A APPENDIX

A.1 Algorithm Details

In order to outline our proposed model in detail, we present the complete pseudo code of EvoNet to illustrate the learning procedure. Given the observations $\langle \mathbf{X}_{1:T}, \mathbf{Y}_{1:T} \rangle$ and parameters $(|\mathcal{V}|, \tau, \mathcal{F}_{\text{state}}, \mathcal{F}_{\text{MP}})$, EvoNet first captures different states by means of the recognition function $\mathcal{F}_{\text{state}}$. It then constructs the evolutionary state graph $\langle \mathbf{G}^{(1:T)} \rangle$ and conducts graph propagation by means of the message function $\mathcal{F}_{\text{MP}}$ and EvoBlock. Finally, the learned representations are fed into an output model for prediction tasks; we use a back-propagation learning algorithm with cross-entropy loss to train the entire networks. More details can be found in Algorithm 1.

Algorithm 1 The learning procedure of EvoNet

```
Input: Observations $\langle X_{1:T}, Y_{1:T} \rangle$, parameters $(|\mathcal{V}|, \tau, \mathcal{F}_{state}, \mathcal{F}_{MP})$
Output: Model parameters $\theta$, event prediction $Y'$
 1: $\mathcal{F}_{state} \leftarrow (X_{1:T}, |\mathcal{V}|)$, train state recognition model
 2: for each segment $X_t \in X_{1:T}$ do
 3:     $\{P(\Theta_v|X_t) \leftarrow \mathcal{F}_{state}(X_t, \Theta_v)\}_{v \in \mathcal{V}}$, get the recognized weights of states as Eq 1
 4:     $G^{(t)} \leftarrow$ construct the evolutionary state graph as Eq 2
 5: end for
 6:
 7: $U^{(0)} \leftarrow \mathbf{0}$ initialize the graph-level representation
 8: for $v \in \mathcal{V}$ do
 9:     $h_v^{(0)} \leftarrow \Theta_v$, initialize the node-level representation of state $v$
10: end for
11: while the parameters of EvoNet have not converged do
12:     take $N$ samples of $\left\{\langle X_{1:T}, Y_{1:T} \rangle, \langle G^{(1:T)} \rangle\right\}$ as a batch
13:     for each $X_t \in X_{1:T}$, $Y_t \in Y_{1:T}$, $G^{(t)} \in G^{(1:T)}$ do
14:         $\left\{H_v^{(t)}\right\}_{v \in \mathcal{V}} \leftarrow \mathcal{F}_{MP}\left(G^{(t)}, \left\{h_v^{(t-1)}\right\}_{v \in \mathcal{V}}\right)$, conduct message passing as Eq 4
15:         $\alpha_t \leftarrow$ compute attention score as Eq 6.3
16:         $\left\{h_v^{(t)}\right\}_{v \in \mathcal{V}} \leftarrow$ node-level propagation as Eq 6.1
17:         $U^{(t)} \leftarrow$ graph-level propagation as Eq 6.2
18:         $h_G^t \leftarrow$ compute current feature embedding as Eq 8
19:         $P\left(Y'_{t+1}|h_G^t\right) \leftarrow$ estimate the probabilities of the next event
20:     end for
21:     $\theta \leftarrow \nabla_\theta \left[\frac{1}{N}\sum_{n=1}^{N}(\mathcal{L})\right]$, back-propagate the loss and train the whole EvoNet as Eq 9
22: end while
```

A.2 Implementation of State Recognition

In this section, we present several implementations for state recognition, including sequence clustering [17], SAX words [24] and Shapelets [26], which have been proven to be competitive for capturing the representative patterns (or *states*), in previous works.

Sequence Clustering. Cluster methods allow us to find the repeated patterns in time-series segments, which can reduce the dimension and allow us to derive insights capable of explaining time-series evolution [1, 17, 20]. Herein, we take Kmeans[21] as example; the aim here is to partition the n segments $\mathbf{X}_t$ into $|\mathcal{V}|$ sets $\Omega = \{\Omega_1, ..., \Omega_{|\mathcal{V}|}\}$, so as to minimize the within-cluster sum of squares, i.e., variance. Formally, the objective is to find:

$$\underset{\Omega}{\arg\min} \sum_{v \in \mathcal{V}} \sum_{\mathbf{X}_t \in \Omega_v} ||\mathbf{X}_t - \Theta_v||^2 = \underset{\Omega}{\arg\min} \sum_{v \in \mathcal{V}} |\Omega_v| \mathrm{Var}\, \Omega_v \tag{10}$$

where Θ_v is the mean of all segments in Ω_v. We then normalize the distance $\mathcal{D}(\mathbf{X}_t, \Theta_v)$ between a segment $\mathbf{X}_t$ and patterns Θ_v as the recognition weight, which can be formulated as follows:

$$\begin{aligned} &\mathcal{D}(\mathbf{X}_t, \Theta_v) = ||\mathbf{X}_t - \Theta_v||^2 \\ &\mathbf{P}(\Theta_v|\mathbf{X}_t) = \frac{\max([\mathcal{D}(\mathbf{X}_t, \Theta_v)]_{v \in \mathcal{V}}) - \mathcal{D}(\mathbf{X}_t, \Theta_v)}{\max([\mathcal{D}(\mathbf{X}_t, \Theta_v)]_{v \in \mathcal{V}}) - \min([\mathcal{D}(\mathbf{X}_t, \Theta_v)]_{v \in \mathcal{V}})} \end{aligned} \tag{11}$$

where we adopt Euclidean distance to measure the similarity between segment $\mathbf{X}_t$ and state patterns Θ_v; the smaller this distance, the more similar they are. We can then construct the evolutionary state graph to represent the relations among different clusters.

SAX word. Symbolic aggregate approximation (SAX) is the first symbolic representation for time series that allows for dimensional reduction and indexing with a lower-bounding distance measure. It transforms the original time-series segments into several average values (PAA representation[6]) and converts them into a string.

Herein, we can consider each SAX word as a state v and extend the corresponding average value a as representative patterns of the time series segments, i.e., $\Theta_v = [a, ..., a]$. Based on this, we can normalize the distance $\mathcal{D}(\mathbf{X}_t, \Theta_v)$ as the recognition weight, following the approach outlined in Eq 1. Subsequently, we can construct the evolutionary state graph to represent the relations among SAX representations.

Shapelet. A shapelet Θ_v is a segment that is representative of a certain class. More precisely, it can separate segments into two smaller sets, one that is close to Θ_v and another that is far from Θ_v according to some specific criteria, such that for a given time series classification task, positive and negative samples can be put into different groups. The criteria for these can be formalized as

$$\mathcal{L}_{\text{shapelet}} = -g\left(\mathcal{D}_{\text{pos}}(\mathbf{X}_t, \Theta_v), \mathcal{D}_{\text{neg}}(\mathbf{X}_t, \Theta_v)\right) \tag{12}$$

where $\mathcal{L}_{\text{shapelet}}$ measures the dissimilarity between positive and negative samples towards the shapelet Θ_v. $\mathcal{D}_*(\mathbf{X}_t, \Theta_v)$ denotes the set of distances with respect to a specific group, i.e., positive or negative class; the function g takes two finite sets as input and returns a scalar value to indicate how far apart these two sets are. This could be information gain or some dissimilarity measurements on sets (i.e., KL divergence). We can then adopt the same approaches as in the above definitions to recognize states' weights and construct the evolutionary state graph to represent the relations among shapelets.

A.3 Implementation of Message Passing

As for the implementations of message passing in local information aggregation, there are many existing works addressing this issue, such as pooling, GGNN [23], GCN [14], GraphSAGE [18], GAT [38], etc.. Herein, we present their implementation details. Broadly speaking, the aim of message passing is to aggregate the *messages*

[6]https://jmotif.github.io/sax-vsm_site/morea/algorithm/PAA.html

of node v's neighbors, and thus to compute its new representation vector, the scheme of which is

$$\mathbf{H}_v^{(t)} = \sum_{v' \in N(v)} \mathcal{F}_{\text{MP}} \left(\mathbf{h}_{v'}^{(t-1)}, e_{(v,v')}^{(t)} \right) \tag{13}$$

where $\mathbf{H}_v^{(t)}$ is the intermediate representation of node v after aggregation; moreover, $\mathcal{F}_{\text{MP}}(\cdot,\cdot)$ is the specific message function, which combines the messages from all v's neighbors $N(v)$ in graph $\mathbf{G}^{(t)}$.

Pooling. Pooling is a simple implementation, which receives the neighbors' messages by computing the production of these neighbors' representation and current transition weight. This approach can be formulated as

$$\mathcal{F}_{\text{MP}} \left(\mathbf{h}_{v'}^{(t-1)}, e_{(v,v')}^{(t)} \right) = m_{(v,v')}^{(t)} \times \mathbf{h}_{v'}^{(t-1)} \tag{14}$$

where $v' \in N(v)$ is a neighbor of node v and $\mathbf{h}_{v'}^{(t-1)}$ is its representation of the last temporal point. $m_{(v,v')}^{(t)}$ is the current relation weight, which is computed by Eq 2 (see details in Section 3.1).

GGNN. Gated Graph Neural Networks [23] implement a message-feedback mechanism: in short, when node v' passes a message to node v via edge $(v' \longrightarrow v)$, v will send a feedback message to v'. This approach aggregates the in-degree and out-degree messages from its neighbors, which is formulated as

$$\begin{aligned}\mathcal{F}_{\text{MP}} \left(\mathbf{h}_{v'}^{(t-1)}, e_{(v,v')}^{(t)} \right) = W_{\text{in}} \cdot \left[m_{(v',v)}^{(t)} \times \mathbf{h}_{v'}^{(t-1)} \right] + \\ W_{\text{out}} \cdot \left[m_{(v,v')}^{(t)} \times \mathbf{h}_{v}^{(t-1)} \right] + b\end{aligned} \tag{15}$$

where W, b is the learnable weight and bias, which is related to the downstream task. From the perspective of the whole graph (adjacency matrix), we in fact build a new graph with the opposite directed edges. Hence, the above scheme can be reformulated as

$$\mathcal{M}^{(t)} = \begin{bmatrix} \mathcal{M}_{in}^{(t)} \\ \mathcal{M}_{out}^{(t)} \end{bmatrix} = \begin{bmatrix} \left[m_{(v,v')}^{(t)} \right]_{v,v' \in \mathcal{V}} \\ \left[m_{(v,v')}^{(t)} \right]_{v,v' \in \mathcal{V}}^{\top} \end{bmatrix} \tag{16a}$$

$$\begin{aligned}\mathbf{H}^{(t)} &= W \cdot \mathcal{M}^{(t)} \cdot \mathbf{h}^{(t-1)} + b \\ &= [W_{in}\ W_{out}] \cdot \begin{bmatrix} \mathcal{M}_{in}^{(t)} \\ \mathcal{M}_{out}^{(t)} \end{bmatrix} \cdot \mathbf{h}^{(t-1)} + [b_{in}\ b_{out}]\end{aligned} \tag{16b}$$

where $\mathcal{M}_{\text{in}} = \left[m_{(v,v')}^{(t)} \right]_{v,v' \in \mathcal{V}}$ is the adjacency matrix in graph $\mathbf{G}^{(t)}$; "$\top$" indicates the transposition operator, i.e., $\mathcal{M}_{\text{out}}$ is actually the transposition matrix of $\mathcal{M}_{\text{in}}$.

GCN. Graph Convolution Networks [14] adopt spectral approaches to represent the graph. It computes the eigendecomposition of the graph Laplacian, defined as

$$\mathbf{H}^{(t)} = \mho^{(t)} g(\Lambda^{(t)}) \mho^{(t)\top} \cdot \mathbf{h}^{(t-1)} \tag{17}$$

where $\mho^{(t)}$ is the matrix of eigenvectors of the normalized graph Laplacian $\mathbf{L}^{(t)} = \mathbf{I}_{\mathcal{V}} - \mathbf{D}^{(t)^{-\frac{1}{2}}} \mathcal{M}^{(t)} \mathbf{D}^{(t)^{-\frac{1}{2}}} = \mho^{(t)} g(\Lambda^{(t)}) \mho^{(t)\top}$ ($\mathbf{D}^{(t)}$ is the degree matrix and $\mathcal{M}^{(t)}$ is the adjacency matrix of the graph $\mathbf{G}^{(t)}$), with a diagonal matrix of its eigenvalues $\Lambda^{(t)}$. $g(\cdot)$ is the filter function, which can be approximated by a truncated expansion in terms of Chebyshev polynomials [12].

GraphSAGE. In order to avoid transductive learning and naturally generalize to unseen nodes, Hamilton et al. [18] proposed the general inductive framework, GraphSAGE, which generates new representation by sampling and aggregating features from a node's local neighborhood. The difference between this approach and the aforementioned GGNN (Eq 15) is that the former does not utilize the full set of neighbors, but rather fixed-size set of neighbors through uniform sampling.

GAT. Graph Attention Networks adopt a *self-attention* strategy, which involves computing the representations of each node attending to it over its neighbors. The attention coefficients are computed in the node pair (v, v')

$$\alpha_{(v,v')}^{(t)} = \frac{\exp\left(\text{LeakyReLU}\left(W\left(\mathbf{h}_v^{(t-1)} \oplus \mathbf{h}_{v'}^{(t-1)}\right)\right)\right)}{\sum_{v'' \in N(v)} \exp\left(\text{LeakyReLU}\left(W\left(\mathbf{h}_v^{(t-1)} \oplus \mathbf{h}_{v''}^{(t-1)}\right)\right)\right)} \tag{18}$$

where $\alpha_{(v,v')}^{(t)}$ is the attention coefficient of node v and v' in $\mathbf{G}^{(t)}$, which reweights the edge $m_{(v,v')}^{(t)}$. We can then adopt an approach similar to Eq 3 to obtain $\mathbf{H}_v^{(t)}$ of each node.

A.4 Hyperparameter Settings

We have discussed several important hyperparameter settings of the proposed model in Section 4.5. We conduct *grid search* for our proposed model and baselines in order to find the adaptive hyperparameters and compare fairly. The remaining aspects of parameter options are introduced below to facilitate better reproductivity.

Hyperparameters in EvoNet. We test EvoNet at the number of states $|\mathcal{V}|$, segment length τ, the size of graph-level representation $|\mathbf{U}|$ (the size of node-level representation $|\mathbf{h}|$ is determined by state recognition, since $\mathbf{h}_v^{(0)} = \mathbf{\Theta}_v$), while the search space may differ between different datasets. We test $|\mathcal{V}|$ with values from 5 to 100 with interval 10, and further test τ with different lengths that are smaller or greater than the period length of the corresponding dataset. We test $|\mathbf{U}|$ from 2^4 to 2^{10} with exponential interval 1. In batch-wise training for EvoNet, the batch size is set to 1000, and we choose the Adam algorithm [22] as the loss optimizer.

Hyperparameters in baselines. As for baselines, we use the source code provided on *TSLearn*[7] for several feature-based models, and code the sequential models by ourselves. For the graphical models, we conduct the experiments on the provided codes in GitHub. If the parameter interface is open, we adopt the same grid search approach to search the best parameters. Due to the binary event prediction tasks, we use XGBoost [8] with same parameters for all methods in order to improve the overall performance.

[7]https://tslearn.readthedocs.io/en/latest