Dynamic Modeling and Forecasting of Time-evolving Data Streams

Yasuko Matsubara*†
ISIR, Osaka University
yasuko@sanken.osaka-u.ac.jp

Yasushi Sakurai*
ISIR, Osaka University
yasushi@sanken.osaka-u.ac.jp

ABSTRACT

Given a large, semi-infinite collection of co-evolving data sequences (e.g., IoT/sensor streams), which contains multiple distinct dynamic time-series patterns, our aim is to incrementally monitor current dynamic patterns and forecast future behavior. We present an intuitive model, namely OrbitMap, which provides a good summary of time-series evolution in streams. We also propose a scalable and effective algorithm for fitting and forecasting time-series data streams. Our method is designed as a dynamic, interactive and flexible system, and is based on latent non-linear differential equations. Our proposed method has the following advantages: (a) It is *effective*: it captures important time-evolving patterns in data streams and enables real-time, long-range forecasting; (b) It is *general*: our model is general and practical and can be applied to various types of time-evolving data streams; (c) It is *scalable*: our algorithm does not depend on data size, and thus is applicable to very large sequences. Extensive experiments on real datasets demonstrate that OrbitMap makes long-range forecasts, and consistently outperforms the best existing state-of-the-art methods as regards accuracy and execution speed.

ACM Reference Format:
Yasuko Matsubara and Yasushi Sakurai. 2019. Dynamic Modeling and Forecasting of Time-evolving Data Streams . In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330947

1 INTRODUCTION

Today, a massive amount of time-stamped data is generated and collected by many advanced technologies and services, including the Internet of Things (IoT) [8, 12], Web-based online data-driven marketing [25, 35, 40], and more [36, 39]. One of the most fundamental demands for data science and engineering is the efficient and effective analysis of big time-series data streams, such as real-time, long-term forecasting without human intervention.

In practice, real data streams contain various types of distinct, dynamic time-series patterns of different durations, and these patterns usually involve latent relationships with each other. That is, there are some kinds of hidden rules, or dynamic time-evolving trends in streams. For example, with an IoT machine monitoring system, we will probably observe some indicators or signs before potential accidents or equipment failure. Similarly, as regards a human motion sensor data stream, we can observe multiple distinct patterns (e.g., stretching, jogging, cooling down, resting), and these patterns sometimes have dynamic transitions (e.g., jogging → cooling down → resting). It is essential to capture such dynamic patterns and their relationships in streams if we are to forecast future activities. Here, we refer to such distinct dynamic time-series patterns as "regimes". We also refer to latent relationships between regimes as "dynamic space transitions".

Our aim is to monitor a semi-infinite collection of time-evolving sequences, provide a good summary of time-series evolution in streams and forecast long-term future values. We present OrbitMap [1], an intuitive model capable of dealing with the above tasks. OrbitMap is designed to be a dynamic, interactive and flexible system, and is based on latent non-linear differential equations. Informally, the problem we wish to solve is as follows:

Informal Problem 1. **Given** *a data stream X of length t_c, which consists of d-dimensional vectors, i.e., $X = \{\boldsymbol{x}(1), \ldots, \boldsymbol{x}(t_c)\}$, where t_c is the current time point,* **Forecast** *an l_s-steps-ahead future value, and more specifically,*

- *find major dynamic time-series patterns (i.e., regimes),*
- *find relationships between regimes (i.e., dynamic space transitions),*
- *report an l_s-steps-ahead future value, i.e., $\boldsymbol{e}(t_c + l_s)$,*

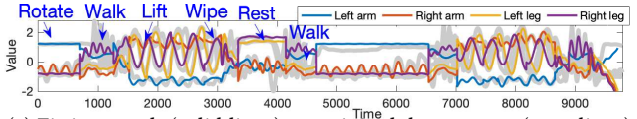
incrementally and quickly, at any point in time.

Preview of our results. Figure 1 shows the results we obtained using OrbitMap with real human motion sensors. This dataset consists of $d = 4$ dimensional event entries, which are collected by four acceleration sensors (100 Hz), mounted on the right/left legs and arms of a worker in a factory. Figure 1 (a) shows our fitting result (solid colored lines) for the original data stream (gray lines). The data stream was composed of several distinct steps such as "rotating" and "walking". Our fit is even visually very good, and captures dynamic time-evolving patterns. Given a sensor stream X, our goal is to (G1) identify current regimes and (G2) their transitions in the stream, and (G3) forecast l_s-steps-ahead future values, continuously and automatically. Each goal is described below.
(G1) Regime identification and segmentation: Figure 1 (b) shows snapshots of real-time regime identification and segmentation at two different time points. Here, the vertical axes in the figure show

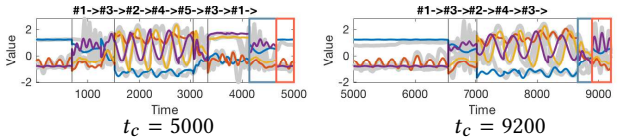
*Artificial Intelligence Research Center, The Institute of Scientific and Industrial Research (ISIR), Osaka University, Mihogaoka 8-1, Ibaraki, Osaka 567-0047, Japan
†Presently with JST, PRESTO

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330947

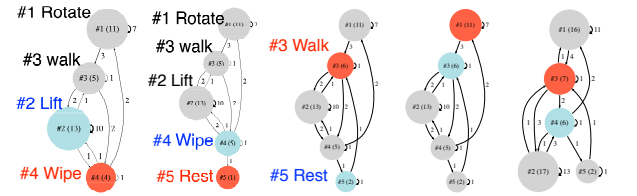


(a) Fitting result (solid lines) vs. original data stream (gray lines)



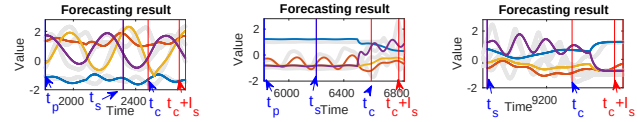
$t_c = 5000$ $t_c = 9200$

(b) Snapshots of real-time regime identification and segmentation



$t_c = 3320$ $t_c = 3840$ $t_c = 4640$ $t_c = 4920$ $t_c = 9040$

(c) Snapshots of dynamic space transitions at different time points



$t_c = 2480$ $t_c = 6600$ $t_c = 9320$

(d) Snapshots of l_s = 200-steps-ahead future value forecasting

Figure 1: Modeling power of OrbitMap for a motion sensor stream: It continuously and automatically finds dynamic patterns and forecasts l_s-steps-ahead future values. (a) Our model fits the original stream very well, and (b) it identifies regimes and their transition points, i.e., segments. (c) It also finds dynamic space transitions at each time point. (d) Snapshots of l_s-steps-ahead future value prediction at three different time points, where, the red vertical axes $t_c, t_c + l_s$ show the current and l_s-steps-ahead time points, respectively.

the transition points of each regime. Our method can incrementally and automatically identify typical regimes (e.g., #1 rotating, #3 walking, #2 lifting) and their transition points (i.e., segments) in a given sensor stream.

(G2) Dynamic space transitions: Figure 1 (c) shows snapshots of the dynamic relationship network between regimes, where, each node represents each regime contained in a data stream, and each edge between nodes indicates that there is a transition between two regimes. A larger node indicates a more frequently appearing regime, and a thicker edge indicates a stronger relationship. The leftmost figure shows the transition network at time point $t_c = 3320$, consisting of four regimes, and the middle and right figures show how it grows at different time points. In addition, the blue and red nodes show the two most recent regimes at the current time point, e.g., in the leftmost figure, the blue node corresponds to #2 lifting, while the red node corresponds to #4 wiping. Most importantly, the transition network evolves over time as a new regime appears in the stream (e.g., #5 resting at $t_c = 3840$).

(G3) l_s-steps-ahead future value prediction: Figure 1 (d) shows snapshots of the l_s = 200-steps-ahead future forecasting at three different time points. Our estimated variables are shown by bold colored lines (the originals are shown in gray), blue vertical axes t_p, t_s show the transition points of recent regimes, and red vertical axes $t_c, t_c + l_s$ show the current and l_s-steps-ahead time points, respectively (please see Figure 3 for a detailed explanation of the axes). As shown in the figure, our method can predict upcoming regime transitions, and generate long-term future values, incrementally, continuously and automatically. Here, we emphasize that unlike the other existing dynamic modeling approaches, e.g., Markov chain models and Bayesian networks, which are based on discrete and stochastic modeling, our model captures dynamic, continuous and numerical time-series patterns by using latent non-linear differential equations. We will explain the functions of dynamic space transitions in detail in Figure 2.

Table 1: Capabilities of approaches.

	HMM/++	pHMM	AutoPlait	SMiLer/++	SARIMA/++	RegimeCast	LSTM/GRU	OrbitMap
Data Stream	-	-	-	✓	-	✓	-	✓
Data compression	✓	✓	✓	-	-	✓	-	✓
Segmentation	-	✓	✓	-	-	-	-	✓
Dynamic space transition	-	-	-	-	-	-	-	✓
Deterministic	-	-	-	✓	✓	✓	-	✓
Forecasting	-	-	-	✓	✓	✓	✓	✓

Contributions. In this paper, we focus on a challenging problem, namely, the dynamic modeling and forecasting of time-evolving data streams, and present OrbitMap, which has the following desirable properties:

(1) **Effective**: it finds important time-evolving patterns (i.e., regimes) and their latent relationships (i.e., dynamic space transitions) in data streams, and performs long-range forecasts.
(2) **General**: we apply OrbitMap to various types of time-evolving data streams including sensors and Web activities.
(3) **Scalable**: it requires a constant time per time point for fitting and forecasting data streams.

We also perform our experiments on real IoT data streams in smart factories, and demonstrate the practicality and effectiveness of our method (please see section 6).

Outline. The rest of the paper is organized in the conventional way: next we describe related work, before moving on to our proposed model, algorithms, experiments and conclusions.

2 RELATED WORK

Time-series data analysis is an important topic that has attracted huge interest in countless fields such as social activity mining [21, 25, 27, 40], online text [15, 16], medical analysis [7, 10, 11, 28, 39], sensor network monitoring [13, 31, 32, 37], and more [6, 14, 19, 36]. Recent studies have focused on non-linear time-series analysis with the aim of understanding the dynamics of IoT data streams and social media [21, 22, 24, 25, 27].

Table 1 illustrates the relative advantages of our method. Only OrbitMap meets all requirements. Hidden Markov models (HMM) and other dynamic statistical models are capable of compression and capturing time-evolving patterns in time sequences. Similarly, a Bayesian network (BN) [33, 34] is a probabilistic model designed to represent the conditional dependencies of different variables.

However, these approaches are *stochastic* (as opposed to *deterministic*) and discrete, and thus cannot describe dynamic and continuous activities, or forecast future dynamic patterns. pHMM [38] and AutoPlait [23] are based on HMMs, and have the ability to capture the dynamics of sequences and perform segmentation, however, they are not designed to capture long-range non-linear evolutions of co-evolving data streams. Data-driven, non-linear forecasting methods, such as SMiLer [41] and F4 [3] tend to provide results that are hard to interpret, and are incapable of modeling dynamic patterns in streams.

Traditional modeling and forecasting approaches typically use linear methods, such as autoregressive integrated moving average (ARIMA), linear dynamical systems (LDS), Kalman filters (KF) and their derivatives including AWSOM [30], TBATS [20], PLiF [18], TriMine [26] and more [6]. Note that these methods are fundamentally unsuitable for our setting; they are all based on *linear* equations, and are thus incapable of modeling data governed by non-linear equations. Similarly, the switching state space model, and the switching Kalman filter (SKF) model [2, 29] are designed as a combination of the hidden Markov model with a set of linear dynamical systems. It can handle multiple distinct patterns in time series, but is not intended to capture dynamic space transitions, and thus cannot model continuous and deterministic behavior between multiple regimes. RegimeCast [22] focuses on the real-time forecasting of event streams, but is not intended to perform regime identification and segmentation. Moreover, it cannot capture transitions between different dynamic patterns.

Deep learning has become one of the most popular methodologies in data analysis tasks [4, 5, 9, 42]. Recurrent neural networks (RNNs) encounter difficulties in modeling long-range dependencies. Long short-term memory (LSTM) and gated recurrent units (GRUs) alleviate this problem, however all DNN variants require a prohibitively high computation cost, especially training cost, for data stream analysis, which is hard to forecast in real time. In addition, most of them require sensitive parameter tuning.

In short, none of the existing methods focuses specifically on the modeling and forecasting of the non-linear dynamics of co-evolving multiple patterns in data streams.

3 PROPOSED MODEL

In this section, we present our proposed model. Assume that we receive a collection of time-evolving sequences, such as IoT/sensor data streams. As we mentioned in the introduction section, real data streams contain various types of distinct, dynamic time-evolving patterns of different durations. In this paper, we refer to such a dynamic time-evolving pattern as a *"regime"*. Also, these patterns (i.e., regimes) usually have latent relationships with each other, namely, *"dynamic space transitions"*. That is, we need to identify any sudden discontinuity in a given data stream, recognize the current regime immediately, and also capture dynamic space transitions between regimes, so that we can predict future dynamics, flexibly and adaptively, at any time. Consequently, we need to capture the following properties: (**P1**) regimes, i.e., time-evolving patterns and (**P2**) dynamic space transitions between multiple distinct regimes. So, what is the simplest mathematical model that can capture both (**P1**) and (**P2**)? We provide the answers below.

3.1 Proposed solution: OrbitMap

Figure 2 shows an illustration of our proposed model. Intuitively, we assume that there are multiple, distinct regimes in data streams, and they have dynamic space transitions. Here, we describe our model in details.

3.1.1 Time-evolving dynamics in a single regime (P1) . We begin with the first step (**P1**), where we have only a single regime.

Figure 2 (a) shows how our model evolves over time in a single regime. In short, our basic model has the following properties.

- A regime dynamical model $\boldsymbol{\theta}$: each regime has its own dynamic latent space $\boldsymbol{s}$ (shown as black/red arrows in a regime in Figure 2 (a)), and it can be described with non-linear differential equations.

In the figure, we only plot $k = 3$ dimensions for the visualization. In our model, we assume two classes of time-evolving sequences.

- Latent variables $\boldsymbol{s}(t)$, i.e., a k-dimensional latent vector at time point t (i.e., $\boldsymbol{s}(t) = \{s_i(t)\}_{i=1}^{k}$), which evolves over time as a dynamical system $\boldsymbol{\theta}$. We also refer to $S = \{\boldsymbol{s}(1), \cdots, \boldsymbol{s}(t)\}$ as a *"latent space trajectory"*, (i.e., *"orbit"*, shown as red arrows in the figure).
- Estimated variables $\boldsymbol{e}(t)$, i.e., a d-dimensional vector at time point t (i.e., $\boldsymbol{e}(t) = \{e_i(t)\}_{i=1}^{d}$), which can be computed by $\boldsymbol{s}(t)$. Also, let E be an estimated sequence, $E = \{\boldsymbol{e}(1), \cdots, \boldsymbol{e}(t)\}$.

Intuitively, E is an estimation of the original sequence $X = \{\boldsymbol{x}(1), \cdots, \boldsymbol{x}(t)\}$. E could be any kind of data sequence, e.g., a d-dimensional sequence generated by d sensors.

Also, the regime dynamical model $\boldsymbol{\theta}$ can generate a specific time-series data sequence with the initial condition vector $\boldsymbol{v}^{in}$.

- Sequence generation with the initial condition vector $\boldsymbol{v}^{in}$: given a regime dynamical model $\boldsymbol{\theta}$, and an initial condition vector $\boldsymbol{v}^{in}$, the system $\boldsymbol{\theta}$ generates latent/estimated sequences, i.e., $S = \{\boldsymbol{s}(1), \cdots, \boldsymbol{s}(t)\}$, $E = \{\boldsymbol{e}(1), \cdots, \boldsymbol{e}(t)\}$, where it has the condition $\boldsymbol{s}(1) = \boldsymbol{v}^{in}$. Here, $\boldsymbol{v}^{in}$ is a k-dimensional vector in the latent space.

Consequently, a single regime dynamical model can be described with the following equations:

Model 1. *Let $\boldsymbol{s}(t)$, $\boldsymbol{e}(t)$ be the latent/estimated variables at time point t. Our single regime is governed by the following equations,*

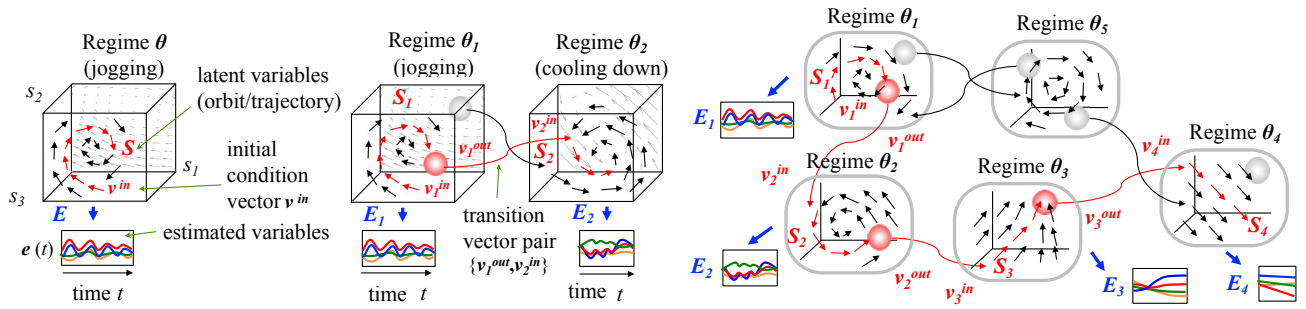
$$\frac{d\boldsymbol{s}(t)}{dt} = \boldsymbol{p} + \mathbf{Q}\boldsymbol{s}(t) + \mathcal{A}\mathrm{S}(t) \tag{1}$$

$$\boldsymbol{e}(t) = \boldsymbol{u} + \mathbf{V}\boldsymbol{s}(t) \tag{2}$$

with the initial condition, $\boldsymbol{s}(1) = \boldsymbol{v}^{in}$.

Note that $d\boldsymbol{s}(t)/dt$ is a derivative with respect to time t, and $\mathrm{S}(t)$ shows the quadratic form matrix of $\boldsymbol{s}(t)$, i.e., $\mathrm{S}(t) = \boldsymbol{s}(t)^T\boldsymbol{s}(t)$. Here, $\boldsymbol{p}$, $\mathbf{Q}$ and $\mathcal{A}$ describe the latent variables $\boldsymbol{s}(t)$, each of which captures the linear, exponential, and non-linear dynamics, respectively, while, $\boldsymbol{u}$, $\mathbf{V}$ show the observation projection, which generates the estimated variables $\boldsymbol{e}(t)$ at each time point t. Also, $\boldsymbol{v}^{in}$ is a k-dimensional initial condition vector in the latent space. Consequently, we have the following:

Definition 1 (Regime dynamical model: $\boldsymbol{\theta}$). *Let $\boldsymbol{\theta}$ be a parameter set of a single regime i.e., $\boldsymbol{\theta} = \{\boldsymbol{p}, \mathbf{Q}, \mathcal{A}, \boldsymbol{u}, \mathbf{V}\}$.*



(a) Dynamic activity of a single regime (**P1**)

(b) Dynamic space transition from regime θ_1 to regime θ_2 (**P2**)

(c) Dynamic space transitions between multiple regimes (i.e., regimes $\theta_1 \rightarrow \theta_2 \rightarrow \theta_3 \rightarrow \theta_4$)

Figure 2: Illustration of OrbitMap. (a) Given a single regime θ, and an initial condition vector $\boldsymbol{v}^{in}$, it generates latent variables $S = \{\boldsymbol{s}(t)\}_t$ (shown as red arrows), and estimated variables $E = \{\boldsymbol{e}(t)\}_t$, (here, $k = 3, d = 4$). (b) Given two regimes θ_1, θ_2, and a transition pair $\{\boldsymbol{v}_1^{out}, \boldsymbol{v}_2^{in}\}$, it generates two sets of estimated variables, $E_1 \rightarrow E_2$. (c) Given a set of regimes $\Theta = \{\theta_1, \cdots, \theta_5\}$, and their transition pairs $\mathcal{V}$, it causes multiple space transitions, and generates long-term patterns, i.e., $E_1 \rightarrow E_2 \rightarrow E_3 \rightarrow E_4$.

3.1.2 Dynamic space transitions between multiple regimes (P2). Thus, we have seen how to generate dynamic time-evolving sequences S, E, using a single regime $\boldsymbol{\theta}$. Now the question is, how can we generate dynamic transitions between multiple regimes, such as $\boldsymbol{\theta}_1$ jogging $\rightarrow \boldsymbol{\theta}_2$ cooling down $\rightarrow \boldsymbol{\theta}_3$ resting? Unlike the other existing discrete and stochastic modeling approaches, e.g., Markov chain models, we want to describe latent relationships between multiple distinct regimes in a stream, using a deterministic, numerical and continuous time-series modeling approach. We thus introduce an additional concept, namely, (**P2**) dynamic space transitions between regimes. Figure 2 (b) shows how OrbitMap generates two distinct sets of estimated variables (e.g., $E_1 \rightarrow E_2$). In short, we add a *"transition"* between two different regimes and connect two different dynamic latent spaces. It is analogous to the connection between a black hole and a white hole, where we can enter a black hole and emerge from a white hole at a different location.

In our model, each regime $\boldsymbol{\theta}_i$ has its specific transition vectors $\boldsymbol{v}_i^{in}, \boldsymbol{v}_i^{out}$ in the latent space, and it is connected with other regimes via the space transition network. To simplify the discussion, let us assume a single space transition between two regimes, $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$, and let $\{\boldsymbol{v}_i^{out}, \boldsymbol{v}_j^{in}\} \in \mathcal{V}$ be an out/in-transition vector (or, black/white hole) pair from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_j$. Intuitively, our model has the following transition rules:

- The regime $\boldsymbol{\theta}_i$ generates its own latent variables $\boldsymbol{s}_i(t)$ at each time point t, according to Model 1. It continues the process until the space transition from $\boldsymbol{\theta}_i$ to the other regime $\boldsymbol{\theta}_j$ occurs.
- If the trajectory/orbit of the latent variables S_i in $\boldsymbol{\theta}_i$ gets close to the transition vector $\boldsymbol{v}_i^{out}$ at time point t_s, it moves to $\boldsymbol{\theta}_j$, and generates the j-th regime's latent variables, starting with the initial condition $\boldsymbol{s}_j(t_s) = \boldsymbol{v}_j^{in}$. We refer to t_s as the transition point.

Here, the space transition from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_j$ occurs, only if the latent vector $\boldsymbol{s}_i(t_s)$ on the trajectory S_i is close enough to the vector $\boldsymbol{v}_i^{out}$ (i.e., the distance between $\boldsymbol{s}_i(t)$ and $\boldsymbol{v}_i^{out}$, $D(t) = ||\boldsymbol{s}_i(t) - \boldsymbol{v}_i^{out}|| \leq \rho$). Note that ρ is the transition strength, and we discuss this in the next section. Consequently, we have:

Model 2. *Let $\boldsymbol{s}(t)$ be the latent vector at time point t. The dynamic space transition from regime $\boldsymbol{\theta}_i$ to regime $\boldsymbol{\theta}_j$ can be described by the following equations,*

$$\boldsymbol{s}(t) = \begin{cases} \boldsymbol{s}_i(t) & (1 \leq t < t_s) \quad // \text{ staying in regime } \boldsymbol{\theta}_i \\ \boldsymbol{v}_j^{in} & (t = t_s) \quad // \text{ transition from } \boldsymbol{\theta}_i \text{ to } \boldsymbol{\theta}_j \\ \boldsymbol{s}_j(t) & (t_s < t) \quad // \text{ staying in regime } \boldsymbol{\theta}_j, \end{cases} \tag{3}$$

such that $t_s = \underset{t \mid D(t) \leq \rho}{\arg\min} D(t)$, where, $D(t) = ||\boldsymbol{s}_i(t) - \boldsymbol{v}_i^{out}||$ shows the distance between two vectors $\boldsymbol{s}_i(t)$ and $\boldsymbol{v}_i^{out}$. Here, $\boldsymbol{s}_i(t)$ is the latent vector generated by $\boldsymbol{\theta}_i$ at time point t, and $\{\boldsymbol{v}_i^{out}, \boldsymbol{v}_j^{in}\}$ is a transition vector pair from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_j$.

Definition 2 (Transition vector set: $\mathcal{V}$). *Let $\mathcal{V}$ be a parameter set of transition vector pairs $\{\boldsymbol{v}_i^{out}, \boldsymbol{v}_j^{in}\} \in \mathcal{V}$ $(i, j = 1, \cdots, r)$.*

In practice, there might be multiple out-vectors $\boldsymbol{v}_i^{out}$ on the same trajectory/orbit, and in this case, we choose the closest out-vector and move to the other regime. Also note that each regime pair may have more than one transition vector pair $\{\boldsymbol{v}_i^{out}, \boldsymbol{v}_j^{in}\}$ (as shown in Figure 2 (b), where the regime pair $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ has two different (red/black) transition arrows), and thus different trajectories may cause completely different space transitions.

Figure 2 (c) illustrates how OrbitMap generates long-term, multi-step space transitions. Given an initial vector $\boldsymbol{v}_1^{in}$, it causes space transitions (i.e., regimes $\boldsymbol{\theta}_1 \rightarrow \boldsymbol{\theta}_2 \rightarrow \boldsymbol{\theta}_3 \rightarrow \boldsymbol{\theta}_4$), where each regime generates a set of latent/estimated sequences (i.e., $\{S_1, E_1\}, \cdots, \{S_4, E_4\}$), according to Models 1 and 2.

Definition 3 (Full parameter set of OrbitMap: $\mathcal{M}$). *Let $\mathcal{M}$ be a complete set of OrbitMap parameters, namely, $\mathcal{M} = \{\Theta, \mathcal{V}\}$, where, Θ consists of r regimes, i.e., $\Theta = \{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_r\}$, and $\mathcal{V}$ is a transition set, i.e., $\{\boldsymbol{v}_i^{out}, \boldsymbol{v}_j^{in}\} \in \mathcal{V}$.*

4 STREAMING ALGORITHM

Thus far, we have shown how our model captures time-evolving patterns (i.e., regimes) and their latent relationships (i.e., dynamic space transitions) in a data stream X.

Table 2: Symbols and definitions.

Symbol	Definition
X	Time-evolving data stream, i.e., $X = \{\boldsymbol{x}(1), \ldots, \boldsymbol{x}(t)\}$
$\boldsymbol{x}(t)$	d-dimensional vector at time point t, i.e., $\boldsymbol{x}(t) = \{x_i(t)\}_{i=1}^{d}$
S	Latent variables, i.e., $S = \{\boldsymbol{s}(1), \cdots, \boldsymbol{s}(t))\}$
E	Estimated variables, i.e., $E = \{\boldsymbol{e}(1), \cdots, \boldsymbol{e}(t))\}$
Θ	Model parameter set of r regimes, i.e., $\Theta = \{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_r\}$
$\mathcal{V}$	Transition vector set, i.e., $\{\boldsymbol{v}_i^{out}, \boldsymbol{v}_j^{in}\} \in \mathcal{V}$
$\mathcal{M}$	Complete set of OrbitMap, i.e., $\mathcal{M} = \{\Theta, \mathcal{V}\}$

4.1 Overview

Our next tasks are: (a) determining a way to find an optimal parameter set $\mathcal{M} = \{\Theta, \mathcal{V}\}$ in a real data stream X, and (b) how to apply our model to real-time forecasting. The formal problem that we want to solve is as follows:

Problem 1. **Given** *a data stream* $X = \{\boldsymbol{x}(1), \ldots, \boldsymbol{x}(t_c)\}$, *where,* $\boldsymbol{x}(t_c)$ *is the most recent value at time point* t_c,
(a) find an optimal OrbitMap *parameter set, i.e.,* $\mathcal{M} = \{\Theta, \mathcal{V}\}$,
(b) report an l_s*-steps-ahead future value, i.e.,* $\boldsymbol{e}(t_c + l_s)$,
incrementally, as quickly as possible.

With respect to the first goal (a), we want to incrementally maintain the model parameter set $\mathcal{M} = \{\Theta, \mathcal{V}\}$ so that it captures regimes Θ and their transition vector pairs $\mathcal{V}$ in a data stream X. Our final goal is (b) to forecast long-term future patterns, i.e., simultaneously with a model update, we want to forecast the l_s-steps-ahead future value $\boldsymbol{e}(t_c + l_s)$, using our dynamic modeling framework.

Here, we introduce our streaming algorithm, namely, OrbitMap-F, which achieves the above goals. Our algorithm estimates the model parameters of regimes and extracts information regarding space transitions in X, and forecasts future values, simultaneously, in a streaming fashion. Figure 3 shows a snapshot of OrbitMap-F at time point t_c. Our dynamic modeling framework constructs the space transition network between regimes, which enables us to provide long-term forecasts with high accuracy, by following the paths of transitions.

Intuitively, the main idea behind our algorithm is to monitor X and keep track of two regimes, the previous regime $\boldsymbol{\theta}_p$ and the current candidate regime $\boldsymbol{\theta}_c$. Specifically, at every time point t_c, given a data stream X, it monitors the current segment $X_c = X[t_s : t_c]$, (i.e., recent subsequence of X), which is assigned to regime $\boldsymbol{\theta}_c$. While checking the end point of X_c (i.e., whether or not the current regime $\boldsymbol{\theta}_c$ should be replaced by another regime, e.g., $\boldsymbol{\theta}_f$), the algorithm estimates the candidate regime $\boldsymbol{\theta}_c$ and its transition vectors between $\boldsymbol{\theta}_p$ and $\boldsymbol{\theta}_c$ (i.e., $\boldsymbol{v}_p^{out}, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}$), and keeps them as the model candidate $C = \{\boldsymbol{\theta}_c, \boldsymbol{v}_p^{out}, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\}$ for stream processing. It also forecast the l_s-steps-ahead future value $\boldsymbol{e}(t_c + l_s)$ using $\mathcal{M}$ and C, according to Models 1, 2.

Parameter estimation for a single regime $\boldsymbol{\theta}_c$. Here, the algorithm estimates optimal parameters so that it minimizes the fitting error between the original segment $X_c = X[t_s : t_c]$ and the estimated variables $E_c = E[t_s : t_c]$, generated by $\{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\}$, using Model 1. That is, we have,

$$\{\boldsymbol{\theta}_c^*, \boldsymbol{v}_c^{in*}, \boldsymbol{v}_c^{out*}\} = \underset{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}}{\arg\min} \; f_D(X_c; \boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}), \tag{4}$$

where, $f_D(\cdot)$ shows the fitting error between the original and estimated variables, i.e., $f_D(X_c; \boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}) = \sum_{t=t_s}^{t_c} ||\boldsymbol{x}(t) - \boldsymbol{e}(t)||$. We use the Levenberg-Marquardt (LM) algorithm and the Runge-Kutta method [17] to estimate the parameters. Note that, vector pair $\boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}$ corresponds to the latent variables at time points t_s and t_c, respectively, i.e., $\boldsymbol{v}_c^{in} = \boldsymbol{s}(t_s)$, $\boldsymbol{v}_c^{out} = \boldsymbol{s}(t_c)$.

Next we describe our algorithms in detail.

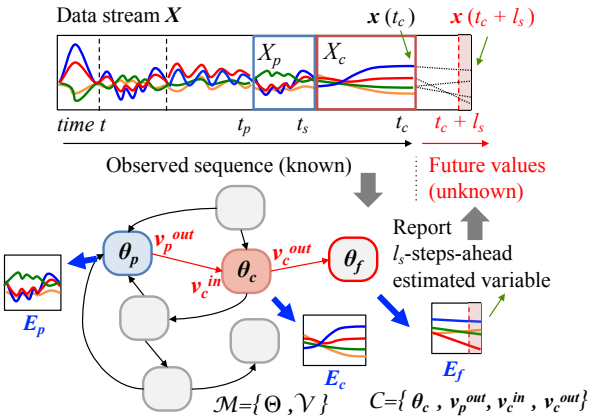


Figure 3: Illustration of OrbitMap-F. Given a data stream X, model parameter set $\mathcal{M} = \{\Theta, \mathcal{V}\}$, and model candidate C, it updates $\mathcal{M}$ and C at every time point t_c. The vertical axes in X show the transition points of each regime. It also forecasts future values $\boldsymbol{e}(t_c + l_s)$ using $\mathcal{M}$ and C.

Algorithm 1 OrbitMap-F $(x(t_c), \mathcal{M}, C)$

1: **Input:** (a) New value $x(t_c)$ at time point t_c
(b) Current OrbitMap parameter set $\mathcal{M} = \{\Theta, \mathcal{V}\}$
(c) Model candidate $C = \{\boldsymbol{\theta}_c, \boldsymbol{v}_p^{out}, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\}$
2: **Output:** (a) Updated OrbitMap parameter set $\mathcal{M}'$
(b) Updated model candidate C'
(c) l_s-steps-ahead future value $\boldsymbol{e}(t_c + l_s)$
3: /* (I) OrbitMap estimation and segmentation */
4: $\{\mathcal{M}', C'\}$ = O-Estimator$(x(t_c), \mathcal{M}, C)$;
5: /* (II) l_s-steps-ahead future value forecasting */
6: $\boldsymbol{e}(t_c + l_s)$ = O-Generator$(\mathcal{M}', C', l_s)$;
7: /* (III) OrbitMap feedback (if required) */
8: $\mathcal{M}'$ = O-Feedback$(\mathcal{M}')$;
9: **return** $\{\mathcal{M}', C', \boldsymbol{e}(t_c + l_s)\}$;

4.2 OrbitMap-F

We now introduce our streaming algorithm, OrbitMap-F. Briefly, it consists of the following sub-algorithms:

(I) O-Estimator: Estimates the OrbitMap parameter set $\mathcal{M}$, and the model candidate C.
(II) O-Generator: Generates an l_s-steps-ahead future value, i.e., $\boldsymbol{e}(t_c + l_s)$, using $\mathcal{M}$ and C.
(III) O-Feedback: Cleans up useless regimes and transitions stored in $\mathcal{M}$ (if required).

Algorithm 1 provides an overview of OrbitMap-F. At each time point t_c, given a new value $\boldsymbol{x}(t_c)$, it incrementally updates the parameter set $\mathcal{M}$ and the model candidate C using O-Estimator, and then generates an l_s-steps-ahead future value $\boldsymbol{e}(t_c + l_s)$ with O-Generator. It also maintains the OrbitMap parameters by using O-Feedback, which cleans up the useless regimes and transitions stored in $\mathcal{M}$.

Next, we describe our detailed algorithms in steps.

4.2.1 O-Estimator. Our first step is O-Estimator, which incrementally monitors X and identifies the current regimes in a data stream. Given the most recent value $\boldsymbol{x}(t_c)$ at time point t_c, O-Estimator incrementally updates model parameter set $\mathcal{M}$ and the model candidate C. Algorithm 2 is the O-Estimator algorithm in detail. To reduce the computation time in the streaming setting, we adopt an insertion-based approach for updating $\mathcal{M}$. In short, O-Estimator consists of two parts: (I) update the current regime and (II) detect a space transition, and more specifically,

(I) Update the current regime: given the current segment X_c and model candidate C, the algorithm first tries to update the current regime $\boldsymbol{\theta}_c$, i.e., $\{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\} = \arg\min_{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}} f_D(X_c; \boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out})$, so that it minimizes the errors between X_c and $\boldsymbol{\theta}_c$. If the current regime $\boldsymbol{\theta}_c$ does not fit well (i.e., $f_D(X_c) > \rho$), it searches for another (better) regime $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ stored in $\mathcal{M}$. If it is still not good enough (i.e., there is no appropriate regime in $\mathcal{M}$), it creates a new regime for X_c using RegimeCreation, and inserts it into $\mathcal{M}$.

(II) Detect a space transition (if any): after the algorithm has estimated the current parameter set $\{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\}$, it then tries to find a space transition. If it cannot find or create any well-fitting regime (i.e., $f_D(X_c) > \rho$), it terminates the current segment and its regime, X_c, $\boldsymbol{\theta}_c$, and starts a new segment $X_c = \boldsymbol{x}(t_c)$. It also inserts the transition pair $\{\boldsymbol{v}_p^{out}, \boldsymbol{v}_c^{in}\}$ into $\mathcal{V}$. After the insertion, it replaces the previous out-vector $\boldsymbol{v}_p^{out} = \boldsymbol{v}_c^{out}$, and resets the regime $\boldsymbol{\theta}_c = \boldsymbol{v}_c^{in} = \boldsymbol{v}_c^{out} = \emptyset$.

RegimeCreation. When the algorithm finds an unknown pattern in X_c, it needs to create a new regime and estimate its model parameter set $\boldsymbol{\theta}$ using X_c. Here, each regime $\boldsymbol{\theta}$ consists of a large number of parameters (i.e., $\boldsymbol{\theta} = \{\boldsymbol{p}, \mathbf{Q}, \mathcal{A}, \boldsymbol{u}, \mathbf{V}\}$), and it is extremely expensive to optimize them all simultaneously. We thus use an efficient and effective optimization method, namely, RegimeCreation. The idea is that we split a full parameter set $\boldsymbol{\theta}$ into two subsets, i.e., $\boldsymbol{\theta}_L = \{\boldsymbol{p}, \mathbf{Q}, \boldsymbol{u}, \mathbf{V}\}$ and $\boldsymbol{\theta}_N = \{\mathcal{A}\}$, each of which corresponds to a linear/non-linear parameter set, and fit the parameter sets separately. Here, we use the expectation-maximization (EM) algorithm to optimize the linear parameters $\boldsymbol{\theta}_L$. It then estimates non-linear elements in $\boldsymbol{\theta}_N$ and $\boldsymbol{v}^{in}, \boldsymbol{v}^{out}$ to minimize the errors $f_D(X_c; \boldsymbol{\theta}, \boldsymbol{v}^{in}, \boldsymbol{v}^{out})$, using the Levenberg-Marquardt (LM) algorithm and the Runge-Kutta method [17]. We vary k and determine appropriate models so as to minimize the fitting errors.[1]

4.2.2 O-Generator. The next algorithm is O-Generator, which incrementally forecasts an l_s-steps-ahead future value $\boldsymbol{e}(t_c + l_s)$, by using the parameter set $\mathcal{M}$ and the model candidate C, which are estimated by O-Estimator. The idea is quite simple and efficient: given the current regime $\boldsymbol{\theta}_c$ and its estimated in-vector $\boldsymbol{v}_c^{in}$ for the recent segment X_c, it computes an l_s-steps-ahead future value $\boldsymbol{e}(t_c + l_s)$, according to Models 1 and 2. Specifically, O-Generator consists of three steps:

(I) Given the current regime $\boldsymbol{\theta}_c$ and its initial vector $\boldsymbol{v}_c^{in}$, it first generates l_s-steps-ahead future latent variables $S = \{\boldsymbol{s}_c(t_c), \cdots, \boldsymbol{s}_c(t_c + l_s)\}$, where $\boldsymbol{s}_c(t_c) = \boldsymbol{v}_c^{in}$.

[1] For the non-linear tensor $\mathcal{A}$, we only estimate parameters for the diagonal elements $a_{ijk} \in \mathcal{A}$ $(i = j = k)$ to eliminate complexity.

Algorithm 2 O-Estimator $(x(t_c), \mathcal{M}, C)$

Input: (a) New value $x(t_c)$ at time points t_c
(b) Current OrbitMap parameter set $\mathcal{M} = \{\boldsymbol{\Theta}, \mathcal{V}\}$
(c) model candidate $C = \{\boldsymbol{\theta}_c, \boldsymbol{v}_p^{out}, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\}$
Output: (a) Updated OrbitMap parameter set $\mathcal{M}'$
(b) Updated model candidate C'
$X_c = X[t_s : t_c]$; // Update current segment X_c
/* (I) Estimate current regime */
/* Update current regime for X_c using model candidate C */
$\{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\} = \arg\min_{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}} f_D(X_c; \boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out})$;
if $f_D(X_c; \boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}) > \rho$ **then**
  /* If the current regime does not fit well, find a better regime in $\boldsymbol{\Theta}$ */
  $\{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\} = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}, \boldsymbol{v}^{in}, \boldsymbol{v}^{out}} f_D(X_c; \boldsymbol{\theta}, \boldsymbol{v}^{in}, \boldsymbol{v}^{out})$;
  if $f_D(X_c; \boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}) > \rho$ **then**
    /* If it is still not good, create new regime and insert it into $\boldsymbol{\Theta}$ */
    $\{\boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\} = \text{RegimeCreation}(X_c)$; $\boldsymbol{\Theta} = \boldsymbol{\Theta} \cup \boldsymbol{\theta}_c$;
  end if
end if
/* (II) Detect space transition */
if $f_D(X_c; \boldsymbol{\theta}_c, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}) > \rho$ **then**
  /* If it does not fit well, current segment X_c ends now */
  $\mathcal{V} = \mathcal{V} \cup \{\boldsymbol{v}_p^{out}, \boldsymbol{v}_c^{in}\}$; // Insert new transition pair into $\mathcal{V}$
  $\boldsymbol{v}_p^{out} = \boldsymbol{v}_c^{out}$; // Replace previous out-vector
  $X_c = x(t_c)$; $\boldsymbol{\theta}_c = \boldsymbol{v}_c^{in} = \boldsymbol{v}_c^{out} = \emptyset$; // Initialize segment and regime
end if
$\mathcal{M}' = \{\boldsymbol{\Theta}, \mathcal{V}\}$; // Update model parameters
$C' = \{\boldsymbol{\theta}_c, \boldsymbol{v}_p^{out}, \boldsymbol{v}_c^{in}, \boldsymbol{v}_c^{out}\}$; // Update model candidate
return $\{\mathcal{M}', C'\}$;

(II) Next, it tries to find any transition vector pair $\{\boldsymbol{v}_c^{out}, \boldsymbol{v}_f^{in}\} \in \mathcal{V}$ that is close to the trajectory of S. If the trajectory is close enough to the vector $\boldsymbol{v}_c^{out}$ (i.e., $||\boldsymbol{s}_c(t_s) - \boldsymbol{v}_c^{out}|| \leq \rho$, $t_c \leq t_s \leq t_c + l_s$), it moves to another regime $\boldsymbol{\theta}_f$ at time point t_s. Consequently, it updates latent variables $S = \{\boldsymbol{s}_c(t_c), \cdots, \boldsymbol{s}_c(t_s - 1), \boldsymbol{v}_f^{in}, \boldsymbol{s}_f(t_s + 1), \cdots, \boldsymbol{s}_f(t_c + l_s))\}$.

(III) Given the latent variables S, generated by (I) and (II), it computes estimated variables E using $\boldsymbol{\theta}_c$ (and $\boldsymbol{\theta}_f$, if any transition), according to Model 1. It then returns an l_s-steps-ahead future value $\boldsymbol{e}(t_c + l_s)$.

Most importantly, if the same regimes and their transitions appear more than once in the data stream X, OrbitMap-F can memorize/store them in the transition network in $\mathcal{M}$, and reuse them efficiently to predict future dynamics.

4.2.3 O-Feedback. We still have two additional issues to deal with: that is, (a) how to determine the optimal transition strength ρ, and (b) how to eliminate noise or meaningless regimes stored in the model parameter set $\mathcal{M}$. Here, with respect to the first issue (a), the regime creation and segmentation rely on transition strength ρ, which corresponds to the prediction accuracy between the original data stream and our estimation. The transition strength also affects the network complexity of $\mathcal{M}$, where the lower variable ρ creates a more complex network (i.e., more regimes and transitions). So, what is the best way to determine the optimal solution? We want to update ρ and create the optimal transition network $\mathcal{M}$ that can best describe current activities in the data stream, so that it can forecast l_s-steps-ahead future values. We thus determine the optimal ρ for minimizing the l_s-steps-ahead forecasting error between

the original value and our estimation, i.e., $\arg\min_\rho \sum_{t=1}^{t_c} ||\boldsymbol{x}(t + l_s) - \boldsymbol{e}(t + l_s)||$.

The second issue (b) is cleaning up the model $\mathcal{M}$. In practice, data streams might contain various types of noise or meaningless patterns that should be ignored. Also, if transition strength ρ is tuned to be a larger variable, we need to shrink the transition network of $\mathcal{M}$ by grouping similar regimes/nodes together. We thus introduce a self-cleansing function, which incrementally maintains and updates regimes and their transitions in $\mathcal{M}$. It consists of two sub-functions, that is, (I) regime elimination: it removes useless regimes from $\mathcal{M}$, and more specifically, it deletes regimes that have never been used by O-Estimator; (II) transition network shrinkage: it merges all the closest regime pairs $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$, whose estimation error between each other is less than ρ.

Theoretical analysis. Let r be the number of regimes in $\mathcal{M}$.

Lemma 1. *The computation time of OrbitMap-F is at least $O(1)$ time per time point, and at most $O(r)$ time per time point.*

Proof. Please see Appendix A. □

5 EXPERIMENTS

We now demonstrate the effectiveness of OrbitMap with real datasets. The experiments were designed to answer the following questions:

(Q1) *Effectiveness*: How successful is our method in modeling and forecasting long-term dynamics in given input streams?
(Q2) *Accuracy*: How well does our method forecast future values?
(Q3) *Scalability*: How does our method scale in terms of computational time?

Our experiments were conducted on an Intel Core i7-3770K 3.50GHz with 32GB of memory, running Linux.

Q1: Effectiveness. We demonstrate the modeling power of OrbitMap in terms of capturing important patterns in data streams and forecasting future values. We performed experiments on eight data streams in multiple domains, e.g., environmental, machine, human sensors and online social user activities.[2] To ensure the repeatability of our results, we used several publicly available datasets. We also performed our experiments on real IoT data streams in smart factories belonging to several companies. Due to space limitations, here we only describe our results for the *(#1) Factory-worker* dataset. This dataset consists of $d = 4$ acceleration sensors (100 Hz), attached to the right/left legs and arms of a worker in a factory. The OrbitMap output has already been presented in Figure 1 of section 1. As already seen, our method automatically and incrementally captures (a) typical regimes (e.g., walking and lifting) in a given stream, and (b) their transition points (see vertical axis in figure (b)), as well as (c) dynamic transition network (e.g., lifting → wiping). Most importantly, the transition network grows over time, as a new regime or transition appears in the stream. Since OrbitMap can continuously capture dynamic regimes and their transitions, it also enables us to perform real-time future value forecasting. Figure 1 (d) shows our real-time forecasting results, and specifically, OrbitMap forecasts an $l_s = 200$-steps-ahead future value, for every time point. Our outputs with the other datasets are shown later in section 6 and Appendix B.

Q2. Accuracy. Next, we discuss the quality of OrbitMap in terms of l_s-steps-ahead forecasting accuracy. We compared our method with the following methods: (a) RegimeCast [22], which is a real-time forecasting algorithm for data streams, (b) SARIMA, and (c) TBATS [20], which are linear forecasting methods. We also compared our method with (d) LSTM and (e) GRU [4], which are recurrent neural network (RNN) models with long short-term memory (LSTM) cells and gated recurrent units (GRUs), respectively. For all methods, we trained the parameters using half of the sequence and then started the future value prediction. Figure 4 shows the forecasting error of OrbitMap for *(#1) Factory-worker*. Specifically, Figure 4 shows the root mean square error (RMSE) between the original value $\boldsymbol{x}(t_c + l_s)$ and the l_s-steps-ahead estimated variables $\boldsymbol{e}(t_c + l_s)$ at each time interval. A lower value indicates a better forecasting accuracy. Similarly, Figure 5 shows the average forecasting error of OrbitMap and its competitors for eight datasets (*#1-#8*). Our method achieves a high forecasting accuracy for every dataset, while other methods cannot forecast future evolutions very well, because they cannot capture multiple distinct regimes and their space transitions. Please also see Appendix B for more details.

Q3. Scalability. We also evaluate the efficiency of our algorithm. Figure 6 compares OrbitMap with its competitors in terms of computation time at each time point t_c. Note that the figures are shown on a linear-log scale. As we discussed in Lemma 1, our algorithm can monitor a data stream for at least $O(1)$ time and at most $O(r)$ time per time point. In fact, OrbitMap is significantly (i.e., up to five orders of magnitude) faster than its competitors for large streams. Figure 7 shows the average computation time at each time point for eight datasets *(#1)-(#8)*.

Consequently, thanks to our careful design, OrbitMap-F provides a quick response in streams, and our method achieves a large reduction in both computation time and forecasting error.

6 ORBITMAP AT WORK

Here, we demonstrate one of our promising applications, namely, real-time mining in smart factories.

Operation monitoring. Figure 8 (a) shows a four-dimensional IoT sensor stream in a semiconductor fabrication plant,[3] and specifically collected using the chemical mechanical polishing (CMP) process, each dimension consists of pad temperature, airbag pressure, atomizer flow and table motor current. Since wafer processing is very sensitive, and the company needs to produce high quality wafers, it is critical to keep the machines in good condition by carefully monitoring the current status in the sensor streams. The use of our modeling approach makes it possible to monitor the sensor streams in the factory, and to provide a visualization of the regular CMP processes. Figure 8 (b) and (c) show our regime identification and transition network at three different time points ($t = 1000, 1400, 1600$), where the blue/red nodes and rectangles are the two most recent regimes. Our approach successfully captures the sensitive and precise procedures of the CMP machine, such as the up/down patterns of the airbag pressure and motor current, the cooling down process and atomizer flows. Also, as shown by the transition network in Figure 8 (c), the graph structure is unique: it

[2] *(#1) Factory-worker, (#2) Semicon, (#3) Engine, (#4) G-outdoor, (#5) G-sports, (#6) Exercise, (#7) Cleaning, (#8) Wandering.*

[3] Provided by Sony Semiconductor Manufacturing Corporation.

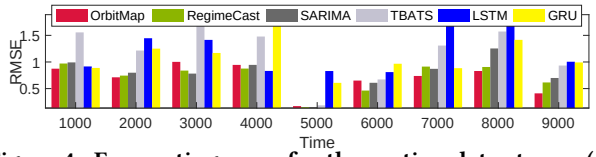


Figure 4: Forecasting error for the motion data stream (*(#1) Factory-worker*). RMSE between original and forecast values of OrbitMap and competitors at each time interval (lower is better).

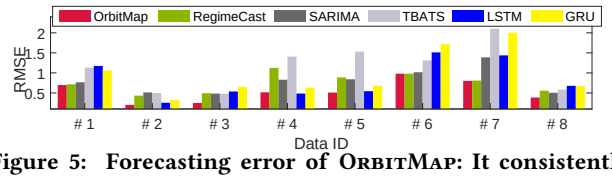


Figure 5: Forecasting error of OrbitMap: It consistently wins. Average RMSE for each dataset (lower is better).

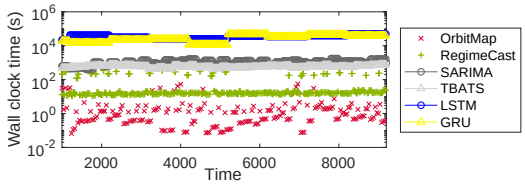


Figure 6: Wall clock time vs. data stream length t_c for *(#1) Factory-worker*. It is up to 580,000x faster than its competitors (shown in linear-log scale).

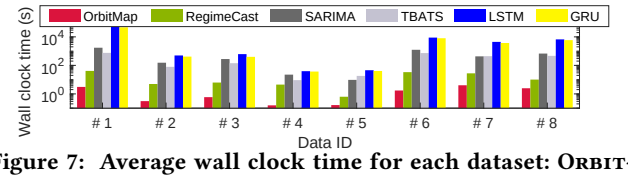
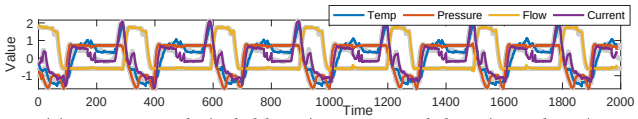


Figure 7: Average wall clock time for each dataset: OrbitMap consistently wins. The results are shown in log scale to accommodate slow competitors.

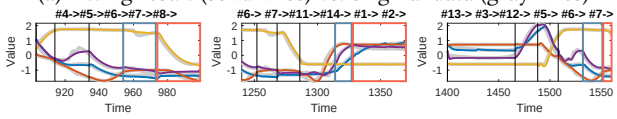
has a long chain consisting of multiple distinct nodes, which indicates that the data stream consists mainly of regular, repeating patterns, with some extra activities. Consequently, using our OrbitMap modeling, a factory manager can remotely and automatically monitor and analyze sensitive machine performance, and can plan a regular maintenance schedule, predict potential accidents and more. In addition, our scalable algorithm allows the real-time forecasting of future values. Figure 8 (d) shows 10-steps-ahead future values at three different time points. The red vertical axes (i.e., $t_c, t_c + l_s$) show the current and l_s-steps-ahead time points, and our method successfully captures future dynamic regime transitions, and forecasts long-term multi-step future patterns.

Machine downtime reduction. Here we introduce another case of IoT data monitoring, namely, machine downtime prediction. Manufacturing problems, such as faulty goods and equipment failure, are caused by various factors. OrbitMap can automatically detect the causes of problems in real time, and this information could be used to improve productivity and quality, and optimize the manufacturing process in smart factories. Here, we used the IoT data stream collected by a computerized numerical control (CNC) machine in an internal-combustion engine plant.[4] Figure 9 (a) shows

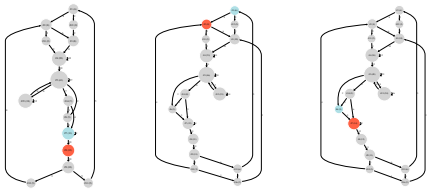
[4] Provided by Mitsubishi Heavy Industries Engine & Turbocharger, Ltd.



(a) Fitting result (solid lines) vs. original data (gray lines)

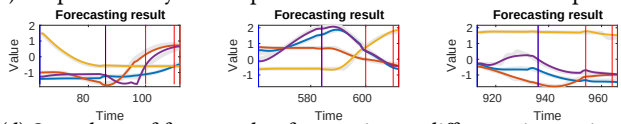


(b) Snapshots of real-time regime identification and segmentation



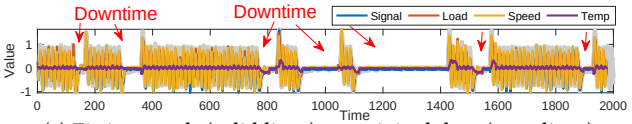
$t_c = 1000$ $t_c = 1400$ $t_c = 1600$

(c) Snapshots of dynamic space transitions at different time points

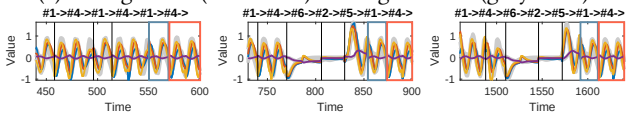


(d) Snapshots of future value forecasting at different time points

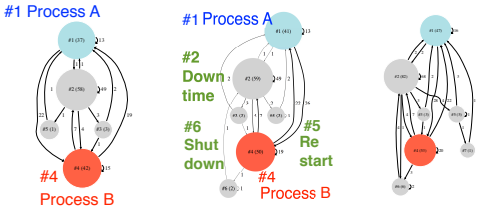
Figure 8: OrbitMap for *(#2) Semicon*.



(a) Fitting result (solid lines) vs. original data (gray lines)

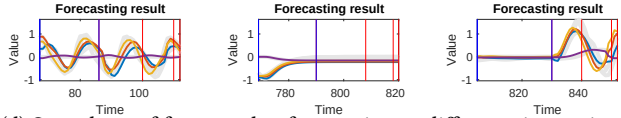


(b) Snapshots of real-time regime identification and segmentation



$t_c = 600$ $t_c = 900$ $t_c = 1700$

(c) Snapshots of dynamic space transitions at different time points



(d) Snapshots of future value forecasting at different time points

Figure 9: OrbitMap modeling for *(#3) Engine*.

the original data stream and our estimation, where we have four sensors, i.e., operation signal, spindle load, spindle speed and spindle temperature. As shown, the data stream contains several regularly repeated activities (e.g., up/down patterns of spindle temperature and speed), and it also has some downtime (please see the red arrows). Figure 9 (b) and (c) show our OrbitMap output at three different time points. The left figure shows the normal operation activities at time point $t = 600$, which consists of two

types of regimes (#1 → #4 → #1 → #4 → ⋯), while the middle and right figures show snapshots at $t = 900, 1700$, which contain downtime patterns. Most importantly, we can observe the same transition for each downtime activity, i.e., #6 → #2 → #5 → #1 → ⋯, where, each regime corresponds to #6 "shutdown" (slowing down the spindle speed, and cooling down the machine), #2 "downtime" (stopping the spindle with a stable temperature), and #5 "restart" (starting the spindle, and warming up), and then #1 "process-A" (normal operation). Consequently, using our regime identification and dynamic transition network, the manager can incrementally check the machine condition and avoid machine downtime as well as reduce extra cost or prevent accidents in advance, simply by monitoring indicators/signs of downtime or trouble. Figure 8 (d) shows 10-steps-ahead future values at three different time points. Our method can capture future unknown patterns, and forecast future values, incrementally and adaptively.

7 CONCLUSIONS

In this paper, we focused on the problem of modeling and forecasting time-evolving data streams, and presented OrbitMap, which exhibits all the desirable properties that we listed in the introduction:

(1) It is **Effective**: it captures important time-evolving patterns (i.e., regimes and their dynamic space transitions) in data streams and provides long-range forecasting at any time;
(2) It is **General**: it matches diverse real data;
(3) It is **Scalable**: we proposed an efficient algorithm that is constant in terms of input data size.

We also performed our analytics on real industrial IoT data streams, and demonstrated the practicality and effectiveness of our dynamic modeling and forecasting approach.

Acknowledgment. We sincerely thank the anonymous reviewers, for their time and effort during the review process. We thank all collaborating companies for providing research funding, as well as valuable datasets, and for their support and feedback regarding this work. This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research Number JP17H04681, JP16K12430, JP18H03245, PRESTO JST, the MIC/SCOPE, #162110003 and Health Labour Sciences Research Grant.

REFERENCES

[1] *OrbitMap.* https://www.dm.sanken.osaka-u.ac.jp/~yasuko/software.html.
[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, 2006.
[3] D. Chakrabarti and C. Faloutsos. F4: Large-scale automated forecasting using fractals. *CIKM*, 2002.
[4] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8(1):6085, 2018.
[5] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
[6] J. J. Dabrowski, A. Rahman, A. George, S. Arnold, and J. McCulloch. State space models for forecasting water quality variables: An application in aquaculture prawn farming. In *KDD*, pages 177–185. ACM, 2018.
[7] I. N. Davidson, S. Gilpin, O. T. Carmichael, and P. B. Walker. Network discovery via constrained tensor analysis of fmri data. In *KDD*, pages 194–202, 2013.
[8] G. De Francisci Morales, A. Bifet, L. Khan, J. Gama, and W. Fan. Iot big data stream mining. In *KDD, Tutorial*, pages 2119–2120, 2016.
[9] I. Fox, L. Ang, M. Jaiswal, R. Pop-Busui, and J. Wiens. Deep multi-output forecasting: Learning to accurately predict blood glucose trajectories. In *KDD*, pages 1387–1395, 2018.
[10] J. Ginsberg, M. Mohebbi, R. Patel, L. Brammer, M. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457:1012–1014, 2009.
[11] K. Greenewald, S. Park, S. Zhou, and A. Giessing. Time-dependent spatially varying graphical models, with application to brain fmri data analysis. In *Advances in Neural Information Processing Systems 30*, pages 5832–5840. 2017.
[12] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, 2013.
[13] D. Hallac, S. Vare, S. P. Boyd, and J. Leskovec. Toeplitz inverse covariance-based clustering of multivariate time series data. In *KDD*, pages 215–223, 2017.
[14] Z. He, S. Gao, L. Xiao, D. Liu, H. He, and D. Barber. Wider and deeper, cheaper and faster: Tensorized lstms for sequence learning. In *Advances in Neural Information Processing Systems 30*, pages 1–11. 2017.
[15] M. D. Hoffman, D. M. Blei, and F. R. Bach. Online learning for latent dirichlet allocation. In *NIPS*, pages 856–864, 2010.
[16] T. Iwata, T. Yamada, Y. Sakurai, and N. Ueda. Online multiscale dynamic topic models. In *KDD*, pages 663–672, 2010.
[17] E. Jackson. *Perspectives of Nonlinear Dynamics.* Cambridge University Press, 1992.
[18] L. Li, B. A. Prakash, and C. Faloutsos. Parsimonious linear fingerprinting for time series. *PVLDB*, 3(1):385–396, 2010.
[19] S. C.-X. Li and B. M. Marlin. A scalable end-to-end gaussian process adapter for irregularly sampled time series classification. In *Advances in Neural Information Processing Systems 29*, pages 1804–1812. 2016.
[20] A. M. D. Livera, R. J. Hyndman, and R. D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.
[21] M. Mathioudakis, N. Koudas, and P. Marbach. Early online identification of attention gathering items in social media. In *WSDM*, pages 301–310, 2010.
[22] Y. Matsubara and Y. Sakurai. Regime shifts in streams: Real-time forecasting of co-evolving time sequences. In *KDD*, pages 1045–1054, 2016.
[23] Y. Matsubara, Y. Sakurai, and C. Faloutsos. Autoplait: Automatic mining of co-evolving time sequences. In *SIGMOD*, 2014.
[24] Y. Matsubara, Y. Sakurai, and C. Faloutsos. The web as a jungle: Non-linear dynamical systems for co-evolving online activities. In *WWW*, 2015.
[25] Y. Matsubara, Y. Sakurai, and C. Faloutsos. Non-linear mining of competing local activities. In *WWW*, pages 737–747, 2016.
[26] Y. Matsubara, Y. Sakurai, C. Faloutsos, T. Iwata, and M. Yoshikawa. Fast mining and forecasting of complex time-stamped events. In *KDD*, pages 271–279, 2012.
[27] Y. Matsubara, Y. Sakurai, B. A. Prakash, L. Li, and C. Faloutsos. Rise and fall patterns of information diffusion: model and implications. In *KDD*, pages 6–14, 2012.
[28] Y. Matsubara, Y. Sakurai, W. G. van Panhuis, and C. Faloutsos. FUNNEL: automatic mining of spatially coevolving epidemics. In *KDD*, pages 105–114, 2014.
[29] K. P. Murphy. Switching kalman filters. Technical report, 1998.
[30] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, hands-off stream mining. In *VLDB*, pages 560–571, 2003.
[31] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.
[32] S. Papadimitriou and P. S. Yu. Optimal multi-scale patterns in time series streams. In *SIGMOD*, pages 647–658, 2006.
[33] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
[34] J. Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge University Press, New York, NY, USA, 2000.
[35] Y. Sakurai, Y. Matsubara, and C. Faloutsos. Mining big time-series data on the web. In *WWW, Tutorial*, pages 1029–1032, 2016.
[36] Y. Sakurai, Y. Matsubara, and C. Faloutsos. Smart analytics for big time-series data. In *KDD, Tutorial*, 2017.
[37] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD*, pages 599–610, 2005.
[38] P. Wang, H. Wang, and W. Wang. Finding semantics in time series. In *SIGMOD Conference*, pages 385–396, 2011.
[39] Y. Wang, D. J. Miller, K. Poskanzer, Y. Wang, L. Tian, and G. Yu. Graphical time warping for joint alignment of multiple curves. In *Advances in Neural Information Processing Systems 29*, pages 3648–3656. 2016.
[40] H.-F. Yu, N. Rao, and I. S. Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In *Advances in Neural Information Processing Systems 29*, pages 847–855. 2016.
[41] J. Zhou and A. K. H. Tung. Smiler: A semi-lazy time series prediction system for sensors. In *SIGMOD*, pages 1871–1886, 2015.
[42] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai. What to do next: Modeling user behaviors by time-lstm. In *IJCAI*, pages 3602–3608, 2017.

APPENDIX

A STREAMING ALGORITHM

Proof of Lemma 1.

Proof. For each time point t_c, O-Estimator updates $\mathcal{M}, C$. If it has an appropriate model candidate C, it can quickly update the parameters, which requires $O(1)$ time. If it cannot find a well-fitting regime, it then tries to find the optimal regime in $\mathcal{M}$, which requires $O(r)$ time. In addition, RegimeCreation requires $O(l_c)$ time to create new regime parameters for the current segment X_c, where l_c is the length of X_c. Similarly, O-Generator requires $O(l_s)$ time to generate l_s-steps-ahead estimated variables, and O-Feedback needs $O(r)$ time to update regimes in $\mathcal{M}$. Note that l_c and l_s are small constant values that are negligible. Thus, the complexity is at least $O(1)$ and at most $O(r)$ time per time point. □

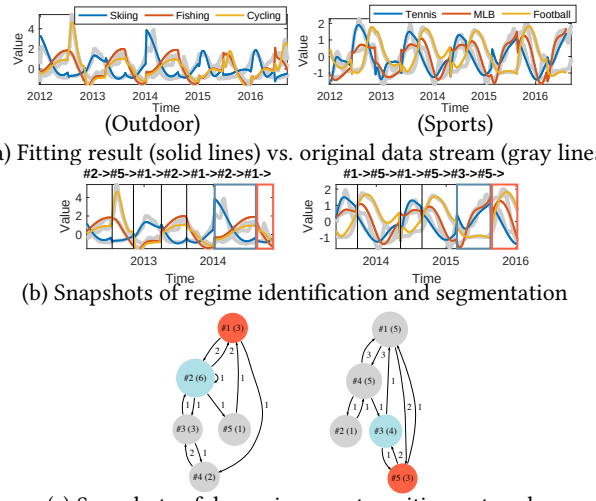
B EXPERIMENTS

Dataset description, settings and additional discoveries. Here, we report the detailed experimental results that we described in Figure 5 and Figure 7. We performed experiments on eight data streams, namely, *(#1) Factory-worker*, *(#2) Semicon*, *(#3) Engine*, *(#4) G-outdoor*, *(#5) G-sports*, *(#6) Exercise*, *(#7) Cleaning*, *(#8) Wandering*. Here, we describe our experimental results for *(#4-#8)*.

(#4) G-outdoor, (#5) G-sports. Figure 10 shows our output for *GoogleTrends*, which consists of the search volumes for various queries (i.e., keywords) on Google.[5] Each query represents search volumes related to keywords obtained on a weekly basis from 2007 to 2017. Since OrbitMap has the ability to detect unknown patterns and their transition without the user's prior knowledge, we can apply it to automatic social activity analysis. Here, we introduce some of our discoveries as regards *GoogleTrends* data streams. The left column in Figure 10 shows our output for three major outdoors-related keywords (namely, skiing, fishing and cycling), while, the right column shows sports-related keywords (i.e., tennis, Major League Baseball (MLB) and American football). We trained our model using half of the data streams (from 2007 to 2011), and undertook dynamic data monitoring from 2012 to 2017. The top, middle and bottom rows show our overall fitting, regime identification and their transition network, respectively. OrbitMap discovers that these keywords have an annual cyclic pattern (e.g., #1 → #2 → #1 → ⋯), which indicates that online users modulate their activity over time. Each keyword always has a certain volume of popularity, however, user behavior changes dynamically according to the season and various annual events (e.g., summer/winter vacations, MLB regular season). OrbitMap also finds some outliers (e.g., a sudden spike in 2013 for cycling). Our streaming algorithm also predicts future user activities. Here, it forecasts the three-month-ahead future value (i.e., $l_s = 13$ weeks), at every time point. The figure shows snapshots of OrbitMap-F for each dataset. Clearly, our algorithm successfully captures multiple regimes and their transitions, and forecasts multi-steps ahead future activities.

(#6) Exercise, (#7) Cleaning, (#8) Wandering. Figure 11, Figure 12 and Figure 13, show our discoveries for motion capture data streams,[6] which consist of four sensors (left/right legs and arms). The figure

[5] http://www.google.com/trends/
[6] http://mocap.cs.cmu.edu/



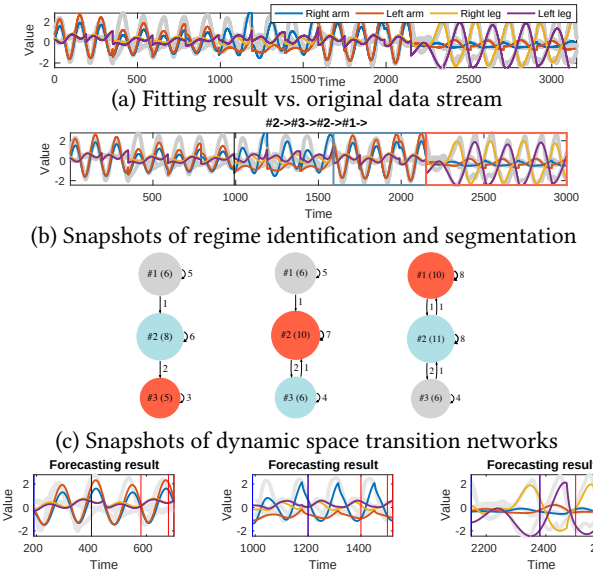
(a) Fitting result (solid lines) vs. original data stream (gray lines)

(b) Snapshots of regime identification and segmentation

(c) Snapshots of dynamic space transition networks

(d) Snapshots of three-month-ahead future value forecasting

Figure 10: OrbitMap modeling for *(#4,#5) GoogleTrends*.



(a) Fitting result vs. original data stream

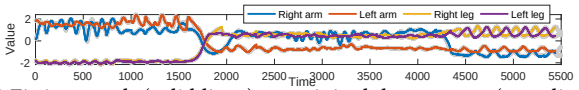
(b) Snapshots of regime identification and segmentation

(c) Snapshots of dynamic space transition networks

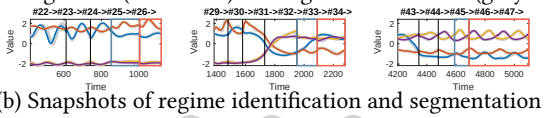
(d) Snapshots of l_s = 100-steps-ahead future value forecasting

Figure 11: OrbitMap modeling for *(#6) Exercise*.

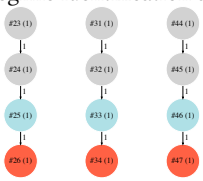
shows our modeling result for "exercise" motion (e.g., stretching, walking), "house-cleaning" motion (e.g., dragging a mop, wiping a window) and "wandering" motion (e.g., walking, turning). As can be seen, our model successfully captures distinct regimes as well as their dynamic space transitions. Also note that these three data streams have completely different transition network structures. The figures also show our real-time forecasting for the *Mocap* stream. For each dataset, we set $l_s = 100$. Similar to other datasets, OrbitMap captures original time-evolving patterns very well, and successfully captures upcoming future values, effectively and adaptively.



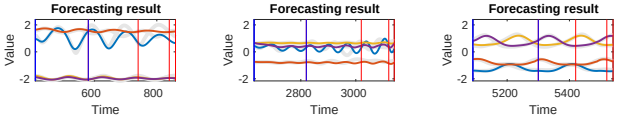
(a) Fitting result (solid lines) vs. original data stream (gray lines)



(b) Snapshots of regime identification and segmentation

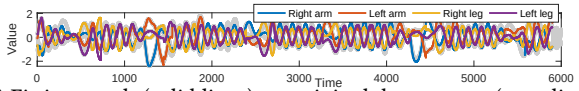


(c) Snapshots of dynamic space transition networks

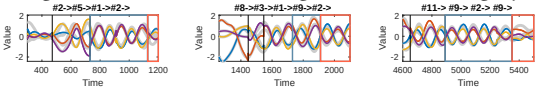


(d) Snapshots of l_s = 100-steps-ahead future value forecasting

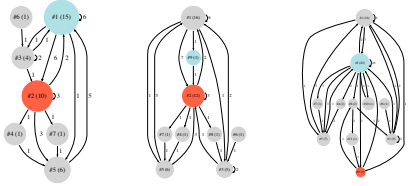
Figure 12: OrbitMap modeling for *(#7) Cleaning*.



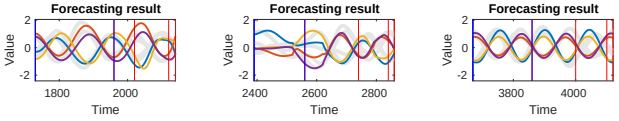
(a) Fitting result (solid lines) vs. original data stream (gray lines)



(b) Snapshots of regime identification and segmentation



(c) Snapshots of dynamic space transition networks



(d) Snapshots of l_s = 100-steps-ahead future value forecasting

Figure 13: OrbitMap modeling for *(#8) Wandering*.

Accuracy and scalability. Figure 14 shows the forecasting error (RMSE) of OrbitMap and its competitors for each dataset. For all methods, we trained the parameters using half of the original data streams, and then started the future forecasting. Also, for SARIMA and TBATS, we set $frequency = l_s$. We used AIC to determine the optimal number of parameters for SARIMA. For LSTM and GRU, we trained the models with Adam optimization and early stopping, and set a dropout of rate 0.5 for the top regressor layer, according to [4]. We used 2-layer and 100 hidden units in each layer. For a fair comparison, we used a single CPU core and no GPUs. Our method achieved a high forecasting accuracy for every dataset, whereas other methods cannot forecast future evolutions very well, because they cannot capture the dynamic space

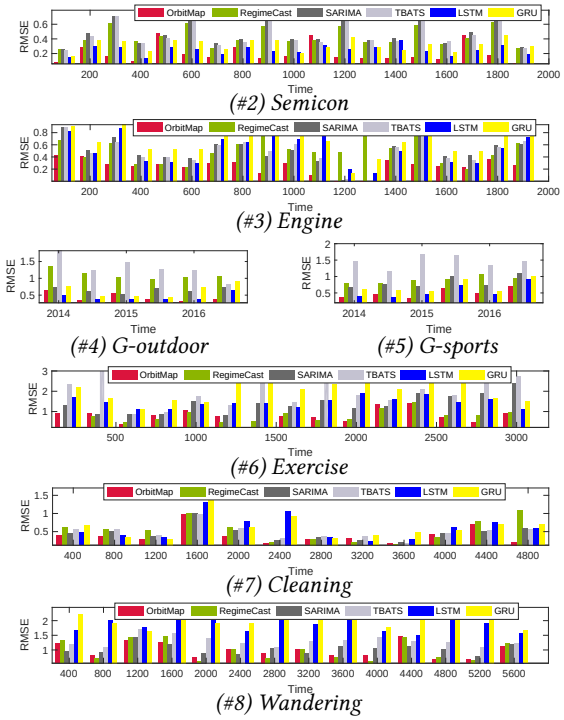


Figure 14: Forecasting error (RMSE) of OrbitMap for other datasets (*#2-#8*), at each time interval (lower is better).

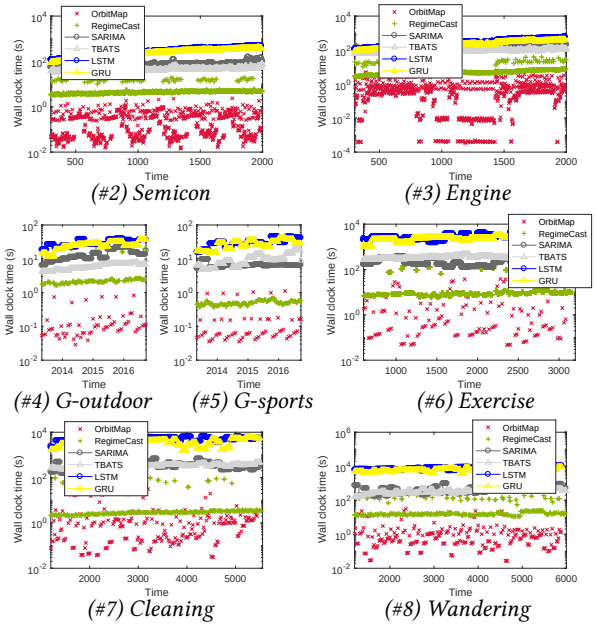


Figure 15: Wall clock time vs. data stream length t_c for other datasets (*#2-#8*), shown in linear-log scale.

transitions between multiple distinct regimes in streams. Similarly, Figure 15 compares OrbitMap with its competitors in terms of computation time for various sequence lengths t_c. Note that the figures are shown on linear-log scales. Clearly, our method achieves a large reduction in both computation time and forecasting error for various types of data streams.