# *CloudPAD*: Managed Anomaly Detection for ICS

Sanjeev Rao
sanjeev.rao@ucalgary.ca
Department of Computer Science
University of Calgary
Calgary, AB, Canada

Majid Ghaderi
mghaderi@ucalgary.ca
Department of Computer Science
University of Calgary
Calgary, AB, Canada

Hongwen Zhang
hongwen.zhang@wedgenetworks.com
Wedge Networks, Inc.
Calgary, AB, Canada

## ABSTRACT

Modern attacks on Industrial Control Systems (ICSs) are the result of several colliding circumstances: historically insecure communication protocols, increased ICS connectivity, and the rise of state-sponsored attackers. Extensive research has been conducted on using anomaly detection (AD) to counter this; here, deviations from an ICS's normal operation are monitored to indicate potentially dangerous situations. However, most works either assume an on-site deployment, or focus only on the neural architecture and disregard the deployment environment altogether. For the former, failure to update local AD can result in otherwise preventable attacks going undetected; as for the latter, directly porting these architectures to a cloud deployment can result in stale predictions due to communication delays, timeout-induced gaps in predictions, and surcharges due to bandwidth costs.

In this work, we present *CloudPAD*, an ICS anomaly detection pipeline that accounts for the issues introduced by an off-premises deployment, which uses the *ClozeLSTM*—a neural network based on the Long Short-Term Memory (LSTM) architecture—to detect anomalies. We train and test the *ClozeLSTM* on the Secure Water Treatment (SWaT) dataset, and show that it outperforms an advanced attention baseline, with a precision-recall AUC of 0.797 vs. 0.717. We also discuss measures to minimize *CloudPAD*'s bandwidth consumption, and show that performance remains competitive with a maximum decrease in PR AUC by 0.01 when running in this mode.

## CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; • **Computer systems organization** → *Sensors and actuators*; • **Networks** → *Cloud computing*.

## KEYWORDS

ICS, anomaly detection, cloud computing, deep learning

**ACM Reference Format:**
Sanjeev Rao, Majid Ghaderi, and Hongwen Zhang. 2022. *CloudPAD*: Managed Anomaly Detection for ICS. In *Proceedings of Workshop on CPS&IoT Security and Privacy (CPSIoTSec 2022).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Industrial Control Systems (ICSs) have historically run on insecure protocols [1]. At the same time, their increasing connectivity to the Internet makes them a prime target for attackers, such as state-sponsored Advanced Persistent Threat (APT) groups [19]. This is reflected in the steadily increasing [16] rate of security incidents since Stuxnet [6] was discovered. Extensive research has been carried out into using *anomaly detection* (AD) as a defence against novel attacks, which operates on the basis that anomalies inherently follow a different distribution than an ICS's normal operation.

The widespread success of machine learning methods in areas such as computer vision and natural language processing (NLP) has led to its adoption in anomaly detection as well [4]. Such methods do not require domain experts to manually specify ICS behaviour, and can therefore generalize across various ICSs in a rapid manner. Recent trends indicate the use of larger, stronger neural networks [15], which promise better accuracy and fewer false alarms. However, such models can be difficult to train, requiring dedicated staff for this purpose; as network sizes increase, so do hardware requirements and training times [10]. Similar logistical, operational and bureaucratic constraints may also prevent the use of newer models, resulting in otherwise preventable attacks remaining undetected.

Cloud computing is a potential panacea: here, economies of scale can be exploited by teams of experts to deliver efficient, up-to-date AD as a *managed service*, promising reduced operational complexity, improved performance, faster turnaround times and the standardization of AD across several classes of ICS. However, existing works on AD for ICS (e.g. [18], [28]) either ignore the deployment model, or focus on a locally-running neural network. A direct cloud deployment of the former (which ignores the challenges introduced by such a deployment) can cause growing pains for the customer: ICS-AD communication delays can result in predictions being stale upon arrival, especially for high-frequency ICSs operating on scales faster than a request round-trip time (RTT). Timeouts due to lost packets can cause unexpected gaps in prediction results. Lastly, constantly querying the AD system can result in unexpectedly high operational costs due to excessive network bandwidth usage.

To address these issues, we propose a *Cloud Pipeline for Anomaly Detection* (*CloudPAD*) in ICS. *CloudPAD* relies on an anomaly *detection engine* controlled by a third-party provider. This is a deep neural network which predicts future ICS states, and compensates for network-induced delays by using *lookahead prediction*. Any neural architecture can be used in *CloudPAD*'s detection engine; due to inadequacies with conventional neural architectures for the aforementioned issues, we develop the *ClozeLSTM*, which predicts future ICS behaviour to detect anomalies at the physical process level [11]. By operating as a denoiser on feature-masked input data (in a process we refer to as *clozing*), we show that it outperforms an advanced attention baseline model [20]. To reduce operational costs, we also discuss different bandwidth minimization measures for *CloudPAD*. Since there is a trade-off between network costs and detection accuracy, we also quantify the performance impact when running *CloudPAD* in reduced-bandwidth mode.

Our main contributions in this paper are as follows:

- We present *CloudPAD*, a partially remote AD pipeline for ICS, and the *ClozeLSTM*, a deep neural network that detects anomalies therein. We train and evaluate the *ClozeLSTM* on the Secure Water Treatment dataset [12], and compare it with a recently-proposed attention baseline [20].
- We discuss how *CloudPAD* compensates for communication delays arising from a cloud deployment, and examine different means of reducing bandwidth consumption to minimize operational costs.
- We show that the *ClozeLSTM* outperforms the baseline when compensating for end-to-end delays, with a precision-recall AUC score of 0.797 vs 0.717; we show that the *ClozeLSTM*'s predictive power in reduced-bandwidth mode remains competitive, with ROC and PR AUCs decreasing by a maximum of 0.02 and 0.01 points respectively.

We cover related works in Section 2, and present both *CloudPAD* and the *ClozeLSTM* in Section 3. Network effects and countermeasures are detailed in Section 4. Experiments to determine the *ClozeLSTM*'s performance are detailed in Section 5, with their results being presented in Section 6. Section 7 concludes the paper.

## 2 RELATED WORKS

We focus mainly on works using semi-supervised learning, since supervised learning methods (e.g. [9]) can have trouble detecting novel attacks [25]. Of these, *reconstruction-based* methods involve a neural network lossily compressing ICS state data into a low-dimensional subspace [30] and reconstructing it; a poor reconstruction implies an anomaly. A common architecture, as done in [21] and [23], involves using undercomplete autoencoders as they are naturally suited to this task [14]. However, such methods can only be used to detect anomalies that have already occurred, and hence are unsuitable for realtime detection in remote deployments, where communication delays need to be accounted for.

Prediction-based methods address this need by predicting an ICS's future state based on its past behaviour; anomalies are detected when the ICS's state deviates from that predicted by the network. Typically, Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) [17] networks are used, as in [18] and [13]. Having found success in the field of NLP, RNNs are well-suited to handle ordered data; with LSTMs being resistant to vanishing gradients [17], they are adept at learning the long, complex patterns representing an ICS's physical processes.

Prediction accuracy, and thus the anomaly detection ability of an LSTM, can be improved in several ways: a *stacked* LSTM uses multiple LSTM layers; this is done in [7]. Another is to use a sequence-to-sequence (seq2seq) architecture; here, two LSTMs are chained to generate predictions. This is done in [8] as well as in [20], which also uses a modified attention layer [2] to increase prediction accuracy on the SWaT dataset [12].

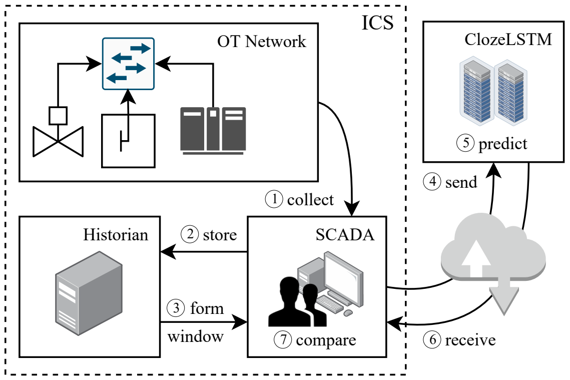

**Figure 1: Control flow of *CloudPAD*.**

The aforementioned works in the literature largely focus on an on-premises setup, or ignore the deployment model. Thus, *next-step prediction* is typically used, since it offers the best prediction accuracy. Here, only the immediately upcoming state is generated; this contrasts with *lookahead prediction*, where a distant future state is predicted. A notable exception is the LSTM architecture in [29], wherein an improvement is claimed when predicting the 150th state in the future, indicating that certain architectures are potentially more suited to lookahead prediction than others.

By focusing on the impact of network effects (such as delay and bandwidth costs) arising from a cloud deployment, *CloudPAD* can be used as a template for deploying the aforementioned works to the cloud. However, as we will show in the following sections, an ideal cloud deployment would use a neural network that is optimized for lookahead prediction (i.e. the *ClozeLSTM*).

## 3 *CLOUDPAD* DESIGN

This section describes both *CloudPAD* and the *ClozeLSTM*, which is the neural network used by *CloudPAD* to detect anomalies.

### 3.1 *CloudPAD* Architecture

We define an ICS AD pipeline as containing a *detection engine*, a *state comparator* and a *threshold*. Given a set of real ICS states, the detection engine produces a synthetic ICS state reflecting the system dynamics learned during training. The state comparator determines the difference between this synthetic state and the actual ICS state at a given point in time as an anomaly score; finally, this is compared with a threshold, above which an anomaly is detected.

*Control Flow.* A control flow diagram is depicted in Figure 1. First, the ICS's state is collected from the Operational Technology (OT) network by the historian software on the SCADA (step 1), and stored in the historian database (step 2). The most recent states are retrieved (step 3), compressed and sent as a JSON REST request to the detection engine (step 4) along with the desired timestep (i.e. a discrete time point). Inference is performed on these states by the detection engine (step 5). The resulting prediction is returned as a response (step 6), and is compared with the ICS state at the corresponding timestep (step 7) by the state comparator on the SCADA; an anomaly is detected if the difference exceeds a threshold.

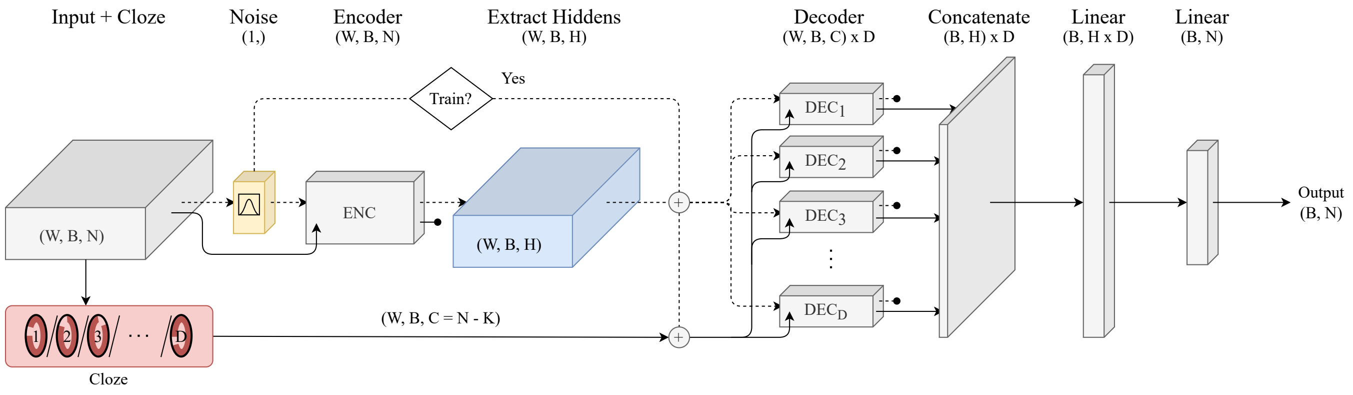

**Figure 2: *ClozeLSTM* architecture. Clozing (in red) involves selectively masking input to each decoder LSTM ($DEC_i$). Corruption (in yellow) involves adding noise to the input and hidden states, which makes the *ClozeLSTM* more resilient to false positives.**

*Detection Engine.* The detection engine of *CloudPAD* is situated off-premises, unlike its state comparator and threshold. While *CloudPAD* is agnostic of the detection engine used, we use the *ClozeLSTM*, which is based on the LSTM architecture.

*State Comparator.* In *CloudPAD*, the metric used to compare ICS states is the mean squared error (MSE) loss function. The *ClozeLSTM* is trained to return predictions with a lookahead; therefore, states are compared by matching predictions with the actual system state at the prediction's intended (future) timestep. By using a local buffer to store predictions until their intended timestep, there are no transmission-related delays, thereby allowing anomaly scores to be calculated in real-time.

*Thresholding.* In *CloudPAD*, anomaly scores are compared with a static threshold, since more complex thresholds assume anomaly scores follow certain behaviours—thereby potentially leading to anomalies passing undetected by subverting these assumptions.

## 3.2 *ClozeLSTM* Architecture

The *ClozeLSTM* is a neural network based on the stateless sequence-to-sequence (seq2seq) LSTM architecture. Unlike a regular seq2seq LSTM—where 1 encoder is connected to 1 decoder—a single encoder here is connected to multiple LSTMs in the decoder. The number of decoders ($D$) depends on the number of features (i.e. sensors/actuators) $N$ in the input and the number of features per decoder ($K$), such that $D = \lfloor \frac{N}{K} \rfloor$. The encoder and decoders both take as input a *window*, which is a set of $W$ samples corresponding to the past $W$ consecutive ICS states. Since the *ClozeLSTM* is stateless, $W$ depends on an ICS's frequency, being sufficiently large to capture system dynamics and yet small enough to allow for quick training.

Furthermore, the $i^{\text{th}}$ feature for the $i^{\text{th}}$ decoder is masked in a process we refer to as *clozing*. The feature is thus deduced based on other features, diminishing the impact of autocorrelations on the output; hence, attacks on a sensor/actuator require modifying others to remain hidden. Overall, each decoder takes as input a tensor of size $(W, B, C = N - K)$, and the encoder a tensor of size $(W, B, N)$, in the (time, batch, feature) dimensions respectively; the output is a single state which is $L$ steps ahead in the future, with size $(B, N)$. To better illustrate this, an architecture diagram is shown in Figure 2.

*Corruption.* This mechanic increases the *ClozeLSTM*'s resilience to false positives; it consists of two parts: *encoder initialization*, and *pre-conditioning corruption*. In the former, the overall standard deviation $\sigma$ of the input is calculated, and an $X \sim \mathcal{N}(0, \sigma^2)$ noise is passed as input to the encoder's hidden layers. The input is also averaged over the batch dimension, and the standard deviation across the time dimension ($\sigma_t$) is calculated. The final hidden and cell states ($H_E$ and $C_E$) of the encoder's last layers are used as the initial states of each decoder; during training only, an $X_i \sim \mathcal{N}(0, \sigma^2)$ noise is added to $H_E$ and set as the $i^{\text{th}}$ decoder's hidden state.

Preconditioning corruption involves adding an $X_i \sim \mathcal{N}(0, \sigma_t^2)$ noise to the $i^{\text{th}}$ decoder's input during training. In effect, the *ClozeLSTM* operates like a denoiser: by corrupting decoder inputs during training only, the neural network learns to ignore small fluctuations in the input during inference. As a result, noise is less likely to be treated as anomalous during inference, reducing the likelihood of false positives.

## 4 NETWORK EFFECTS

In this section, we discuss pertinent network effects encountered due to the detection engine of *CloudPAD* being located off-premises, and detail the respective countermeasures used in the context of the *ClozeLSTM*.

## 4.1 Delay Compensation

One of the main ways that *CloudPAD* handles end-to-end delays introduced by moving the server outside the bounds of the ICS is by operating the detection engine with a lookahead. Here, predictions occur multiple timesteps after the most recent input timestep. By using multiple (fallback) *ClozeLSTMs* with decreasing lookaheads, *CloudPAD* also becomes resilient to fluctuations in delay due to jitter and timeouts. For example, consider the scenario shown in Table 1. Here, an input window of the past 3 timesteps is sent to the detection engine every timestep, and the response has a lookahead of 2. A ✗ in the "Received?" row indicates the server could not

**Table 1: Correcting for packet loss by varying lookaheads.**

| Timestep | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Received? | - | - | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ |
| Without correction | - | - | - | - | 5 | 6 | ✗ | 8 | ✗ |
| With correction | - | - | - | - | 5 | 6 | 7 | 8 | 9 |

receive the request at that time; thus, when not corrected for, the response that would normally arrive for 2 timesteps in the future will instead be missing.

Correction thus involves using a neural network model with a lower lookahead when a packet is lost. Here, a model with a lookahead of 1 (i.e. next-step prediction) is used to ensure the response arrives on time; concurrently, an inference is performed for the model with a lookahead of 2, so that responses for timesteps 7 and 8 arrive in time, and the client can switch back to requesting the larger lookahead. Otherwise, any packet losses would result in a slow shift to lower lookaheads over time.

## 4.2 Bandwidth Reduction

In *CloudPAD*, the detection engine uses the most bandwidth because it requires several system states to make a prediction, unlike other components which only require the system state at a specific point in time. Since it is located outside the ICS perimeter and is controlled by a third party provider, ingress and egress costs can apply; depending on the data format, monitoring frequency and number of monitored sensors and actuators present, bandwidth consumption can quickly rise.

As a result, reducing the bandwidth consumed may be worth the impact in performance, depending on the intended use case. Different methods of bandwidth reduction are discussed below; all of them reduce the amount of data transmitted, but the way they go about doing so differs. A major caveat of such methods is that they cannot be used when substantial damage can be inflicted in the interim period between transmissions, since anomaly scores are not updated during such periods.

*Periodic sampling.* Here, samples are collected every $S$ timesteps (where $S$ is the *subsampling rate*) when forming an input window. The end result is a system that behaves in real-time for the *new*, reduced frequency; because the neural network has to be trained on this new sample rate, prediction accuracy can decrease due to the limited data available for training (e.g. from 300,000 samples at 1 s/sample to 60,000 at 5 s/sample). This is balanced by the longer context present in the input window; periodic sampling may even be necessary when dealing with high frequency inputs which would otherwise require a very large input window. This is shown in Figure 3; compared to the "Discarding" scenario, the effective window size for periodic sampling is much larger, and contains data points at regular intervals, thereby reducing the likelihood of an attack occurring in the time between samples.

*Aggregation.* Here, system states for the past $A$ timesteps are collected, compressed and sent in a single batch, rather than sending an entire window each timestep; the detection engine, rather than the client, forms windows from the input, and predictions are returned for each timestep in $A$. To ensure predictions from the

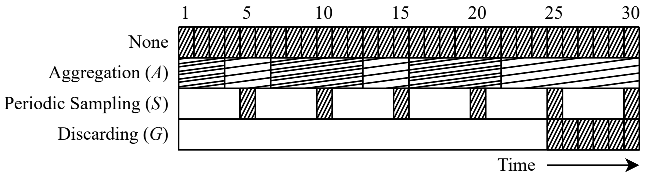

**Figure 3: Different reduction methods ($A = 9$, $S = 5$, $G = 30$). Each box represents 1 s, with a window size of 6 s used here.**

beginning of the batch are not empty, past samples are collected as well. For example, at time $T = 21$, with $W = 6$ and $A = 9$, the samples from $[7, 21]$ will be collected, before waiting another 9 timesteps to send samples from $[16, 30]$ (pictured in Figure 3). However, as $A$ increases, so does the time to detect an anomaly; furthermore, reductions hit diminishing returns as compression ratios slowly increase.

*Discarding.* This can be considered a special case of aggregation, where the main difference is that intermediate states are not collected; only the last window in a given time period $G$ is transmitted here. Like aggregation, this method is simple to implement and can be started and stopped on-the-fly; however, it is only applicable in cases where the detection engine is stateless (i.e. it does not carry forward its internal state).

## 5 EXPERIMENTAL SETUP

In this section, we describe the experiments conducted to show the end-to-end performance of *CloudPAD*. We discuss the experimental setup, including the metrics, dataset and test environment used, as well as the baseline to compare against the *ClozeLSTM*.

*Dataset.* We use the SWaT dataset [12] to evaluate our work. This is a publicly available dataset for ICS anomaly detection, comprised of operational traces of a miniaturized water treatment testbed [24] when operating both normally and under attack. Pandas [26] was used to carry out the preprocessing steps recommended in [28] on the historian data. We also removed features P102, P204, P206, P403 and P603, as their variance was 0 in the training set. After preprocessing, the dataset contains 24 features, with 496,800 samples in the training set and 449,919 in the test set.

*Baseline.* We use the neural network from [20] as a baseline, which is a seq2seq LSTM with a modified attention layer, for the following reasons: with a publicly available implementation, verification of the architecture is easy; also, it is a newer work (compared to e.g. [29]) that uses seq2seq LSTMs, thus allowing for a more direct comparison. Finally, since the attention layer is designed to improve the prediction accuracy of the LSTM, an equivalent or better result without using it indicates a viable alternative architecture.

*Parameters.* We used a fixed window size of $W = 120$ samples for the input and a maximum batch size of $B = 1208$, resulting in an input shape of $(120, 1208, 24)$ in the (time, batch, feature) dimensions respectively. The hidden size ($H$) was set to 64. Each decoder consisted of 2 layers; the encoder contained 6, with the last 2 used to initialize each decoder's hidden states. The Adam optimizer [22] was used with a learning rate of 0.002, with the features per decoder

**Table 2: Number of compensatory lookaheads (i.e. $L < 10$) used in a 24 h time period for a given link loss %.**

| Lookahead | Link Loss (%) | | | | |
|---|---|---|---|---|---|
| (L) | 0.5 | 1 | 2 | 5 | 10 |
| 1 | 0 | 0 | 0 | 0 | 9 |
| 2 | 0 | 0 | 1 | 5 | 22 |
| 4 | 1 | 1 | 1 | 3 | 44 |
| 8 | 5 | 15 | 65 | 378 | 1889 |

($K$) set to 1 to isolate individual features. Testing was done using the same lookahead ($L$) during training, with 120 timesteps as input and the $(120 + L)^{\text{th}}$ timestep as an output. Finally, the parameters for the baseline in [20] were retained: a hidden size of 64, and an input window and hint of size 90 and 9 respectively. To compare the two directly, a single attention network (rather than 1 per stage) was trained on the same preprocessed dataset and lookahead(s).

*Environment and metrics.* We used PyTorch [27] and PyTorch Lightning [5] to implement and train the neural networks on a CentOS 8 machine with an NVIDIA V100 GPU, for 80 and 240 epochs on the full and subsampled datasets respectively; Weights & Biases [3] was used to track experiments. To compare performance with the baseline, we used the area under the curve (AUC) of the precision-recall (PR) and the receiver operating characteristic (ROC) curves; these metrics were chosen as they characterize the model's performance across all possible thresholds. We also implemented *CloudPAD*'s topology (a connection between an ICS and a detection server) in Mininet with varying link parameters to quantify the impact of network effects. The following tests were conducted:

**Delay compensation.** Links between the SCADA and the remote server were simulated in Mininet with losses of 0.5%, 1%, 2%, 5% and 10%. Timeout-induced delays resulted in lower lookaheads being selected to compensate; the number of such lookaheads used in a 24 h period was measured.

**Per-lookahead performance.** The *ClozeLSTM* and the baseline were trained with lookaheads of $L = 1, 2, 4, 8, 10$ and 20, and their performance for each $L$ was measured.

**Ablation.** The *ClozeLSTM*'s performance was measured for increasing values of $K$. Since $D = \lfloor \frac{N}{K} \rfloor$ and each decoder sees $N - K$ features, performance is expected to reduce as $K$ increases, both due to fewer decoders being present, and due to the reduced context available to each decoder.

**Bandwidth reduction.** The performance of the *ClozeLSTM* was gauged when operated under each of the bandwidth reduction schemes discussed in Section 4.2.

## 6 PERFORMANCE EVALUATION

In this section, we present and discuss the results for the experiments listed in Section 5.

### 6.1 Regular Operation

*Delay compensation.* With poorer connections, lookaheads lower than the default need to be used to prevent "gaps" in the predictions from appearing, even in lower-frequency datasets such as SWaT. This is evident in Table 2; here, the default is $L = 10$, and the number of compensatory (i.e. where $L < 10$) lookaheads used is inversely proportional to the link quality. While it is worth noting that loss percentages greater than 2% are unlikely to be encountered for such services in practice, avoiding any and all compensatory lookaheads is not possible even at lower levels.

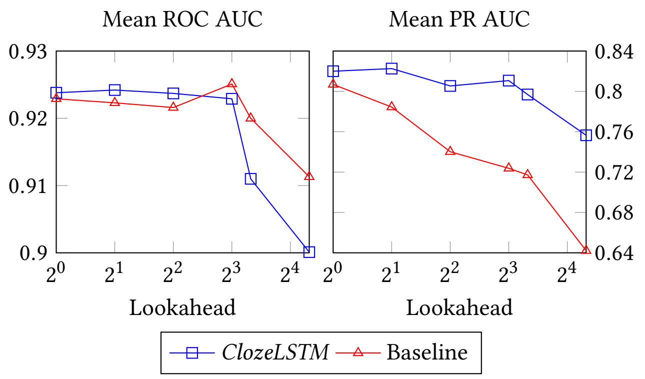

**Figure 4: Mean AUC scores for the *ClozeLSTM* and the baseline across various lookaheads ($L$). The *ClozeLSTM*'s PR AUC degrades slower than the baseline as $L$ increases.**

*Per-lookahead performance.* The ROC and PR AUCs for the baseline and the *ClozeLSTM* are shown in Figure 4. While the baseline's ROC AUC is better than the *ClozeLSTM* for lookaheads of 8 or more, the *ClozeLSTM*'s PR AUC remains consistently higher than the baseline, indicating a better ability to detect anomalies.

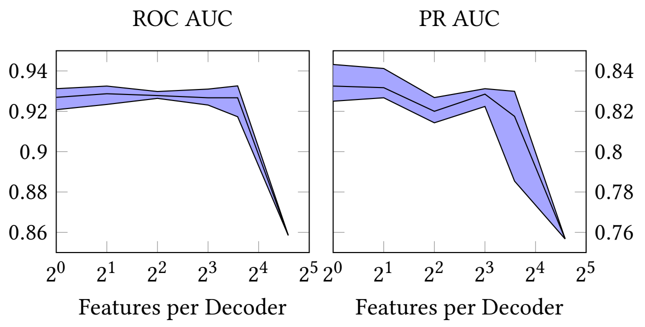

**Figure 5: Effect of increasing the features per decoder ($K$) on the *ClozeLSTM*'s performance. As decoder context decreases with increasing $K$, ROC and PR AUCs drop accordingly.**

*Ablation.* As shown in Figure 5, the PR AUC reduces with increasing $K$, as available context to each decoder decreases; on the other hand, the ROC AUC remains relatively independent of $K$. However, both PR and ROC AUCs plummet to 0.757 and 0.859 respectively when $K = N$, since every feature is clozed at this point, resulting in each decoder receiving only a tensor of noise as input.

### 6.2 Reduced Bandwidth

Use of the following methods impacts performance in different ways: they introduce a delay in the time to detection for an anomaly

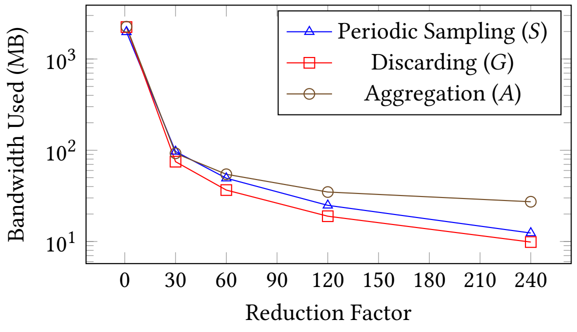

**Figure 6: Total bandwidth consumed with various reduction methods. Since discarding and periodic sampling are both lossy, these achieve larger reductions than lossless aggregation.**

and (with the exception of aggregation) also result in "reporting gaps", wherein successive predictions are separated by a period longer than that of the original dataset. Thus, when considering whether an anomaly has occurred during this period or not, a *retroactive* detection is considered; in effect, anomaly scores are applied from the time the last result was obtained.

*Overall Reduction.* Reductions in bandwidth due to the aforementioned methods are shown in Figure 6. Periodic sampling and discarding both transmit the same number of samples (see Figure 3) and thus achieve similar reductions, with differences arising primarily due to compression efficiency. Aggregation also achieves large reductions initially as groups of samples, rather than windows, are sent; however, this eventually levels off once the number of samples in a window exceeds $W$.

*Periodic Sampling.* The AUC of the *ClozeLSTM* under various subsampling schemes is shown in Figure 7. Unlike regular operation where the *ClozeLSTM* runs with a lookahead, next-step prediction is used here instead, as it is equivalent to a lookahead of $S$. As $S$ increases, so does the time taken to form the first window; as a result, any attacks that occur within this period are disregarded.

*Aggregation.* Results for this method are similar to those for the *ClozeLSTM* in Figure 4, since all the predictions generated by the detection engine are returned for the samples collected during the period $A$. The main difference between regular and aggregated prediction is the time to detect an attack. On average, the latter will be delayed by $\frac{A-L}{2}$ timesteps more than the former, since all samples in $A$ must be collected before it can be sent for analysis; only attacks within the lookahead period $L$ are detected instantly. In contrast, a window is sent every timestep in regular operation, thus allowing for immediate detection.

*Discarding.* The *ClozeLSTM*'s AUC for various discard periods $G$ is shown in Figure 8; a performance *increase* is noted, unlike previous methods. This is because applying the same anomaly score for the entire discard period is equivalent to adding a processing step such as a *rolling sum* to the state comparator, which increases the effective contrast between anomalous and normal operation. This effect starts to wear off for longer periods of $G > 240$ (not shown here), as the likelihood of false positives and false negatives increase due to anomaly scores for normal and anomalous operation increasingly overlapping.

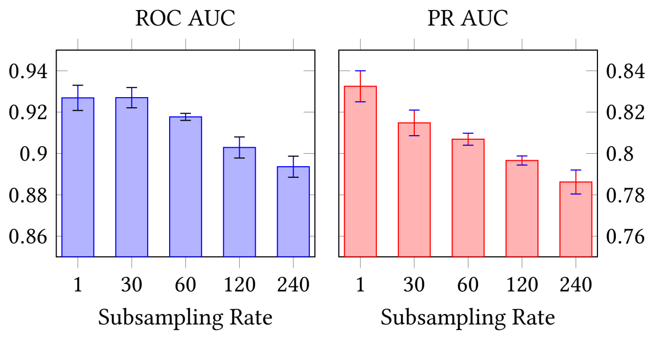

**Figure 7: Effect of different subsampling rates ($S$) on the *ClozeLSTM*'s performance. ROC and PR AUCs decrease as $S$ increases, due to fewer training samples being present and correlations across time becoming increasingly sparse.**

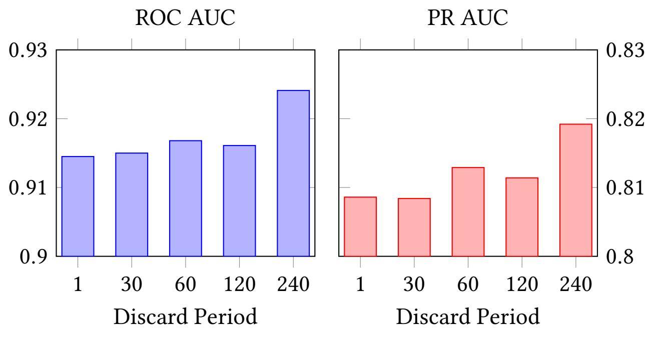

**Figure 8: Effect of different discard periods ($G$) on the *ClozeLSTM*'s performance. AUCs increase since the impact of erroneous predictions are diminished.**

## 7 CONCLUSION

A swath of benefits can be realized by taking advantage of the cloud for ICS anomaly detection, such as quick turnaround times, improved prediction accuracy from state-of-the-art solutions, and its standardization across several types of ICS. To this end, we specify a means for cloud AD in *CloudPAD*, and develop the *ClozeLSTM*, a neural network that outperforms an advanced attention baseline, with a precision-recall (PR) AUC of 0.797 vs. 0.717 in this environment. We also show how *CloudPAD* can minimize its bandwidth consumption; performance remains competitive in this mode, with a maximum decrease in PR AUC by 0.011 points.

## ACKNOWLEDGMENTS

This work was supported by Wedge Networks Inc., Alberta Innovates and the Natural Sciences and Engineering Research Council of Canada.

## REFERENCES

[1] Simon Duque Anton, Daniel Fraunholz, Christoph Lipps, Frederic Pohl, Marc Zimmermann, and Hans D Schotten. 2017. Two decades of SCADA exploitation: A brief history. In *2017 IEEE Conf. Appl. Inf. Netw. Secur. (AINS)*. IEEE, 98–104.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. (2014). arXiv:arXiv:1409.0473

[3] Lukas Biewald. 2020. *Experiment Tracking with Weights and Biases*. https://www.wandb.com/

[4] Kukjin Choi, Jihun Yi, Changhwa Park, and Sungroh Yoon. 2021. Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines. *IEEE Access* 9 (2021), 120043–120065. https://doi.org/10.1109/ACCESS.2021.3107975

[5] William Falcon and The PyTorch Lightning team. 2019. *PyTorch Lightning*. https://doi.org/10.5281/zenodo.3828935

[6] Nicolas Falliere, Liam O Murchu, and Eric Chien. 2011. W32. stuxnet dossier. *White paper, symantec corp., security response* 5, 6 (2011), 29.

[7] Cheng Feng, Tingting Li, and Deeph Chana. 2017. Multi-level anomaly detection in industrial control systems via package signatures and LSTM networks. In *2017 47th Annu. IEEE/IFIP Int. Conf. on Dependable Syst. Netw. (DSN)*. IEEE, 261–272.

[8] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. 2016. Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model. (2016). arXiv:arXiv:1612.06676

[9] MR Gauthama Raman, Nivethitha Somu, and Aditya P Mathur. 2019. Anomaly detection in critical infrastructure using probabilistic neural network. In *Int. Conf. Appl. and Techn. in Inf. Secur.* Springer, 129–141.

[10] Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. 2021. *AI and Memory Wall*. https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

[11] Jairo Giraldo, David Urbina, Alvaro Cardenas, Junia Valente, Mustafa Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. 2018. A Survey of Physics-Based Attack Detection in Cyber-Physical Systems. *ACM Comput. Surv.* 51, 4, Article 76 (jul 2018), 36 pages. https://doi.org/10.1145/3203245

[12] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. 2016. A dataset to support research in the design of secure water treatment systems. In *Int. Conf. on Crit. Inf. Infrastructures Secur.* Springer, 88–99.

[13] Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. 2017. Anomaly Detection in Cyber Physical Systems Using Recurrent Neural Networks. In *2017 IEEE 18th Int. Symp. High Assurance Syst. Eng. (HASE)*. 140–145. https://doi.org/10.1109/HASE.2017.36

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[15] Will Douglas Heaven. 2021. *2021 was the year of monster AI models*. https://www.technologyreview.com/2021/12/21/1042835/2021-was-the-year-of-monster-ai-models/

[16] Kevin E Hemsley, E Fisher, et al. 2018. *History of industrial control system cyber incidents*. Technical Report. Idaho National Lab.(INL), Idaho Falls, ID (United States).

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[18] Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M Poskitt, and Jun Sun. 2017. Anomaly detection for a water treatment system using unsupervised machine learning. In *2017 IEEE Int. Conf. Data Mining Workshops (ICDMW)*. IEEE, 1058–1065.

[19] Anastasis Keliris and Michail Maniatakos. 2017. Demystifying advanced persistent threats for industrial control systems. *Mech. Eng.* 139, 03 (2017), S13–S17.

[20] Jonguk Kim, Jeong-Han Yun, and Hyoung Chun Kim. 2019. Anomaly detection for industrial control systems using sequence-to-sequence neural networks. In *Comput. Secur.* Springer, 3–18.

[21] SungJin Kim, WooYeon Jo, and Taeshik Shon. 2020. APAD: Autoencoder-based payload anomaly detection for industrial IoE. *J. Appl. Soft Comput.* 88 (2020), 106017.

[22] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. (2014). https://doi.org/10.48550/ARXIV.1412.6980 arXiv:arXiv:1412.6980

[23] Moshe Kravchik, Battista Biggio, and Asaf Shabtai. 2021. Poisoning attacks on cyber attack detectors for industrial control systems. In *Proc. 36th Annu. ACM Symp. Appl. Comput.* 116–125.

[24] Aditya P. Mathur and Nils Ole Tippenhauer. 2016. SWaT: a water treatment testbed for research and training on ICS security. In *2016 Int. Workshop Cyber-physical Syst. Smart Water Netw. (CySWater)*. 31–36. https://doi.org/10.1109/CySWater.2016.7469060

[25] Gauthama Raman MR, Chuadhry Mujeeb Ahmed, and Aditya Mathur. 2021. Machine learning for intrusion detection in industrial control systems: challenges and lessons from experimental evaluation. *J. Cybersecur.* 4, 1 (2021), 1–12.

[26] The pandas development team. 2020. *pandas-dev/pandas: Pandas*. https://doi.org/10.5281/zenodo.3509134

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Inf. Process. Syst. 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[28] Ángel Luis Perales Gómez, Lorenzo Fernández Maimó, Alberto Huertas Celdrán, and Félix J García Clemente. 2020. MADICS: A Methodology for Anomaly Detection in Industrial Control Systems. *J. Symmetry* 12, 10 (2020), 1583.

[29] Dmitry Shalyga, Pavel Filonov, and Andrey Lavrentyev. 2018. Anomaly detection for water treatment system based on neural network with automatic architecture optimization. (2018). arXiv:arXiv:1807.07282

[30] Riccardo Taormina and Stefano Galelli. 2018. Deep-learning approach to the detection and localization of cyber-physical attacks on water distribution systems. *J. Water Resour. Planning and Manage.* 144, 10 (2018), 04018065.