applied sciences

MDPI

Article

Deep Learning-Based Log Parsing for Monitoring Industrial ICT Systems

Yuqian Yang, Bo Wang and Cong Zhao *

National Engineering Laboratory for Big Data Analytics, Xi'an Jiaotong University, Xi'an 710049, China; yuqian.yang@stu.xjtu.edu.cn (Y.Y.)
* Correspondence: congzhao@xjtu.edu.cn

Abstract: For rapidly developing smart manufacturing, Industrial ICT Systems (IICTSs) have become critical to safe and reliable production, and effective monitoring of complex IICTSs in practice is necessary but challenging. Since such monitoring data are organized generally as semi-structural logs, log parsing, the fundamental premise of advanced log analysis, has to be comprehensively addressed. Because of unrealistic assumptions, high maintenance costs, and the incapability of distinguishing homologous logs, existing log parsing methods cannot simultaneously fulfill the requirements of complex IICTSs simultaneously. Focusing on these issues, we present LogParser, a deep learning-based framework for both online and offline parsing of IICTS logs. For performance evaluation, we conduct extensive experiments based on monitoring log sets from 18 different real-world systems. The results demonstrate that LogParser achieves at least a 14.5% higher parsing accuracy than the state-of-the-art methods.

Keywords: Industry 4.0; monitoring; log parsing; log analysis; deep learning

Citation: Yang, Y.; Wang, B.; Zhao, C. Deep Learning-Based Log Parsing for Monitoring Industrial ICT Systems. *Appl. Sci.* **2023**, *13*, 3691. https://doi.org/10.3390/app13063691

Academic Editor: Andrea Prati

Received: 23 December 2022
Revised: 9 March 2023
Accepted: 10 March 2023
Published: 14 March 2023

1. Introduction

The recent development of information and communication technologies (ICT) has promoted the progression of the industrial revolution from digital to intelligence (i.e., Industry 4.0 [1]). Industrial ICT systems (IICTSs) have been demonstrated to be a critical component of smart manufacturing, as their failure causes catastrophic consequences, such as manufacturing accidents and financial disasters [2]. Drawing increasing interest from both industry and academia, effective monitoring holds great significance in preventing IICTSs from variable and intense physical and cyber threats in practice [3].

Generally, the IICTS monitor aims to detect anomalies and predict threats by collecting and analyzing operational information about system components [3]. Such monitoring data are usually formed as *logs*, in which data mining methods are commonly applied to knowledge extraction, i.e., log mining. Conventional log mining approaches have three basic steps, i.e., *log parsing*, *matrix generation*, and *data mining* [4]. As the essential premise of effective mining, log parsing identifies system-specified events according to the corresponding logs. An example of log parsing is shown in Figure 1; the corresponding even labels, such as *super user action*, *informative report*, *event report*, etc., are generated for the next data mining tasks.

Existing log parsing methods usually depend on conventional data mining techniques, such as clustering and iterative partitioning [4], and a few methods leverage program source code, which is hardly available in practice [5]. However, parsing logs from complex IICTSs have the following requirements, which cannot be satisfied simultaneously by the current solutions:

- *Generalization*: The parser should be applicable to all logs with different textual properties. Conventional log parsing methods are based generally on the assumption that words with high-presence frequencies in logs imply corresponding types of events,

which, however, may not hold for IICTS logs. For example, according to our evaluation results, such an assumption does not hold on the publicly available Web Log dataset [6]. A parser with no log textual property requirement is necessary for IICTSs.

- *Low human labor cost*: The parser should cost less human labor while processing multi-sourced logs. Conventional log parsing methods require a set of parameters tuned for logs from each log source (i.e., a system component/device), which is quite expensive considering the massively different log sources in practical IICTSs, especially when the system needs to be continuously maintained. For example, logs collected from the ThingWorx platform [7] contain 87 components' logs. It is cost-effective to use a parser specifically designed for such a multi-sourced log scenario, which just requires affordable human labor.
- *Transferability*: The parser should be able to satisfy various task-specified parsing requirements on logs (from the same source or different sources) with similar structures (i.e., homologous logs). Conventional log parsing methods are basically driven by log textual properties, and no information regarding successive mining tasks is considered. In fact, for log mining, the parsing results of similar logs in different mining tasks are obviously different, especially in IICTSs with multiple services. It is of great significance to construct a parser that is capable of adjusting to different tasks and producing task-specified parsing results on homologous logs.

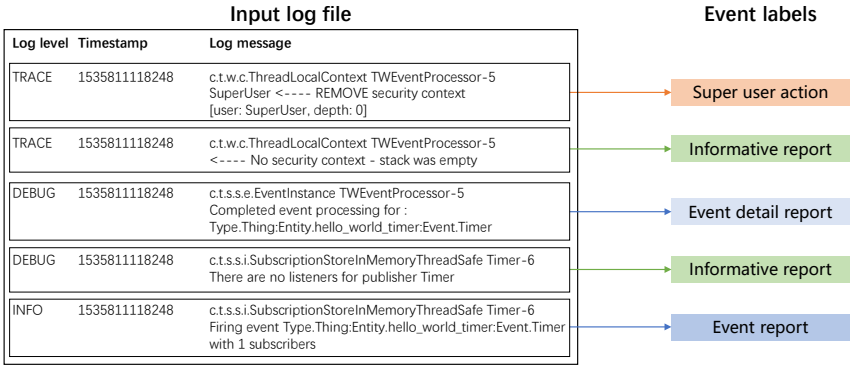

Figure 1. A log parsing sample: generating corresponding event label according to log records.

Treating all logs from an IICTS as *an aggregation of multiple runtime event sequences*, we present LogParser, a deep learning (DL) based log parsing framework for both online and offline log analysis. Our contributions can be briefly summarized as follows:

1. We construct both offline and online DL-based parsers with no specific textual property requirement for IICTS logs. Our approach can handle different task-specified log parsing requirements on both heterogeneous and homologous logs.
2. LogParser unifies *log parsing* and *matrix generation* for consistent event labeling across log mining tasks without task-specific preprocessing.
3. LogParser (code and data are available at github.com (https://github.com/jas0n-bot/LogParser) accessed on 4 August 2020) achieves higher parsing accuracies than the state-of-the-art methods on 18 real-world log sets. Specifically, our online parser has a negligible accuracy loss compared to our offline parser and outperforms the existing methods on heterogeneous/homologous multi-source log sets.

The rest of the article is organized as follows. The related works are presented in Section 2. Section 3 provides the preliminaries of our work. Section 4 describes our methodology. Our DL-based log parsing approach is explicitly presented in Section 5. Section 6 demonstrates our evaluation methodology and results. We conduct a discussion on parameter settings in Section 7. Section 8 concludes the article.

2. Related Work

Log parsing is a fundamental issue in the operation of ICT systems in practice. Previous studies on log parsing have applied various clustering algorithms based on different criteria such as word frequency or feature words. However, these methods often suffer from low accuracy, scalability, or robustness. Heuristic algorithms have also been proposed to address some of these issues, but they require manual tuning and domain knowledge. Recently, machine learning techniques have been explored for log parsing, which can potentially overcome some of the drawbacks of traditional methods.

One of the earliest efforts is the Simple Logfile Clustering Tool (SLCT) [8], which sets the tone for successive clustering-based methods. It uses the frequency of fixed words (tokens) in the log text as the basis for clustering. LogSig [9] uses word pairs as the feature of one log record and then aggregates them into groups. The log template of each group is generated based on the common pairs. LKE [10] takes the strategy that two clusters of log records are simply joint if their distance is below a particular threshold, which suffers from an obvious accuracy loss on complex log sets.

Other than the conventional clustering-based methods, many log parsers use heuristic algorithms to analyze logs. IPLoM [11] heuristically parses logs by record length, which, however, requires careful preprocessing since inappropriate preprocessing may cause incorrect splitting. Drain [12] enhances IPLoM by maintaining a trie (prefix tree) for faster searching, which, however, does not address the problem caused by the length-based parsing in IPLoM. POP [13] aims to solve this issue by merging groups in the last step, which surely improves the parsing recall and the overall parsing accuracy.

The natural language processing (NLP)-based log parser [14] uses latent Dirichlet allocation (LDA) to perform the classification after the tokenization, semantic processing, vectorization, and model compression steps. LogDTL [15] constructs a deep transfer neural network model for log template generation, which applies the transfer learning technique for data training augmentation purposes. NuLog [16] employs masked-language modeling to vectorize input tokens with random masking. A two-layer neural network encoder processes the vectorized input, and a Softmax linear layer produces a result matrix that maps to event templates.

However, unlike LogParser, none of the methods above manages to address the issue of parsing logs from multiple sources, especially from homologous data sources with a different parsing task.

3. Preliminary

For ease of understanding, we briefly introduce some basic concepts in terms of DNN design and text data processing used in the rest of the article.

3.1. DNN Layers

Word2vec [17] acts as the *word embedding* layer in a DNN. It is a two-layer neural network that learns the contextual relationship of a certain word in a corpus of text. Specifically, we use the continuous bag-of-word (CBOW) model, which uses the surrounding context words to predict the current word.

Dropout [18] is a simple and effective filtering layer that reduces redundant information, which can improve a DNN's performance in terms of generalization. A typical usage is to apply a dropout layer before a dense input layer.

The **long short-term memory** (LSTM) [19] network is a kind of recurrent neural network (RNN) that can remember long-term information over several steps back. It has been successfully applied to fields such as language translation, speech recognition, and chat robots.

3.2. Batch Learning and Online Learning

In the field of machine learning, **batch learning**, the actual learning method used in offline learning, takes in the entire training set and generates a model after a certain number

of training iterations. In each iteration, all samples in the training set are used. The model is evaluated according to its performance on a separate testing set. **Online learning**, however, does not require access to the entire training set. It continuously consumes samples from the input stream. The model is iteratively optimized based on each incoming sample. The model is evaluated according to its performance on all consumed samples.

4. Methodology

Real-world log parsing tasks require both batch and stream processing modes, where the former parses historical logs, and the latter parses continuously generated logs. Considering this, we design a deep learning-based framework comprising both offline and online parsers for batch and stream processing, respectively. To achieve the three aforementioned requirements of IICTS log parsing, we have the following specific designs:

- **Log parsing consistency:** For *generalization*, the event labels teach our parsers the parsing standards directly without task-specific data preprocessing. Additionally, the online parser needs to follow the same parsing standards as the offline parser. Our framework, as shown in Figure 2, ensures this by periodically importing the offline parser models into the online parser.
- **Quality of log record representations:** To lower *the human labor cost* of our method (mainly for labeling log records), we significantly reduce the number of labeled records required for model training by enhancing the quality of record representations. In particular, we use word2vec [17] to vectorize the log records based on a neural network trained on the corpus of all log text.
- **Supervised learning by event labels:** Our method leverages event labels directly to train our neural network model, which enables it to adjust itself for different parsing criteria. In this way, our method achieves the *transferability* that is needed for various log parsing tasks.

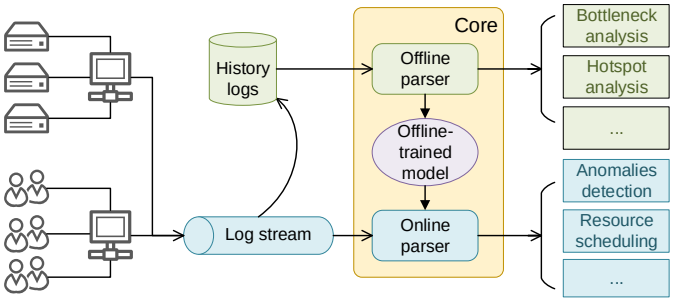

Figure 2. Deep learning-based log parsing framework.

Since log parsing influences all subsequent log analyses and mining tasks, log parsing accuracy should be considered the key metric. In particular, we have the following designs to improve the accuracy of offline and online log parsing:

Accuracy of offline parsing: To attain high offline *parsing accuracy*, we design an LSTM-based offline parser that exploits the natural contextual relevance of log messages, which aids in recognizing the essential word representations. Our method leverages vectorized log records, which are word representations in a unified semantic space.

Adaptability of online parsing: For stream processing, to achieve high *parsing accuracy*, online parsing needs to cope with the challenge of concept drift, i.e., the dynamic changes in patterns in the data stream over time. To address this challenge, we propose an online parser that incorporates BiLSTM [20] and attention mechanisms [21] to enhance its adaptability to new data patterns.

5. The Deep Learning-Based Log Parsing Method

In this section, we explicitly present our deep learning-based log parsing architecture, LogParser.

5.1. Overview

The general architecture of LogParser is shown in Figure 2, whose workflow can be described as follows. Firstly, all logs from the monitored IICTS are aggregated as a *stream pip*, where each incoming log is replicated and stored in a *historical log database*. Then, the *offline parser*, which is described in detail in Section 5.3, completes the parsing task by outputting a matrix representing all processed logs and corresponding event labels. Such results are used by successive mining tasks, such as bottleneck analysis [22], daily check analysis [23], etc. Cooperating with the offline parser, the *online parser*, which is described in detail in Section 5.4, periodically loads the latest offline model, and directly consumes logs from the stream pip. It refines the model through online learning, which adaptively learns different log events. Similar to the offline parser, the online parser generates a vector representing the processed log and a label of the corresponding event type. Such results are used by online analyses, such as anomaly detection [24], resource scheduling [25], etc.

5.2. Word Representation

In LogParser, we use *word2vec* [17] to vectorize input logs. Specifically, given a log record composed of a sequence of T words $x = \{x_1, x_2, \ldots, x_T\}$, each word x_t $(1 \leq t \leq T)$ is converted as a real-valued n^{w2v}-dimensional vector $\boldsymbol{e}_t$ according to the embedding matrix $W^{w2v} \in \mathbb{R}$. Here, $\mathbb{R}$ represents the full vocabulary, and W^{w2v} is a size n^{w2v} word embedding learned by a *word2vec* neural network. We use the full log dataset to train our *word2vec* model, where the *CBOW* model is selected due to its faster training speed.

5.3. Offline Parser

As shown in Figure 3, as a specifically designed *recurrent neural network* (RNN), our offline parser has three major components:

- *A word embedding layer* that transforms x to $\boldsymbol{e}$;
- *An LSTM layer* that extracts features from log records;
- *A decision layer* that identifies the log event type.

Additionally, to enhance the generalization capability, we integrate two *dropout layers* into the offline parser, i.e., one between the word embedding layer and the LSTM layer and another between the LSTM layer and the decision layer.

Fundamentally, the core layer of our offline parser can be formalized as follows:

$$h_t = H(W_{e'h}e'_t + W_{hh}h_{t-1} + b_h), \quad (1)$$

$$o_t = W_{ho}h_t + b_o \quad (2)$$

where x is the input log record, i.e., the word sequence; h is the hidden vector sequence; o is the output sequence of the LSTM layer; the W terms denote the weight matrices; H is the hidden layer function; and the b terms denote the bias vectors.

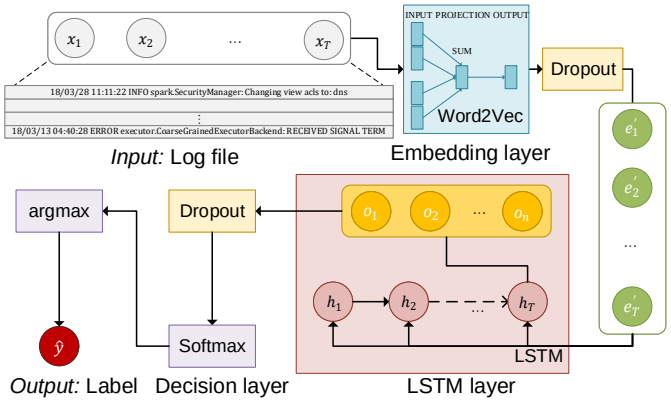

Figure 3. Deep learning-based log parsing framework: offline architecture.

Specifically, our hidden layer function H is designed as a standard LSTM, which is built upon LSTM cells with four major components, i.e., input gate i, forget gate f, output gate o, and memory cell c. Explicit cell implementation is as follows:

$$i_t = \sigma(W_{e'i}e'_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \quad (3)$$
$$f_t = \sigma(W_{e'f}e'_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \quad (4)$$
$$c_t = f_t c_{t-1} + i_t \tanh(W_{e'c}e'_t + W_{hc}h_{t-1} + b_c), \quad (5)$$
$$o_t = \sigma(W_{e'o}e'_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \quad (6)$$
$$h_t = o_t \tanh(c_t), \quad (7)$$

where σ is the sigmoid function.

Then, the LSTM layer's output o is transformed into o' by the dropout layer. Finally, the decision layer uses the Softmax function to calculate the probability distribution over all log event types y as well as the predicted label $\hat{y}$:

$$y = softmax(W_{o'y}o' + b_y), \quad (8)$$
$$\hat{y} = \underset{i}{\text{argmax}}(y_i). \quad (9)$$

One thing that should be noted is that, different from other kinds of data for mining, a single log record contains limited words. Therefore, we believe that *our offline parser can be effectively trained upon a small training set*, which will be verified further and discussed in Section 6.

5.4. Online Parser

Since our online parser focuses on directly consuming streaming log records (i.e., one at a time), it has to learn as quickly as possible. To address this issue, we use attention-based BiLSTM to construct our online parser.

Specifically, as shown in Figure 4, our online parser has four major components, i.e., *a word embedding layer*, *a BiLSTM layer*, *an attention layer*, and *a decision layer*. Two *dropout layers* are placed before the BiLSTM and decision layers, respectively. The explicit component descriptions are as follows.

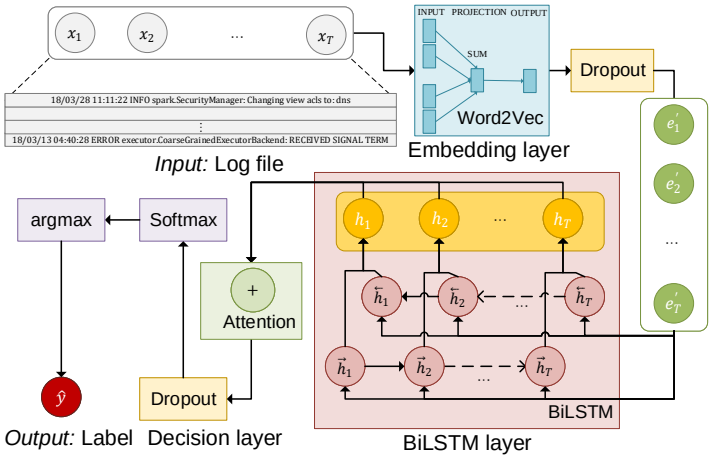

Figure 4. Deep learning-based log parsing framework: online architecture.

To start, the input log record x is converted as e' through the word embedding and dropout layers. Then, let $\overrightarrow{h}$ and $\overleftarrow{h}$ denote the forward and backward hidden layers, respectively. The BiLSTM layer in our online parser can be formally described as follows:

$$\overrightarrow{h}_t = H_{lstm}(e'_t, \overrightarrow{h}_{t-1}, b_{\overrightarrow{h}}), \tag{10}$$

$$\overleftarrow{h}_t = H_{lstm}(e'_t, \overleftarrow{h}_{t-1}, b_{\overleftarrow{h}}), \tag{11}$$

$$h_t = \overrightarrow{h}_t \bigoplus \overleftarrow{h}_t, \tag{12}$$

where $\bigoplus$ denotes the element-wise sum operation.

Then, an attention model (or mechanism) is added between the BiLSTM and decision layers. Specifically, our attention model consumes h and adjusts the weights of the BiLSTM output as follows:

$$g_t = \tanh(W_{hg} h_t + b_g), \tag{13}$$

$$\alpha_t = softmax(w^T g_t), \tag{14}$$

$$o = \sum_{t=1}^{T} \alpha_t h_t. \tag{15}$$

Finally, the decision layer takes the re-calculated and filtered output o' from the attention model to determine the log event type $\hat{y}$ (similar to our offline parser).

One thing that should be noted is that, since the training of BiLSTM is quite time-consuming, we train our online parser only based on falsely predicted records to reduce the training time.

6. Evaluation and Results

In this section, we evaluate the performance of LogParser. We first provide an explanation of our experimental methodology and then present the corresponding evaluation results of our offline and online parsers, respectively.

6.1. Experimental Methodology

Selected Datasets: For a comprehensive performance evaluation, we selected 18 different log sets collected from different practical systems, as shown in Table 1, where 'Log Size' indicates the number of log records, and 'Events' indicates the number of log event types. Specifically, *HDFS* [5] is generated in a private cloud environment using a benchmark workload. *BGL* [26] is a supercomputer log set collected at Lawrence Livermore National Labs (LLNL) in Livermore, CA. *Spark* is collected from our three-node cluster server that hosts our lab's calculation tasks. *Apache* is an open-source Apache server access log set. *UofS* [27] is a trace set containing all HTTP requests to the WWW server at the University of Saskatchewan within seven months. *Jul95* [27] is a trace set containing all HTTP requests to the WWW server at NASA Kennedy Space Center in Florida, within two months. *Nginx* is an open-source Nginx server access log set. *Openstack* [28] is generated in CloudLab. *Security Log* [29] contains connection logs collected and used by Security Data Analysis Labs. *Thunderbird* [26] is a supercomputer log set collected from Sandia National Labs (SNL) in Albuquerque, NM. *Big Brother* [30] is a diagnostic log set from IEEE VAST Challenge 2013. *Web Log* [6] is a publicly available web log set. *ThingWorx* [7] contains system runtime logs collected from the ThingWorx platform, an Industrial Internet of Things (IIoT) platform, and *4SICS-151020, 4SICS-151021, and 4SCIS-151022* [31] are network traffic data captured from industrial network equipment by the industrial cyber security conference 4SICS. *EMSD* includes 30 days of Energy Management System logs (https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets accessed on 4 August 2020). *IoT sentinel* [32] contains the traffic emitted during the setup of 31 smart home IoT devices.

Table 1. Summary of evaluation datasets.

Logs	Log Size	Events	Description
HDFS	11197705	25	Hadoop runtime log
BGL	4747963	155	HPC Blue Gene/L runtime log
Spark	98671134	41	Spark runtime log
Apache	1643182	28	Apache server access log
UofS	2408625	9	University of Saskatchewan WWW server access log
Jul95	1891714	7	NASA Kennedy Space Center WWW server access log
Nginx	1067357 1067459 1067357	4 2 20	Nginx server access log
Openstack	189386	17	Openstack runtime log
Security Log	22694356	10	Connection log from Security Data Analysis Labs
Thunderbird	211212192	40	HPC Thunderbird runtime log
Big Brother	68666	8	Big Brother diagnostic log data
Web Log	1125760	15	Security Repo access log
ThingWorx	1849361	12	ThingWorx platform log
4SICS-151020	246137	15	4SICS Geek Lounge Pcaps
4SICS-151021	1253100	19	4SICS Geek Lounge Pcaps
4SICS-151022	2274747	29	4SICS Geek Lounge Pcaps
EMSD	5764522	68	Energy Management System log
IoT sentinel	129371	19	IoT devices' captures

Comparisons: We compared the performance of LogParser with that of multiple log parsing methods, i.e., *IPLoM* [11], *Drain* [12], *POP* [13], and *NuLog* [16]. All comparatives are implemented in Python. For POP, we used the single-node version. All parameters were set first as recommended and then tuned according to the actual performance.

Evaluation Metric: We used *the F1-score*, or the harmonic mean of the *precision* and *recall*, to evaluate the log parsing performance. Specifically, for a certain type of event, precision is defined as the number of correctly predicted records divided by the number of all records identified as such a type of event, and recall is defined as the number of correctly predicted records divided by the number of all records actually labeled as such a type of event (i.e., the ground truth).

Implementation: For the implementation of LogParser, to take advantage of state-of-the-art techniques, we used Tensorflow (https://www.tensorflow.org/ accessed on 4 August 2020) (v1.7.0), an open-source software library for high-performance numerical computation. More specifically, we used a high-level API of Tensorflow, Keras (https://keras.io/ accessed on 4 August 2020) (v2.1.5), to implement LogParser in Python. All of our evaluations were conducted on a Linux server (with 64-bit Ubuntu 16.04.1, Linux kernel 4.13.0) with an Intel Xeon E5-2650v4 CPU and 512 GB RAM.

6.2. Performance of the Offline Parser

In this set of experiments, we trained our offline parser and all comparatives for three rounds on all 18 selected datasets and conducted *log parsing for event identification*

based on the best performing model from each log parsing method, respectively. Unless specified, the training set drawn from each dataset contained 200 randomly selected log records labeled with each type of event. If the number of records of a specific type of event was less than 200, all records were used.

6.2.1. General Performance

Table 2 shows the log parsing accuracy based on IPLoM, Drain, POP, NuLog, and our offline parser on all selected datasets. It is obvious that our method significantly outperforms all comparatives on all datasets. Furthermore, we notice that the conventional methods induce a quite low performance in certain scenarios, e.g., IPLoM on Jul95. We believe that the reason for this is two-fold: First, such methods inherently require domain knowledge for accuracy improvement while only basic data preprocessing was conducted in our evaluations. Second, the classification is not conducted based on the structural features of the log text. According to Table 2, on Nginx, when the classification criterion is 'request type' (i.e., Nginx-4 events), all methods manage to achieve accurate classification. However, when the criterion is changed to 'protocol version' (i.e., Nginx-2 events), unlike our offline parser, none of the comparatives demonstrates an acceptable performance. NuLog, which adopted machine learning techniques, achieved higher log parsing accuracy than other conventional methods. However, it still does not reach an acceptable performance (less than 0.70 precision) on some datasets, i.e., Nginx-2, Nginx-20, Thunderbird, Web log, and ThingWorx. We believe that this is due to the lack of task-specific data preprocessing. The self-supervised learning approach could not automatically adapt to the mining task requirement. Such results indicate that *our offline parser manages to support user-defined classification of log events that cannot be achieved by existing methods*.

Table 2. Offline parsing result among all evaluation datasets.

	IPLoM	Drain	POP	NuLog	Offline
HDFS	0.88	0.88	0.88	0.99	1.00
BGL	0.46	0.29	0.66	0.95	0.93
Spark	0.57	0.63	0.63	0.99	0.99
Apache	0.06	0.48	0.93	1.00	1.00
UofS	0.11	0.01	0.28	0.89	0.91
Jul95	0.00	0.91	0.28	1.00	1.00
Nginx-4 events	1.00	0.97	0.83	1.00	0.99
Nginx-2 events	0.08	0.50	0.53	0.65	0.99
Nginx-20 events	0.00	0.48	0.52	0.65	0.99
Openstack	0.73	0.74	0.72	0.99	1.00
Security Log	1.00	0.30	0.59	1.00	0.99
Thunderbird	0.07	0.04	0.26	0.67	0.93
Big Brother	0.18	0.54	0.55	0.73	0.95
Web log	0.00	0.05	0.08	0.12	0.96
ThingWorx	0.22	0.26	0.72	0.52	0.99
4SICS-151020	0.61	0.77	0.73	0.95	1.00
4SICS-151021	0.44	0.70	0.64	0.93	0.99
4SICS-151022	0.38	0.64	0.58	0.91	0.95
EMSD	0.27	0.60	0.46	0.74	0.95
IoT sentinel	0.54	0.65	0.54	0.88	0.95

Then, we further analyzed the results on ThingWorx, which implied that there are two major reasons for the low accuracy based on comparatives: First, the comparatives can

barely recognize certain types of events, e.g., the type five event 'anonymous error message' cannot be recognized by IPLoM (i.e., with 0.0 precision). Then, all comparatives induce low recalls except for those from POP and NuLog, which are still significantly lower than that from our method. Such results indicate that, unlike our offline parser, *the existing methods cannot effectively differentiate events in certain scenarios*.

6.2.2. Impact of Log Set Size

We also investigated the impact of log set size on the performance of both our offline parser and all comparatives. Specifically, for each dataset, we randomly selected subsets with different sizes from the entire log set (i.e., 200 records labeled with each type of event) for performance evaluation. For illustration, the results on HDFS and Spark are listed in Table 3. As we can see in Figure 5, when the log set size increases, our offline parser induces a negligible accuracy loss (or even enhances the accuracy, i.e., 4% on Spark), and Nulog performs the same and even achieves a better accuracy (i.e., 1.00 accuracy) on Spark(4703). We believe that overfitting caused our method to have a 5% lower accuracy than NuLog on Spark (4703). Conversely, all conventional comparatives suffer from obvious accuracy loss (e.g., POP, the best performing comparative, still induces 5% and 8% accuracy losses on HDFS and Spark, respectively). The reason, we believe, is that certain unexpected words appear relatively more frequently in large log sets, which hinders comparatives from identifying the correct log event type.

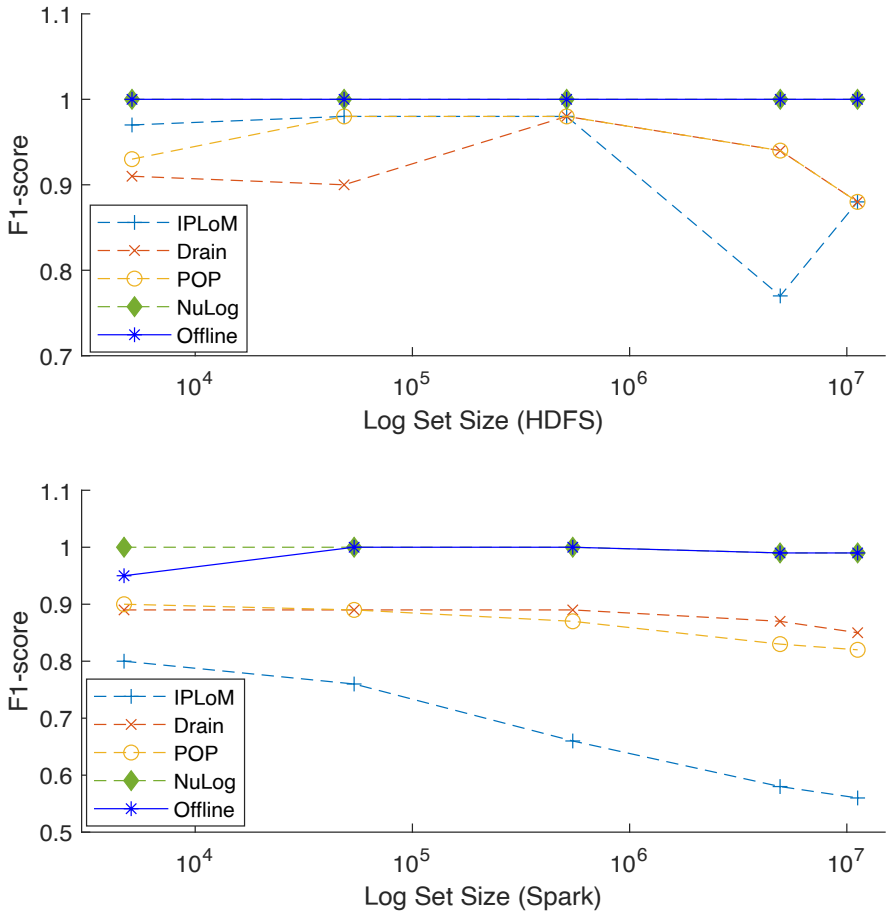

Figure 5. Offline parsing accuracy on datasets of different sizes.

Table 3. Offline parsing result over different log size subset from HDFS and Spark datasets.

	IPLoM	Drain	POP	NuLog	Offline
HDFS(5117)	0.97	0.91	0.93	1.00	1.00
HDFS(48472)	0.98	0.90	0.98	1.00	1.00
HDFS(513053)	0.98	0.98	0.98	1.00	1.00
HDFS(4938022)	0.77	0.94	0.94	1.00	1.00
HDFS(11197705)	0.88	0.88	0.88	1.00	1.00
Spark(4703)	0.80	0.89	0.90	1.00	0.95
Spark(53908)	0.76	0.89	0.89	1.00	1.00
Spark(547260)	0.66	0.89	0.87	1.00	1.00
Spark(4927590)	0.58	0.87	0.83	1.00	0.99
Spark(11215954)	0.56	0.85	0.82	0.99	0.99

6.2.3. Performance of Multi-Source Log Parsing

In this set of experiments, we compared the performance of LogParser to those of all comparatives on logs from multiple sources. To achieve this, we randomly selected a subset of log records with sizes from 100 k to 300 k from different log sets in Table 1 and constructed a *heterogeneous multi-source log set* (see Table 4) and a *homologous multi-source log set* (see Table 5), respectively. Specifically, *heterogeneous* represents logs generated by different components, and *homologous* represents logs generated by similar components. The experimental results are demonstrated in Figure 6 and Figure 7, respectively.

Table 4. Heterogeneous multi-source dataset.

Label	Source Dataset	Subset Size	Subset Events
heter_s_1	HDFS	191469	14
heter_s_2	Spark	336890	23
heter_s_3	Apache	175383	14
heter_s_4	BGL	355266	23
heter_s_5	Openstack	113885	9
heter_s_6	Security Log	67833	5
heter_s_7	Thunderbird	140007	7

Table 5. Homologous multi-source dataset.

Label	Source Dataset	Subset Size	Subset Events
homo_s_1	Nginx	41369	3
homo_s_2	Nginx	40002	2
homo_s_3	Web Log	149702	9
homo_s_4	Apache	120006	6
homo_s_5	Nginx	91379	8
homo_s_6	UofS	160008	8
homo_s_7	Jul95	70787	4

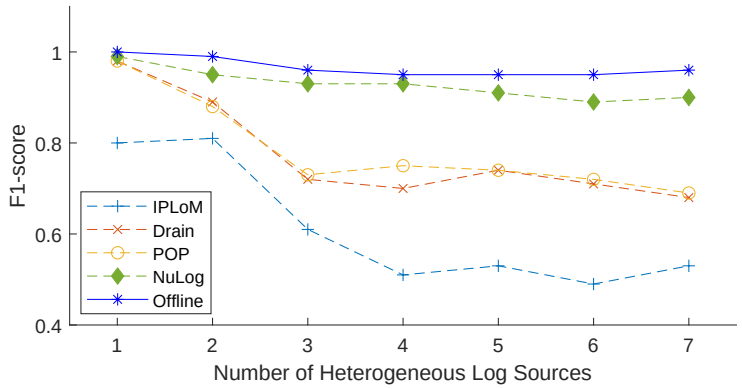

Figure 6. Offline parsing accuracy on heterogeneous multi-source datasets.

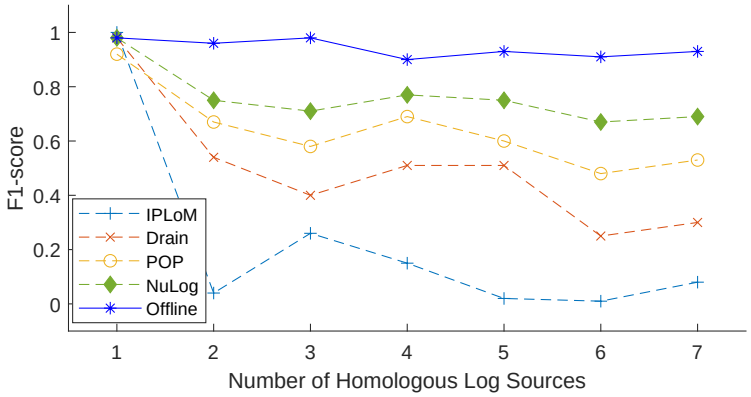

Figure 7. Offline parsing accuracy on homologous multi-source datasets.

According to Figure 6, we can see that, as the number of heterogeneous log sources increases, our offline parser has a negligible accuracy loss (i.e., no more than 5%), and NuLog has a slight accuracy loss (i.e., no more than 10%), while the accuracy of all comparatives drops significantly (i.e., from 29% to 32%). The reason, we believe, is that integrating heterogeneous log records from new sources into the original log set changes the original pattern of word distribution drastically. NuLog is able to learn the new pattern according to its self-supervise mechanism. However, it was still less effective than our method, which learned from labels. All conventional comparatives, however, rely on a single set of parameters to identify the event type of heterogeneous logs from multiple sources, which is difficult to achieve and downgrades the performance significantly.

According to Figure 7, the results on the homologous multi-source log set are similar to those on the heterogeneous set, and our offline parser (with an accuracy loss of no more than 8%) outperforms all comparatives (with accuracy losses from 31% to 99%) more obviously. The reason, we believe, is that all comparatives cannot differentiate log records with similar textual structures but different event identification criteria, despite the fact that we intentionally added the source index to enhance the dissimilarity between log records during the evaluation.

6.3. Performance of the Online Parser

As described in Section 5.4, our online parser periodically loads the latest offline parser and conducts online parsing of logs from incoming streams directly.

6.3.1. General Performance

According to our design, before the online parser is updated, each incoming log record may imply a new type of event, i.e., the training set of the loaded offline parser contains

no log record implying the same event type of such an incoming log record. We use r to represent the *ratio of log records implying new types of events to the number of all records from the stream within a certain period of time*. Theoretically, a higher r should induce a more obvious impact on the online parsing accuracy, and an ideal online log parser should be able to prevent the accuracy from downgrading when r increases. Therefore, in this set of experiments, we evaluated the performance of our online parser on log streams with different settings of r.

As streams, the log sets used by the evaluation of our offline parser (see Table 1) were fed into our online parser, our offline parser, Drain, and NuLog, respectively, for comparisons. Specifically, we fed complete log sets to our offline parser, Drain, and to NuLog as our baseline and comparative, respectively. Furthermore, log sets with $r \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ were fed to our online parser, respectively, to study the impact of r (for example, $r = 0.2$ represents that log records implying 20% of all event types are not selected by the training set of the offline parser). Figure 8 shows the online parsing results on the *HDFS* and *Spark* log sets (due to the page limitation, we only present such results for illustration), which demonstrates the F1-scores of all methods when different numbers of log records (up to 10^7) were fed.

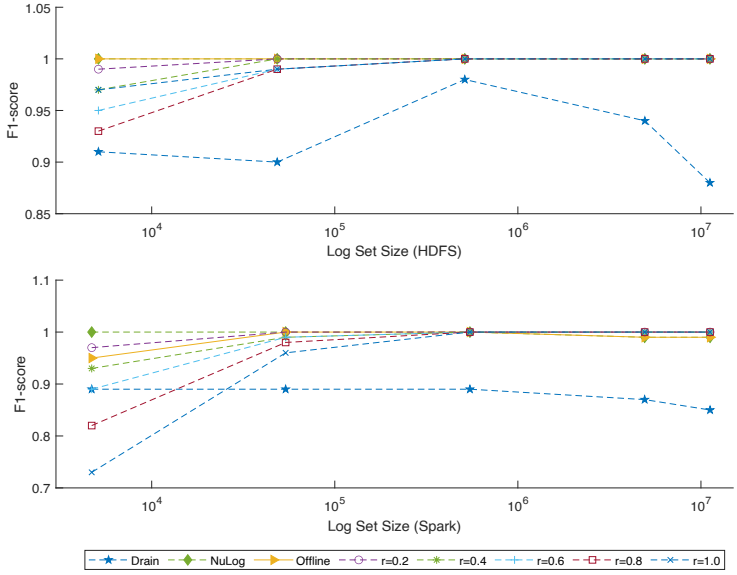

Figure 8. Online parsing accuracy on datasets of different sizes.

According to Figure 8, it is obvious that our online parser outperforms Drain in all scenarios (i.e., from 2% to 15% in terms of accuracy), except for when $r > 0.6$ and 4703 records were fed on the Spark log set. NuLog, on the other hand, performs evenly with our online parser since it already trained the whole dataset in advance. In cases of such small dataset sizes with limited trained log event types, learning cost leads to a reduction in the final accuracy.

For our online parser, as we expected, a higher r induces a more obvious accuracy loss. However, the parsing accuracy increases as the number of log records fed increases. When more than 5 million records are fed, a 1.0 F1-score is achieved, even when $r = 1.0$ (i.e., all incoming logs are with new event types). Meanwhile, for all settings of r, the accuracy difference between our online and offline parsers keeps reducing, which becomes negligible (i.e., up to 4%) when 50 k records were fed.

For certain cases, e.g., 4703 records, $r = 0.2$, more than 5 million records, and $r \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, our online parser achieves higher (up to 2%) accuracy compared to our offline parser. We believe that this comes from the difference between the offline evaluation procedure and the online evaluation procedure. The offline parser needs

to find a global template to identify all log events, while the online parser only needs to focus on the recent log records.

6.3.2. Performance of Multi-Source Log Parsing

In this set of experiments, we investigated the performance of our method on parsing multi-source log records. Specifically, we used the same log sets shown in Tables 4 and 5 to construct our heterogeneous and homologous multi-source log streams, respectively.

Online Parsing of Heterogeneous Logs: We first evaluated our online parser's capability to adapt to five different structural patterns of the heterogeneous log steam as follows:

1. *Full length, fixed order*: all log records from each source were fed sequentially;
2. *Short length, fixed order*: 400 log records from each source were fed sequentially in an iterative manner;
3. *Medium length, fixed order*: 5000 log records from each source were fed sequentially in an iterative manner;
4. *Medium length, random order*: 5000 log records from a random source were fed in an iterative manner;
5. *Fully random*: each log fed was from a random source.

Figure 9 demonstrates the results of our online parser on streams containing heterogeneous log records from different numbers of sources. It is obvious that our online parser performs well on streams with all structural patterns except for the 'short length, fixed order' one (i.e., the parsing accuracy is significantly downgraded on streams with six and seven sources when $r \geq 0.8$). Comparing to results under other settings, we believe that the input stream structure has a significant impact on the performance of our online parser, and it is quite challenging to achieve accurate parsing of streams with continuously shifting log patterns (i.e., sets of log records with a relatively limited size from different sources). In this case, the learning rate of the online parser might require explicit fine-tuning to guarantee the parsing accuracy. However, since our online parser performs well (both in terms of accuracy and stability) in the 'fully random' scenario, potential structural modifications can be conducted on log streams in practice to guarantee the effectiveness of our solution.

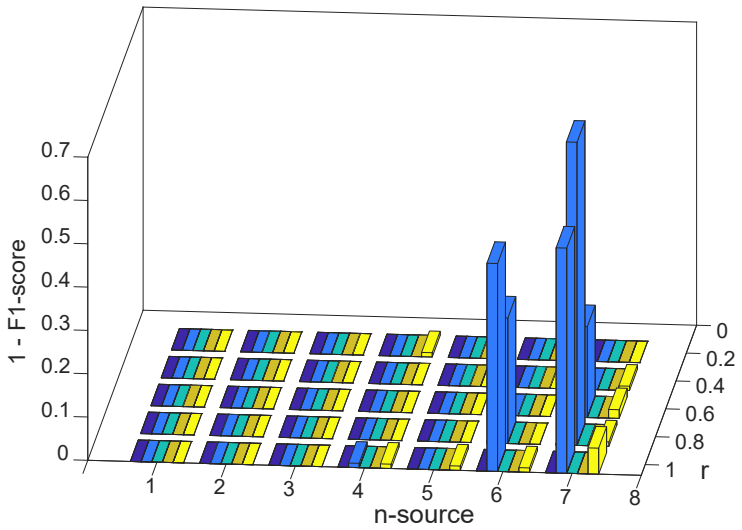

Figure 9. Online parsing accuracy on heterogeneous multi-source datasets legend from left to right: (1) full length from each source; (2) fixed length (400) from every single source with fixed order; (3) fixed length (5000) from every single source with fixed order; (4) fixed length (5000) from every single source with random order; (5) fully random.

Online Parsing of Homologous Logs: Similar to that of our offline parser, we also evaluated the performance of our online parser on homologous multi-source log streams. The results are shown in Figure 10, which demonstrates that our online parser obviously

outperforms Drain and NuLog (up to 75%/32% in terms of accuracy). Moreover, our online parser performs evenly with our offline parser.

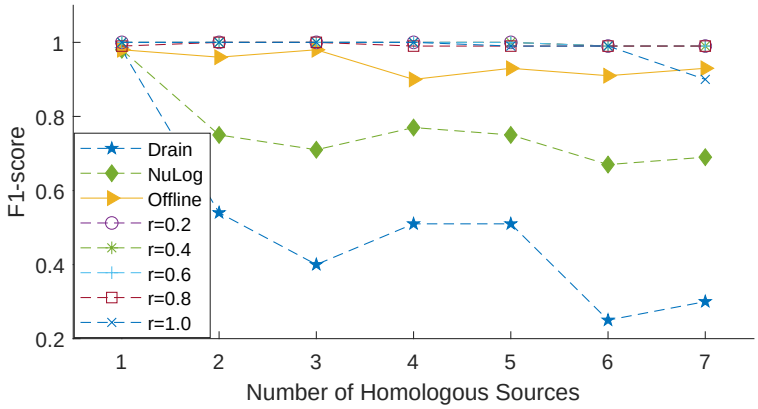

Figure 10. Online parsing accuracy on homologous multi-source datasets.

7. Discussion

In this section, we conduct an empirical study on the parameter settings of LogParser for a comprehensive discussion.

7.1. Parameter Tuning of Word2vec

During our experiments, we observed that the parameter tuning of Word2vec was extremely important to the performance of LogParser. In general, n^{w2v} determines the size of the vector e_t, i.e., the translation of a word x_t and words appearing no more than min_count times should be ignored. According to our experimental results, a high n^{w2v} leads to model overfitting and unstable performance, and a low n^{w2v} causes excessive information loss during the translation from x_t to e_t, which severely downgrades the final accuracy.

Realizing the properties above, for LogParser, we set $n^{w2v} = 30$, which is relatively low since log records normally have a smaller vocabulary compared to general texts without affecting the translation effectiveness. The setting of min_count, however, is non-trivial. It acts as a filter eliminating information that is non-essential for later training. For LogParser, we used three.

7.2. Dropout Rate

The dropout rate is commonly recommended to be set as 0.5, i.e., the dropout layer randomly ignores half of the inputs. However, according to our experimental results, a 0.3 dropout rate manages to slightly enhance the performance. We believe the reason is that a dropout layer is applied before the LSTM/BiLSTM layer, and dropping information too early could downgrade the final accuracy to a certain extent.

8. Conclusions

In this article, we present LogParser, a deep learning-based log parsing framework for both online and offline IICTS monitoring. Specifically, LogParser requires no specific textual property of IICTS logs and is designed to simultaneously parse multi-source logs with different task-specified criteria. The results of extensive experiments based on 18 real-world log sets demonstrate that LogParser obviously outperforms state-of-the-art offline and online log parsers in terms of parsing accuracy (i.e., 59.3%/45.1%/40.3%/14.5% higher on average). Furthermore, LogParser manages to achieve indisputable advantages over the existing methods in heterogeneous/homologous multi-source log parsing scenarios (i.e., with a 46%/92% higher parsing accuracy at most).

Author Contributions: Conceptualization, Y.Y.; data curation, Y.Y.; formal analysis, Y.Y. and B.W.; investigation, Y.Y.; methodology, Y.Y. and B.W.; supervision, C.Z.; validation, Y.Y. and C.Z.; visualization, Y.Y. and B.W.; writing—original draft preparation, Y.Y.; writing—review and editing, Y.Y. and C.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the National Key Research and Development Program of China under Grants 2020YFA0713900 and 2022YFA1004100; the National Natural Science Foundation of China under Grants 61772410, 61802298, 62172329, U1811461, U21A6005, and 11690011, and the China Postdoctoral Science Foundation under Grants 2020T130513 and 2019M663726.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lee, J.; Bagheri, B.; Kao, H.A. Recent advances and trends of cyber-physical systems and big data analytics in industrial informatics. In Proceedings of the 12th International Conference on Industrial Informatics (INDIN), Porto Alegre, Brazil, 27–30 July 2014; pp. 1–6.
2. Wang, Y.; Hong, K.; Zou, J.; Peng, T.; Yang, H. A CNN-based visual sorting system with cloud-edge computing for flexible manufacturing systems. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4726–4735. [CrossRef]
3. Wan, J.; Tang, S.; Li, D.; Wang, S.; Liu, C.; Abbas, H.; Vasilakos, A.V. A manufacturing big data solution for active preventive maintenance. *IEEE Trans. Ind. Inform.* **2017**, *13*, 2039–2047. [CrossRef]
4. He, P.; Zhu, J.; He, S.; Li, J.; Lyu, M.R. An evaluation study on log parsing and its use in log mining. In Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Toulouse, France, 28 June–1 July 2016; pp. 654–661.
5. Xu, W.; Huang, L.; Fox, A.; Patterson, D.; Jordan, M.I.T. Detecting large-scale system problems by mining console logs. In Proceedings of the 22nd ACM Symposium on Operating Systems Principles, Big Sky, MT, USA, 11–14 October 2009; pp. 117–132.
6. Samples of Security Related Data. Available online: https://www.secrepo.com/ (accessed on 4 August 2020).
7. Thingworx Platform. Available online: https://developer.thingworx.com/en/platform (accessed on 4 August 2020).
8. Vaarandi, R. A data clustering algorithm for mining patterns from event logs. In Proceedings of the IEEE IPOM Workshop, Kansas City, MO, USA, 3 October 2003; pp. 119–126.
9. Tang, L.; Li, T.; Perng, C.S. LogSig: Generating system events from raw textual logs. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management, Glasgow, UK, 24–28 October 2011; pp. 785–794.
10. Fu, Q.; Lou, J.G.; Wang, Y.; Li, J. Execution anomaly detection in distributed systems through unstructured log analysis. In Proceedings of the IEEE ICDM, Miami Beach, FL, USA, 6–9 December 2009; pp. 149–158.
11. Makanju, A.A.; Zincir-Heywood, A.N.; Milios, E.E. Clustering event logs using iterative partitioning. In Proceedings of the The 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 1255–1264.
12. He, P.; Zhu, J.; Zheng, Z.; Lyu, M.R. Drain: An online log parsing approach with fixed depth tree. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA, 25–30 June 2017; pp. 33–40.
13. He, P.; Zhu, J.; He, S.; Li, J.; Lyu, M.R. Towards automated log parsing for large-scale log data analysis. *IEEE Trans. Dependable Secure Comput.* **2018**, *15*, 931–944. [CrossRef]
14. Aussel, N.; Petetin, Y.; Chabridon, S. Improving performances of log mining for anomaly prediction through nlp-based log parsing. In Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), Milwaukee, WI, USA, 25–28 September 2018; pp. 237–243.
15. Nguyen, T.; Kobayashi, S.; Fukuda, K. Logdtl: Network log template generation with deep transfer learning. In Proceedings of the 17th IFIP/IEEE International Symposium on Integrated Network Management, Bordeaux, France, 17–21 May 2021; pp. 848–853.
16. Nedelkoski, S.; Bogatinovski, J.; Acker, A.; Cardoso, J.; Kao, O. Self-supervised log parsing. In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Bilbao, Spain, 13–17 September 2021; pp. 122–138.
17. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. In Proceedings of the Conference on Learning Representations, ICLR 2013, Scottsdale, AZ, USA, 2–4 May 2013.
18. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
19. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
20. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **2005**, *18*, 602–610. [CrossRef] [PubMed]

21. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015.
22. Li, L.; Chang, Q.; Ni, J. Data driven bottleneck detection of manufacturing systems. *Int. J. Prod. Res.* **2009**, *47*, 5019–5036. [CrossRef]
23. Ji, C.; Liu, S.; Yang, C.; Wu, L.; Pan, L. Ibdp: An industrial big data ingestion and analysis platform and case studies. In Proceedings of the International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI), Beijing, China, 22–23 October 2015; pp. 223–228.
24. Zhang, F.; Kodituwakku, H.A.D.E.; Hines, J.W.; Coble, J. Multilayer data-driven cyber-attack detection system for industrial control systems based on network, system, and process data. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4362–4369. [CrossRef]
25. Li, X.; Wan, J.; Dai, H.N.; Imran, M.; Xia, M.; Celesti, A. A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4225–4234. [CrossRef]
26. Oliner, A.; Stearley, J. What supercomputers say: A study of five system logs. In Proceedings of the International Conference on Dependable Systems and Networks (DSN), Edinburgh, UK, 25–28 June 2007; pp. 575–584.
27. Arlitt, M.F.; Williamson, C.L. Web server workload characterization: The search for invariants. *ACM SIGMETRICS Perform. Eval. Rev.* **1996**, *24*, 126–137. [CrossRef]
28. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
29. Sconzo, M.; Dorsey, D. Connection Log. Security Data Analysis Labs. 2014. Available online: https://github.com/sooshie/Security-Data-Analysis (accessed on 4 August 2020).
30. Committee, I.V.C. Big Brother Data. 2013. Available online: http://vacommunity.org/VAST+Challenge+2013 (accessed on 4 August 2020).
31. Industrial cyber security conference 4SICS, T. Capture Files from 4SICS Geek Lounge. 2015. Available online: https://www.netresec.com/index.ashx?page=PCAP4SICS (accessed on 4 August 2020).
32. Miettinen, M.; Marchal, S.; Hafeez, I.; Asokan, N.; Sadeghi, A.R.; Tarkoma, S. Iot sentinel: Automated device-type identification for security enforcement in iot. In Proceedings of the 37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, 5–8 June 2017; pp. 2177–2184.