An Efficient One-Class SVM for Novelty Detection in IoT

Kun Yang *
Columbia University
ky2440@columbia.edu

Samory Kpotufe
Columbia University
skk2175@columbia.edu

Nick Feamster
University of Chicago
feamster@uchicago.edu

Abstract

One-Class Support Vector Machines (OCSVMs) are a set of common approaches for novelty detection due to their flexibility in fitting complex nonlinear boundaries between normal and novel data. However, conventional OCSVMs can introduce prohibitive memory and computational overhead in detection. This work designs, implements, and evaluates an efficient OCSVM for such practical settings. We extend Nyström and (Gaussian) Sketching approaches to OCSVM, combining these methods with clustering and Gaussian mixture models to achieve 15-30x speedup in prediction time and 30-40x reduction in memory requirements without sacrificing detection accuracy.

1 Introduction

As devices ranging from consumer electronics to building control systems become connected to the Internet as part of the "Internet of Things" (IoT), both these devices and the network itself are subject to new threats. Moreover, many IoT deployments require *fast* novelty detection in field deployments (e.g., embedded devices such as home network routers or embedded sensors), where both computational and memory requirements may be limited. In operational deployments, there may be the need to quickly detect an attack, a rogue device, or other malfunction.

Novelty detection, which aims to detect unusual activity based on the observable properties of network traffic, is a common defense. One-Class Support Vector Machines (OCSVM) are one of the state-of-the-art approaches for novelty detection[2], due to their ability to identify a wide range of nonlinear classification boundaries. Such flexibility is appropriate for Internet of Things (IoT) devices and applications, which exhibit complexity due to the vast heterogeneity of devices and the wide range of traffic patterns under different operating modalities. However, OCSVMs can be computationally expensive at *detection time*. Given a new observation x to classify as normal or novel, detection consists of evaluating a scoring function $f(x)$—of the form $\sum_{i=1}^{n} \alpha_i K(X_i, x)$, defined with respect to training data $\{X_i\}$ of size n and a so-called *kernel function* K; such evaluation of $f(x)$ takes time and space $\Omega(n)$ for typically large training data size n in the thousands.

To address this issue, we extend Nyström and (Gaussian) Sketching approaches to OCSVM [16, 13, 8], combining these methods with clustering and Gaussian mixture models to speed up detection time and reduce memory requirements of OCSVM, while preserving detection performance. It should be noted that in this work, our focus is on *detection* time and space, rather than *training* time and

*The work was done at Columbia University.

[2]In the context of security, novelty detection is often referred to as *anomaly detection*; we use the term *novelty detection* to refer to the same class of algorithms, as the problem is equivalent. We prefer the use of novelty detection in this paper because the classes of events that we aim to detect include conventional anomalies (in the security sense), as well as a broader class of novel events, activities, and devices that might be simply "new", though these new events may not necessarily have a negative connotation.

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Workshop on ML for Systems Workshop.

space. For simplicity we will henceforth refer to these approaches respectively as OC-Nyström and OC-KJL, where *OC* stands for *One Class* (as in OCSVM) to emphasize the unsupervised nature of these methods. We evaluate OC-Nyström and OC-KJL, both with and without automatic GMM parameter selection, on multiple IoT datasets encoding a variety of detection use-cases of interest, from detecting benign new devices to malicious activity from infected devices. The experimental results show that our methods have:

- *Significant reduction in detection time and space.* We observe typical detection time speedups (w.r.t. the baseline OCSVM) between 14 to 20 times faster using either of OC-Nyström or OC-KJL, and 40+ times for some datasets. Typical space complexities decrease by a factors of 20 or more w.r.t. OCSVM.
- *Equivalent or improved detection performance.* Upon proper calibration of all three procedures, both OC-Nyström or OC-KJL achieve detection performance on par with the baseline OCSVM as measured by area under the curve (AUC). Both slightly outperform OCSVM in some cases, which is likely due to the fact that the new mapping ϕ' acts as a lower-dimensional projection which at times recovers intrinsic structure not present in abnormal traffic. Under rule-of-thumb choices of the main hyperparameter shared by all three procedures, i.e., a so-called *kernel bandwidth* parameter, OC-Nyström and OC-KJL (with automatic choices of number of GMM components k) attain at least 0.85% of OCSVM's AUC on most datasets, and manages improvements in AUC over that of OCSVM on many datasets.

2 Basic Background

2.1 (Gaussian Kernel) OCSVM

OCSVM first maps data $x \in \mathbb{R}^D$ as $\phi(x)$ into an infinite dimensional space $\mathcal{H}$ (a so called *reproducing kernel Hilbert space* or RKHS) and in the space, there exists a hyperplane that isolates normal data from future anomalous observations. Such a hyperplane can be estimated without actually computing $\phi(X_i) \in \mathcal{H}$, simply through geometrical operations encoded by all pairwise inner-products $\langle \phi(X_i), \phi(X_j) \rangle$ given by a *kernel function* $K(X_i, X_j)$. These inner-products are encoded for convenience in a so-called *gram matrix* $\mathcal{K} \in \mathbb{R}^{n \times n}$, $\mathcal{K}_{i,j} = K(X_i, X_j)$ so the training phase just operates on $\mathcal{K}$ to return an implicit representation of the separating hyperplane in the form of coefficients $\{\alpha_i\}_{i=1}^n$ and a threshold α_0 used as follows: A future test point $x \in \mathbb{R}^D$, is deemed anomalous if it maps as $\phi(x)$ to *the wrong side* of the hyperplane, that is, if $f(x) \doteq \sum_{i=1}^n \alpha_i K(X_i, x) < \alpha_0$. It should be clear by now that computational complexity is determined by the number $\tilde{n} \leq n$ of nonzero α_i's.

2.2 Nyström and KJL Sketching

A main approach adopted recently to speedup training time, e.g., in the context of SVMs, is to reduce operations on the gram matrix $\mathcal{K} \in \mathbb{R}^{n \times n}$ by approximating it with a rank $d \ll n$ matrix $\mathcal{K}' \in \mathbb{R}^{n \times n}$ that might induce faster operations, while preserving much of the geometry induced by the kernel K on the implicit mapping $\{\phi(X_i)\}_{i=1}^n \in \mathcal{H}$. These come in different forms under the name of Nyström and Sketching. In particular, in the case of SVM, the Nyström or Sketching matrix $\mathcal{K}'$ manages to preserve the same simple linear relationships between classes of datapoints encoded in $\mathcal{K}$. In other words, one can simply proceed as usual with $\mathcal{K}'$ in place of $\mathcal{K}$ and train a linear classifier. Unfortunately, in the unsupervised case of OCSVM, we lose the ability to learn such simple linear relation (which can be in seen in Appendix F) between classes under Nyström or Sketching $\mathcal{K}'$—an issue particularly true in IoT—requiring us to develop a different approach on top of Nyström or Sketching.

3 Proposed Efficient Detection Methods

To address this issue of nonlinearity in the case of OCSVM after applying Nyström or Sketching, we rely on recent interpretations of these speedup approaches [16, 13, 8] which view these transformations as a data mapping $x \mapsto \phi(x) \mapsto \phi'(x)$ that preserves distances between original transformed points $\phi(X_i), \phi(X_j)$, even if linearity between classes is not preserved. As inter-point distances

are preserved, one might then expect that *cluster* structures are preserved, i.e., dense groups of points under ϕ remain clumped together under ϕ'. Building on this intuition, detection will therefore just consist of flagging any future query point x as *abnormal* if $\phi'(x)$ falls far from clusters in the remapped training data $\{\phi'(X_i)\}_{i=1}^n$. In this section, we first introduce our improved/approximated embedding ϕ's, Then, we describe our proposed detection methods in detail.

3.1 Approximated embedding ϕ'

let S_m denote a random subsample of size $m \ll n$ of the training data $S_n \doteq \{X_i\}_{i=1}^n$ (w.l.o.g., we can let $S_m \doteq \{X_i\}_{i=1}^m$). Furthermore, for any subset of indices $I, J \subset \{1, \ldots, n\}$, let $\mathcal{K}_{I,J}$ denote the submatrix of $\mathcal{K}$ corresponding to rows in I, and columns in J. Then, for $I = \{1 : m\}$ and $J = \{1 : n\}$, we will consider the submatrices, $\mathcal{K}_{I,I} \in \mathbb{R}^{m \times m}$ – i.e., the gram matrix on S_m, and $\mathcal{K}_{I,J} \in \mathbb{R}^{m \times n}$, the gram submatrix of inner-products between S_m and S_n.

- *Nyström ϕ'.* Let $K_{I,I}^{-1}$ denote a rank d pseudo-inverse of $K_{I,I}$; then setting $\mathcal{K}' = K_{I,J}^\top \cdot K_{I,I}^{-1} \cdot K_{I,J}$, the problem is to come up with $\phi' \in \mathbb{R}^d$ such that $\langle \phi'(X_i), \phi'(X_j) \rangle$ is exactly $\mathcal{K}'_{i,j}$. Let $\Lambda \in \mathbb{R}^{d \times d}$ denote the diagonal matrix containing the top d eigenvalues $\lambda_1, \ldots, \lambda_d$ of $\mathcal{K}_{I,I}$, and $V = [v_1, \ldots, v_d] \in \mathbb{R}^{m \times d}$ contains the corresponding (column) eigenvectors v_i's. Now, for any $x \in \mathbb{R}^D$, let $K(x)$ denote $[K(x, X_1), \ldots, K(x, X_m)]^\top$, we then have

$$\phi'(x) \doteq P \cdot K(x), \text{ where we let } P \doteq \Lambda^{-1/2} \cdot V^\top. \tag{1}$$

  We can verify that setting $K_{I,I}^{-1} = V \cdot \Lambda^{-1} \cdot V^\top$, indeed recovers $\mathcal{K}'$ as defined above.

- *KJL Sketching ϕ'.* In general, Sketching consists of multiplying $\mathcal{K}$ (or $K_{I,I}$) by a matrix Z with random entries. It was recently shown [8] that when Z has i.i.d. $\mathcal{N}(0, 1)$ Gaussian entries, sketching can be understood as a random projection operation in $\mathcal{H}$, leading to the following mapping $\phi' \in \mathbb{R}^d$. For any $x \in \mathbb{R}^D$, let $K(x)$ again denote the vector $[K(x, X_1), \ldots, K(x, X_m)]^\top$, and let $Z \in \mathbb{R}^{d \times m}$ with random $\mathcal{N}(0, 1)$ entries. We then have:

$$\phi'(x) \doteq P \cdot K(x) \text{ where we let } P \doteq Z \cdot K_{I,I}. \tag{2}$$

Notice that in both cases of Nyström and KJL, we only have to retain $P \in \mathbb{R}^{d \times m}$ at testing time, along with the m datapoints in S_m. Namely, the model ϕ' requires space complexity exactly $m \cdot (d + D)$. Similarly the time complexity for evaluating $\phi'(x)$ is of order $m \cdot (d + D)$, so does not depend on n.

3.2 Proposed Efficient Detection Procedures

Once the data is mapped to $\mathbb{R}^d$ as $\{\phi'(X_i)\}_{i=1}^n$ through Nyström or KJL, our next step is to model clusters in $\{\phi'(X_i)\}_{i=1}^n$ as components of a Gaussian Mixture Model (GMM), which has the benefit of allowing for a simple detection rule based on *density levels*. Finally, as the GMM model introduces a new hyperparameter on top of vanilla OCSVM, namely the number k of Gaussian components (or number of clusters), we further propose a basic approach to automatically set such a parameter k by estimating high density regions of $\{\phi'(X_i)\}_{i=1}^n$ via existing methods such as ***QuickShift++*** [6]. Once f is learned, detection consists of flagging x as a novelty if $f(\phi'(x))$ is small than a threshold t.

Meta Procedures. Given a Gaussian kernel K with bandwidth h, embedding choices $m, d \ll$ *training size* n, the final OC-Nyström and OC-KJL approaches are summarized below:

Training: Given normal data $\{X_i\}_{i=1}^n \in \mathbb{R}^D$ do:
- Embed X_i's as $\phi'(X_i) \in \mathbb{R}^d$ via Nyström equation 1 or KJL equation 2;
- Parameter k is passed in, or is chosen via Quickshift++ (see Appendix A) on embedded data $\{\phi'(X_i)\}_{i=1}^n \in \mathbb{R}^d$;
- Estimate a GMM f with k components on $\{\phi'(X_i)\}_{i=1}^n$;
- **Return** GMM f, along with projection ϕ' (i.e., matrix P and subsample S_m) $\square$;

Detection: Given new $x \in \mathbb{R}^D$, and model (ϕ', f), do:
- Embed x as $\phi'(x)$ into $\mathbb{R}^d$;
- Flag x as novelty iff $f(\phi'(x)) \leq$ *threshold* t $\square$

4 Experiments

4.1 Experimental setup

In this work, we evaluate our methods on three devices (one server and two IoT devices (Nvidia Nano (NANO) and Raspberry Pi (RSPI)) in Appendix B) and seven datasets (in Appendix C) with various performance metrics (in Appendix D), which includes detection AUC (the Area Under the Curve), training and testing time, and testing space. Moreover, to reduce uncertainty in reported results, we repeat each experiment in 5 times and report averages and standard deviations on performance metrics. When reporting *speedups* for OC-Nyström and OC-KJL over OCSVM, we use the corresponding average performance of OCSVM, say μ. In other words, if we observe AUCs $a_1, \ldots, a_5$ for OC-KJL, we report the mean of $a_1/\mu, \ldots a_5/\mu \pm$ the std of these ratios. We proceed similarly for time ratios.

4.2 Result Analysis

In the main body, we present the results obtained by different methods when some validation is available to tune model hyperparmeters as described in Appendix A. Results for the case of no tuning reveal similar speedups in time and space as shown here, and are given in detail in Appendix J. We further focus here on the OC-Nyström-QS and OC-KJL-QS—for fair comparison as OCSVM comes with a single parameter h—while results for OC-Nyström and OC-KJL (which require h, k to be tuned) are given in Appendix I.

OCSVM Baseline Performance. Table 1 shows OCSVM baseline performance. All training is performed on the server; testing is performed on all platforms. The table reports (1) AUC, same for all machines, since the same models and test data are used; (2) training time on the server; (3) test time for all three machines; and (4) test space, which is the same across all machines.

Table 1: *OCSVM baseline performance. Time is in milliseconds per 100 datapoints and space is in kiloBytes.*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
AUC		0.67 ± 0.01	0.66 ± 0.02	0.99 ± 0.00	0.85 ± 0.00	0.93 ± 0.00	0.90 ± 0.00	0.93 ± 0.00
Server Train Time (ms)		93.93 ± 4.75	77.87 ± 1.72	82.59 ± 1.53	84.19 ± 1.93	80.83 ± 7.72	111.26 ± 6.59	110.53 ± 5.77
Test Time (ms)	RSPI	125.86 ± 0.08	118.19 ± 4.54	74.53 ± 0.05	124.58 ± 0.22	120.12 ± 0.22	125.43 ± 0.22	124.54 ± 0.28
	NANO	83.87 ± 0.18	86.71 ± 0.82	58.57 ± 0.05	86.89 ± 0.80	69.62 ± 0.09	88.80 ± 0.08	84.07 ± 0.04
	Server	19.84 ± 0.16	19.97 ± 1.36	11.22 ± 0.14	19.99 ± 0.23	16.15 ± 0.20	19.98 ± 0.03	19.81 ± 0.01
Space (kB)		1763.53 ± 0.45	961.52 ± 0.75	2792.46 ± 0.48	1042.92 ± 0.45	641.58 ± 0.10	1441.75 ± 0.14	1202.39 ± 0.30

Table 2: *Retained AUC (method/OCSVM) and server train time speedup (OCSVM time/method time).*

Method \ Dataset	UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
OC-KJL-QS: AUC Retained	1.33 ± 0.02	1.05 ± 0.03	0.95 ± 0.03	1.03 ± 0.01	0.97 ± 0.02	1.07 ± 0.00	0.99 ± 0.00
Train Speedup	1.36 ± 0.07	1.10 ± 0.02	1.65 ± 0.03	1.14 ± 0.03	0.97 ± 0.09	1.18 ± 0.07	1.19 ± 0.06
OC-Nyström-QS: AUC Retained	1.37 ± 0.02	1.13 ± 0.04	0.90 ± 0.03	1.06 ± 0.02	0.96 ± 0.01	1.09 ± 0.01	0.98 ± 0.01
Train Speedup	1.32 ± 0.07	1.12 ± 0.02	1.74 ± 0.03	1.16 ± 0.03	0.93 ± 0.09	1.26 ± 0.07	1.17 ± 0.06

AUC Retained. Table 2 show that our proposed methods manage to retain the accuracy of the baseline OCSVM, and also do not sacrifice training efficiency. Our detection methods largely retain the detection performance of OCSVM, often within a ratio of 1 or more, except in the case of MAWI, SFRFIG, and DWSHR – which are still high AUCs considering OCSVM's very good performance on these datasets. Moreover, for some datasets such as UNB and CTU, all procedures manage to actually outperform OCSVM. It is likely that such higher performance is due to the additional regularization inherent in the dimension reduction performed by our methods. Here, we also re-emphasize that all methods presented are tuned fairly against the baseline OCSVM, over the same bandwidth parameter h, as detailed in Section A.Versions with Quickshift++, namely OC-Nyström-QS and OC-KJL-QS, tend to achieve slightly smaller AUC compared to the non-Quickshift++ counterparts where the number of components k is tuned by validation (see Table I.1), yet they also manage to maintain or sometimes outperform the baseline AUC of OCSVM.

Training Time. Our main goal was to maintain the training efficiency of OCSVM, and we see in Table 2 that our methods using Quickshift++ achieve this goal, and even manage some minor speedup over OCSVM. We will see in Appendix I.1 that our method without Quikshift++ achieve significantly faster training time, although with the added complexity of an additional tuning parameter k. OC-Nyström-QS, as their performance is similar to their non Quickshift++ counterparts, while at the same time they are more applicable in all scenarios, including when no validation data is available for tuning (results of Section J).

Significant Savings in Detection Time and Space. We report significant savings on detection time and space, for all proposed variants of our approach, which is the main motivation of this work. Table 3 and Table 4 present results for OC-KJL-QS and OC-Nyström-QS, while similar time and space savings under OC-KJL and OC-Nyström are presented in Appendix Tables I.2 and I.3.

Table 3: *OC-KJL-QS: Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	24.68 ± 0.02	24.22 ± 0.93	15.61 ± 0.01	27.56 ± 0.05	26.77 ± 0.05	26.38 ± 0.05	25.19 ± 0.06
	NANO	27.18 ± 0.06	31.67 ± 0.30	21.29 ± 0.02	33.95 ± 0.31	27.20 ± 0.04	30.44 ± 0.03	28.78 ± 0.01
	Server	26.33 ± 0.21	28.10 ± 1.92	17.17 ± 0.21	29.72 ± 0.34	24.15 ± 0.30	32.32 ± 0.05	30.94 ± 0.01
Space Reduction		41.36 ± 0.01	36.30 ± 0.03	27.18 ± 0.00	38.21 ± 0.02	32.83 ± 0.00	40.73 ± 0.00	37.42 ± 0.01

Table 4: *OC-Nyström-QS: Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	22.86 ± 0.02	22.63 ± 0.87	15.54 ± 0.01	27.08 ± 0.05	26.48 ± 0.05	26.24 ± 0.05	25.56 ± 0.06
	NANO	24.73 ± 0.05	29.63 ± 0.28	21.53 ± 0.02	33.56 ± 0.31	27.39 ± 0.04	30.07 ± 0.03	29.14 ± 0.01
	Server	24.16 ± 0.20	25.90 ± 1.77	17.31 ± 0.21	29.44 ± 0.33	24.36 ± 0.30	31.73 ± 0.05	31.23 ± 0.01
Space Reduction		39.99 ± 0.01	35.09 ± 0.03	27.21 ± 0.00	37.98 ± 0.02	32.97 ± 0.00	40.66 ± 0.00	37.63 ± 0.01

Our approaches are at least 15 times faster than OCSVM on every machine we considered. Speedups on RSPI and the server are most considerable, up to 20+ times faster than OCSVM on many datasets. We see a small distinction between OC-KJL-QS and OC-Nyström-QS, whereby the former tends to achieve higher speedups on all machines for most datasets. As such both approaches seem to offer a tradeoff where, as per Table 2, the Nyström based approaches tend to achieve slightly higher AUC on most datasets.

In all cases, we observe significant space reductions, as our models can be stored upwards of 27 times less space, and up to 41 times less that the baseline OCSVM model. This smaller memory footprint implies the possibility for a wider deployment than a conventional OCSVM, especially on memory-restricted devices such as on the embedded devices we evaluated. Although we focused much of our evaluation on memory-constrained devices, which is a common deployment scenario for IoT, the space efficiency of these models is important even in server settings where a server might host large numbers of detection tools each dedicated to monitoring a given machine on client networks.

5 Conclusion

In this paper, we have extended the state-of-the-art OCSVM to more efficient representations using projection and clustering. We have demonstrated that these procedures result in 15-40x improvements in both time and space, without sacrificing detection accuracy across a wide range of novelty detection problems in IoT. These approaches, OC-Nyström-QS and OC-KJL-QS, are more widely applicable under practical use cases of novelty detection in IoT, and in particular deployable not only on powerful servers but also on single-board computing devices with more limited memory and computing resources. Our evaluation of these techniques also exposed some clear tradeoffs: when minimally tuned with a few labeled data, OC-Nyström-QS tends to achieve higher detection performance than OC-KJL-QS, at the cost of some increase in computation time.

References

[1] ARM. ARM Cortex-A72 BCM2711 processor. https://github.com/raspberrypi/documentation/blob/develop/documentation/asciidoc/computers/processors/bcm2711.adoc, 2016. Accessed: 2021-10-11.

[2] P. Biondi. Scapy. https://scapy.net/index, 2021.

[3] S. García. Malware on IoT Dataset (CTU IoT). https://www.stratosphereips.org/datasets-iot, 2019. Accessed: 2019-12-13.

[4] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, and P. V. et al. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[5] H. Jiang, J. Jang, and S. Kpotufe. Quickshift++. https://github.com/google/quickshift. Accessed: 2020-06-13.

[6] H. Jiang, J. Jang, and S. Kpotufe. Quickshift++: Provably good initializations for sample-based mean shift. *arXiv preprint arXiv:1805.07909*, 2018.

[7] An efficient one-class svm for novelty detection source code. https://github.com/kun0906/kjl.

[8] S. Kpotufe and B. Sriperumbudur. Gaussian sketching yields a jl lemma in rkhs. In *International Conference on Artificial Intelligence and Statistics*, pages 3928–3937, 2020.

[9] A. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. Technical report, 2013.

[10] K. Naga and R. Kaizaki. MAWI WIDE Dataset (MAWI). https://mawi.wide.ad.jp/mawi/, 2020. Accessed: 2020-09-13.

[11] C. W. O'Brien, L. L. Costis Toregas, and D. Pruitt-Mentle. National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) Data. https://www.netresec.com/?page=MACCDC, 2012. Accessed: 2020-05-13.

[12] T. Oliphant. Numpy configuration. https://numpy.org/devdocs/reference/generated/numpy.show_config.html, 2005. Accessed: 2019-12-13.

[13] A. Rudi, R. Camoriano, and L. Rosasco. Less is more: Nyström computational regularization. In *NIPS*, pages 1657–1665, 2015.

[14] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[15] K. Yang, S. Kpotufe, and N. Feamster. A comparative study of network traffic representations for novelty detection. *arXiv preprint arXiv:2006.16993*, 2020.

[16] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. *Advances in neural information processing systems*, 25:476–484, 2012.

A Experiment Details

Data Sources. We consider both publicly available traffic traces and traces collected on private consumer IoT devices. We aim to evaluate a representative set of devices, from multi-purpose devices, such as laptop PCs and Google Home, to less complex electronics and appliances with few modes of operations, such as smart cameras or smart fridges. Furthermore, we aim at a representative set of *novelties*, from benign novelties (new activity or a new device type) to novelties due to malicious activities (DDoS attack). Table C.1 of the Appendix describes these datasets, along with associated types of novelty being detected.

Although some of the devices that we test are multipurpose (as such might display more modalities than a special-purpose IoT device), including them allows us to test how well our approach scales. In particular, we will see that efficient detection is possible even in such cases, as even then $k \leq 20$ clusters suffice to maintain detection performance over these datasets, keeping $d = 5$ and $m = 100$.

Representation: Flows. Our unit of measurement consists of traffic flows, described below, i.e., as we aim to flag flows as normal or novel. We parse bidirectional flows from datasets in Table C.1 using Scapy [2] and extract interarrival times and packet sizes. Because certain devices can have arbitrarily long flows, we truncate each flow from a given dataset to have duration at most that of the 90th upper-percentile of flow durations in the dataset. Henceforth, a *flow* refers to these choices of flows involving truncation. We randomly split the obtained data into training, validation, and test sets of sizes detailed in Table C.2 in Appendix.

Representation: Features. Every flow is represented as a vector of the interarrival times between packets, i.e., in microseconds elapsed between consecutive packets, along with the size in bytes of each packet in the flow (IAT+SIZE). We select these features as it results in competitive detection accuracy for OCSVM. This is illustrated, e.g., against 2 common alternative feature choices, namely STATS, and SAMP_SIZE, as shown in the Table A.1. A different choice, STATS+HEADER corresponds to common statistics on flows, e.g., flow duration, mean, standard deviation and quantiles of packet sizes, in addition to packet header information [15] as explained in detail in Appendix K.2.

Table A.1: *Average AUCs for alternative features.*

Dataset	MAWI	SFRIG	DWSHR
IAT+SIZE	0.99 ± 0.00	0.93 ± 0.00	0.93 ± 0.00
STATS	1.00 ± 0.00	0.86 ± 0.00	0.65 ± 0.00
SAMP_SIZE	0.99 ± 0.00	0.93 ± 0.00	0.92 ± 0.00

Results for the alternative features set STATS+HEADER are given in Appendix K.3, further demonstrating that our speedups generally hold over choices of data representation, as significant savings in time and space over baseline OCSVM remain. Such generality is expected because the main source of savings in both time and space stands, namely, the succinct finite-dimensional modeling of the infinite-dimensional representation inherent in OCSVM, made possible by the few modalities displayed by typical IoT devices.

Implementation and Hyperparameters. All detection procedures are implemented in Python, calling on the `scikit-learn` package for existing procedures such as OCSVM and GMM. While OCSVM training uses the standard `libsvm` package, we re-implemented its detection routines (as described in Section 2.1) using `Numpy` to ensure fair, apples-to-apples execution time comparison with OC-Nyström and OC-KJL, which are implemented in `Numpy`, a Python library that calls on fast algebraic operations and parallel processing on multicore machines [4]. The Nyström and KJL projections are implemented as described above; All the source codes can be seen at [7]

Training Scenarios. We consider two practical scenarios: one where some small amount of labeled *novelty* data is available to *validate* hyperparameter choice in a controlled lab environment, and one with no such labeled validation data, where we have to result in default choices of hyperparameters. Although each detection procedure may have many internal parameters, this distinction in scenarios only applies to two key choices of hyperparameters:

- *Kernel Bandwidth h.* For all methods, i.e., OCSVM, OC-Nyström, OC-KJL, we use a Gaussian kernel of the form $K(x, x') \propto \exp(-\|x - x'\|^2/h^2)$, where the *bandwidth* h is to be picked as a quantile of $\binom{n}{2}$ distances between the n training datapoints. In all our results we consider 10 quantiles $[0.1, 0.2, \ldots, 0.9] \cup \{0.95\}$ of increasing interpoint distances.

- *Number of GMM components k.* As explained above, OC-Nyström and OC-KJL also require a choice of number of GMM components to fit. We consider choices in the range $[1, 4, 6, 8, 10, 12, 14, 16, 18, 20]$. Thus the number of components, or *clusters* k is capped at 20, as IoT devices are expected to display relatively few modes of operations–i.e., clusters of normal network activity.

  For the results presented in the main text, the choice of k is made by QuickShift++ *and the resulting procedures are denoted* **OC-Nyström-QS***, and* **OC-KJL-QS**: these versions of our fast methods therefore only leave the choice of bandwidth h, and will be our main focus onwards.

Next, we discuss how the above parameters are picked in each of the use cases or scenarios discussed above.

- **Minimal Tuning**. To simulate the first training scenario where some small amount of labeled novelty data is available, we subsample a small amount of the novelty data, which paired with equal amount of normal data is used to form a *validation set* to be used in hyperparameter choice. We then proceed to choose h or k (when Quickshift++ is not used) to minimize AUC over the validation data, so that these choices are independent of the random test set on which final results are reported.

- **No Tuning.** In this case, we choose the bandwidth h by a common rule-of-thumb as the 0.3 quantile of increasing interpoint distances on the training data. The choice of the number of components k is then always made by Quickshift++. We observed the same speedups under these settings, although these results are related to Appendix J for space.

All Other Algorithmic Parameter Choices are Fixed: We now describe all other choices inherent in our procedures, OC-Nyström and OC-KJL, and their variants OC-Nyström-QS, and OC-KJL-QS.

- *Projection Parameters.* As discussed in Section 3.2, subsamples size m, and projection dimension d, are fixed to $m = 100$ and $d = 5$, choices which remarkably preserve detection performance across datasets and types of novelty, despite the considerable amount of *compression* they entail.

- *Quickshift++ Parameters.* We use the implementation of [6, 5], which requires internal parameters β set to 0.9 (this performs density *smoothing*) and number of neighbors set to $n^{2/3}$ (to build a *dense* neighborhood graph whose connectivity encodes high-density regions), two choices which work well across device datasets and types of novelty.

Here, due to variability in the data, Quickshift++ can often return too many *outlier* clusters (despite the conservative setting of its internal parameters). To remove those, we only retain *large* clusters, namely the smallest number of clusters that account for at least 95% of the data, if this number is less than 20, otherwise we retain the 20 largest clusters discovered by Quickshift++.

Gaussian Mixture Models Parameters. We have the choice of using either *full* Gaussian covariances in fitting a GMM model to the projected data after KJL or Nyström, or of using only diagonal covariances for faster fitting – especially when operating in high dimensional settings – but at the usual cost of some loss in accuracy. Since GMMs are fit after projection to low dimension $d = 5$, it turns out that full Gaussian covariances are in fact efficient to fit in our case, so we only report results for full covariances.

When using Quickshift++, we initialize GMM with the clusters returned, i.e., local means and covariances of these clusters, and train till convergence.

B Computing Platforms

We perform our experiments on two computing platforms: (1) a well-provisioned server, for the use case where all training and detection might occur offline; and (2) resource-constrained devices, specifically a Raspberry Pi and an NVidia Jetson Nano, corresponding to the use case where detection is to be real-time, local to the IoT device.

Table. B.1 shows all three machines' information (i.e., Large server, Raspberry Pi, and Nvidia Nano), which includes operating system, CPU, memory, storage, programming language, and a scientific computing package (Numpy). Note that 'lscpu' command is used to get CPU and CPU cache information. In addition, for Raspberry Pi, we get the cache information from the ARM official document of the Cortex-72 processor [1].

Table B.1: *We train on server and test on all 3 machines.*

Machine	Description
Large Server	64-bit, running Debian GNU/Linux 9 (stretch) with Intel(R) Xeon(R) processor (32 CPU Cores, 1200-3400 MHz each), 100GB memory, and 2TB disk. Caches: 3 layers of CPU caches (i.e., 32KiB L1d and L1i, 256KiB L2, and 20MB L3). Programming language: Python 3.7.9. Numpy 1.19.2 (install from the numpy wheel that includes an OpenBLAS implementation of the BLAS and LAPACK linear algebra APIs [12]).
Raspberry Pi (RSPI)	32-bit, running Raspbian GNU/Linux 10 (buster) with Cortex-A72 processor (4 CPU cores, 600-1500 MHz each), 8GB memory, and 27GB disk. Caches: 2 layers of CPU caches (i.e., 32KiB L1d, 48KiB L1i, and 1MiB L2 [1]). Programming language: Python 3.7.3 with '–enable-optimizations' option. Numpy 1.18.2 (install from the numpy source codes that searches for BLAS and LAPACK dynamic link libraries at build time as influenced by the system environment variables [12]).
Nvidia Nano (NANO)	64-bit, running Ubuntu 18.04.5 LTS (Bionic Beaver) with Cortex-A57 processor (4 CPU cores, 102-1479 MHz each), 4GB memory, and 30GB disk. Caches: 2 layers of CPU caches (i.e., 32KiB L1d, 48KiB L1i, and 2MiB L2) Programming language: Python 3.7.3 with '–enable-optimizations' option. Numpy 1.18.2 (install from the source codes [12]).

C Datasets

Table C.1: *Datasets' details.*

Datasets	Description	Devices	Type of Novelty
Lab IoT SFRIG	Data traces are generated by a Samsung Fridge (SCam) with IP '192.168.202.43' in a private lab environment. It has two types of traffic traces labeled as *normal* when there is no human interaction and, *novel* when being operated by a human (such as opening the fridge).	One fridge	Novel activity
Lab IoT AECHO	Data traces are generated by an Amazon ECHO (AECHO) with IP '192.168.202.74' in a private lab environment. It has two types of traffic traces labeled as *normal* when there is no human interaction, and *novel* when being operated by a human (such as buying food by the ECHO).	One Amazon ECHO	Novel activity
Lab IoT DWSHR	Data traces are generated by a dishwasher (DWSHR) with IP '192.168.202.76' in a private lab environment. It has two types of traffic traces labeled as *normal* when there is no human interaction, and *novel* when being operated by a human (such as opening the dishwasher). We also add another novel traffic (such as buying food by an AECHO) collected from an Amazon ECHO (with IP '192.168.202.174') into *novel* to get a bigger testing set.	One dishwasher and one Amazon ECHO	Novel activity
CTU IoT [3]	Bitcoin-Mining and Botnet traffic traces generated by two Raspberries; we use Botnet traffic (with IP '192.168.1.196') as *normal* and Bitcoin-Mining traffic (with IP '192.168.1.195') as *novel*.	Two infected Raspberry Pis	Novel (infected) device
UNB IDS [14]	Normal traces are generated by one personal computer (PC) with IP address is '192.168.10.9'. Attack traces are generated by three PCs with IP addresses: '192.168.10.9', '192.168.10.14', and '192.168.10.15'.	Four PCs	DDoS attack
MAWI [10]	Normal traffic are collected on July 01, 2020; we choose one kind of traffic generated by a PC with IP '203.78.7.165' as *normal*, and another kind of traffic generated by a PC with IP address '185.8.54.240' as *novel*.	Two PCs	Novel (normal) device
MACCDC [11]	Data traces are collected in 2012; we choose one kind of traffic generated by a PC with IP '192.168.202.79' as *normal* and one kind of traffic generated by a PC with IP '192.168.202.76' from another pcap as *novel*.	Two PCs	Novel (normal) device

Datasets and Types of Novelty. Table C.1 describes the datasets we used in the main paper, along with the associated types of novelty being detected. There are seven datasets in total, in which three of them are IoT datasets collected from three IoT devices deployed at the University of Chicago, and the remaining four are public datasets (i.e., CTU IoT, UNB IDS, MAWI, and MACCDC). Furthermore, the types of novelty vary from benign novelties (new activity or a new device type) to novelties due to malicious activities (DDoS attack).

Dataset Sizes and Dimensions. Table C.2 shows Train set, Validation (Val.) set, Test set and feature dimensions. The size of validation sets are always 1/4 of corresponding test set sizes. In Validation set and Test set, the number of normal and abnormal data is equal. Moreover, the IAT+SIZE dimension of each dataset is less than 50, except for MAWI (its dimension is 121).

Table C.2: *Dataset sizes (# of data points) and dimensions.*

Dataset	UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Train Set	10,000	10,000	5,720	10,000	10,000	10,000	10,000
Val. Set	462	1,250	1,040	1,250	1,250	280	508
Test Set	1,854	5,000	4,160	5,000	5,000	1,120	2,032
Dimensions	43	23	121	25	15	35	29

Data Splitting. For each dataset, first all flows (normal and abnormal) are preprocessed into the IAT+SIZE features, which creates at set of *normal* and novel data, from which we we draw random subsamples. Experiments on each dataset follow the steps outlined below.

(i) Draw a subsample of size 600 to 2500 from normal data, and a subsample of size 600 to 2500 from novelty data to form a test dataset of size 1200 to 5000. Exact sizes are given in Appendix Table C.2.

(ii) Repeat 5 times for accurate AUC:
- Draw a subsample of size $n = 10K$ from normal data, to form the training data, except for MAWI ($n = 5.7K$).
- *If tuning:* draw a validation sample (1/4 test set size).
- Choose parameters h, k as described in Section A.
- Train with the choice of h, k and save model on disk.
- Load and test model on Test data: repeat 100 times for accurate timing on machine (retain aggregate time).

D Evaluation Metrics

Detection Performance. In novelty detection, there is a well-known tension between *false detection a.k.a. false positive rates* (FDR, i.e., the proportion of normal data wrongly flagged as novel) – and *true detection a.k.a. true positive rates* (TDR, i.e., the percentage of abnormal data rightly flagged as novel). Such tradeoffs are well captured by a Receiver Operating Characteristic (ROC) curve, which plots the detection rate TDR against the false alarm rate FDR as the detection threshold t is varied from small to large; thus, the area under the ROC curve (**Area Under the Curve** (AUC)) when it is large, i.e., close to 1, indicates that good tradeoffs are achieved by the given detection approach. In contrast, AUC below 0.5 signals poor tradeoffs. AUC is therefore commonly adopted as a sensible measure of detection performance, as it captures tradeoffs under the complete range of detection choices.

In practice, a single threshold is chosen, driven by application-specific constraints, as one might prefer high TDR over low FDR, or vice versa (think of an infected medical device, e.g., a pacemaker, where high TDR would be preferred, vs. an infected smart home appliance, e.g., a toaster, where low FDR might be preferred). Large AUC, thus, indicates that the detector allows for good choices in any of these situations. For our proposed fast detectors, we will be interested in the fraction of AUC retained over OCSVM, i.e., the AUC of our detector divided by that of OCSVM.

Training and Detection (or Testing) Time. We will measure time as the *wall-clock time* taken by any of the methods for training (not-including data preprocessing into feature vectors, but inclusive of all actual training, i.e., modeling fitting) and *testing*, i.e., actual detection computations, on given machine environments (see Section B) after a model is obtained. We report the *speedup*, the ratio of wall-clock time for OCSVM over that of our detector, separately for training and testing.

Detection (or Testing) Space. We report the space taken by the model returned by the detection procedure in kiloBytes. Namely, we report the minimal amount of information on the learned model to be saved towards future detection. That is, (1) support vectors and coefficients for OCSVM, and

(2) projection parameters and GMM components for OC-Nyström and OC-KJL (with or without Quickshift++).

E OCSVM's Detection Time and Space

It should be clear by now that computational complexity is determined by the number $\tilde{n} \leq n$ of nonzero α_i's. The corresponding datapoints X_i's are called the *support vectors*, and have to be kept in memory to estimate $f(x)$. Thus the OCSVM detector takes space $\tilde{n} \cdot (D+1)$, while computation time for f is $\Omega(\tilde{n} \times D)$. Unfortunately, it is often the case that $\tilde{n} = n$ or is of the same order, while the larger n, the more accurate the detector is.

F OCSVM Mapping

A main intuition behind the mapping ϕ, implicit in both supervised SVM and unnsupervised OCSVM, is that it manages to *separate* classes of data, i.e., pull corresponding datapoints far apart in $\mathcal{H}$, even when they are not easily separable in their original representation in $\mathbb{R}^D$. This is illustrated in Figure F.1. It follows that, after the mapping ϕ, the data might become linearly separable in $\mathcal{H}$, i.e., the two classes of data, *normal* and *abnormal*, fall on different sides of a hyperplane in $\mathcal{H}$.

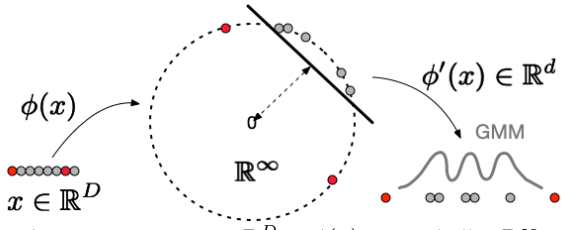

Figure F.1: *OCSVM maps datapoints $x \in \mathbb{R}^D$ as $\phi(x)$ onto a ball in $\mathbb{R}^\infty$, inducing linear separation between normal points (gray) and yet unseen novel points (in red). We consider mappings ϕ' which then remap down to $\mathbb{R}^d$, $d \ll D$, while maintaining separation (into clusters), but not necessarily linear separability. Given the few modalities of IoT devices, we can then learn a GMM with few components to model the remapped* normal *data.*

G KJL/Nyström Projection

This is illustrated in Figure G.1, on simulated data of size $n = 10000$, with two classes that are not easily clustered in $\mathbb{R}^D$, but which are clusterable not only in $\mathcal{H}$, but also after Nyström of KJL. In that simulation we used $d = 2$, and $m = 100$. Similar small values are used for our experiments on real-world IoT data (see experimental setup in Section A).

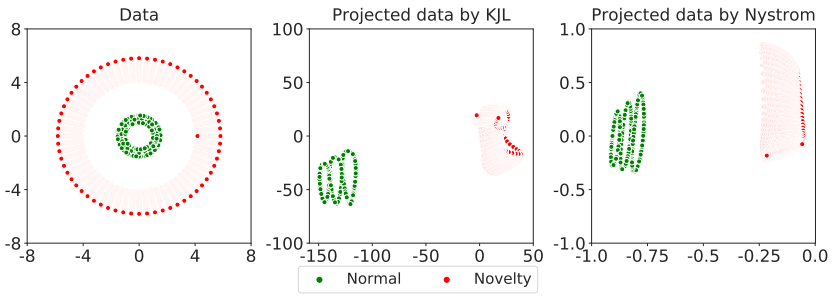

Figure G.1: *Clusters after mapping ϕ': the simulation data* Cluster in Cluster *has 5000 points, shown before and after KJL/Nyström mapping. The KJL/Nyström mapping ϕ' – shown on the right' retains the clusters uncovered by the initial kernel mapping ϕ.*

H OC-KJL vs OC-KJL-SVM(linear)

We compare OC-KJL (using a GMM) and OC-KJL-SVM (using a linear separtor). Figure H.1 shows that we compare fitting a linear separator after KJL projection (denoted OC-KJL-SVM) to our proposed method (OC-KJL). The detection performance metric is the AUC, which is consistently higher for OC-KJL across datasets.

I OC-Nyström and OC-KJL Results (no Quickshift++)

In the main paper body, we left out some of the detection time and space savings results for the OC-KJL and OC-Nyström variants (which do not use Quickshift++ for automatic cluster-number identification). Here we consider the effect of additionally tuning the number of GMM components rather than selecting them automatically via Quickshift++. The main message here is that not much is lost in AUC by automatically choosing k via Quickshift++.

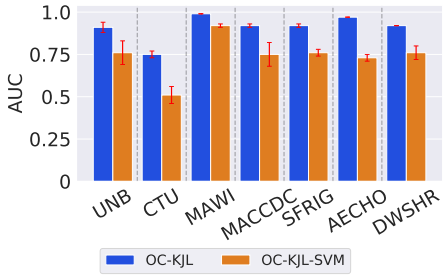

Figure H.1: *OC-KJL (using a GMM) performs better than OC-KJL-SVM (using a linear separtor).*

I.1 Retained AUC and Training Efficiency

Table I.1 compares AUC and training times of OC-KJL and OC-Nyström to that of the baseline OCSVM.

Table I.1: *Retained AUC (method/OCSVM) and server train time speedup (OCSVM time/method time).*

Method \ Dataset	UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
OC-KJL: AUC Retained	1.36 ± 0.04	1.13 ± 0.04	0.99 ± 0.01	1.08 ± 0.01	0.99 ± 0.01	1.08 ± 0.00	0.99 ± 0.00
Train Speedup	2.75 ± 0.14	2.44 ± 0.05	4.13 ± 0.08	2.47 ± 0.06	2.22 ± 0.21	2.84 ± 0.17	2.75 ± 0.14
OC-Nyström: AUC Retained	1.41 ± 0.02	1.19 ± 0.03	0.99 ± 0.01	1.07 ± 0.03	0.97 ± 0.01	1.09 ± 0.00	0.99 ± 0.01
Train Speedup	3.14 ± 0.16	2.59 ± 0.06	3.68 ± 0.07	2.60 ± 0.06	2.17 ± 0.21	2.94 ± 0.17	2.72 ± 0.14

AUC Retained. We see that our detection methods, OC-Nyström and OC-KJL, retain the detection performance of OCSVM, all within a ratio of 1 or more, except in the case of MAWI, SFRFIG, and DWSHR – which are still high AUCs considering OCSVM's high accuracy on MAWI, SFRIG, and DWSHR (Table 1). Moreover, for some datasets, such as UNB and CTU, all procedures manage to actually outperform OCSVM. It is likely that such higher performance is due to the additional regularization inherent in the dimension reduction performed by our methods.

Training Time. Although our original goal was just to maintain the training efficiency of OCSVM, especially considering the various additional steps inherent in our methods, our methods without Quickshift++ in fact achieve speedup – factors of 2-4 in some cases – over OCSVM training time which involves more expensive model fitting steps.

I.2 Significant Savings in Detection Time and Space

Tables I.2 and I.3 presents results on detection time and space savings for both OC-KJL and OC-Nyström.

Testing Time Speedup. We observe speedups of at least 13 times over the baseline OCSVM detection times across all machines and datasets.

Space Reduction. As before, space reductions are significant w.r.t. the baseline OCSVM from 26 to 42+ times less space than required by the baseline.

Table I.2: *OC-KJL: Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	26.66 ± 0.02	22.81 ± 0.88	13.25 ± 0.01	25.91 ± 0.05	26.20 ± 0.05	24.52 ± 0.04	25.07 ± 0.06
	NANO	29.66 ± 0.06	29.90 ± 0.28	18.24 ± 0.02	31.45 ± 0.29	27.13 ± 0.04	27.36 ± 0.02	28.38 ± 0.01
	Server	28.80 ± 0.23	26.41 ± 1.80	14.73 ± 0.18	28.19 ± 0.32	23.63 ± 0.29	29.11 ± 0.05	30.48 ± 0.01
Space Reduction		42.55 ± 0.01	35.12 ± 0.03	26.52 ± 0.00	36.74 ± 0.02	32.58 ± 0.00	39.20 ± 0.00	37.13 ± 0.01

Table I.3: *OC-Nyström: Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	27.17 ± 0.02	24.30 ± 0.93	12.59 ± 0.01	27.02 ± 0.05	24.87 ± 0.05	25.33 ± 0.04	24.49 ± 0.06
	NANO	29.98 ± 0.07	31.92 ± 0.30	17.14 ± 0.02	33.62 ± 0.31	25.88 ± 0.03	28.24 ± 0.02	27.94 ± 0.01
	Server	28.31 ± 0.23	27.60 ± 1.88	13.95 ± 0.17	29.71 ± 0.34	23.22 ± 0.29	30.24 ± 0.05	29.91 ± 0.01
Space Reduction		42.79 ± 0.01	36.55 ± 0.03	26.25 ± 0.00	37.98 ± 0.02	31.60 ± 0.00	39.82 ± 0.00	36.89 ± 0.01

J Results under No Tuning

We now consider the scenario where no validation data is available to tune any of the procedures, i.e., in choosing the bandwidth parameter h. While in general it is preferable to perform some minimal tuning before deployment, in practice, it may be difficult to obtain labeled data for the types of novel activities of interest that commonly arise in actual deployment environments.

In the practice of novelty detection with OCSVM, when no labeled data is available, various rule-of-thumbs are used, a popular one being to pick h as a quantile of inter-point distances. For uniformity, as explained in Section A, here we pick h for all methods, as the 30th percentile of increasing inter-point distances in the training data.

Naturally, detection performance suffers w.r.t. that of a tuned procedure for any of the methods. Furthermore, since the choice of bandwidth affects the learned model, it is to be expected that time and space comparisons would also differ from that under minimal tuning as in the previous Section 4.2.

J.1 Baseline OCSVM Performance

Table J.1 shows the performance of the baseline OCSVM. We observe a decrease in AUC for most datasets, most considerably for UNB and CTU, which already were hard datasets even under tuning (Table 1). Interestingly, MAWI, MACCDC, SFRIG, and DWSHR still admit high AUCs even without tuning, attesting to the general appeal of OCSVM as an adaptable and robust novelty detection approach. Although AECHO and MACCDC dropped in accuracy (from 0.85+), they still retain reasonable accuracies with AUC's at 0.78.

Table J.1: *OCSVM baseline performance,* **no tuning**. *Time is in ms per 100 datapoints and space is in kB.*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
AUC		0.60 ± 0.01	0.57 ± 0.01	0.98 ± 0.00	0.78 ± 0.00	0.85 ± 0.00	0.78 ± 0.00	0.87 ± 0.01
Server Train Time (ms)		94.23 ± 1.66	81.47 ± 2.39	78.48 ± 0.93	84.72 ± 1.01	85.15 ± 2.99	117.65 ± 6.46	117.29 ± 8.81
Test Time (ms)	RSPI	124.54 ± 0.12	124.12 ± 0.35	74.48 ± 0.64	124.17 ± 0.59	123.70 ± 0.36	126.31 ± 0.95	125.46 ± 0.72
	NANO	89.30 ± 0.11	86.32 ± 0.17	50.71 ± 0.05	86.70 ± 0.21	83.50 ± 0.17	88.85 ± 0.02	88.00 ± 0.09
	Server	19.20 ± 0.13	19.42 ± 0.22	9.76 ± 0.13	19.10 ± 0.27	18.62 ± 0.15	19.95 ± 0.02	19.72 ± 0.09
Space (kB)		1761.21 ± 0.17	961.40 ± 0.23	2798.32 ± 0.78	1041.34 ± 0.13	641.07 ± 0.05	1441.75 ± 0.14	1201.38 ± 0.32

J.2 Retained AUC and Training Efficiency

Table J.2 compares AUC and training times of OC-KJL-QS and OC-Nyström-QS to that of the baseline OCSVM, using the exact same default choice of bandwidth h as OCSVM.

Table J.2: No tuning. *Retained AUC (method over OCSVM) and train time speedup (OCSVM over method).*

Method \ Dataset	UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
OC-KJL-QS: AUC Retained	1.49 ± 0.02	1.20 ± 0.05	0.14 ± 0.00	0.90 ± 0.14	0.98 ± 0.03	1.20 ± 0.03	0.95 ± 0.02
Train Speedup	1.32 ± 0.02	1.14 ± 0.03	1.37 ± 0.02	1.16 ± 0.01	0.98 ± 0.03	1.29 ± 0.07	1.26 ± 0.09
OC-Nyström-QS: AUC Retained	1.54 ± 0.03	1.19 ± 0.10	0.17 ± 0.00	0.88 ± 0.17	0.77 ± 0.11	1.18 ± 0.00	0.90 ± 0.05
Train Speedup	1.32 ± 0.02	1.14 ± 0.03	1.47 ± 0.02	1.14 ± 0.01	0.97 ± 0.03	1.26 ± 0.07	1.22 ± 0.09

AUC Retained. OC-KJL-QS and OC-Nyström-QS manage to retain the AUC of OCSVM on most datasets. However, on MAWI, neither OC-Nyström-QS nor OC-KJL-QS does well, arriving at just a fraction of the baseline AUC. MACCDC and SFRIG appear to cause problems for OC-Nyström-QS under the default h setting.

Training Time. As before, training time remains competitive with that of OCSVM with some significant reduction in time for instance in the case of UNB, AECHO, and DWSHR.

J.3 Significant Savings in Detection Time and Space

Tables J.3 and J.4 present results on detection time and space savings for both OC-KJL-QS and OC-Nyström-QS, again with the same default choice of bandwidth h as OCSVM. The trends on savings are similar, but in fact even better than those under minimal tuning of these 3 methods.

Table J.3: *OC-KJL-QS,* **no tuning***. Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	24.30 ± 0.02	25.81 ± 0.07	12.39 ± 0.11	27.07 ± 0.13	27.37 ± 0.08	25.66 ± 0.19	25.40 ± 0.15
	NANO	28.94 ± 0.04	32.18 ± 0.07	15.99 ± 0.02	34.43 ± 0.08	34.21 ± 0.07	29.97 ± 0.01	30.88 ± 0.03
	Server	25.80 ± 0.18	28.28 ± 0.33	13.38 ± 0.18	28.85 ± 0.41	28.30 ± 0.23	32.34 ± 0.03	31.49 ± 0.15
Space Reduction		41.30 ± 0.00	36.50 ± 0.01	26.28 ± 0.01	38.43 ± 0.00	33.57 ± 0.00	40.67 ± 0.00	37.80 ± 0.01

Table J.4: *OC-Nyström-QS,* **no tuning***. Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	22.70 ± 0.02	24.50 ± 0.07	12.38 ± 0.11	26.31 ± 0.13	26.79 ± 0.08	23.75 ± 0.18	24.79 ± 0.14
	NANO	26.36 ± 0.03	30.28 ± 0.06	15.97 ± 0.02	33.22 ± 0.08	33.29 ± 0.07	27.37 ± 0.01	29.92 ± 0.03
	Server	23.06 ± 0.16	26.89 ± 0.31	13.38 ± 0.18	28.06 ± 0.40	28.37 ± 0.23	29.33 ± 0.03	30.60 ± 0.15
Space Reduction		39.94 ± 0.00	35.34 ± 0.01	26.28 ± 0.01	37.78 ± 0.00	33.02 ± 0.00	39.28 ± 0.00	37.31 ± 0.01

Testing Time Speedup. We observe speedups of at least 12 times over the baseline OCSVM detection times across all machines and datasets.

Finally, we again observe the trend where OC-KJL-QS manages faster times than OC-Nyström-QS in most cases, especially on Raspberry Pi and the server.

Space Reduction. As before, space reductions are significant w.r.t. the baseline OCSVM from 26 to 41+ times less space than required by the baseline.

K Alternative Features

K.1 SAMP-SIZE Features Description

SAMP-SIZE: a flow is partitioned into small time intervals of equal length, and the total packet size (i.e., byte count) in each interval is recorded; thus, a flow is represented as a time series of byte counts

in small time intervals. Here, we obtain time intervals according to different quantiles (i.e., [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95]) of flow durations. To ensure that each sample has the same dimension D, we select D for all flows as the 90th percentile of all *flow lengths* in the dataset (here, *flow length* stands for the number of packets a flow – not its duration).

Now for any given flow, if the number of fixed time intervals in the flow is less than D, we append 0's to arrive at a vector of dimension D. If instead the number of fixed time intervals is greater than D, we truncate the resulting vector representation down to dimension D.

K.2 STATS+HEADER Features Description

STATS+HEADER: a set of statistical quantities compiled from a flow. In particular, we choose 10 of the most common such statistics in the literature (see e.g., [9]), namely, flow duration, number of packets sent per second, number of bytes per second, and the following statistics on packet sizes (in bytes) in a flow: mean, standard deviation, the first to third quantiles, the minimum, and maximum. Also, We incorporate packet header information (i.e., Time to Live (TTL) and TCP flags (FIN, SYN, RST, PSH, ACK, URG, ECE, and CWR) into the STATS to form the STATS+HEADER feature.

K.3 Results under STATS+HEADER

Table K.1: *OCSVM performance with STATS+HEADER. Time is in ms per 100 datapoints and space is in KB.*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
AUC		0.60 ± 0.01	0.61 ± 0.00	1.00 ± 0.00	0.76 ± 0.00	0.86 ± 0.00	0.98 ± 0.00	0.65 ± 0.00
Server Train Time (ms)		77.20 ± 1.75	94.54 ± 2.73	64.63 ± 0.82	92.23 ± 2.25	88.69 ± 0.90	100.58 ± 2.78	92.59 ± 2.93
Test Time (ms)	RSPI	123.90 ± 0.39	119.29 ± 3.31	72.39 ± 0.08	114.20 ± 0.33	124.53 ± 0.57	123.42 ± 0.34	125.70 ± 0.22
	NANO	76.86 ± 0.26	87.13 ± 0.31	54.33 ± 0.01	87.22 ± 0.06	82.89 ± 0.05	78.33 ± 0.04	83.53 ± 0.06
	Server	17.60 ± 0.38	19.08 ± 0.36	11.17 ± 0.16	19.21 ± 0.18	19.09 ± 0.23	19.45 ± 0.02	19.83 ± 0.02
Space (kB)		1655.41 ± 2.72	1240.96 ± 0.16	1831.75 ± 0.00	1280.82 ± 0.13	1084.04 ± 0.22	1482.07 ± 0.30	1363.32 ± 0.28

K.3.1 Results Under Minimal Tuning

Table. K.1 shows the baseline results obtained by OCSVM under minimal turning. Accuracies of the baseline OCSVM are similar to those using the features of IAT+SIZE in the main paper and are reported for completion.

Similar to the case of IAT+SIZE features, both OC-KJL and OC-Nyström with or without Quickshift++ retain the AUC of the baseline OCSVM (on UNB and DWSHR, our procedures even get higher AUCs than OCSVM). Moreover, these procedures have 2-4 times train time speedup. More details are shown in Table K.2.

Table K.2: *Retained AUC (method over OCSVM) and server train time speedup (OCSVM over method) with STATS+HEADER.*

Method \ Dataset	UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
OC-KJL: AUC Retained	1.33 ± 0.07	0.94 ± 0.08	1.00 ± 0.00	0.94 ± 0.01	1.01 ± 0.01	1.00 ± 0.01	1.18 ± 0.03
Train Speedup	2.63 ± 0.06	3.11 ± 0.09	3.02 ± 0.04	2.98 ± 0.07	2.63 ± 0.03	3.33 ± 0.09	2.52 ± 0.08
OC-KJL-QS: AUC Retained	1.13 ± 0.05	0.87 ± 0.04	0.98 ± 0.01	0.89 ± 0.02	0.98 ± 0.01	0.98 ± 0.01	1.05 ± 0.08
Train Speedup	1.02 ± 0.02	1.34 ± 0.04	1.40 ± 0.02	1.39 ± 0.03	1.31 ± 0.01	1.34 ± 0.04	1.18 ± 0.04
OC-Nyström: AUC Retained	1.45 ± 0.02	1.02 ± 0.05	0.99 ± 0.00	0.94 ± 0.05	0.96 ± 0.02	0.99 ± 0.01	1.18 ± 0.04
Train Speedup	2.45 ± 0.06	3.19 ± 0.09	3.24 ± 0.04	3.15 ± 0.08	3.15 ± 0.03	3.17 ± 0.09	2.60 ± 0.08
OC-Nyström-QS: AUC Retained	1.42 ± 0.03	0.94 ± 0.04	0.96 ± 0.03	0.84 ± 0.04	0.94 ± 0.01	1.00 ± 0.01	1.06 ± 0.03
Train Speedup	1.03 ± 0.02	1.29 ± 0.04	1.46 ± 0.02	1.48 ± 0.04	1.31 ± 0.01	1.39 ± 0.04	1.19 ± 0.04

We also see that these methods under the alternative features attain significant detection time speedups over OCSVSM, which are shown in Tables K.3 and K.4.

Testing Time Speedup. We observe speedups of at least 16 times over the baseline OCSVM detection times across all machines and datasets.

Space Reduction. As before, space reductions are significant w.r.t. the baseline OCSVM from 26 to 41+ times less space than required by the baseline.

K.3.2 Results Under No Tuning

OCSVM results under no tuning for STATS+HEADER features are presented in Table K.5. As with the case of our preferred features of IAT+SIZE, we observe a significant decrease in AUC w.r.t. the tuned OCSVM case.

Table K.3: *OC-KJL-QS: Test time speedup (OCSVM over method) and space reduction (OCSVM over method) with STATS+HEADER.*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	23.81 ± 0.07	28.17 ± 0.78	16.45 ± 0.02	27.00 ± 0.08	29.66 ± 0.14	24.92 ± 0.07	26.83 ± 0.05
	NANO	25.74 ± 0.09	36.96 ± 0.13	21.17 ± 0.01	35.58 ± 0.02	35.52 ± 0.02	25.93 ± 0.01	33.73 ± 0.02
	Server	27.23 ± 0.59	30.73 ± 0.58	17.37 ± 0.25	29.35 ± 0.27	30.53 ± 0.37	29.77 ± 0.03	36.73 ± 0.03
Space Reduction		39.72 ± 0.07	40.88 ± 0.01	26.49 ± 0.00	40.72 ± 0.00	40.07 ± 0.01	40.39 ± 0.01	40.82 ± 0.01

Table K.4: *OC-Nyström-QS: Test time speedup (OCSVM over method) and space reduction (OCSVM over method) with STATS+HEADER.*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	23.79 ± 0.07	27.04 ± 0.75	16.43 ± 0.02	27.22 ± 0.08	29.88 ± 0.14	25.11 ± 0.07	28.03 ± 0.05
	NANO	25.66 ± 0.09	35.34 ± 0.13	21.14 ± 0.01	35.17 ± 0.02	35.65 ± 0.02	25.24 ± 0.01	34.63 ± 0.02
	Server	27.08 ± 0.59	29.11 ± 0.55	17.34 ± 0.25	29.16 ± 0.27	30.67 ± 0.37	28.76 ± 0.03	37.39 ± 0.03
Space Reduction		39.70 ± 0.07	40.33 ± 0.01	26.48 ± 0.00	40.64 ± 0.00	40.12 ± 0.01	39.94 ± 0.01	41.48 ± 0.01

Table K.5: *OCSVM performance with STATS+HEADER, **no tuning**. Time is in ms per 100 datapoints and space is in kB.*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
AUC		0.15 ± 0.00	0.41 ± 0.00	0.99 ± 0.00	0.50 ± 0.00	0.79 ± 0.00	0.96 ± 0.00	0.38 ± 0.01
Server Train Time (ms)		92.28 ± 2.55	89.55 ± 2.28	62.00 ± 2.11	96.15 ± 1.96	89.70 ± 1.70	95.50 ± 2.22	97.06 ± 2.16
Test Time (ms)	RSPI	124.86 ± 0.96	123.78 ± 0.82	70.74 ± 0.36	124.65 ± 0.10	123.51 ± 0.23	127.09 ± 1.05	124.99 ± 0.49
	NANO	89.86 ± 0.90	87.65 ± 0.31	46.35 ± 0.04	86.49 ± 0.14	84.04 ± 0.19	88.14 ± 0.07	85.96 ± 0.08
	Server	19.91 ± 0.65	19.32 ± 0.36	10.00 ± 0.05	19.61 ± 0.20	18.91 ± 0.23	20.82 ± 0.01	19.91 ± 0.02
Space (kB)		1641.24 ± 0.26	1241.11 ± 0.12	1832.14 ± 0.31	1282.76 ± 0.54	1081.45 ± 0.22	1481.54 ± 0.12	1361.80 ± 0.24

We also get similar significant test time speedup and space reduction results for both methods as shown in Tables K.7 and K.8. This goes to show that the reductions inherent in our approach are likely not tied to feature representations of the networking data.

Testing Time Speedup. We observe speedups of at least 12 times over the baseline OCSVM detection times across all machines and datasets.

Space Reduction. As before, space reductions are significant w.r.t. the baseline OCSVM from 25 to 43+ times less space than required by the baseline.

Table K.6 shows that OC-KJL-QS and OC-Nyström-QS, retain the AUC and train time of the baseline OCSVM using the STATS+HEADER features.

Table K.6: ***no tuning****. Retained AUC (method over OCSVM) and train time speedup (OCSVM over method) with STATS+HEADER.*

Method \ Dataset	UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
OC-KJL-QS: AUC Retained	2.29 ± 0.50	1.02 ± 0.06	0.22 ± 0.23	0.93 ± 0.09	1.00 ± 0.02	0.52 ± 0.28	1.60 ± 0.09
Train Speedup	1.41 ± 0.04	1.24 ± 0.03	1.19 ± 0.04	1.38 ± 0.03	1.22 ± 0.02	1.28 ± 0.03	1.27 ± 0.03
OC-Nyström-QS: AUC Retained	2.32 ± 0.05	1.20 ± 0.11	0.13 ± 0.01	1.11 ± 0.06	0.57 ± 0.20	0.49 ± 0.20	1.70 ± 0.10
Train Speedup	1.40 ± 0.04	1.22 ± 0.03	1.31 ± 0.04	1.43 ± 0.03	1.24 ± 0.02	1.31 ± 0.03	1.24 ± 0.03

Table K.7: *OC-KJL-QS with STATS+HEADER,* ***no tuning****: Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	28.50 ± 0.22	28.91 ± 0.19	12.99 ± 0.07	28.50 ± 0.02	28.83 ± 0.05	25.42 ± 0.21	28.04 ± 0.11
	NANO	38.34 ± 0.39	36.42 ± 0.13	15.53 ± 0.01	35.37 ± 0.06	36.16 ± 0.08	29.71 ± 0.03	36.86 ± 0.03
	Server	38.39 ± 1.25	30.22 ± 0.56	14.20 ± 0.07	29.82 ± 0.30	30.26 ± 0.37	32.25 ± 0.01	39.59 ± 0.04
Space Reduction		43.53 ± 0.01	40.89 ± 0.00	25.14 ± 0.00	40.85 ± 0.02	39.83 ± 0.01	40.65 ± 0.00	41.90 ± 0.01

Table K.8: *OC-Nyström-QS with STATS+HEADER,* ***no tuning****: Test time speedup (OCSVM over method) and space reduction (OCSVM over method).*

Dataset		UNB	CTU	MAWI	MACCDC	SFRIG	AECHO	DWSHR
Test Time Speedup	RSPI	28.16 ± 0.22	28.70 ± 0.19	12.98 ± 0.07	27.15 ± 0.02	28.74 ± 0.05	24.38 ± 0.20	28.08 ± 0.11
	NANO	37.42 ± 0.38	36.29 ± 0.13	15.51 ± 0.01	33.71 ± 0.06	36.28 ± 0.08	27.86 ± 0.02	35.73 ± 0.03
	Server	37.60 ± 1.23	30.18 ± 0.55	14.01 ± 0.07	28.55 ± 0.28	30.39 ± 0.37	29.88 ± 0.01	37.92 ± 0.04
Space Reduction		43.22 ± 0.01	40.87 ± 0.00	25.13 ± 0.00	39.94 ± 0.02	39.88 ± 0.01	39.61 ± 0.00	41.37 ± 0.01