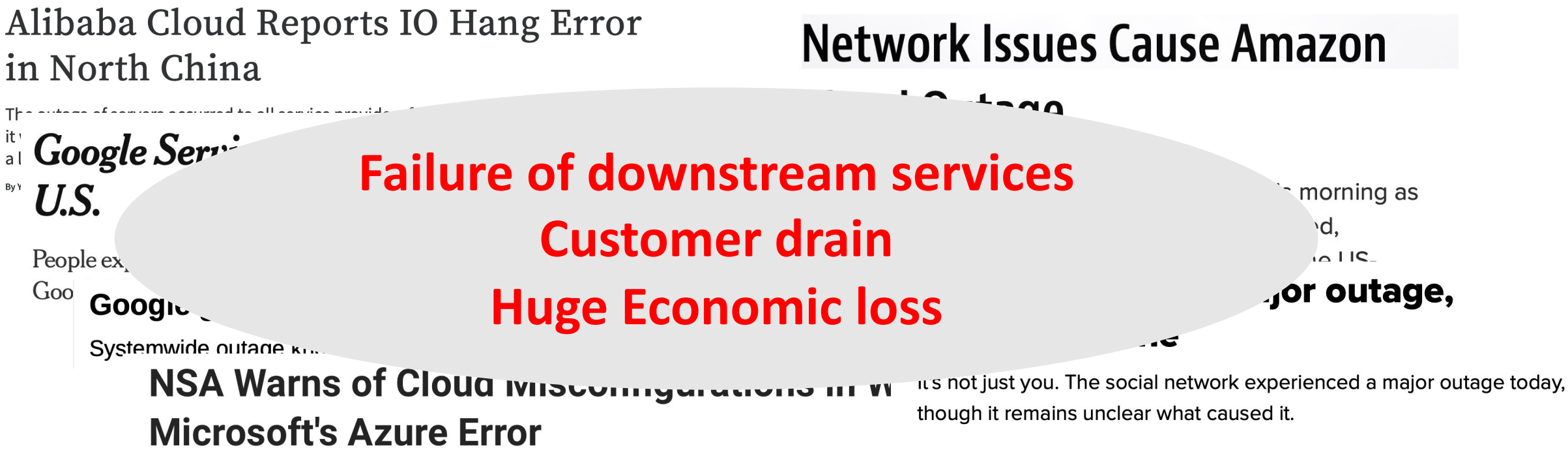
SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults

Xiaoyun Li, Pengfei Chen*, Linxiao Jing, Zilong He and Guangba Yu

School of Data and Computer Science, Sun Yat-sen University
Guangzhou, China

2020.10.12

Severe Anomalies in Cloud

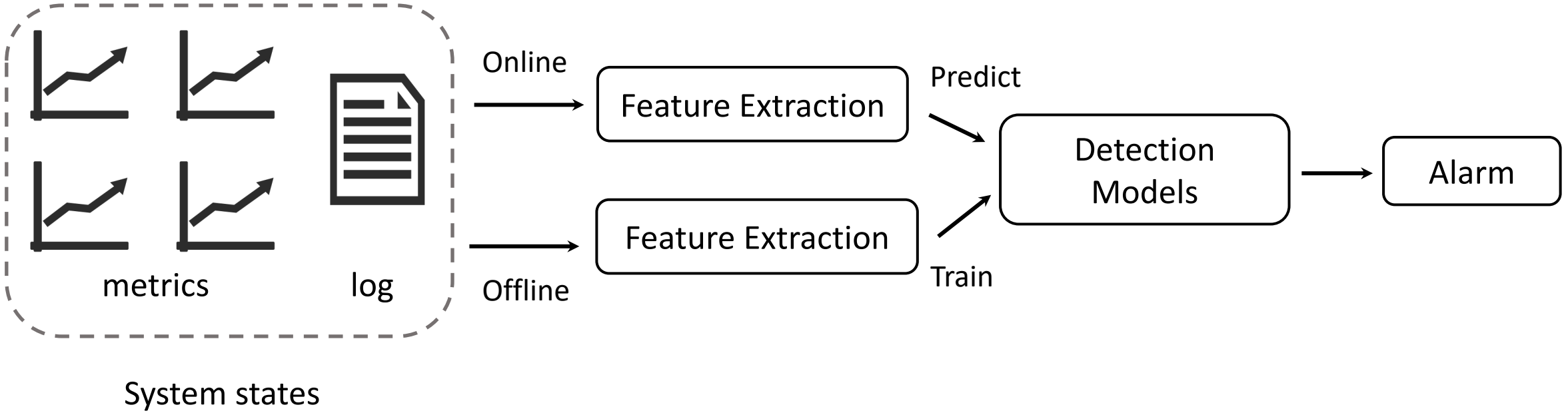


Failure of downstream services

Customer drain

Huge Economic loss

How to detect anomalies: common solutions



Our goal: to reduce MTTR (Mean Time To Recovery)

Log-based anomaly detection methods

Detecting Large-Scale System Problems by Mining Console Logs

Wei Xu* Ling Huang† Armando Fox* David Patterson* Michael I. Jordan*

Anomaly Detection Using Program Control Flow Graph Mining from Execution Logs

Animesh Nandi‡, Atri Mandal‡, Shubham Atreja◇, Gargi B. Dasgupta‡, Subhrajit Bhattacl

DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning

Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar

Mining Program Workflow from Interleaved Tra

Jian-Guang Lou[1] Qiang Fu[1] Shengqi Yang[2] Jiang Li[1] B

[1]Microsoft Research Asia
Beijing, P. R. China
{jlou, qifu, jiangli}@microsoft.com

[2]Dept. of Computer Scie
Beijing Univ. of Posts and Telecom
wubin@bupt.edu.cn

Identifying Impactful Service System Problems via Log Analysis

Shilin He*† Qingwei Lin Jian-Guang Lou
ese University of Hong Kong Microsoft Research Microsoft Research
Hong Kong, China Beijing, China Beijing, China

Robust Log-Based Anomaly Detection on Unstable Log Data

Xu Zhang* Yong Xu Hongyu Zhang Yingnong Dang
oft Research Qingwei Lin University of Newcastle Microsoft Azure
g, China Bo Qiao Callaghan, Australia Redmond, USA
University Microsoft Research
ng, China Beijing, China

LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs

Weibin Meng[1,5], **Ying Liu**[1,5], **Yichen Zhu**[2], **Shenglin Zhang**[3*], **Dan Pei**[1,5], **Yuqing Liu**[3], **Yihao Chen**[1,5], **Ruizhi Zhang**[4], **Shimin Tao**[4], **Pei Sun**[4] and **Rong Zhou**[4]

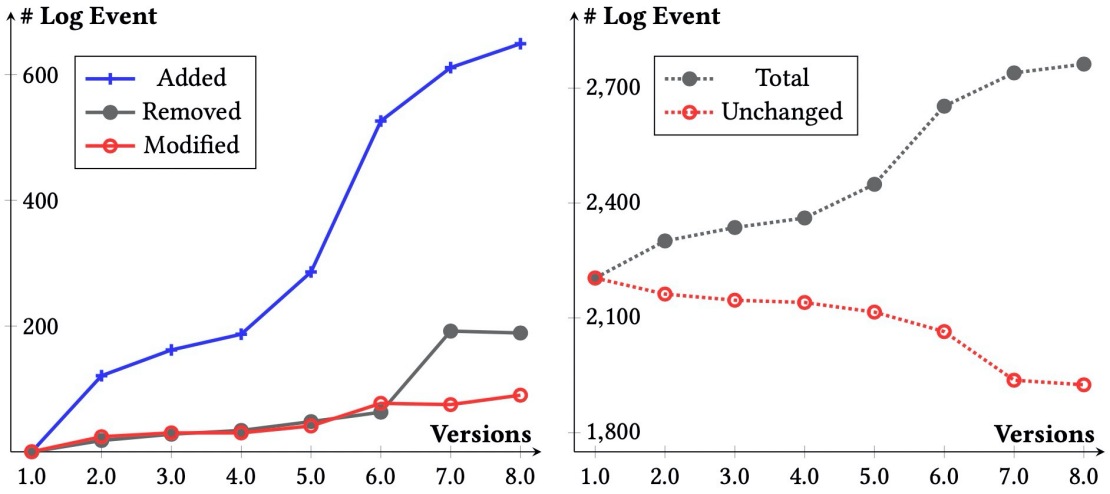
Microsoft Azure
Redmond, USA

Jian-Guang Lou Murali Chintalapati Furao Shen
Microsoft Research Microsoft Azure Nanjing University
Beijing, China Redmond, USA Nanjing, China

Dongmei Zhang
Microsoft Research
Beijing, China

Real-world environment is more complex

Log templates are changing



(a) Changed log events (b) All log events

Figure 2: The Evolution of Log Events across Versions

[1]

TABLE I: An overview of the studied applications (all metrics calculated using the latest HEAD of the repository)

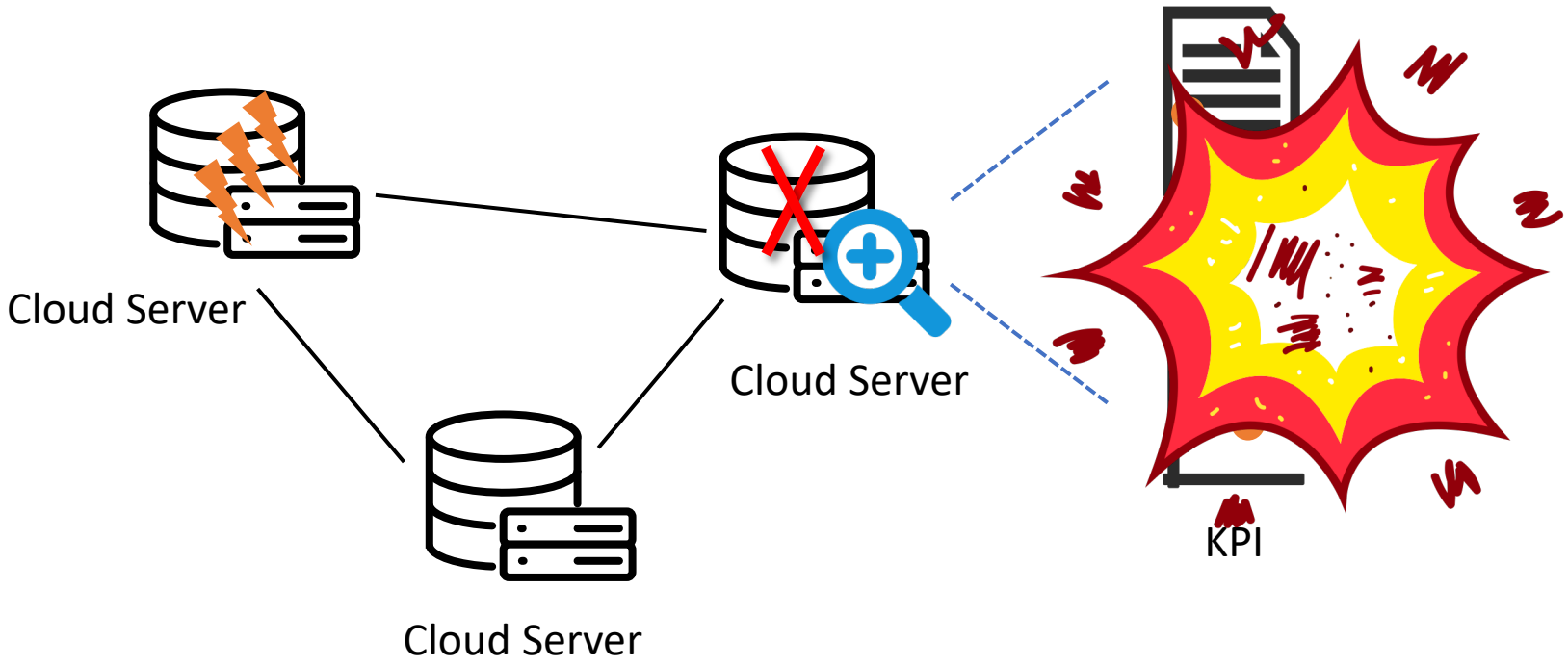
	ActiveMQ	Camel	CloudStack	Liferay
# of logging statements	5.1K	6.1K	9.6K	1.8K
# of commits	11K	21K	29K	143K
# of years in repository	8	8	4	4
# of contributors	41	151	204	351
# of added lines of code	261K	505K	1.09M	3.9M
# of deleted lines of code	114K	174K	750K	2.8M
# of added logging statements	4.5K	5.1K	24K	10.4K
# of deleted logging statements	2.3K	2.4K	17K	8.1K
% of logging-related changes	1.8%	1.1%	2.3%	0.3%

[2]

[1] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *ESEC/FSE'19: Proc. of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019, pp. 807–817.

[2] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empirical Software Engineering*, vol. 23, no. 1, pp. 290–333, 2018.

Performance issues are silent



Performance issues are the common manifestation of “partial failures”

Performance issues are not so easy to detect

So…How to build a model to be robust to changing events and be effective to detect performance issues?

We first observe log data, and analyze the requirements in real-world:

- Effective to diverse faults
  - Sequence order change (severe issues)
  - Log time interval change (performance issues)
- Robust to changing events

Observation from real-world log data

09:59:38 INFO Receiving block
09:59:38 INFO AllocateBlock
09:59:39 INFO Receiving block
09:59:39 INFO Receiving block
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO AddStoredBlock
10:00:01 INFO AddStoredBlock
10:00:02 INFO AddStoredBlock

Normal block

09:37:53 INFO AllocateBlock
09:37:54 INFO Receiving block
09:37:54 INFO Receiving block
09:37:54 INFO Receiving block
09:38:49 INFO PacketResponder terminating
09:38:49 INFO Received block size
09:38:49 INFO PacketResponder terminating
09:38:49 INFO Received block size
09:38:49 INFO PacketResponder terminating
09:38:49 INFO Received block size
09:38:49 INFO AddStoredBlock
09:38:49 INFO AddStoredBlock
09:38:51 INFO AddStoredBlock
09:38:49 WARN Redundant addStoredBlock request

Sequence order change

Observation from real-world log data

09:59:38 INFO Receiving block
09:59:38 INFO AllocateBlock
09:59:39 INFO Receiving block
09:59:39 INFO Receiving block
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO AddStoredBlock
10:00:01 INFO AddStoredBlock
10:00:02 INFO AddStoredBlock

Normal block

09:59:38 INFO Receiving block
09:59:38 INFO AllocateBlock
09:59:42 INFO Receiving block
09:59:42 INFO Receiving block
10:00:04 INFO PacketResponder terminating
10:00:04 INFO Received block size
10:00:04 INFO PacketResponder terminating
10:00:04 INFO Received block size
10:00:04 INFO PacketResponder terminating
10:00:04 INFO Received block size
10:00:04 INFO AddStoredBlock
10:00:04 INFO AddStoredBlock
10:00:05 INFO AddStoredBlock

Log time interval change

Observation from real-world log data

```
09:59:38 INFO Receiving block
09:59:38 INFO AllocateBlock
09:59:39 INFO Receiving block
09:59:39 INFO Receiving block
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO AddStoredBlock
10:00:01 INFO AddStoredBlock
10:00:02 INFO AddStoredBlock
```

```
09:59:38 INFO Receiving block from ip
09:59:38 INFO AllocateBlock
09:59:39 INFO Receiving block from ip
09:59:39 INFO Receiving block from ip
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO PacketResponder terminating
10:00:01 INFO Received block size
10:00:01 INFO AddStoredBlock
10:00:01 INFO AddStoredBlock
10:00:02 INFO AddStoredBlock
```

Changing events

Thus, we propose *SwissLog*

Facing these *challenges* in real-world

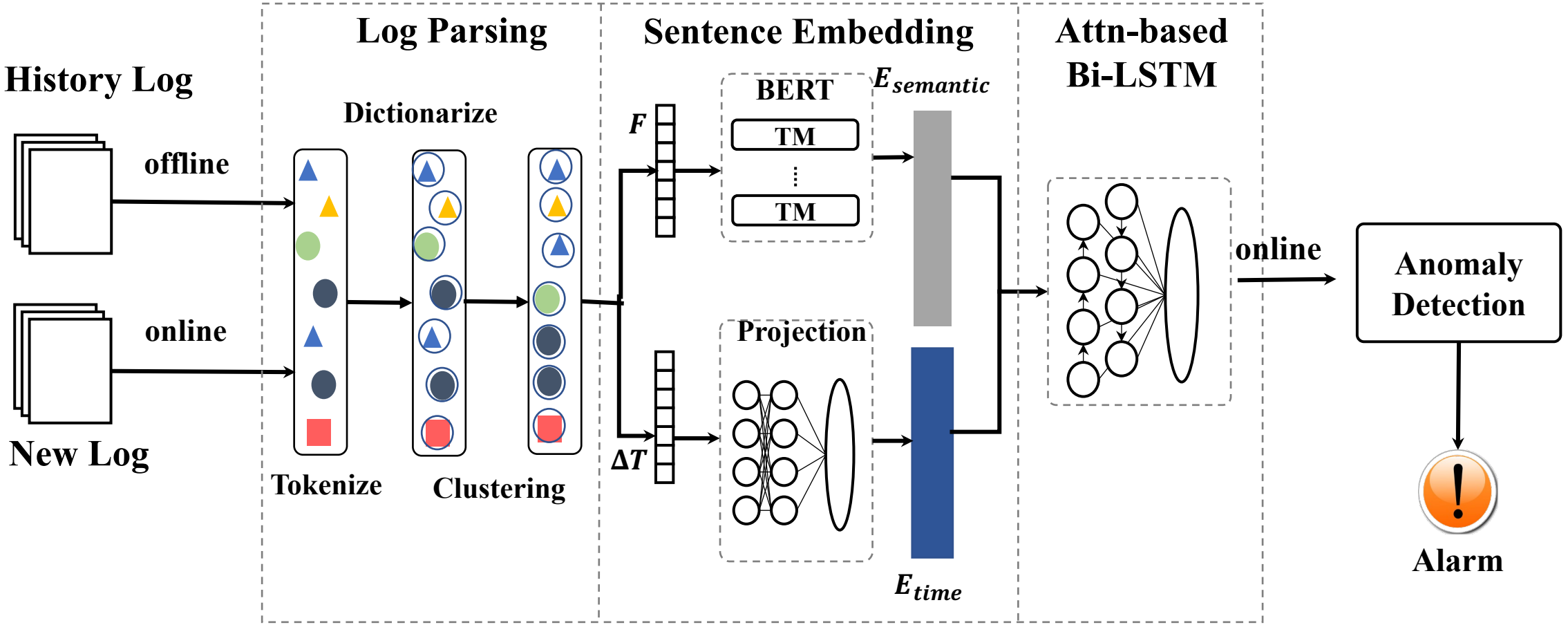
- Diverse Faults
  - Sequence order change
  - Log time interval change
- Changing Events

SwissLog

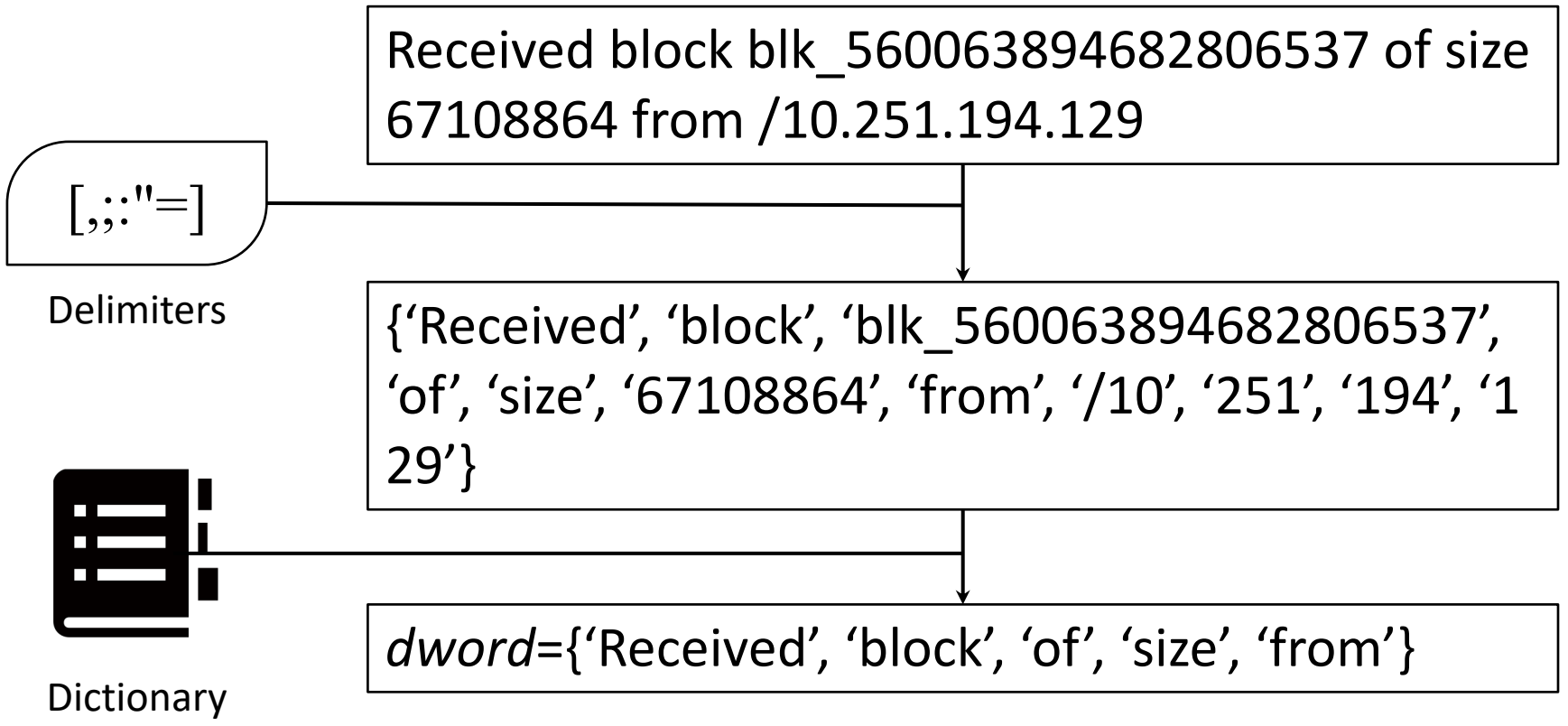
- Robust
- Unified
- Versatile

Swiss Army Knife

Overview of *SwissLog*



Log Parsing: Tokenize and Preprocess

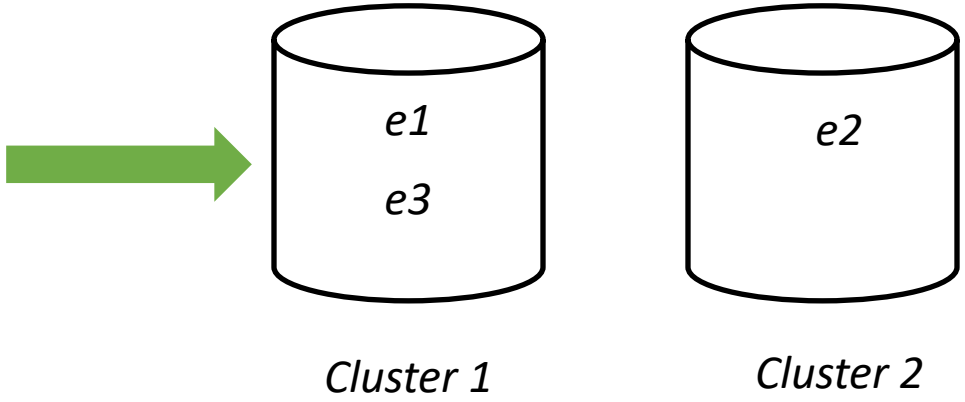


Log Parsing: Clustering by Wordset

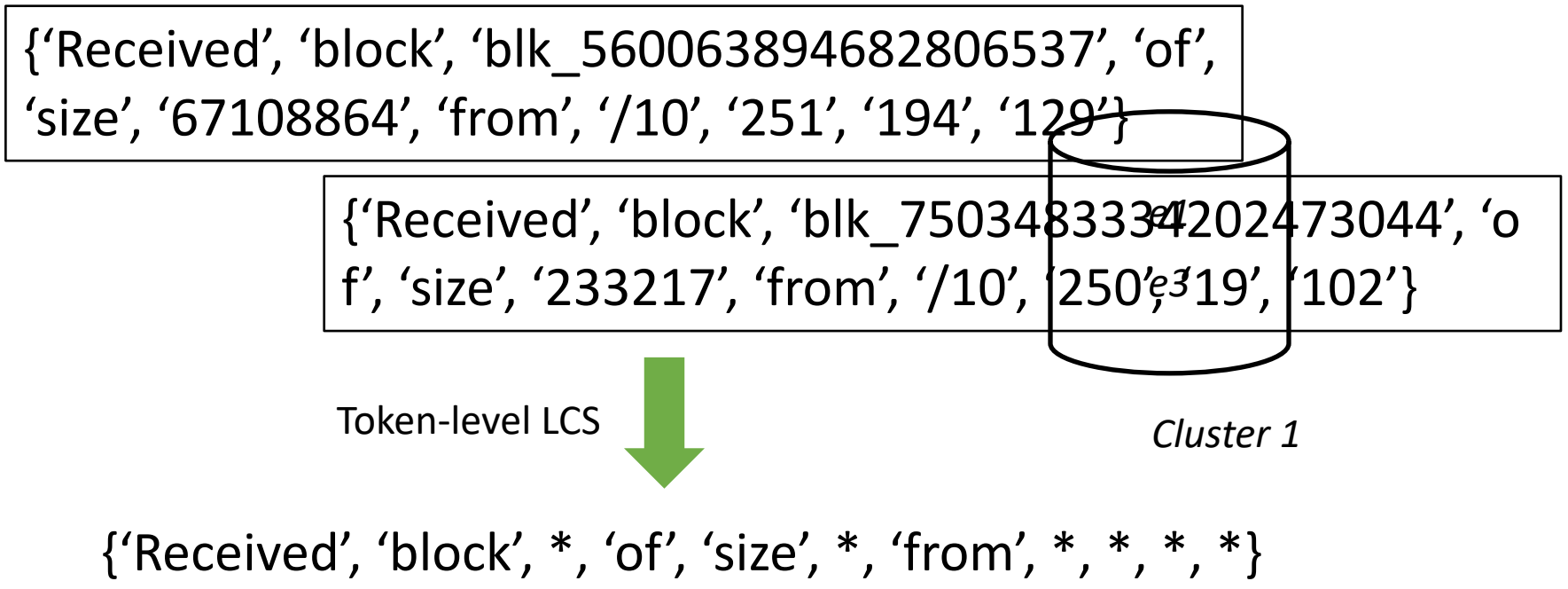
dword1={‘Received’, ‘block’, ‘of’, ‘size’, ‘from’}

dword2={‘Block’, ‘name’, ‘system’, ‘allocate’, ‘block’}

dword3={‘Received’, ‘block’, ‘of’, ‘size’, ‘from’}



Log Parsing: Mask Variables with LCS

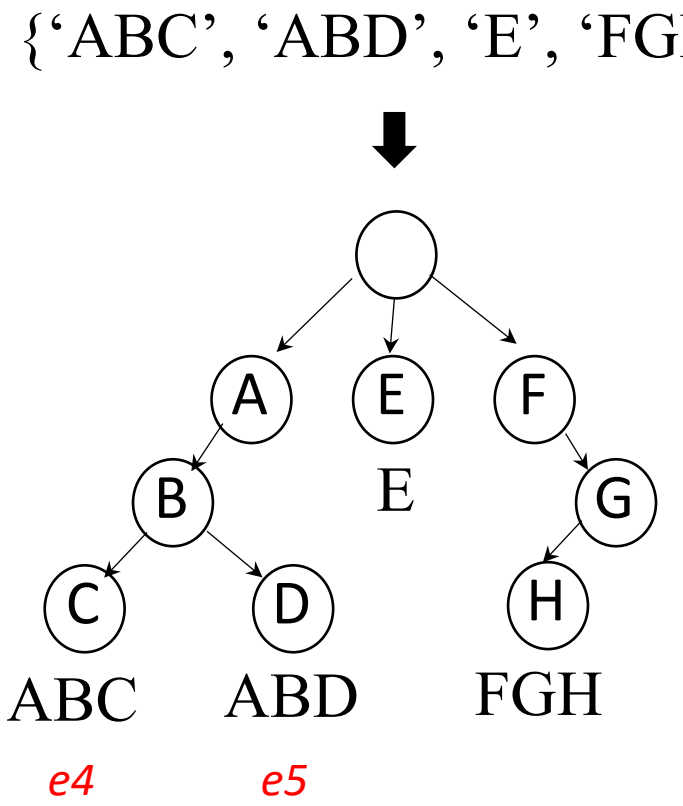


Log Parsing: Clustering using Prefix Tree

Template:
Disconnecting: Too many authentication failures for * [preauth]

e4: Disconnecting: Too many authentication failures for admin [preauth]
e5: Disconnecting: Too many authentication failures for root [preauth]

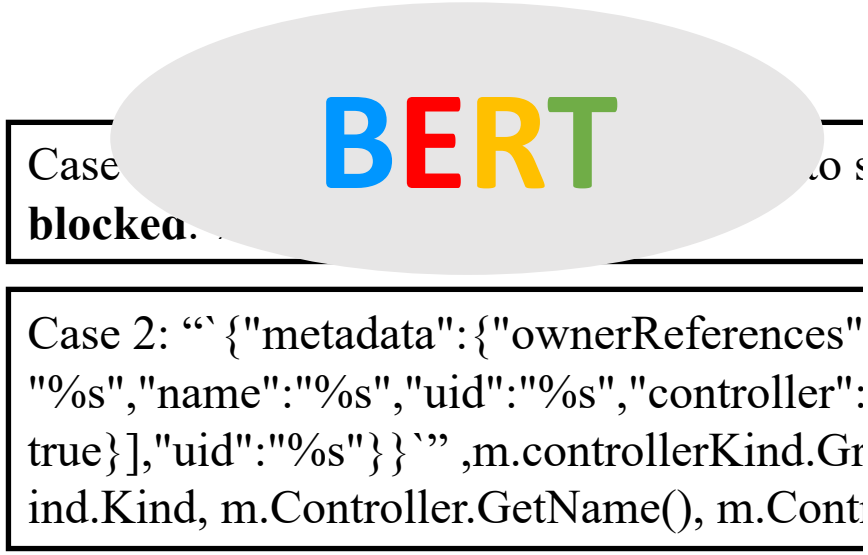
Templates: “Disconnecting: Too many authentication failures for * [preauth]”

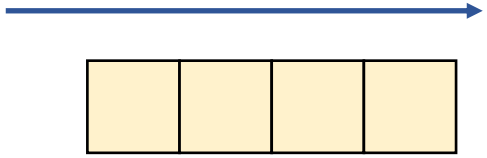


Sentence Embedding with Semantic Info

Log formats are under active evolution

Existing approaches using limited





Ordered sentence

Sentence Embedding with Temporal Info

Why to introduce log time interval? **Latency** is the common manifestation

$\Delta t_1 = 0$

$\Delta t_2 = 3$

$\Delta t_3 = 0$

09:59:38 INFO Receiving block

09:59:38 INFO AllocateBlock

09:59:39 INFO Receiving block

09:59:39 INFO Receiving block

⋮

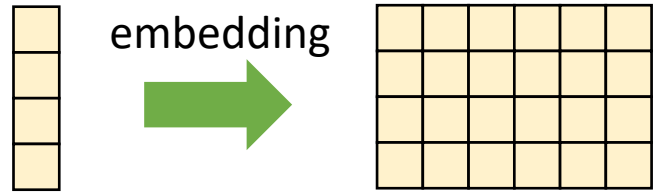
10:00:02 INFO AddStoredBlock

$$\Delta T = \{padding, \Delta t_1, \Delta t_2, \dots, \Delta t_i, \dots\}$$
$$= \{-1, 0, 3, 0, \dots\}$$

1. the time interval vibrates in a task-related time range

$$\theta = \frac{1}{\Delta t}$$

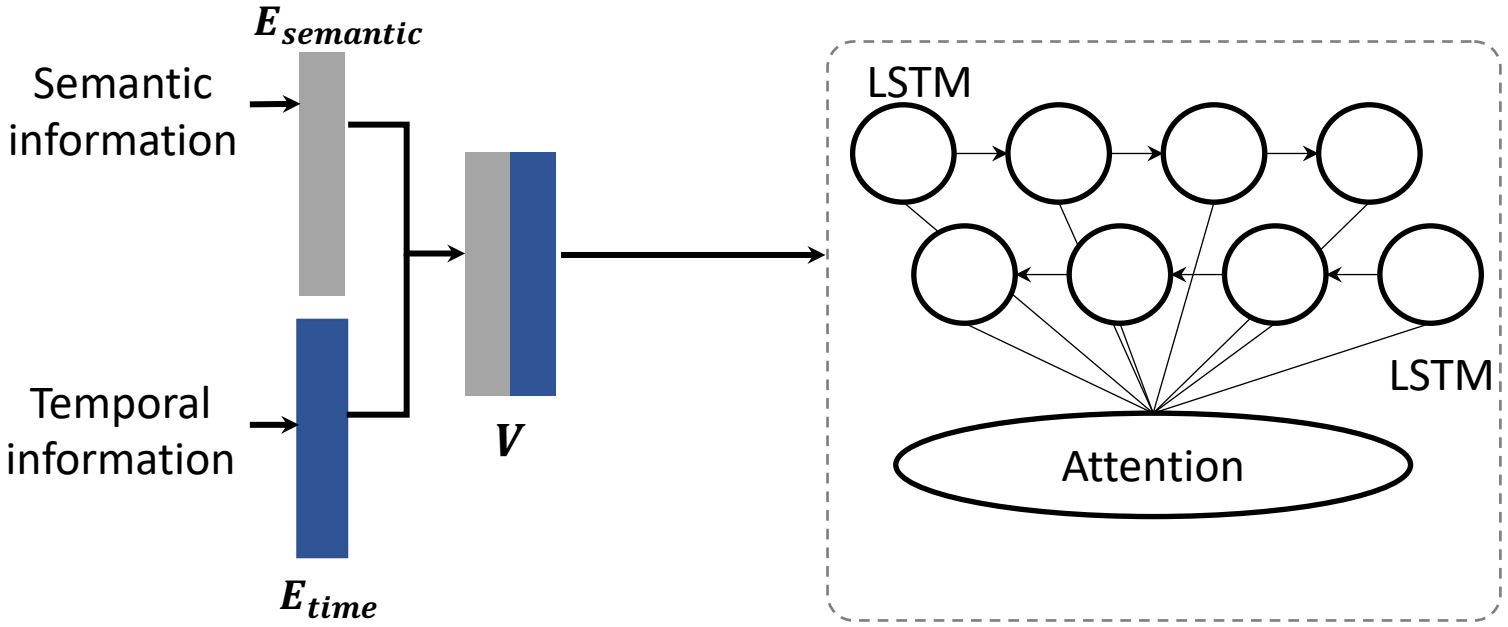
2. 1-dimension temporal data exhibits limited information



1-dimension high-dimension

Attention-based Bi-LSTM: Training

Assign different weights to different log statements



09:59:38 INFO Receiving block

09:59:38 INFO AllocateBlock

09:59:39 INFO Receiving block

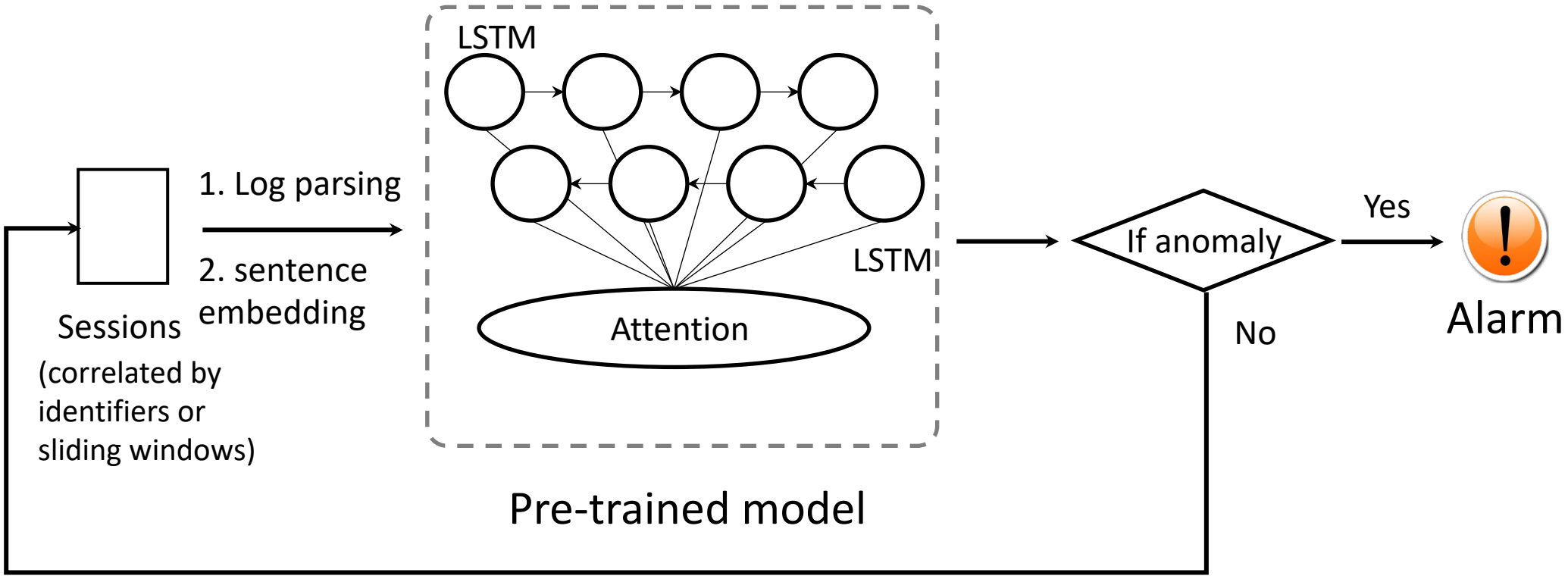
09:59:39 INFO Receiving block

⋮

10:00:02 INFO AddStoredBlock

$V = concat\{E_{semantic}, E_{time}\}$ Deep learning model

Anomaly Detection



Evaluation

Research Questions:

- RQ1: How effective and robust is the proposed log parser?
- RQ2: How effective is the BERT encoder on anomaly detection? Do other log parsers perform as well as the proposed log parser using BERT encoder?
- RQ3: How robust is ***SwissLog*** on those log data with changing events?
- RQ4: Can ***SwissLog*** detect performance issues? How sensitive is ***SwissLog*** to log time deviations?

Evaluation

Datasets:

- Real-World datasets

Log Type	#Messages	#Templates	Labeled
HDFS	11,175,629	30	Yes
Blue Gene /L	4,747,963	377	Yes
Android	30,348,042	76,923	No

- Synthetic datasets

Injected Type	Notes
Changing events	Only unimportant words are inserted or removed. Ratio ranges from 5% to 30% to the original HDFS log data.
Performance issues	apply the time interval latency injection to mimic CPU hog, memory hog, disk write burn and network delay with ratio 5% to those log whose original time interval is less than 2.

Evaluation

Metrics:

- Parsing Accuracy: $PA = \frac{count(correct\ event\ ID\ group)}{count(all\ event\ ID\ group)}$. The ratio of correctly parsed log messages over the total number of log messages.
- Precision: $P = \frac{TP}{TP + FP}$. The percentage of correctly detected anomalies amongst all detected anomalies.
- Recall: $R = \frac{TP}{TP + FN}$. The percentage of correctly detected anomalies amongst all real anomalies.
- F1-Score: $F1 = \frac{2 * P * R}{P + R}$. The harmonic mean of Precision and Recall.

RQ1: The Effectiveness and Robustness of Log Parser

TABLE II
COMPARISONS OF SWISSLOG AND SOTA OF PARSING ACCURACY ON DIFFERENT LOG DATASETS

Dataset		HDFS	Hadoop	Spark	Zookeeper	BGL	HPC	Thunderbird	OpenStack	Mac
Parsing Accuracy	**SwissLog**	**1.000**	**0.992**	**0.997**	**0.985**	**0.970**	**0.910**	**0.992**	**1.000**	0.840
	SOTA	1.000	0.957	0.994	0.967	0.963	0.903	0.955	0.871	**0.872**
Dataset		Windows	Linux	Andriod	HealthApp	Apache	Proxifier	OpenSSH	Average	
Parsing Accuracy	**SwissLog**	**1.000**	**0.869**	**0.954**	**0.901**	**1.000**	**0.990**	**1.000**	**0.962**	
	SOTA	0.997	0.701	0.919	0.822	1.000	0.967	0.925	0.865	

PM response took <*> ms (<*>, powerd)
PM response took <*> ms (<*>, QQ)
PM response took <*> ms (<*>, WeChat)

→ <*>

Challenges in template definition

Customize dictionary when preprocessing according to different template discriminant

RQ1: The Effectiveness and Robustness of Log Parser

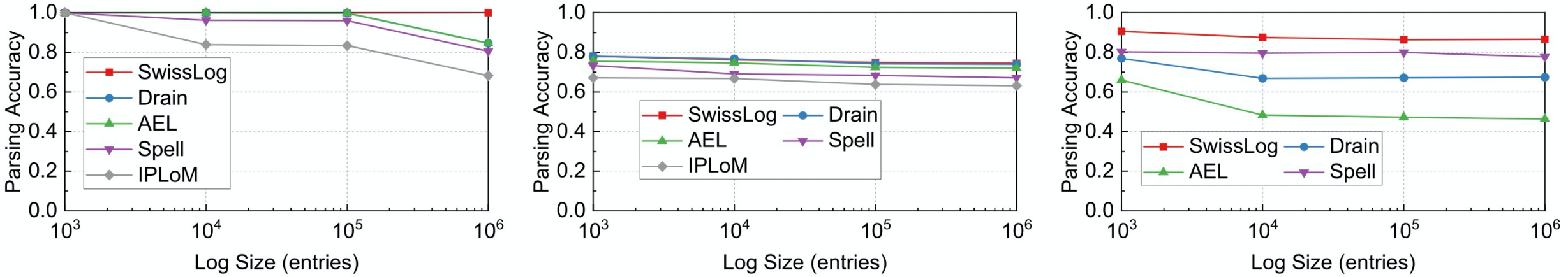


Fig. 8. Comparisons among different log parsers of parsing accuracy on different volumes of logs

Confirm the effectiveness in large datasets

RQ2: The Effectiveness of Semantic-based Model

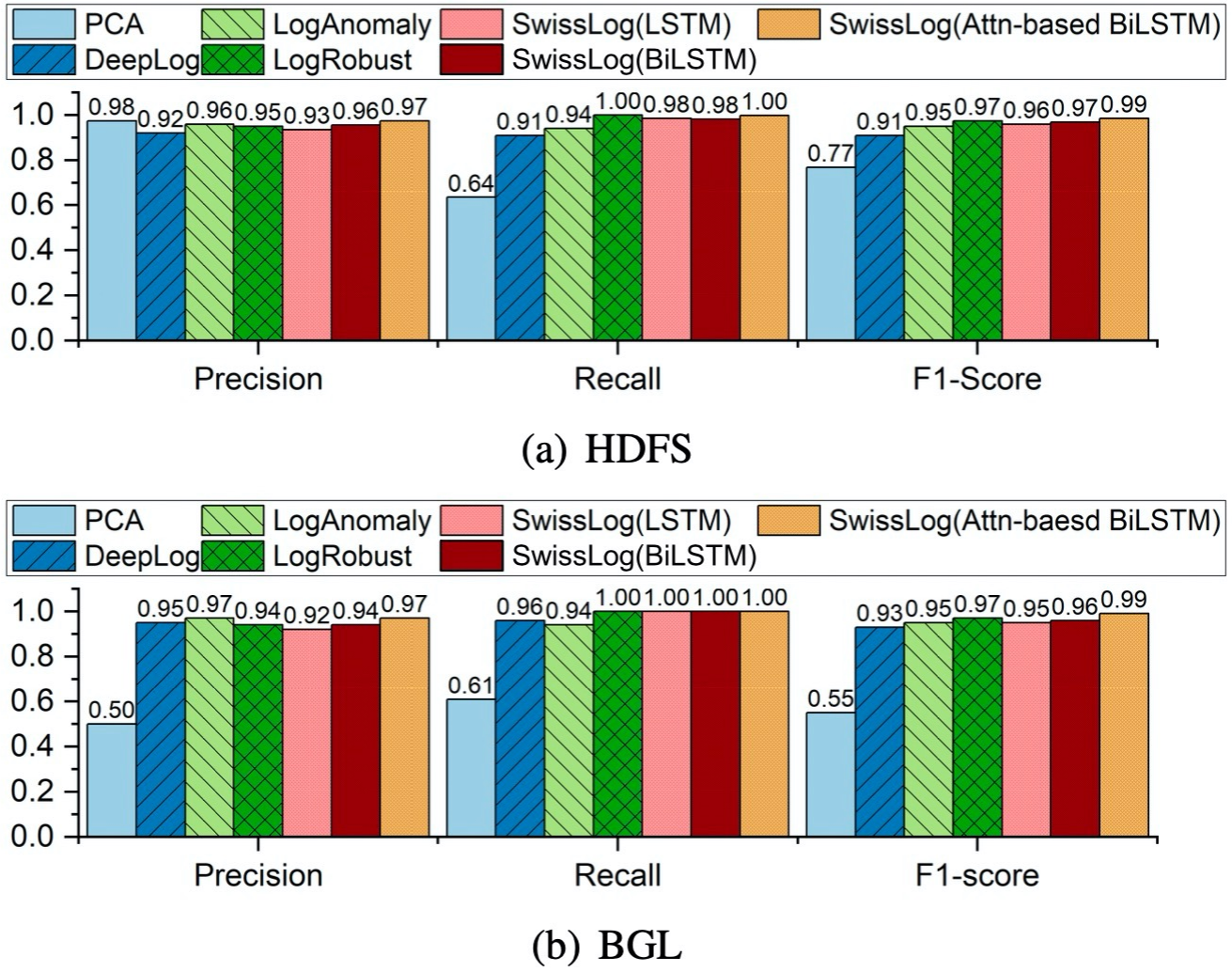


Fig. 10. Comparisons of different approaches on different datasets

TABLE III
RESULTS OF DIFFERENT LOG PARSERS USING BERT ENCODERS

LogParser	Precision	Recall	F1-Score
Drain	0.95	0.96	0.96
AEL	0.96	0.97	0.97
SwissLog	**0.97**	**1.00**	**0.99**

The fixed variable of log parsers and anomaly detection method shows the effectiveness of semantic-based model

RQ3: The Robustness on Changing Log Data

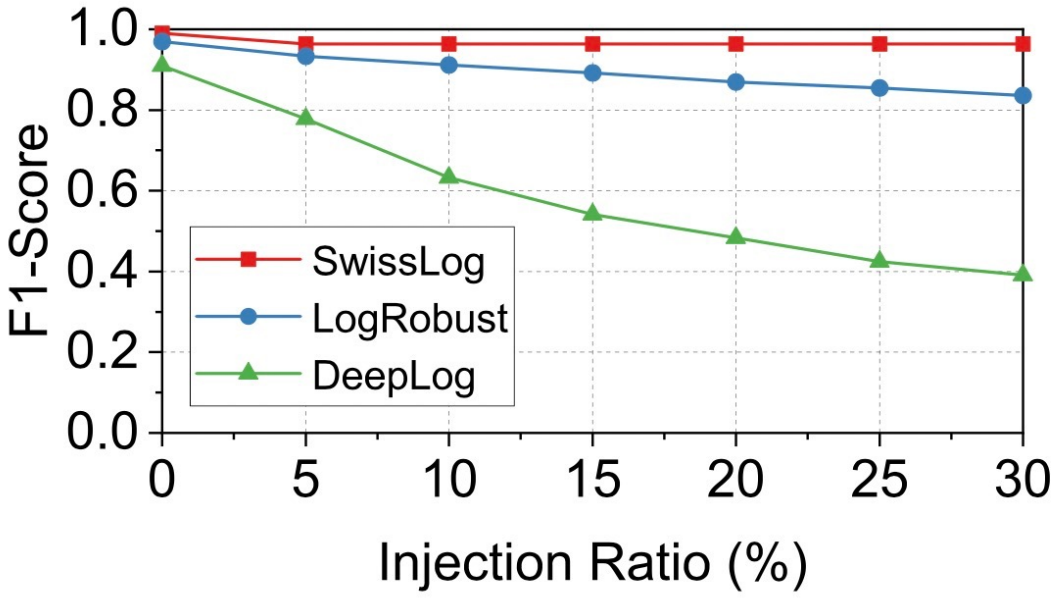


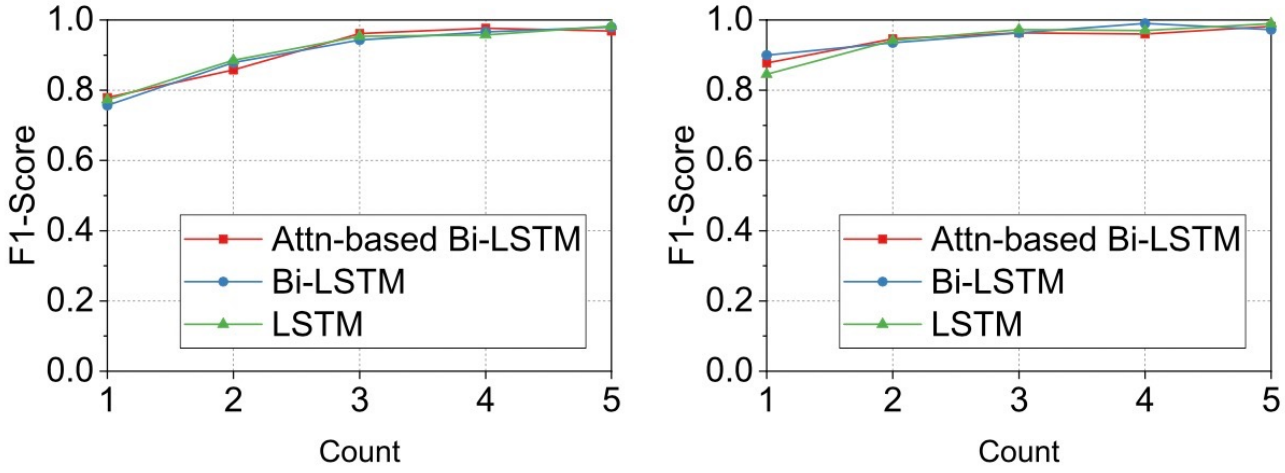
Fig. 11. F1-Score on the dataset TestingEvent

SwissLog with BERT encoders is almost not affected by changing events, showing a good robustness.

RQ4: The Effectiveness and Sensitivity of Time Embedding

TABLE IV
RESULTS ON DIFFERENT OPERATION FOR TIME EMBEDDING

Time embedding	Precision	Recall	F1-Score
Raw time	0.67	0.68	0.42
Mean	0.67	0.94	0.76
WO_Reciprocal	0.65	0.66	0.56
SwissLog	**0.92**	**0.99**	**0.95**



(a) Injected seconds=1 (b) Injected seconds=2

Fig. 12. F1-Score on TestingPerf

Attn-based Bi-LSTM outperforms Bi-LSTM and LSTM in detecting log sequential anomalies

Discussions

Threats To Validity:

- Dictionary selection
- Different tasks still have different execution time in different software systems

Efficiency:

- log parsing: 4.5 ms/l
- sentence embedding: 2.6ms/l
- network training: 800.0 ms/l
- anomaly detection: 4.5 ms/l

Conclusion

- Challenges in log-based anomaly detection:
  - Changing events
  - Performance issues
- We propose ***SwissLog,*** a robust and unified anomaly detection model for diverse faults:
  - Sequential log anomalies
  - Performance issues
- SwissLog includes 4 phases:

  Log parsing -> Sentence embedding -> Attention-based Bi-LSTM -> Anomaly detection
- Evaluation on real-world datasets and synthetic datasets to answer 4 research questions:
  - The effectiveness of log parsers and semantic-based models
  - The robustness on changing events
  - The effectiveness and sensitivity of time embedding
- Future works
  - Collect performance issues related datasets
  - Optimize the performance and adaptiveness of ***SwissLog***