UC Riverside

UC Riverside Previously Published Works

Title

Matrix profile XVIII: Time series mining in the face of fast moving streams using a learned approximate matrix profile

Permalink

https://escholarship.org/uc/item/55j6w5d7

Journal

Proceedings IEEE International Conference on Data Mining Icdm, 2019-November

ISSN

1550-4786

ISBN

978-1-7281-4604-1

Authors

Zimmerman, Z

Shakibay Senobari, N

Funning, G

et al.

Publication Date

2019-11-01

DOI

10.1109/ICDM.2019.00104

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at https://creativecommons.org/licenses/by/4.0/

eScholarship.org Powered by the California Digital Library
University of California

Matrix Profile XVIII: Time Series Mining in the Face of Fast Moving Streams using a Learned Approximate Matrix Profile

Zachary Zimmerman, Nader Shakibay Senobari, Gareth Funning, Evangelos Papalexakis, Samet Oymak, Philip Brisk, and Eamonn Keogh
University of California, Riverside
{zzimm001, nshak006, gareth}@ucr.edu, oymak@ece.ucr.edu, {epapalex, philip, eamonn}@cs.ucr.edu

***Abstract*—In recent years, the Matrix Profile has emerged as a promising approach to allow data mining on large time series archives. By efficiently computing all of the "essential" distance information between subsequences in a time series, the Matrix Profile makes many analytic problems, including classification and anomaly detection, easy or even trivial. However, for many tasks, in addition to archives of data, we may face never-ending streams of newly arriving data. While there is an algorithm to maintain a Matrix Profile in the face of newly arriving data, it is limited to streams arriving on the order of one Hz and with small archives of historical data. However, in domains as diverse as seismology, neuroscience and entomology, we may encounter datasets that stream at rates that are orders of magnitude faster. In this work we introduce LAMP, a model that predicts, in constant time, the Matrix Profile value that *would* have been assigned to an incoming subsequence. This allows us to exploit the utility of the Matrix Profile in settings that would otherwise be untenable. While learning LAMP models is computationally expensive, this stage is done offline with an arbitrary computational paradigm. The models can then be deployed on resource-constrained devices including wearable sensors. We demonstrate the utility of LAMP with experiments on diverse and challenging datasets with billions of datapoints on a simple desktop machine. We achieve more than 10000x speedup over exact methods on the same data.**

Keywords*—*Time Series, Streaming Data, Matrix Profile

I. Introduction

As time series data becomes ever more pervasive in personal, industrial and scientific settings, there has been an explosion of interest in creating algorithms to analyze such data. The Matrix Profile (MP) has emerged as a promising tool to support many such time series data mining tasks [29][31][32]. The MP is simply a data structure that contains the nearest neighbor distance for every subsequence in a time series. It has two unexpected properties: it can be computed *very* efficiently, and, given *just* the MP, most time series analytic tasks are easy or even trivial to solve. In particular, it has been shown that we can use the MP for classification, motif discovery [29][23], anomaly detection [31], evolving pattern (chain [32]) discovery, summarization and segmentation [29][31].

Moreover, the MP can be computed incrementally, meaning that we can create streaming versions of the algorithms noted above. STOMP*I* is the current state of the art algorithm for maintaining the matrix profile on streaming data. However, STOMP*I* has a problem: the time required to update the MP slowly grows as a function of how much data we have seen. Suppose we start monitoring a new 5 Hz process at midnight on Sunday. Initially, we can use STOMP*I* to maintain the MP, and have plenty of cycles to spare. However, by Wednesday at 10:25 AM, when we have seen just over one million datapoints and we can no longer maintain the MP fast enough[1], the next datapoint will arrive before STOMP*I* is finished updating the matrix profile for the last datapoint.

We can push back this time horizon with faster machines, but the reprieve is temporary. At some point, the growing computational demands will outstrip our resources. To make this concrete let us preview two real-world applications of our system that we will later revisit in our experiments. In Fig. 1.*top* we show a classification problem for telemetry for insects. The recording apparatus produces a snippet that we must classify in to one of several classes. We have just 1/100th of a second to do this, before the next snippet arrives.

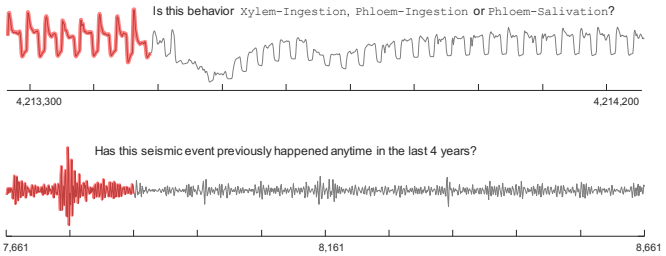


Fig. 1: Two time series subsequences (shown in red) that need to be quickly processed. *top*) An example of data from an insect EPG (Electrical Penetration Graph) apparatus. *bottom*) An example of a trace from a seismograph.

In Fig. 1.*bottom* we show a snippet from a seismograph. Here the sampling rate (after some inline processing) is slower, with new snippets arriving every 1/20th of a second. However, to answer the question posed, we need to compare this data with four years of data, or 2.53×10^9 datapoints.

The problem is exacerbated by the fact that we would like to deploy MP-based algorithms on embedded devices with very little computational power. This would potentially allow analytics to be done "at the edge" [18], reducing the network and power overhead of transmitting data.

In this work, we propose to solve this problem by introducing a Learned Approximate Matrix Profile (LAMP), which enables constant time approximation of the MP value given a newly arriving time series subsequence. With this approximate value, we can do most of the analytics based on the MP, including anomaly detection and classification.

[1] Assuming an off-the shelf desktop machine. Full details of this calculation are deferred to [13] to enhance the flow of the text.

The rest of this paper is organized as follows. In Section II we introduce background material and review related work. This allows us to introduce our algorithm in Section III. Section IV offers an extensive empirical evaluation of our ideas, before we offer conclusions and directions for future work in Section V.

II. RELATED WORK AND BACKGROUND

In this section, we first introduce all necessary definitions before considering related work.

A. Definitions

We begin by defining the data type of interest, *time series*:

Definition 1: A time series T is a sequence of real-valued numbers t_i: $T = t_1, t_2, ..., t_n$ where n is the length of T:

For most *time series* data mining tasks, we are interested not in *global*, but *local* properties of a time series. A local region of a time series is called a *subsequence*:

Definition 2: A subsequence $T_{i,m}$ of a time series T is a continuous subset of the values from T of length m starting from position i. Formally, $T_{i,m} = t_i, t_{i+1},\ldots, t_{i+m-1}$, where $1 \leq i \leq n-m+1$.

Given a query subsequence $T_{i,m}$ and a time series T, we can compute the correlation between $T_{i,m}$ and *all* the subsequences in T. We call this a *correlation profile*:

Definition 3: A *correlation profile* C_i corresponding to query $T_{i,m}$ and time series T is a vector of the Pearson correlations between a given query subsequence $T_{i,m}$ and each subsequence in time series T. Formally, $C_i = [c_{i,1}, c_{i,2}, ..., c_{i,n-m+1}]$, where $c_{i,j}$ $(1 \leq j \leq n-m+1)$ is the Pearson correlation between $T_{i,m}$ and $T_{j,m}$.

Note that prior work [29][31][32] has defined the matrix profile in terms of the Euclidean distance between z-normalized subsequences. However, in this work, we define the matrix profile in terms of the Pearson correlation. This is because it creates results limited to the intuitive range of [-1, 1]. For example, seismologists may prefer to filter out weakly matching sequences for some analytic task, perhaps by setting a correlation threshold to say 0.8 [23]. Working with correlation allows them to reuse such a threshold on multiple datasets, without having to worry about the sampling rate of the length of the subsequences. In contrast, for Euclidean distance, any threshold discovered would have to be recalibrated for new sampling rates or subsequence lengths.

It is important to recognize that using correlation does not change the information contained in the matrix profile, as the Pearson correlation can be converted to z-normalized Euclidean distance in constant time [16]. Moreover, the ranking of all the top-K nearest neighbors to a time series is *identical* under Pearson correlation and between z-normalized Euclidean distance.

Once we obtain C_i, we can extract the nearest neighbor of $T_{i,m}$ in T. Note that if the query $T_{i,m}$ is a subsequence of T, the i^{th} location of correlation profile C_i is 1 (i.e., $c_{i,i} = 1$) and close to 1 just to the left and right of i. This is called a *trivial match* in the literature. We avoid such matches by ignoring an "exclusion" zone of length $m/4$ before and after i, the location of the query. In practice, we simply set $c_{i,j}$ $(i-m/4 \leq j \leq i+m/4)$ to negative infinity, and the nearest neighbor of $T_{i,m}$ can thus be found by evaluating $\max(C_i)$.

We wish to find the nearest neighbor of every subsequence in T. This nearest neighbor information is stored in two "meta time series", the *matrix profile* and the *matrix profile index*:

Definition 4: A *matrix profile* P of time series T is a vector of the Pearson correlation between every subsequence of T and its nearest neighbor in T. Formally, $P = [\max(C_1), \max(C_2), ..., \max(C_{n-m+1})]$, where C_i $(1 \leq i \leq n-m+1)$ is the correlation profile C_i corresponding to query $T_{i,m}$ and time series T.

The i^{th} element in the matrix profile P tells us the Pearson correlation from subsequence $T_{i,m}$ to its nearest neighbor in time series T. However, it does not tell us the *location* of that nearest neighbor; this is stored in the companion *matrix profile index*:

Definition 5: A *matrix profile index* I of time series T is a vector of integers: $I=[I_1, I_2, \ldots I_{n-m+1}]$, where $I_i=j$ if $c_{i,j} = \max(C_i)$.

Fig. 2 shows the relationship between correlation matrix, correlation profile (Definition 3) and matrix profile (Definition 4). Each element of the correlation matrix $c_{i,j}$ is the correlation between $T_{i,m}$ and $T_{j,m}$ $(1 \leq i, j \leq n-m+1)$ of time series T.

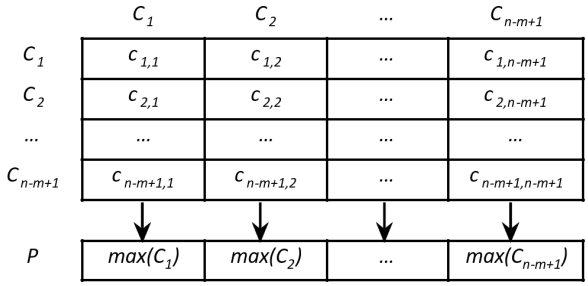


Fig. 2: The relationship between the correlation matrix, correlation profile and matrix profile. A correlation profile is a column (also a row) of the correlation matrix. The matrix profile stores the maximum (off diagonal) value of each column of the correlation matrix; the location of the maximum value within each column is stored in the companion matrix profile index.

Fig. 3 shows a visual example of a correlation profile and a matrix profile created from the same time series T.

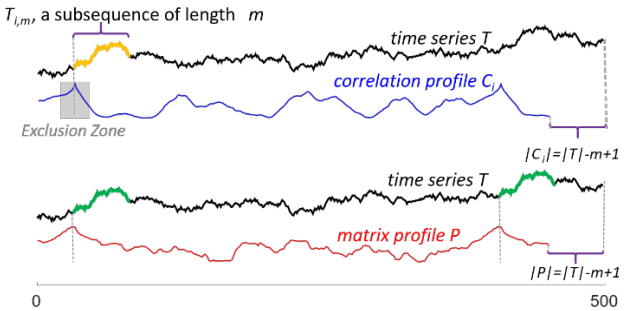


Fig. 3: *top*) A correlation profile C_i created from $T_{i,m}$ shows the correlation between $T_{i,m}$ and all the subsequences in T. The values in the dark zone are ignored to avoid trivial matches. *bottom*) The matrix profile P is the element-wise maximum of all the correlation profiles (C_i is one of them). Note that the two highest values in P are at the location of the 1st motif in T. (Figure adapted from [33], used with permission).

Note that as presented above, the matrix profile can be considered a *self-join* [29]: for every subsequence in a time series *T*, it records information about its (non-trivial-match) nearest neighbor in the same time series *T*. However, we can trivially generalize it to be an AB-join [29]: for every subsequence in a time series *A*, it records information about its nearest neighbor in time series *B*. Note that *A* and *B* can be of different lengths, and that in general, AB-join ≠ BA-join.

We are now able to introduce the definitions immediately relevant to the problem at hand. First, we consider the output of the STOMP*I* algorithm [29], which is the *exact* matrix profile for all data seen up to the current time in a streaming setting. This will serve as our ground truth or *oracle*.

Definition 6: An *Oracle Matrix Profile (OMP)* of a stream *S* is the matrix profile of the entire stream; it encodes the nearest neighbor for *all* subsequences in the history of the stream, where the nearest neighbor can be any observed subsequence of S.

Given a new set of **k** *consecutive* subsequences observed from S: the OMP can be updated via STOMP*I* in time O((S+k)k), which is the time it takes to compute the AB-join between S and k and the self-join of k. For data with a very low sample rate, it might be enough to simply maintain the OMP. However, in many cases this is untenable because the cost of maintaining the OMP grows as more data is observed. Therefore, we assume we have a representative subset of *S* that we can use as a proxy.

Definition 7: A *Representative Matrix Profile (RMP)* of a stream *S* is the matrix profile of the entire stream, where the nearest neighbor can only occur in some representative subset *R* consisting of observed subsequences of *S*. Formally, *RMP* is the AB-join between *S* and *R* where *RMP* encodes the nearest neighbor of each subsequence of *S* in *R*. Note that an exclusion zone must be applied to each subsequence in *R* when comparing to its 'original copy' in *S*.

We can update the RMP in time O(Rk) which is the time it takes to compute the AB-join between R and k. Note that the time complexity per update no longer depends on the entire history of the stream. For some applications this might be enough. However, for applications with a high sample rate, large R, or on systems with low computational power, this will still likely be above our compute budget and we will need something better.

Definition 8: A *Learned Approximate Matrix Profile (LAMP)* of a stream *S* is the output of a learned model that compresses R into a fixed-size compressed representation. It approximates the *RMP* of *S*.

Using this definition, we can now perform updates in O(k), no longer depending on the size of the representative dataset.

B. Motivation and Formal Problem Statement

Assume we have a continuously arriving stream of time series from a sensor. We may wish to take the most recent subsequence of length *m* and compare it with an archive of previously collected data. There are multiple reasons why we may wish to do this, including:

- **Classification**: We may have partitioned our archive of previously collected data into labeled subsets, for example {`wild-type | mutant`} [5] or {`ingestion | probing | salivation`} [28]. In this case we have an implicit nearest neighbor classifier.
- **Anomaly Detection**: In some domains, we can expect that all newly arriving subsequences should be close to a pattern we have already observed. A pattern that is not (formally a "time series discord" [2]) may signal the discovery of an anomaly.
- **Segmentation**: In [10] it was shown that a very competitive time series sematic segmentation algorithm can be built on top of the Matrix Profile.

A more formal problem statement is:

Problem Statement: Given a streaming time series and representative subset of data from that stream, subject to the constraint that data must be analyzed at the time of arrival, approximate the matrix profile values associated with this newly arriving data, such that they closely approximate the matrix profile values that would be produced by existing exact methods.

C. Dismissing Apparent Solutions

Before introducing LAMP, here we will take the time to dismiss some *apparent* solutions to the task at hand.

- **Indexing**: Would it be possible to just index the data, and perform a nearest neighbor search for each arriving subsequence? Recall that the seismology example shown in Fig. 1.*bottom* would require us to index 2.53×10^9 datapoints. The fastest query times for datasets approaching this size are three to four orders of magnitude slower than our required processing rate [8]. Moreover, virtually every indexing techniques takes a variable and unpredictable amount of time to answer queries. Thus, even if we had a *much* slower arrival rate where the index could keep up *on average,* and we had a highly optimized index running in main memory, it is always possible that we could see multiple slow-to-process subsequences in a row, and therefore run out of time.

- **Dictionary Building/Numerosity Reduction**: Could we not just build a compact "dictionary" of events and brute force search it in the time allowed? There are many papers that suggest something like this, and it is good idea in limited circumstances. For example, it seems to be possible to explicitly build a full dictionary of heartbeats; several papers have explicitly suggested this "*We model heartbeats by dictionaries*.."[6]. However, heartbeats are a relatively easy case, as there are algorithms that can robustly extract individual phase-aligned beats. In contrast, we are interested in datasets where this is not possible in general, because the target behavior is highly polymorphic, and only weakly labeled. Consider Fig. 4, which shows some examples of a single behavior from an insect. We know it reflects a single behavior because an entomologist labeled the entire five-minute session with the label `Xylem-Ingestion`. However, it not clear that we could build a dictionary to summarize this class, either with an algorithm or using significant human labor.

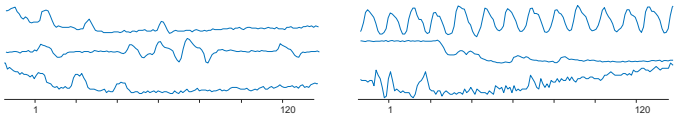


Fig. 4: Six random examples of insect `Xylem-Ingestion` behavior, from a single insect, taken from a five-minute window.

As the reader will come to appreciate, in a sense LAMP is *implicitly* both indexing and dictionary building. The intuition behind LAMP can be summarized as: If a data object is conserved in the training data (dictionary building) then make sure it is represented in the LAMP model (index). However, unlike indexing, LAMP can return an answer in strictly bounded constant time, and unlike dictionary building, LAMP does not need carefully curated data.

D. Related Work

Our proposed system touches on many aspects of data mining, time series analysis, classification, streaming data and deep learning. However, we believe that there is no direct competitor to LAMP. Our work extends and exploits the Matrix Profile, which has recently gained significant attention because of its generality and simplicity [29][31][32][8]. While there exists a technique to incrementally update the Matrix Profile [29], it is limited to settings where the update rate is relatively slow, on the order of ~1 Hz. As the original authors point out [29], there are many domains where this is more than sufficient. However, domains in medicine, seismology and life sciences (i.e. entomology) can produce data at least two orders of magnitude faster than this.

While one instance of LAMP uses a deep neural network, it is important for us to note that we are not claiming any contribution to deep learning. We simply assume that the current state-of-the-art can be plugged into our framework.

III. Method

In order to avoid ever-growing computational cost as more data is observed, we will assume that we have some representative time series, R, observed from the stream. For example, the insect data we empirically consider in Section IV is collected each day, seven days a week in ten to sixteen-hour sessions. We can take a single day of this data, and use it as our R, for all sessions recorded on subsequent days.

Given R, we can generate the RMP (Definition 7) for the stream. The RMP is illustrated in Fig. 5. If our comparison data is truly representative of the stream, then this RMP will very closely resemble the oracle OMP (Definition 6).

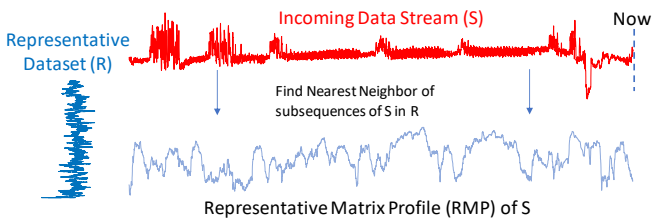


Fig. 5: Illustration of comparisons to compute the ab-join MP with a representative dataset (Linear in the length of R)

If R is small and our sample rate is low enough, say, under one million datapoints, with a sample rate of 1 Hz, then we are done. We can approximate the matrix profile values of new subsequences in time O(||R||). However, if the representative dataset is large, or we are working with low power hardware, then the computational complexity would likely still be above our compute budget for typical streaming rates.

Thus, in order to truly have a useful method in the general case, we need an algorithm that does not depend on the full size of the representative dataset. To this end, we implement LAMP (Definition 8), which models the representative dataset in a compressed, fixed-memory-size model, which requires a fixed time budget to process each arriving datapoint. Fig. 6 illustrates this idea.

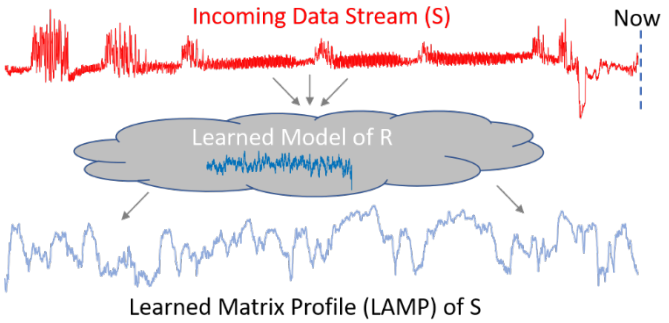


Fig. 6: Illustration of approximation of the matrix profile using a model learned from a representative dataset

There are many learned models we can choose to summarize the training data; we choose to highlight two of these in this work.

A. Top K Diverse Motifs

Table 1 explains our method for extracting the Top-k diverse motifs from a training dataset. In line 2 we compute its exact MP using GPU-STOMP [31], then in line 3-4 we sort the MP to generate a list of the top motifs. Then in lines 6-11 we create a model which contains a set of diverse motifs such that no pair is closer to each other than the diversity threshold.

There are other methods for selecting a diverse set of *k* motifs. For example, prior work investigated the *k*-diversification problem for time series [9]. We leave such considerations for future work.

There are several advantages to using the Diverse Motifs model as the subroutine for LAMP. It is highly interpretable; every subsequence in the model (everything that the model "knows") can be directly visualized. Additionally, adding examples to the model is as simple as appending to a list.

Moreover, because the motifs are sorted by their utility, we can use the Diverse Motifs model as an *anyspace* model. An anyspace model is the analogue of an *anytime* algorithm [29], but with memory as the limiting factor. For example, suppose we create a 10,000 motif model to use in an insect monitoring task in a lab (See Fig. 12). However, later the entomologist has the idea to run experiments in the field with a small memory

limited device such as a raspberry pi, which only has space or compute resources for say 2,500 motifs. We can simply truncate off the bottom ¾ of the motifs and push the model onto the smaller device. The main disadvantage of this method is that it is essentially uncompressed. Since the size of the model affects the number of operations required to perform the prediction, performing inference when *k* is large can be slow and prohibitively expensive to run in real-time.

TABLE 1: TRAINING AND PREDICTING USING THE TOP-K MOTIFS LAMP REPRESENTATION

	Inputs: time series **t**, subsequence length **m**, number of motifs to extract **k**, diversity threshold **d**.
1	**Train**(t,m,k,d):
2	MP = GPU-STOMP(t, m)
3	indexes = sequence(0,length(MP))
4	indexes = argsort(indexes, MP)
5	model = [], count = 0
6	**while** count < length(MP) **and** length(model) < k **do**
7	considered = t[indexes[count]:indexes[count]+m]
8	**if** IsDiverse(considered, model, d) **then**
9	model.append(considered)
10	**endif**
11	count++
12	**done**
13	**return** model
1	**IsDiverse**(motif, currentMotifs, threshold)
2	**for** sequence **in** currentMotifs **do**
3	**if** correlation(motif, sequence)> threshold **then**
4	**return** false
5	**endif**
6	**done**
7	**return** true
1	**Predict** (sequence, model)
2	maxcorr = -1;
3	**for** motif **in** model **do**
4	corr **=** correlation(motif, sequence)
5	**if** corr > maxcorr **then**
6	maxcorr = corr
7	**endif**
8	**return** maxcorr

B. *Neural Networks*

Using a neural network as the basis of the LAMP model has many advantages. We can utilize any of the infrastructures built up around deep learning over the last several years, including GPU optimized code, embedded platform support, and ongoing research in accuracy/speed tradeoffs, which can allow us to adapt to a stream's sample rate according to our platform.

While LAMP is agnostic to the actual network used, in this work we use the simplified version of Resnet [11] proposed by [27] for time series classification, but with the activation on the output layer changed to sigmoid to enable regression. We also modify the input and output of the Resnet model to support multiple predictions at once. i.e. Each of our inputs consists of **J** z-normalized subsequences of length **M** from the data, extracted with stride **S**. This procedure defines an extraction window in the time series, **W**, where $||\boldsymbol{W}|| = \boldsymbol{JS} + \boldsymbol{M} - 1$. We can slide **W** across the time series and extract a new input for the neural network for each position of **W**. Following this logic, each input to the neural network is a vector of length **M** with **J** channels, where we set **M** as the subsequence length parameter of the matrix profile. For each input, the neural network outputs $\boldsymbol{JS}$ LAMP values, one for each subsequence in **W**. Fig. 7 shows the outline of our scheme.

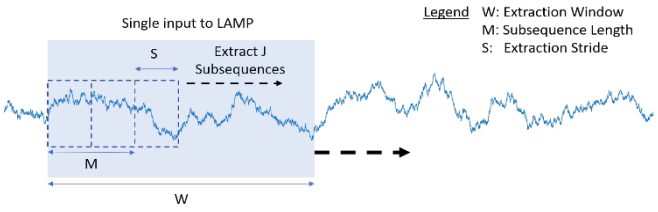


Fig. 7: LAMP neural network input scheme.

This input scheme has three main advantages over a single subsequence input scheme:

1. Reduces overfitting by increasing the dimensionality of the output space. Intuitively, a larger output dimension provides regularization and leads to smoother predictions.
2. Enables faster processing by GPUs and CPUs by exposing additional parallelism through the added dimensionality.
3. Enables the convolutional network to learn short-term time dependencies in the data.

It is important to note that when the subsequence length is very long, the inputs to the neural network also get large. Though it is possible to perform subsampling and other types of dimensionality reduction on the input before sending it to a LAMP model, we have found that the most effective way to reduce the amount of input to the model is to increase the extraction stride **S**. In almost all applications, this is a reasonable assumption. For example, if a classifier correctly predicts that you are `running` at time 17_{sec}, *and* that you are `running` at time 19_{sec}, it is a reasonable assumption that you were also `running` at all times in-between. For very long subsequence lengths, there is a large overlap between the information contained in consecutive or close-by subsequences, so a moderate increase in the extraction stride typically causes an imperceptible accuracy loss in these cases.

For the experiments in this paper, we set W=256, S=8, J=32. We set the learning rate to 1e-3 and we use the Adam [7] optimizer for stochastic gradient descent with a batch size of 32. We optimize the network for the mean squared error loss between the predicted and exact RMP values for our training data. These all reflect common values/practices in the literature. We did not carefully optimize the model, as we wish to demonstrate the robustness of our overall system. The network is implemented in Keras and available at [13].

C. *Configuring the Model Size*

It is useful to consider how to select the size of a LAMP model, as this can be done deterministically before model construction. Given the computational capability **c** (FLOP/s) of a system and the sample rate **r** (Hz) of a stream we can compute the maximum size of a model, in terms of the number of FLOPs possible per inference step using the equation $FLOPs = \boldsymbol{cr}$. Once we know our limitations, we can choose a model appropriate for our specific use case.

It is also important to note that in many of these applications we do not need the result *immediately* even in a real-time application. For example, if our sample rate is 100Hz, perhaps we don't need to make decisions based on every single new

datapoint, but only once per second. In this case, we can process multiple subsequences at once via batching, like in our neural network input scheme, which is more efficient computationally, and can help give additional context to our predictions.

IV. Empirical evaluation

To ensure that our experiments are reproducible, we have built a website [13] which contains all data/code/raw spreadsheets for the results, in addition to many experiments that are omitted here for brevity. Unless otherwise stated, all experiments were run on a system with an Intel Core i7-8700K CPU and 32GB RAM.

For neural network LAMP, we used the parameters discussed in Section III.B for all experiments. Clearly tuning the neural networks could produce improved results, however we wanted to demonstrate the generality of LAMP models and to show that they can work well "out of the box". Similarly, for Diverse Motifs LAMP we use a hard-coded diversity threshold of 0.95 unless otherwise noted.

In the following section we evaluate LAMP in the most direct way possible. Recall that the goal of LAMP is to predict the value that the much slower full Matrix Profile algorithm *would* have produced, thus we can both visualize and measure the difference between the OMP and LAMPs output. However, in some sense this is an indirect measurement for most practitioners. They typically only care about the classification or detection accuracy of their higher-level tasks which would exploit LAMP. Thus, in the remainder of the paper we will offer detailed case studies to demonstrate that LAMP can offer real-time performance even in challenging scenarios.

A. *LAMP method evaluation*

In Table 2, we compare the performance of various model types with a subsequence length of 100 on various architectures. The values in the table are measured in subsequences per second. The first two rows show the Diverse Motifs model with various settings for K. We did not implement these methods on the GPU, which is why no results are reported for the Tesla P100. For these models, our implementation was unbatched; it used only a single thread and processed just a single subsequence at a time.

Table 2: LAMP Inference Performance for m = 100

Method	NVIDIA Tesla P100	Desktop CPU I7-8700K	Raspberry Pi 3
Diverse Motifs Inference Rates (Hz)			
K = 1000	N/A	4852	434.8
K = 60000	N/A	403	16.9
Neural Network Inference Rates (Hz)			
J = 1, S = 1, Batch = 1	125	200	9.2
J = 32, S = 8, Batch = 1	51.2K	85.3K	2782
J = 32, S = 8, Batch = 128	482K	206K	5461

The bottom 3 rows show the results for our neural network scheme with various levels of batching. The first row is completely unbatched. The neural network is shown every subsequence individually and predicts a matrix profile value for each one. The second row uses the default settings that we presented in the Section III.B: with an input of 32 subsequences with stride 8, each inference produces 256 LAMP values. This enables increased efficiency and other advantages described in Section III.B. The last row uses a second level of batching where the neural network inputs are batched. Depending on how quickly decisions must be made, a user can choose a method of batching to suit their constraints. Differences in speed between single input and batched input are only because of the added data locality and dimensionality, which allow for exploiting multiprocessor and SIMD architectures. As mentioned previously, LAMP model inference time is dataset agnostic, depending only on the model size and input size.

We note that the neural network is much more efficient in general, due in part to a multitude of optimizations implemented by the Tensorflow developers and the deep learning community at large. Given the resources, a more efficient solution to inference with diverse motifs could be developed. However, it is also true that we have not actively optimized the neural network for any particular inference task. Depending on the dataset and other parameters of the problem, it might be possible to also make the neural network significantly faster via speed/accuracy tradeoffs. For example, adjusting the extraction stride S, applying quantization [25] or resource-constrained structure learning frameworks such as Morphnet [3]. We defer such an investigation to future work.

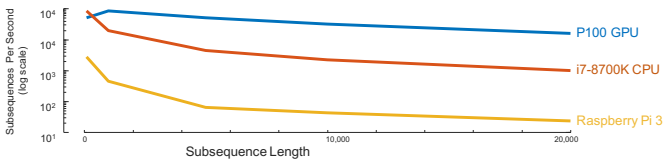


Fig. 8: Tradeoff between input subsequence length and inference rate for our neural network method on three different architectures.

It is also important to note that because we are extracting subsequences and sending them to the model for inference, the subsequence length parameter influences both the size of the models and inference time, as more FLOPS are required to perform a single inference using larger subsequences. Fig. 8 illustrates this tradeoff for our default neural network method defined in Section III.B.

Table 3: Comparison of LAMP model performance to Oracle

Dataset		Correlation to Oracle (OMP)			
Name	Train/Test Split	Exact RMP	Neural Network	Diverse Motifs	
				K	Correlation
Earthquake	20M/10M	.965	.887	60179	.731
Street Mall	59K/17K	.986	.690	128	.615
Insect EPG	2.5M/5M	.973	.959	11602	.625

Table 3 illustrates how well our model fits the oracle for various datasets. The RMP's performance can be viewed as a performance measure of the training data. A perfect LAMP model would achieve performance similar to the RMP. Note that as mentioned previously, this is an indirect measurement, as practitioners will be mostly concerned with classification or detection accuracy, which we discuss in the following sections. As mentioned before, there is room for improvement here via

parameter tuning, but we have explicitly kept a single set of parameters for Neural Network LAMP to show its robustness. The performance variation for the street mall dataset might be addressed via parameter tuning or the addition of more training data. For the Diverse Motifs model, we report the number of motifs used for that particular dataset. Due to the sensitivity of this parameter and the differences in the size of each of the datasets we evaluated, we could not achieve acceptable performance with the same K across the board.

Fig. 9 shows a visual comparison of the performance of various LAMP models on a snippet of the seismogram dataset from TABLE 3. Note the smoothness of the neural network model and the improvement of the matches as K is increased for the diverse motifs model. For this figure, the diverse motif models were generated using a diversity threshold of 0.85, the neural network settings were set to the default.

We have not reported training times for our experiments; however, most of the Neural Network LAMP models were quickly trainable in under an hour or two on a Tesla P100 GPU. The only exception to this is the large dataset presented in the next section, which took approximately 1 day to train on the GPU.

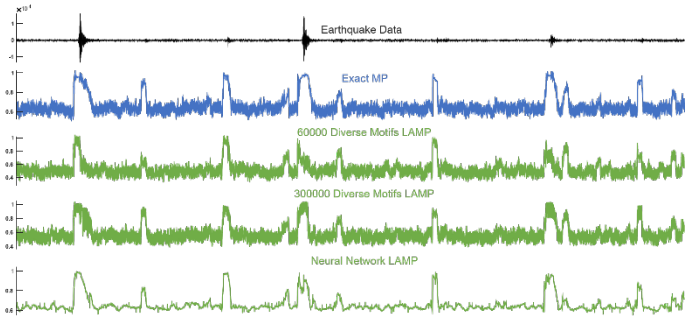


Fig. 9: Visual Comparison of LAMP methods for a snippet of earthquake data

B. *Case Study in Seismology*

Real-time seismic event detection is a primary task in seismology that has a direct impact on earthquake physics, fault behavior and seismic hazard assessment studies [1][22][15]. Most modern seismic networks have implemented real-time streaming of data from their remotely installed instruments to their seismic observatories. There, real-time event detection methods are used to monitor earthquake activity and provide basic information on event occurrence, timing, and magnitude. This can have a direct impact on seismic hazard assessment and response and early warning systems [1][15].

One of the most common real-time event detection methods is the short-term-average/long-term-average (STA/LTA) method, which computes the ratio between the average seismogram amplitude over a short time window and a longer time window. We expect that an earthquake will cause a sudden increase in the amplitude of the seismic waveform in the short term, leading to a spike in the ratio [24]. This method is widely used due to the ease of implementation and is effective at detecting large and/or close-by events. However, it can fail to detect smaller or more distant earthquakes, whose average waveform amplitudes are close to the noise floor. As a result, we speculate that many small events are missing from seismic event catalogs. Finding these small events is of particular significance in the seismology domain as they have a direct impact on studies of seismic triggering, short-term earthquake forecasting, foreshock and aftershock behaviors, etc. [4].

In some cases, seismologists apply more sensitive detection methods to 'mine' these smaller events from the continuous waveform data. One recent example [20], used query search ('matched filtering' in the terminology used by seismologists) to identify an order of magnitude more small events than had been detected using traditional methods in southern California. Such efforts show the power of a more sophisticated approach, although this improvement in sensitivity is not without cost – in the southern California case, the necessary computation required many hundreds of thousands of GPU hours [20]. Another limitation is that such methods use queries ('template waveforms' in seismology) from the existing seismic catalog in order to search for more events, leaving the possibility of a remaining population of undetected earthquakes for which there are no appropriate queries in the catalog.

We argue that LAMP is a potential solution for sensitive, rapid and inexpensive real-time seismic event detection. A common sample rate for local earthquake detection is 100 Hz, which is in the range of sample rates for which LAMP can produce the MP for the stream of seismic data in real-time, even on a relatively inexpensive device.

Note that there are many machine learning-based methods that have been proposed for earthquake event detection in recent years [21][30]. These methods are usually trained using existing catalogs, and are based on classified earthquake-noise training data sets. The training data set in this case might pass on the insensitivity of the catalogs to the models. LAMP is trained using the exact MP calculated from one year of data. The MP for one year of data is very sensitive to earthquake occurrence and can increase the number of event detections by ~16 times [34]. Rather than the binary classification of earthquake and non-earthquake (noise), LAMP weights waveforms based on a range of MP values (e.g. ~0.5 to 1), based on their similarity to other events.

Here we test LAMP for a real seismic waveform data set and compare the results with the existing catalog of earthquakes. We use a data set from a sensitive, low-noise, borehole seismic station (station name 'VCAB', network ID 'BP') near Parkfield in central California, close to a segment of the San Andreas fault where earthquakes occur frequently.

TABLE 4: COMPARISON OF DETECTION RATES OF VARIOUS TYPES OF SEISMIC EVENTS FOR VARIOUS DETECTION THRESHOLDS

	Threshold	W0 (%)	W1 (%)	W2 (%)	W3 (%)	W4 (%)	Total (%)	"False" Positives (%)
conservative	0.95	76	37	34	27	15	61	0.56
::	0.90	89	65	60	54	32	79	1.77
::	0.85	95	80	76	70	42	89	3.94
liberal	0.8	98	90	88	83	47	94	7.75

We train a neural network LAMP model using a 20 percent contiguous sample of this exact MP that we obtain from the

author of [34] for 580 days (2003-11-28 to 2005-07-09) of 20Hz seismic data. We then use LAMP to estimate the MP for 5.5 years of data (from 2005-07-10 to 2011-01-01) for the same seismic station. The total inference time for this dataset of around 4 billion datapoints was approximately 20 minutes using the large batch inference configuration from TABLE 2 on a single GPU. By extrapolating the performance of [34]'s exact GPU implementation to 4 billion datapoints, this is a speedup of over four orders of magnitude.

We then use four different thresholds of 0.8, 0.85, 0.9 and 0.95 for detecting motifs. The smaller values are more liberal (sensitive), and more likely to include some false positives.

Then in TABLE 4 we compare our detection with earthquake information that we obtained from the Northern California Seismic Network (NCSN) catalog [17]. Here we use two different catalogs to validate the LAMP outputs. The first catalog contains events whose seismic signals have been observed and picked at the station of interest, either by human analysts or by event detection algorithms (e.g., the STA/LTA method). These seismic signal observations are reported with five different weights based on confidence (W0 to W4, from 'very strong detection' to 'weak detection'). In this work, we refer to this catalog as the 'event-station catalog'. The second catalog contains all detected earthquakes, whether or not they were observed at this station, and we refer to it as the 'event-only catalog'. In this 5.5-year period, there were 9546 events in the event-station catalog and 26255 in the event-only catalog. Note that the event-station catalog is a subset of the event-only catalog.

We list our true positive detection rates for four different MP thresholds and for different event-station weighted events in TABLE 4. In general, for the event-station catalog we detect 94 percent of all events and 98 percent of the W0 events using a threshold of 0.8. For the thresholds of 0.95, 0.9 and 0.85 we have a true positive rate of 76, 89 and 95 percent for strong detected events (weighed 0). This indicates that we had a very high true positive rates with respect to the event-station catalog. Fig. 10 shows an example of a detected event waveform and the predicted MP for that event.

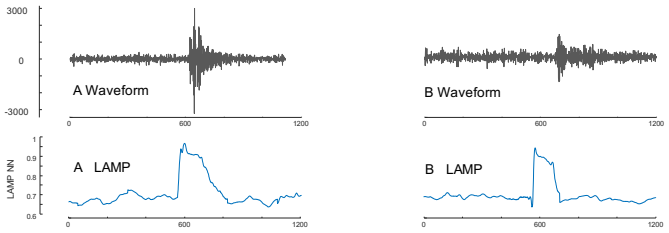


Fig. 10: a) Example of an event from the event-station catalog detected by LAMP. b) Example of an event detected by LAMP that was not in the event-station catalog but was in the event catalog.

One interesting thing that we observe by experimenting with LAMP on this data set is that when using the 0.9 threshold we detect 1962 events from the event-only catalog that are not in the event-station catalog. This could be because these events occur far from the station, and thus produce weak seismic signals that a human analyst or the STA/LTA method could not identify, but have sufficiently similar characteristics to other events that LAMP could identify. Fig. 10.*b* is one example of such an event.

After removing these ~2k events plus the true positives from the event-station catalog, we end up with 48454 detected motifs that are not associated with any catalog events (i.e. not in either catalog; our "false" positives from TABLE 4). By visually inspecting these detected motifs, we group them into four categories:

i) **Earthquake waveforms** for events missed by the catalog (Fig. 11.*a*).
ii) **Station glitches** (Fig. 11.*b*), which can be caused by voltage surges or the electromagnetic radiation from a lightning strike.
iii) **Station artifacts**, such as internal instrument calibration pulses (Fig. 11.*c*).
iv) **Harmonic noise**, possibly related to human activity or surface processes (Fig. 11.*d*). For example, a gust of wind or earthmoving equipment.

Clearly type (i) is the most exciting for seismologists, allowing them to populate their models and catalogs with additional examples that are currently missing.

Type (ii) and (iii) motifs can be easily removed from the data set by applying a simple query search using one of these instrumental errors as a query. Note that future LAMP models could be trained to ignore those signals. Type (iv) motifs can potentially be investigated by using LAMP on several stations to constrain their locations, which may be diagnostic of the source (e.g. a source located in the ocean might be caused by ocean waves and storms; a source located at the land surface could be weather or human-mediated).

Approximately 5% of the motifs discovered do not fit into this classification and are currently being investigated.

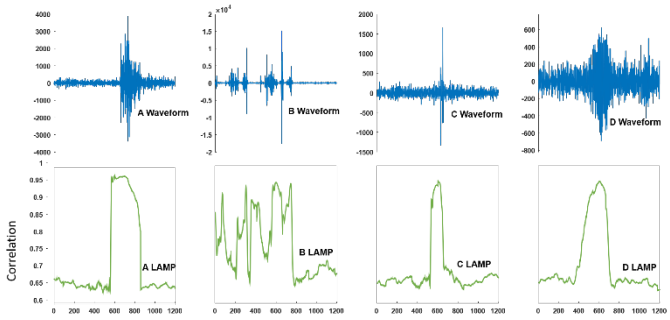


Fig. 11: Examples of various non-catalog events detected by LAMP. a) Earthquake not in any catalog b) Station glitches c) Station artifacts d) Harmonic noise.

C. *Case Study in Entomology*

Across the world, there are hundreds of species of insects that feed by ingesting plant fluids. Some of these insects can cause damage to their host plants by transmitting pathogens. As a concrete example, the Asian citrus psyllid (*Diaphorina citri*) shown in Fig. 12.*left* is a vector of the pathogen causing huánglóngbìng (citrus greening disease), which has already caused billions of dollars of damage to Florida's citrus industry in recent years, and is poised to do more damage worldwide. To design effective interventions, entomologists worldwide are attempting to understand the feeding behavior of such insects.

As Fig. 12 hints at, one of the most important tools used to study such insects is an EPG apparatus, which records the insects behavior as a one-dimensional time series [28].

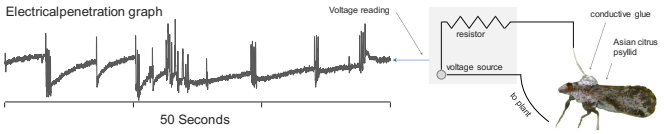


Fig. 12: *left*) Fifty seconds of data collected from an EPG apparatus (*center*), which measures the changes in resistance as an insect interacts with a host plant. This data comes from a psyllid (*right*).

Dozens of labs worldwide collect such data, but to the best of our knowledge, all analytics are conducted in post-hoc batch sessions, missing the opportunity to test hypotheses in real time. For example, a recent paper suggests that the Asian citrus psyllid changes its behavior in response to some "*combination of long and short wavelengths*" [19]. Other research has suggested that various cocktails of volatile organic compounds can modify their behavior [2]. With such a huge search space of optical and semiochemicals parameters progress in designing interventions has been slow. Researchers have resorted to making a single change each session then adjusting the intervention for next day's session. However, if we could measure the behavior the insects in *real-time*, the entomologists could adaptively tune the optical and/or chemical mixture to optimize its effectiveness. Below we will show that LAMP makes this possible.

We consider a dataset of insect behavior that records an Asian citrus psyllid feeding on a *Citrus natsudaidai* (a type of orange). We took the first seven hours of data (2,500,000 datapoints), and using the annotations of [28], we created two classes:

Class **A**: `Xylem Ingestion/Stylet Passage` (181 min)

Class **B**: `Non-Probing` (235 min)

Note that as the data snippets shown in Fig. 1 and Fig. 12 hint at, the data here is very complex and noisy. Moreover, each class is polymorphic: **A** features two different stages of a feeding behavior and class **B** is something of a "catch-all" [28]. For our testing data we consider 1,013 minutes of data, collected from the same insect in a later session. The class balance in that session happens to be almost equal, whereas the class balance in the training data is more skewed, at about five to one.

We can use a combination of multiple LAMP models to create a nearest neighbor classifier. Given a representative dataset of class **A** (R_A), another from class **B** (R_B), and stream of EPG data (S), we can train two separate LAMP models. The first LAMP model, M_A, is trained to approximate the RMP of S where matches can only occur in R_A. The second model, M_B, is trained to approximate the RMP of S where matches can only occur in R_B. Given a new subsequence **I** from S, we can produce the output of M_A and M_B when they are shown I. We can then use the class represented by the model that generated the largest value as the label for I. Table 5 shows the results for EPG classification across all models.

TABLE 5: COMPARISON OF EPG CLASSIFICATION RESULTS

Method	Accuracy (%)
Exact RMP	97.7
LAMP Diverse Motifs (K = 1600)	86.5
LAMP Neural Network	97.8
Direct Neural Network Classifier	99.2

Note that the neural network performs very slightly better than the exact RMP for the same task, but the difference is not statistically significant. Note also that we have trained a direct classifier using the same neural network used for LAMP but minimizing the binary cross entropy of the predicted labels versus the true labels. As expected, this classifier performs better than a LAMP NN-classifier, as LAMP is not trained directly for classification, However, LAMP remains competitive.

Fig. 13 shows how the accuracy of the diverse motifs method improves as K is increased. Note that the tradeoff between model size, efficiency, and accuracy is not always clear cut.

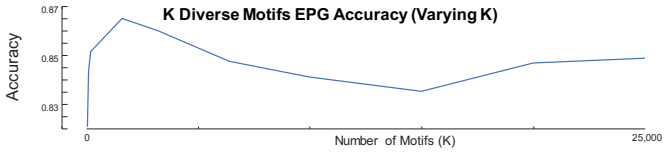


Fig. 13: Effect on accuracy of varying the number of diverse motifs in the LAMP model.

D. Case Study on Pedestrian Traffic

The two previous case studies highlight the use of LAMP to predict *high* correlations, which are indicative of conserved structure. However, as we noted above, LAMP also predicts *low* correlations, which can be indicative of anomalies. To test the utility of LAMP in this context, we conducted the following experiment. As shown in Fig 14.*top*, we consider pedestrian traffic data from Bourke Street in Melbourne. We trained LAMP on 6.7 years of such data, beginning at 04/30/2009. For test data, we consider the following two years. While the test data surely has natural "anomalies" (usual weather/cultural events), to have some ground truth we embedded three synthetic anomalies:

- **Reversed**: A week of data was flipped backwards.
- **Replaced**: A week of data was replaced by a week of data from a different location in Melbourne (Southbank).
- **Diminished**: We simulated a sensor that slowly began to undercount over a week.

The first two anomalies are so subtle that they defy human inspection (Fig 14.*top*), and the third happens so slowly that examining only a few days at a time, it would be impossible to detect. Nevertheless, as Fig 14.*bottom* shows, a LAMP model with *m* = three days is able to correctly detect each of our three anomalies.

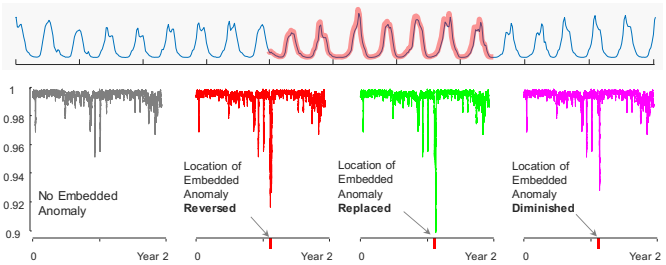


Fig 14: *top*) About 2% of the **Reversed** test dataset with embedded anomaly highlighted. *bottom*: *left* to *right*) The MP predicted by LAMP on two years of data with: no anomaly, the **Reversed** anomaly, the **Replaced** anomaly, and the **Diminished** anomaly. All three anomalous datasets have a significant dip in the LAMP output at the appropriate location.

E. When can LAMP fail?

The results above offer evidence that the LAMP framework can be useful in diverse settings, for diverse domains. Nevertheless, it is instructive to consider situations in which it can fail. LAMP very clearly can fail in the presence of concept drift, new motifs that have never been seen before may arise from new underlying processes, and something that *was* a motif before may become an anomaly in the future (and vice-versa).

We defer a detailed discussion on retraining of LAMP to mitigate the effects of concept drift to future work. However, one simple way it can be done is to keep track of the last segment of observed data and use that to augment the representative dataset used to train LAMP. Every so often (or constantly in the background) we can retrain LAMP based on this augmented training data, and when the new model is ready, we can hot-swap the old model with the new and continue our inference.

V. Conclusion

We introduced LAMP, a flexible and generic framework that allow us to approximate the Matrix Profile values in the face of fast-moving streams. Because the Matrix Profile is at the heart of many time series algorithms for classification [23], motif discovery [29], anomaly detection [31], segmentation [10] etc., LAMP allows such higher-level algorithms to be used in real-time settings on fast moving streams that are currently untenable with the standard Matrix Profile.

VI. Acknowlegements

This work was supported by NSF awards 1528181, 1544969, 1631776, 1763795, as well as Google, Mitsubishi and NetApp.

References

[1] Allen, R. M., Brown, H., Hellweg, M., Khainovski, O., Lombard, P., & Neuhauser, D. (2009). Real-time earthquake detection and hazard assessment by ElarmS across California. *Geophysical Research Letters*, *36*(5).

[2] Alquézar, Berta et al. "β-caryophyllene emitted from a transgenic Arabidopsis or chemical dispenser repels Diaphorina citri, vector of Candidatus Liberibacters." Scientific reports vol. 7,1 5639. 17 Jul. 2017, doi:10.1038/s41598-017-06119-w

[3] Ariel, G., et al. "Morphnet: Fast & simple resource-constrained structure learning of deep networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[4] Brodsky, E. E. (2019). The importance of studying small earthquakes. *Science*, *364*(6442), 736-737.

[5] Brown, A.E., et. al. A dictionary of behavioral motifs reveals clusters of genes affecting C. elegans locomotion. Proc. Natl. Acad. Sci. 110.2 (2013): 791-796

[6] Carrera, D. Rossi, B. Fragneto, P and Boracchi, G. Domain adaptation for online ecg monitoring. In Proceedings of the IEEE ICDM, pages 775–780, 2017.

[7] Diederik P., and Ba. J.n "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*(2014).

[8] Echihabi, K. Zoumpatianos, K. Palpanas, T. and Benbrahim, H. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. Proceedings of the VLDB Endowment (PVLDB) Journal, 2019.

[9] Eravci, B., Ferhatosmanoglu, H. Diverse Relevance Feedback for Time Series with Autoencoder Based Summarizations. IEEE Trans. Knowl. Data Eng. 30(12): 2298-2311 (2018)

[10] Gharghabi, S. et. al. Domain agnostic online semantic segmentation for multi-dimensional time series. Data Min. Knowl. Discov. 33(1): 96-130 (2019)

[11] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[12] Kolb, I., et. al. Evidence for long-timescale patterns of synaptic inputs in CA1 of mice. Journal of Neuroscience 38.7(2018): 1821-1834.

[13] LAMP supporting webpage: https://sites.google.com/view/lamp2019

[14] Lomax, A., Satriano, C., & Vassallo, M. (2012). Automatic picker developments and optimization: FilterPicker—A robust, broadband picker for real-time seismic monitoring and earthquake early warning. *Seismological Research Letters*, *83*(3), 531-540.

[15] Minson, S., Meier, M., Baltay, A., Hanks, T., and Cochran, S. "The limits of earthquake early warning: Timeliness of ground motion estimates," Sci. Adv., vol. 4, no. 3, 2018, Art. no. eaaq0504.

[16] Mueen, A., Nath, S., and Liu, J. Fast approximate correlation for massive time-series data. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. pp, 171-182.

[17] Northern California Seismic Network catalog. URL, reterived May 20th 2019. NCSN; http://www.ncedc.org/ncsn/

[18] Oyekanlu, E. "Predictive edge computing for time series of industrial IoT and large scale critical infrastructure based on open-source software analytic of big data". Proc. IEEE Int. Conf. Big Data 2017, pp. 1663–69.

[19] Paris T, Allan S, Udell B, Stansly P. Evidence of behavior-based utilization by the Asian citrus psyllid of a combination of UV and green or yellow wavelengths. PLoS One. 2017 Dec 13;12(12).

[20] Ross, Z. E., Trugman, D. T., Hauksson, E., & Shearer, P. M. (2019). Searching for hidden earthquakes in Southern California. *Science*, *364*(6442), 767-771.

[21] Ross, Z., Meier, M., and Hauksson, E. "P wave arrival picking and first-motion polarity determination with deep learning," J. Geophys. Res., vol. 123, pp. 5120–5129, Jun. 2018.

[22] Satriano, C., Lomax, A., & Zollo, A. (2008). Real-time evolutionary earthquake location for seismic early warning. *Bulletin of the Seismological Society of America*, *98*(3), 1482-1494.

[23] Senobari N. et al. Super Efficient Cross Correlation (SECC): A Fast Matched Filtering Code Suitable for Desktop Computers. Seismological Research Letters November 07, 2018, Vol.90, 322-334.

[24] Sharma, B.., Kumar, A., & Murthy, V. (2010). Evaluation of seismic events detection algorithms. *Journal of the Geological Society of India*, *75*(3), 533-538.

[25] Song, H.,Mao, H., and Dally, W. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." *arXiv preprint arXiv:1510.00149* (2015).

[26] Szigeti, B., Deogade, A. and Webb, B. Searching for motifs in the behaviour of larval Drosophila melanogaster and Caenorhabditis elegans reveals continuity between behavioural states. Journal of the Royal Society Interface 12.113 (2015): 20150899.

[27] Wang, Z, Weizhong Y, and Oates. T. "Time series classification from scratch with deep neural networks: A strong baseline." 2017 International joint conference on neural networks (IJCNN).

[28] Willett, D., George, J., Willett, N. Stelinski, L. and Lapointe, S. Machine learning for characterization of insect vector feeding. PLoS computational biology, 12.11 (2016): e1005158.

[29] Yeh, M., et. al. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. IEEE ICDM 2016: 1317-1322.

[30] Zhu W, and Beroza, G. "PhaseNet: A deep-neural-network-based seismic arrival-time picking method," Geophys. J. Int., vol. 216, no. 1, pp. 261–273, 2019.

[31] Zhu, Y., et. al. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016: 739-748.

[32] Zhu, Y., Imamura, M., Nikovski, D., and Keogh, E. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. IEEE ICDM 2017: 695-704.

[33] Zhu, Y., Yeh, M., Zimmerman, Z., Kamgar, K., and Keogh, E. "Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds". ICDM 2018: 837-846.

[34] Zimmerman. Z., et al. Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond ACM SoCC 2019. In Press.