Deep Sequence Modeling for Event Log-based Predictive Maintenance

Yun Zhou
yunzzhou@amazon.com
Amazon Generative AI Innovation Center

Yawei Wang
yawenwan@amazon.com
Amazon Generative AI Innovation Center

Huan Song
huanso@amazon.com
Amazon Titan Labs

Tesfagabir Meharizghi
mehariz@amazon.com
Amazon Generative AI Innovation Center

Mohamad Al Jazaery
mohjaz@amazon.com
Amazon Generative AI Innovation Center

Denisse Colin-Magana
dcolin-magana@lnw.com
Light & Wonder, Inc.

Aruna Abeyakoon
aabeyakoon@lnw.com
Light & Wonder, Inc.

Panpan Xu
xupanpan@amazon.com
Amazon Titan Labs

ABSTRACT

Predictive maintenance (PdM) is one of the most important machine learning (ML) use cases in a wide range of businesses that own or manufacture machinery. Many PdM applications utilize event logs/sequences that record machine operating conditions and health data. However, industrial scale PdM use cases bring a variety of challenges which makes it difficult to directly apply state-of-the-art models. In this work, we start with a PdM use case from a company Light & Wonder (L&W) Inc., which requires us to work with about 280 million events from ~500 gaming machines accumulated over 10 months, and a short 24-hour time window could contain from 0 to approximately 29,000 events. Developing performant models for event sequence data at such scale is a non-trivial task. In this work, we propose a generic, generalizable modeling framework which we refer to as *ElasticPdM* for industrial scale PdM applications using event log data, which uses novel methods including time-adaptive event sequence binning and time-aware event sequence embedding to process long event sequences. Applied on the real-world L&W data, our *ElasticPdM* is **13-20%** more accurate than state-of-the-art baselines on *Recall at High Precision* metrics used commonly for PdM.

KEYWORDS

deep sequence modeling; predictive maintenance; temporal event sequence

ACM Reference Format:
Yun Zhou, Yawei Wang, Huan Song, Tesfagabir Meharizghi, Mohamad Al Jazaery, Denisse Colin-Magana, Aruna Abeyakoon, and Panpan Xu. 2023. Deep Sequence Modeling for Event Log-based Predictive Maintenance. In *Proceedings of (KDD MiLeTS '23).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD MiLeTS '23, Aug. 07, 2023, Long Beach, CA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Predictive maintenance (PdM) is one of the most important machine learning (ML) use cases for businesses with physical equipment or machinery [6, 20, 27] and is becoming much more prevalent in recent years due to the increasing availability of connected devices, cheap storage systems and computational power. With PdM, businesses can act more proactively and take corrective actions that prevent potential machine failures, which reduces machine downtime and improves operating efficiency. While many PdM use cases rely on sensors that record physical properties such as temperature and vibration [1, 13, 16], a wide range of applications utilize machine event logs, where each event is associated with a marker and a timestamp [11, 12, 21]. Modeling such event sequence data is therefore the key to building an effective PdM solution. Recent advances in ML demonstrated that deep neural networks (DNNs), especially Transformers are the state-of-the-art architectures for sequence modeling tasks [23]. However, directly applying these models is difficult due to a variety of challenges in real-world PdM use cases. One challenge is related to the large number and uneven distribution of events over time, where a small time window could contain a few to tens of thousands of events or more.

In this work, we address a real-world industrial scale predictive maintenance use case and share the practical lessons and insights we gained building a Transformer-based model for productionization. The use case is from Light&Wonder (L&W), a leading cross-platform gaming company that provides gaming equipment and services. L&W remotely streams event log data from gaming machines distributed globally and want to use the data for PdM. The event log data is expected to cover approximately half-million machines when reaching its full potential. Currently, a 10-month log from approximately 500 machines already contains *280 million* events which makes it imperative to build a scalable data processing and modeling solution. To address this challenge, we propose *ElasticPdM*, a modeling framework that uses novel methods including *time-adaptive event sequence binning* and *time-aware event sequence*

embedding to process long event sequences. We validate the design of *ElasticPdM* through comparisons with baseline models as well as ablation studies. The result demonstrated that our approach performs **13-20%** better on key *Recall at High Precision* metrics commonly used for evaluating PdM models. At the same time, L&W is actively working on deployment of *ElasticPdM*.

2 RELATED WORK

PdM techniques utilize ML to determine the conditions of equipment and devices in a look-ahead time window, and allows proactive actions to be planned to avoid potential breakdowns. PdM has applications in a wide range of domains with physical equipment such as medical devices [21], automated teller machines (ATM) [11], electric propulsion systems [3], rolling-element bearings [14], computer workstations [19] etc. Many existing work on PdM utilize conventional ML models, including support vector machine (SVM) [8] and decision tree-based models [11, 21]. An important benefit of these models is their interpretability that can help reveal contributors to machine breakdowns. Recently, Deep Neural Networks (DNNs) demonstrate superior performance on modeling temporal sequences for tasks like natural language and speech processing [9, 23, 24] and time series analysis [18, 22]. However, only a handful of work adopt DNNs for PdM task so far. For example, Juodelyte et. al [14] focused on continuous vibration signals (instead of discrete events) and utilized fully connected networks to predict bearing degradation from time- and frequency-domain signals. Hafeez et. al develops long short-term memory networks (LSTMs) tackling the large number of variables in predicting the next fault event from vehicular Diagnostic Trouble Code (DTC) [12]. Our work differs from the above in that our *ElasticPdM* architecture consists of a suite of fully connected, convolutional and transformer DNN modules, and we designed a systematic hyperparameter-tuning/architectural search approach to determine the optimal network structure. Another line of work formulate the problem as the remaining useful life (RUL) regression, such as the regression based method [17], covariate based hazard models [25], estimation with dynamic Bayesian network [5], and those based on neural networks such as RNN [26]. The main reason that we formulated it as a classification problem is to convey actionable instructions to end users who deploy our PdM framework and prefer a simpler decision: whether to dispatch workers onsite or not. To reduce label noise we augmented the data with overlapping sliding windows such that sufficient training signal is fed into the model. Below, we propose *ElasticPdM*, a scalable method for modeling large scale machine event logs for PdM with novel features including *time-aware event sequence embedding* and *time-adaptive event sequence binning*.

3 BACKGROUND AND PROBLEM SETTING

Our work starts with a real-world, large-scale predictive maintenance use case from Light&Wonder (L&W), a leading cross-platform gaming company that provides gaming equipment and services. L&W recently developed a secure solution to stream telemetry and health data remotely from gaming machines, which is expected to cover approximately half-million machines distributed globally when reaching its full potential. Approximately 500 different types

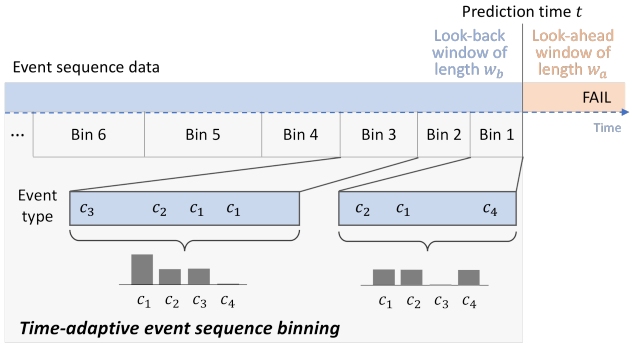

Figure 1: Our problem formulation and the proposed time-adaptive event sequence binning method. At any prediction time t, we utilize historical event sequences from a look-back time window, to predict the probability of machine breakdown within the look-ahead window. To handle large number of events and long event histories, we apply time-adaptive sequence binning which uses finer resolution for recent history. The event frequencies are computed in each bin as model input.

of machine events are being monitored in near real-time to give a full picture about machine conditions.

L&W's goal is leveraging the streamed machine events data to enable predictive maintenance. The benefit is two-fold: by focusing service teams' limited bandwidth on machines with potential failures, L&W can reduce the effort spent on regular inspections; with advanced alerts for machines with high-probability of breakdowns, L&W can proactively dispatch service teams to reduce machine downtime and avoid significant revenue loss for customers.

Next, we formally define the problem and introduce the notations used in the paper. The problem formulation can be easily applied to other predictive maintenance use cases. We denote event stream from a single machine as $\mathcal{D} = \{d_i\}_{i=1}^{T}$. Each d_i includes an event marker and a timestamp, $d_i = (e_i, t_i)$, where e_i belongs to a predefined set of event types which denote as $C = \{c_k\}_{k=1}^{K}$. $K = \|C\|$ is the number of different event types. For each machine at any time t, we can collect the recent event history up to t with a look-back time window of length w_b, which we denote as $\mathcal{H}_{t-w_b:t} = \{(e_i, t_i) : t - w_b < t_i \leq t\}$. Note that due to the uneven distribution of events over time, $\mathcal{H}_{t-w_b:t}$ may contain varying number of events. For the gaming machines from L&W, we found that a 24 hour time window could contain 0 to approximately 29,000 events. Besides event logs, each machine is also associated with a set of time-invariant features which we refer to as machine metadata. Each feature could be a categorical variable (*e.g.* the operating system), a set (*e.g.* the game themes) or a numerical variable.

Given a machine's recent event history $\mathcal{H}_{t-w_b:t}$ and the associated metadata $\boldsymbol{X}^{meta}$, our goal is predicting the conditional probability of machine failure within a look-ahead window w_a, denoted as

$$P^{t:t+w_a}(y = 1 | \mathcal{H}_{t-w_b:t}, \boldsymbol{X}^{meta}) \quad (1)$$

where $y = 1$ indicates that the machine encountered malfunctions within the time window $(t, t + w_a]$ and vice versa. The length of the look-ahead window w_a is use case dependent and is set to six hours for the gaming machines in L&W's case.

Symbol	Meaning
$\mathcal{D} = \{d_i\}_{i=1}^{T}$	An event sequence of length T
$d_i = (e_i, t_i)$	Data contained in an event sequence includes an event marker e_i and a timestamp t_i
$C = \{c_k\}_{k=1}^{K}$	Set of predefined event types with K as the total number of unique events
w_b	Look-back window length
w_a	Look-ahead window length
$\mathcal{H}_{t_i:t_j}$	Set of events between time t_i and t_j
B	Number of bins
l	Shortest, most recent bin length
s	Bin length increase step
r	Bin length increase factor
$\boldsymbol{X}^{bin} = \{\boldsymbol{x}_i\}_{i=1}^{B}$	Binned representations
$\boldsymbol{T}^{bin} = \{t_i\}_{i=1}^{B}$	Timestamps associated with the binned representations
$\boldsymbol{X}^{meta}$	Metadata

Table 1: Notations used in the paper.

4 METHODOLOGY

In this section, we describe our proposed modeling framework *ElasticPdM* for predictive maintenance of gaming machines based on machine event log data.

4.1 Proposed modeling framework: *ElasticPdM*

4.1.1 Time-adaptive event sequence binning. As discussed in the last section, in event log-based PdM applications, the number of events in a time window can be huge and varies dramatically over time. Therefore an effective model needs to be able to handle extremely long sequences while still retaining meaningful representations for very short ones. We propose the use of time-adaptive event sequence binning to handle this challenge in the feature extraction stage, inspired by the findings that more distant events in the history may contain less useful signals for sequence prediction [4]. Specifically, we divide a look-back time window into bins of varying lengths and calculate the event frequencies in each bin. We use shorter bins to preserve finer time resolution in recent history and use longer bins to cover distant history. Varying bin lengths allow us to integrate information from a longer look-back window (which usually improves model performance, as shown in Table 5) without drastically increasing computational cost. This is especially important for Transformer-based models due to their quadratic time and space complexity with respect to the length of the sequence.

Formally, we specify the length of the shortest bin (bin ending at time t) as l, and update the bin length for every s bins with the multiplier of r ($r > 1$). The bin length for i-th bin is:

$$l_i = l \cdot r^{\lfloor \frac{i-1}{s} \rfloor} \tag{2}$$

Figure 1 illustrates it using a toy sample. In this example, the look-back window is divided into six gradually longer bins, and the event frequencies are calculated for each bin.

Due to the exponential increase in bin length, the number of bins required to cover a fixed-length look-back window is reduced significantly, as compared to evenly binning. Specifically, the number of bins needed to cover a fixed-length window w_b under selected s, l, r can be calculated as:

$$B = s \cdot \lceil \log_r (\frac{(r-1)w_b}{sl} + 1) \rceil \tag{3}$$

Therefore, the time and space complexity for every layer of the transformer is $O((\log w_b)^2)$. This is compared to the complexity of $O(w_b^2)$ for the evenly binning approach.

We denote the start and end time of the i-th bin as t_{i+1} and t_i. The event history contained in the i-th bin is $\mathcal{H}_i = \mathcal{H}_{t_{i+1}:t_i}$. We count the frequency of each event c_k in $\mathcal{H}_i$ (noted as $\#c_k^{(\mathcal{H}_i)}$), and normalize it by the average frequency of c_k in i-th bin, calculated based on all the training samples. This reduces the overwhelming effect of frequent, but less important events such as "start of game" in the gaming machines. Formally,

$$x_i^{(k)} = \#c_k^{(\mathcal{H}_i)} / \overline{\#c_k^{(i)}} \tag{4}$$

For each i-th bin, we can obtain its representation $\boldsymbol{x}_i \in \mathbf{R}^K$, where K is the number of unique events in the dataset. Binned event frequencies $\boldsymbol{X}^{bin} = \{\boldsymbol{x}_i\}_{i=1}^{B}$ as well as the timestamps for each bin $\boldsymbol{T}^{bin} = \{t_i\}_{i=1}^{B}$ are inputs to *ElasticPdM* as illustrated in Figure 2, along with machine metadata.

4.1.2 Multi-modal embedding. As illustrated in Figure 2, the embedding module of *ElasticPdM* contains:

Time-aware event sequence embedding. The event sequence data contains both the event sequences and the timestamps. Although it appears that the event itself is the predominant signal, we found that the timestamp information is indispensable in achieving performant predictions. To this end, we extract *time-aware embedding* that tightly incorporate time and event sequence embedding. We capture timestamp information in the model by extracting features from the timestamps associated with the bins, *i.e.* $\boldsymbol{T}^{bin} = \{t_i\}_{i=1}^{B}$. Each t_i is converted into a feature vector by extracting its month-of-year, day-of-month, day-of-week, hour-of-day, minute-of-hour and etc. Each extracted feature is converted to a normalized numerical variable that ranges from -1 to 1[1]. For example, Monday, Tuesday and Sunday are first marked as 0, 1, and 6 respectively, and then scaled to [-1,1]. We also feed the event frequency representation of each bin, i.e. $\boldsymbol{x}_i \in \mathbf{R}^K$ through a learnable linear layer to obtain event embedding. The event embedding and timestamp embedding are then concatenated along the feature channel in each bin. The time-varying signals need to be captured early on by the model. Therefore, we feed the concatenated embedding at the very beginning of the model architecture as input to the CNN module.

Metadata embedding. Metadata contains machine-specific features. Example of machine metadata from L&W include locations, game themes, etc. To provide comprehensive signal for predictive maintenance, we include metadata information in *ElasticPdM*. We use one-hot encoding for categorical features (*e.g.* the operating system) and multi-hot encoding for features containing set information (*e.g.* the game themes) respectively. The encoded features are concatenated and an MLP is used to obtain the final metadata embedding. Considering the static nature of the metadata, we feed it

[1]Implemented with GluonTS [2]

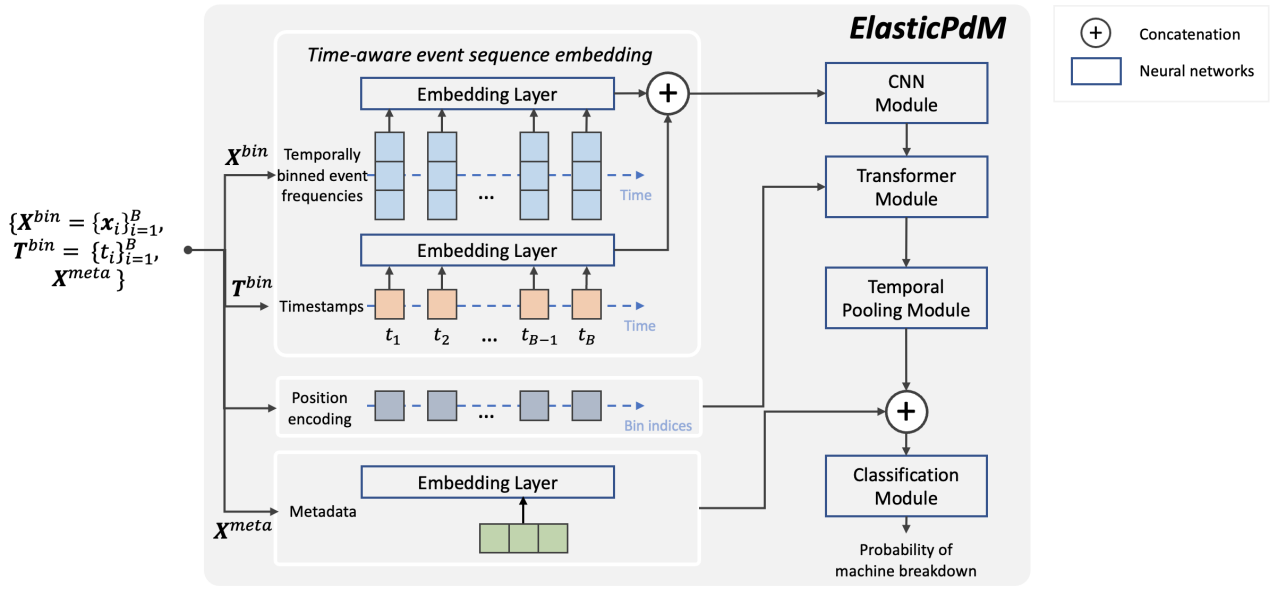

Figure 2: Our proposed *ElasticPdM* framework for predictive maintenance which takes in binned event sequences with timestamps and metadata to predict probability of machine breakdown. *ElasticPdM* extracts *time-aware event sequence embedding* as well as static metadata embedding as input to the downstream neural network.

into the model in a later stage and concatenate it with the temporally pooled event sequence embedding prior to the classification module.

4.1.3 Putting it all together. *ElasticPdM* takes the multi-modal embedding and predicts machine failure probability. Figure 2 shows the model architecture. The *CNN module* collects the temporal local information from neighboring bins. We define a CNN block as a residual block which contains two sequentially connected 1D convolutional layers with batch normalization, ReLU and dropout. The 1D convolutional module preserves the length of the input sequence for easy stacking with other modules. The total number of CNN blocks in the module is a tunable hyper-parameter (including the value of zero, meaning the absence). The *transformer module* is stacked on top of the CNN module to capture the temporal dependencies in the entire look-back time window. The module takes in positional encoding along with the output from CNN. Each block in this module is comprised of a multi-head self-attention structure [23]. The total number of blocks in the transformer module is also a tunable hyper-parameter. The *temporal pooling module* further pools the transformer embedding to obtain sequence-level representation. We utilize a local average-pooling layer followed by a linear layer, ReLU, batch normalization, and a global max pooling layer for full temporal aggregation. The sequence-level embedding is concatenated with the metadata embedding as input to the classification module. The *classification module* predicts the probability of machine failure using two fully connected layers with batch normalization and ReLU in between.

5 IMPLEMENTATION

Considering the high class imbalance between machine breakdown and normal operation, we utilize class-weighted cross entropy loss to train the *ElasticPdM* model. The class weights are tunable hyper-parameters and designed to adjust for imbalanced distribution between positive (machine failure) and negative (normal operation) classes. We perform the training using Adam optimizer and learning rate decay. We utilize Amazon SageMaker Hyper-parameter Optimization (HPO)[2] to efficiently search the large space of training hyper-parameters, as well as model architectures determined by the number of layers in the CNN and Transformer modules (details about the hyper-parameter search setting in Amazon SageMaker HPO can be found in Appendix A.2). The number of layer parameter includes the option of value 0, meaning the complete absence of the corresponding module. This unified HPO approach allows us to identify the optimal hyper-parameters and model architecture at the same time. SageMaker HPO carries out a Bayesian search with multiple GPU instances in parallel and we evaluate on a held-out validation set. The best model architecture we identified contains two CNN blocks, one Transformer block with four heads. We use Amazon Athena[3] to collect time-windows with random start time for model training and evaluation. The architecture diagram of the model training and inference pipeline can be found in Appendix A.1.

6 EXPERIMENTS

In this section, we present *ElasticPdM*'s performance on the real-world anonymized gaming machine log data provided by L&W.

6.1 Evaluate *ElasticPdM*

6.1.1 Data setup. LnW data contains about 280 million events from ~500 gaming machines accumulated over 10 months, and a short

[2]https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning-how-it-works.html
[3]https://aws.amazon.com/athena/

Table 2: *ElasticPdM* wins: Comparison of *ElasticPdM* and different baseline methods. All results are obtained based on the 48-hour look-back window data. Results are in the format of *mean (std.)* and in *percentage*. Our approach consistently outperforms baseline models on all metrics (compare AutoGluon with *ElasticPdM*-Ensemble, both are ensembled model).

Model	AUROC	AUPRC	ARHP	Recall@80%P	Recall@70%P	Prec@1%	Prec@2%
XGBoost	69.62	43.8	17.82	11.19	17.83	89.58	82.94
LightGBM	68.73	42.24	15.49	9.35	15.01	88.29	81.3
AutoGluon	70.82	45.52	20.26	12.89	19.91	91.35	84.93
CNN	72.13 (0.42)	48.16 (0.48)	24.69 (0.82)	17.37 (0.93)	24.86 (0.88)	90.61 (0.63)	86.66 (1.18)
LSTM	71.07 (0.54)	47.11 (0.70)	23.82 (0.69)	16.03 (0.53)	24.18 (0.88)	88.66 (1.19)	85.38 (0.74)
CNN-LSTM	72.30 (0.08)	48.91 (0.43)	27.05 (1.06)	19.57 (1.55)	27.47 (1.10)	88.73 (1.08)	86.37 (0.46)
Ours: *ElasticPdM*	**73.51** (0.57)	**51.36** (0.25)	**30.57** (0.50)	**23.56** (1.23)	**30.82** (0.37)	90.84 (2.58)	**89.08** (1.41)
Ours: *ElasticPdM*-Ensemble	**74.88**	**53.32**	**32.62**	**25.87**	**32.73**	**92.48**	**90.62**

24-hour time window could contain maximally 29,000 events. We utilize temporal split to construct training, validation, and testing sets to avoid data leakage. In particular, we extract data in the time range 10/1/2021 - 6/16/2022 for training, 6/16/2022 - 7/1/2022 for validation, and 7/1/2022 - 7/31/2022 for testing. All the results are reported on the test set. For each data set, we randomly sample the timestamps as the prediction time t, and extract the corresponding look-back windows, as well as ground-truth prediction labels based on the presence of machine breakdown in the look-ahead window. The train-validation-test ratio is 7:1:2 with a total size of 1 million. We experiment different look-back window length w_b and find that increasing window length significantly improves model performance (detail in Table 5). For comparative study with baseline models as well as ablation studies, we use $w_b = 48$ hours for all methods.

6.1.2 Baselines. Our baselines include decision tree-based models, which are widely used in PdM literature and neural network models including CNN, Long short-term memory (LSTM) and a combination of CNN and LSTM. For decision tree-based models, we utilize AutoGluon [10] which helps efficiently benchmark against XGBoost [7] and LightGBM [15]. AutoGluon automates data preprocessing, hyper-parameter tuning, and model ensemble with AutoML. We feed the following features into AutoGluon for model training and evaluation: machine metadata, look-back window start time and end time, binned event frequencies concatenated into a flattened feature vector[4]. For neural network models including CNN, LSTM and CNN-LSTM, we use Amazon SageMaker HPO to find the best training and architecture hyper-parameters. The HPO search setting as well as the best hyper-parameter can be found in Appendix A.2 and Section 6.1.3.

6.1.3 Hyperparameter settings for neural networks. The following hyper-parameters are shared: batch size is 256, dropout rate is 0.25, learning rate decay factor 0.316, learning rate decay steps 20.

CNN: kernel size is 7, dimension of model is 160, number of bins is 120, number of CNN blocks is 1, class weight in loss function is 0.2, learning rate is 1e-4.

LSTM: LSTM output size is 128, dimension of model is 320, number of bins is 120, number of LSTM blocks is 1, class weight in loss function is 0.5, learning rate is 1e-4.

CNN+LSTM: kernel size is 7, LSTM output size is 128, dimension of model is 640, number of bins is 120, number of CNN blocks is 9, number of LSTM blocks is 2, class weight in loss function is 0.213422, learning rate is 1e-4.

ElasticPdM : CNN kernel size is 7, number of attention heads is 4, dimension of model is 320, number of bins is 120, number of CNN blocks is 2, number of Transformer block is 1, class weight in loss function is 0.35, learning rate is 1e-3.

6.1.4 Evaluation metrics. In our predictive maintenance use case with L&W, precision is considered more important than recall, since any unnecessary maintenance due to false alarm will drive up the cost. To measure model performance under high precision, we introduce a new metric, named Average Recall at High Precision (ARHP). ARHP is defined as the averaged recall rate at 60%, 70%, and 80% precision. ARHP is used as the most important metric by L&W to evaluate the PdM model. We also report precision at top K% (Prec@K%, K=1,2), which evaluates the precision among the top K% instances ranked by the predicted probability. Additionally, we also report AUROC (Area Under the Receiver Operating Characteristic) and AUPRC (Area Under the Precision-Recall Curve) as the supporting metrics. For neural network based models, we repeat each experiment for five times with random weight initialization and report the mean and standard deviation (std.) of the results[5]. We note that a high priority is put in obtaining model that meets the requirements from the end-users, instead of defining the success metric on our own. We decided to prioritize the ARHP metric after extensive discussion with our collaborators that are responsible for producing and maintaining the machines. While respecting the end-user preference our framework is also flexible to be adjusted to optimize an alternative metric, if the use case changes, and we believe this is one of the advantages of our system (by changing the HPO code block illustrated in Appendix A.2).

6.1.5 Baseline comparison results. Table 2 summarizes the experimental results from baselines and our *ElasticPdM* model. *ElasticPdM* outperforms all baseline methods. For example, it achieves almost 83% relative improvement in Recall@80%Precision, and 50.9% relative improvement in ARHP compared to AutoGluon. Compared with CNN-LSTM, the best model among the baseline techniques, *ElasticPdM* achieves 20.4% relative improvement in Recall@80%Precision and 13.0% relative improvement in ARHP. This

[4]Due to the large number of features after concatenation of bins, we can maximally use 12 bins in AutoGluon without crashing the memory (ml.r5.24xlarge instance on Amazon SageMaker, 768GB RAM).

[5]AutoGluon is ensemble-based and has negligible std.

means that *ElasticPdM* can capture significantly more malfunctions before they occur. Model ensembling further improves the performance, where we used the mean ensemble of five independently initialized *ElasticPdM* models. For all NN-based models we use 48 hour look-back time window with 120 evenly sized bins. Figure 3 further shows that using *ElasticPdM* with *time-adaptive event sequence binning* can achieve performance on par with evenly sized 120 bins with significantly less amount of bins (60) and 50% less training time.

6.1.6 Ablation study results. In this section, we report results from detailed ablation studies on the following components of *ElasticPdM*: time-adaptive sequence binning (Section 4.1.1), time-aware event sequence embedding and metadata embedding (Section 4.1.2), look-back window length w_b (Section 3).

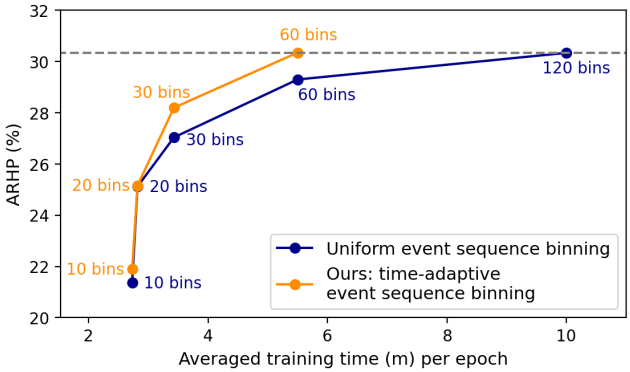

Figure 3: Using *time-adaptive event sequence binning*, the performance improvement is much faster and more significant when increasing the number of bins. With 60 bins, time-adaptive binning is able to achieve ARHP on-par with 120 uniform bins, which reduces 50% of the training time.

Effect of time-adaptive event sequence binning. We study how time-adaptive event sequence binning can improve model performance with lower computational footprint. Figure 3 compares uniform event sequence binning with time-adaptive event sequence binning under different bin number settings, and plots ARHP against the training time needed per epoch. Notice that for both uniform and time-adaptive binning settings, the model performance consistently improves with increased bin numbers and plateaus later on. However, with time-adaptive event sequence binning, the performance improvement is much faster and more significant. With 60 bins, time-adaptive binning is able to achieve ARHP on-par with 120 uniform bins, which reduces 50% of the training time needed. The following settings in Table 4 are used to conduct the ablation study on the effect of time-adaptive event sequence binning and the number of bins in Section 6.1.6:

Effect of time-aware event sequence embedding and metadata embedding. In Table 3, we compare *ElasticPdM* with its variants: ***ElasticPdM*$_{\text{-T}}$** excludes timestamp embedding in event sequence data; ***ElasticPdM*$_{\text{-M}}$** excludes metadata embedding. We observe that the integration of timestamp embedding and metadata embedding greatly improves model performance, especially on the ARHP metric, and the combination of all modalities achieves the best performance. The result demonstrates the effectiveness and necessity of our multi-modal embedding design.

Effect of different look-back window lengths. We perform ablation study on the look-back window length (Section 3) to investigate how it affects the model performance. Table 5 show that increasing the window length significantly improves model performance, as they contain more historical information to help with the prediction. All the results are obtained using same hyper-parameter settings described in Section 6.1.3 with 120 evenly sized bins. We plan to evaluate even longer look-back windows in the next steps as L&W deploys *ElasticPdM*.

7 CONCLUSION AND DISCUSSION

In this paper, we propose *ElasticPdM*, a novel modeling framework for real-world, industry-scale predictive maintenance use cases using machine event log data. *ElasticPdM* contains two critical novel designs for performance and scalability, i.e. time-aware event sequence embedding and time-adaptive temporal binning. *ElasticPdM* significantly outperforms strong baselines on real-world machine log data with 280 million events from ~500 machines.

ElasticPdM is under active deployment at Light&Wonder to provide early warnings for predictive maintenance of large-scale gaming machines. L&W is also looking into collecting more machine telematics data to further improve predictive performance. Such data could include but is not limited to: machine CPU temperature, machine vibrations and etc.

8 ACKNOWLEDGMENTS

We would like to thank Evan McMillian, Shane Rai and Raj Salvaji from Amazon AWS, as well as Anand Singh and Christopher Alvarez from Light & Wonder for their generous support and partnership.

REFERENCES

[1] 2023. Amazon Lookout for Equipment. https://aws.amazon.com/lookout-for-equipment/features/

[2] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen, and Y. Wang. 2019. GluonTS: Probabilistic Time Series Modeling in Python. *arXiv preprint arXiv:1906.05264* (2019).

[3] Azzeddine Bakdi, Nicolay Bjørlo Kristensen, and Morten Stakkeland. 2022. Multiple Instance Learning With Random Forest for Event Logs Analysis and Predictive Maintenance in Ship Electric Propulsion System. *IEEE Transactions on Industrial Informatics* 18, 11 (2022), 7718–7728.

[4] João Bento, Pedro Saleiro, André F Cruz, Mário AT Figueiredo, and Pedro Bizarro. 2021. TimeSHAP: Explaining recurrent models through sequence perturbations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2565–2573.

[5] Fatih Camci and Ratna Babu Chinnam. 2010. Health-state estimation and prognostics in machining processes. *IEEE Transactions on automation science and engineering* 7, 3 (2010), 581–597.

[6] Thyago P Carvalho, Fabrízzio AAMN Soares, Roberto Vita, Roberto da P Francisco, João P Basto, and Symone GS Alcalá. 2019. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering* 137 (2019), 106024.

[7] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[8] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20 (1995), 273–297.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (2020).

Table 3: Time-aware event sequence embedding and metadata embedding are both critical to obtain the most performant model. Results are in the format of *mean (std.)* and in *percentage*.

Model	AUROC	AUPRC	ARHP	Recall@80%P	Recall@70%P	Prec@1%	Prec@2%
ElasticPdM$_{-\mathrm{T}}$	71.90 (0.30)	47.16 (0.39)	22.80 (0.33)	15.37 (0.56)	23.03 (0.17)	90.63 (1.56)	86.09 (1.02)
ElasticPdM$_{-\mathrm{M}}$	70.92 (0.22)	43.52 (0.37)	15.62 (0.96)	8.92 (0.56)	15.52 (1.28)	85.07 (1.50)	80.41 (0.58)
ElasticPdM	73.51 (0.57)	51.36 (0.25)	30.57 (0.50)	23.56 (1.23)	30.82 (0.37)	90.84 (2.58)	89.08 (1.41)

Table 4: Time-adaptive event sequence binning settings

Setting ID	B	s	l	r
1	10	2	0.4	2.4477
2	20	5	0.4	2.4433
3	30	5	0.4	1.5609
4	60	6	0.4	1.1469

Table 5: Longer look-back time window improves model performance significantly. Results are in *percentage*.

Look-back window	AUPRC	ARHP	Recall@80%P	Recall@70%P
6 hours	45.37	17.91	12.1	17.32
12 hours	46.5	18.47	12.30	18.10
24 hours	49.02	22.7	15.6	21.63
48 hours	51.36	30.57	23.56	30.82

[11] Antoine Guillaume, Christel Vrain, and Elloumi Wael. 2020. Time series classification for predictive maintenance on event logs. *arXiv preprint arXiv:2011.10996* (2020).

[12] Abdul Basit Hafeez, Eduardo Alonso, and Aram Ter-Sarkisov. 2021. Towards Sequential Multivariate Fault Prediction for Vehicular Predictive Maintenance. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 1016–1021.

[13] Deokwoo Jung, Zhenjie Zhang, and Marianne Winslett. 2017. Vibration analysis for iot enabled predictive maintenance. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 1271–1282.

[14] Dovile Juodelyte, Veronika Cheplygina, Therese Graversen, and Philippe Bonnet. 2022. Predicting Bearings Degradation Stages for Predictive Maintenance in the Pharmaceutical Industry. (2022), 3107–3115.

[15] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[16] Kedar Kulkarni, Umamaheswari Devi, Amith Sirighee, Jagabondhu Hazra, and Praveen Rao. 2018. Predictive maintenance for supermarket refrigeration systems using only case temperature data. In *2018 Annual American Control Conference (ACC)*. IEEE, 4640–4645.

[17] YG Li and P Nilkitsaranont. 2009. Gas turbine performance prognostic for condition-based maintenance. *Applied energy* 86, 10 (2009), 2152–2161.

[18] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. 2021. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting* 37, 4 (2021), 1748–1764.

[19] Alexander Nikitin and Samuel Kaski. 2022. Human-in-the-Loop Large-Scale Predictive Maintenance of Workstations. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3682–3690.

[20] Yongyi Ran, Xin Zhou, Pengfeng Lin, Yonggang Wen, and Ruilong Deng. 2019. A survey of predictive maintenance: Systems, purposes and approaches. *arXiv preprint arXiv:1912.07383* (2019).

[21] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. 2014. Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*. 1867–1876.

[22] Huan Song, Deepta Rajan, Jayaraman Thiagarajan, and Andreas Spanias. 2018. Attend and diagnose: Clinical time series analysis using attention models. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[24] DeLiang Wang and Jitong Chen. 2018. Supervised speech separation based on deep learning: An overview. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26, 10 (2018), 1702–1726.

[25] Xuejing Zhao, Mitra Fouladirad, Christophe Bérenguer, and Laurent Bordes. 2010. Condition-based inspection/replacement policies for non-monotone deteriorating systems with environmental covariates. *Reliability Engineering & System Safety* 95, 8 (2010), 921–934.

[26] Shuai Zheng, Kosta Ristovski, Ahmed Farahat, and Chetan Gupta. 2017. Long short-term memory network for remaining useful life estimation. In *2017 IEEE international conference on prognostics and health management (ICPHM)*. IEEE, 88–95.

[27] Tiago Zonta, Cristiano André Da Costa, Rodrigo da Rosa Righi, Miromar Jose de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. 2020. Predictive maintenance in the Industry 4.0: A systematic literature review. *Computers & Industrial Engineering* 150 (2020), 106889.

A *ELASTICPDM* CONFIGURATIONS

A.1 Training and Inference Architecture

Figure 4 illustrates solution architecture for predictive maintenance of gaming machines at Light&Wonder. The training pipeline illustrated in the top half of the figure uses Amazon S3[6] to store historical event logs streamed from remote gaming machines. The event logs are stored in partitions using their timestamps to allow for faster data query via Amazon Athena. Using Amazon Athena, we obtained randomly sampled windows from train, validation and test periods and store the data for training in S3. Amazon SageMaker Notebook is used by scientists to initiate HPO / training jobs running in parallel on multiple GPU instances. The model is then deployed as a SageMaker inference endpoint. The inference pipeline illustrated in the bottom half of the figure supports the following workflow, (assuming that latest data is continuously updated in S3): 1) user requests inference on the probability of machine failures at a selected timestamp in a dashboard; 2) data extraction request is triggered to load data from Amazon S3 using Athena; 3) extracted data is saved back to Amazon S3; 4) newly saved data triggers AWS Lambda[7] function; 5) lambda function triggers Amazon SageMaker batch transform for inference; 6) predicted data from SageMaker Transform job is saved back to S3; 7) prediction results are visualized in the dashboard.

A.2 HPO Settings in Amazon SageMaker

We setup the HPO training jobs on g4dn.8xlarge GPU instances. A total of 300 jobs (i.e., hyper-parameter settings) was experimented with the Bayesian optimization algorithm to search for the best validation ARHP. A maximum of 15 parallel jobs was allowed. We focused on the model hyper-parameters, such as the number of layers in different modules, dimension of the model; and training hyper-parameters, such as learning rate, class weight in the

[6] https://aws.amazon.com/s3/

[7] https://aws.amazon.com/lambda/

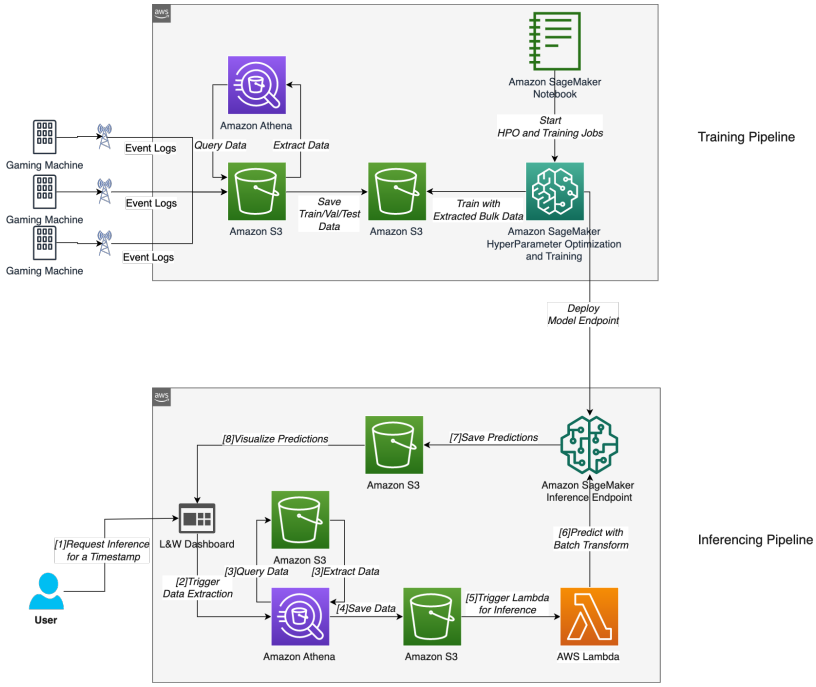

Figure 4: Solution architecture on AWS for predictive maintenance of gaming machines at Light&Wonder. The top half illustrates the training pipeline and the bottom half illustrates the inference pipeline.

loss function, etc. The HPO script contains the range for each hyperparameter being searched in *ElasticPdM* as well as some fixed hyperparameter settings.

```
import sagemaker
from sagemaker.tuner import (
    IntegerParameter,
    CategoricalParameter,
    ContinuousParameter,
    HyperparameterTuner,
)
from sagemaker.pytorch import PyTorch

# define training setuo
estimator = PyTorch(
    entry_point=os.path.join(DIR,
        'hpo_train.py'), # entry script
    ... # other settings
    py_version="py3",
    instance_count=1,
    instance_type="ml.g4dn.4xlarge",
    hyperparameters={"num_epochs": 50,
                     "num_workers":8,
                     "batch_size":256,
                     "time_window_length":48
                     },
    max_run=432000,
)

# define hyperparameters and search ranges
hyperparameter_ranges = {
    "fc1_out_dim": CategoricalParameter([20,40,80,160]),
    "learning_rate": ContinuousParameter(5e-4, 1e-3,
        scaling_type="Logarithmic"),
    "loss_weight": ContinuousParameter(0.1, 0.9),
    "num_bins": CategoricalParameter([10, 40, 60,
        120, 240]),
    "dropout_rate": CategoricalParameter([0.1, 0.2,
        0.3, 0.4, 0.5]),
    "dim_model": CategoricalParameter([160,320,480,640]),
    "num_cnn_layers": IntegerParameter(0,10),
    "cnn_kernel": CategoricalParameter([3,5,7,9]),
    "num_transformer_layers": IntegerParameter(0,4),
    "num_heads": CategoricalParameter([4,8]),
    "num_rnn_layers": IntegerParameter(0,10), # optional
    "dim_rnn":CategoricalParameter([128,256])
}

# define the name of objective metric
objective_metric_name = "dev ARHP"
# define the metric strategy
objective_type = "Maximize"
# define the regex to capture
# the objective metric from the model training log
metric_definitions = [{"Name": "dev ARHP",
    "Regex": "Current ARHP: (.*?),"}]

# define the hyperparameter tuner
tuner = HyperparameterTuner(
    estimator,
    objective_metric_name,
    hyperparameter_ranges,
    metric_definitions,
    max_jobs=300,
    max_parallel_jobs=15,
    objective_type=objective_type,
)

# start tuning
tuner.fit({"training": inputs})
```