KDD23.
[3] [n. d.]. https://iotdb.apache.org/.
[4] [n. d.]. https://iotdb.apache.org/UserGuide/Master/Operators-Functions/Machine-Learning.html#ar.
[5] [n. d.]. https://www.influxdata.com/.
[6] Ildar Absalyamov, Michael J. Carey, and Vassilis J. Tsotras. 2018. Lightweight Cardinality Estimation in LSM-based Systems. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 841–855. https://doi.org/10.1145/3183713.3183761
[7] Anthony J. Bagnall and Gareth J. Janacek. 2004. Clustering time series from ARMA models with clipped data. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel (Eds.). ACM, 49–58. https://doi.org/10.1145/1014052.1014061
[8] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.
[9] Sunandan Chakraborty, Ashwin Venkataraman, Srikanth Jagabathula, and Lakshminarayanan Subramanian. 2016. Predicting Socio-Economic Indicators using News Events. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 1455–1464. https://doi.org/10.1145/2939672.2939817
[10] James Durbin. 1960. The fitting of time-series models. *Revue de l'Institut International de Statistique* (1960), 233–244.
[11] Chenguang Fang, Shaoxu Song, and Yinan Mei. 2022. On Repairing Timestamps for Regular Interval Time Series. *Proc. VLDB Endow.* 15, 9 (2022), 1848–1860. https://www.vldb.org/pvldb/vol15/p1848-song.pdf
[12] Chenguang Fang, Shaoxu Song, Yinan Mei, Ye Yuan, and Jianmin Wang. 2022. On Aligning Tuples for Regression. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, Aidong Zhang and Huzefa Rangwala (Eds.). ACM, 336–346. https://doi.org/10.1145/3534678.3539373
[13] Graham Clifford Goodwin. 1977. Dynamic system identification: experiment design and data analysis. *Mathematics in science and engineering* 136 (1977).
[14] Clive WJ Granger and Roselyne Joyeux. 1980. An introduction to long-memory time series models and fractional differencing. *Journal of time series analysis* 1, 1 (1980), 15–29.
[15] Ulf Grenander and Murray Rosenblatt. 2008. *Statistical analysis of stationary time series*. Vol. 320. American Mathematical Soc.
[16] Jingxuan He and Xixian Han. 2022. Efficient Skyline Computation on Massive Incomplete Data. *Data Sci. Eng.* 7, 2 (2022), 102–119. https://doi.org/10.1007/s41019-022-00183-7
[17] David J Hill and Barbara S Minsker. 2010. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environmental Modelling & Software* 25, 9 (2010), 1014–1022.
[18] Yuyuan Kang, Xiangdong Huang, Shaoxu Song, Lingzhe Zhang, Jialin Qiao, Chen Wang, Jianmin Wang, and Julian Feinauer. 2022. Separation or Not: On Handing Out-of-Order Time-Series Data in Leveled LSM-Tree. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 3340–3352. https://doi.org/10.1109/ICDE53745.2022.00315
[19] Zhigang Li, Margaret H. Dunham, and Yongqiao Xiao. 2002. STIFF: A Forecasting Framework for SpatioTemporal Data. In *Mining Multimedia and Complex Data, KDD Workshop MDM/KDD 2002, PAKDD Workshop KDMCD 2002, Revised Papers (Lecture Notes in Computer Science, Vol. 2797)*, Osmar R. Zaïane, Simeon J. Simoff, and Chabane Djeraba (Eds.). Springer, 183–198. https://doi.org/10.1007/978-3-540-39666-6_12
[20] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth J. O'Neil. 1996. The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica* 33, 4 (1996), 351–385. https://doi.org/10.1007/s002360050048
[21] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. 2021. CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 24804–24816. https://proceedings.neurips.cc/paper/2021/hash/cfe8504bda37b575c70ee1a8276f3486-Abstract.html
[22] Dai Hai Ton That, Mohammadsaleh Gharehdaghi, Alexander Rasin, and Tanu Malik. 2021. LDI: Learned Distribution Index for Column Stores. In *2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, December 15-18, 2021*, Yixin Chen, Heiko Ludwig, Yicheng Tu, Usama M. Fayyad, Xingquan Zhu, Xiaohua Hu, Suren Byna, Xiong Liu, Jianping Zhang, Shirui Pan, Vagelis Papalexakis, Jianwu Wang, Alfredo Cuzzocrea, and Carlos Ordonez (Eds.). IEEE, 376–387. https://doi.org/10.1109/BigData52589.2021.9671318
[23] Meir Toledano, Ira Cohen, Yonatan Ben-Simhon, and Inbal Tadeski. 2017. Real-time anomaly detection system for time series at scale. In *Proceedings of the KDD 2017 Workshop on Anomaly Detection in Finance, ADF@KDD 2017, Halifax, Nova Scotia, Canada, August 14, 2017 (Proceedings of Machine Learning Research, Vol. 71)*, Archana Anandakrishnan, Senthil Kumar, Alexander R. Statnikov, Tanveer A. Faruquie, and Di Xu (Eds.). PMLR, 56–65. http://proceedings.mlr.press/v71/toledano18a.html
[24] Ruey S Tsay, Daniel Pena, and Alan E Pankratz. 2000. Outliers in multivariate time series. *Biometrika* 87, 4 (2000), 789–804.
[25] Xinfeng Wang, Xiang Zhou, Jiahua Rao, Zhu-Jin Zhang, and Yue-Dong Yang. 2022. Imputing DNA Methylation by Transferred Learning Based Neural Network. *J. Comput. Sci. Technol.* 37, 2 (2022), 320–329. https://doi.org/10.1007/s11390-021-1174-6
[26] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. 2017. Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing. *Proc. VLDB Endow.* 10, 10 (2017), 1046–1057. https://doi.org/10.14778/3115404.3115410

A APPENDIX

A.1 Proof of Proposition 3.2

Proof. According to Formula (1), if the value x_h of point h is modified to x_h, only two terms will change, i.e., $\frac{1}{n-i}x_h x_{h-i}$ and $\frac{1}{n-i}x_h x_{h+i}$. Thus, the updated auto-covariances have value

$$\begin{aligned}\gamma_i' &= \gamma_i - \frac{1}{n-i}(x_h x_{h-i} + x_h x_{h+i}) + \frac{1}{n-i}(x_h' x_{h-i} + x_h' x_{h+i}) \\ &= \gamma_i + \frac{1}{n-i}(x_h' - x_h)(x_{h-i} + x_{h+i}).\end{aligned}$$

□

A.2 Proof of Proposition 4.1

Proof. The aggregated auto-covariances of $\mathbf{x}[1:n]$ have the form

$$\gamma_i = \frac{1}{n-i}\sum_{l=1}^{n-i} x_l x_{l+i}.$$

It can be further split into

$$\gamma_i = \frac{\sum_{l=1}^{h-i} x_l x_{l+i}}{n-i} + \frac{\sum_{l=h-i+1}^{h} x_l x_{l+i}}{n-i} + \frac{\sum_{l=h+1}^{n} x_l x_{l+i}}{n-i}. \tag{3}$$

Note that the auto-covariances of $\mathbf{x}[1:h]$ and $\mathbf{x}[h+1:n]$ have the following form, respectively,

$$\gamma_i^{(1)} = \frac{1}{h-i}\sum_{l=1}^{h-i} x_l x_{l+i}, \tag{4}$$

$$\gamma_i^{(2)} = \frac{1}{n-h-i}\sum_{l=h+1}^{n} x_l x_{l+i}. \tag{5}$$

By combining (3), (4) and (5), the proposition holds. □

A.3 Proof of Proposition 4.3

Proof. Given d and Δx, the interpolated series is $\{x_h + \frac{\Delta x}{d}, x_h + 2\frac{\Delta x}{d}, \ldots x_h + (d-1)\frac{\Delta x}{d}\}$, the auto-covariances of the interpolation series could be calculated by

$$\begin{aligned}\gamma_i' &= \frac{1}{d-i-1}\sum_{l=1}^{d-i-1}\left(x_h + l\frac{\Delta x}{d}\right)\left[x_h + (l+i)\frac{\Delta x}{d}\right] \\ &= \frac{1}{d-i-1}\sum_{l=1}^{d-i-1}\left[x_h^2 + (2l+i)x_h\frac{\Delta x}{d} + (l+i)l\left(\frac{\Delta x}{d}\right)^2\right] \\ &= x_h x_g + \frac{(d-i)i}{2}\left(\frac{\Delta x}{d}\right)^2 + \frac{(d-i)(2d-2i-1)}{6}\left(\frac{\Delta x}{d}\right)^2.\end{aligned}$$

The aggregated auto-covariances is equal to the aggregated covariances of $\mathbf{x}[1:h]$, the interpolated series and $\mathbf{x}[g:n]$. Similar to Proposition 4.1, we obtain the aggregated auto-covariances. □

A.4 Proof of Proposition 4.5

Proof. Note that $d-i-1 \le 0$ suggests that the interpolation series (as shown in the proof of Proposition 4.3) has no more than i points, that is, the auto-covariances of the interpolation series is 0. To obtain the aggregated auto-covariances, we first aggregate $\mathbf{x}[1:h]$ and the interpolated series into $\{x_1, x_2, \ldots, x_h, x_h + \frac{\Delta x}{d}, x_h + 2\frac{\Delta x}{d}, \ldots, x_h + (d-1)\frac{\Delta x}{d}\}$. And then aggregate the following series and $\mathbf{x}[g:n]$ referring to Proposition 4.1. □

A.5 Proof of Proposition 4.7

Proof. We denote the points in the low-version segment as $\mathbf{x}^{(1)}$ and the points in the high-version segment as $\mathbf{x}^{(2)}$. We denote the merged segment as $\mathbf{x}''$, which has the following form,

$$x_l'' = \begin{cases} x_l^{(2)}, & \text{if} \quad x_l^{(2)} \in \mathbf{x}^{(2)}[g:n] \\ x_l^{(1)}, & \text{otherwise} \end{cases}.$$

Thus the merged auto-covariances can be represented by

$$\gamma_i = \frac{1}{n-i}\sum_{l=1}^{n-i} x_l'' x_{l+i}''.$$

Since u is the earliest point in $[g:n]$ with $u > h$, the points from u to n in $\mathbf{x}''$ must be the same as those in $\mathbf{x}^{(2)}$, and we thus have

$$\begin{aligned}(n-i)\gamma_i &= \sum_{l=1}^{h-i} x_l'' x_{l+i}'' + \sum_{l=h-i+1}^{u-1} x_l'' x_{l+i}'' + \sum_{l=u}^{n-i} x_l'' x_{l+i}'' \\ &= \sum_{l=1}^{h-i} x_l'' x_{l+i}'' + \sum_{l=h-i+1}^{u-1} x_l'' x_{l+i}'' + \sum_{l=u}^{n-i} x_l^{(2)} x_{l+i}^{(2)} \\ &= (h-i)\gamma_i^{(1)'} + \sum_{l=h-i+1}^{u-1} x_l'' x_{l+i}'' + (n-u-i+1)\gamma_i^{(2)'}, \end{aligned} \tag{6}$$

where $\gamma_i^{(1)'}$ in (6) is the auto-covariances of the segment $\mathbf{x}''[1:h]$, i.e., the updated segment $\mathbf{x}'[1:h]$, and $\gamma_i^{(2)'}$ is the auto-covariances of the segment $\mathbf{x}^{(2)}[u:n]$, i.e., the non-updated segment $\mathbf{x}[u:n]$.

Note that if $u = h+1$, the middle term in (6) has the form

$$\sum_{l=h-i+1}^{u-1} x_l'' x_{l+i}'' = \sum_{l=h-i+1}^{h} x_l'' x_{l+i}'' = \sum_{l=1}^{i} x_{h+l} x_{h+l-i},$$

which is the same as Proposition 4.1. In such case, the aggregated auto-covariances γ_i is thus equal to the aggregated auto-covariances of adjacent segments $\mathbf{x}'[1:h]$ and $\mathbf{x}[u:n]$.

On the other hand, if $u > h+1$, there are missing values from h to u. In such case, the middle term in (6) can be further split into two or three parts, the same as Propositions 4.3 or 4.5. Following the similar method of the proofs for Propositions 4.3 and 4.5, the aggregated auto-covariances γ_i is also equal to the aggregated auto-covariances of disjoint segments $\mathbf{x}'[1:h]$ and $\mathbf{x}[u:n]$.

In summary, the aggregated auto-covariances γ_i is equal to the aggregated auto-covariances of non-overlapped segments $\mathbf{x}'[1:h]$ and $\mathbf{x}[u:n]$, regardless of whether there are missing values between u and h. □

A.6 Proof of Proposition 4.8

Proof. Following the definition of auto-covariances, we have

$$(n-g+1)\gamma_i = \sum_{l=g}^{n-i} x_l x_{l+i} = \sum_{l=g}^{u-1} x_l x_{l+i} + \sum_{l=u}^{v-i} x_l x_{l+i} + \sum_{l=v-i+1}^{n-i} x_l x_{l+i}.$$

The auto-covariances of the segment $\mathbf{x}[u:v]$ is

$$\gamma_i' = \frac{1}{v-u+1}\sum_{l=u}^{v-i} x_l x_{l+i}.$$

By combining the equations above, the proposition holds. □