AnRAD: A Neuromorphic Anomaly Detection Framework for Massive Concurrent Data Streams

Qiuwen Chen, Ryan Luley, Qing Wu, *Member, IEEE*, Morgan Bishop, *Member, IEEE*, Richard W. Linderman, *Fellow, IEEE*, and Qinru Qiu, *Member, IEEE*

***Abstract*— The evolution of high performance computing technologies has enabled the large-scale implementation of neuromorphic models and pushed the research in computational intelligence into a new era. Among the machine learning applications, unsupervised detection of anomalous streams is especially challenging due to the requirements of detection accuracy and real-time performance. Designing a computing framework that harnesses the growing computing power of the multicore systems while maintaining high sensitivity and specificity to the anomalies is an urgent research topic. In this paper, we propose anomaly recognition and detection (AnRAD), a bioinspired detection framework that performs probabilistic inferences. We analyze the feature dependency and develop a self-structuring method that learns an efficient confabulation network using unlabeled data. This network is capable of fast incremental learning, which continuously refines the knowledge base using streaming data. Compared with several existing anomaly detection approaches, our method provides competitive detection quality. Furthermore, we exploit the massive parallel structure of the AnRAD framework. Our implementations of the detection algorithm on the graphic processing unit and the Xeon Phi coprocessor both obtain substantial speedups over the sequential implementation on general-purpose microprocessor. The framework provides real-time service to concurrent data streams within diversified knowledge contexts, and can be applied to large problems with multiple local patterns. Experimental results demonstrate high computing performance and memory efficiency. For vehicle behavior detection, the framework is able to monitor up to 16 000 vehicles (data streams) and their interactions in real time with a single commodity coprocessor, and uses less than 0.2 ms for one testing subject. Finally, the detection network is ported to our spiking neural network simulator to show the potential of adapting to the emerging neuromorphic architectures.**

***Index Terms*— Anomaly detection, general purpose graphics processing unit (GPGPU), heterogeneous systems, machine learning, neuromorphic computing.**

Manuscript received April 20, 2016; revised September 5, 2016 and January 11, 2017; accepted February 17, 2017. This work was supported by the Air Force Research Laboratory under Contract FA8750-12-1-0251.

Q. Chen is with the Department of Electrical Engineering and Computer Science, Syracuse University, NY 13224 USA during the time of the work (e-mail: qchen14@syr.edu).

R. Luley, Q. Wu, and M. Bishop are with the Air Force Research Laboratory, Information Directorate, RITB, Rome, NY 13441 USA (e-mail: ryan.luley@us.af.mil; qing.wu.2@us.af.mil; morgan.bishop@us.af.mil).

R. W. Linderman was with the Air Force Research Laboratory, Information Directorate, RITB, Rome, NY 13441 USA. He is now with the Office of the Under Secretary of Defense (Acquisition, Technology and Logistics), Washington, DC 22202 (e-mail: richard.w.linderman.civ@mail.mil).

Q. Qiu is with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244 USA (e-mail: qiqiu@syr.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2017.2676110

I. Introduction

DETECTING abnormal data stream is not a trivial task. First of all, labeled training data are often difficult or expensive to obtain, and not all abnormal classes appear in the available training set. Second, the algorithm must be capable of real-time learning and detection in order to handle nonstopping incoming data streams. Third, anomalies occur in different contexts, and thus it is desirable for the detection framework to handle diverse data distributions adaptively. These restrictions exclude most of the classical anomaly detection approaches, which rely on classification or off-line examination. Although many novel techniques have been proposed to address the aforementioned challenges [10], a few of them are specially tuned to harness the ever-growing computing power of the parallel architecture.

This paper presents anomaly recognition and detection (AnRAD), an autonomous anomaly detection framework on heterogeneous multicore platforms, which provides real-time unsupervised learning and prompt anomaly detection in time series data streams. It is motivated by the idea that human information processing is massively parallel and the learning is from the associations among the features extracted from unlabeled data. Those correlated features form an expectation of normality and any unexpected patterns will cause surprises.

The AnRAD framework learns the structure of a probabilistic inference network [15] using unlabeled training data. The configuration is comprised of a brief arrangement of nodes, which express certain features or feature compositions. These nodes are referred to as *lexicons*, since they store the symbols of all conceivable features. In this paper, we refer to node and lexicon interchangeably. The connections between nodes capture their associativity. Among the nodes, those with fan-in connections are *key nodes*, which serve as the primary testing units. With a learned topology, the network ingests new data and refines the link weights. The collection of weights between the symbols of two lexicons is implemented as conditional probability matrices, whose nonzero elements are named *knowledge links*. A new input pattern is tested with respect to the trained network for the "amount of surprises" in each key node, and the results are accumulated to make a network-wide decision. Parallel implementations are deployed

to speed up the calculation. Our framework is applied to a variety of tasks, including vehicle behavior monitoring and host/network-based intrusion detection.

The approach has several desirable characteristics. First, it does not assume the availability of labeled training data. Second, it considers both the spatial and temporal association among features; hence it can detect either an abnormal data point or abnormal time series. Third, the learning and inference resemble the neuron system, thus can be implemented using neuromorphic hardware/software. Finally, by pushing the complexity to the feature (lexicon) space during the self-structuring stage, AnRAD is able to perform accurate detection with simple network topology, which enables fast online learning and high concurrency. From the implementation perspective, the framework is inherently parallel: networks constructed for different testing instances can be processed independently; within each network, the inconsistency tests of different key nodes can be processed in parallel; for each key node, the likelihoods of all possible observations are also assessed concurrently. The parallelism in different layers can be exploited by the state-of-the-art many-core processors to offer computation acceleration and model scalability.

This paper focuses on both the algorithm design and the implementation. On the one hand, we investigate the learning and model construction techniques that improve the detection accuracy. On the other hand, we exploit high performance computing architectures to enable real-time performance and scalability. The main contributions are summarized as follows.

1) An algorithm analysis is presented that compares the training complexity of AnRAD with that of a neural network. It shows that the Bayesian property of AnRAD enables it to use much less training samples to achieve the asymptotic error. The discovery explains some of our design choices adopted for network structuring and incremental learning.
2) The detection accuracy of the self-structuring network is evaluated with discussed experimental setups, and compared with both traditional and neuromorphic detectors.
3) We extend the self-structured network [15], which handles single data stream and single normal model, to a more general framework that monitors concurrent streams following diversified behavioral patterns.
4) The complexity of the recall algorithm is analyzed. Fine-grained parallelization as well as efficient design of memory layout are implemented and benchmarked on different multicore architectures. More than 1000× speedup over CPU program is achieved. Up to 16 000 subjects can be handled in real time using a commodity coprocessor.
5) The tradeoff between computational speed and power consumption is tested and analyzed.
6) We port the AnRAD network to our spiking neural network (SNN) simulator [3] to demonstrate its close relevancy to emerging neuromorphic architectures.

The rest of this paper is organized as the following. Section II reviews some existing works in anomaly detection and machine learning using High Performance Computing (HPC). Section III introduces and analyzes the detection algorithm. Section IV extends the framework and reviews the self-structuring method. Section V evaluates the detection accuracy and compares AnRAD with the baselines. Section VI implements the parallel recall algorithm on computing platforms, and presents testing results under different setups. In Section VII, we run the detection network on spiking neural simulator. Finally, Section VIII concludes this paper.

II. Related Works

Many studies have been carried out for outlier detection [5], [10], [11]. Classification methods, such as support vector machine [27], use labeled data to train classifiers that assign input samples into normal or abnormal classes. Density-based detectors assume that outliers are distant from their neighbors. Local reachability distance [6] or distance ranking [24] has been used to measure the local density. Online approaches [28], [39] are also studied. Clustering methods do not require labels, and expect that regular samples appear in clusters while outliers are sparse [9]. The parametric statistical models learn the normal data distributions and suppose the abnormal data happen with low probability [43]. Graphic models, such as conditional random fields [1], are studied to capture the spatial-temporal features [45]. AnRAD is also a graph-model-based approach. It differs from the previous works, because our self-structuring algorithm enables a short and concurrent inference pipeline for parallel implementations.

Since AnRAD features a bioinspired detection mechanism, we are interested in other neuromorphic approaches for anomaly detection. Replicator neural network (RNN) [20] trains symmetric hidden layers to reconstruct the input sample, and uses the reconstruction error as the anomaly indicator. Self-organized map (SOM) [8], [42], [44] leverages competitive training to map the high-dimensional data into 2-D neuron layers. Testing samples' abnormalities are ranked based on their distances to the best matching units (BMUs). Growing hierarchy self-organizing maps [26], [38] are studied to overcome the static network structures. Hierarchical temporal memory (HTM) [19] is a neuromorphic model based on the cortical learning algorithm. Anomalies are identified by the percentage of active pooler columns that were falsely forecasted [36]. Most existing methods are the derivations of neural networks, and a very few are based on the inference networks.

In recent years, the joint design of learning models and its parallel implementation has received extensive consideration [4]. The intelligent text recognition system [40] explores the confabulation [21] network to achieve machine reading on heterogeneous platform with IBM cell processors. Ahmed *et al.* [2] further accelerated the pattern matching stage of the system with Intel Xeon Phi coprocessor. However, the association stage still uses CPU-based multithreading. Our preceding works [13]–[15] adopt the concept for anomaly detection. But they only considered single data stream with single knowledge context. The Bayesian network, as a kin to cogent confabulation, was also studied from the HPC perspective. Linderman *et al.* [29] focused on accelerating the Bayesian network learning process on graphics processing

units (GPUs). Besides HPC systems, efforts were devoted to neuromorphic architecture [18], [31] featuring SNN, which was also proposed to efficiently represent spatial-temporal memory [23].

III. Detection Algorithms

A. Confabulation-Based Anomaly Detection

Following our previous work [14], [15], we adopt *cogent confabulation* [21] as the computing model for probabilistic inference. Cogent confabulation is an association-based cognitive model, which connects the dependent symbolic features. It describes the basic dimensions of the observation using a set of features (e.g., color and shape). The attributes of a given feature (e.g., red color and round shape) are called *symbols*, which analogize the biological neurons. Their pairwise conditional probabilities are called *knowledge links*, which are equivalent to the interneuron synapse plasticity. The link updates follow Hebbian learning, and can be potentially learned using Spike Time Dependent Plasticity (STDP) rule [34], which is known to be the biological process to adjust the strength of neuron connections. The neurons capturing the same feature (e.g., shapes) are collected in the same lexicon, while the knowledge links among neurons of two lexicons are stored as a matrix of conditional probabilities, where the i, jth element of the matrix is the probability $p(s_i|t_j)$ of symbol s_i in the source lexicon given t_j in the target lexicon. We refer to the presence of such probability matrix as a *connection* between two lexicons. The lexicons and their connections constitute a graph; accordingly, we also denote lexicons as *nodes*. The knowledge links are formed and tuned when the corresponding neurons are activated simultaneously.

Whenever a feature input is received, the associated neuron is triggered, and an activation is transmitted to other neurons over the knowledge links. For a neuron t in lexicon l, its excitation is computed as the sum of all fan-in links by

$$y(t) = \sum_{k \in F_l} \left\{ \sum_{s \in S_k} \left[I(s) \ln \frac{p(s|t)}{p_0} \right] + B \right\}, \quad t \in S_l \tag{1}$$

where t denotes one of the neurons in lexicon l. F_l is the set of lexicons that connect to l, and S_k is the set of symbols in lexicon k. Given the occurrence of symbol s, $I(s)$ indicates 0 or 1. p_0 is the minimum probability empirically selected to ensure that t has positive excitations. B is called bandgap; it favors those neurons that gather more excitations from distinct nodes. We use $B = 0$ in this paper to let the abnormality be determined only by the synaptic weights. Basically, the excitation level of neuron t is its log-likelihood given the status of other neurons. In the implementation, we store $v(s,t) = \ln((p(s|t))/p_0)$ as the value of knowledge links for faster calculation.

A set of lexicons are selected and referred to as the *key lexicons*. They are the essential testing entities. The rests are not tested, and are called the *supporting lexicons*. Knowledge links are formed from supporting lexicons to key lexicons and among key lexicons. Equation (1) computes the excitation levels of all neurons in a key lexicon, and the one with the highest likelihood is elected as the predicted symbol $t_{\max}$. Given the input t, the *anomaly score* of a key lexicon is evaluated using

$$\alpha_l(t) = \frac{y(t_{\max}) - y(t)}{y(t_{\max})}, \quad t, t_{\max} \in S_l. \tag{2}$$

As shown in (2), we use the normalized discrepancy between excitations of the input symbol t and the reference symbol $t_{\max}$ as the node anomaly score. Here, $t_{\max}$ is the symbol that holds the highest excitation $y(t_{\max})$ in lexicon l. The score indicates the cogency between the context and the input. The individual key lexicons are merged to obtain the network anomaly score by weighting their excitation levels with the priors $y^*(t) = y(t) + \ln(p(t)/p_0$. Substituting y in (2) with y^*, the network anomaly score A is obtained by

$$A(t_{l=1\ldots L}) = \frac{\sum_{l=1}^{L} \alpha_l^*(t_l)}{L} \tag{3}$$

$$\alpha_l^*(t) = \frac{y^*(t_{\max}) - y^*(t)}{y^*(t_{\max})}, \quad t, t_{\max} \in S_l \tag{4}$$

where L is the total number of key lexicons. t_l is the observed symbol of lexicon l, and α_l^* is the prior-weighted node score. The accumulated score is in the range of [0, 1].

B. Algorithm Analysis

While most existing neuromorphic methods are based on neural networks, we choose inference network instead. The motivation lies in the size of the training sample for anomaly detection. According to Zimek *et al.* [47], ensemble of models with smaller samples is preferable for anomaly detection compared with training a large model with all the data. Although confabulation network, which models $p(s,t)$, is supposed to have a higher asymptotic error than that of neural network which directly learns $p(t|s)$, it approaches the bound faster. In other words, the inference network provides the foundation for learning accurate models with small sample (Section IV-B).

In (1), $p(s|t)$ is learned by $\hat{p}(s|t) = c(s,t)/c(t)$, where $c(.)$ counts the occurrences of the event in the training set $\{s^i, t^i\}_{i=1}^M$. Here, M is the training sample size. To simplify the analysis, let $B = 0$, $S_l = \{t, t'\}$, and $\sum_{s \in S_k} I(s) = 1, k \in F_l$. A threshold of 0 is used for (2), i.e., raising alarms whenever $t \neq t_{\max}$. Let s_k denotes the only activated input symbol in the kth supporting lexicon. Assume that the observation t is abnormal, it can be detected if and only if inequity (5) hold true

$$\varphi_l(s) = \sum_{k=1}^{|F_l|} \ln \frac{\hat{p}(s_k|t')}{\hat{p}(s_k|t)} > 0. \tag{5}$$

That said, under the input, t' is predicted as $t_{\max}$ (i.e., the expected symbol). If $\varphi_l(s) < 0$, a false negative error is generated. In the following, we focus our discussion only on how to bound the false negative error; the discussion of false positive error is similar by taking the reciprocals for each summation term in (5). With two symbols in the key lexicon, the anomaly detection problem is simplified to a binary classification problem. The classifier is defined

$h_{\text{CFB}} : s \to S_l$ and the asymptotic version is h^{∞}_{CFB}, which is trained by infinite amount of data. We compare h_{CFB} with logistic regression h_{NN} as an example for neural networks, and use the same incoming connections F_l.

We use $\varepsilon(.)$ to denote the error rate of a mapping. According to previous studies [35], the asymptotic error rate is smaller for neural networks, i.e., $\varepsilon(h^{\infty}_{\text{NN}}) < \varepsilon(h^{\infty}_{\text{CFB}})$. However, for some constant ϵ_0, to make $\varepsilon(h_{\text{NN}}) \leq \varepsilon(h^{\infty}_{\text{NN}}) + \epsilon_0$ with high probability, we need sample size $M = \Omega(|F_l|)$, i.e., in the order of the learned parameters.

In the case of confabulation, the learned parameters $\hat{p}(s_k|t)$ can approach the asymptotic version $p(s_k|t)$ with less data. Let some $\epsilon > 0$, $c(t) = \beta M$ for $0 < \beta < 1$, $\delta = \epsilon\sqrt{\beta M}$, by additive Chernoff Bound [33], for each knowledge link, we have

$$\begin{aligned}
&\Pr[|\hat{p}(s_k|t) - p(s_k|t)| \geq \epsilon] \\
&\quad = \Pr[\beta M|\hat{p}(s_k|t) - p(s_k|t)| \geq \sqrt{\beta M}\delta] \\
&\quad \leq 2e^{-2\delta^2} = 2e^{-2\beta M\epsilon^2}. \qquad (6)
\end{aligned}$$

Since there are $2|F_l|$ such parameters, to make the union bound of the error $2|F_l| \cdot 2e^{-2\beta M\epsilon^2} \leq \rho$ for some constant $\rho > 0$, it suffices to pick $M = O(\ln|F_l|)$. In other words, with high probability, $\hat{p}(s_k|t)$ is within $\omega = O((\ln|F_l|/M)^{1/2})$ of $p(s_k|t)$.

To bound the error rate, we consider the case when h_{CFB} makes a false negative and h^{∞}_{CFB} does not [i.e., $\varphi^{\infty}_l(s) > 0$ and $\varphi_l(s) < 0$]. This happens when $\varphi^{\infty}_l(s)$, obtained by replacing $\hat{p}(s_k|t)$ with $p(s_k|t)$ in (5), is within $(0, \omega|F_l|)$. So the difference with the ideal model can be represented by

$$\varepsilon(h_{\text{CFB}}) \leq \varepsilon\big(h^{\infty}_{\text{CFB}}\big) + \Pr\big(\varphi^{\infty}_l(s) \in (0, \omega|F_l|)\big). \qquad (7)$$

Given the normal pattern t', by the nonnegativity of KL-divergence, each term of $\varphi^{\infty}_l(s)$ has positive mean and $\Omega(1)$ of them is far away from 0. So the expectation $E[\varphi^{\infty}_l(s)] = \Omega(|F_l|) = \gamma|F_l|$ for some $\gamma > 0$. Based on the assumption of confabulation model that all $p(s_k|t)$ are independent, by Chernoff Bound [33], let $\delta = (\gamma - \omega)/\gamma \in (0, 1)$, we have

$$\begin{aligned}
\Pr\big[\varphi^{\infty}_l(s) \in (0, \omega|F_l|)\big] &\leq \Pr\big[\varphi^{\infty}_l(s) < (1-\delta)\gamma|F_l|\big] \\
&< e^{-\delta^2\gamma|F_l|/2} \leq O(e^{-(\gamma-\omega)^2|F_l|}) \qquad (8)
\end{aligned}$$

which is exponentially small with respect to $|F_l|$ when ω is a constant. Therefore, by picking $M = \Omega(\ln|F_l|)$, with high probability, we have $\varepsilon(h_{\text{CFB}}) \leq \varepsilon(h^{\infty}_{\text{CFB}}) + \epsilon_0$ for some constant $\epsilon_0 > 0$.

From the analysis, we draw design insights. First of all, in order to satisfy the assumption of lexicon independence, we will decouple the lexicons with their feature distance during the network self-structuring procedure (Section IV-A). Furthermore, the confabulation model has higher asymptotic error for classification than a neural network, but can approach it $\Omega(\ln|F_l|)$ faster. So it is beneficial to apply incremental learning where the network coefficients are learned by merging the trained results from multiple short episodes of the training sequences (Section IV-B).

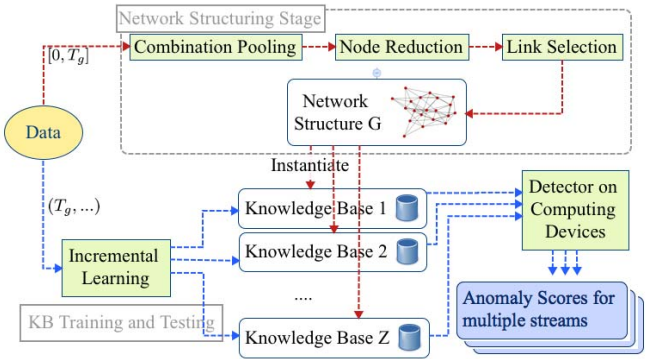


Fig. 1. AnRAD workflow.

IV. Network Construction and Learning

The mechanism of cogent confabulation [21] is similar to that of a probabilistic graphic model. Activating the power of such simplified model usually requires carefully tuned network architecture. Instead of going deep and complicated in the network inference pipeline, AnRAD pushes the complexity to the initial structuring stage and builds hierarchical structure in the lexicon design space. This approach results in a very succinct network configuration. In fact, the anomaly inference only propagates two layers (support lexicons to key lexicons and key lexicons to score), and every excitation integration can be parallelized (Section VI). The overall model is a simple but highly concurrent network built on top of unified processing elements (i.e., neurons), which is analogous to the massive parallel structure of the biological neural system. The method enables fast online learning and highly concurrent detection.

AnRAD is an inference-based anomaly detection framework (Fig. 1), whose inputs can be represented as N streams $\{\{x^1_1, x^2_1, \ldots, x^t_1, \ldots\}, \{\ldots, x^t_2, \ldots\}, \ldots, \{\ldots, x^t_N, \ldots\}\}$. Here, x^t_n represents a record tuple of the nth stream at time t. The qth feature of the record is labeled as $x^t_n(q)$, while the total number of features is denoted by Q. An initial duration of streams at time $[0, T_g]$ is used to configure the network structure G using our self-structuring algorithm [15]. The topology captures the general correlations between the lexicons. Combination pooling finds those potentially useful feature combinations from an enormous number of possible ones; node reduction selects a succinct set of key lexicons from the pooling results; the link selection connects lexicons to learn knowledge associations.

After the network is constructed, we let multiple knowledge contexts share a global network structure, and train separate knowledge bases using their local data samples. Take wide area vehicle behavior monitoring, for example, we break the area into hundreds of small zones. Data streams (individual vehicle trajectories) at time $(T_g, T_0]$ are directed to the local zones $\Psi = \{1, 2, \ldots, Z\}$ to tune the knowledge bases $\Theta^G_z(T_g : T_0), z \in \Psi$ [i.e., modeling $p(s|t)$'s]. Then, continuous streams starting from T_0 will be tested using these knowledge bases. Meanwhile, the new samples also improve the knowledge bases $\Theta^G_z(T_g : t)$ by performing incremental learning. A sliding window of length W, $\{x^{t-W}_n, \ldots, x^{t-1}_n, x^t_n\}$ is used at each time frame. The anomaly detection module

is accelerated by the state-of-the-art multicore processors for real-time processing. The performance of the inference network largely depends on the quality of the knowledge graph. In this section, we inherit the self-structuring procedure [15] to construct the confabulation network.

A. Self-Structured Inference Network

1) Hierarchical Key Lexicon Organization: Since confabulation only models the first-order dependency of features, more complex feature interaction has to be captured by building lexicons that record feature compositions. Our architecture of network contains lexicon hierarchy in which nodes at higher levels are constructed as the combinations of nodes at lower layers. At the bottom level, the lexicons store singe features. They are referred to as the primary features, and are predefined to describe the input samples. Those lexicons at higher levels combine primary features, and they model more abstract interaction patterns between features. Because the lexicons store discrete symbols, real-valued features are quantized using equal-width bins before activating the lexicon symbols. The combination process is applied to both the feature and temporal domains. For instance, we can combine multiple features from the same time frame, such as $\langle x_n^t(q), x_n^t(q')\rangle, q, q' \in Q$, or the same feature from multiple time frames as in $\langle x_n^t(q), x_n^{t-\Delta t}(q)\rangle$, $\Delta t < W$. Thus, temporal patterns can also be modeled.

Such layered feature composition translates data tuples into neuron activations in the lexicons, but with increasing Q and W, the network's complexity may increase exponentially. To reduce the complexity and improve the accuracy, AnRAD adopts a pooling-and-reduction procedure to optimize the network structure.

Feature combination pooling generates an initial set of lexicon candidates to capture higher order associations. As shown in [15], combinations of sufficiently correlated features can detect anomalies that are indistinguishable to single features. Therefore, we define $d(q_i, q_j) = [1 - \mathrm{MI}(q_i, q_j)] \in [0, 1]$ as the feature distance, in which MI(.) calculates the normalized mutual information between the features q_i and q_j. A smaller distance indicates higher correlation between the two features. To determine whether a composition Q_l is kept in the lexicon candidate set, the pooling process performs a relevancy test as in

$$\Gamma(Q_l) = \prod_{q_i, q_j \in Q_l} I[d(q_i, q_j) < d_{\mathrm{prox}}] \tag{9}$$

where $I(.)$ is the identity function that equals to 1 when the test result is true.

The test ensures that all component features of composition Q_l are close to each other. The pooling process applies the test to a subset of composition features whose cardinality is less than a predefined *max_order*. The features, which pass the relevancy test, will be chosen as the lexicon candidates. We use parameter d_{prox} to control the size of the candidate pool.

2) Key Lexicon Reduction: After the pooling, a set of most relevant lexicon candidates are selected, and node reduction further packs them into smaller size. Since the training data do not contain labels, we adapt a feature clustering approach [32] to remove the redundant candidates. The process clusters the compositions based on the feature distance, and then uses only the medoids to represent the clusters of candidates. Using the normalized mutual information as the distance $d(Q_l, Q_{l'})$ between the combinations, the algorithm repeatedly prunes the candidate set by K-nearest-neighbor approach. At every iteration, the node who has the smallest K-distance is chosen, while its K neighbors are eliminated from the candidates. Then, the K value is reduced until the next most compact candidate has a K-distance smaller than the given threshold [15]. We repeatedly prune until K reduces to 1. The selected medoids form the key lexicons.

In order to detect abnormal patterns in temporal domain, we extend the feature composition to the temporal domain. The lexicons sampled from different frames form a new set of primary features $Q_l^{-t} = \{Q_l^0, Q_l^{-1}, \ldots, Q_l^{-W}\}$, where the superscripts give their time stamp. The similar key lexicon selection method is then applied on the feature set to construct new combinations and a key node is expressed as a 2-D composition $R_l \sim [(q_{l_1}, q_{l_2}, \ldots, q_{l_i}, \ldots)^{-t_1}, (\ldots, q_{l_i}, \ldots)^{-t_2}, \ldots (\ldots, q_{l_i}, \ldots)^{-t_j}, \ldots]$.

3) Link Selection: We use the mutual information among lexicons to identify the set of supporting lexicons that best infer the symbols in the key lexicon. The link selection maximizes the relevancy between key nodes and their supporting nodes, and minimizes the dependency among those supporting nodes connecting to the same key node. Here, the supporting nodes are selected from those primary features, because we have already modeled the composition features with key lexicons. For each key lexicon, we select lexicons representing single features $\{q^{-t}, q \in Q, t < W\}$. Starting from a target key node R_l, the method [15] first sorts the candidates by their feature distance to R_l in ascending order. Traversing the sorted list, the process puts the feature to result supporting nodes only when: 1) it does not belong to the feature combination of R_l; 2) it has close distance to R_l; and 3) it is distant from those supporting nodes that have already been selected for R_l.

After the processes, the network is configured properly. Note that the structuring stage only configures the connections among the lexicons, but the knowledge links of the connections (between symbols) are established and strengthened during the online learning of local knowledge bases.

B. Incremental Learning

Based on the network configuration, AnRAD builds the knowledge bases using the new data streams. It could simply count co-occurrences $c(s, t)$ from all the training samples, which works fine for prediction tasks. However, a large training set size does not necessarily improve the accuracy of anomaly detection. Based on the analysis in Section III-B, our framework uses incremental learning with episodes [15]. We periodically reset the co-occurrence counters after time T. New counter values merge into those of the previous episodes to update the knowledge link weights

$$v^{\Lambda+1}(s,t) = \frac{v^{\Lambda}(s,t)\Lambda + \ln[p(s|t)/p_0]}{\Lambda+1} \tag{10}$$

$$v^{0}(s,t) = \ln\left(\frac{p(s|t)}{p_0}\right) \tag{11}$$

where v^{Λ} is the stored knowledge value at episode Λ. It can be substituted to excitation $y(t) = \sum_{k\in F_l}\{\sum_{s\in S_k}[I(s)v^{\Lambda}(s,t)] + B\}$. In the first training episode, i.e., v^0, the knowledge value is calculated following (1). Basically, the episode update functions as the ensemble of subsamples in time domain.

V. Evaluations

A. Data Sets

Three different tasks are studied for evaluating the AnRAD framework.

1) Vehicle Traces [13], [14]: In this task, vehicle behavior data are collected from a road network. The data are processed to obtain ten primary features, among which five are related to an individual vehicle (vehicle size, coordination, velocity, and direction), and the other five are related to vehicle interactions (the distance to neighboring vehicles, their relative speeds, positions, directions, and sizes). The data are collected at a sampling period of 1 s. The area is partitioned into 342 detection zones using traffic density balance method [14]. The training set contains normal records with length of 240 min. The testing set contains 10 min of unseen records

2) DARPA Intrusion Detection Data Set [30]: We extract network traffic statistics from the DARPA 1998 packages. Data tuples are generated in 300-ms sampling frames for each endpoint pair; 21 primary features are extracted, which contain information, such as port number, byte/package count, TCP flags, and so on. Since we focus on real-time detection of concurrent streams, we did not use the preprocessed KDD 99 [48], because it lumps data sequence into sessions and the detection can only be made at the end of a session. The model construction is fully automated with almost no expert knowledge of attacks. The normal data for learning have 20k frames sampled from the seven weeks of training period. For testing, the data contain 7k normal streams and 422 abnormal streams from 24 attack classes.

3) ADFA-LD [16], [17]: The data set consists of syscall traces of malicious and benign programs. A clean training set containing around 10k system calls are sampled. Also, we build another tainted training set of about 50k system calls, among which 1/5 of the samples are randomly extracted from the attacks and treated as normal during training. This is to evaluate the detection performance under nonideal or compromised training set. For testing, the normal data contains 6k programs and the abnormal data has 746 attacks.

B. Comparison Methods

1) Incremental Local Outlier Factor [39]: It is the incremental version of the classical density-based local outlier factor (LOF) detector [6]. Given a testing subject, the method uses the ratio between the neighbors' local reachability distances and that of the testing sample's as the anomaly indicator.

2) Cross-Feature Analysis [7]: This is a fast rule-based unsupervised detector. For each feature, the method builds a CART decision tree and uses the other features to infer the probability of the target feature. Then, the probabilities from all trees are summed to indicate the abnormality.

For neuromorphic baselines, we reproduce the followings.

3) Replicator Neural Network [20]: Similar to an autoencoder, the method builds a symmetric five-layer neural network and uses back propagation to minimize the reconstruction error. The mean squared error between the input features and the reconstruction outputs is used for the anomaly score.

4) Self-Organizing Map: The neural network uses competitive learning to map the high-dimensional data to a 2-D neural layer. It was used for anomaly detection in various fashions [8], [42]. In this baseline, we use the sum of input's distances to its nearest BMU to detect anomalies [44].

5) Hierarchical Temporal Memory [19]: The emerging neuromorhpic model based on cortical learning algorithm and sparse coding [41] identifies anomaly by the percentage of active spatial pooler columns that were incorrectly predicted by the temporal pooler [36]. The method works in streaming fashion and does not require sliding windows for the input sequences. For multifeatured data set (DARPA), we train predictive models for each feature, and generate the anomaly score by summing up scores from each feature model. All hyperparameters are selected using the in-package swarm algorithm.

All evaluation methods output anomaly scores for each data frame. We use the leaky bucket method to make an anomaly decision from the score sequences. Whenever a score exceeds a threshold, we add one unit to the bucket. Otherwise, one unit is leaked from the bucket. An anomaly sequence is reported if the bucket overflows. We vary the score threshold to find tradeoff between detection rate and false alarm and also to analyze the sensitivity of the bucket size. The comparative studies are conducted on the fully labeled public data sets (DARPA and ADFA-LD). For methods requiring continuous features (LOF, RNN, and SOM), one-hot encoding is applied to the system calls. Different AnRAD configurations are tested on all three data sets.

TABLE I
AUC Scores for Local Knowledge Bases

Context	Zone 1	Zone 2	Zone 3	Zone 4
AUC	1.0	0.968	1.0	1.0

C. Vehicle Behavior Detection Over Zones

AnRAD trains local knowledge bases for different zones. To evaluate the effectiveness of the design, we pick four zones out of the whole area. A zone contains 20 to 70 normal vehicles during the 10-min test period. To each zone, five or six synthetic anomalies (speeding, deviating from roads, tailgating, and so on) are inserted randomly.

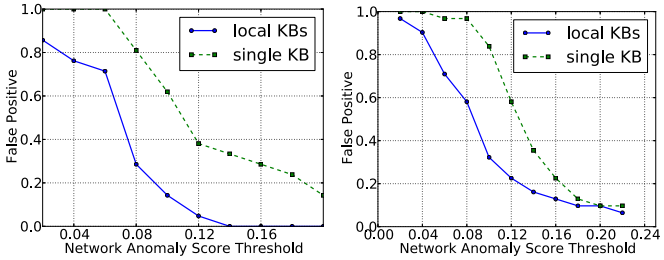


Fig. 2. Comparison between local and single knowledge bases. (a) Zone 1. (b) Zone 2.

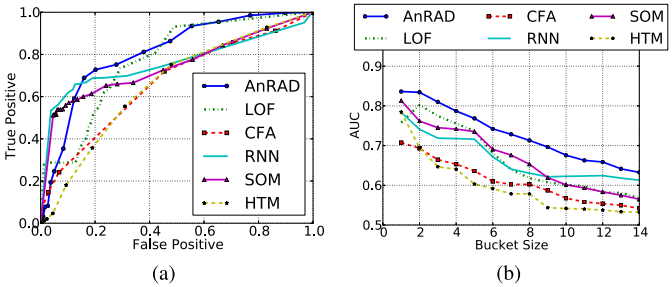


Fig. 3. Results on DARPA data set. (a) ROC curves with bucket size 4. (b) AUC with different buckets.

We first build separated local knowledge bases for each zone using their own vehicle traces. We collect the receiver operation curve (ROC) area under curve (AUC) scores generated from the test set. A high AUC score means high detection rate and low false alarm. As shown in Table I, scores of 1 or almost 1 are achieved in the four zones and the localized training is very effective with the vehicle traces.

We then compare the localized knowledge base with a single large knowledge base built for all four zones. While the single knowledge base can also achieve high detection rates, the localized training method responds better to the normal vehicles in all zones. In Fig. 2, the Y-axis gives the false positive rates when the detection rate is 1, and the X-axis specifies the thresholds of anomaly scores. The false positive rates using the local knowledge bases are about 40% lower than that of the single knowledge base at the same thresholds.

D. Comparative Evaluations

For the DARPA data set, the ROC for the comparison methods is shown in Fig. 3(a). The X-axis shows the false positive rate, and the Y-axis shows the true positive rate. The rates are the averages of all abnormal classes. The decisions are generated from the scores using leaky bucket of size 4 for all methods. LOF outperforms cross-feature analysis (CFA), because it works better with continuous features. Both neural network methods, RNN and SOM, obtain similar curves and perform better at low-false-positive region. HTM does not adapt well, because the original models are tuned for single features, and they probably should not be linearly combined. AnRAD outperforms the baselines especially at operation region with high detection rates. It also achieves the best AUC score, as shown in Table II column 2.

TABLE II
AUC SCORES FOR DIFFERENT DETECTORS

Methods	DARPA	ADFA-LD clean	ADFA-LD compromised
LOF	0.775	0.849	0.770
CFA	0.665	0.814	0.824
RNN	0.759	0.799	0.734
SOM	0.745	0.820	0.746
HTM	0.656	**0.905**	**0.872**
AnRAD	**0.810**	0.888	**0.872**

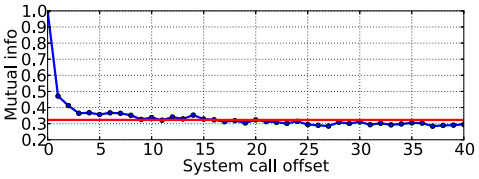


Fig. 4. Mutual Info between ADFA-LD system call and its previous calls.

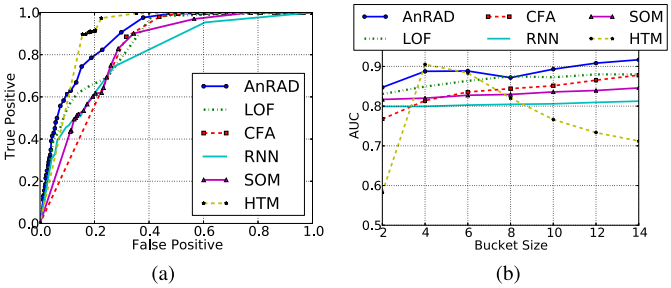
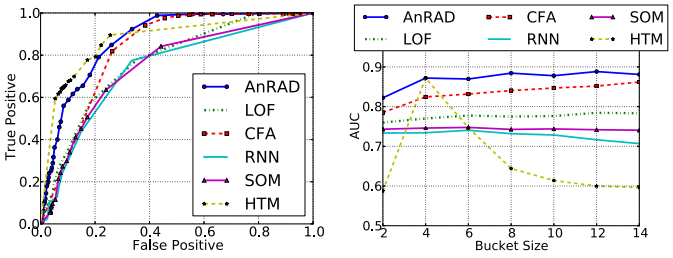


Fig. 5. Results on ADFA-LD using clean training data. (a) ROC curves with bucket size 4. (b) AUC with different buckets.

Since leaky buckets are used for all approaches, the bucket size plays an important role in the output. We also analyze the sensitivity of the detection with this parameter. In Fig. 3(b), the X-axis varies the bucket sizes and the Y-axis marks the AUC scores of the comparison methods. For DARPA data set, all approaches favor a smaller bucket, because the abnormal pattern of an intrusion usually occurs within a short time. The relative standings between the methods do not change much with different buckets.

For ADFA-LD data set, the length of the moving window is 20 syscalls, as suggested in [22]. This is justified by Fig. 4, which plots the mutual information (Y-axis) between a system call and its previous calls with different offsets (X-axis). The red line denotes the mutual information of the call with offset 20, after which the mutual information does not vary much.

Fig. 5(a) shows the ROC curves when using clean training set and bucket size 4. HTM has the best result in this case. Such univariate data do not fully exploit the lexicon compositions, but AnRAD still achieves a competitive result in Table II column 3. Against different bucket sizes in Fig. 5(b), AnRAD has the most robust performance. HTM outputs bipolar scores (either near 1 or near 0) alternatively in this data set, therefore is more sensitive to the buckets.



(a) (b)

Fig. 6. Results on ADFA-LD using tainted training data. (a) ROC curves with bucket size 4. (b) AUC with different buckets.

TABLE III
AUC SCORES FOR DIFFERENT NETWORK STRUCTURES

Methods	DARPA	ADFA-LD clean	ADFA-LD compromised
Random	0.711	0.821	0.810
AnRAD	**0.810**	**0.888**	**0.872**

For compromised ADFA-LD training data, AnRAD uses the incremental learning algorithm to counter the effects. In Fig. 6(a), the neural networks and LOF do not perform well in tainted training data. AnRAD only suffers a small accuracy degradation according to Table II, and it is advantageous at high-true-positive region. In Fig. 6(b), the AnRAD performance is stable and leads the AUC scores over different bucket sizes.

These results show that AnRAD's detection accuracy is competitive or superior to the comparison methods.

E. Effects of Self-Structuring

In this section, we present experimental data to show the advantage of using self-structured network. In this experiment, the reference designs are a set of randomly generated networks with the same size (number of key lexicons and connections) as the self-structured AnRAD network. The reference design is trained and tested in the same way as in AnRAD.

For the DARPA data set, the self-structured network consists of 123 key lexicons and 2421 connections. Ten networks with the same size are randomly generated and their average performance is reported. It is shown in Table III that the self-structuring algorithm always has better AUC scores and outperforms the random network by around 17%. Similar experiments have been carried out for ADFA-LD data set. The networks have 40 key lexicons and 410 connections. Table III shows that the self-structured network has slightly better performance than the random network with both clean and compromised training data. The advantage is smaller than that of DARPA, because system calls within the windows are more correlated, so even randomly generated feature combinations could benefit the detection.

Finally, Table IV summarizes the impact of the self-structuring algorithm to the complexity of the network. The first two rows give the maximum order of composition in the feature and temporal domain. The rest of the rows give the potential number of nodes and connections without reduction and the actual number of nodes and links after reduction. In all three data sets, our self-structuring technique achieves significant network size reduction.

TABLE IV
NETWORK COMPLEXITY IMPACT OF SELF-STRUCTURING

Datasets		DARPA	ADFA-LD	Vehicle
Composition max order	Feature	5	1	5
	Temporal	5	5	3
Key node selection	Potential	446320	21700	2548
	Selected	123	40	44
	Reduction	99.97%	99.82%	98.27%
Connections for selected nodes	Potential	12915	800	1320
	Selected	2421	410	570
	Reduction	81.25%	48.75%	56.82%

TABLE V
COMPLEXITY ANALYSIS

Model	Training time per frame	Training complexity N samples	Recall time per frame	Recall complexity per frame
LOF	4854.9ms	$O(QN^2)$	684.1ms	$O(QNlogN)$
RNN	2916.9ms	$O(TNS)$	0.27ms	$O(S)$
CFA	4.30ms	$O(QHN)$	7.78ms	$O(QH)$
AnRAD	1.57ms	$O(NLF)$	243.1ms	$O(LDF)$

Q – number of features; N – number of training samples; T – neural network iterations; S – neural network connections; H – decision tree height; L – key lexicon numbers; D – average # of neurons in a key lexicon; F – average # of active knowledge links connected to a neuron.

VI. PARALLEL IMPLEMENTATIONS

Section V demonstrates the framework's efficiency in anomaly detection. But from the aspect of sequential computation complexity, AnRAD is not superior to the other approaches during the detection. This section further leverages the highly concurrent inference structure of AnRAD to deploy the computation-intensive framework in real-time scenarios using many-core systems.

A. Complexity Analysis

To accelerate, the algorithm requires an understanding of its bottleneck first. In Table V, the computation times of different methods on the same DARPA data stream are collected. The programs are all single threaded with moderate optimization effort. In terms of training, AnRAD is much faster than the others because at each frame, it only updates a single entry in the corresponding knowledge link matrix. Therefore, real-time processing and incremental training can be achieved without much difficulty. However, the detection function of AnRAD is merely faster than the incremental LOF, whose complexity scales with the volume of the training samples. Fortunately, we can exploit the parallel structure of AnRAD to accelerate the computation.

According to (1) and (2), processing one sample has time complexity of $O(LDF)$, where L is the number of key

lexicons, D indicates the average number of neurons in a lexicon, and F denotes the average number of incoming links connecting to one neuron. The score computations of key lexicons are independent, thus the L lexicons can be assigned to parallel computation units, such as Compute Unified Device Architecture (CUDA) [49] blocks. The complexity terms D and F were induced from accumulating the value of knowledge links connecting to each candidate symbols. Such computation can be made parallel by mapping different links to different CUDA threads or vector processors.

In terms of space complexity, the AnRAD model is dominated by $O(LFU)$ in which U is the average size of the knowledge link matrices. The actual space consumption can be lower. For example, features, such as "shared links" [13], which share the knowledge matrices for links with similar context meanings (e.g., links from different neighbor vehicles to the target vehicle), may be adopted to reduce knowledge base size. Because most of the connections are sparse matrices, a compact storage format is preferable.

B. Baseline Solutions

Two baseline solutions are considered in this paper.

1) CPU Multithreading: A straightforward parallel implementation is to map each evaluation of (4) to one thread. Basically, a thread pool is allocated with a maximum number of simultaneous threads. A workload dispatcher assigns key-lexicon computations to the available threads, or waits if all the worker threads were occupied. Obviously, the limited number of CPU cores prevents us from fully exploiting the structural parallelism of AnRAD.

2) Naive GPU Implementation: General Purpose GPUs (GPGPUs) provide a potential option to fully parallelize the key-lexicon computations, because even a low-end GPU has more cores than a state-of-the-art CPU. A simple design is to directly replace each CPU thread with a GPU thread using kernel Algorithm 1. Each CUDA thread handles one key lexicon. This design may introduce the following two major problems.

Algorithm 1 Naive Recall Kernel

```
1: procedure naive_recall(KB, IN):        # KB: knowledge base; IN: input symbols
2: ref ← 0
3: for symbol t ∈ KB.lexicons[threadIdx.x]:
4:         initialize y
5:         for kl ∈ KB.lexicons[threadIdx.x].links:
6:             y += kl[t][IN[link.input_idx]] + band_gap
7:         ref = max(ref, y)
8:         obs = y if t == IN[threadIdx.x]
9: Score[threadIdx.x] = (ref − obs)/ref
```

First of all, it gives inefficient knowledge base management. The knowledge base is a set of sparse matrices. Certain types of compression are required for efficient storage. In the original CPU implementation, the knowledge link matrices are stored in hash tables, which provide efficient memory usage and $O(1)$ lookup time. However, massive concurrent random accesses from GPUs will severely degrade the cache performance and induce a lot of stalls. Second, it will lead to imbalanced workload distribution among threads. The number of symbols in different key lexicons is determined by the nature of the targeted application and they vary in a wide range. Such workload imbalance may produce serious control divergence, since the CUDA threads are executed in warps. If the threads had to wait for their neighbors for outstanding workloads, the acceleration would be completely diminished.

To address the above-mentioned limitations, we further improve the GPU implementation from the perspectives of knowledge base storage and workload distribution. We also generalize the design to other parallel architectures, such as the Xeon Phi coprocessor [25].

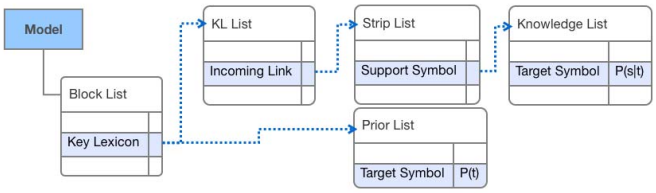
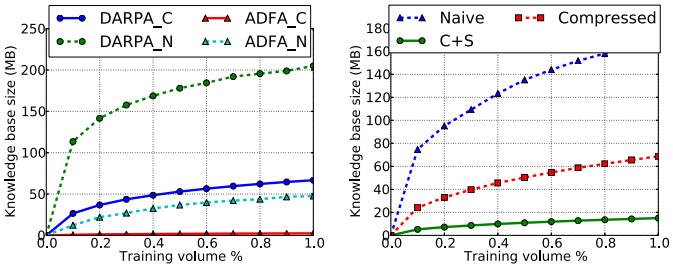


Fig. 7. In-memory knowledge base layout.

C. In-Memory Knowledge Base

To fully utilize the computing resources on the GPU, knowledge bases storage needs to be designed for both space efficiency and query convenience.

The knowledge base of confabulation network is flattened and stored in the device memory. There can be multiple knowledge bases on the device, and Fig. 7 shows the structure of one knowledge base. It maintains a *Block List*. Each entry in the *Block List* is the record of a key lexicon. It contains a pointer to the list of all incoming connections (*KL List*) to this lexicon. The key lexicon neurons' excitation levels are stored in the shared memory, and the usage of shared memory determines how many blocks can be coscheduled on a stream processor. Hence, a key lexicon may be divided into multiple blocks based on the size of its candidate symbols in order to optimize the GPU occupancy. A *KL entry* in the *KL List* has a pointer that points to the corresponding knowledge matrix. Because the matrix usually has a very high sparsity, it is stored in a list of list format (LIL). Each entry of the LIL gives the conditional probability $p(s_j|t_i)$, where s_j and t_i are symbols in corresponding support lexicon and target lexicon, respectively. Let each row of the knowledge matrix correspond to a symbols s_j and each column correspond to a symbol t_i, the LIL is arranged in row-major order. Each entry in the *Strip List* represents a row in the matrix, and each entry in the *Knowledge List* represents a nonzero entry in that row. Since different target candidates $t_i, t_i \in key\_lexicon$ need knowledge links from the same support symbol $s_j, s_j \in support\_lexicon$ to calculate their excitation level, this arrangement makes sure that the algorithm accesses *Knowledge List* sequentially for a better memory locality. Finally, the block entry also contains pointers to the prior probabilities (*Prior List*) of the key symbols for the calculation of (3).



(a) (b)

Fig. 8. Memory usage of individual models. (a) DARPA and ADFA usages. (b) Vehicle usage of zone 190.

The size of the trained knowledge bases for DARPA and ADFA data sets is plotted in Fig. 8(a). The naive implementations (DARPA_N, ADFA_N) give the potential size of knowledge base without compression. As we can see, the memory usages grow quickly as the size of the training data increases. The compressed implementation (DARPA_C, ADFA_C) compresses the sparse matrices using LIL and reduces the memory consumptions significantly. For DARPA and ADFA data sets, it gives 67% and 95% reduction in memory usage, respectively. Such improvement is particularly notable for ADFA, because features associated with the system calls are extremely sparse. For both implementations, we can see that the size of knowledge base gradually becomes stabilized.

Shared knowledge link is also implemented, where the knowledge links connecting different neighbor vehicles to the target vehicle are merged into the same probability matrices [e.g., $\langle neighbor(1).distance\rangle$ and $\langle neighbor(2).distance\rangle$ can share the same knowledge matrix]. This not only makes the nodes of interactive features more general and expose them to more training samples, but also reduces the memory usage. As Fig. 8(b) shows, using the compressed knowledge storage reduces the memory requirement by 60%, and further implementing the shared knowledge link (C+S) can give an additional 78% of memory reduction.

D. Workload Mapping and Anomaly Score Computation

Instead of mapping each key lexicon to individual threads, we map it to a CUDA block, which consists of up to 192 threads. Such implementation has two benefits: first, blocks can be dynamically scheduled, thus uneven workloads among different lexicons would no longer introduce control divergence; second, given the large number of threads in a CUDA block, different symbols in the key lexicon and different knowledge links associated to the symbol can be processed in parallel at thread level. Hence, it reduces the runtime by a factor of D and F as defined in Section VI-A).

Such workload partition and mapping is limited by the amount of hardware resources. A large key lexicon with many symbols will have to be divided and mapped to multiple CUDA blocks, in order to fit all symbols in the shared memory. When initializing the system, the learned knowledge bases are off-loaded to the device memory. The testing data are then transformed into corresponding format and dispatched to GPU frame by frame. Each CUDA block corresponds to a key lexicon. For those large lexicons, multiple CUDA blocks are assigned to them.

The kernel for computing anomaly score follows the typical MapReduce style. Two stages are defined: excitation mapping and score reduction, as shown in Fig. 9.

The mapping stage calculates the excitations of the symbols in a key lexicon, and stores them in the shared memory buffer. Consider a key lexicon l with symbol set S_l and supporting lexicon set F_l. When the kernel receives a new input for support lexicon $k \in F_l$, it uses the input symbol $s \in S_k$ to locate the activated strip from the knowledge link LILs. This strip contains the nonzero conditional probability $p(s|t)$, where t is a symbol in key lexicon. The traditional way to calculate the anomaly score of a key lexicon is to process its candidate symbols one by one. For each candidate symbol t, strips associated to all the support nodes are searched for the specific $p(s|t)$ and the values are accumulated. Such approach constantly loads different strips and hence has a low cache performance. We adopt a reversed approach that processes strips one by one. An active strip corresponding to symbol s in the support lexicon is read by multiple threads, which will then add the obtained knowledge value $p(s|t)$ to the memory location that corresponds to the excitation level $y(t)$. Atomic add is used here since the same variable $y(t)$ is accessed by multiple threads. To prevent the control divergence, the strip lengths are warp aligned so that the threads of a warp follow the same control flow. In this way, the cache performance is optimized because the strip access patterns are continuous. The excitations of the key symbols S_l are stored in the shared memory for efficient atomic addition and interthread operations. If a lexicon has more symbols than the predefined shared memory usage $U_{\max}$ (1536 in Fig. 9), it is partitioned into multiple blocks. The block dimension (192 in Fig. 9) is jointly chosen with $U_{\max}$ for higher device utilization (on Tesla C2075, the configuration offers eight concurrent active blocks on one multiprocessor and a theoretical occupancy of 100%). The synchronization required by the atomic addition does not cause much performance degradation. If the lexicon had many symbols, the possibility that multiple threads writing the same symbol would be low; if the lexicon had a very few symbols, the computation of this lexicon itself would be less likely to be on the critical path.

In the reduction stage, all the excitations $\{y(t), t \in S_l\}$ buffered in the shared memory are compared and the most likely symbol $t_{\max}$ is selected. The excitation values of this reference symbol and the observed symbol generate the key lexicon anomaly score, which is then collected to calculate the network anomaly score based on (3). The formal representation of the process is shown in Algorithm 2.

E. Implementation on Xeon Phi

In addition to the NVIDIA GPU, we investigate the performance of AnRAD on another emerging multicore architecture, the 64-core Intel Xeon Phi processor [25]. In this paper, we use the coprocessor in offload mode. The same memory layout, as in Fig. 7, is adopted. The main algorithm is also similar

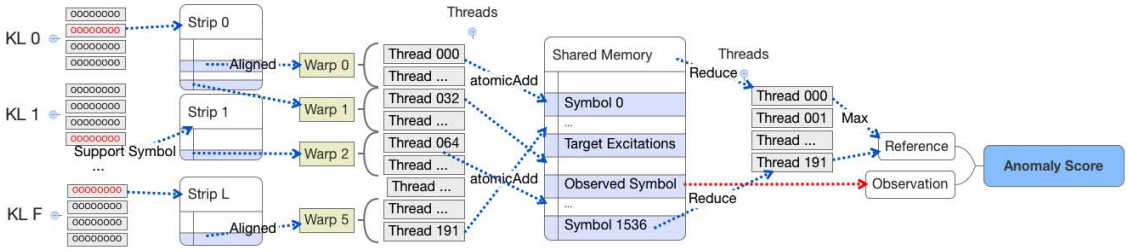


Fig. 9. Anomaly score computation.

Algorithm 2 Optimized Recall Kernel

```
procedure optimized_recall(KB, IN):        # KB: knowledge base; IN: input symbols
shared memory: exbuf[U_max]
block ← KB.blocks[blockIdx.y];
N ← block.num_symbols
for t = threadIdx.x : blockDim.x : N:
        exbuf[t] ← KB.prior[t]
sync threads
—       # Knowledge mapping
for each kl ∈ block.KLs:
        if IN[kl.input_idx] is empty:
            continue
        strip ← kl.strips[IN[kl.input_idx]]
        for t = threadIdx.x : blockDim.x : strip.len:
            e ← strip.entries[t]
            atomicAdd(exbuf[e.key], e.value+B)
sync threads
—       # Excitation reduction
obs ← exbuf[IN[block.input_idx]]
b ← threadIdx.x
for t = b+blockDim.x : blockDim.x : N:
        exbuf[b] ← max(exbuf[b], exbuf[t])
sync threads
ref ← threadReduceMax(exbuf[0:blockDim.x])
Score[blockIdx.x] = (ref − obs)/ref
```

as in Section VI-D. However, the workloads are mapped to different units of the coprocessor architecture. Typically, Xeon Phi KNC chip has less physical cores than GPU does, but each of the Xeon Phi's cores is a fully featured processor, and thus more powerful than the CUDA core in NVIDIA GPU. In particular, each Xeon Phi core is equipped with a 256-b vector engine, which can be effective in the mapping-reduction process. Therefore, the lexicon-wised CUDA block computations are mapped to individual OpenMP threads, and within each OpenMP thread, vector operation is used to realize the parallelism originally enabled by the multiple CUDA threads.

F. Evaluation on Single Data Streams

We implemented AnRAD in the aforementioned parallel architecture, including CPU multithreading, naive GPU acceleration, optimized GPU acceleration, and Xeon Phi offloading. We compare the four designs using test data from vehicle monitoring, DARPA, and ADFA data sets. For CPU multithreading, we use Intel Xeon W5580 with four cores (16 logic cores) running at 3.20-GHz clock frequency. For GPU-based implementations, we use NVIDIA Tesla C2075 with 448 cores running at 1.15-GHz clock frequency and 6-GB device memory. $U_{\max} = 1536$ and blockDim.x= 192 are selected to achieve full theoretical occupancy. The Intel coprocessor implementation is on a Xeon Phi 5100 with 60 cores running at 1.053 GHz and 16-GB memory capacity. A maximum of 240 threads are allocated.

TABLE VI
SINGLE STREAM PER-FRAME RUNTIMES

Implementation		DARPA	ADFA-LD	Vehicle
CPU 1-thread	time	200.5ms	123.4ms	78.6ms
	speedup	1	1	1
CPU 16-thread	time	25.7ms	23.4ms	12.2ms
	speedup	7.8	5.3	6.4
GPU naive	time	146.9ms	44.5ms	150.1ms
	speedup	1.4	2.8	0.52
GPU optimized	time	0.210ms	0.0742ms	0.178ms
	speedup	**955**	**1663**	**442**
Xeon Phi KNC	time	4.04ms	1.02ms	3.27ms
	speedup	50	121	24

Table VI compares the performance of those different implementations. As observed from the table, CPU with 16 threads achieves five to eight times of speedup compared with the serial implementation. It is not linearly scaled with thread number mainly due to the memory stalls caused by concurrent memory accesses. The naive implementation on GPU, however, only provides marginal improvement or even runs slower than the single-thread baseline, because imbalanced workloads across threads produce control divergences. The optimized GPU implementation provides substantial speedup over the baseline methods. Finally, the improvement by Xeon Phi is also significant, since the workflow follows the same principle of the GPU implementation. The reason that Xeon Phi implementation is not as fast as GPU is that the single testing stream does not generate enough workload for the maximal 240 threads. For example, the generated DARPA network has only 123 key lexicons, so can only utilize about half of the thread capacity. Thus, the GPU may offer better responsiveness on small and randomly arrived service requests, while the processing power of Xeon Phi would be better utilized by larger workload batches.

TABLE VII
POWER AND PERFORMANCE

Device		DARPA	ADFA-LD	Vehicle
K20 GPU	time	0.270ms	0.0622ms	0.231ms
	speedup	49.9	55.9	33.4
	active power	102.4W	98.87W	86.3W
	active energy	27.6mJ	6.1mJ	19.9mJ
Jetson TK1	time	13.48ms	3.477ms	7.709ms
	active power	2.5W	3.5W	2.6W
	active energy	33.7mJ	12.1mJ	20.0mJ

G. Power and Performance Tradeoff

Furthermore, we evaluated the portability and performance of the AnRAD framework on high-performance workstation (NVIDIA Tesla K20c GPU) and embedded system (NVIDIA Jetson TK1). The former has 2496 CUDA cores running at 706-MHz clock frequency and 5-GB device memory; while the later has 192 CUDA cores running at 852-MHz clock frequency and 1.75-GB device memory. The power consumption of K20 workstation and Jetson board is 194.7 W and 3.4 W, respectively, when they are in idle state. We measure active power as the difference between average executing power and average idle power. Similar to the earlier analysis, we compare performance using data from vehicle monitoring, DARPA, and ADFA.

Table VII compares the performance of the implementation on both devices. It shows that the K20 GPU achieves speedup of approximately 30 to 50 times compared with the Jetson GPU. However, the Jetson system on average consumes an active power of 2.87 W, while the K20 system's average active power is 95.86 W. Our result shows that the execution on Jetson is effectively energy neutral as compared with K20 despite the longer runtime per frame. However, it gives significant efficiency in active power consumption in all cases. Despite its low power, Jetson still exceeds the real-time requirements—for example, it achieves a 13.5 ms per frame processing rate on DARPA stream whose input rate is only 300 ms per frame.

H. Multistream Extension on Wide-Area Monitoring

The previously discussed parallel implementation assumes that there is only one active knowledge base and one set of lexicons. In this section, we use wide-area vehicle behavior monitoring as a case study to demonstrate how the parallel implementation extends to concurrent streams with multiple contexts.

After the universal network structure is built, separate knowledge bases are trained for each zone using their local traffic streams. A context selection module is used to associate the input vehicles with their corresponding zone knowledge base. Then, the workloads are assigned to the computing platforms for anomaly scoring. The vehicles' key lexicons are mapped to OpenMP (CPU or Xeon Phi) threads or CUDA blocks (GPU). Using the GPU implementation as an example shown in Fig. 10, the concurrent streams (i.e., vehicles appeared in the same time frame) are mapped into a 2-D

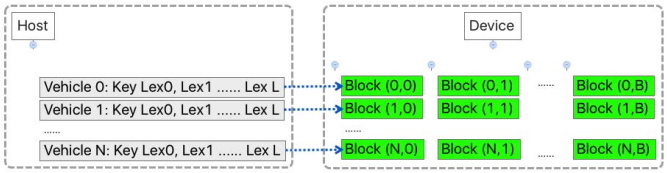


Fig. 10. 2-D workload mapping.

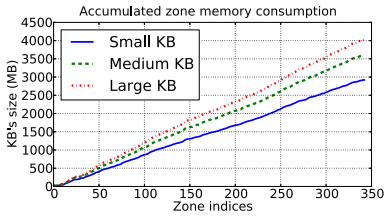
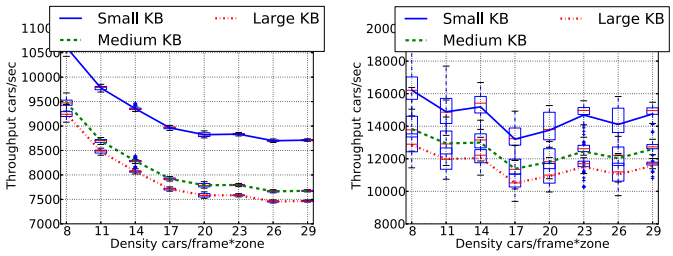


Fig. 11. Memory consumption of multiple zones.



(a) (b)

Fig. 12. Vehicle detection throughputs. (a) Tesla C2075. (b) Xeon Phi 5100.

CUDA grid. Distinct vehicles are assigned across the first dimension of the grid (gridDim.x), and their key lexicons are mapped to the second dimension (gridDim.y). This design allows the pipeline to handle input streams with varying volumes. Finally, the results from the devices are collected to generate vehicle status reports.

To evaluate the performance of AnRAD in multi-stream multimodel scenarios, different training set volumes (small—80 min, medium—160 min, and large—240 min) are evaluated in order to test the system performance under different levels of knowledge.

Each zone creates a new knowledge base, and the number of zone partitions decides the memory usage. Fig. 11 shows the device memory consumptions for the knowledge bases. The X-axis gives the number of the detection zones, and the Y-axis gives the accumulated knowledge base size in megabytes. As the number of zones increases, the memory consumption increases nearly linearly, which indicates a balanced distribution of the knowledge. The maximum storage requirement for all 342 zones ranges from around 3 to 4 GB, which is much lower than the available device memory of C2075 (6 GB) or Xeon Phi (8 GB). Therefore, a single device can support all 342 zones.

Vehicles are represented as concurrent data stream inputs to the monitoring system. We define the throughput as the number of vehicles that AnRAD can check (for abnormal behavior) in each second; 100 zones are randomly picked and have their throughput collected with different vehicle densities. Fig. 12 shows the detection throughputs of AnRAD on Tesla C2075 and Xeon Phi. The X-axis shows the density,

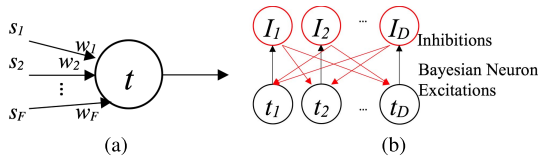


Fig. 13. Spiking neuron model. (a) Bayesian neuron. (b) Lexicon circuit.

i.e., the average number of vehicles that appear simultaneously in a zone. The Y-axis is the overall throughput of the 100 zones. On both devices, the throughputs drop as the vehicle density increases, because a denser neighboring among vehicles induces more processing of the interactive features. The throughputs get stable when the interaction-related key lexicons are saturated. GPU generally has smaller variations in different traffic patterns as in the box plots Fig. 12. Also, larger knowledge bases cause lower throughputs, but the degradation is smaller with knowledge bases approaching convergence. The throughputs of Xeon Phi 5100 on Fig. 12(b) are roughly twice as high as those in Fig. 12(a) of Tesla C2075. This generally agrees to the specifications that Xeon Phi 5100 peaks at 1011Gflop [25] and Tesla C2075 peaks at 515Gflop [37] of double precision performances.

VII. Neuromorphic Implementation of AnRAD

In order to achieve even better power and cost efficiency, AnRAD is also extended to SNN. SNN is capable of modeling sequence data [46] and providing real-time inferences [12]. It can be implemented on the emerging nonconventional neuromorphic hardware [31] for performance and energy efficiency. The structure of the AnRAD network can be directly mapped to an SNN, where neurons representing symbols and synapses representing knowledge links. The likelihood calculation can be carried out by the integrate-and-fire function and we use the neuron's firing rate to proportionally represent the likelihood of the input. Furthermore, using the STDP rule [34], it can be proved that the synaptic strength will asymptotically approach to the probability of the postsynaptic neuron firing given the condition of the presynaptic neuron status. In this paper, we map the AnRAD detection network to a stochastic SNN [3] to demonstrate that it can be implemented using this emerging neuromorphic architectures.

We use the Bayesian neuron model [34] to build the inference network. Fig. 13(a) shows a neuron t, whose membrane potential $u(\tau)$ is computed as

$$u(t,\tau) = w_0 + \sum_{i=0}^{F} w_i s_i(\tau) \tag{12}$$

where w_i is the weight of the synapse connecting t to its ith presynaptic neuron s_i. $s_i(\tau)$ is 1 if neuron s_i issues a spike at time τ and w_0 models the intrinsic excitability of the neuron t. The firing probability of t depends exponentially on the membrane potential, $p(t \text{ fires at time } \tau) \propto e^{u(t,\tau)}$. For the anomaly detection network, every symbol of a lexicon is mapped to one of the Bayesian neurons, and receives fan-in synapses from the other lexicons. The weights of the synapse are the learned $p(s|t)$ from the AnRAD knowledge

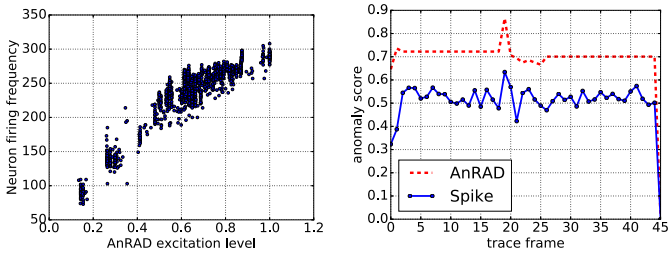


Fig. 14. Vehicle trace activities. (a) Correlation of $y(t)$ and spikes(t). (b) Trace of anomaly scores.

bases. Among a lexicon, the Bayesian neurons also have 1-on-1 inhibitors that competitively suppress the other symbols as in Fig. 13(b).

We run the SNN simulation through 46 frames of the vehicle traces, and collect 1242 key symbol activities in lexicon ⟨speed, neighbor.distance⟩. Fig. 14(a) shows the scatter plot between the AnRAD excitation levels (X-axis) of symbols in key lexicons and the spike frequencies (Y-axis) of their corresponding SNN neurons. The frequency is sampled within a window of 5000 ticks [i.e., $\tau \leq 5000$] at the beginning of each frame. Strong correlation is observed and the correlation coefficient is 0.925, which confirms that the spiking rate of a neuron can be used to represent the likelihood of its corresponding symbol. Although excitations are represented by spike counts over a time span, a rate-coding window-based pipeline is implemented for event-driven processing. In Fig. 14(b), we present the normalized spiking rate difference between the anomaly symbol and the reference symbol, i.e., $[\text{spikes}(t_{\max}) - \text{spikes}(t)]/\text{spikes}(t_{\max})$. The X-axis indicates the frames of the vehicle trace, and the Y-axis is the computed anomaly scores. Compared with the AnRAD anomaly score calculated by (2), the spike rate clearly reflects the trend of the anomaly activity. It has a scale shift, which indicates the need to adjust the decision threshold when implementing the AnRAD network on SNN architectures.

VIII. Conclusion

We have presented an HPC-based neuromorphic anomaly detection framework that processes concurrent data streams in real time. The confabulation theory is adopted and incorporated as basic testing unit in a hierarchical cognitive architecture. The framework uses data-driven approach to configure the network structure for different applications. Furthermore, the method is accelerated on GPUs and Xeon Phi processors with fine-grained and coarse-grained parallelization. We also implement the network on an SNN simulator to show that the framework can be extended to emerging neuromorphic hardware.

Acknowledgment

Received and approved for public release by AFRL on 03/06/2015, case number 88ABW-2015-0875. Any Opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of AFRL or its contractors.

REFERENCES

[1] B. Ahmed *et al.*, "Hierarchical conditional random fields for outlier detection: An application to detecting epileptogenic cortical malformations," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2014, pp. 1080–1088.

[2] K. Ahmed, Q. Qiu, P. Malani, and M. Tamhankar, "Accelerating pattern matching in neuromorphic text recognition system using Intel Xeon Phi coprocessor," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2014, pp. 4272–4279.

[3] K. Ahmed, A. Shrestha, and Q. Qiu, "Simulation of Bayesian learning and inference on distributed stochastic spiking neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2016, pp. 1044–1051.

[4] E. Begoli, "A short survey on the state of the art in architectures and platforms for large scale data analysis and knowledge discovery from data," in *Proc. WICSA/ECSA Companion Vol.*, 2012, pp. 177–183.

[5] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 303–336, 1st Quart., 2014.

[6] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, vol. 29. 2000, pp. 93–104.

[7] J. B. Cabrera, C. Gutiérrez, and R. K. Mehra, "Ensemble methods for anomaly detection and distributed intrusion detection in mobile ad-hoc networks," *Inf. Fusion*, vol. 9, no. 1, pp. 96–119, 2008.

[8] Q. Cai, H. He, and H. Man, "Spatial outlier detection based on iterative self-organizing learning model," *Neurocomputing*, vol. 117, pp. 161–172, Oct. 2013.

[9] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Comput. Commun.*, vol. 35, no. 7, pp. 772–783, 2012.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 5, pp. 823–839, May 2012.

[12] Q. Chen and Q. Qiu, "Real-time anomaly detection for streaming data using burst code on a neurosynaptic processor," in *Proc. Conf. Design, Autom. Test Eur. (DATE)*, 2017, pp. 1–6.

[13] Q. Chen, Q. Qiu, H. Li, and Q. Wu, "A neuromorphic architecture for anomaly detection in autonomous large-area traffic monitoring," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2013, pp. 202–205.

[14] Q. Chen, Q. Qiu, Q. Wu, M. Bishop, and M. Barnell, "A confabulation model for abnormal vehicle events detection in wide-area traffic monitoring," in *Proc. Int. Inter-Discipl. Conf. Cognit. Methods Situation Awareness Decision Support (CogSIMA)*, 2014, pp. 216–222.

[15] Q. Chen, Q. Wu, M. Bishop, R. Linderman, and Q. Qiu, "Self-structured confabulation network for fast anomaly detection and reasoning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2015, pp. 1–8.

[16] G. Creech and J. Hu, "Generation of a new IDS test dataset: Time to retire the KDD collection," in *Proc. Wireless Commun. Netw. Conf. (WCNC)*, 2013, pp. 4487–4492.

[17] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 807–819, Apr. 2013.

[18] S. K. Esser *et al.*, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2013, pp. 1–10.

[19] J. Hawkins, S. Ahmad, and D. Dubinsky, "Hierarchical temporal memory including htm cortical learning algorithms," Numenta, Inc., Palo Alto, CA, USA, Tech. Rep., 2011. [Online]. Available: https://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf

[20] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier detection using replicator neural networks," in *Data Warehousing and Knowledge Discovery*. Berlin, Germany: Springer, 2002, pp. 170–180.

[21] R. Hecht-Nielsen, *Confabulation Theory: The Mechanism of Thought*. Berlin, Germany: Springer, 2007.

[22] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *J. Comput. Secur.*, vol. 6, no. 3, pp. 151–180, 1998.

[23] J. Hu, H. Tang, K. C. Tan, and H. Li, "How the brain formulates memory: A spatio-temporal model research frontier," *Comput. Intell. Mag.*, vol. 11, no. 2, pp. 56–68, 2016.

[24] H. Huang, "Rank based anomaly detection algorithms," Ph.D. dissertation, Syracuse Univ., Syracuse, NY, USA, 2013.

[25] *Intel Xeon Phi Product Family: Product Brief*, Intel, Santa Clara, CA, USA, 2013.

[26] D. Ippoliti and X. Zhou, "A-GHSOM: An adaptive growing hierarchical self organizing map for network anomaly detection," *J. Parallel Distrib. Comput.*, vol. 72, no. 12, pp. 1576–1590, 2012.

[27] I. Kang, M. K. Jeong, and D. Kong, "A differentiated one-class classification method with applications to intrusion detection," *Expert Syst. Appl.*, vol. 39, no. 4, pp. 3899–3905, 2012.

[28] R. Laxhammar and G. Falkman, "Online learning and sequential anomaly detection in trajectories," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1158–1173, Jun. 2014.

[29] M. D. Linderman, R. Bruggner, V. Athalye, T. H. Meng, N. Asadi, and G. P. Nolan, "High-throughput Bayesian network learning using heterogeneous multicore computers," in *Proc. Int. Conf. Supercomput.*, 2010, pp. 95–104.

[30] R. P. Lippmann *et al.*, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," in *Proc. DARPA Inf. Survivability Conf. Expo.*, vol. 2. 2000, pp. 12–26.

[31] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[32] P. Mitra, C. A. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 3, pp. 301–312, Mar. 2002.

[33] R. Motwani and P. Raghavan, *Randomized Algorithms*. London, U.K.: Chapman & Hall, 2010.

[34] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," *PLoS Comput. Biol.*, vol. 9, no. 4, p. e1003037, 2013.

[35] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 14. 2002, p. 841.

[36] *The Science of Anomaly Detection*, Numenta, Redwood City, CA, USA, 2014.

[37] *NVIDIA Tesla c2075 Companion Processor*, NVIDIA, Santa Clara, CA, USA, 2011.

[38] E. J. Palomo, J. M. Ortiz-de-Lazcano-Lobato, E. Domínguez, and R. M. Luque, "An anomaly detection system using a GHSOM-1," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2010, pp. 1–7.

[39] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *Proc. Symp. Comput. Intell. Data Mining (CIDM)*, 2007, pp. 504–515.

[40] Q. Qiu, Q. Wu, M. Bishop, R. E. Pino, and R. W. Linderman, "A parallel neuromorphic text recognition system and its implementation on a heterogeneous high-performance computing cluster," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 886–899, May 2013.

[41] G. J. Rinkus, "A cortical sparse distributed coding model linking mini- and macrocolumn-scale functionality," *Frontiers Neuroanatomy*, vol. 4, Jun. 2010, Art. no. 17.

[42] M. L. Shahreza, D. Moazzami, B. Moshiri, and M. Delavar, "Anomaly detection using a self-organizing map and particle swarm optimization," *Sci. Iranica*, vol. 18, no. 6, pp. 1460–1468, 2011.

[43] F. Simmross-Wattenberg, J. I. Asensio-Perez, P. Casaseca-de-la-Higuera, M. Martin-Fernandez, I. A. Dimitriadis, and C. Alberola-Lopez, "Anomaly detection in network traffic based on statistical inference and alpha-stable modeling," *IEEE Trans. Depend. Sec. Comput.*, vol. 8, no. 4, pp. 494–509, 2011.

[44] J. Tian, M. H. Azarian, and M. Pecht, "Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm," in *Proc. Eur. Conf. Prognostics Health Manage. Soc.*, 2014, p. 9.

[45] J. Yin, D. H. Hu, and Q. Yang, "Spatio-temporal event detection using dynamic conditional random fields," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, vol. 9. 2009, pp. 1321–1327.

[46] Q. Yu, R. Yan, H. Tang, K. C. Tan, and H. Li, "A spiking neural network system for robust sequence recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 621–635, Mar. 2016.

[47] A. Zimek, M. Gaudet, R. J. Campello, and J. Sander, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *Proc. Int. Conf. Knowl. Discovery Data Mining (SIGKDD)*, 2013, pp. 428–436.

[48] *UCI Machine Learning Repository*, accessed on Aug. 2014. [Online]. Available: http://kdd.ics.uci.edu/databases/kdd cup99/kddcup99.html

[49] *NVIDIA Cuda Parallel Computing Platform and Programming Model*, accessed on Oct. 2014. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

Qiuwen Chen received the B.S. and M.S. degrees in electrical engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2009 and 2012 respectively, and the Ph.D. degree from the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA, in 2016.

He is currently a Software Engineer with Microsoft, Redmond, WA, USA. His current research interests include machine learning, neural networks, neuromorphic computing, and high performance computing.

Ryan Luley received the B.S. degree from the Mathematics Department, St. Lawrence University, Canton, NY, USA, in 2001, and the M.S. degree from the Electrical Engineering and Computer Science Department, Syracuse University, Syracuse, NY, USA, in 2008, where he is currently pursuing the Ph.D. degree with the Electrical Engineering and Computer Science Department.

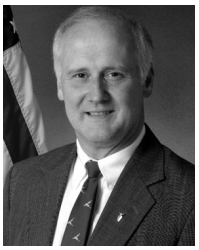
He was a Software Engineer for Lockheed Martin Management and Data Systems, Valley Forge, PA, USA. He is currently a Mathematician with the United States Air Force Research Laboratory, Information Directorate, Rome, NY, USA. His current research interests include general purpose graphics processing unit computing, high performance embedded computing, and parallel discrete event simulation.

Qing Wu (M'96) received the B.S. and M.S. degrees from the Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China, in 1993 and 1995, respectively, and the Ph.D. degree from the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA, in 2002.

He was an Assistant Professor with the Department of Electrical and Computer Engineering, State University of New York at Binghamton, Binghamton, NY, USA. He is currently a Principal Electronics Engineer with the United States Air Force Research Laboratory, Information Directorate, Rome, NY, USA. He has authored or co-authored over ninety research papers in international journals and conferences. His current research interests include neuromorphic computing architectures, high-performance computing architectures, deep neural networks, and memristor-based neuromorphic circuits and systems.

Morgan Bishop (M'09) received the B.A. degree from the Department of Computer Science, State University of New York at Geneseo, Geneseo, NY, USA, in 2004.

He was the Lead Developer for Jeansee Corporation, Merriam, KS, USA, where he investigated DNA binding algorithms to achieve optimal DNA codes for use in parallel computing architectures. He is currently a Computer Scientist with the United States Air Force Research Laboratory, Information Directorate, Rome, NY, USA. He has authored or co-authored over fifteen research papers in journals and conferences throughout the world. His current research interests include scalable algorithm development for heterogeneous high performance computers, basic research in next-generation massively parallel systems, and the development of brain-inspired intelligence models for real-world application.



Richard W. Linderman (F'01) received the B.S., M.S., and Ph.D. degrees from the Department of Electrical Engineering, Cornell University, Ithaca, NY, USA, in 1980, 1981, and 1984, respectively.

He was commissioned as a Second Lieutenant in 1980. Upon completing four years of graduate studies, he entered active-duty, teaching computer architecture courses and leading related research with the Air Force Institute of Technology. He was assigned to Rome Air Development Center, Rome, NY, USA, in 1988, where he led surveillance signal processing architecture activities. In 1991, he transitioned from active-duty to civil service as a Senior Electronics Engineer with Rome Laboratory, Rome, becoming a Principal Engineer in 1997. During these years, he pioneered three-dimensional packaging of embedded architectures and led the Department of Defense community exploring signal and image processing applications of high performance computers. He is currently the Deputy Director of Information Systems and Cyber Security with the Office of the Under Secretary of Defense (Acquisition, Technology and Logistics). He was a member of the Scientific and Professional Cadre of Senior Executives, and also the Chief Scientist with the Air Force Research Laboratory, Information Directorate, Rome. The Information Directorate leads the discovery, development, and integration of affordable warfighting information technologies for air, space, and cyberspace forces. He serves as the Directorate's Principal Scientific and Technical Adviser and primary authority for the technical content of the science and technology portfolio. He provides principal technical oversight of a broad spectrum of information technologies, including fusion and exploitation; command and control; advanced architectures; information management; communications and networking; defensive information warfare; and intelligent information systems technologies. He has authored or co-authored over 70 journal, conference and technical papers. He holds six U.S. patents.

Qinru Qiu (M'00) received the B.S. degree from the Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China, in 1994, and the M.S. and Ph.D. degrees from the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA, in 1998 and 2001, respectively.

She has been an Assistant Professor and then an Associate Professor with the Department of Electrical and Computer Engineering, State University of New York at Binghamton, Binghamton, NY, USA. She is currently a Professor and the Program Director of Computer Engineering with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA. Her current research interests include neuromorphic computing and high performance energy efficient computing systems.

Dr. Qiu is the TPC Member of Design, Automation and Test in Europe, Design Automation Conference, International Symposium on Low Power Electronics and Design, International Symposium on Quality Electronic Design, International Conference on Very Large Scale Integration, and International Conference On Computer Aided Design. She served as the Associate Editor of the *ACM Transactions on Design Automation of Electronic Systems*.