Matrix Profile Index Approximation for Streaming Time Series

Maryam Shahcheraghi[1], Trevor Cappon[1], Samet Oymak[2], Evangelos Papalexakis[1],
Eamonn Keogh[1], Zachary Zimmerman[1], and Philip Brisk[1]
[1]*dept. of Computer Science and Engineering*
[2]*dept. of Electrical and Computer Engineering*
University of California Riverside
{sshah073, tcapp001, zzimm001}ucr.edu, oymak@ece.ucr.edu, {epapalex, eamonn, philip}@cs.ucr.edu

***Abstract*—Discovery of motifs (repeated patterns) in time series is a key factor across numerous industries and scientific fields. These and related problems have effectively been solved for offline analysis of time series; however, these approaches are computationally intensive and do not lend themselves to streaming time series, where the sampling rate imposes real-time constraints on computation and there is strong desire to locate computation as close as possible to the sensor. One promising solution is to use low-cost machine learning models to provide approximate answers to these problems. For example, prior work has trained models to predict the similarity of the most recently sampled window of data points to a representative time series used for training. This work addresses a more challenging problem: to predict not only the "strength" of the match, but also the relative location in the representative time series where the match occurs. We evaluate our approach on two different real world datasets; we demonstrate speedups as high as 40× compared to exact computations, with predictive accuracy as high as 87.9%, depending on the granularity of the prediction.**

***Index Terms*—Streaming time series, Matrix Profile, Indexing**

I. Introduction

This paper describes a machine learning system that predicts similarities between a streaming time series and a representative time series used to train the model. Specifically, the system predicts the regions in the representative time series that closely match the window containing the most recently sampled data points from the streaming time series. Prior work has computed this information exactly at great cost [2], [15], [17], [18], [20], [21], and cannot meet the real-time constraints on computation that are needed to process streaming data. One existing technique can predict similarities to a representative time series, but without predicting *where* these similarities occur [1]. For example, this method predicts if a recently sampled window of seismograph readings matches *something* in the historical record; whereas, the system presented in this paper predicts whether or not a match occurs, *and* what region or regions within in the historical record are likely to contain the match.

This paper solves the index prediction problem using a tree of machine learning models of varying granularity. Using seismic activity prediction as an example, suppose that our window contains the most recent minute of sampled datapoints. An exact search would compute the similarity of this window to each minute in the historical record. Assuming that we limit the historical record to a year's worth of data, the tree would first predict the month(s) during which similar subsequences occur; then, within each predicted month, it would then predict the weeks, days, hours, and finally minutes, leveraging the tree structure to limit the search to the most promising regions of the historical record. This approach readily generalizes to other domains.

This paper makes the following contributions:

- We introduce a novel machine learning system that predicts if and when newly-observed time series patterns have occurred in the time series used for training.
- We introduce a retraining mechanism which significantly improves the accuracy of our system.
- We show how to extend our system to support incremental updates and online retraining in the face of new data.
- We report predictive accuracy up to 87.9% and speedups as high as 40× compared to computing the similarities exactly.

The paper is organized as follows: Section II presents definitions and background context that form the basis for the technical contribution. Section III summarizes related work on time series and data structures that influenced our work. Section IV describes training and inference using our system in detail. Section V presents the empirical evaluation of our system. Section VI concludes the paper and outlines directions for future work.

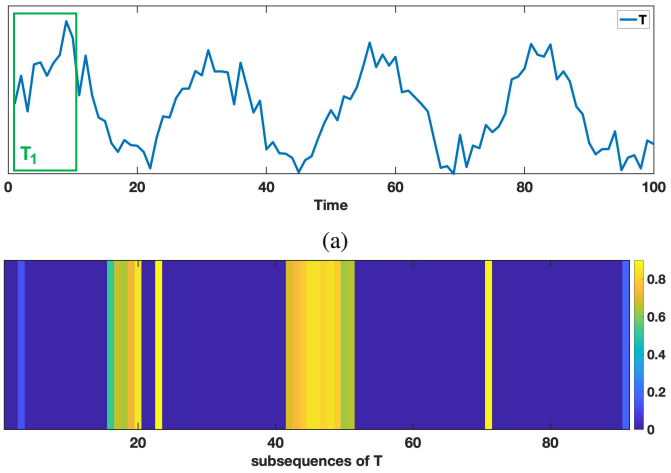
II. Background material

In this section, a summary of the required background and definitions of commonly-used terms are introduced.

Time series: A *Time series* $T = \langle t_1, t_2, \ldots, t_n \rangle$ is an ordered sequence of n scalar data points.

Subsequence: $T_{i,m} = \langle t_i, t_{i+1}, \ldots, t_{i+m-1} \rangle$ is the *subsequence* of T starting at position i and having length $m \ll n$; if m is obvious, we can write T_i in place of $T_{i,m}$. We write $T_i \prec T$ to denote that T_i is a proper subsequence of T'.

Correlation Profile: Let $c_{i,j}$ be the Pearson correlation between subsequences $T_i, T_j \prec T$ [17]. The vector $C_i = \langle c_{i,j} \mid 1 \leq j \leq n - m + 1 \rangle$ is T_i's *Correlation Profile*. Figure 1(b) shows the Correlation Profile of subsequence $T_{1,10}$ of a synthetic time series T ($n = 100$).



(a)

(b)

Fig. 1: (a) A synthetic time series T with subsequence $T_{1,10}$ identified; and (b) the correlation profile C_1.

Exclusion Zone: Any subsequence matches itself perfectly. By convention, an *exclusion zone*, defined by an integral parameter $\delta \geq 0$, forces $c_{i,j}$ to 0 if $|i-j| \leq \delta$. The exclusion zone is implicit in all subsequent definitions.

The maximum Pearson correlation value in C_i is defined as

$$c_i^{max} = \max_{1 \leq j \leq n-m+1} \{c_{i,j}\}. \quad (1)$$

Nearest Neighbor: The *Nearest Neighbor* of T_i, denoted $\eta(T_i)$, is any subsequence $T_j \prec T$ satisfying $c_{i,j} = c_i^{max}$.

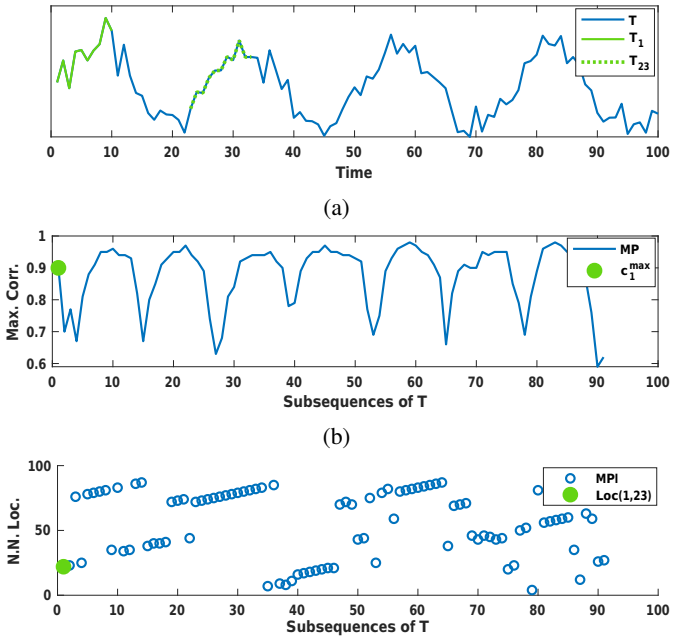
The Matrix Profile (see Figure 2(b)), introduced below, is expensive to compute, but renders many important time series data analysis trivial, including *motif* discovery, classification, clustering, anomaly detection, etc. The Matrix Profile Index and Self-Join, which associate each subsequence with its nearest neighbor, are naturally computed as side products. These definitions emphasize the similarity (or lack thereof) among subsequences within a single time series.

Matrix Profile (MP): The *Matrix Profile* is a vector that contains the correlations of the nearest neighbors of each subsequence in T: $P(T) = \langle c_i^{max} |\ 1 \leq i \leq n-m+1 \rangle$; the MP is itself a time series [15].

Matrix Profile Index (MPI): The *Matrix Profile Index* is a vector containing the index of each nearest neighbor: $I(T) = \langle j \mid T_j = \eta(T_i), 1 \leq i \leq n-m+1 \rangle$. Figure 2 shows the Matrix Profile and its Index for a time series that contains the index of each subsequence's nearest neighbor.

We do not use the Matrix Profile Index directly in this paper; however, we do use machine learning to predict general regions of a time series where the nearest neighbor of a subsequence is likely to occur. Moreover, we often care not so much about one specific nearest neighbor of a subsequence, but any neighbor with a sufficiently high correlation.

k Nearest Neighbors (kNN): Let $c_i^{(j)}$ denote the rank-j Pearson correlation value in C_i. The k *Nearest Neighbors* of T_i are collected into a vector of subsequences, denoted $\eta^{[k]}(T) = \langle T_l \prec T_B |\ c_{i,l} = c_i^{(j)}, 1 \leq j \leq k \rangle$.



(a)

(b)

(c)

Fig. 2: (a) A synthetic time series T with length-10 subsequence T_1 and its nearest neighbor T_{23} identified; (b) Matrix Profile (MP) and (c) Matrix Profile Index (MPI).

kNN Correlation Profile: The *kNN Correlation Profile* $C_i^{[k]} = \langle c_i^{(j)} |\ 1 \leq j \leq k \rangle$ is the vector of the k highest-ranked Pearson correlation values in C^i.

kNN Matrix Profile (kNN MP): The *kNN Matrix Profile* is an $(n-m+1) \times k$ matrix that contains the correlations of the k Nearest Neighbors of each subsequence in T, denoted $P^{[k]}(T) = \langle C_i^{[k]} |\ 1 \leq i \leq n-m+1 \rangle$.

The preceding definitions are limited to correlations of equal-length subsequences within a single time series; they naturally generalize to correlations between equal-length subsequences occuring within two distinct time series.

AB-Correlation Profile: Let T_A and T_B be distinct time series having lengths n_A and n_B, respectively. Let $T_i \prec T_A$ and $T_j \prec T_B$ be length-m subsequences whose Pearson correlation is $c_{i,j}^{AB}$. The *AB-Correlation Profile* for T_i is the vector $C_i^{AB} = \langle c_{i,j}^{AB} \mid 1 \leq j \leq n_B - m + 1 \rangle$. The maximum Pearson correlation in C_i^{AB} is

$$c_i^{AB,max} = \max_{1 \leq j \leq n_B-m+1} \{c_{i,j}^{AB}\}. \quad (2)$$

AB-Nearest Neighbor: The *AB-Nearest Neighbor* of $T_i \prec T_A$ in T_B, denoted $\eta_{AB}(T_i)$, is any subsequence $T_j \prec T_B$ satisfying $c_{i,j}^{AB} = c_i^{AB,max}$.

AB-Matrix Profile (MP): The *AB-Matrix Profile* is a vector that contains the AB-nearest neighbors of each subsequence in T_A: $P_{AB}(T_A, T_B) = \langle c_i^{AB,max} |\ 1 \leq i \leq n_B - m + 1 \rangle$.

AB-kNN Correlation Profile: Let $c_i^{AB(j)}$ be the rank-j Pearson correlation in C_i^{AB}. The *AB-kNN Correlation Profile*

is the vector $C_i^{AB[k]} = \langle c_i^{AB(j)} |\ 1 \leq j \leq k \rangle$, which contains the k highest-ranked Pearson correlation in C_i^{AB}.

AB-k Nearest Neighbors (AB-kNN): The k *Nearest Neighbors* of $T_i \prec T_A$ in T_B are a vector of subsequences $\eta_{AB}^{[k]}(T_i) = \langle T_l \prec T_B |\ c_{i,l}^{AB} = c_i^{AB(j)}, 1 \leq j \leq k \rangle$.

AB-kNN Matrix Profile (AB-kNN MP): The *kNN Matrix Profile* is an $(n_A - m + 1) \times k$ matrix that contains the correlations of the AB-k Nearest Neighbors: $P_{AB}^{[k]}(T_A, T_B) = \langle C_i^{AB[k]} |\ 1 \leq i \leq n_B - m + 1 \rangle$.

Computing and updating the Matrix Profile and/or any of its various is impractical for streaming time series. Doing so would entail running $O(n^2)$ algorithms, with $n \to \infty$.

Learned Approximate Matrix Profile (LAMP): A *Learned Approximate Matrix Profile (LAMP)* is a *learned* function $L : T \to [0, 1)$ that predicts the Matrix Profile of a time series T. A LAMP model is trained on a labeled dataset $\Gamma = (T_R, P(T_R))$, where a *representative* time series T_R comprises the features and the target is its Matrix Profile $P(T_R)$ [1].

The LAMP concept is independent of the choice of machine learning model; the LAMP paper employed a *1-dimensional convolutional neural network (1D-CNN)* with *residual* connections; this paper adopts the same model. In the streaming context, LAMP predictions (1D-CNN inference) run orders of magnitude faster than computing Matrix Profiles or AB-joins directly. This makes LAMP a suitable option for real-time applications deployed on embedded systems.

III. Related Work

Time series analysis is generally concerned with techniques such as classification, clustering, motif discovery, and anomaly detection [14]–[16], [22]. Among the techniques presented in the preceding section, the Matrix Profile and its extensions can solve these problems offline once the time series has been fully collected [2], [15], [17], [18], [20], [21], while LAMP can solve them online through prediction [1].

The work presented here exhibits similarities to time series indexing methods such as iSAX [6], iSAX 2.0 [7], iSAX2+ [8] and ADS+ [9], among others. Our technique constructs a tree of LAMP models of varying granularity, which is similar to TARDIS [12], which is a tree-structured indexing method built upon iSAX. All these methods index a time series, not its Matrix Profile.

Our work is also influenced by a learned indexing methods for database structures, such as B-Trees [23]. This paper claims that all indices can be learned and predicted, while focusing on the location of keys within a specific range of an in-memory data set. Our system applies a similar approach to streaming time series data, and serves as an additional use case that adds credence to their claim.

The TSR and BTSR-Trees [11] extend the R-Tree to index geolocated time series. The R-Tree [25] is a data structure that indexes multi-dimensional information, such as geographical coordinates, with each node pointing to the disk page that contains it; geo-iSAX [13] uses the BTSR-tree for supports visual exploration and analysis of geolocated time series. These tools represent potential directions for future work, including the specification of a Matrix Profile, LAMP, etc. tailored to geolocated time series.

IV. Matrix Profile Index Approximation

Matrix Profile Index Approximation aims to find general regions of the nearest neighbors of a relatively short time series segment within a longer time series. In the context of LAMP, the shorter time series is the recently sampled window of data points from a streaming time series, and the longer time series is the representative time series T_R, used to train the LAMP model. Our solution employs a *LAMP-Tree*: a tree of LAMP models of varying granularity, which we use to predict approximate nearest neighbor indices. Figure 3 shows a generic example which has been trained on a representative time series T_R containing 48 weeks of data.

In Figure 3, the root node (w_{1-48}) contains a LAMP model that encompasses 48 weeks of T_R; the internal (non-leaf) nodes, (w_{1-4}, etc.) encompass 4-week intervals; and the leaf nodes (w_1 ... w_{48}) encompass one-week intervals.

Leaf node w_i represents the i^{th} week; internal node w_{i-j} represents weeks i through j; the corresponding subsequences of T_R are T_{w_i} and $T_{w_{i-j}}$. It is important not to conflate T_{w_i} with T_i, the subsequence starting at the i^{th} datapoint in T_R, not the i^{th} week. L_{w_i} and $L_{w_{i-j}}$ are the LAMP models associated with w_i and w_{i-j}.

A. LAMP-Tree Queries (Inference)

LAMP-Trees predict which regions ($T_{w_1} ... T_{w_N}$) of the representative time series (T_R) likely contain a match to the input. The input is a time series segment T_Q called a *query*, which represents the m most recently-sampled datapoints of a streaming time series, and a threshold value $\theta \in [0, 1)$.

The next two definitions are provided here for context but are solely used when evaluating LAMP-Trees in Section V. The predictive querying (inference) procedure for LAMP-Trees is defined afterwards.

Exact LAMP-Tree Query: An *Exact LAMP-Tree Query* returns the set of leaves whose time series contain at least one subsequence whose Pearson correlation to T_Q is at least θ, i.e., the set $Q_{Exact} = \{w_i |\ P_{AB}(T_{w_i}, T_Q) \geq \theta\}$. Since subsequence length is $|T_Q|$, the AB-Correlation Profile is (one-dimensional), i.e., identical to the AB-Matrix Profile.

Exact kNN LAMP-Tree Query: An *Exact kNN LAMP-Tree Query* returns the set of leaves whose time series contain at least k subsequences whose Pearson correlations with T_Q are at least θ, i.e., the set $Q_{Exact}^{[k]} = \{w_i |\ c_{w_i}^{AB(k)} \geq \theta\}$, where $c_{w_i}^{AB(j)}$ denotes the rank-j Pearson correlation value returned by the AB-kNN Matrix profile $P_{AB}^{[k]}(T_{w_i}, T_Q)$.

Predicted LAMP-Tree Query: Q_{Exact} and $Q_{Exact}^{[k]}$ are expensive to compute. As an alternative, a *Predicted LAMP-Tree Query* returns the set of leaves whose LAMP models predict Pearson correlations for T_Q that are at least θ, i.e., the set $Q_{LAMP} = \{w_i |\ L_{w_i}(T_Q) \geq \theta\}$.

Match: A *match* occurs at leaf node w_i if $L_{w_i}(T_Q) \geq \theta$, or at internal node w_{i-j} if $L_{w_{i-j}}(T_Q) \geq \theta$.

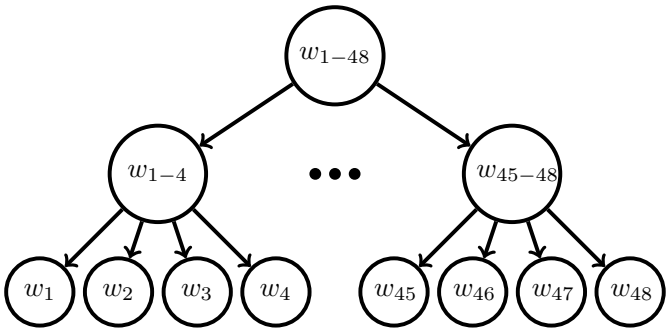


Fig. 3: A representative LAMP-Tree trained on 48 week's of data: internal nodes represent 4-week intervals and leaf nodes represent individual weeks.

Let w_j be a leaf where $P_{AB}(T_{w_j}, T_Q) \geq \theta$; then there exists a subsequence $T_l \prec T_{w_j}$ whose Pearson correlation with T_Q exceeds θ. Internal node w_{i-k} is an ancestor of w_j if $i \leq j \leq k$; then $T_l \prec T_{w_j} \prec T_{w_{i-k}}$ implies that $P_{AB}(T_{w_{i-k}}, T_Q) \geq \theta$.

Accuracy Assumption: The LAMP Tree *accuracy assumption* is satisfied if the following holds for each leaf node w_j: $L_{w_j}(T_Q) \geq \theta \implies L_{w_{i-k}}(T_Q) \geq \theta, 1 \leq j \leq k$.

Starting at the root, the search descends in depth- or breadth-first order. At each internal node, the search continues if a match occurs; otherwise, the search is pruned. Each leaf node where a match occurs is added to Q_{LAMP}.

Lemma: A Predicted LAMP-Tree Query correctly computes Q_{LAMP} if the accuracy assumption is satisfied.

Figure 4 shows an example. A match occurs at the root, w_{1-48}. The search proceeds to each of the root's children in succession : $w_{1-4}, \ldots, w_{13-16}, \ldots, w_{45-48}$ (other subtrees are not shown for brevity). No match occurs at w_{1-4} so the search is pruned there; a match occurs at w_{13-16}, so the search continues to its children: w_{14} and w_{16} match and are added to Q_{LAMP}. No matches occur at w_{45-48}.

B. LAMP-Tree Training

Without loss of generality, we describe how to train the 48-week LAMP-Tree in Figure 3 on representative time series T_R. The model developer specifies the arity of each level of the tree. Figure 3 has an arity of 12 at the root and 4 at internal nodes. We introduce overlap between consecutive training segments $T_{w_i}, T_{w_{i+1}}$ to ensure that no correlations are lost at subsequence boundaries. Training proceeds bottom-up, starting at the leaves. Let w_i be a leaf node. We construct a LAMP model L_{w_i} trained on dataset $\Gamma_{w_i} = (T_{w_i}, P(T_{w_i}))$. A *Merge Operation* (defined below) creates LAMP models at internal nodes.

LAMP Profile: Let $L^{[k]} = \{L_1, L_2, \ldots L_k\}$ be a set of k LAMP models. The *LAMP Profile* of time series T applies each LAMP model to each subsequence $T_i \prec T$ (recall: n is the length of T and $m \ll n$ is the length of T_i), and collects the maximum predicted correlations in a vector:

$$P_{LAMP}(T, L^{[k]}) = \langle \max_{1 \leq j \leq k} \{L_j(T_i)\} |\ 1 \leq i \leq n-m+1 \rangle \quad (3)$$

Merge Operation: Consider an internal node w_{i-k} whose children are leaves $w_i, w_{i+1}, \ldots w_k$ with LAMP models $L_{w_i}, L_{w_{i+1}}, \ldots L_{w_k}$; we collect the LAMP models into a vector $L_{w_{i-k}}^{[k-i+1]}$ and construct a new training dataset $\Gamma_{w_{i-k}} = (T_{w_{i-k}}, P_{LAMP}(T_{w_{i-k}}, L_{w_{i-k}}^{[k-i+1]}))$. We then construct a new LAMP model $L_{w_{i-k}}$ for w_{i-k} that is trained on $\Gamma_{w_{i-k}}$. The merge operation proceeds analogously if the children are internal nodes rather than leaves.

C. LAMP-Tree Retraining

The LAMP-Tree training algorithm described above suffers from one key drawback: LAMP models for nodes further from the root are trained on shorter subsequences, and tend to be more susceptible to misprediction as a result. We have observed inconsistencies where matches predicted at internal nodes disappear at the leaves. Our solution is to retrain the leaves with additional training data prior to deployment. The user specifies a parameter p, which throttles the amount of additional training data provided to each leaf node; recalling that N is the number of leaves in the LAMP-Tree, each leaf node is retrained with up to $\lceil pN \rceil$ additional subsequences, in order to yield a more robust LAMP model.

Figure 5(a) depicts the training process for the first four leaves of the LAMP-Tree shown in Figure 3. The matrix indicates that the LAMP model L_{w_i} for each leaf node $w_i, 1 \leq i \leq 4$, was trained on $\Gamma_{w_i} = \{(T_{w_i}, P(T_{w_i})\}$, as described in the preceding section. To illustrate the problem, consider subsequence T_{w_2}; toward the right-side of T_{w_2}, there is a tall peak, followed by a smaller double-peak. In the context of $P(T_{w_2})$, this tall peak is an anomaly; however, there is a similar tall peak in T_{w_3}, which suggests that it might be a motif in representative time series T_R. If the query T_Q contains a similar tall peak, the LAMP model $L_{w_{1-48}}$ at the root should predict a match if it is accurate; however the LAMP models L_{w_2} and L_{w_3} at the leaves might not, because those models view the peak as an anomaly. Our proposed solution is to ***retrain*** LAMP model L_{w_2} using the AB-Matrix Profile $P_{AB}(T_{w_2}, T_{w_3})$. To do this, we construct a new training data set $\Gamma_{w_2,w_3} = (T_{w_2}, P_{AB}(T_{w_2}, T_{w_3}))$ and use it to retrain LAMP model L_{w_2}.

To automate this process, we consider time series subsequence T_{w_i} for $1 \leq i \leq N$. We randomly select $min\{i-1, \lceil pN \rceil\}$ leaf nodes in the range $\{w_1, w_2, \ldots, w_{i-1}\}$ to be retrained using T_{w_i}. For each selected leaf node w_j, we construct a retraining dataset $\Gamma_{w_j,w_i} = (T_{w_j}, P_{AB}(T_{w_j}, T_{w_i}))$ which we use to retrain LAMP model L_{w_j} with AB-Matrix Profile $P_{AB}(T_{w_j}, T_{w_i})$ as the new target. Once L_{w_j} has been retrained, $P_{AB}(T_{w_j}, T_{w_i})$ can be discarded; alternatively, the retraining datasets for all LAMP models can be generated upfront, followed by batch retraining.

The process for retraining internal nodes is identical to what was described in the preceding section. The only difference is that the LAMP Profile is constructed from LAMP models at the leaves that have been retrained.

Fairness: We use a weighted random selection function to select which leaf nodes to retrain. Let r_i denote the number

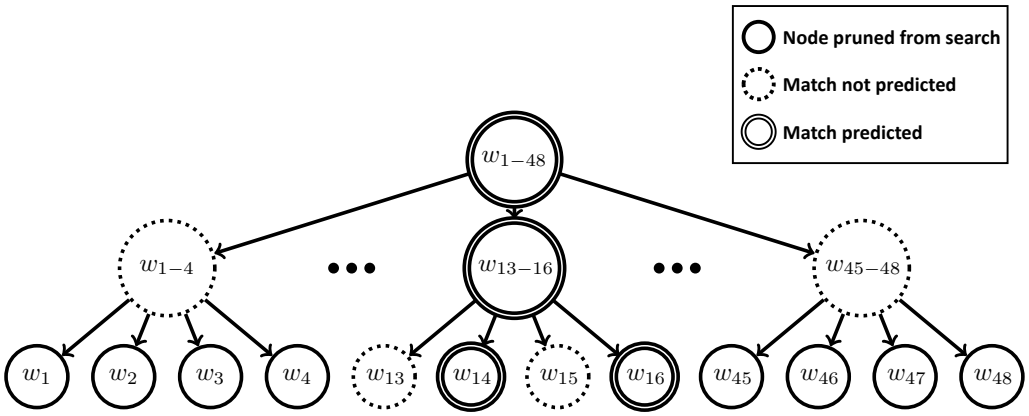


Fig. 4: Querying the LAMP tree is a top-down process: when an internal node predicts a match, the inference process visits each of its children, eventually stopping at leaves; otherwise the search is pruned.

of times LAMP model L_{w_i} is retrained. The selection weight associated with w_i is set to $\frac{1}{r_i+1}$; r_i is incremented each time w_i is selected for retraining. This ensures that the probability that w_i is subsequently selected for retraining is reduced, but never fully eliminated, which ensures *fairness*.

Diversity: Let w_j be a leaf node selected to have its LAMP model L_{w_j} retrained using AB-Matrix Profile $P_{AB}(T_{w_j}, T_{w_i})$. Retraining could alter "facts" that the LAMP model sans retraining would have learned differently. For example, the retrained LAMP model could learn that an anomaly within T_{w_j} is actually a motif due to a similar subsequence occurring in T_{w_i}; this provides *diversity*.

Symmetric Retraining: Under retraining, as described above, leaf nodes with smaller indices are more likely to be selected than leaf nodes with larger indices. To offset this bias, we propose *symmetric retraining*. Assume that we are processing subsequence T_{w_i} and leaf node w_j is selected for retraining. Under the prior scheme, we retrain LAMP model L_{w_j} using data set $\Gamma_{w_j,w_i} = (T_{w_j}, P_{AB}(T_{w_j}, T_{w_i}))$; under symmetric retraining, we *also* retrain LAMP model L_{w_i} using training data set $\Gamma_{w_i,w_j} = (T_{w_i}, P_{AB}(T_{w_i}, T_{w_j}))$. This reduces the aforementioned bias.

V. Experimental Results

Experimental Setup: We used an Intel Core i9-9900 CPU running at 3.1 GHz with 32 GB RAM and running Ubuntu 18.04.4 LTS. We trained the 1D-CNNs employed in our LAMP models and computed Matrix Profiles (for comparative purposes) using the SCAMP algorithm [17] on an Nvidia TU102 GPU.

Datasets: Table I summarizes two publicly available datasets that we used for evaluation: *Seismic:* a dataset collected near Parkfield, CA in 2004 that contains several catalogued earthquakes [26]; and *Chicken:* a dataset obtained by attaching an accelerometer to a chicken and tracking its movements for one day [24].

LAMP-Trees: We trained five LAMP-Trees for the Seismic dataset and four for the Chicken dataset. We use naming convention **aL-b** for each tree, where **a** is the number of levels and **b** is the number of datapoints per leaf. For example, a **3L-4M** LAMP-Tree has three levels and four million datapoints per leaf. The first 85% of each time series is used for training; the remaining 15% is used for evaluation.

TABLE I: Summary of the time series used for evaluation. Length (last column) is the number of datapoints in the time series.

Dataset	Duration	Start Date	Length
Seismic	112 days	2004-08-15	140 M
Chicken	1 day	-	8 M

TABLE II: Summary of the LAMP-trees used for evaluation.

Dataset	Structure	Leaves	Internal
Seismic	**3L-4M**	30×4M	5×24M
	3L-10M	12×10M	3×40M
	2L-20M	6×20M	-
	2L-24M	5×24M	-
	2L-40M	3×40M	-
Chicken	**3L-500K**	12×0.5M	3×2M
	3L-1M	6×1M	3×2M
	2L-2M	3×2M	-
	2L-3M	2×3M	-

Table II summarizes the LAMP-Trees that we trained, including the number of leaves and internal nodes, as well as the number of datapoints per leaf and internal node. For example 30×4M leaves means that the tree contains thirty leaves and that each leaf encompasses four million datapoints.

Table III reports each LAMP-Tree's training time; those with the most nodes tend to have the longest training times.

Our evaluation procedure mimicked a streaming scenario using the remaining 15% of both time series. The evaluation subsequence was partitioned into 256-bit windows from which 32 subsequences of length 100 were formed. If we relabel the first datapoint in each window to start with index 1, the subsequences are $\langle t_1 \dots t_{100} \rangle$, $\langle t_{33} \dots t_{132} \rangle$, $\langle t_{65} \dots t_{164} \rangle$, etc., within the window; note that some of the sequences start within the window but extend beyond it. Each subsequence was then used to query the LAMP-Tree.

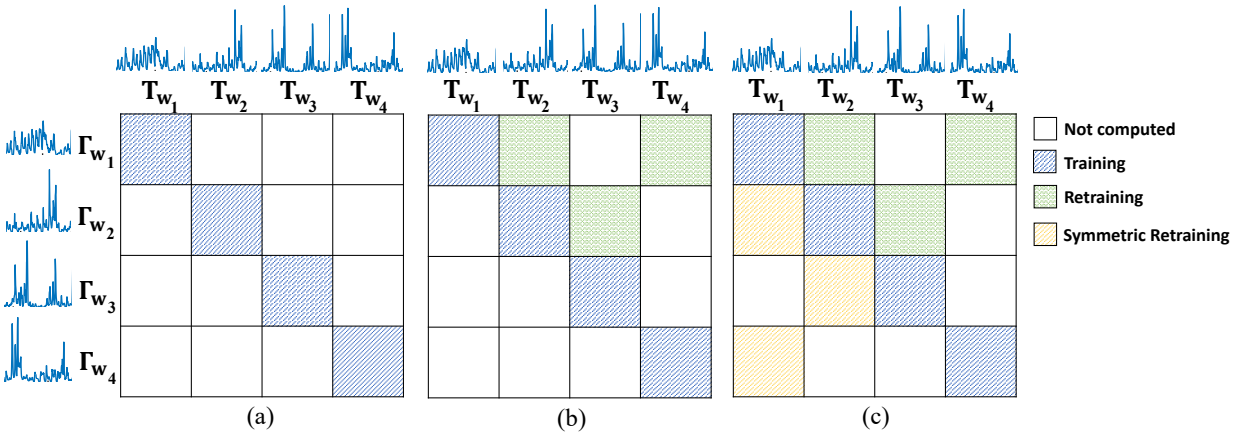


Fig. 5: (a) LAMP-Tree Training: we train a LAMP model L_{w_i} to predict w_i's Matrix Profile $P(T_{w_i})$; (b) LAMP-Tree retraining: for subsequence T_{w_i}, a leaf node $w_j, j < i$ is selected randomly; AB-Matrix Profile $P_{AB}(T_{w_j}, T_{w_i})$ is added to Γ_{w_j}, which we use to retrain L_{w_j}; (c) Symmetric Retraining: in addition, AB-Matrix Profile $P_{AB}(T_{w_i}, T_{w_j})$ is added to Γ_{w_i}, which we use to retrain L_{w_i}.

TABLE III: LAMP-Tree structures training time (T) in minutes

DATASET	STRUCTURE	TIME (MINUTES)
SEISMIC	**3L-4M**	533.11
	3L-10M	293.57
	2L-20M	154.96
	2L-24M	149.86
	2L-40M	118.37
CHICKEN	**3L-500K**	16.898
	3L-1M	11.926
	2L-2M	6.4085
	2L-3M	5.3755

Tradeoffs: There is an inherent tradeoff between the number of leaves and the efficiency of index prediction. Increasing the number of leaves (fewer datapoints per leaf) leads to more precise predictions and faster AB-Matrix Profile Index computation when a match occurs; however, this also means that less data is available to train the LAMP model at the leaf, so model accuracy becomes a concern. Fewer leaves trained with more datapoints per leaf are likely to yield higher accuracy LAMP models, but will require more expensive computations to identify the matching indices.

A. Indexing Evaluation

We evaluate the accuracy of the LAMP-Tree in terms of *precision*, *recall*, and F_1 *score* (the harmonic mean of precision and recall). For a given threshold θ:

- A *true positive (tp)* occurs when the LAMP-Tree predicts a match at leaf w_i and the AB-Matrix Profile confirms the match, i.e., $w_i \in Q_{Exact} \cap Q_{LAMP}$.
- A *false positive (fp)* occurs when the LAMP-Tree predicts a match at leaf w_i, but the AB-Matrix Profile yields no match, i.e., $w_i \in Q_{LAMP} \setminus Q_{Exact}$.
- A *false negative (fn)* occurs when the LAMP-Tree predicts no match at leaf w_i, but the AB-Matrix Profile yields a match, i.e., $w_i \in Q_{Exact} \setminus Q_{LAMP}$.

Let Σtp, Σfp, and Σfn be the number of true positives, false positives, and false negatives observed across all leaves when querying the LAMP-Tree during evaluation; pruning a leaf is interpreted as a prediction of a non-match. Then:

$$Precision = \frac{\Sigma tp}{\Sigma tp + \Sigma fp} \tag{4}$$

$$Recall = \frac{\Sigma tp}{\Sigma tp + \Sigma fn} \tag{5}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6}$$

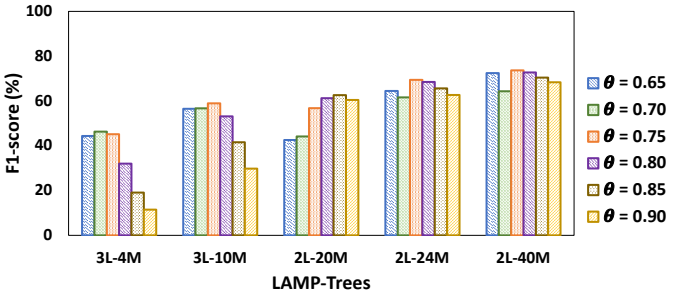
Figure 6 reports the F_1 scores for all nine LAMP-Trees generated for the Seismic and Chicken datasets, without retraining, and with varying threshold (θ) values.

The general trend is that the highest accuracy (F_1 scores in the 75-80% range) is obtained for LAMP-Trees whose nodes have been trained on the largest subsequence lengths; this led us to suspect that the inaccuracies observed for LAMP-Trees whose nodes have been trained on shorter subsequence lengths could be improved through retraining. The most accurate threshold values for the most accurate LAMP-Trees are $\theta = 0.75$ for Seismic and $\theta = 0.60$ for Chicken.

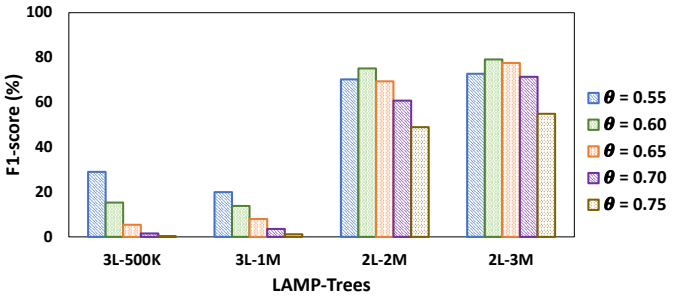
B. Retraining Evaluation

Next, we evaluated the impact of retraining on LAMP-Trees' predictive accuracy. We set the threshold (θ) value to 0.75 for Seismic and 0.65 for Chicken, with $p \in \{0.0, 0.3, 0.6\}$ to vary the amount of retraining; we used symmetric retraining in all cases. The tree 2L-3M for Chicken had a small number of leaf nodes, which were fully retrained with $p = 0.3$; subsequent retraining with $p = 0.6$ were not needed in this case. Figure 7 reports the F_1 scores.

Increasing p from 0 to 0.3 significantly improved the F_1 score for most LAMP-Trees, most notably for tree 3L-4M and 2L-20M for Seismic and 3L-1M for Chicken; subsequently increasing p from 0.3 to 0.6 led to marginal improvements in



(a) Seismic



(b) Chicken

Fig. 6: F_1 scores for different LAMP-Trees and threshold (θ) values for the (a) Seismic and (b) Chicken datasets.

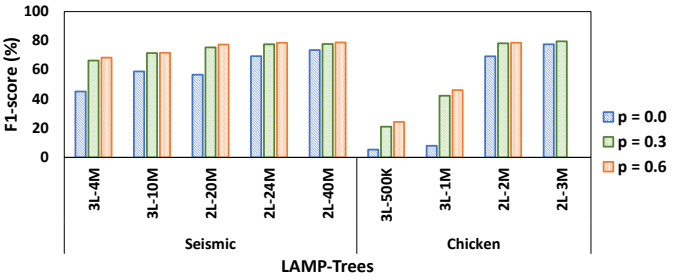


Fig. 7: F_1 score for each LAMP-Tree using symmetric retraining ($p \in \{0, 0.3, 0.6\}$).

most other cases. In general, LAMP-trees whose leaf nodes trained on the shortest subsequence lengths benefitted the most from symmetric retraining. Lamp-Trees 3L-4M and 3L-10M (Chicken) demonstrated incremental improvements in F_1 score, but remain substantially less accurate than the other seven.

C. Retraining Evaluation (kNN Matrix Profile)

We recomputed the accuracy results from V-A substituting $Q^{[k]}_{Exact}$ for Q_{Exact} in the definitions of true positive, false positive, and false negative. Figure 8 reports the F1-score results of this experiment with $k \in 1, 2, 4$-nearest neighbors and $p = 0.3$.

Figure 8 shows that F1-score increases up to 80% or more with $k = 2$ and 85% or more with $k = 4$ for the seven most accurate LAMP-Trees. The definitions of true and false positive do not change under $Q^{[k]}_{Exact}$, i.e., if w_i is a true/false positive with Q_{Exact}, then it will remain a true/false positive

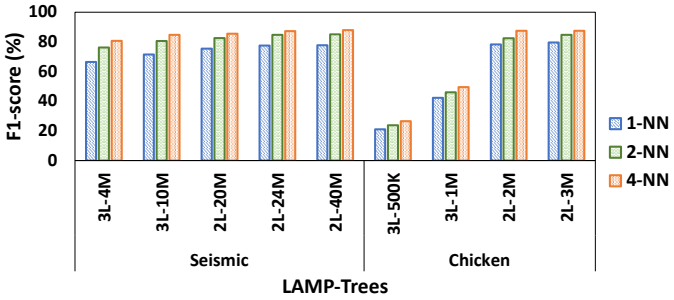


Fig. 8: F_1 score for each LAMP-Tree using the AB-kNN Matrix Profile ($k \in \{1, 2, 4\}$).

with $Q^{[k]}_{Exact}$. Thus, Precision will not change. Recall, on the other hand, can go down. Suppose that no match is predicted at leaf node w_i.We can test w_i for membership in both Q_{Exact} and $Q^{[k]}_{Exact}$ using the AB-kNN Matrix Profile at w_i, itself a vector of correlations: $P^{[k]}_{AB}(T_{w_i}, T_Q) = \{c^{AB(j)}_{w_i} | \ 1 \leq j \leq k\}$.

- If $c^{AB(1)}_{w_i} \geq \theta$, then $w_i \in Q_{Exact}$, which classifies w_i as a false negative by the original definition; this increases Σfn in the denominator of Equation 5.
- If, in addition, $c^{AB(j)}_{w_i} < \theta$ for any $j \in \{2, \ldots k\}$, then $w_i \notin Q^{[k]}_{Exact}$ which classifies w_i as a true negative under $Q^{[k]}_{Exact}$; w_i does affect Σfn or Equation 5.

Our intent is not to claim that LAMP-Tree evaluation using $Q^{[k]}_{Exact}$ is inherently superior to using Q_{Exact} because it leads to higher Recall, nor is it to demonstrate how to proverbially "cook the books;" in fact, there is room to debate alternative ways to compute $Q^{[k]}_{Exact}$. Our opinion is that the evaluation mechanism should be appropriate to the larger system or context in which a LAMP-Tree is deployed.

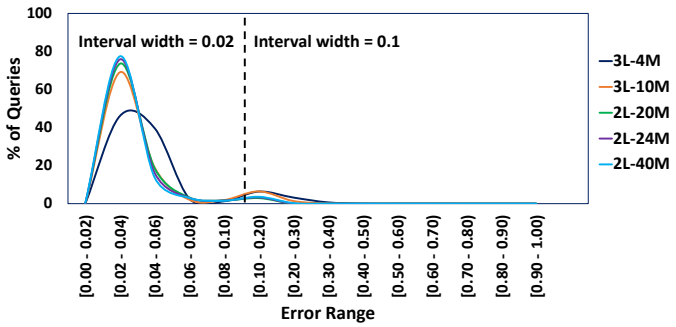
D. Correlation Prediction Error

For query T_Q, the *error* at leaf node w_j is defined as $E_{w_j}(T_Q) = |L_{w_j}(T_{w_j}, T_Q) - P_{AB}(T_{w_j}, T_Q)|$; the error at internal node w_{i-k}, denoted $E_{w_{i-k}}(T_Q)$ is defined similarly. A query that is pruned at w_{i-k} never reaches the leaves; in this case, we propagate w_{i-k}'s error to the leaves in its subtree by setting $E_{w_j}(T_Q) \leftarrow E_{w_{i-k}}(T_Q), i \leq j \leq k$.

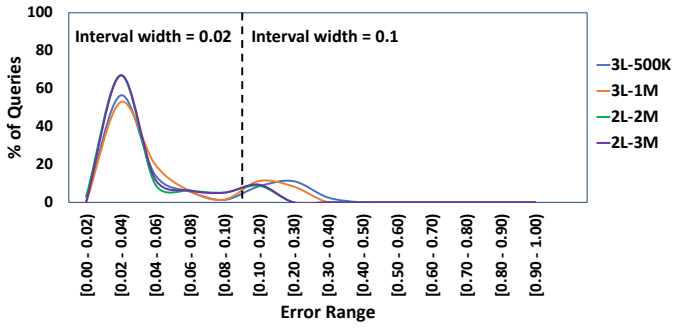
T_Q's error is averaged over all N leaves in the LAMP-Tree:

$$E_{Query}(T_Q) = \frac{1}{N} \sum_{i=1}^{N} E_{w_i}(T_Q) \tag{7}$$

Figure 9 displays the error distribution, which exhibits two peaks. At the first peak, approximately 80% of queries are predicted with error of 4% or less. The second peak, which occurs in the 20-30% error range, is an artifact of pruning. For a match, the typical Pearson Correlation computed by the AB-Matrix Profile is around 0.80-0.85; for a non-match, the typical Pearson correlation is 0.55-0.60. When a non-match (false positive or false negative) occurs, the typical difference between the Pearson correlation computed by the AB-Matrix profile and the Pearson correlation predicted by the LAMP Model is in around 0.20-0.25.



(a) Seismic



(b) Chicken

Fig. 9: Error distribution for all queries in the (a) Seismic and (b) Chicken datasets. Non-uniform x-axis spacing is used to illustrate the respective locations of the two peaks.

E. Hard vs. Easy Queries

We wish to differentiate between easy and hard queries. We leverage the 2NN-AB-Matrix Profile $P^{[2]}_{AB}(T_{w_i}, T_Q) = \langle c^{AB[1]}_{w_i}, c^{AB[2]}_{w_i} \rangle$ at leaf node w_i. Intuitively, we expect a query (including parameter θ) to be "hard" if the difference between the first and second nearest neighbor correlations, $\Delta_{w_i} = |c^{AB[1]}_{w_i} - c^{AB[2]}_{w_i}|$ is large. We offer two interpretations of this claim: (1) Δ_{w_i} being large implies that T_Q's nearest neighbor in T_{w_i} is likely an anomaly; (2) Δ_{w_i} being large increases the likelihood that $c^{AB[1]}_{w_i} \geq \theta \wedge c^{AB[2]}_{w_i} < \theta$, which increases the likelihood of misprediction.

Difficulty: We quantify the *difficulty* of T_Q at leaf w_j as:

$$D_{w_j}(T_Q) = \frac{c^{AB(1)}_{w_j}}{c^{AB(2)}_{w_j}} \tag{8}$$

If T_Q fails to match at an internal node w_{i-k}, the difficulty $D_{w_{i-k}}(T_Q)$ is computed using the same formula and is propagated to all of the leaf nodes in w_{i-k}'s subtree, i.e., $D_{w_j}(T_Q) \leftarrow D_{w_{i-k}}(T_Q), i \leq j \leq k$.

Difficulty Threshold: In order to categorize T_Q as easy or hard, we need to compare $D_{w_j}(T_Q)$ to an appropriately-defined *difficulty threshold*. Considering leaf node w_j in isolation, we might categorize the query as being hard if $c^{AB(2)}_{w_j} > c^{AB(1)}_{w_j} - E_{w_j}(T_Q)$, where $E_{w_j}(T_Q)$ is the error at w_j, as defined in the preceding subsection: as $E_{w_j}(T_Q)$ grows, the matches become more difficult to accurately predict. At the same time, our notion of hardness need not consider a single leaf node in isolation. To account for these factors, we introduce the following difficulty threshold:

$$\tau_Q = \frac{1}{N} \sum_{j=1}^{N} \frac{c^{AB(1)}_{w_j}}{c^{AB(1)}_{w_j} - E_{w_j}(T_Q)} \tag{9}$$

Hardness: We categorize query T_Q as being *hard* if $D_{w_j}(T_Q) > \tau_Q$; otherwise, we categorize T_Q as *easy*.

Table IV reports the error and difficulty threshold for the nine LAMP trees, averaged over all queries; all LAMP-Trees were symmetrically retrained with $p = 0.3$. The queries in our evaluation datasets are categorized as easy and hard. Table IV also reports the Precision, Recall, and F_1 scores for the easy and hard queries for each LAMP-Tree.

Except for 3L-500K and 3L-1M (Chicken), which are wholly inaccurate, Precision tends to be higher for hard queries and lower for easy queries: matches are less likely, since only the first nearest neighbor is correlated correlation to the query; thus, there are fewer false positives. Recall tends to be higher for easy queries and lower for hard queries: with multiple nearest neighbors being correlated to the query, false negative rates tend to decrease. These factors also lead to higher F_1 scores for easy queries.

F. Indexing Time

Lastly, we compare the runtime of querying the nine LAMP-Trees to computing the AB-Matrix Profile Index directly. These LAMP trees did not include retraining, and the reported runtimes do not factor accuracy. Figure 10 reports the speedup for both datasets, varying the accuracy threshold θ in increments of 0.05. The speedups were an order of magnitude greater for the Seismic Dataset compared to the Chicken dataset, due to its larger size, which manifests in substantially greater Matrix Profile computation times [17]. In general, speedups increased as θ increased due to fewer matches and more pruning.

The two most inaccurate LAMP-Trees, 3L-500K and 3L-1M, had the highest and lowest reported speedups for Chicken (technically, a slowdown for 3L-1M). Here, the dataset is small enough that the time required to compute the AB-Matrix Profile Index directly so small that LAMP-Tree-based index cannot achieve a substantial speedup.

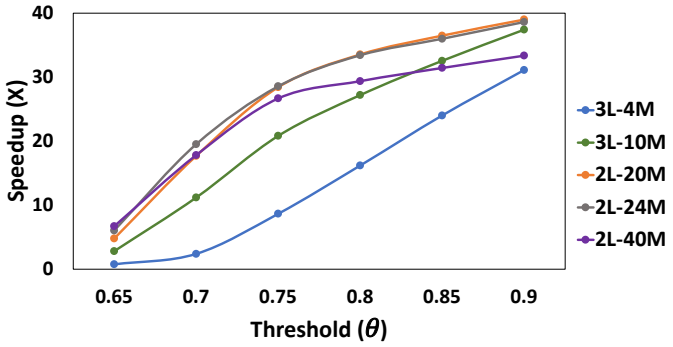
Notably, the LAMP-Tree with the fewest datapoints per leaf node achieved the highest speedup for Chicken (3L-500K) and the lowest speedups for Seismic (3L-4M). Beyond that, there were no discernable trends that indicated better performance for 2- or 3-level LAMP-Trees.

VI. Conclusion and Future Work

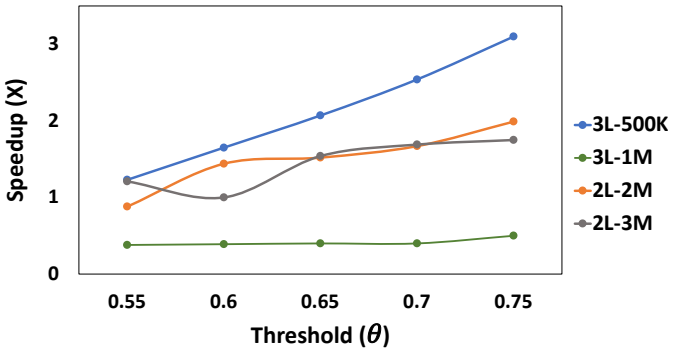
Approximating the Matrix Profile Index is considerably faster than computing the Matrix Profile directly, especially for large time series, while achieving what we consider to be acceptable accuracy when (re-)trained on a sufficiently large representative data set. Consequently, we believe that the LAMP-Trees will be sufficiently fast for deployment in real-time applications on cost-constrained embedded systems. Future work will examine implementation issues in greater

TABLE IV: Error and Diffuclty Threshold (**DT**) for each LAMP-Tree, averaged over all queries; each tree was trained with symmetric retraining and $p = 0.3$. Precision, Recall, and F_1 scores for the Hard (**-H**) and Easy (**-E**) queries.

DATASET	LAMP-TREE	ERROR	DT	PREC-H	PREC-E	REC-H	REC-E	F1-H	F1-E
SEISMIC	**3L-4M**	0.0568	1.062	81.67	71.51	38.02	70.04	51.89	70.95
	3L-10M	0.0467	1.049	85.68	76.42	43.41	75.52	57.62	75.97
	2L-20M	0.0388	1.040	85.33	76.72	52.98	81.90	65.37	79.23
	2L-24M	0.0383	1.040	85.83	78.22	57.06	83.56	68.55	80.80
	2L-40M	0.0376	1.039	86.95	79.74	53.91	82.89	66.56	81.28
CHICKEN	**3L-500K**	0.0702	1.075	17.25	24.84	31.18	17.97	22.21	20.85
	3L-1M	0.2247	1.290	46.26	48.89	85.14	36.71	59.95	41.93
	2L-2M	0.0418	1.044	86.79	77.62	60.07	82.13	71.00	79.81
	2L-3M	0.0356	1.037	85.16	84.59	53.58	80.83	65.78	82.67



(a) Seismic Dataset



(b) Chicken Dataset

Fig. 10: Indexing speedup of LAMP-Tree compared to exact Matrix Profile Index computation. LAMP-Tree predictions may terminate early if no matches are found. Increasing the threshold (fewer matches) tends to reduce the LAMP-Tree prediction time.

detail, including lower-cost machine learning models than 1D-CNN that we are presently using, and detection of concept drift and online retraining after a system employing the LAMP-Tree has been deployed.

Another direction for future work is to focus on specific deployment scenarios. For example, the original LAMP paper [1] evaluated a single LAMP model (equivalent to the root of a LAMP-Tree) on a Raspberry Pi, and showed that real-time inference could be achieved, depending on the target sampling rate and the length of the query. LAMP-Tree could be evaluated on Raspberry Pi or another ARM-based embedded CPU platform as well. LAMP and LAMP-Tree could also benefit from hardware acceleration, for example, leveraging technologies such as FPGAs, Edge TPUs, or other AI-specific architectures.

ACKNOWLEDGMENT

This work was supported in part by NSF Awards #1528181, #1763795, #1901379, and #1932254. P. Brisk has a small equity stake in Shapelets, a company providing decision-support software for time series. The authors declare no other competing interests.

REFERENCES

[1] Z. Zimmerman, N. Shakibay Senobari, G. Funning, E. Papalexakis, S. Oymak, P. Brisk, and E. Keogh, "Matrix Profile XVIII: Time Series Mining in the Face of Fast Moving Streams using a Learned Approximate Matrix Profile," 2019 IEEE International Conference on Data Mining (ICDM), pp. 936–945.

[2] Y. Zhu, Z. Zimmerman, N. Shakibay Senobari, C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins," 2016 IEEE 16th international conference on data mining (ICDM), pp. 739–748, 2016.

[3] Y. Zhu, M. Imamura, D. Nikovski, and E. Keogh , "Matrix profile VII: Time series chains: A new primitive for time series data mining (best student paper award)," 2017 IEEE International Conference on Data Mining (ICDM), pp.695–704.

[4] K.P. YChan, and A. W. Fu, "Efficient time series matching by wavelets," Proceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337), pp.126–133, 1999.

[5] Lin, Jessica and Keogh, Eamonn and Wei, Li and Lonardi, Stefano, "Experiencing SAX: a novel symbolic representation of time series," Data Mining and knowledge discovery, vol. 15, pp.107–144, 2007 .

[6] J. Shieh, and E. Keogh, "i SAX: indexing and mining terabyte sized time series," Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.623–631, 2008.

[7] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh, "iSAX 2.0: Indexing and mining one billion time series," 2010 IEEE International Conference on Data Mining, pp.58–67, 2010.

[8] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh, "Beyond one billion time series: indexing and mining very large time series collections with iSAX2+," Knowledge and information systems, vol. 39, No. 1, pp.123–151, 2014.

[9] K. Zoumpatianos, S. Idreos, T. Palpanas, "Indexing for interactive exploration of big data series," Proceedings of the 2014 ACM SIGMOD international conference on Management of data, pp.1555–1566, 2014.

[10] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pp.322–331, 1990.

[11] G. Chatzigeorgakidis, D. Skoutas, K. Patroumpas, S. Athanasiou, and S. Skiadopoulos, "Indexing geolocated time series data," Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp.1–10, 2017.

[12] L. Zhang, N. Alghamdi, M. Y Eltabakh, and E. A Rundensteiner, "TARDIS: Distributed Indexing Framework for Big Time Series Data," 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp.1202–1213, 2019.

[13] G. Chatzigeorgakidis, K. Patroumpas, D. Skoutas, S. Athanasiou, and S. Skiadopoulos, "Visual Exploration of Geolocated Time Series with Hybrid Indexing," Big Data Research , vol.15, pp.12–28, 2019.

[14] M. Linardi, Y. Zhu, T. Palpanas, and E.Keogh, "Matrix profile X: VALMOD-scalable discovery of variable-length motifs in data series," Proceedings of the 2018 International Conference on Management of Data, pp.1053–1066, 2018.

[15] C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh , "Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets," 2016 IEEE 16th international conference on data mining (ICDM), pp.1317–1322, 2016.

[16] N. Shakibay Senobari, G. J Funning, E. Keogh, Y. Zhu, C. M. Yeh, Z. Zimmerman, and A. Mueen, "Super-Efficient Cross-Correlation (SEC-C): A Fast Matched Filtering Code Suitable for Desktop Computers," Seismological Research Letters, vol. 90, No, 1, pp.322–334, 2019.

[17] Z. Zimmerman, K. Kamgar, N. Shakibay Senobari, B. Crites, G. Funning, P. Brisk, and E. Keogh, "Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond," Proceedings of the ACM Symposium on Cloud Computing, pp.74–86, 2019b.

[18] F. Madrid, S. Imani, R. Mercer, Z. Zimmerman, N. Shakibay, and E. Keogh, "Matrix Profile XX: Finding and Visualizing Time Series Motifs of All Lengths using the Matrix Profile," 2019 IEEE International Conference on Big Knowledge (ICBK), pp.175–182, 2019.

[19] Z. Zimmerman, K. Kamgar, Y. Zhu, N. Shakibay Senobari, B. Crites, G. Funning, P. Brisk, and E. Keogh, "Scaling Time Series Motif Discovery with GPUs: Breaking the Quintillion Pairwise Comparisons a Day Barrier," Preprint],[Online]. Available at: https://www. cs. ucr. edu/% 7Eeamonn/public/GPU% 5C_Matrix% 5C_profile% 5C_VLDB% 5C_30DraftOnly. pdf, 2018.

[20] C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, Z. Zimmerman, D. F. Silva, A. Mueen, and E. Keogh, "Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile," Data Mining and Knowledge Discovery, vol. 32, No. 1, pp.83–123, 2018.

[21] Y. Zhu, C. M. Yeh, Z. Zimmerman, and E. Keogh, "Matrix Profile XVII: Indexing the Matrix Profile to Allow Arbitrary Range Queries," 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp.1846–1849, 2020.

[22] M. Karimi, A. Jahanshahi, A. Mazloumi, and H. Zamani Sabzi, "Border Gateway Protocol Anomaly Detection Using Neural Network," 2019 IEEE International Conference on Big Data (Big Data), pp.6092–6094, 2019.

[23] T. Kraska, A. Beutel, E. H Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," Proceedings of the 2018 International Conference on Management of Data, pp.489–504, 2018.

[24] A. Abdoli, S. Alaee, S. Imani, A. Murillo, A. Gerry, L. Hickle, and Keogh, Eamonn, "Fitbit for Chickens? Time Series Data Mining Can Increase the Productivity of Poultry Farms," Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp.3328–3336, 2020.

[25] A. Guttman, "R-trees: A dynamic index structure for spatial searching," Proceedings of the 1984 ACM SIGMOD international conference on Management of dat, pp.47–57, 1984.

[26] Northern California Earthquake Data Center, "Northern California Earthquake Data Center," HRSN (2014), High Resolution Seismic Network. UC Berkeley Seismological Laboratory. Dataset. doi:10.7932/HRSN., 2014.

[27] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, and et. al., "roceedings of the national academy of sciences, vol. 114, No. 13, pp.3521–3526, 2017.

[28] X. Chen, S. Wang, B, Fu, M. Long, and J. Wang, "Catastrophic forgetting meets negative transfer: Batch spectral shrinkage for safe transfer learning," Advances in Neural Information Processing Systems, pp.1908–1918, 2019.

[29] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A Comprehensive Survey on Transfer Learning," CoRR, vol.abs/1911.02685, 2019.