Prolego: Time-Series Analysis for Predicting Failures in Complex Systems

Anwesha Das[1, 2] Alex Aiken[2]
[1]*University of Chicago* [2]*SLAC National Accelerator Laboratory/Stanford University*

***Abstract*—Failures in large, complex systems can be difficult to diagnose and expensive for both the system maintainers and users. We present techniques for predicting failures when there is sparse ground truth in sensor logs and usage modes of the system evolve rapidly. We demonstrate our methods using real-world logs from three different systems. Our method achieves over 80% prediction accuracy for various amounts of lead time.**

***Index Terms*—Time-Series, Deep Learning, Failures**

I. Introduction

We consider the problem of predicting failures from monitored measurements in complex heterogeneous systems. Increasing scale and diversity of components in systems lead to heterogeneity and complex behaviour. Diagnosing faulty subsystems is difficult for operators because of the high dimensional space of parameters and the need to minimize recovery time. The time difference between a failure manifestation and when an imminent failure is flagged is the *lead time*. As with any system [23], the ability to predict failures with some lead time can benefit users and operators by at least providing timely warnings and in the best case allowing action to be taken to avoid complete system failure. We present *Prolego* (i.e., *predict* or *forecast* in Greek), an end-to-end methodology for making such predictions using minimal domain knowledge.

While log-based failure prediction is not a new problem [39, 55], our main contributions deal with three significant difficulties that have not been adequately addressed in the literature. First, the instability of a highly dynamical system leads to frequent changes in its normal behaviour [20], which is not an unusual feature of complex systems (e.g., consider the power grid at different times of day and in different weather conditions). Consequently, despite the availability of many signals over a long duration, representative data available for a specific usage pattern is limited for predictive modeling. Our solution is to use frequent retraining on recent windows of data, and we show that with parallel, distributed computations we can achieve execution times for prediction that are short enough to be useful in practice.

Second, be it computing platforms or cyberphysical systems (CPS) and industrial control systems (ICS), there exist scenarios where production failures for a specific system configuration are relatively few. Also, for some systems, the operator logged reports help to know the approximate parts of the day when failures happened, not the exact failure times or the specific faulty signals that cause a failure, giving rise to insufficient ground truth. Moreover, automated distinction of unintended versus intended anomalous conditions (e.g., due to manual tuning or planned maintenance) from the data is difficult as they often require assistance from human-in-the-loop experts [30]. Given this situation predicting failures in an unsupervised manner is non-trivial. Prolego automates finer label generation and uses history data judiciously to enhance prediction.

Third, forecasting failures with signals comprising of irregularities some of which do not have temporal trends is not always feasible. Some CPS-centric solutions adopt simpler fault models involving manual fault injection or fewer signals [35, 40], while cloud-centric solutions have relatively fewer irregularities in signals [57]. Prolego examines density of time-series considering signals from the entire system to achieve useful lead times. Our study has the potential to enable better autonomous control [13] in complex platforms that experience various anomalous conditions.

Contributions: We make the following contributions:
1. Prolego first uses a small number of signals to automatically label additional failures using a deep learning model from the sparse ground truth, increasing our ground truth dataset.
2. A simple ranking scheme based on coefficient of variation is used to determine which signals are most likely to be predictive of failure. This ranking is done for multiple short time windows to capture the different time-series patterns and a set of the highest ranked signals is selected from these trials.
3. Prolego uses the selected signals to design a predictive model that forecasts failures and estimates accuracy.
4. Prolego is evaluated on real-world datasets from three diverse systems. Prolego achieves as high as 86.2% accuracy, has improvements over 15% with baseline comparisons, and obtains as much as 8-hours of lead time.
5. Prolego shows opportunities for lead time optimization using a state-of-the-art programming model that improves feature selection time from a high dimensional space.

II. Background

Most systems archive logs that include monitored sensor measurements and other derived parameters. A signal or parameter is a physical quantity (e.g., voltage, current) of a subsystem, with a specific unit of measurement. A signal is a univariate time-series consisting of *(timestamp, value)* pairs, that could as well include optional alarm-related fields such as *status* and *severity*. The number of signals account for the parameter space. Failures are unexpected anomalous conditions caused by various subsystems malfunctioning, requiring human intervention to recover. Partial non-catastrophic failures or degraded performance [61] that do not require

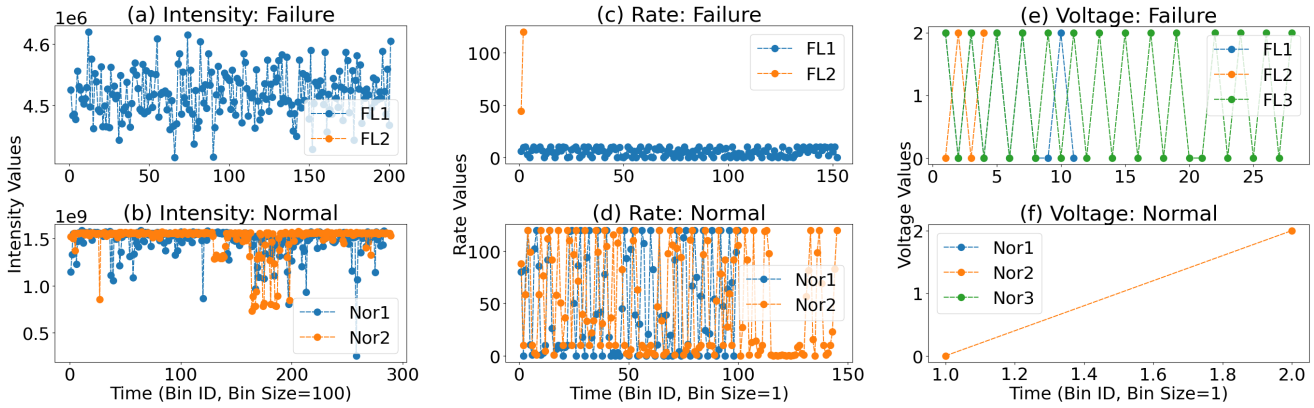

Fig. 1: *The figures show variations of 3 signals (Intensity, Rate, and Voltage), during different failure (FL) and normal (Nor) windows, marked by FL1, FL2, FL3, Nor1, Nor2, and Nor3, respectively, as discussed in Section II*

manual intervention for recovery is beyond the scope of this work. For any specific time period, if a signal undergoes substantial deviations in its measurements compared to what is generally observed during healthy operational states, the signal is considered anomalous. Normal times typically refer to times of healthy operating conditions with no reported failures. However, times with reported failures can have signals with anomalous characteristics linked to some defects. Normal times can also be intentional system *off* periods when user jobs complete or operators deliberately shut down the system.

Challenges: While logs produce a wealth of data, this data has limitations that make it difficult to directly apply data mining methods. Several systems [28, 37, 56] face similar hurdles, where threshold-based alarms do not suffice for maintaining scheduled uptime. Generally, class imbalance and noise across signals in a high dimensional space make differentiation of anomalous trends through automated learning models difficult. Some of the other challenges are:

• **Sparse Ground Truth**: Manual labeling is cumbersome in most facilities [37, 57]. The exact start and end times of many anomalies are unspecified, nor are they documented by the operators post-recovery, apart from anomalies injected in a controlled environment [26]. The corresponding coarse-level time-window of a failure is noted; i.e., for 2 hours of failure in a 8-hour window, one needs to decipher approximately which 2 hours (out of 8) correspond to failure via statistical learning.

Fig. 1c shows 2 windows with 4 to 5 hours of failure marked by absence of data compared to normal times in Fig. 1d. In these figures, the y-axis is the average signal value and the x-axis is the bin number, where each time-series is split into a number of bins and the average of the observations within each bin is plotted; *FL* and *Nor* indicate failure and normal times, respectively. Sparse ground truth with no expert curated data makes failure prediction with high accuracy non-trivial. Prolego circumvents this sparsity by creating finer labels for ground truth classification (§ Sec. IV-A). Recent work on labeling [45, 57, 58] relies on engineers to label an anomaly, uses known labels to create new labels assuming different source and target domains, or employs positive-unlabeled learning with partial labels for anomaly detection. In contrast, we use a data-driven method to generate shorter fine-grained labels from longer coarse-grained labels for the same domain.

• **Irregularities**: Besides the presence of missing values, the sampling rate varies across signals giving rise to unsynchronized data. The range of sampling intervals can be large (e.g., 1 sec to 30 mins), which requires handling the sparse irregular (unevenly spaced) and dense regular time-series together at suitable time-granularity for multivariate analysis. Certain signals are logged at `regular` rate (i.e., at regular intervals), some only when there is any `change`, while others only if there is a `significant` change of the sensed measurement. This storage efficient logging gives rise to skipped measurements that need not indicate faults. A device trip can result in missing values in its related signal, while other signals can have missing data for having no change in their values from past observations.

Figures 1b and 1d show raw data during normal times for two signals of a particle accelerator [36], namely, *beam intensity* (regular) and, *beam rate* (irregular), respectively. Since *intensity* is a dense signal, each bin has 100 observations, while the sparse *rate* signal has a bin size of 1. Figs. 1a and 1c versus 1b and 1d clearly show the smaller amount or lower values of data seen during failure times. Similar traits are discernible during certain normal times as well (e.g., window Nor2 in Fig. 1d has values close to 0 for bin IDs 112 to 125) related to intentional shutdown periods or short-lived signal drops (implying degraded performance but not failure). Moreover, signals associated with no faults can show non-stationary trends similar to anomalies. Figs. 1e and 1f show one such signal related to voltage supply. Voltage power trips lead to failure resulting in missing data (Fig. 1e), yet during normal times this signal did not contain too many data points either because this device was turned off or there were no changes in observations (Fig. 1f). Certain anomalous traits can lead to an unsteady system instead of a failure, resulting in false positives during prediction. Prolego examines signal sparsity to impute time-series (e.g., backfill-based upsampling), whenever applicable, and selects signals from multiple subspaces for model training to address such irregularities (§ Sec. IV-B).

• **Unstable System States**: Even during healthy conditions defining normal behaviour is difficult. A set of signal patterns

implying a stable operating condition for a specific timeframe can stop being indicative of normalcy for another timeframe when dissimilar applications with different configurations are run on a system. Prolego uses frequent retraining of a subset of signals chosen from recent history for prediction (§ Sec. IV-C).

• **Failure Duration**: The statistics of failure instances vary across systems. The downtime or failure duration can range between a few minutes to several hours. The duration of anomalous conditions influence predictability and the achievable lead time. Often short-lived anomalies causing brief signal fluctuations appear similar to normal patterns. This makes abrupt short-term failures hard to isolate during model training. Prolego inspects different sample sizes to accommodate this diverse duration of failures (§ Sec. V).

III. Related Work

We briefly survey the extensive literature on log analysis and anomaly detection methods.

Time-Series Analysis: Time-series logs have been studied for anomaly detection [18, 21, 46, 51, 59], forecasting, imputation [15, 41] and root cause analysis [24]. Some outlier detection algorithms [12, 60] propose noise robust heuristics on synthetic datasets. Several studies [44, 56] use some form of RNN or CNN (Recurrent/Convolutional Neural Network) model to diagnose anomalies. In contrast to these methods we perform long-term forecasting on real production logs for practical usage to design our predictor.

Fault Studies with System Logs: Prior solutions analyze system faults and performance variations [47] for cyberphysical systems [35], data centers [29], and supercomputers using monitored performance (e.g., resource usage) and system logs. Studies using cross-correlations [39], Principal Component Analysis (PCA) [38], clustering [32, 45], or decision trees generally require some feature engineering of the data to be scalable compared to our ranking-based analysis. Studies on anomaly detection [14, 37], root cause analysis [33, 53], prediction (e.g., service outage, storage error, workload performance [16, 17, 55]), and application error diagnosis [54] use methods such as hierarchical clustering, Bayesian networks, random forests (RanFor), AdaBoost, autocorrelation, longest common subsequence, item-set mining, graphical models, deep learning, and sequential state switching. The main difference between these solutions and our work is in making use of limited ground truth for prediction in a high dimensional and irregular data space with minimal supervision.

Complex Systems: Statistical methods and Machine Learning (ML) have been used in physical systems for subsystem tuning, alignment, optics correction [22], radio-frequency (RF) fault diagnosis [36], magnet monitoring, anomaly detection [10], and event identification [23, 31]. Often used methods or numerical measures are either supervised or not scalable with a large number of parameters (e.g., support vector machine (SVM), local outlier factor, Bayesian optimization, Monte-Carlo estimation, correlation coefficients, dynamic time warping [9]). Statistical models requiring less data are more popular; neural networks are less explored. Some studies employ subsystem-specific signals to assess a certain fault type or perform detection and classification [48] instead of prediction. In contrast to these, we do not pre-select parameters via manual engineering efforts [10] or expert guidance (e.g., RF, Beam Loss Monitor signals [36]) and use unlabeled continuous time-series data as opposed to data that is discrete, labeled, simulated or obtained through controlled system conditions [49].

IV. Prolego Design

Figure 2 summarizes the three parts of Prolego. First, Prolego builds a label generator to create fine-grained labels from low-quality sparse ground truth based on an autoencoder model using a small number of signals. Second, Prolego chooses a subset of available signals based on temporal variations, studies inter-signal correlations, and classifies signals for resampling, to produce a transformed dataset suitable for forecasting. Third, Prolego develops a predictor using an autoregressive model and evaluates accuracy for different lead times. Finally, scalability experiments are conducted (Optimizer) to assess potential speedups in feature selection to improve lead times.

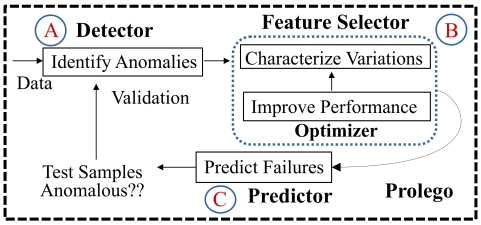

Fig. 2: Methodology

A. Label Generation

For systems that lack precise labels, having a method to generate labels [57, 58] from the available imprecise coarse ground truth can benefit model validation. For certain injected faults or application-level failures [54], the anomalous times may be relatively precise. However, there are systems with insufficient information about the onset and termination of failures, where imprecise failure windows are leveraged for anomaly detection [52]. Such systems benefit from a data-driven model to identify failure times other than the fixed coarse-level ground truth, to validate inferred results with reference to recorded failures during subsequent predictions.

Failure Detection (§ Algo. 1) summarizes the general idea of label generation. A few high-level signals indicative of system performance are identified and combined to form a unified representative signal (M_{Rep}), followed by a split into failure and normal times based on the known coarse-grained labels. These times are further split into windows smaller than the coarse labels (Win), if needed. Sample errors and dynamic thresholds are estimated during model training (normal times) and testing (failure times). For known ground truth, accuracy is computed, else for unlabeled timeframes new labels are generated.

One or more reliable signals of a system often reflect the overall system performance; Prolego uses them to create fine-grained labels. Forming a representative signal (we use normalized dot product) may not be always feasible in case

of subsystem fault modeling or partial failure analysis [61]. In our context of anomalous incidents leading to crashes or halts affecting the high-level system behaviour, aggregating indicator signals [38] helps to assess system health. The combined signal strength is used to train a detector that infers failure and normal times for time-granularity lower than the available ground truth.

Algorithm 1 Detection and Label Generation

Require: Coarse Ground Truth, System Logs
Ensure: Finer Labeled Samples ▷ Accuracy, Generated Labels
1: **procedure** FAILURE DETECTION(Coarse Labels, System Logs)
2: M_{Rep} ← Representative signal for a target system
3: $[M_{Rep}^{Anomaly}, M_{Rep}^{Normal}]$ ← Coarse Labels
4: $[M_{Win}^{Anom}, M_{Win}^{Normal}]$ ← Win ▷ |Windows| < Coarse Labels
5: $\text{Thresh}_{\text{train}} \leftarrow \text{Model}_{\text{detect}}(M_{Rep}^{Normal} \| M_{Win}^{Normal})$ ▷ Training
6: $\text{Pred}_{\text{Test}} \leftarrow \text{Model}_{\text{detect}}(M_{Rep}^{Anomaly} \| M_{Win}^{Anom})$ ▷ Testing
7: S_{Error} ← Test Sample Error ($\text{Orig}_{\text{Test}}$, $\text{Pred}_{\text{Test}}$)
8: $\text{Acc}_{\text{detect}}$ ‖ New Labels $[L_i]$ ← [(S_{Error}>$\text{Thresh}_{\text{train}}$)?, Validate with Ground Truth] ▷ Detection accuracy or new labels
9: $\text{Label}_{\text{Correct}}$← Manual Validation([$L_i$], Failure Durations)

10: **procedure** TEST SAMPLE ERROR($\text{Orig}_{\text{Test}}$, $\text{Pred}_{\text{Test}}$)
11: $\text{Orig}_{\text{Test}}$=[..$ts_j$..], $\text{Pred}_{\text{Test}}$=[..$ts_{j'}$..] ▷ Test values
12: $F_{\text{Val}} \leftarrow \text{Opt}_{[ts_j \in \text{Orig}_{\text{Test}}]}$ ▷ Anomalous Values
13: **for** ($ts_j \in \text{Orig}_{\text{Test}}$) and ($ts_{j'} \in \text{Pred}_{\text{Test}}$) **do**
14: **if** ($ts_j \leftarrow F_{\text{Val}}$) and ($ts_{j'} > ts_j$) **then** ▷ Penalize
15: $\text{error}_j \leftarrow (||ts_{j'} - ts_j||) * P_f$ ▷ $1 < P_f \leq N$
16: **else**
17: $\text{error}_j \leftarrow (||ts_{j'} - ts_j||)$
18: $S_{\text{Error}} \leftarrow \text{Mean}\ (\sum \text{error}_j)$
19: **return** S_{Error} ▷ Assess Failure

We observe that failure times are usually characterized by anomalous values such as zeroes or magnitudes higher or lower than normal times, i.e., dips and spikes. Such traits are common in a variety of systems (e.g., burst of disk swapping implying a crippled database server [38], or decreased storage signal strength due to a data stream outage [16]). Based on this observation a reconstruction error function is formulated to identify failure times. *Test sample error* (#10, § Algo. 1) illustrates the basic strategy of error computation between the true and predicted values of a specific test sample. Anomalous values are identified first (F_{Val}). If the original value is an anomalous value observed during failures (e.g., 0 or max value), and the predicted value deviates from the anomalous value, the penalty is P_f times higher as opposed to just the absolute difference, to indicate times with poorer system performance (generally P_f ≈1.5 to 1.9 works).

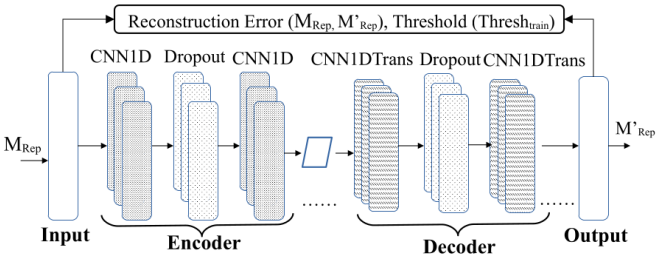

Fig. 3: Detector Model

We use a reconstruction error model where the ability to reconstruct a test sample is expected to be poorer if the test data looks too different from the training data. The intuition is to penalize instances during model testing, when predicted values deviate significantly from the original, guided by the statistical nature of failures. Figure 3 gives an overview of our detector, which is a sequential model with 3 layers each of 1D-CNN and 1D-CNN Transpose with dropouts, serving as the encoder and decoder respectively, using the *Adam* optimizer, and *mse* loss function. As an autoencoder model helps to assess the reconstruction loss, we employ an encoder-decoder architecture. Dropout layers help in regularization and better training.

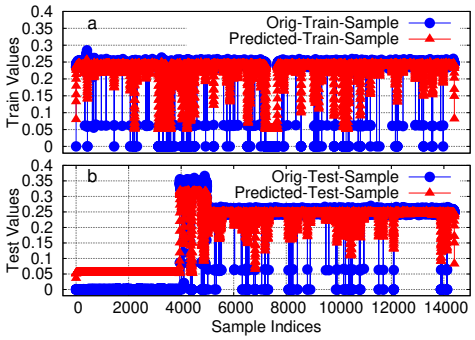

Fig. 4: Train and Test Sample

Figure 4 shows the true and predicted values corresponding to a specific train (4a) and test sample (4b) of size 8 hours, respectively. As seen, the predicted values are not too distant from their true values. For minimum values in the true set their corresponding predicted values clearly deviate with magnitude greater than the minimum. Based on this observation we define the reconstruction error metric for test sample score calculation to distinguish anomalous times from normal.

B. Feature Selection

As with many ML-based methods [31, 58] suitable feature selection from a large parameter space helps to discard signals that are noisy, redundant or irrelevant for prediction. To decide which features to use and how many, Prolego uses *Coefficient of Variation* or COV for signal ranking. COV is the ratio of standard deviation to mean of the specified data. This choice is guided by the following factors:

1. COV is a dimensionless quantity that captures time-series dispersion irrespective of the dissimilarity in signal scales and measurement units. In feature selection, wrapper (e.g., SVM [43]) methods known to be computationally expensive integrate feature selection and learning together. The COV-based filter is algorithmically simple separating feature selection from the predictor for flexible model updates. Many dimensionality reduction methods (e.g., PCA [60], RanFor [37]) require standardized data or target variables, some of which are more suitable for regular or stationary time-series, compared to irregular signals. Also, COV has better interpretability over embedded methods (e.g., regularization [34] to pick variables during training). Analyzing higher-ranked signals over different time intervals helps to identify potential cause and effect relationships across signals.
2. Time-series can be retained without any transformation (e.g., interpolation) saving sampling overheads and noise-induced

effects on data. An empty signal implies either it is turned off or has no fluctuation compared to signals undergoing drifts (slowly emerging variations), hence it is fine to have such signals ranked low. Signals with missing data due to faults usually contain data prior to failure, showing some variability. We notice the *mean* values to be similar to the *mode* for certain time periods; which indicates unless there is a fault, some signals do not deviate much. This makes function of *mean* a suitable choice for signal selection. Moreover, analyzing COV for multiple time-granularity shows evidence of the change in the system's usage pattern with changing applications.

3. Sparse signals can have COV scores different from their transformed (interpolated) counterparts, impacting their ranks. This does not adversely affect the outcome provided the chosen set of features used in training can differentiate between failure and normal times. Recent work on saliency [25] showing better feature selection by decoupling time-steps from features for time-series data has a similar motivation.

Algorithm 2 Feature Selection

Require: T_k *in* D_j, Tot_M ▷ Windows in Duration *j*, Signal List
Ensure: $TF^k_{[S_1 \ldots S_N]}$, M^j_{List} ▷ Shortlisted Features
1: **procedure** SELECT FEATURES(D_j, Tot_M) ▷ Feature selection
2: noisy $\leftarrow \emptyset$; $\delta \leftarrow$ difference in adjoining COV scores
3: **for each** $T_k \in D_j$ **do** ▷ $1 \leq k \leq W$; 1 to W periods
4: $[\ldots S_j \ \ldots] \leftarrow$ Rank signals based on COV
5: $[S_{denoise}] \leftarrow [S_j] \setminus$ noisy; noisy $\leftarrow \exists_{sig} \in |COV|^{high}$
6: $[S_N] \leftarrow [S_{denoise}]$, where $(\delta > \theta)$ ▷ Pick top N signals
7: $[TF^k_{[S_1 \ldots S_N]}] \leftarrow ([S_N])$ ▷ Chosen signals in T_k
8: $M^j_{List} \leftarrow TF^k_{[S_1 \ldots S_N]}$, if $|D_j| = 1$ ▷ Single window
9: **if** $|D^k_j| \geq 2$ **then** ▷ Multiple windows in D_j
10: $jointM_{D_j} \leftarrow (\cap\ [TF^k_{[S_1 \ldots S_N]}])$ ▷ Common signals
11: $disjointM_{D_j} \leftarrow [(\cup\ [TF^k_{[S_1 \ldots S_N]}]) \setminus jointM_{D_j}]$
12: getSignals $\leftarrow \forall$ Signals $\in disjointM_{D_j}$, where $(\delta > \alpha)$
13: $M^j_{List} \leftarrow (jointM_{D_j} \cup$ getSignals$)$
14: **return** M^j_{List} ▷ Chosen signals in D_j

Select Features (§ Algo. 2) shows Prolego's feature selection approach. For a specific time-window, signals are ranked in decreasing order of their COV scores. Signals with very high COV can be potential noise, hence Prolego eliminates them. A suitable signal count N is automatically determined based on the distribution of COV scores, by examining the successive drop in COV magnitudes (δ). Signals with $\delta >$ threshold (θ) are selected; very small δ indicates a relatively steady signal. For feature selection in duration D, we pick the common signals (#10) from all the sample windows (TF) in that duration. From signals uncommon across windows (#11), we limit to those whose δ is $>$ threshold (α) (#12). θ and α are adaptively derived from the spread of the COV scores. The intuition is to select parameters locally before forming an ensemble over any spectrum of time. Past studies [19] have used COV for performance analyses; we use COV to eliminate signals across multiple subspaces in a simple yet effective manner.

To determine the number of features (N) to be considered for prediction, we examine the relevance of the top-ranked signals during failures correlating them with the available ground truth. Table I illustrates a few sample cases of failures with associated signals and failure causes, for a particle accelerator system. For fan (#2) and magnet power supply faults (#4), many relevant subsystems showed up within the top 72 signals. For degraded voltage controllers leading to power trips (#6) the affected RF phase signals show similar behaviour during certain normal times as well. Most relevant signals show up before the COV drops to a certain point; Prolego utilizes this finding to bound the number of signals. For e.g., in Fig. 5, the first 3 signals of time period T_B are noisy (e.g., 1041 is an outlier). δ drops to 0.01 after the first 80 signals when the COV scores stabilize.

TABLE I. Impact of Subsystem Faults

#	Top N	Few Affected Signals	Comments on Faults
1	100	Timing, Supply Temp., LOSS, TORR, EnergyJitter	Power Supply Failure
2	72	SCR_Fan_Temp_FLT, FBCK, BRATIO	SCR Temp. Fault; Fan Assembly Unit Replaced
3	81	SCR_FAN_STATE, Reset, BVLT, DQ_V	Procedural Error, RF System Trip
4	62	Magnet-related Signals, Water-Temp, Forward Power	Magnet Power Supply Ground Faulted
5	58	Phase Control, Amplitude, Protection System	Software Fault on L2 Phase of RF System
6	69	RF-based Phase Signals, Interlock Reset, TGT Fault	Degraded Voltage Controller; Volt. power $<$ than expected

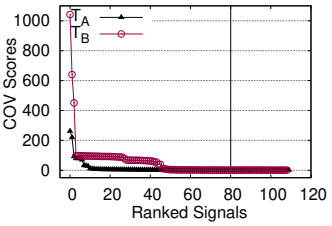

Fig. 5: Ranking

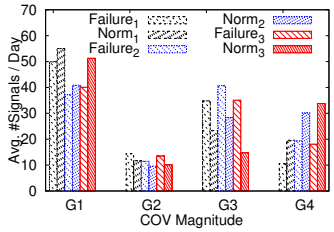

Fig. 6: Grouping (COV)

For the studied systems, we consult operator-logged comments about various subsystem faults and failures to identify signals related to or affected by the failure cause, wherever feasible. We find that some highly ranked signals relate to common subsystems that are affected no matter what subsystem faulted based on our direct interaction with domain experts. To inspect how the ranked signals relate to each other, the signals are classified based on their COV scores across 4 groups, as shown in Table II. Figure 6 shows that:

1. Often *cause-related signals* have lower COV (G1) than *affected signals* (G3). The latter show deviations for a variety of fault conditions. Failure times have similar average number of signals per day in G1 and G3. For normal times, the number of signals across G3 and G4 do not clearly stand out, implying lack of major temporal variations.

2. Signals in G2 are often those affected by the faulty subsystem. Using too few signals from a single group alone may not be helpful in capturing the system state for predicting failures.

TABLE II. Groups

Id#	COV Magnitude
G1	≤ 1
G2	> 1, and ≤ 10
G3	> 10, and ≤ 100
G4	> 100

TABLE III. Types

Signal Type	#Values/Hour
Sparse	< 500
Medium Sparse	≥ 500, and ≤ 1000
Dense	> 1000

Feature selection methods incurring high feature engineering costs [53] with minor accuracy changes often depict that no

single model consistently outperforms in a highly unstable environment. Prolego chooses signals that approximate system state suitable for inexpensive dynamic retraining.

C. Failure Prediction

To assess if there is an imminent failure, Prolego takes into account the following factors:

1. Prolego considers the top-ranked signals of training data to design the predictor, as test data may not be available during training. For the same reason, the rescaling parameters of training data are applied to the test data. Prolego does not use the well known RMSE metric (root mean squared error) [42], since using the real (i.e., original) values of future time-steps makes use of information that is not available for a scenario with incoming real-time logs.
2. The test times are strictly ahead of the training times. The test data contains both failure and normal times in increasing order of time to assess the model's efficacy.
3. As data imbalance exists, for certain datasets appending normal times to training history over a moving window helps. However, with moderate recent history (e.g., >1 week), Prolego excludes the old history (e.g., past month), to limit the range of training data within close temporal proximity.

Normal Days → [5-7, 11-16, 18, 20-23, 25-30]
Failure Days → [8, 9, 10, 17, 19, 24, 31]

TABLE IV. Train:Test Pairs

#	Train	Test
1	[5, 6, 7]	[8, 9, 10, 11, 12, 13, 14, 15, 16]
2	[11, 12, 13, 14, 15, 16]	[17, 18]
3	[11, 12, 13, 14, 15, 16, 18]	[19, 20, 21, 22, 23]
4	[20, 21, 22, 23]	[24, 25, 26, 27, 28, 29, 30]
5	[25, 26, 27, 28, 29, 30]	[31] . . . Next month

Data Splitting: With limited training data, forward chaining is employed to form train-test pairs following the *temporal order* of samples, referred to as *Data_Split* (§ Algo. 3). Table IV shows an illustrative example of data splitting by forming train-test pairs from 7 failure and 20 normal days, respectively, for a month. Training set 3 uses forward chaining by appending the 18^{th} day to training set 2 increasing the history size from 1 day to 7 days. However, for training sets 2, 4 and 5, with at least 4 days each, forward chaining is not used to remain within recent history. This ensures prediction from most recent history as opposed to longer history that may reduce the model's forecasting power. The time period between two subsequent failures is considered for training, in the absence of which the previous non-empty training set (e.g., previous month) is chosen, to form non-overlapping train-test splits.

Feature Selection for Training: For each train-test pair, *GetSignalsToTrain* (#7, § Algo. 3) chooses a set of signals for sample training (#3). Algo. 2 fetches the top signals of all the training times (#10). The maximum of α values used across the sample training times forms the new threshold. The key insight is that thresholds have to be dynamically picked from a union of subspaces to include signals of interest. For each training set, signals common to all times, and signals with δ higher than a threshold from the remaining disjoint set of signals are used for training (#11-13), similar to Algo. 2.

Predict Failure (§ Algo. 3) summarizes the major steps of prediction. The selected signals for training are extracted as per the data split and classified as sparse, medium sparse, and dense based on their temporal density, as shown in Table III. This density-based grouping helps to examine suitable resampling methods for various time-series types, to convert variable-length signals to fixed-size ones. Signals are resampled to form a regular 2D matrix and this processed data (*ProcessData*) is fed to the predictor as per train-test pairs (#4-5). The predictor predicts values of signals (future time-steps) from which sample scores are computed, and compared with thresholds to decide if samples are anomalous (#6). The rationale is that the predicted samples of anomalous versus normal times would be dissimilar generating different scores, which can help identify failures.

Algorithm 3 Failure Prediction

Require: M_{Data}, Tot_M ▷ System Data, Total Signals
Ensure: Failure or Normal samples ▷ Failure or Not
1: **procedure** PREDICT FAILURE(M_{Data}, Tot_M)
2: $Train_L$, $Test_L$ ← Data_Split(M_{Data}) ▷ Train, Test pairs
3: ML_{tr} ← GetSignalsToTrain($Train_L$, Tot_M) ▷ Features to train
4: $M^{'}$ ← ProcessData(M_{Data}, ML_{tr}, $Train_L$, $Test_L$) ▷ Pre-process
5: $Test^{Sample}_{Scores}$ ← $F_{Predict}(M^{'})$ ▷ Prediction
6: Failure ‖ Normal ← ($Test^{Sample}_{Scores} > \gamma_{tr}$)? ▷ Threshold

7: **procedure** GETSIGNALSTOTRAIN($Train_L$, Tot_M)
8: $[S_i]$ ← $Train_L$; $1 \leq i \leq Y$; $[D_j]$ ← S_i ▷ Duration in S_i
9: **for each** $S_i \in Train_L$ **do** ▷ Each training set
10: $[M^j_{List}]$ ← Select Features(D_j, Tot_M) $\forall$ $D_j \in S_i$
11: $common_i$ ← ($\bigcap M^j_{List}$), $disjoint_i$ ← ($\biguplus M^j_{List}$)
12: $AddM_i$ ← $\forall$ Signals $\in disjoint_i$, if ($\delta > \max[\alpha^{D_j}]$)
13: ML^{S_i} ← ($common_i \cup AddM_i$) ▷ Shortlisted Signals
14: ML_{tr} ← $[\bigcup_{i=1}^{Y} ML^{S_i}]$ ▷ Append ML^{S_i} to the list
15: **return** ML_{tr}

Figure 7 gives an overview of the predictor, which is an autoregressive model performing many to many forecasts, with two 1D-CNN layers, a Maxpool, and three dense layers using the *Adam* optimizer, and *mse* loss function. Such a multivariate CNN model helps to predict values of multiple signals at each time-step. Sample scores in the form of weighted sum and variance are compared to the threshold (formed from the training samples) to flag failures.

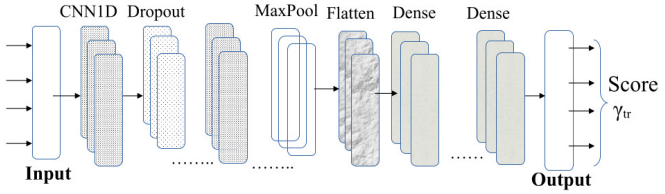

Fig. 7: Prediction Model

GAN-based models [18, 27] often require sufficient high quality training data not available in some settings. Our input is a 2D matrix, so a CNN benefits from the spatio-temporal arrangement of data in face of limited moderate quality training data and ground truth. Subsystem associations are not leveraged for prediction as establishing multi-component

TABLE V. Data Details (FI: Failure Instances)

System	Domain	#Signals	#FI	Mean Failure Span	Train Size	Test Size	Normal (%)	#Features
X-ray Laser	Beam Physics	2036	208	1.6 hrs	1929K	6163K	75.1	108-112
Spark Cluster	Distributed System	2283	7	35 mins	27K	18K	52.2	85-99
Oil Plant	Petroleum Industry	8	36	15 mins	257K	180K	64.3	4-5

TABLE VI. Evaluation Metrics

Metrics	Formula	Definition
FP Rate (%)	FP/(FP+TN) (FPR)	FP=A normal sample inferred as anomalous (False Positive)
FN Rate (%)	FN/(TP+FN) (FNR)	FN=An anomalous sample inferred as normal (False Negative)
Accuracy (%)	(TP+TN)/ (FP+FN+TP+TN)	Accuracy=Fraction of correct predictions, Errors=(1-Accuracy)
F1 Score (%)	TP/(TP+$\frac{FP+FN}{2}$)	Harmonic mean of Precision+Recall

interaction needs dedicated studies [21, 38], especially for non-traditional platforms where the heterogeneity in subsystems is more, and domain insights may be needed. In real-life environments, where subsystem signal correlations are not yet fully understood, Prolego is still applicable.

V. Evaluation

System Environment: We use a shared high performance computing (HPC) cluster of 160 nodes with CPU-only and multi-GPU nodes for experimental evaluation. Most of our experiments are run with 8 to 12 CPUs and 3 GPUs per task, across 3 to 6 nodes. Across all the systems, a suitably chosen sample size determines the lead time.

Datasets and Baseline Comparison: We evaluate real-world datasets from three different domains, namely, an X-ray laser, an Apache Spark cluster [8], and an oil plant [6], as shown in Table V. The mean failure duration of failure instances (FI) across the systems are 1.6 hrs, 35, and 15 mins, respectively. Column *Features* indicate the number of signals used for training after feature selection. Prolego [7] is compared with two baseline methods, both of which have shown recent success with time-series prediction [37, 44]: a) Random Forest (RanFor) [53], a commonly used supervised model for prediction, and b) Long Short-Term Memory (LSTM) [46], a recurrent neural network (RNN) model often used for time-series forecasting. Table VI shows the evaluation metrics used.

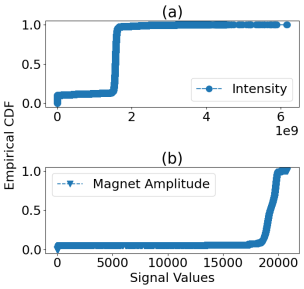

Fig. 8: Signal CDFs

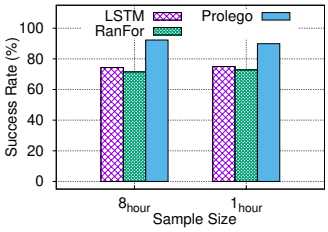

Fig. 9: Detection

X-ray Laser: The LCLS [4] X-ray laser system is a large-scale particle accelerator, a CPS generating high energy beams for scientific experiments. Various environmental aberrations, dynamic changes in parameter configurations, and subsystem faults (e.g., magnet, RF, water system) of this complex system result in anomalous signals. Some faults degrade performance (i.e., lower the beam intensity without completely losing signal strength), others recover automatically from temporary fluctuations, while the rest lead to beam downtime. We use production logs that contain sensor data from across the entire system as continuous time-series of normal and anomalous times. Fig. 8 shows the distribution of 2 signals over the same timeframe through CDF (cumulative distribution function) plots. The *Intensity* (a) and *Amplitude* (b) signals show a heavy tail, and a non-uniform distribution, respectively. Several signals with non-stationary mean exhibit changing trends over time. For predictive modeling, capturing trends at suitable time-scales helps. Accordingly, we choose sample sizes for our analysis. We study failures, i.e., downtimes, based on the available coarse ground truth in terms of 8-hour windows of a day. For instance, a specific water system fails between 8:00 and 16:00 hours leading to 1-hr downtime. The actual time of the onset and completion of anomaly during those 8 hours is not labeled. First, Prolego generates finer labels from such 8-hour windows.

We leverage two well monitored signals related to beam intensity for labeling; similar customary signals, not requiring much domain knowledge are observed in complex systems for daily maintenance activities. The intensity signals are dense time-series corresponding to two physically nearby regions of the system that quantify the system performance adequately, hence we choose these signals over others. The detector is run with 8-hr sample size to check the model's accuracy as validation with ground truth is feasible. Normal data is used for training, and failure data for testing. 8-hr windows are split to 1-hr windows to estimate which hours within a 8-hr window are anomalous. Labeling multiple 1-hr samples is fine-grained enough to deduce labels for 4-hr samples later used in prediction. In this case, the test data contains a mix of anomalous and normal samples. Manual verification of generated labels via direct consultation with experts showed the presence of sparse or lower magnitude of data compared to normal times (as in Fig. 1a vs. 1b), supporting the hypothesis that the classification as an anomalous sample is correct. Prolego verifies the generated fine-grained labels w.r.t. the recorded failure durations. Figure 9 shows that for 8-hr periods, Prolego's mean success rate is as high as 92.3%, and for 1-hr 89.9%. The detection and labeling times for 80 to 90 samples are over 2 hours using CPU, and 30 to 50 mins using GPUs. Detection of anomalous times is an offline computation, and so these runtimes are acceptable.

After selecting signals for training, experiments are conducted for two cases, a) 8 hours of lead time with 16 hours of history (16:8) and b) 4 hours of lead time with 4 hours of history (4:4). We could not obtain better accuracy for more than 8 hours lead time. The 4-hour sample size is chosen as it is adequate for operators to respond in case of any imminent failure. Table. VII shows the results across all the methods for different lead times. Prolego obtains 84.2%, and 81.4% accuracy for 8-hour, and 4-hour lead times, respectively, with

TABLE VII. Prediction Results

Methods	X-Ray Laser						Spark Cluster						Oil Plant			
	Accuracy		F1		FPR	FNR	Accuracy		F1		FPR	FNR	Accuracy	F1	FPR	FNR
	8-hr	4-hr	8-hr	4-hr	8-hr	8-hr	10-min	5-min	10-min	5-min	10-min	10-min	15-min	15-min	15-min	15-min
RanFor	63.1	63.7	65.5	62.4	42.8	31.03	61.9	69.6	65.2	71.2	45.0	31.8	72.4	77.7	41.6	17.6
LSTM	71.9	72.5	70.3	73.04	31.25	24.02	71.4	62.5	68.4	68.6	22.7	35.03	68.9	58.3	35.1	30.2
Prolego	84.2	81.4	84.7	81.08	20.6	10.7	80.9	82.1	73.3	85.7	11.5	19.2	86.2	90.47	18.7	5.1

over 80% F1-score. Presence of anomalous times with short-term failures (e.g., 6 mins, <1-hr) and similarity of certain train-test samples are possible causes for the errors. We calculate the false positives and false negatives based on the available ground truth that accounts for these errors. The FPR and FNR for 8-hour samples are comparatively $2\times$ lower with Prolego. The number of normal samples in the test set is generally higher than anomalous samples, that could lead to higher false positive rate. One potential reason for errors is that once the system recovers from a failure, its operating condition can be statistically different due to a change in the user experiment.

1-hour labeled samples obtained from prior labeling phase is used to deduce labels for 4-hour samples. If $\geq$75% of the 1-hour samples are labeled as anomalous, and the corresponding 4-hour sample is *predicted* as anomalous, we do not count it as an error. We examined the predicted failures with reference to the 4-hour samples. If a failure duration is 7 hours, and the related 4-hour sample is predicted as normal, it is treated as an error. As Table VII gives a coarse-grained measure of accuracy for the case of 4-hour sample based on the deduced labels and known ground truth, the false positive or false negative rates are not explicitly computed further.

Spark Cluster: This dataset contains operating system (OS) and application-level signals related to stream processing applications running on a Apache Spark cluster [26]. Anomalies are injected in the traces by generating bursty input causing memory pressure, eventually crashing jobs. Timeframes related to such crashes or failures serve as our ground truth data. During anomalous times the observations rise and fall relatively more both in frequency and magnitude over normal periods. Normal times are used for training, with sample sizes 10- and 20-mins, respectively. The average fraction of normal times in relation to the anomalous times are below 60%, hence shorter sample sizes are chosen. Our labeling method has 85.3 to 86.9% accuracy which is at least 10% more than other methods. As ground truth is available for a specific range of values and our approach performs relatively better, we skip the comparative results of labeling for brevity. Some of the signals that have insufficient distinct observational range needed for feature selection are omitted. Since the amount of training data corresponding to a single usage pattern is limited, we choose moderate windows of 10-min and 5-min for prediction. Table. VII shows that the F1-scores are higher and errors are generally lower with Prolego. Since the sample size is not long enough, some short-lived dips and spikes are predicted as anomalies leading to few errors. We omit the FPR and FNR of 5-min samples as additional details about the distributed system may be needed for ascertaining correctness.

Oil Plant: Oil wells are equipped with valves and sensors such as beam pumps and hydraulic systems [50]. Anomalous events such as oil flow blockage, flow instability, valve faults can disrupt productivity by causing slugging. This dataset has 8 monitored signals related to pressure and temperature of different subsystems. Deviation of expected sediment flow rate, spurious valve closure, and presence of hydrates are some unwanted failures that waste resources. We study real anomalies related to flow instability and control valve problems. This dataset is densely labeled (i.e., each row is labeled as to whether it relates to an anomaly). Hence, we do not need to generate additional labels. Yet, we run our detector to assess its efficiency and found that $\approx$83% of the samples are correctly detected. The source of errors in this case is lack of sufficient range in the anomalous windows. Since our approach is designed for collective contextual anomalies, if too few observations are part of the anomalous segment it may be missed. Often the pressure and temperature signals of a transducer are used to assess the health of the oil plant. In this case anomalies are characterized by lack of periodicity in the signals.

A subset of these 8 signals based on Algo. 2 is used for prediction. Normal data is used for training, using 15-min samples. This sample size is chosen based on the range of the anomalous segments and the temporal trends observed in the signals. Table VII shows that the error rates of Prolego are lower by at least $2\times$ compared to other methods. As the number of dimensions are relatively lower in this system, *RanFor* performs better than the other systems. Lack of adequate trends in signals is a potential source of misprediction.

Across all the datasets, the average per sample training time is 4 to 10 mins ($\pm$2.3) for prediction. This is the mean runtime for learning one sample of a train-test pair. Per sample inference time is below 0.5 millisecs, with a pre-trained model available. We monitor the validation and training losses to prevent model overfitting. For all analyzed lead times the training times are on the order of few minutes, with sub-second inference time.

Acceptable Errors: Abrupt subsystem faults leading to rapid failures can make prediction infeasible. For such cases false negatives are acceptable. Operators prefer early warnings even if there is no failure to monitor degrading subsystems, for which false positives are acceptable. For e.g., 8 false positives of which 5 are related to magnet problems and 3 to manual energy change, causing system instabilities, can direct the operators to monitor magnet subsystem or parameter tuning adjustments. Many physical systems do not have any preemptive measures in place for failures; thus false alarms do not waste resources.

Some form of forewarning is helpful in most complex systems as the cost of errors are not high. Despite few errors, Prolego can help plan user experiments or maintenance activities as opposed to not having any assessment about future failures.

Performance Optimization: Among the phases of our system, actual inference time is minimal (<1 sec) and requires few hardware resources. The timeliness of the COV computation and prediction, which are done more frequently over shorter time-windows determines how much lead time can be retained. As an estimate, COV-based ranking and feature selection from 1200 signals for 8-hour samples take ≈30 hours of time on CPUs. Known models (e.g., MaxVar [34] or PCA [38] with added costs of normalization, model training for feature elimination etc.) have similar or higher time complexity.

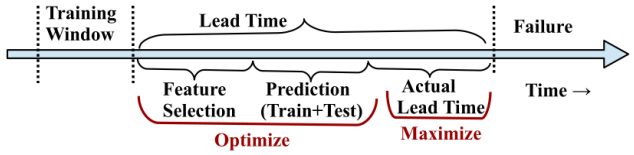

Fig. 10: Online Prediction

The stage that requires the most acceleration is the online feature selection, because depending on the number of signals (can be $\mathcal{O}(10^3)$ to $\mathcal{O}(10^6)$ in some systems) or sample size, it is easily possible for this stage to consume all of the lead time for prediction. For instance, to predict a failure in the next 4 hours, taking 15 mins for feature selection, training, and inference leaves 3.75 hours of actual lead time. Signal preparation and prediction time must be minimized to maximize lead time for operators, as shown in Figure 10. For continuous forecasting, the feature selection phase can benefit from acceleration via moderate scaling (e.g., variance operation on a DataFrame for 2500 signals across 3 to 12 nodes), though once a trained model is available single node inference can suffice.

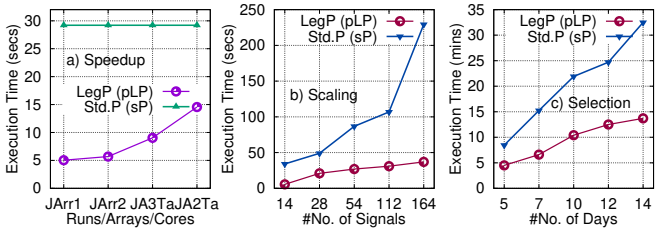

Fig. 11: Performance Optimization

Extensively used Pandas and NumPy operations can be accelerated with Dask or Numba [2, 5] libraries. These libraries by default do not use multiple cores even for single-node computations. Prolego uses the recently developed Legate system [11] that transparently scales computations, to accelerate Pandas `mean` and `variance` operations across multiple cores over multi-GPU nodes. Legate [1] is a programming model that enables scalable computations. Prolego compares the COV computation time with and without Legate to assess runtime gains. Our experiments show reduced signal selection time before prediction, providing evidence that predictive models and runtime systems together can enable timely maintenance.

Figure 11 shows the performance of Legate Pandas [3] (pLP) and standard Pandas (sP) operations for different parallel configurations, number of signals, and time durations. For `sP`, experiments are run sequentially on a single node. We experiment with the number of tasks, job arrays, cores, and Legate-specific parameters (e.g., #gpus), to improve execution times for `pLP`. In plots 11a (Speedup) and 11b (Scaling), the y-axis is the COV calculation time. Fig. 11a shows that for a fixed input size of 14 signals, `pLP` can be 6× faster than `sP`, where the x-axis indicates different parallel settings. With an increase in the number of signals, `pLP` can achieve 2× to 6× (e.g., 229 secs down to 37 secs) speedup over `sP`, as seen in Fig. 11b. In Fig. 11c (Selection), the y-axis is the aggregate runtime of ranking and selection of 12 signals from 164, and the x-axis is the number of days in a training set. `pLP` is at least 2× faster than `sP`, lowering the signal selection time for 14 days by ≈18 mins. Less than 17 cores per task were needed to improve the execution time. Improved runtimes from limited optimizations on a small scale suggests the viability of forecasting speedup through enhanced performance of various NumPy or Pandas operations across thousands of signals. A single node can potentially become a bottleneck with bandwidth constrained storage I/O, or higher number of signals. Such distributed runtime scalability can render timely support during online complex system monitoring.

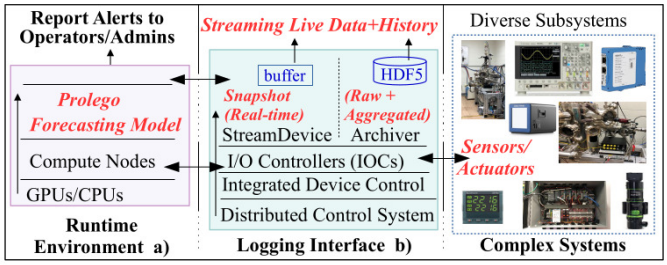

Fig. 12: System Monitoring

Prolego Application: Modern complex systems comprising heterogeneous IoT, sensor-enabled dissimilar subsystems are often monitored with a virtual machine (VM)-based distributed control system for database (e.g., HDF5) archival. Community efforts include efficient live data streaming from device controllers. Figure 12 shows that data from the logging interface ⓑ can be fed to a predictor ⓐ, like Prolego, for timely data analysis. Alternately, specific cluster nodes can be directly configured to stream or archive signals from 10s to 1000s of Input/Output controllers (IOCs) for rapid access. Prolego-like frameworks can complement CPSs/ICSs for better reliability.

Discussion: We find that a few minutes (<1 hour) of lead time is insufficient for hardware fault repair in many systems but user experiments can be gracefully stopped. A few hours of lead time is helpful in coordinating onsite recovery. For software controlled repair not requiring onsite activity, 10 to 15 minutes of lead time suffice. Besides alerting operators, warning users of future failures is also helpful. Obtained sub-second inference time is fast enough to give operators warning of potential failures that will occur on the scale of minutes to hours in the future. Dynamic retraining cost over short time windows (e.g., 1 or 2 weeks) is reasonable, as feature selection and model training time is usually below 30 minutes with Prolego.

Efficient imputation of irregular real-world multivariates besides simulated univariates [15, 41] and availability of adequate recent history can further enhance prediction efforts.

VI. Conclusion

This paper presents Prolego, a methodology to predict failures from complex systems. Prolego obtains 5 minutes to a few hours of lead time with over 80% prediction accuracy. Prolego achieves sub-second inference time during prediction making it viable for online monitoring. We show insights to signal deviations during normal and anomalous times to assess the impact of failures on time-series logs. Prolego demonstrates the potential of performance optimization via scalable feature selection to improve lead times in practice.

Acknowledgment

We are grateful to William Colocho and other engineers for sharing their knowledge about LCLS, Wonchan Lee for his timely support with Legate, the anonymous reviewers for their helpful feedback, and acknowledge the support received from the Scientific Data Facility for evaluation. This work is supported by LCLS development program, and the U.S. DOE, under contract DE-AC02-76SF00515 at SLAC NAL.

References

[1] *cuNumeric*, https://github.com/nv-legate/cunumeric.
[2] *DASK*, https://dask.org/.
[3] *Legate Pandas*, https://github.com/nv-legate/legate.pandas.
[4] *Linac Coherent Light Source*, https://lcls.slac.stanford.edu/.
[5] *Numba*, https://numba.pydata.org/.
[6] *Oil Plant*, https://github.com/petrobras/3W/tree/main/dataset.
[7] *Prolego*, https://github.com/adaptsyslearn/Prolego.
[8] *Spark*, https://github.com/exathlonbenchmark/exathlon.
[9] P. Arpaia *et al.*, "Machine learning for beam dynamics studies at the CERN Large Hadron Collider," *Nucl. Instrum. Meth. A:*, vol. 985, 2021.
[10] G. Azzopardi *et al.*, "Automatic spike detection in beam loss signals for LHC collimator alignment," *Nucl. Instrum. Meth. Sec. A:*, 2019.
[11] M. Bauer and M. Garland, "Legate NumPy: accelerated and distributed array computing," in *SC*, 2019, pp. 23:1–23:23.
[12] A. Blázquez-García *et al.*, "A review on outlier/anomaly detection in time series data," *ACM Comput. Surv.*, 2021.
[13] A. Bombarda *et al.*, "Towards an evaluation framework for autonomous systems," in *ACSOS-C*. IEEE, 2022, pp. 43–48.
[14] A. Borghesi *et al.*, "Online anomaly detection in HPC systems," in *IEEE AICAS*, 2019, pp. 229–233.
[15] E. D. Brouwer *et al.*, "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series," in *NeurIPS*, 2019.
[16] Y. Chen *et al.*, "Outage prediction and diagnosis for cloud service systems," in *WWW*, 2019, pp. 2659–2665.
[17] R. C. Chiang *et al.*, "Matrix: Achieving predictable virtual machine performance in the clouds," in *USENIX ICAC*, 2014.
[18] Y. Choi *et al.*, "GAN-Based anomaly detection and localization of multivariate time series data for power plant," in *IEEE, BigComp*, 2020.
[19] E. Costa *et al.*, "Systematically inferring I/O performance variability by examining repetitive job behavior," in *SC*, 2021.
[20] Z. Z. Darban *et al.*, "Deep learning for time series anomaly detection: A survey," *CoRR*, vol. abs/2211.05244, 2022.
[21] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *AAAI*, 2021.
[22] E. Fol *et al.*, "Detection of faulty beam position monitors using unsupervised learning," *Physical Review Accelerators and Beams*, 2020.
[23] Q. Han *et al.*, "Aquaeis: Middleware support for event identification in community water infrastructures," in *Middleware*. ACM, 2019.
[24] A. Iseki *et al.*, "Estimating the causal effect from partially observed time series," in *AAAI*, 2019, pp. 3919–3926.
[25] A. A. Ismail *et al.*, "Benchmarking deep learning interpretability in time series predictions," in *NeurIPS*, 2020.
[26] V. Jacob *et al.*, "Exathlon: A benchmark for explainable anomaly detection over time series," *Proc. VLDB Endow.*, 2021.
[27] F. Khoshnevisan and Z. Fan, "RSM-GAN: A convolutional recurrent GAN for anomaly detection in contaminated seasonal multivariate time series," *CoRR*, vol. abs/1911.07104, 2019.
[28] D. Li *et al.*, "MAD-GAN: multivariate anomaly detection for time series data with generative adversarial networks," in *ICANN*, 2019.
[29] L. Li *et al.*, "Conan: Diagnosing batch failures for cloud systems," in *ICSE SEIP*, 2023.
[30] N. Li *et al.*, "Reasoning about when to provide explanation for human-involved self-adaptive systems," in *ACSOS*. IEEE, 2020, pp. 195–204.
[31] Y. Li *et al.*, "Data stream event prediction based on timing knowledge and state transitions," *Proc. VLDB Endow.*, 2020.
[32] Z. Li *et al.*, "Generic and robust localization of multi-dimensional root causes," in *IEEE ISSRE*, 2019.
[33] F. F. Lin *et al.*, "Fast dimensional analysis for root cause investigation in a large-scale service environment," *Sigmetrics*, 2020.
[34] M. Luo *et al.*, "Adaptive unsupervised feature selection with structure regularization," *IEEE Trans. Neural Networks Learn. Syst.*, 2018.
[35] Y. Luo *et al.*, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," *ACM Comput. Surv.*, 2021.
[36] D. Marcato *et al.*, "Machine learning-based anomaly detection for particle accelerators," in *CCTA*. IEEE, 2021.
[37] B. Nie *et al.*, "Mining multivariate discrete event sequences for knowledge discovery and anomaly detection," in *DSN*, 2020.
[38] A. J. Oliner and A. Aiken, "Online detection of multi-component interactions in production systems," in *IEEE DSN*, 2011.
[39] A. J. Oliner, A. V. Kulkarni, and A. Aiken, "Using correlated surprise to infer shared influence," in *IEEE/IFIP DSN*, 2010.
[40] D. Ratasich *et al.*, "Adaptive fault detection exploiting redundancy with uncertainties in space and time," in *SASO*. IEEE, 2019, pp. 23–32.
[41] Y. Rubanova *et al.*, "Latent ordinary differential equations for irregularly-sampled time series," in *NeurIPS*, 2019.
[42] S. Siami-Namini *et al.*, "A comparison of ARIMA and LSTM in forecasting time series," in *IEEE, ICMLA*, 2018.
[43] I. Steinwart and A. Christmann, *Support vector machines*. Springer Science & Business Media, 2008.
[44] Y. Su *et al.*, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *ACM SIGKDD*, 2019.
[45] M. Sun *et al.*, "CTF: anomaly detection in high-dimensional time series with coarse-to-fine model transfer," in *IEEE INFOCOM*, 2021.
[46] S. Tariq, S. Lee, Y. Shin *et al.*, "Detecting anomalies in space using multivariate convolutional LSTM with mixtures of probabilistic PCA," in *ACM SIGKDD*, 2019.
[47] O. Tuncer *et al.*, "Online diagnosis of performance variation in HPC systems using machine learning," *IEEE TPDS*, 2019.
[48] G. Valentino *et al.*, "Anomaly detection for beam loss maps in the Large Hadron Collider," in *Journal of Physics*, 2017.
[49] G. Valentino and B. Salvachua, "Machine learning applied at the LHC for beam loss pattern classification," in *IPAC*, 2018.
[50] R. E. V. Vargas *et al.*, "A realistic and public dataset with rare undesirable real events in oil wells," *Journal of Petroleum Science and Eng.*, 2019.
[51] W. Wang *et al.*, "Active-MTSAD: Multivariate time series anomaly detection with active learning," in *DSN*, 2022.
[52] T. Wittkopp *et al.*, "PULL: Reactive log anomaly detection based on iterative PU learning," in *HICSS*, 2023.
[53] C. Wu *et al.*, "Identifying root-cause metrics for incident diagnosis in online service systems," in *ISSRE*. IEEE, 2021.
[54] X. Xu *et al.*, "Pod-diagnosis: Error diagnosis of sporadic operations on cloud applications," in *IEEE DSN*, 2014.
[55] A. Yazdi *et al.*, "SEFEE: lightweight storage error forecasting in large-scale enterprise storage systems," in *SC*. IEEE/ACM, 2020.
[56] C. Zhang *et al.*, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *AAAI*, 2019.
[57] S. Zhang *et al.*, "Robust KPI anomaly detection for large-scale software services with partial labels," in *ISSRE*. IEEE, 2021, pp. 103–114.
[58] X. Zhang *et al.*, "Cross-dataset time series anomaly detection for cloud systems," in *USENIX ATC*, 2019.
[59] Y. Zhang *et al.*, "Unsupervised deep anomaly detection for multi-sensor time-series signals," *IEEE TKDE*, 2021.
[60] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *ACM SIGKDD*, 2017.
[61] Zúñiga *et al.*, "Classical failure modes and effects analysis in the context of smart grid cyber-physical systems," *Energies*, 2020.