Log-based Anomaly Detection

LOG6309E: Intelligent DevOps

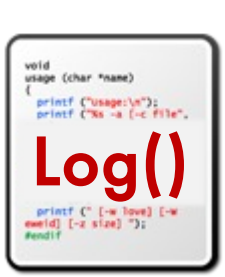
Research on Intelligent DevOps

Log parsing	Anomaly detection	Failure diagnostics
Monitoring automation	 Software System	Event/Incident prediction
Monitoring practices		Perf. analysis & prediction
Autonomous configuration	MLOps analysis	Security analysis

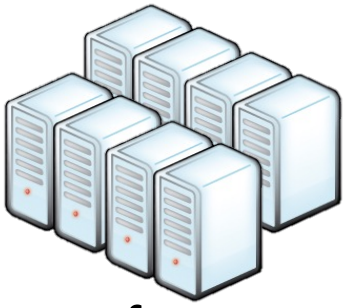
Monitoring code made right

Monitoring data used right

Developers insert logging code that produces log messages at runtime



Logging code



Software system

Log messages

```
Log.info("Stopping server on " + port);
```

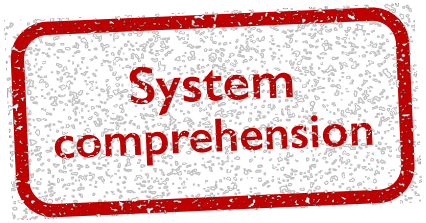
```
2016-07-23 17:56:16 INFO Stopping server on 8032
```

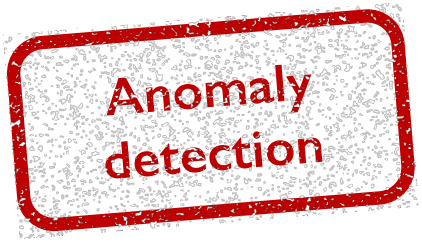
Log messages are widely used in software development and operation efforts

Fu et al., Contextual analysis of program logs for understanding system behaviors. MSR '13

Xu et al., Detecting large-scale system problems by mining console logs. SOSP '09

Yuan et al., Sherlog: Error diagnosis by connecting clues from run-time logs. ASPLOS '10





Research on Intelligent DevOps

Log parsing	Anomaly detection	Failure diagnostics
Monitoring automation	 Software System	Event/Incident prediction
Monitoring practices		Perf. analysis & prediction
Autonomous configuration	MLOps analysis	Security analysis

Monitoring code made right

Monitoring data used right

Tools and Benchmarks for Automated Log Parsing

Ref: Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. "**Tools and benchmarks for automated log parsing**." In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 121-130. IEEE, 2019.

Logs are used to record runtime information of a system

```
def get[T: ClassTag](blockId: BlockId): Option[BlockResult] = {
  val local = getLocalValues(blockId)
  if (local.isDefined) {
    logInfo(s"Found block $blockId locally")
    return local
  }
  val remote = getRemoteValues[T](blockId)
  if (remote.isDefined) {
    logInfo(s"Found block $blockId remotely")
    return remote
  }
  None
}
```

Logs are used to record runtime information of a system

```
logInfo(s"Found block $blockId remotely")
```

```
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_11 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_12 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_14 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_13 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_20 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_22 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_23 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_24 remotely
```

Logs are used to record runtime information of a system

```
logInfo(s"Found block $blockId remotely")
```

```
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_11 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_12 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_14 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_13 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_20 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_22 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_23 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_24 remotely
```

Logs are used to record runtime information of a system

```
logInfo(s"Found block $blockId remotely")
```

```
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_11 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_12 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_14 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_13 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_20 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_22 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_23 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_24 remotely
```

Logs are used to record runtime information of a system

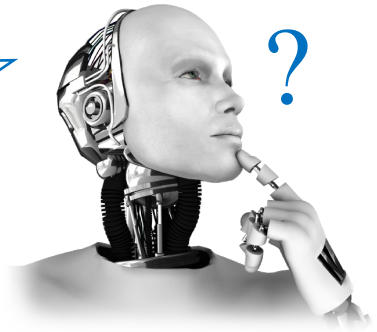
```
logInfo(s"Found block $blockId remotely")
```

```
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_11 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_12 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_14 remotely
17/06/09 20:11:10 INFO storage.BlockManager: Found block rdd_42_13 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_20 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_22 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_23 remotely
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_24 remotely
```

Logs are unstructured

Raw Log 2008-11-11 03:41:48. Received block blk_90 of size 67108864 from /10.250.18.114

What's this? I only understand structured data



Log Analysis AI

Log analysis models require structured input

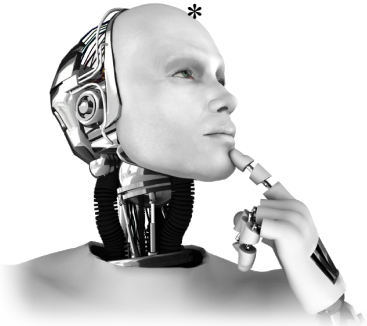
Raw Log

2008-11-11 03:41:48. Received block blk_90 of size 67108864 from /10.250.18.114

Log Parsing

Structured Log

EventId	EventTemplate	ParameterList
Event 1	Received block * of size * from *	[blk_90, 67108864, /10.250.18.114]



Oh! This is Event 1 happened on block blk_90

Log Analysis AI

Log parsing

Raw Log

2008-11-11 03:41:48. Received block blk_90 of size 67108864 from /10.250.18.114

Structured Log

EventId	EventTemplate	ParameterList
Event 1	Received block * of size * from *	[blk_90, 67108864, 67108864, /10.250.18.114]

The goal of log parsing is to distinguish between constant part and variable part from the log contents.

Log parsing

Raw Log

2008-11-11 03:41:48. Received block blk_90 of size 67108864 from /10.250.18.114

Structured Log

EventId	EventTemplate	ParameterList
Event 1	Received block * of size * from *	[blk_90, 67108864, 67108864, /10.250.18.114]

The goal of log parsing is to distinguish between constant part and variable part from the log contents.

Log parsing

Raw Log

2008-11-11 03:41:48. Received block blk_90 of size 67108864 from /10.250.18.114

Structured Log

EventId	EventTemplate	ParameterList
Event 1	Received block * of size * from *	[blk_90, 67108864, 67108864, /10.250.18.114]

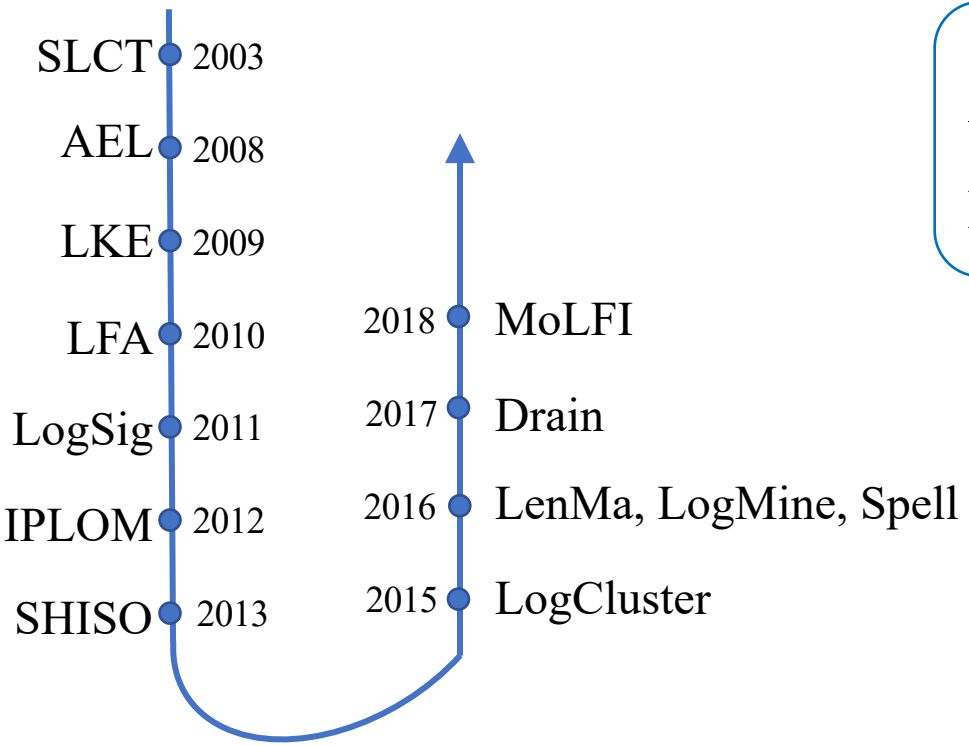
The goal of log parsing is to distinguish between constant part and variable part from the log contents.

Log parsing is

the first and a significant step

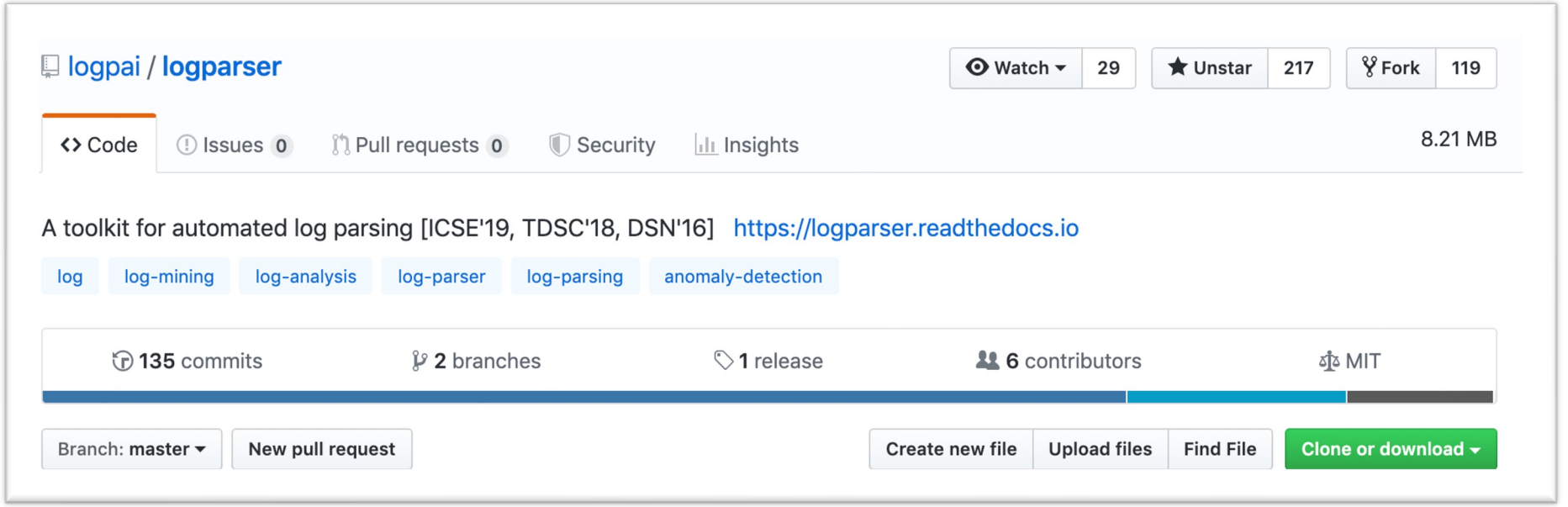
towards the whole process of log analysis

Many log parsers are produced



How they perform?
Which to choose?
How to use?

Toolkit: Logparser



The toolkit containing 13 log parsing methods is open source on

https://github.com/logpai/logparser

Datasets

Dataset	Description	Time Span	Data Size	#Messages	#Templates (total)	#Templates (2k)
Distributed system logs						
HDFS	Hadoop distributed file system log	38.7 hours	1.47 GB	11,175,629	48	14
Hadoop	Hadoop mapreduce job log	N.A.	48.61 MB	394,308	298	114
Spark	Spark job log	2.75 hours	2.72 GB	33,236,604	456	36
ZooKeeper	ZooKeeper service log	26.7 days	9.95 MB	74,380	95	50
OpenStack	OpenStack software log	N.A.	60.01 MB	207,820	51	43
Supercomputer logs						
BGL	Blue Gene/L supercomputer log	214.7 days	708.76 MB	4,747,963	619	120
HPC	High performance cluster log	31.9 days	32.00 MB	433,489	104	46
Thunderbird	Thunderbird supercomputer log	244 days	29.60 GB	211,212,192	4,040	149
Operating system logs						
Windows	Windows event log	226.7 days	26.09 GB	114,608,388	4,833	50
Linux	Linux system log	263.9 days	2.25 MB	25,567	488	118
Mac	Mac OS log	7.0 days	16.09 MB	117,283	2,214	341
Mobile system logs						
Android	Android framework log	N.A.	3.38 GB	30,348,042	76,923	166
HealthApp	Health app log	10.9 days	22.44 MB	253,395	220	75
Server application logs						
Apache	Apache server error log	263.9 days	4.90 MB	56,481	44	6
OpenSSH	OpenSSH server log	28.8 days	70.02 MB	655,146	62	27
Standalone software logs						
Proxifier	Proxifier software log	2.4 hours	2.91 MB	21,329	9	8

Distributed system logs
Supercomputer logs
Operating system logs
Mobile system logs
Server application logs
Standalone software logs

Datasets

Dataset	Description	Time Span	Data Size	#Messages	#Templates (total)	#Templates (2k)
Distributed system logs						
HDFS	Hadoop distributed file system log	38.7 hours	1.47 GB	11,175,629	48	14
Hadoop	Hadoop mapreduce job log	N.A.	48.61 MB	394,308	298	114
Spark	Spark job log	N.A.	2.75 GB	33,236,604	456	36
ZooKeeper	ZooKeeper service log	26.7 days	9.95 MB	74,380	95	50
OpenStack						43
BGL						120
HPC						46
Thunderbir						149
Windows						50
Linux						118
Mac						341
Android						166
HealthApp	Health app log	10.5 days	22.44 MB	253,395	220	75
Server application logs						
Apache	Apache server error log	263.9 days	4.90 MB	56,481	44	6
OpenSSH	OpenSSH server log	28.4 days	70.02 MB	655,146	62	27
Standalone software logs						
Proxifier	Proxifier software log	N.A.	2.42 MB	21,329	9	8

Time Span: ~5 years

# of lines: 440 million

Total Size: 77 GB

2k labeled log message for each dataset

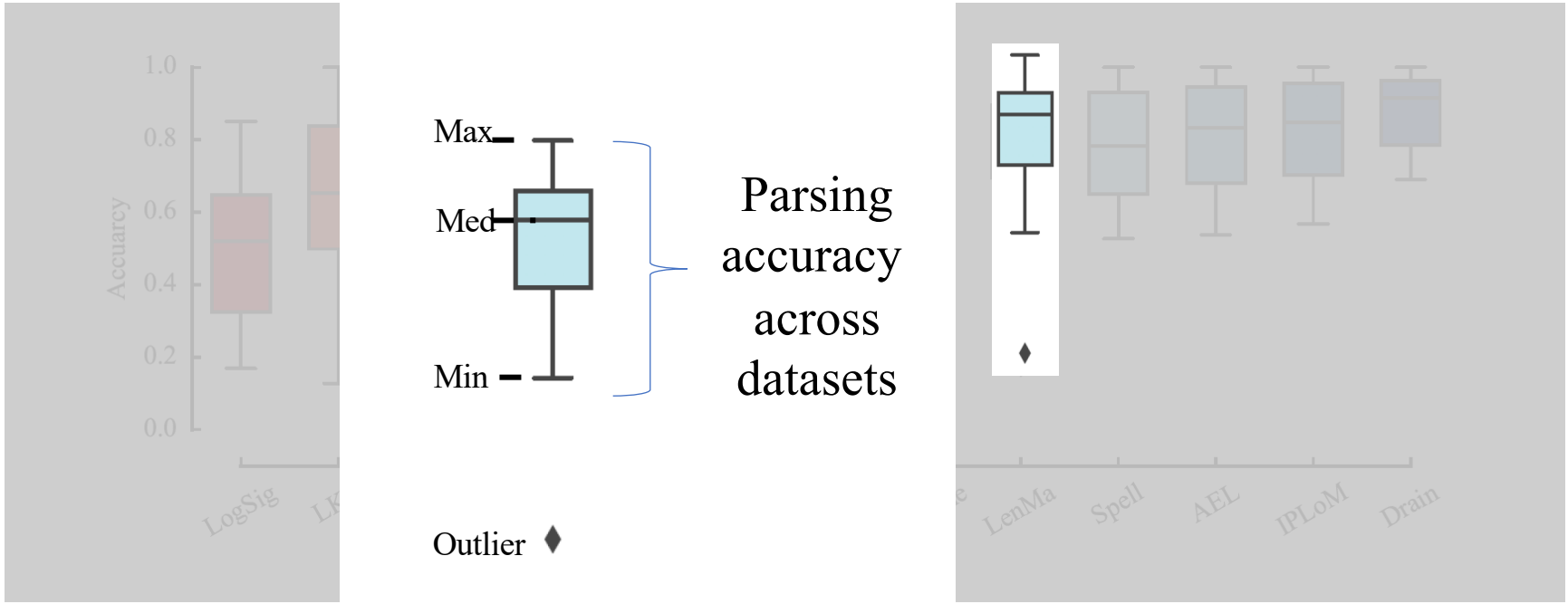
Evaluation

RQ1: What is the **accuracy** of the state-of-the-art log parsing methods?

RQ2: What is the **robustness** of these methods on different types and volumes of logs?

RQ3: What is the **efficiency** of these methods?

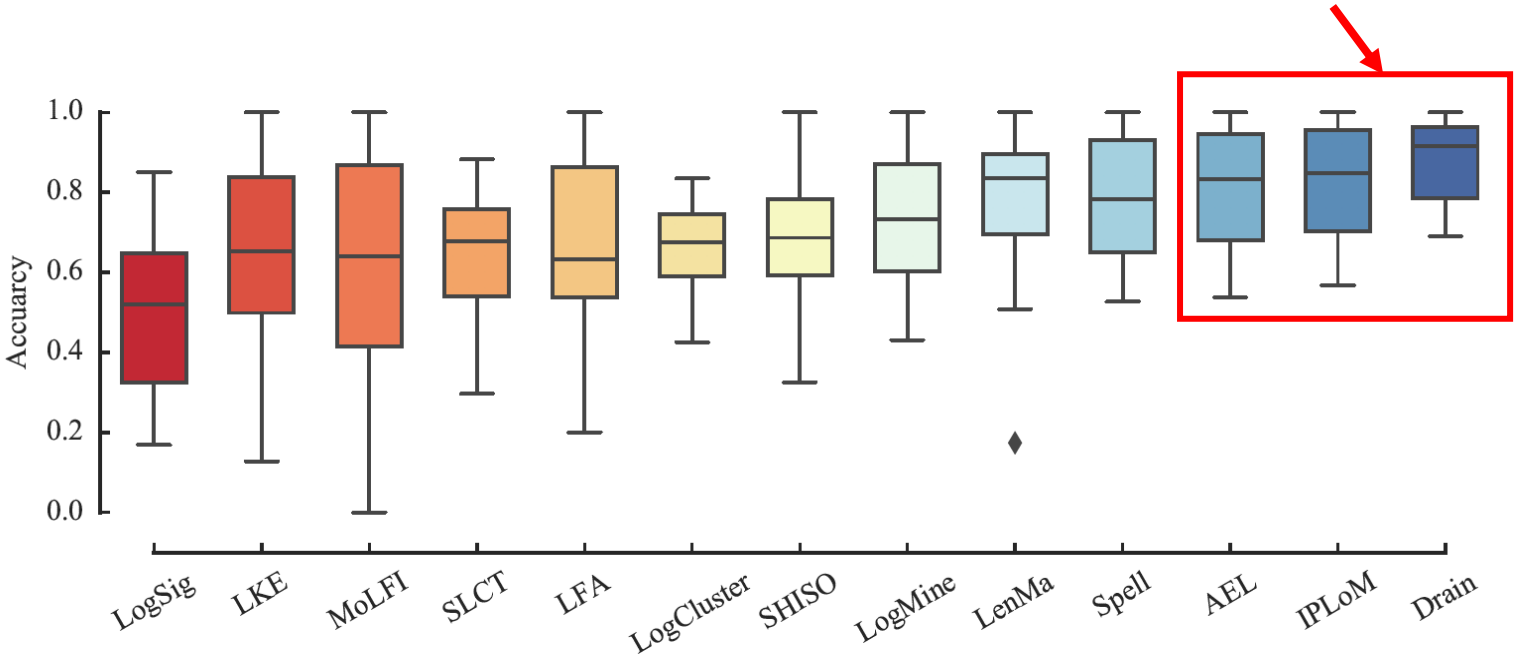
RQ1: Accuracy RQ2: Robustness RQ3: Efficiency



Accuracy Distribution of Log Parsers on **Different Log Types**

RQ1: Accuracy RQ2: Robustness RQ3: Efficiency

Drain* > IPLoM > AEL > Others



Accuracy Distribution of Log Parsers on Different Log Types

* [**ICWS'17**] Drain: An Online Log Parsing Approach with Fixed Depth Tree, by Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu.

RQ1: Accuracy RQ2: Robustness RQ3: Efficiency

Parsing Methods

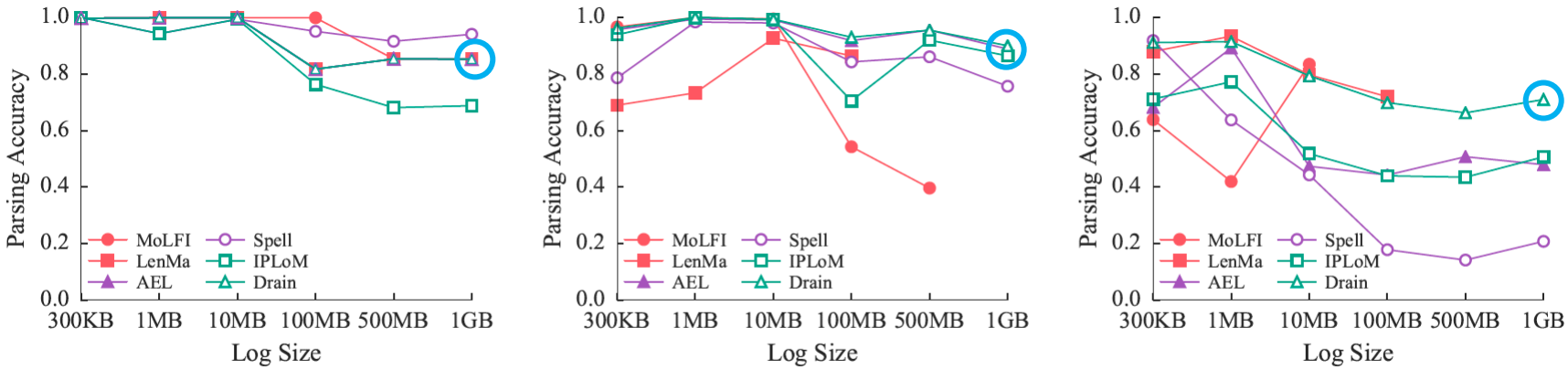
Dataset	SLCT	AEL	IPLoM	LKE	LFA	LogSig	SHISO	LogCluster	LenMa	LogMine	Spell	Drain	MoLFI	Best
HDFS	0.545	**0.998**	**1***	**1***	0.885	0.850	**0.998**	0.546	**0.998**	0.851	**1***	**0.998**	**0.998**	**1**
Hadoop	0.423	0.538	**0.954**	0.670	**0.900**	0.633	0.867	0.563	0.885	0.870	0.778	**0.948**	**0.957***	**0.957**
Spark	0.685	**0.905**	**0.920**	0.634	**0.994***	0.544	**0.906**	0.799	0.884	0.576	**0.905**	**0.920**	0.418	**0.994**
Zookeeper	0.726	**0.921**	**0.962**	0.438	0.839	0.738	0.660	0.732	0.841	0.688	**0.964**	**0.967***	0.839	**0.967**
OpenStack	0.867	0.758	0.871*	0.787	0.200	0.200	0.722	0.696	0.743	0.743	0.764	0.733	0.213	0.871
BGL	0.573	0.758	**0.939**	0.128	0.854	0.227	0.711	0.835	0.69	0.723	0.787	**0.963***	**0.960**	**0.963**
HPC	0.839	**0.903***	0.824	0.574	0.817	0.354	0.325	0.788	0.830	0.784	0.654	0.887	0.824	**0.903**
Thunderb.	0.882	**0.941**	0.663	0.813	0.649	0.694	0.576	0.599	**0.943**	**0.919**	0.844	**0.955***	0.646	**0.955**
Windows	0.697	0.690	0.567	**0.990**	0.588	0.689	0.701	0.713	0.566	**0.993**	**0.989**	**0.997***	0.406	**0.997**
Linux	0.297	0.673	0.672	0.519	0.279	0.169	0.701	0.629	0.701*	0.612	0.605	0.690	0.284	0.701
Mac	0.558	0.764	0.673	0.369	0.599	0.478	0.595	0.604	0.698	0.872*	0.757	0.787	0.636	0.872
Android	0.882	0.682	0.712	**0.909**	0.616	0.548	0.585	0.798	0.880	0.504	**0.919***	**0.911**	0.788	**0.919**
HealthApp	0.331	0.568	0.822*	0.592	0.549	0.235	0.397	0.531	0.174	0.684	0.639	0.780	0.440	0.822
Apache	0.731	**1***	**1***	**1***	**1***	0.582	**1***	0.709	**1***	**1***	**1***	**1***	**1***	**1**
OpenSSH	0.521	0.538	0.802	0.426	0.501	0.373	0.619	0.426	**0.925***	0.431	0.554	0.788	0.500	**0.925**
~~Proxifier~~	~~0.518~~	~~0.518~~	~~0.515~~	~~0.495~~	~~0.026~~	**~~0.967*~~**	~~0.517~~	**~~0.951~~**	~~0.508~~	~~0.517~~	~~0.527~~	~~0.527~~	~~0.013~~	**~~0.967~~**
Average	0.637	0.754	0.777	0.563	0.652	0.482	0.669	0.665	0.721	0.694	0.751	0.865*	0.605	N.A.

Dataset

Accuracy of Log Parsers on Different Datasets

RQ1: Accuracy RQ2: Robustness RQ3: Efficiency

Drain achieves relatively stable accuracy

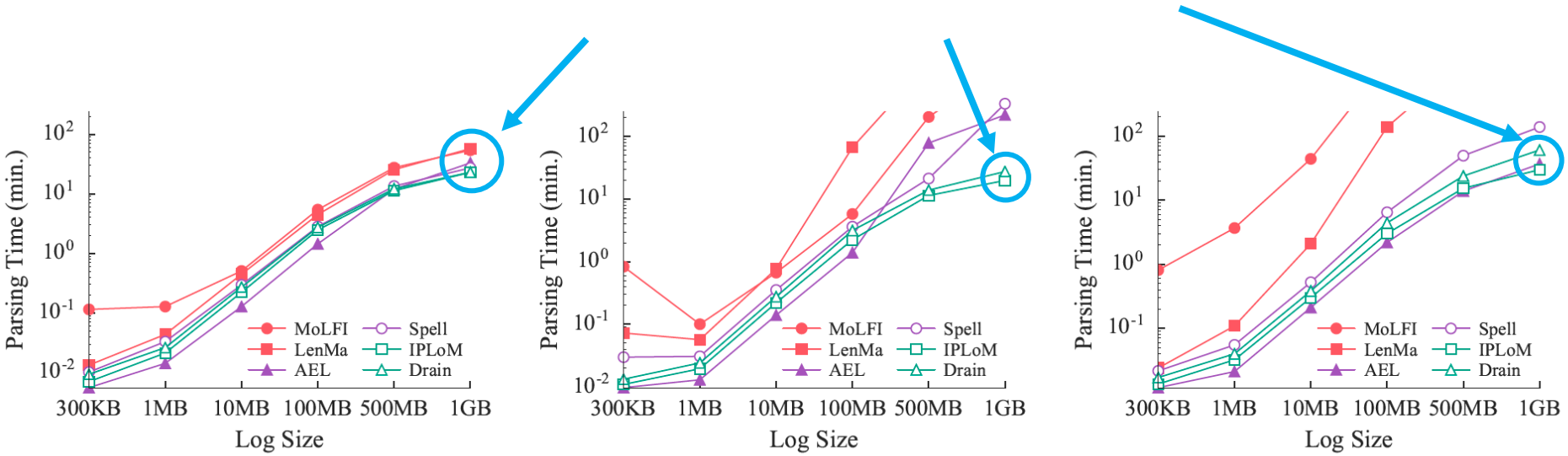


(a) HDFS (b) BGL (c) Android

Accuracy of Log Parsers on Varying Log Size

RQ1: Accuracy RQ2: Robustness RQ3: Efficiency

IPLoM ≈ AEL ≈ Drain > others



(a) HDFS (b) BGL (c) Android

Efficiency of Log Parsers on Varying Log Size

Log Parsing Exercise (Logparser)

- Demo instructions:
  - https://logparser.readthedocs.io/en/latest/demo.html
- Check the raw log to be parsed:
  - https://github.com/logpai/logparser/blob/master/logs/HDFS/HDFS_2k.log
- Make sure Docker is installed

1. Create an empty folder (name it logparser)
2. Load and run the logparser Docker container:
   - docker run --name logparser_py2 -it -v logparser:/logparser logpai/logparser:py2 bash
3. Clone logparser into the container:
   - git clone https://github.com/logpai/logparser.git /logparser/
4. Run log parsing using any parser available (e..g, Drain):
   - Cd /logparser/demo/; python Drain_demo.py
5. Check the parsed log data (csv files)
   - HDFS_2k.log_structured.csv and HDFS_2k.log_templates.csv

PILAR: Studying and Mitigating the Influence of Configurations on Log Parsing

Ref: Dai, Hetong, Yiming Tang, Heng Li, and Weiyi Shang. "**PILAR: Studying and Mitigating the Influence of Configurations on Log Parsing**." In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 818-829. IEEE, 2023.

Log parsing pipeline in LogPai

Datasets

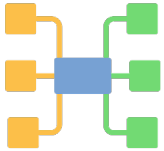
- Android
- Apache
- BGL
- Hadoop
- HDFS
- …

Log parsing tools

- Drain
- AEL
- IPLoM
- LenMa
- Spell
- …

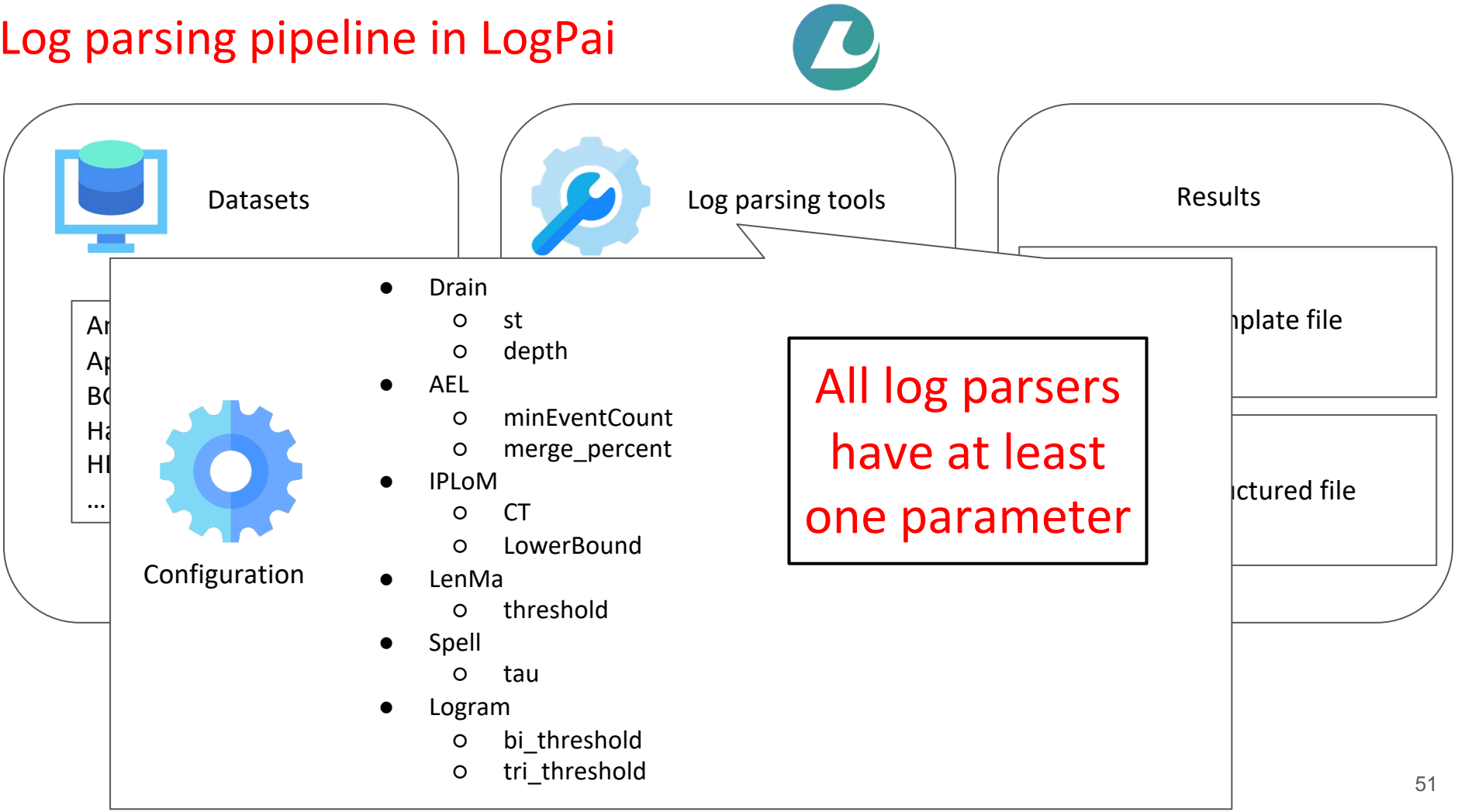
Results

Template file

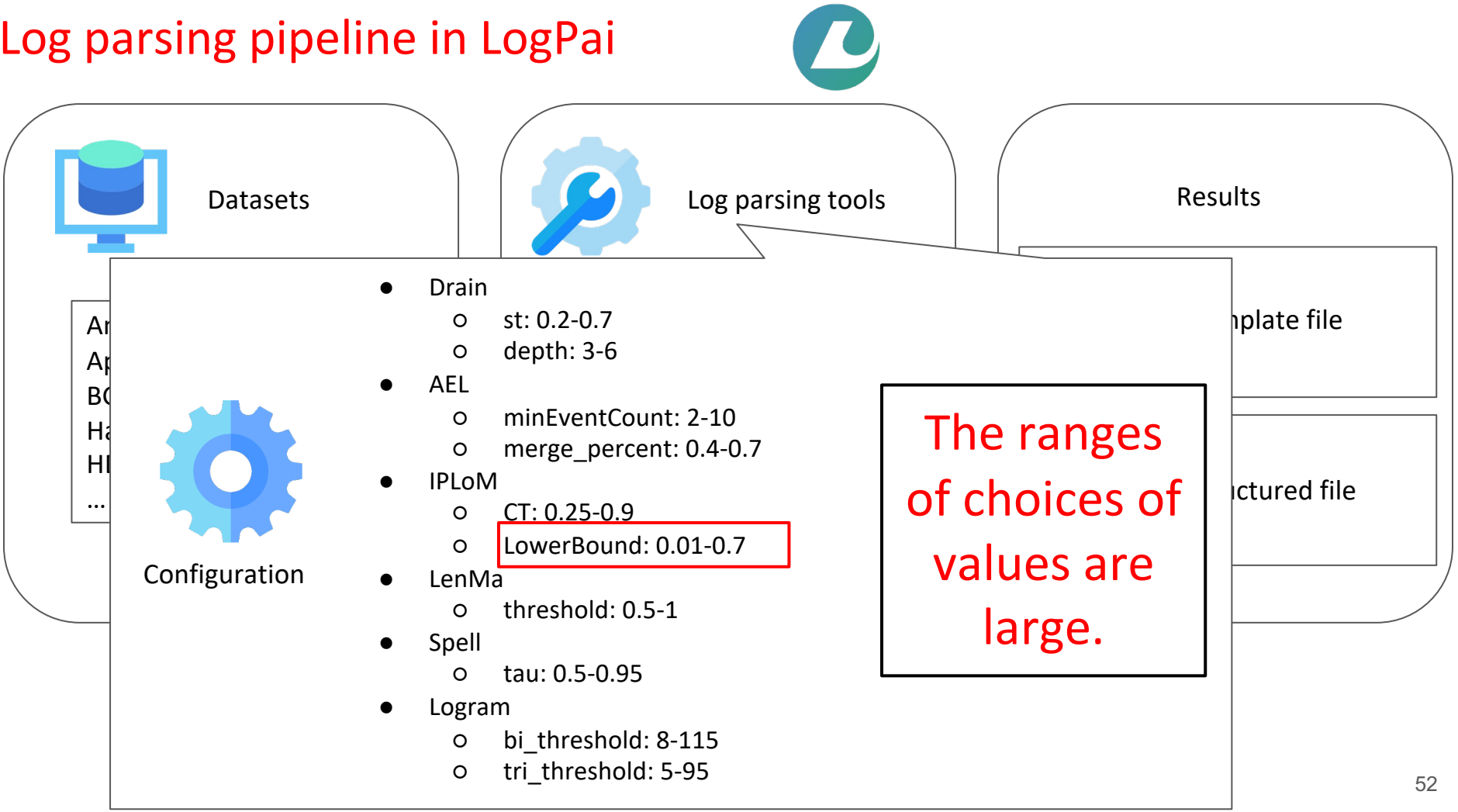


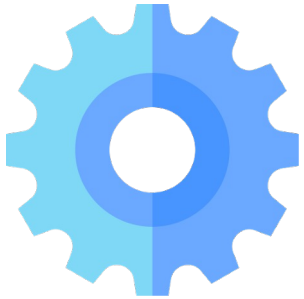
Structured file

Log parsing pipeline in LogPai



Log parsing pipeline in LogPai



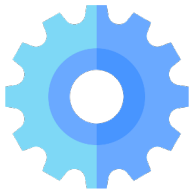


How big is the influence of configuration parameter values on log parsing results?

Log parsing pipeli

Datasets

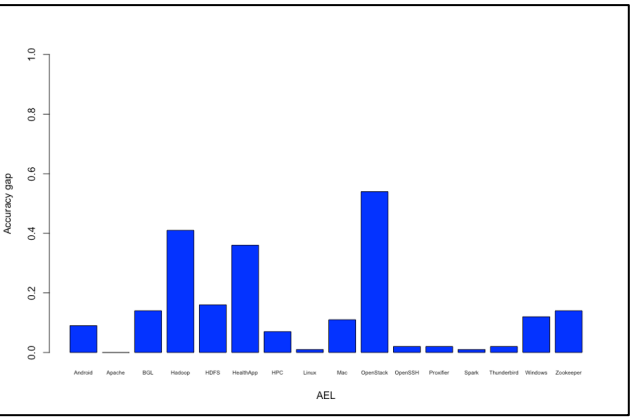
Varying parameters on the same dataset

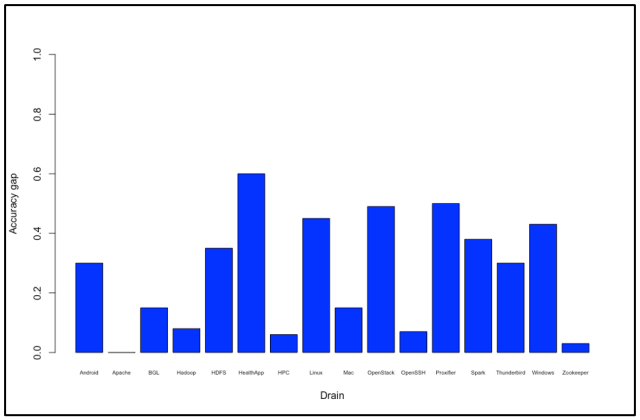


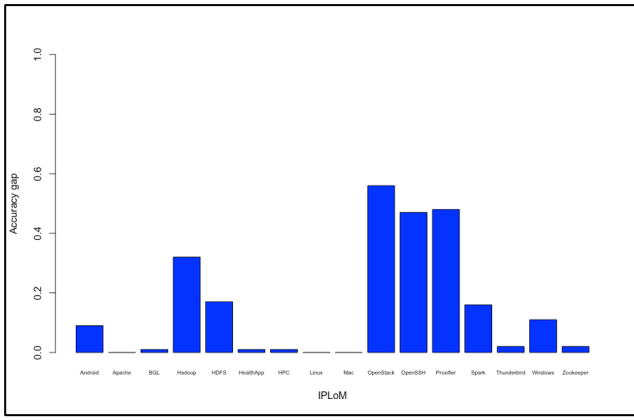
Configuration

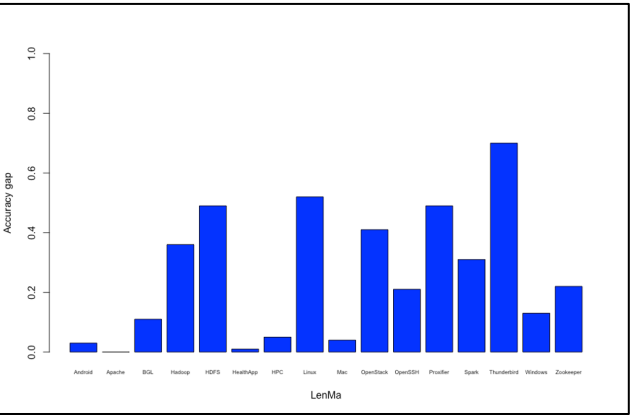
- Drain
  - st: 0.2-0.7
  - depth: 3-6
- AEL
  - minEventCount: 2-10
  - merge_percent: 0.4-0.7
- IPLoM
  - CT: 0.25-0.9
  - LowerBound: 0.01-0.7
- LenMa
  - threshold: 0.5-1
- Spell
  - tau: 0.5-0.95
- Logram
  - bi_threshold: 8-115
  - tri_threshold: 5-95

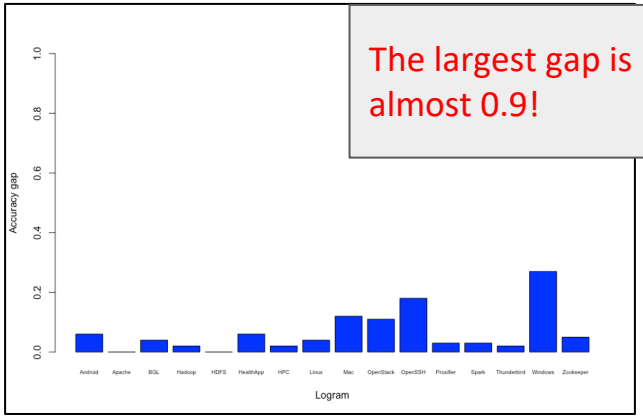
The gap of accuracy is HUGE when altering parameter values



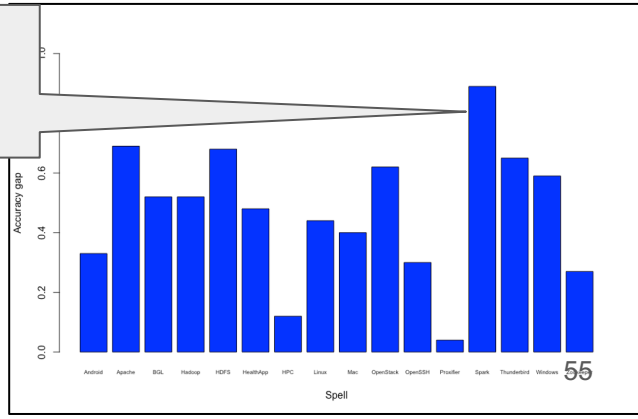




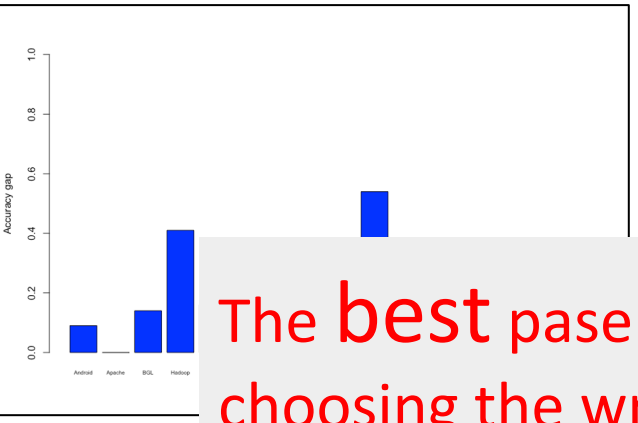


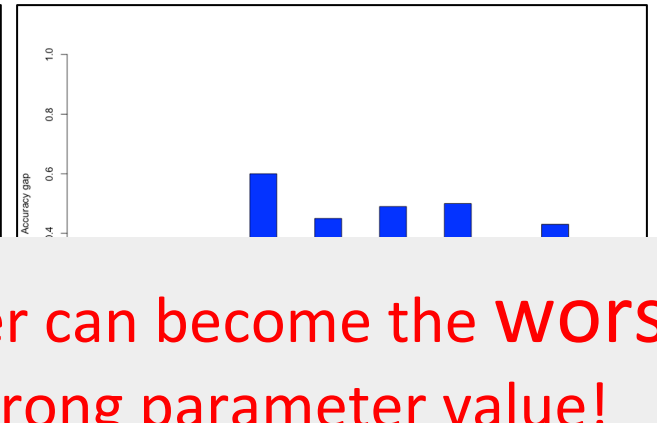


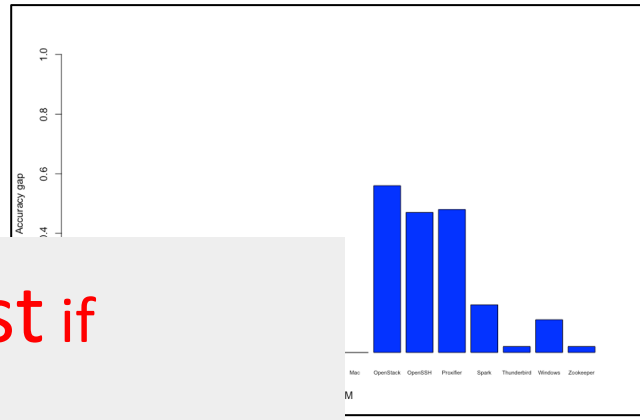
The largest gap is almost 0.9!



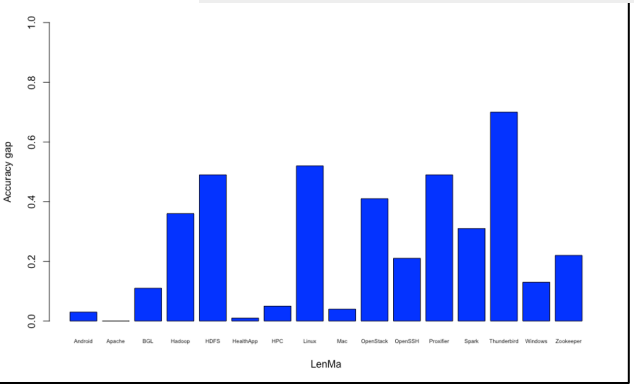
The gap of accuracy is HUGE when altering parameter values

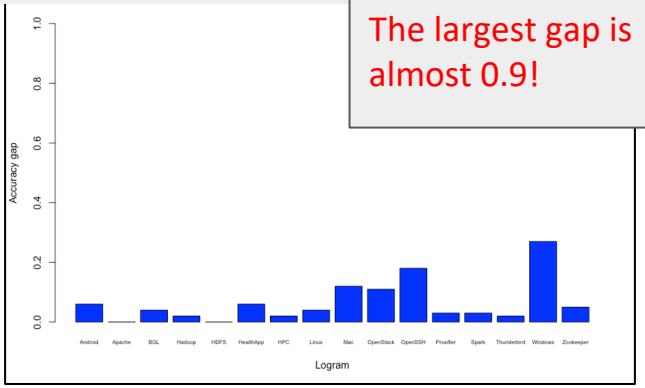




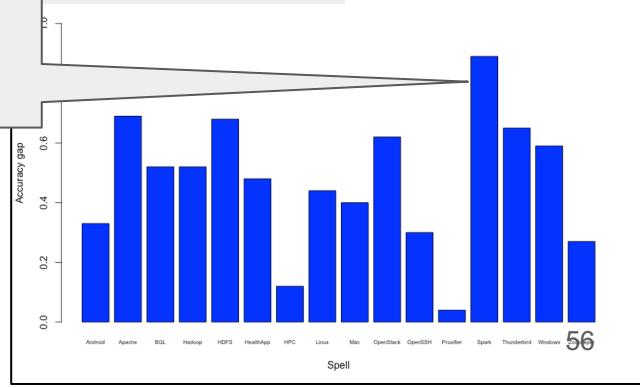


The **best** paser can become the **worst** if choosing the wrong parameter value!





The largest gap is almost 0.9!



Log parsing pipeline in LogPai

Datasets

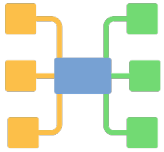
Android
Apache
BGL
Hadoop
HDFS
…

Log parsing tools

Drain
AEL
IPLoM
LenMa
Spell
…

Results

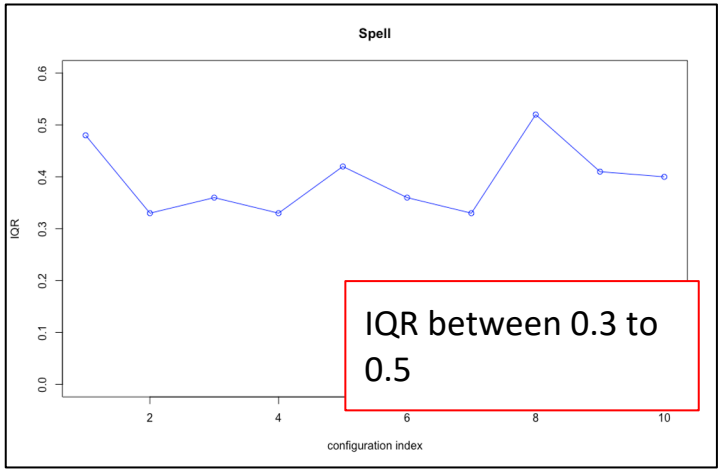
Template file



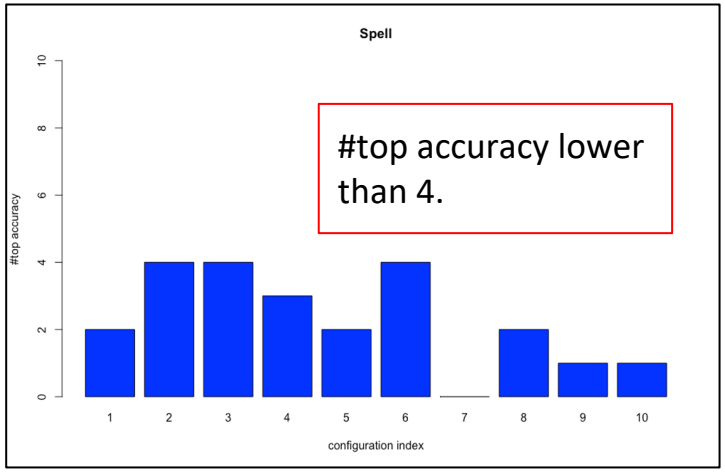
Structured file

Fixed parameters on different datasets

None of the parsers can achieve constant accurate parsing result with fixed parameters on different datasets



Inter-Quartile Range (IQR): the difference between the first quartile and the third quartiles of the accuracy distribution across the log datasets

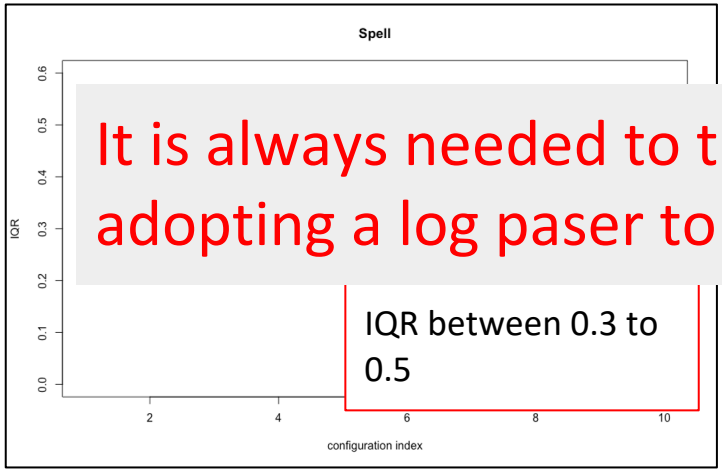


#top accuracy: the number of datasets where the log parser using the combination of parameters achieves the highest accuracy

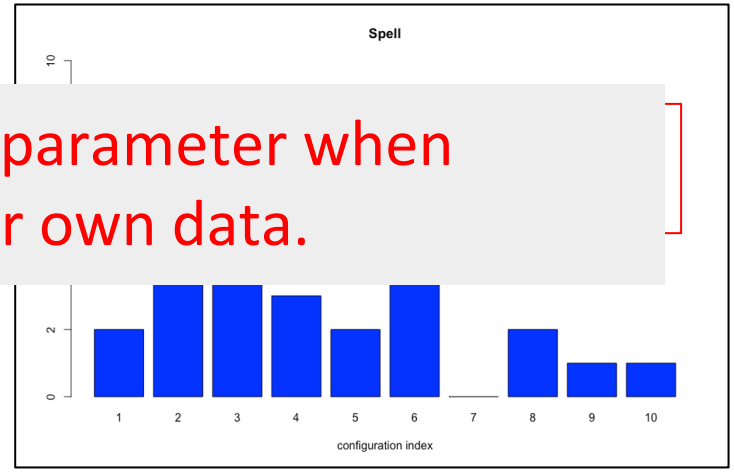
Same parameter has different accuracy among datasets!

Best performing parameter only work for a few datasets

None of the parsers can achieve constant accurate parsing result with fixed parameters on different datasets



Inter-Quartile Range (IQR): the difference between the first quartile and the third quartiles of the accuracy distribution across the log datasets



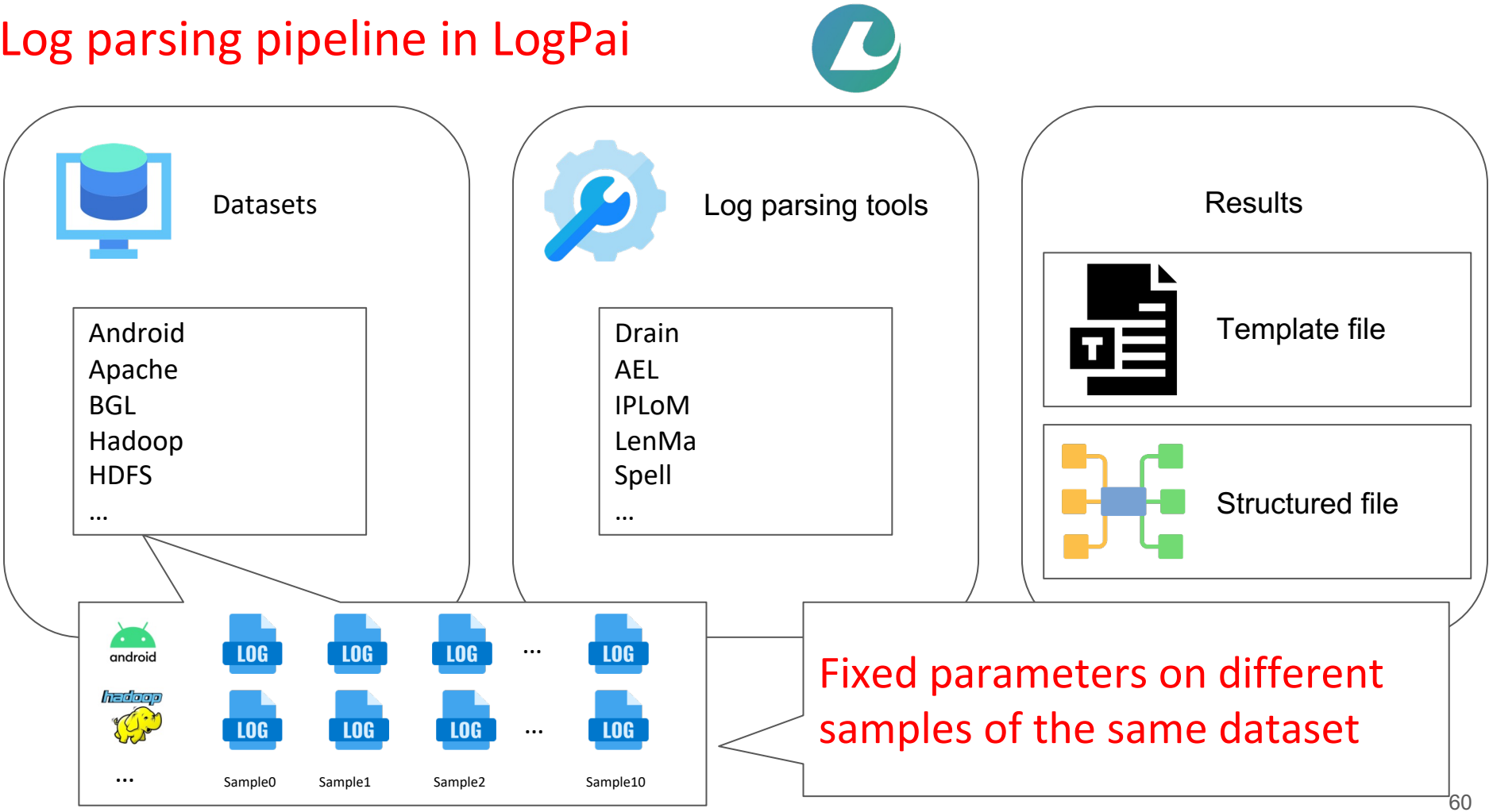
#top accuracy: the number of datasets where the log parser using the combination of parameters achieves the highest accuracy

It is always needed to tune parameter when adopting a log paser to their own data.

Same parameter has different accuracy among datasets!

Best performing parameter only work for a few datasets

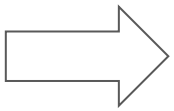
Log parsing pipeline in LogPai

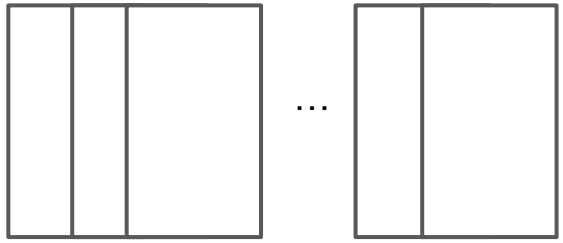


Fixed parameters on different samples of the same dataset

Log File

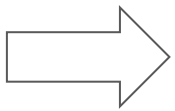
Generating sample datasets

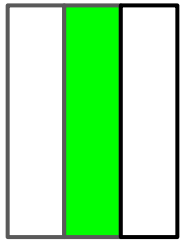




11 samples with 10000 log lines, each two consecutive samples share 5,000 overlapping log lines

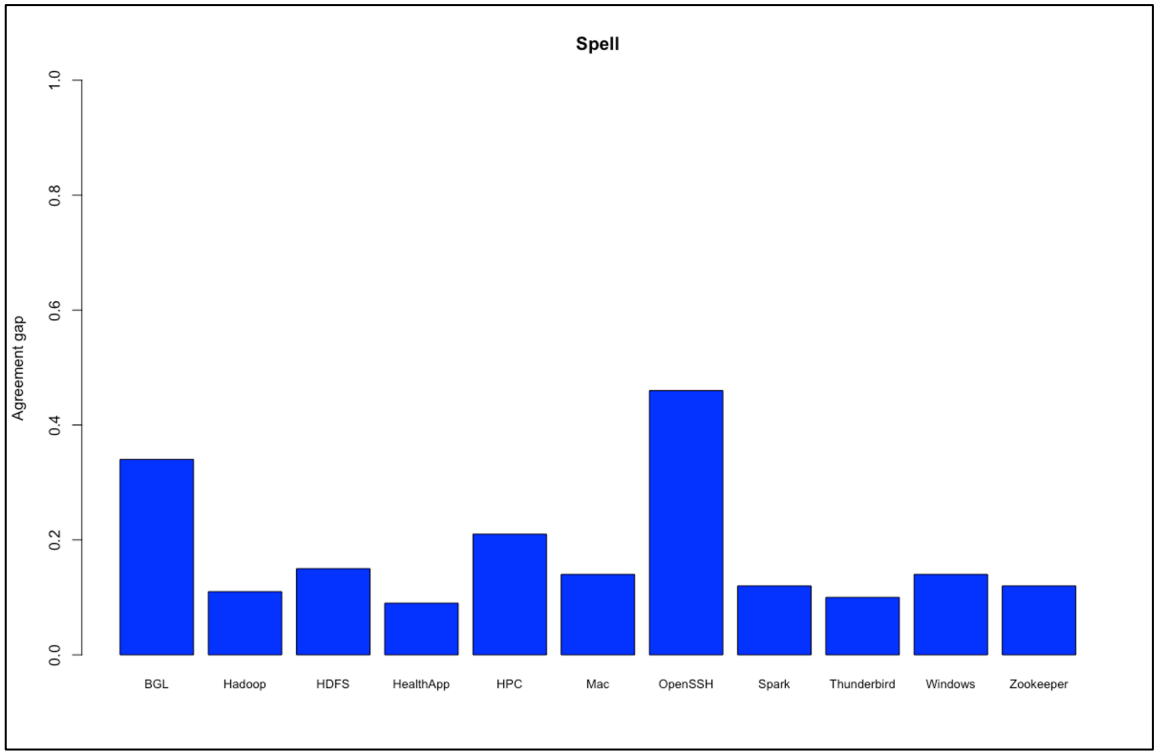
Measuring agreement of parsing results





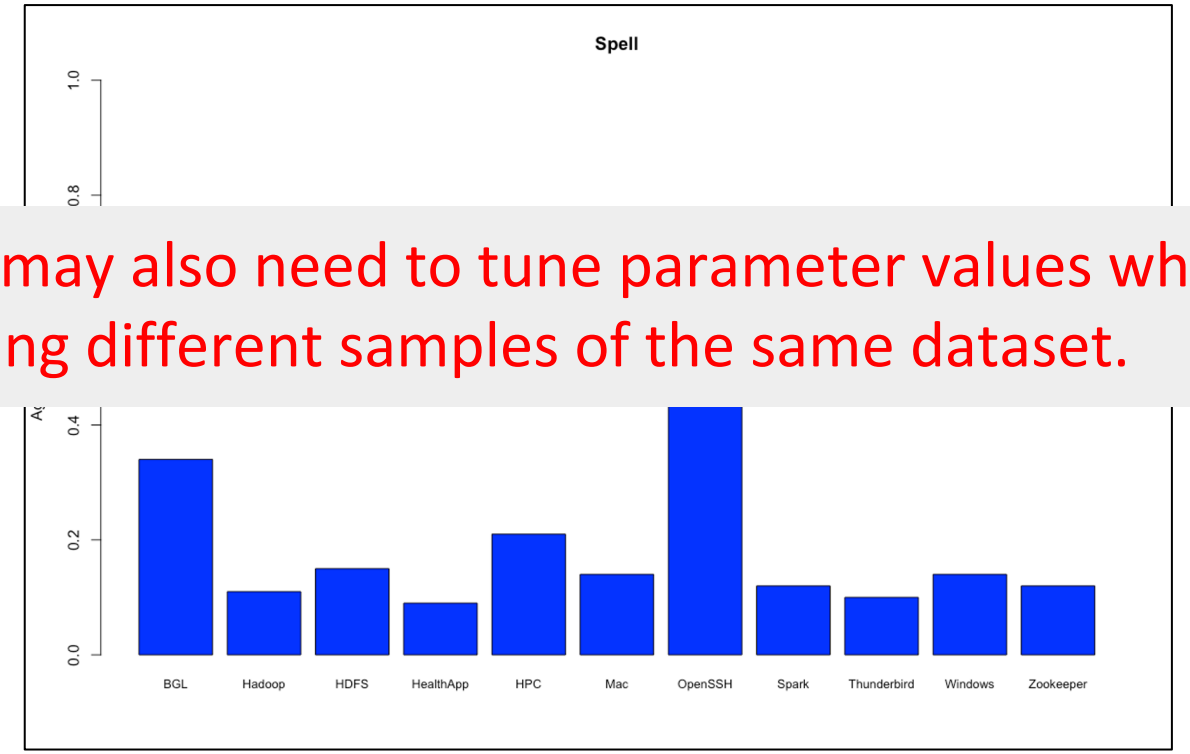
Check the agreement of overlapping part

Parameter values can influence stability of the log parsers on different samples



The agreement gap is calculated by the highest agreement and lowest agreement with different parameter

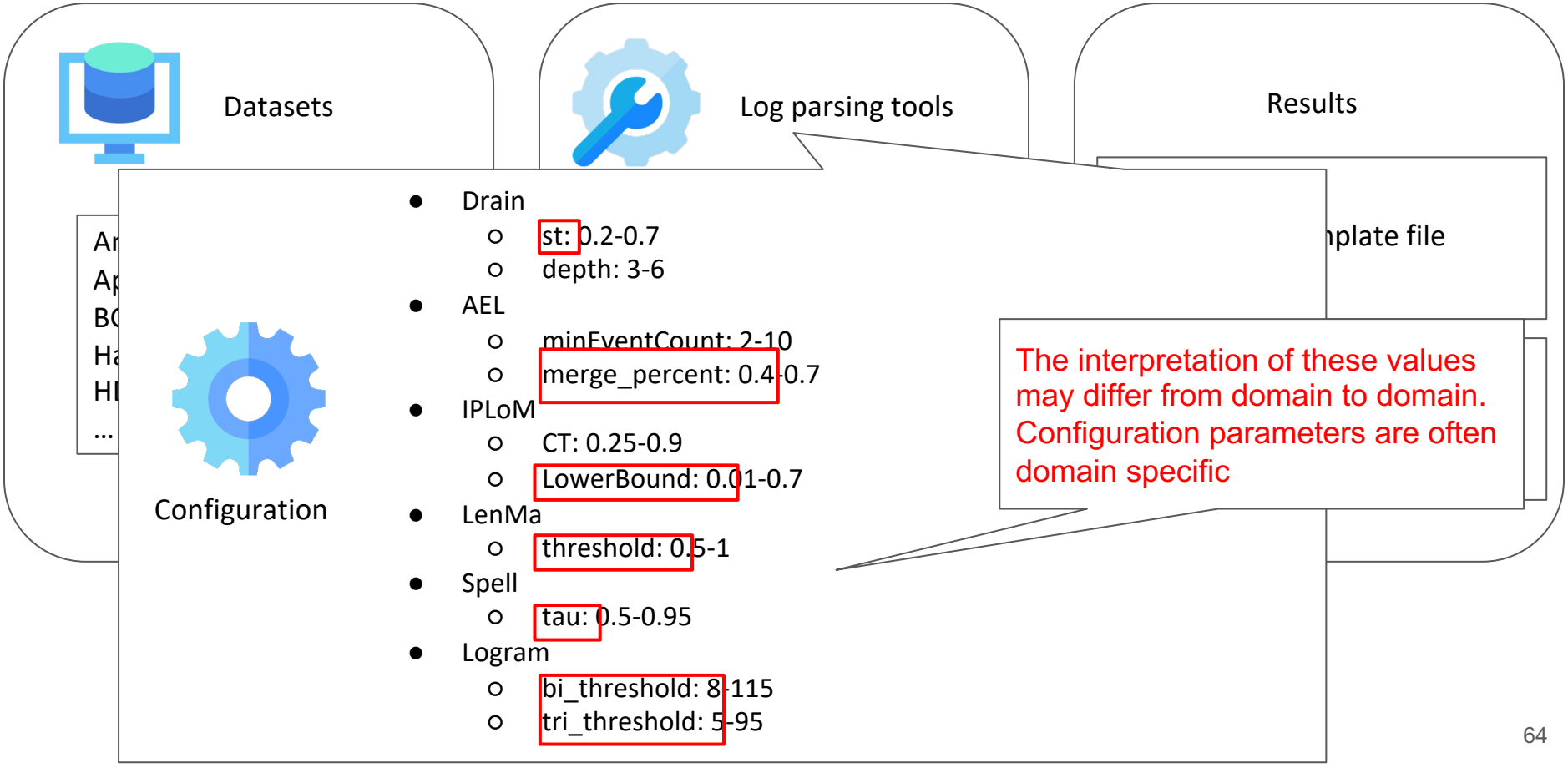
Parameter values can influence stability of the log parsers on different samples



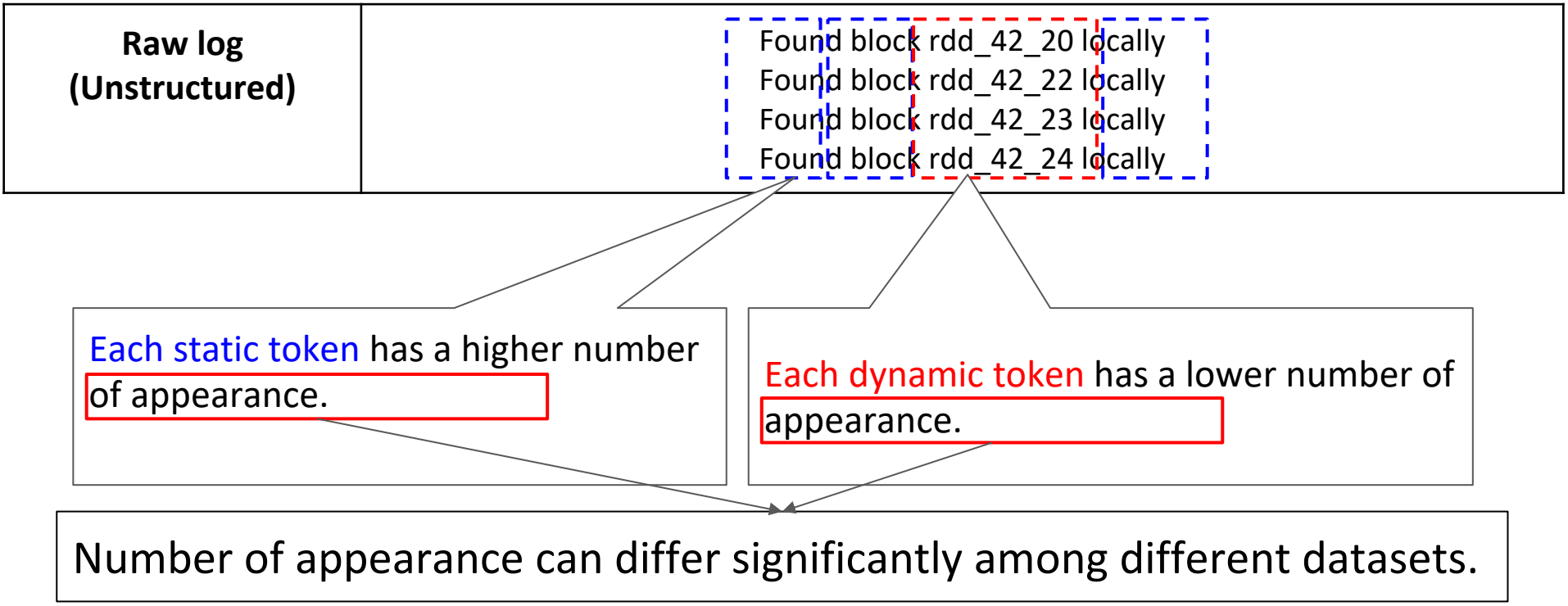
One may also need to tune parameter values when parsing different samples of the same dataset.

The agreement gap is calculated by the highest agreement and lowest agreement with different parameter

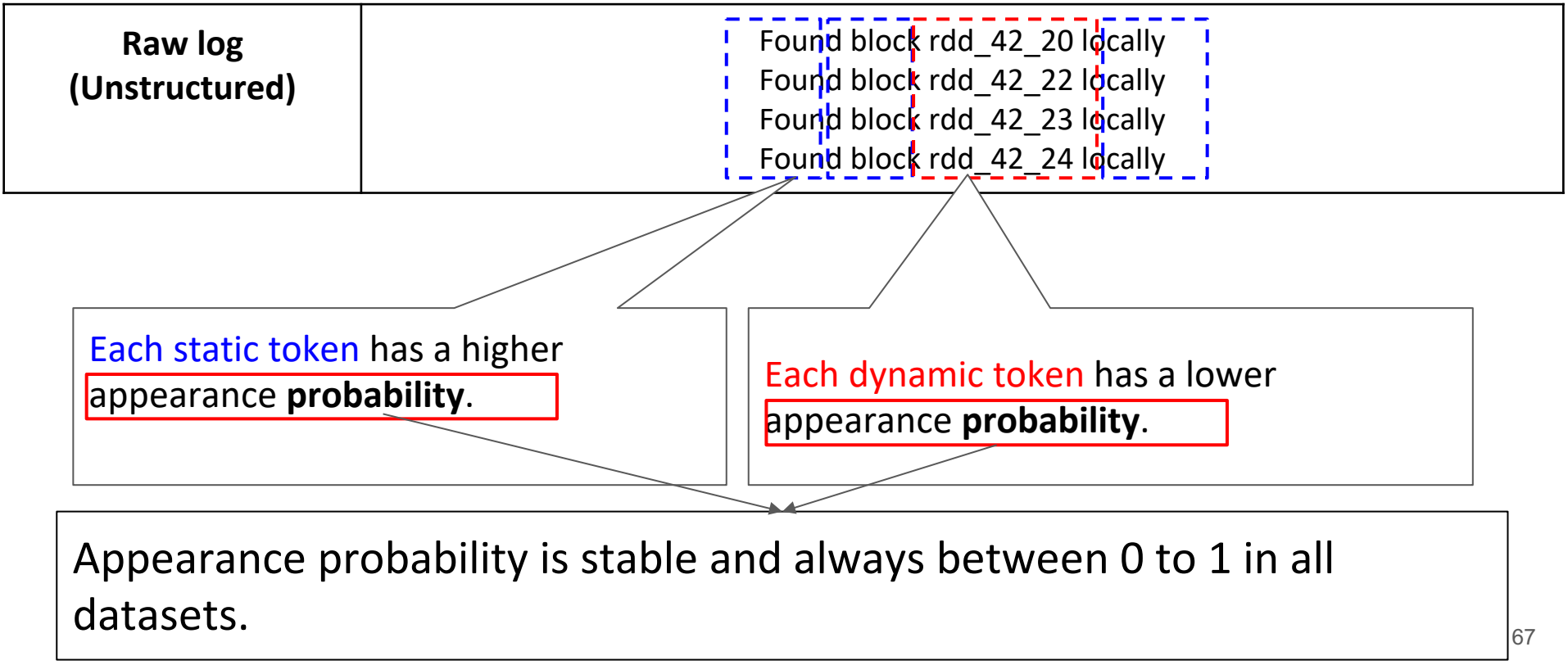
Reasons for configuration parameters being sensitive



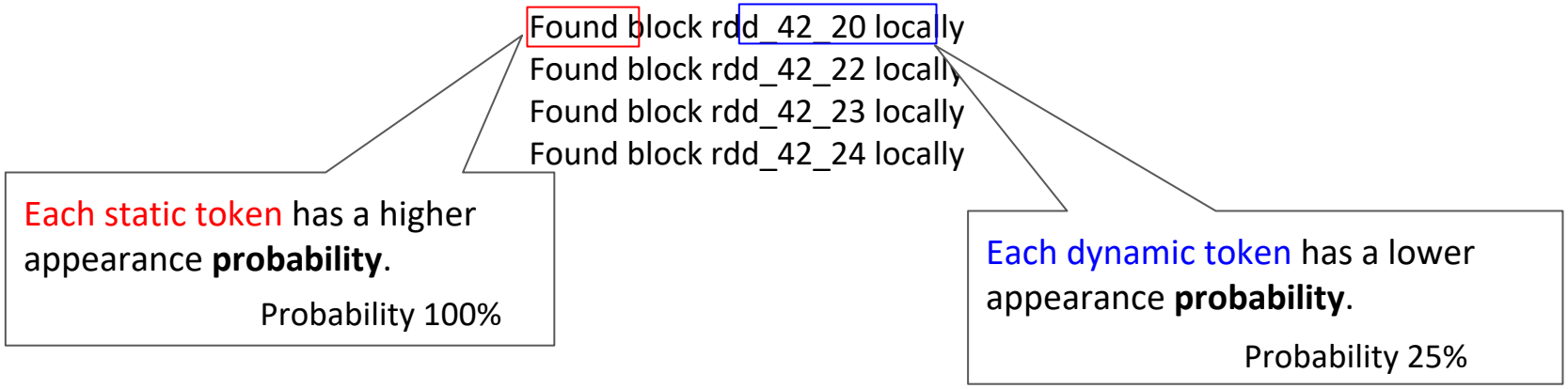
Why do parameter values impact Logram?



We choose an insensitive measurement instead of appearance



Main idea to reduce the effect of configuration parameters



We calculate probability by calculating $\frac{appearance\ of\ n\ gram}{appearance\ of\ (n-1)\ gram}$

The entropy would be likely to all follow the same characteristics, i.e., stable to each other

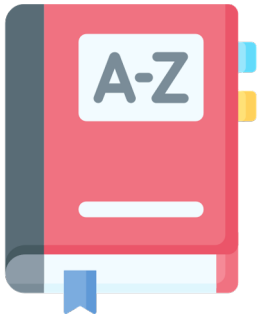
Pipeline of PILAR

Raw log

Preprocess

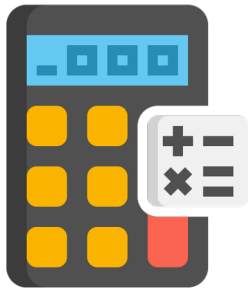
Cooked log

Generate



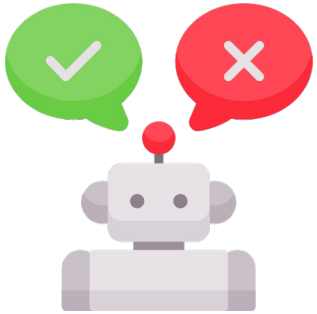
n-grams dictionary

Calculate



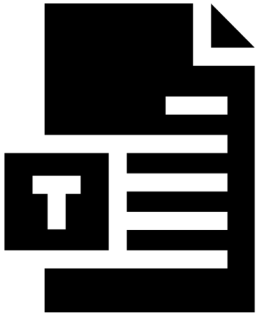
Entropy-based probability for log tokens

Parse



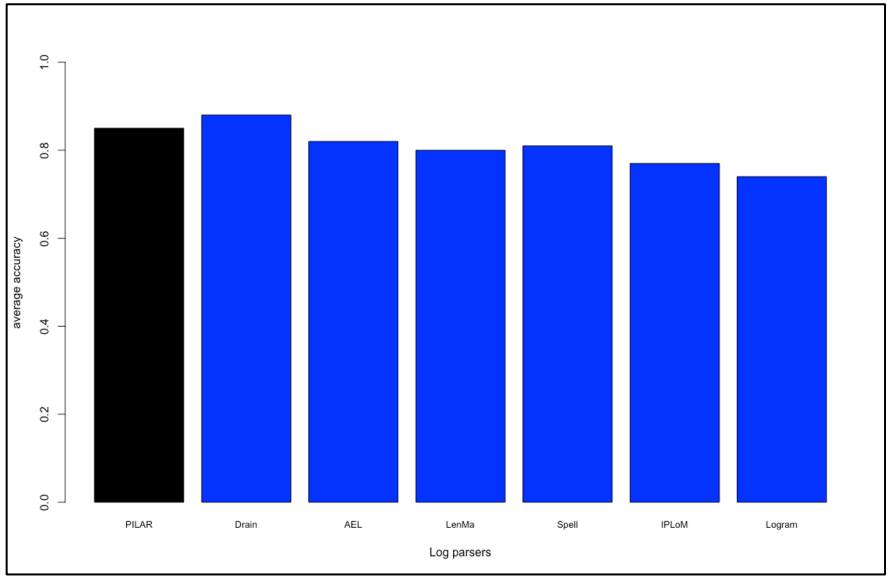
Static text/dynamic variable

Generate

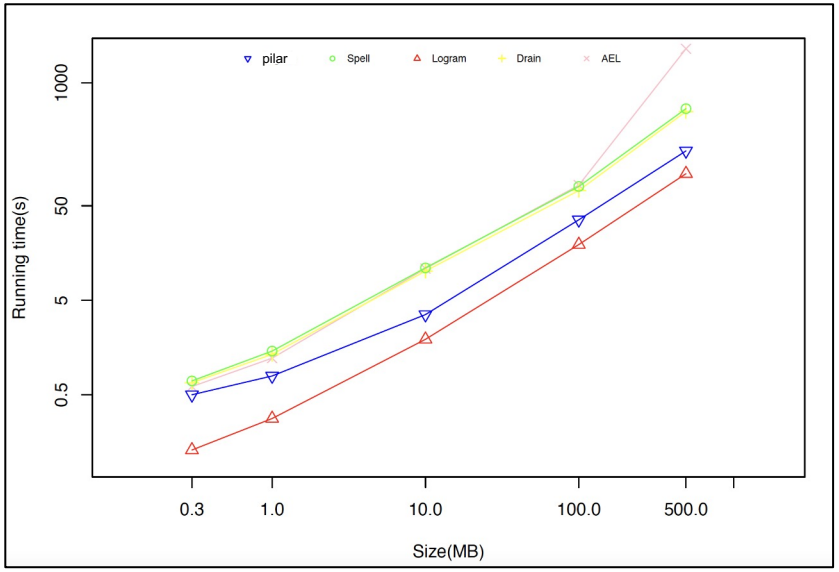


Log templates

We first evaluate the general qualities of PILAR



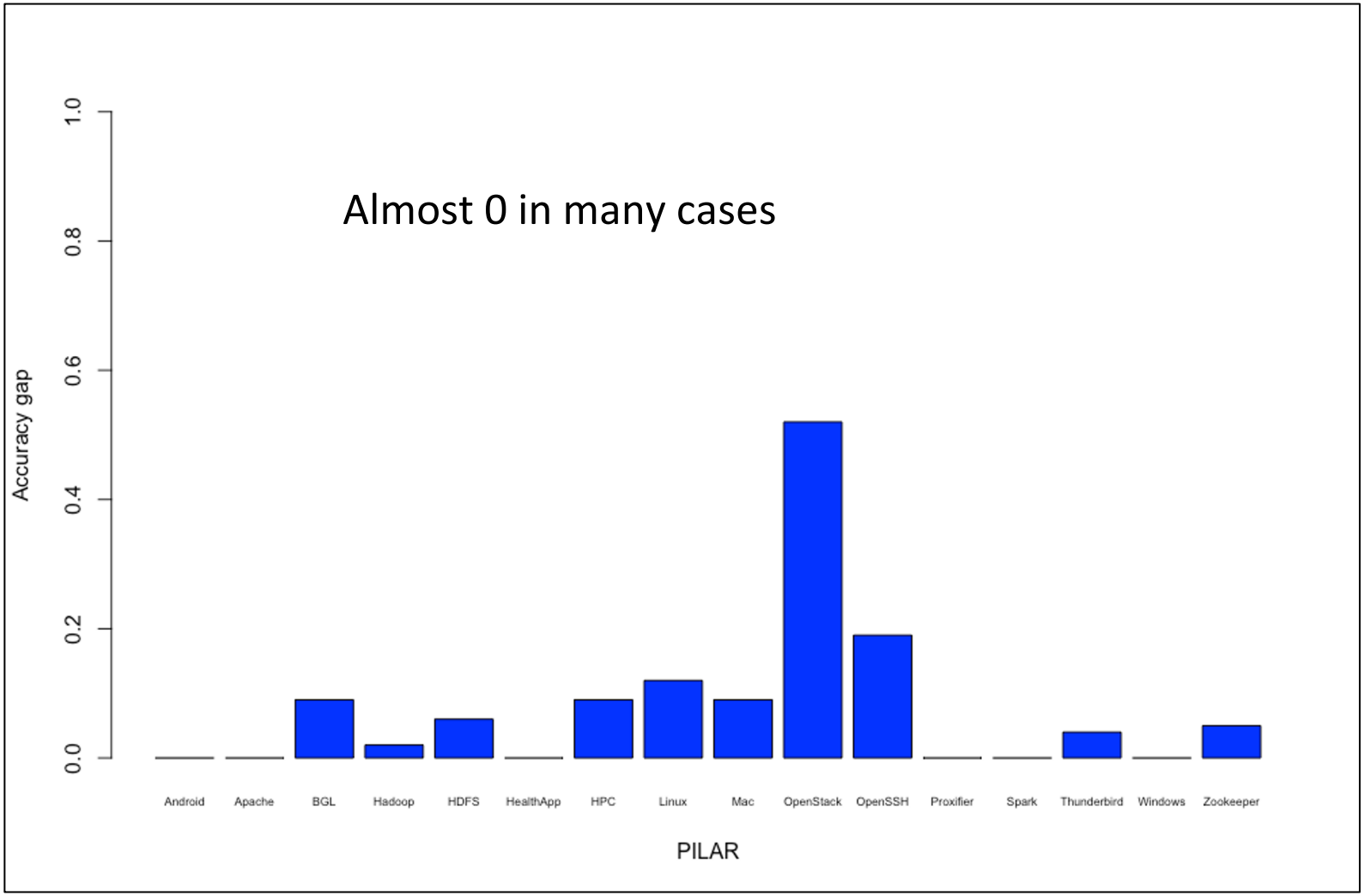
Accuracy



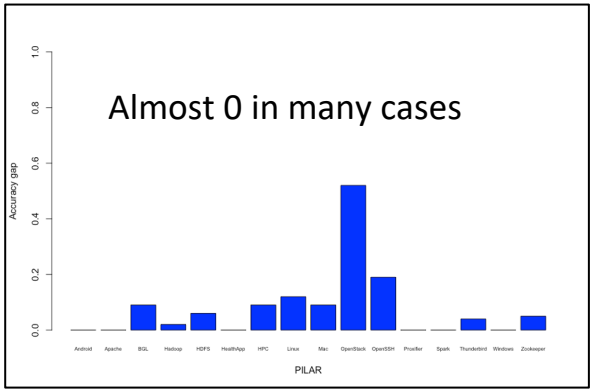
Efficiency

PILAR has the second highest accuracy and efficiency among all log parsers, being only slightly behind the most accurate parser Drain and the most efficient parser Logram.

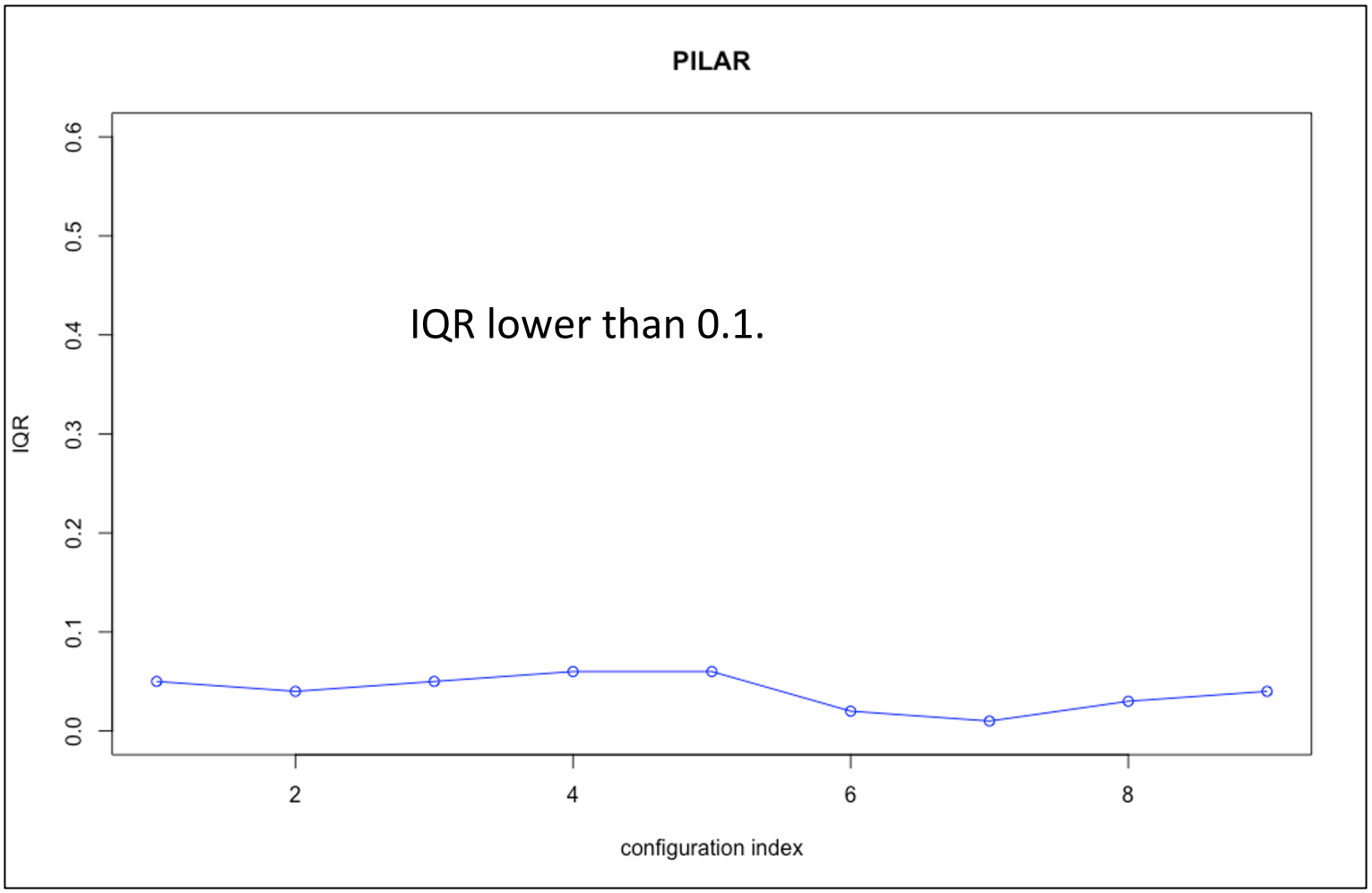
We then evaluate influence of configuration parameter on PILAR


Almost 0 in many cases
Accuracy gap
0.0
0.2
0.4
0.6
0.8
1.0
Android
Apache
BGL
Hadoop
HDFS
HealthApp
HPC
Linux
Mac
OpenStack
OpenSSH
Proxifier
Spark
Thunderbird
Windows
Zookeeper
PILAR

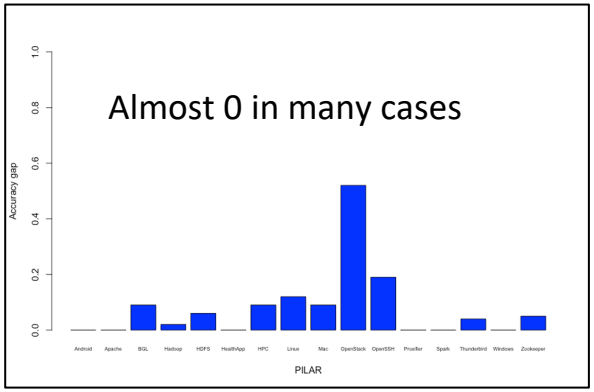
We then evaluate influence of configuration parameter on PILAR



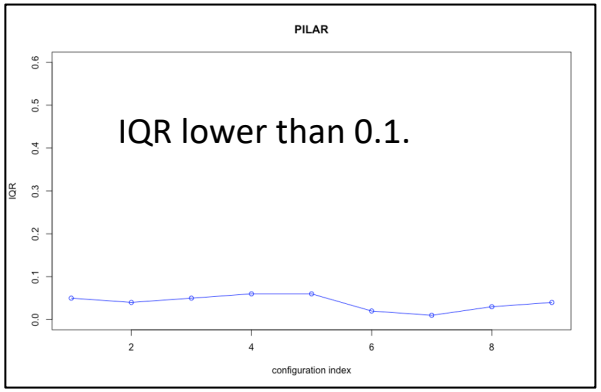
Accuracy gaps with varying parameter on the same datasets


PILAR
IQR lower than 0.1.
IQR
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2
4
6
8
configuration index

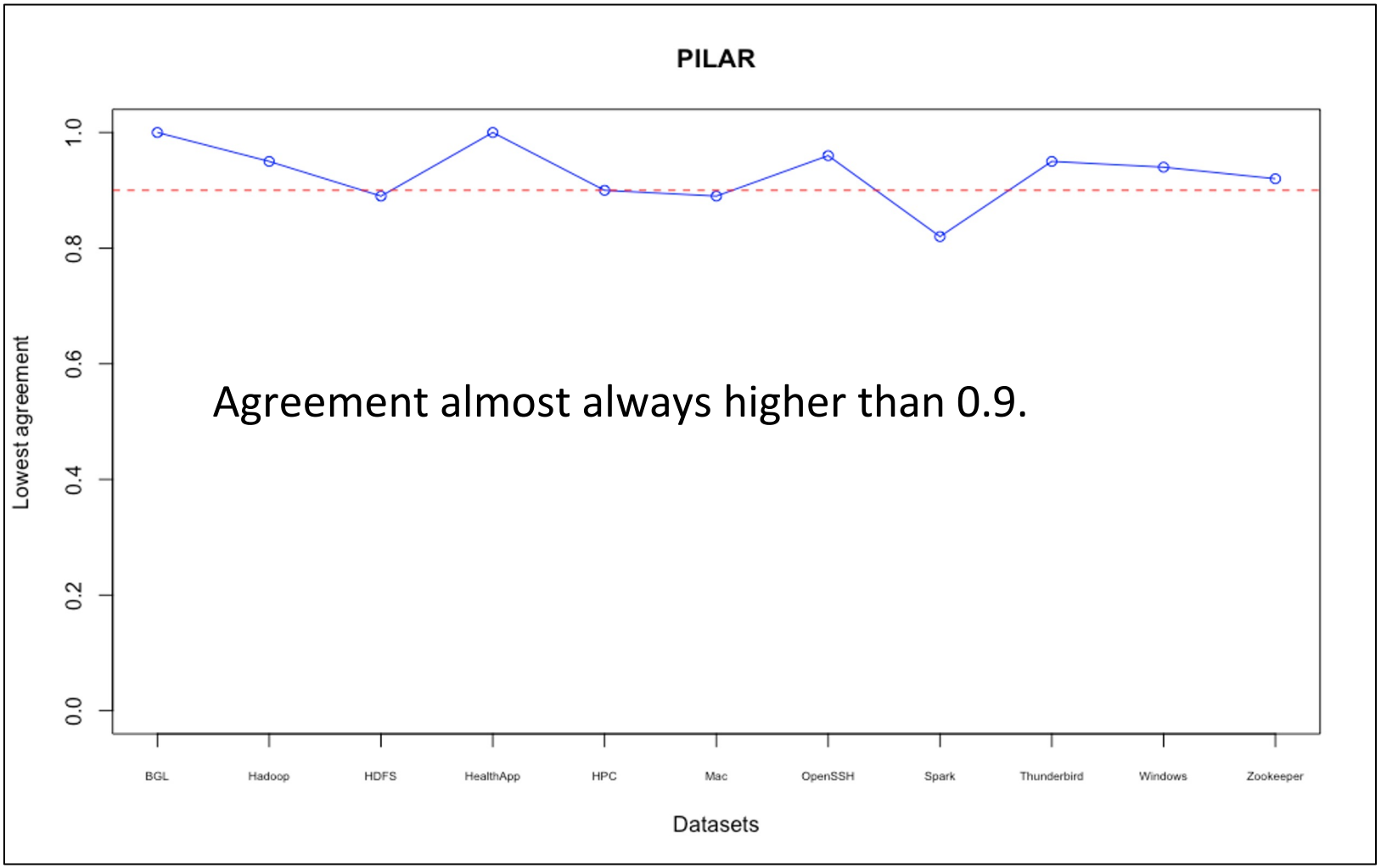
We then evaluate influence of configuration parameter on PILAR



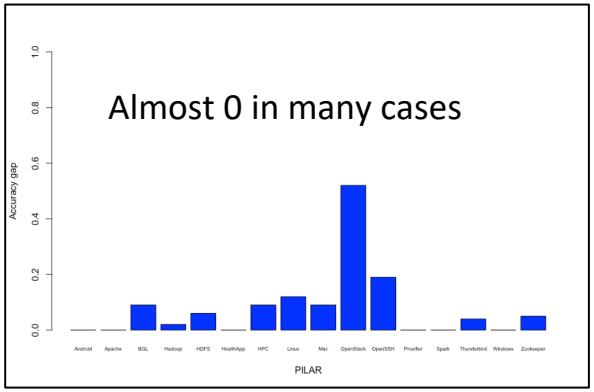
Accuracy gaps with varying parameter on the same datasets



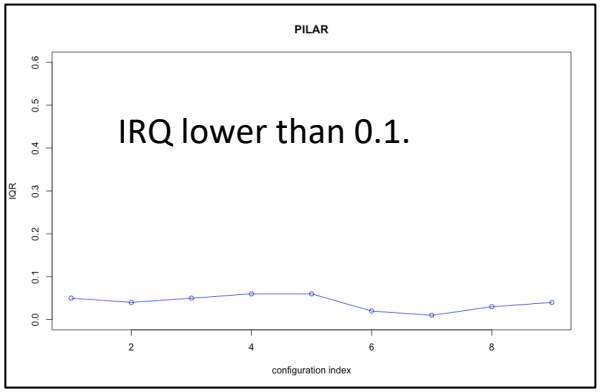
IQR with fixed parameters on different datasets


PILAR
Lowest agreement
1.0
0.8
0.6
0.4
0.2
0.0
Agreement almost always higher than 0.9.
BGL
Hadoop
HDFS
HealthApp
HPC
Mac
OpenSSH
Spark
Thunderbird
Windows
Zookeeper
Datasets

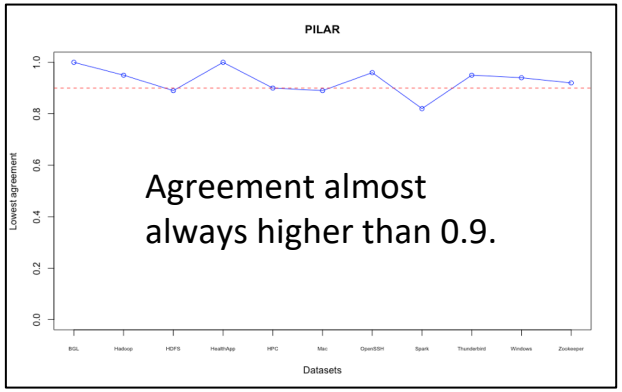
We then evaluate influence of configuration parameter on PILAR



Accuracy gaps with varying parameter on the same datasets

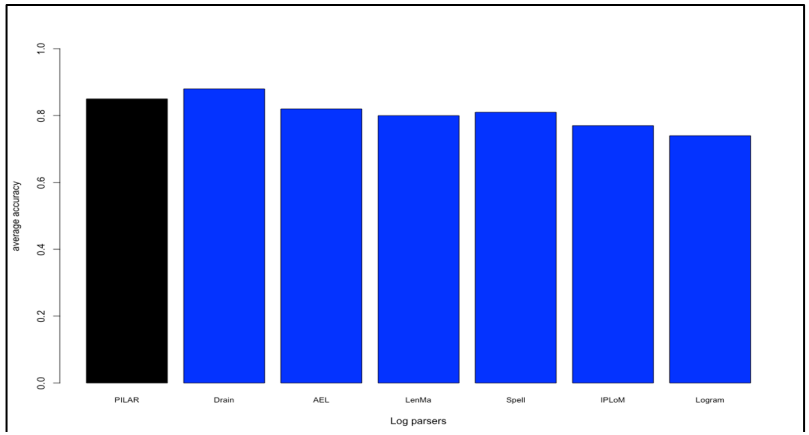


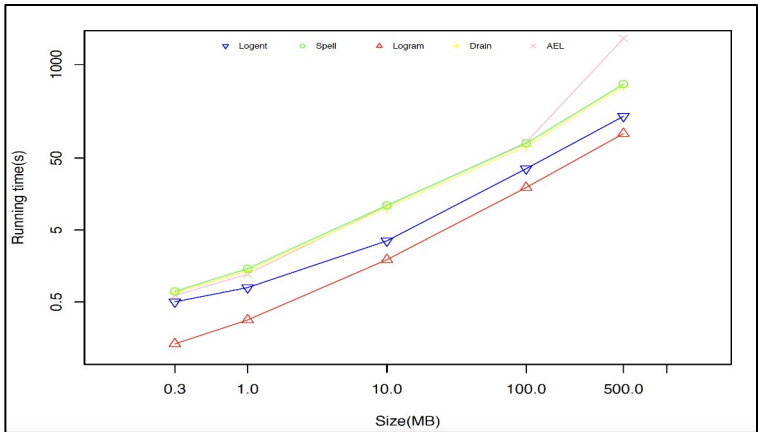
IQR with fixed parameters on different datasets



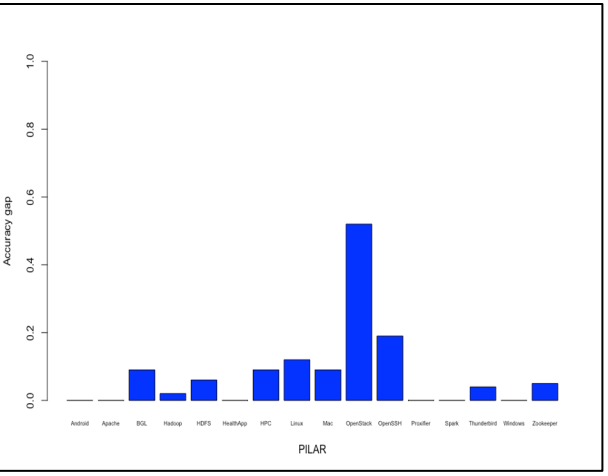
Lowest agreement achieved by fixed parameters on different samples of the same dataset

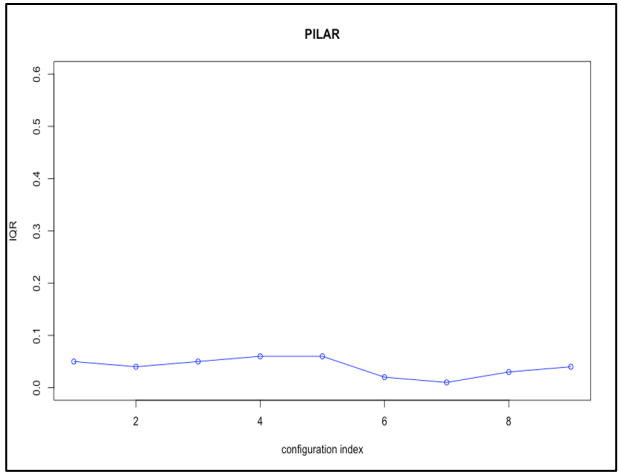
PILAR is parameter-insensitive in all scenarios compared to all other log parsers.

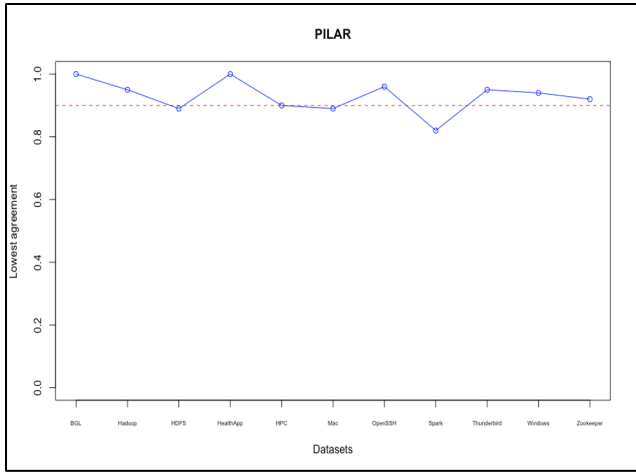




PILAR is not only accurate and efficient but also parameter-insensitive







PILAR: https://github.com/senseconcordia/PILARData_ICSE2023

Research on Intelligent DevOps

Log parsing	Anomaly detection	Failure diagnostics
Monitoring automation	 Software System	Event/Incident prediction
Monitoring practices		Perf. analysis & prediction
Autonomous configuration	MLOps analysis	Security analysis

Monitoring code made right

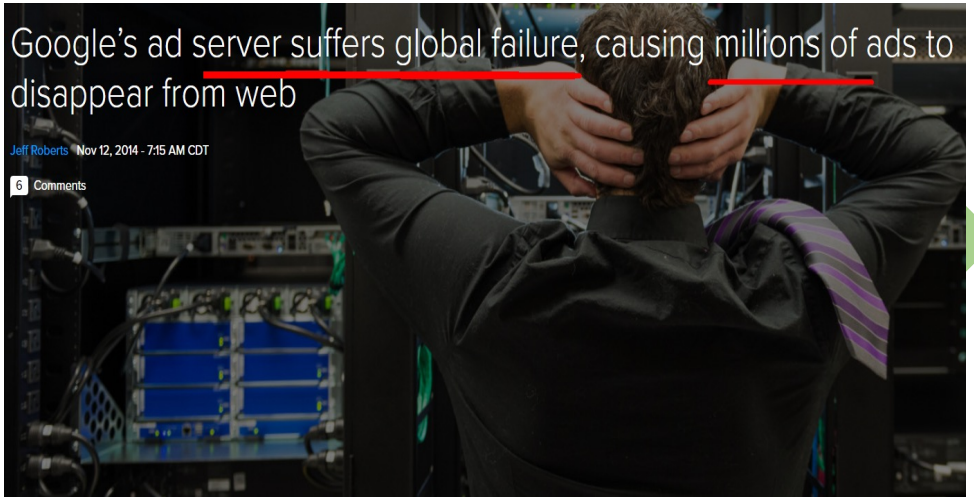
Monitoring data used right

Experience Report: System Log Analysis for Anomaly Detection

Ref: Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. "**Experience report: System log analysis for anomaly detection**." In 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 207-218. IEEE, 2016.

Background

- System breakdown causes significant revenue loss.



Delta Says Computer Breakdown Cut Revenue by $100 Million

by **Michael Sasso**
September 2, 2016 — 9:05 AM EDT *Updated on* September 2, 2016 — 9:17 AM EDT

Delta Air Lines Inc. said the computer failure that caused 2,300 flight cancellations last month cut sales about $100 million and reduced a key revenue figure.

Passenger revenue for each seat flown a mile, an industry benchmark, fell 9.5 percent in August, in part because of the outage and subsequent recovery efforts, the carrier said in a statement Friday. The breakdown reduced unit revenue, as the measure is also known, by two percentage points, Delta said.

The country's second-largest airline earlier forecast that third-quarter unit revenue would fall 4 percent to 6 percent.

A power-control module at Delta's Atlanta computer center failed and caught fire Aug. 8, shutting down electricity to the system. About 300 of the airline's 7,000 servers weren't wired to backup power, the company had said.

The shares fell 1 percent to $36.45 at 9:15 a.m. in premarket trading. The stock tumbled 27 percent this year through Thursday.

- Anomaly detection could pinpoint issues promptly and help resolve them immediately.

Background

Logs :

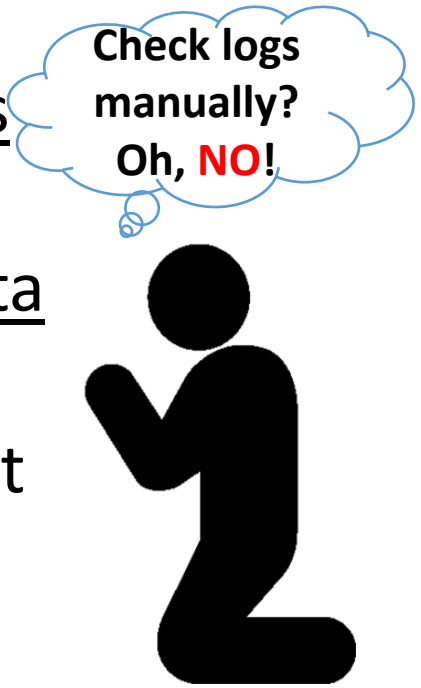
- Logs are the **main data source** for system anomaly detection.
- Logs are **routinely** generated by systems (e.g., 24 x 7 basis).
- Logs record detailed **runtime information**, e.g., timestamp, state, IP address.

```
1 | 2008-11-09 20:55:54 PacketResponder 0 for block blk_321 terminating
2 | 2008-11-09 20:55:54 Received block blk_321 of size 67108864 from /10.251.195.70
3 | 2008-11-09 20:55:54 PacketResponder 2 for block blk_321 terminating
4 | 2008-11-09 20:55:54 Received block blk_321 of size 67108864 from /10.251.126.5
5 | 2008-11-09 21:56:50 10.251.126.5:50010:Got exception while serving  blk_321 to /10.251.127.243
6 | 2008-11-10 03:58:04 Verification succeeded for blk_321
7 | 2008-11-10 10:36:37 Deleting block blk_321 file /mnt/ hadoop/dfs/data/current/subdir1/blk_321
8 | 2008-11-10 10:36:50 Deleting block blk_321 file /mnt/ hadoop/dfs/data/current/subdir51/blk_321
```

Background

Manual inspection of logs becomes **impossible**!

- Systems are often implemented by hundreds of developers.
- Logs are generated at a high rate & Noisy data are hard to distinguish.
- Systems generate duplicated logs due to fault tolerant mechanism.



Many automated log-based anomaly detection methods are proposed!

Background

Log-based anomaly detection methods:

- Failure diagnosis using decision trees [**ICAC'04**]
- Failure prediction in IBM bluegene/l event logs [**ICDM'07**]
- Detecting largescale system problems by mining console logs [**SOSP'09**]
- Mining invariants from console logs for system problem detection. [*USENIX* **ATC'10**]
- Log Clustering based Problem Identification for Online Service Systems [**ICSE'16**]

...

Framework

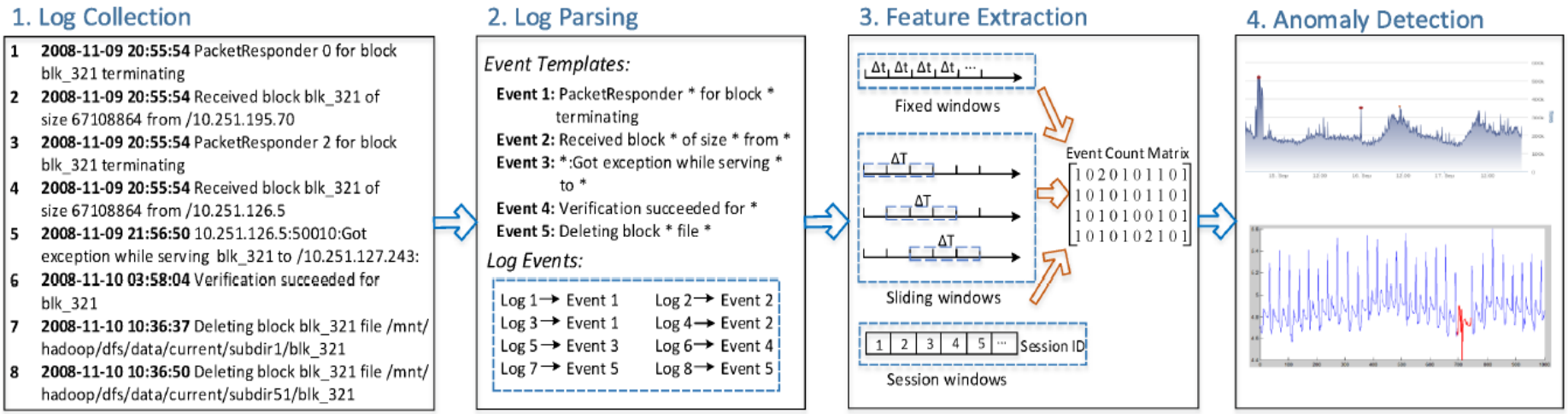
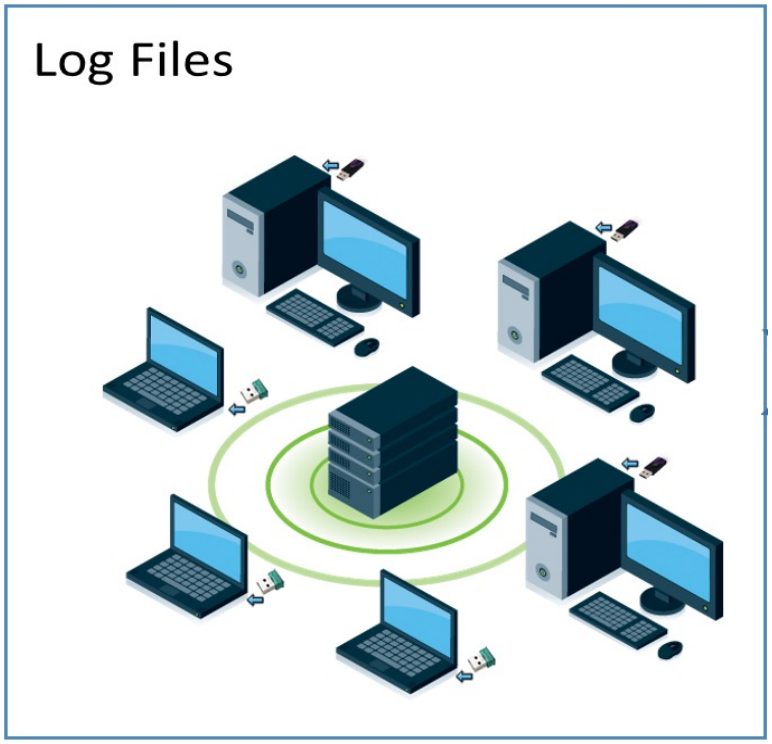


Figure 1: Framework of anomaly detection

1. Log Collection



1 **2008-11-09 20:55:54** PacketResponder 0 for block blk_321 terminating
2 **2008-11-09 20:55:54** Received block blk_321 of size 67108864 from /10.251.195.70
3 **2008-11-09 20:55:54** PacketResponder 2 for block blk_321 terminating
4 **2008-11-09 20:55:54** Received block blk_321 of size 67108864 from /10.251.126.5
5 **2008-11-09 21:56:50** 10.251.126.5:50010:Got exception while serving blk_321 to /10.251.127.243:
6 **2008-11-10 03:58:04** Verification succeeded for blk_321
7 **2008-11-10 10:36:37** Deleting block blk_321 file /mnt/hadoop/dfs/data/current/subdir1/blk_321
8 **2008-11-10 10:36:50** Deleting block blk_321 file /mnt/hadoop/dfs/data/current/subdir51/blk_321

2. Log Parsing

1 **2008-11-09 20:55:54** PacketResponder 0 for block blk_321 terminating
2 **2008-11-09 20:55:54** Received block blk_321 of size 67108864 from /10.251.195.70
3 **2008-11-09 20:55:54** PacketResponder 2 for block blk_321 terminating
4 **2008-11-09 20:55:54** Received block blk_321 of size 67108864 from /10.251.126.5
5 **2008-11-09 21:56:50** 10.251.126.5:50010:Got exception while serving blk_321 to /10.251.127.243:
6 **2008-11-10 03:58:04** Verification succeeded for blk_321
7 **2008-11-10 10:36:37** Deleting block blk_321 file /mnt/hadoop/dfs/data/current/subdir1/blk_321
8 **2008-11-10 10:36:50** Deleting block blk_321 file /mnt/hadoop/dfs/data/current/subdir51/blk_321

Event Templates:

Event 1: PacketResponder * for block * terminating
Event 2: Received block * of size * from *
Event 3: *:Got exception while serving * to *
Event 4: Verification succeeded for *
Event 5: Deleting block * file *

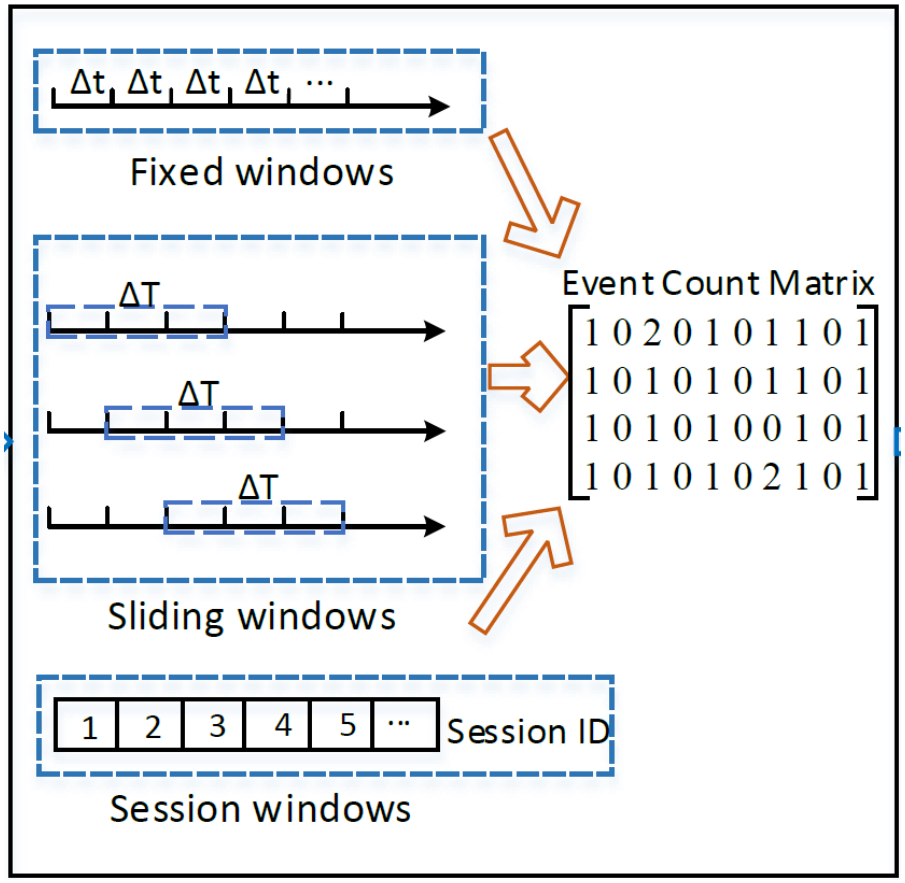
Log Events:

Log 1 → Event 1	Log 2 → Event 2
Log 3 → Event 1	Log 4 → Event 2
Log 5 → Event 3	Log 6 → Event 4
Log 7 → Event 5	Log 8 → Event 5

1 **2008-11-09 20:55:54** PacketResponder 0 for block blk_321 terminating
3 **2008-11-09 20:55:54** PacketResponder 2 for block blk_321 terminating

Event 1: PacketResponder * for block * terminating

3. Feature Extraction

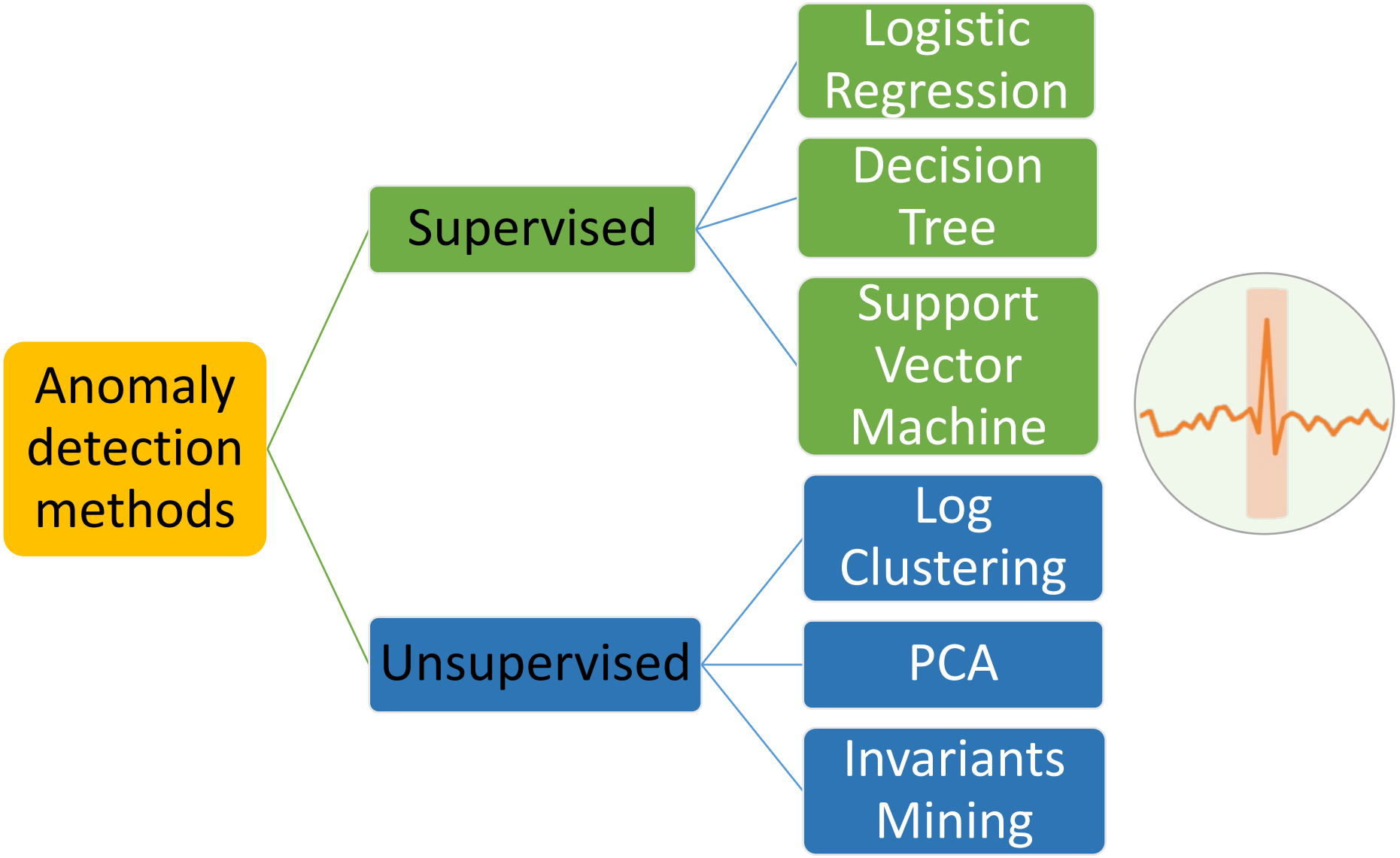


Divide all logs into different log sequences (windows)

log sequence <=> row in the event count matrix.

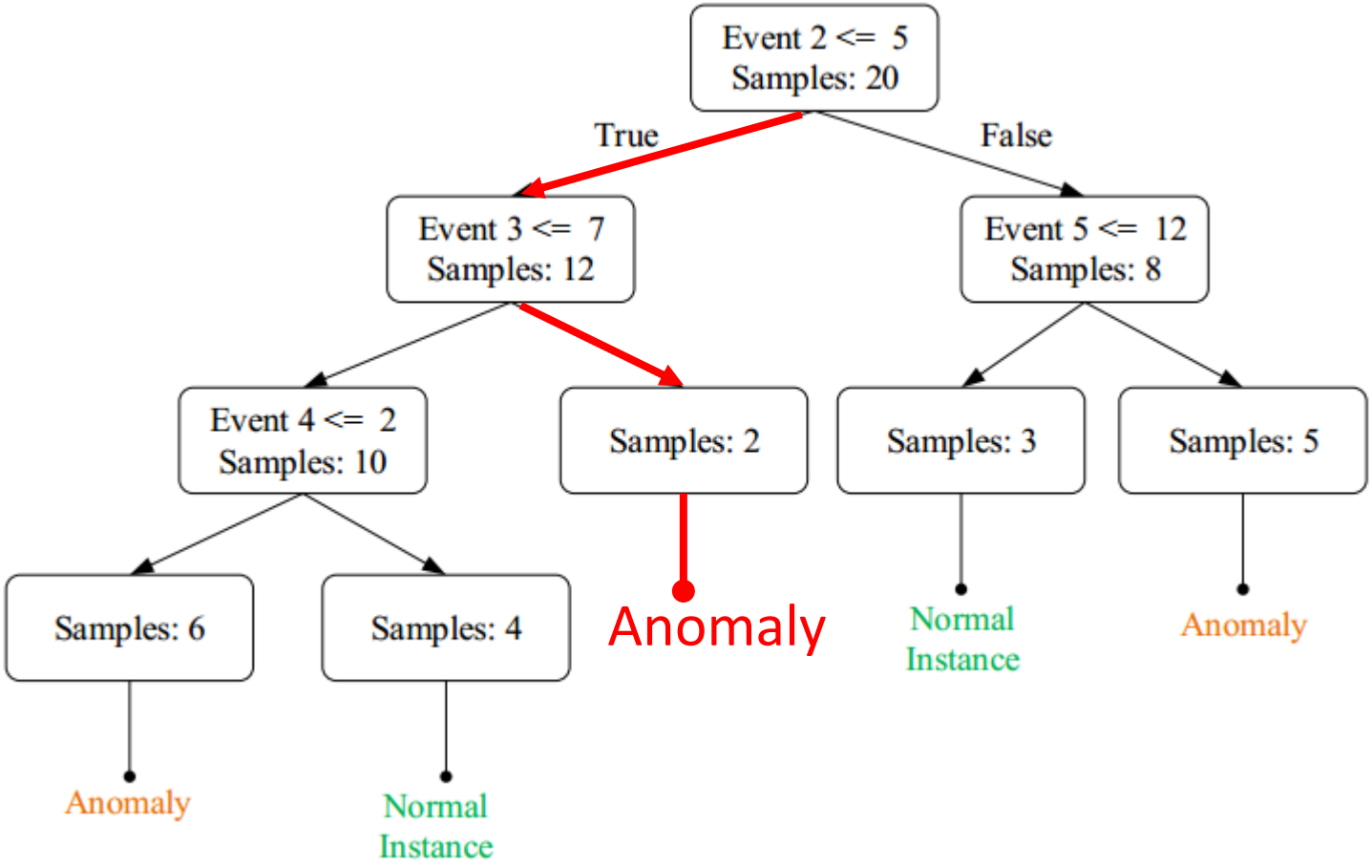
Windows	Basis
Fixed windows	Time
Sliding windows	Time
Session windows	Identifiers

4. Anomaly Detection



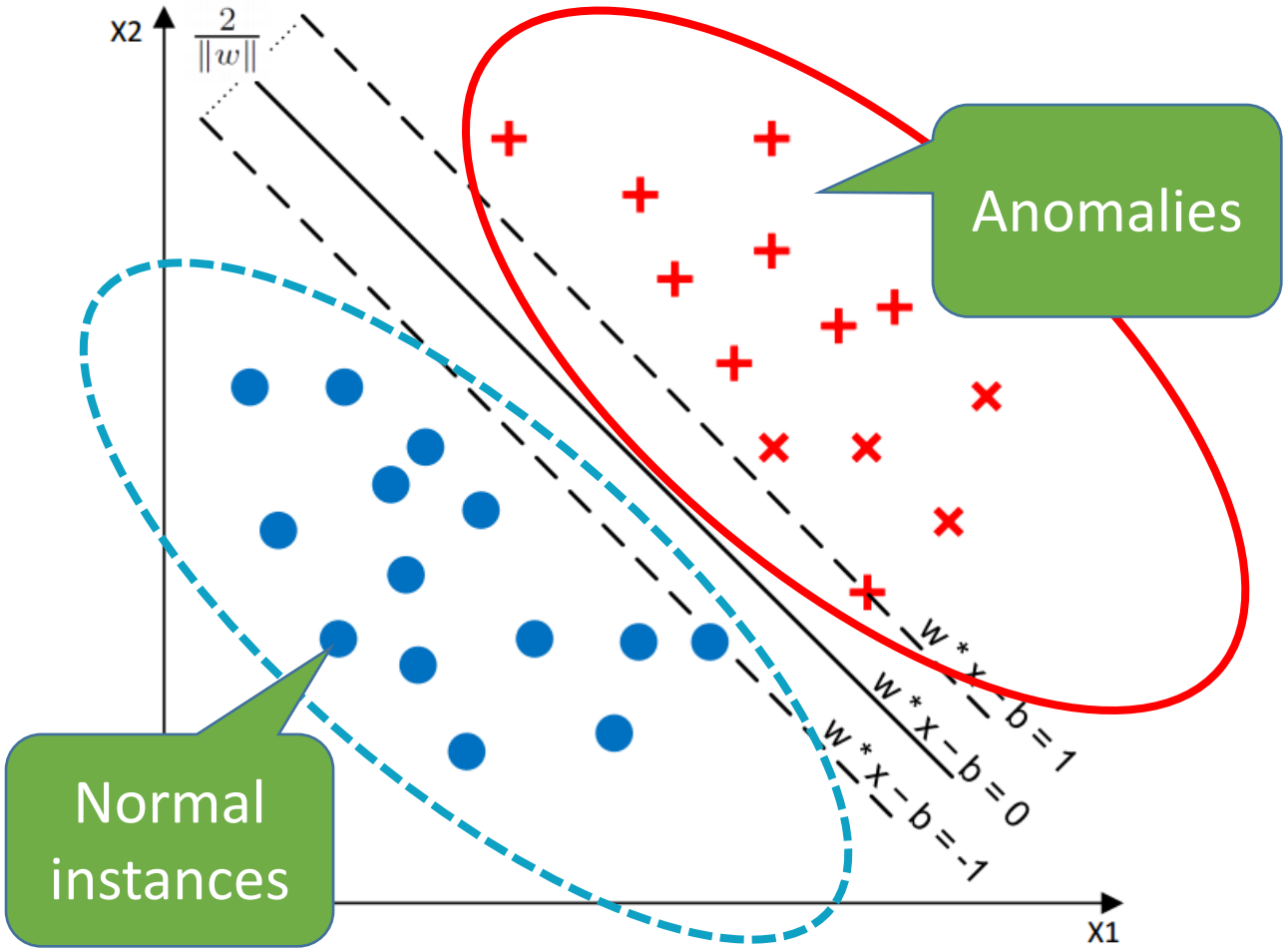
Decision Tree (Supervised)

Trained Decision Tree Example:

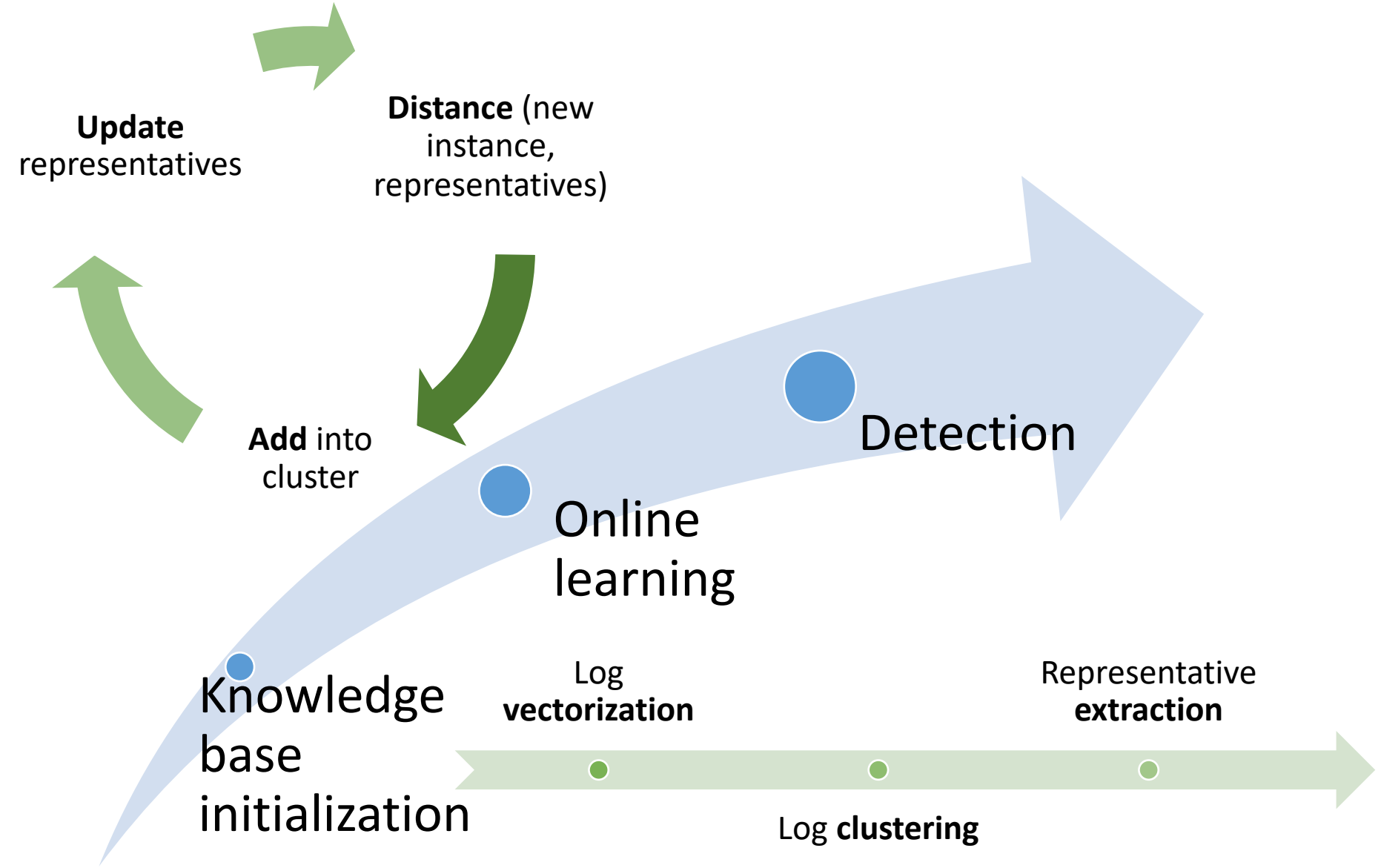


Support Vector Machine (Supervised)

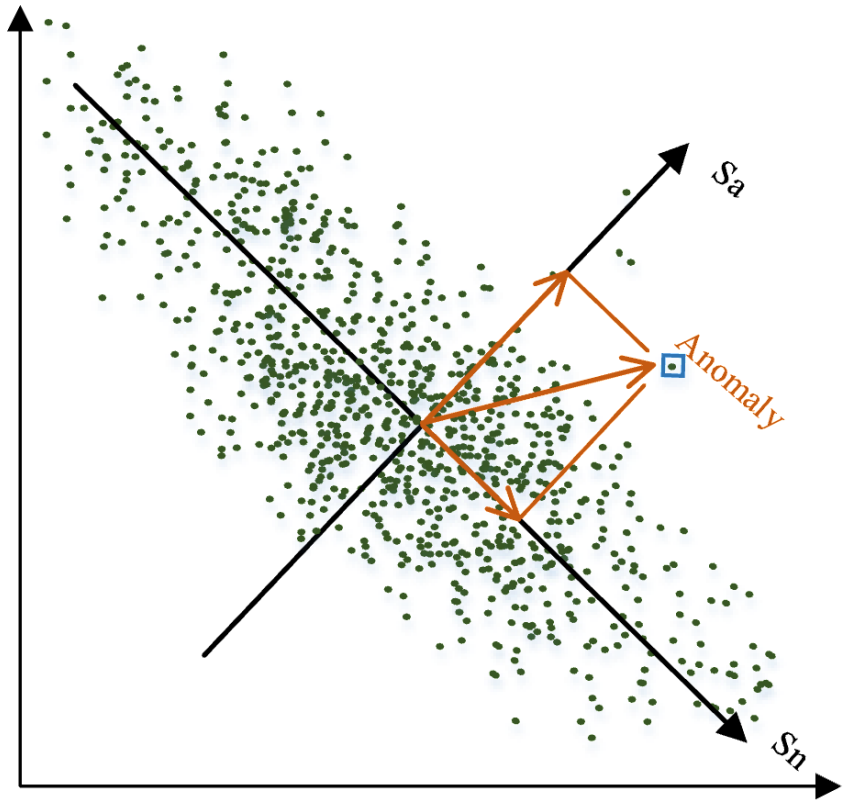
Trained SVM Example:



Log Clustering (Unsupervised)



PCA (Unsupervised)



Two subspaces are generated by PCA:

1. **S_n: Normal Space**, constructed by first *k* principal components.
2. **S_a: Anomaly Space**, constructed by remaining *(n-k)* components.

Evaluation

Data sets

Table I: Summary of datasets

System	#Time span	#Data size	#Log messages	#Anomalies
BGL	7 months	708 M	4,747,963	348,460
HDFS	38.7 hours	1.55 G	11,175,629	16,838

Fixed windows & Sliding windows

Session windows

Performance metric

$$Precision = \frac{\#Anomalies\ detected}{\#Anomalies\ reported}$$

$$Recall = \frac{\#Anomalies\ detected}{\#All\ anomalies}$$

$$F\_measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Evaluation

Q1: What is the accuracy of supervised anomaly detection?

Q2: What is the accuracy of unsupervised anomaly detection?

Q3: What is the efficiency of these anomaly detection?

Evaluation

1. Accuracy of Supervised Methods

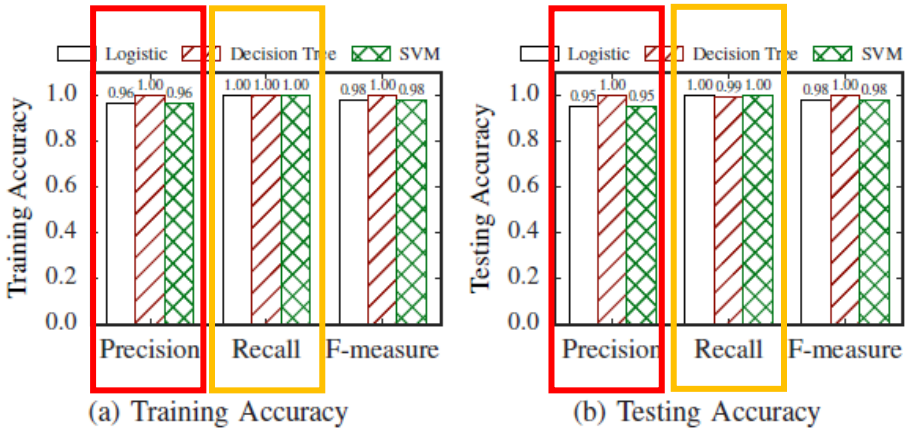


Figure 5: Accuracy of supervised methods on HDFS data with session windows

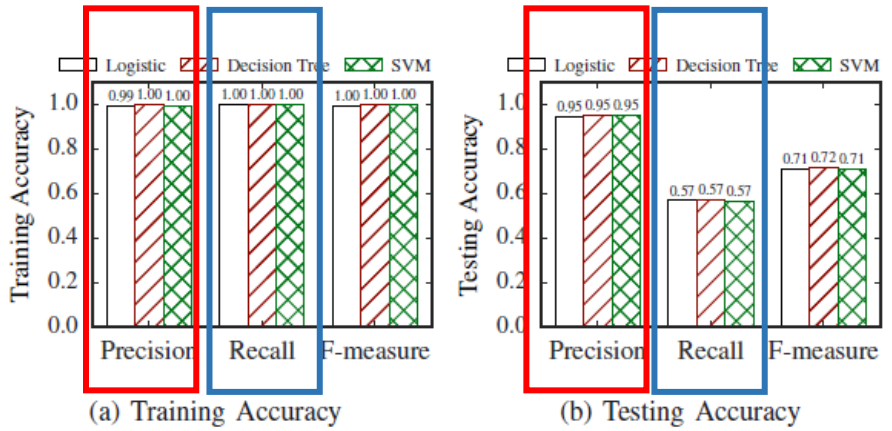


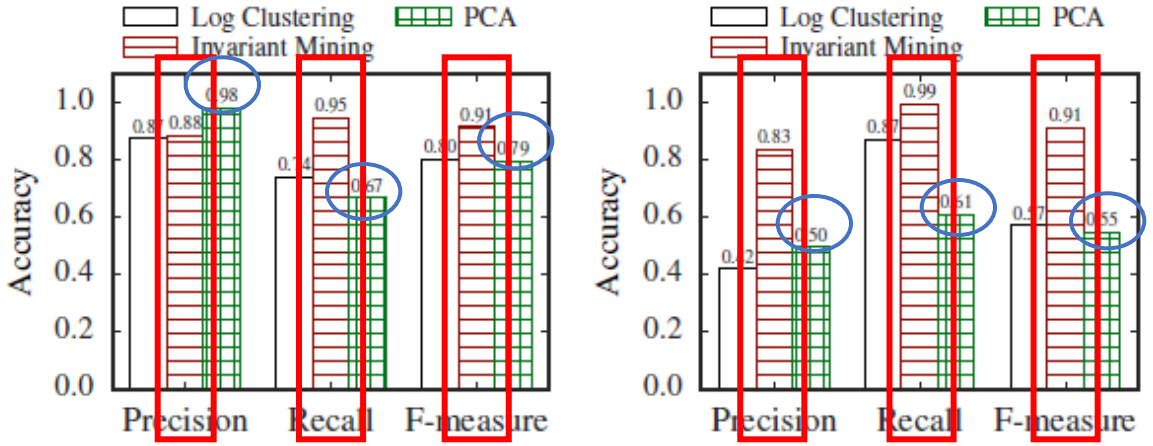
Figure 6: Accuracy of supervised methods on BGL data with fixed windows

More sensitive

Finding 1: Supervised anomaly detection achieves **high precision**, while **recall varies**.

Evaluation

2. Accuracy of Unsupervised Methods



(a) Accuracy on HDFS data (b) Accuracy on BGL data

Figure 9: Accuracy of unsupervised methods on HDFS data and BGL data

Finding 3: Unsupervised methods are **not as good as** supervised methods except Invariants Mining

Evaluation

4. Efficiency of Anomaly Detection Methods

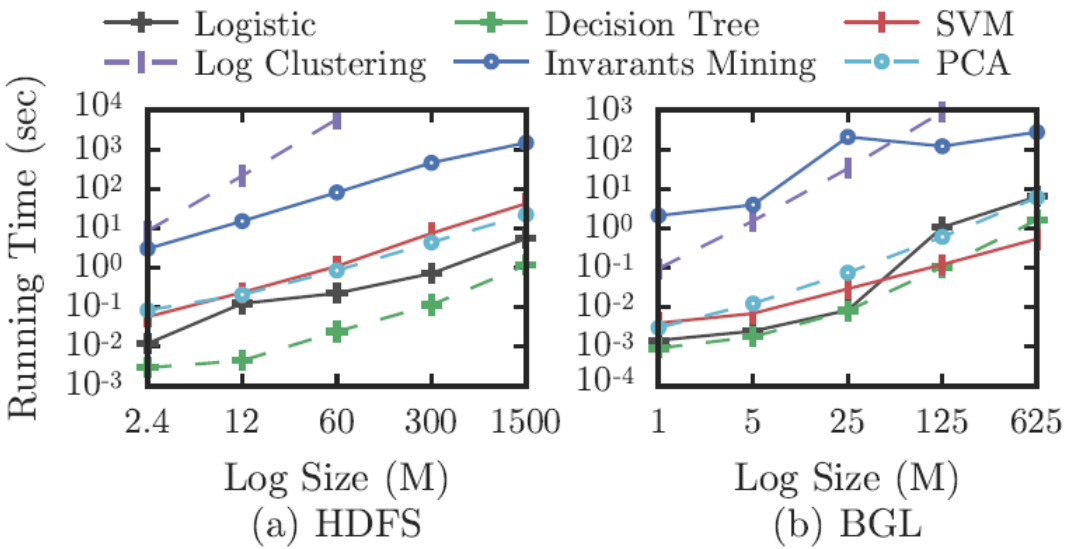
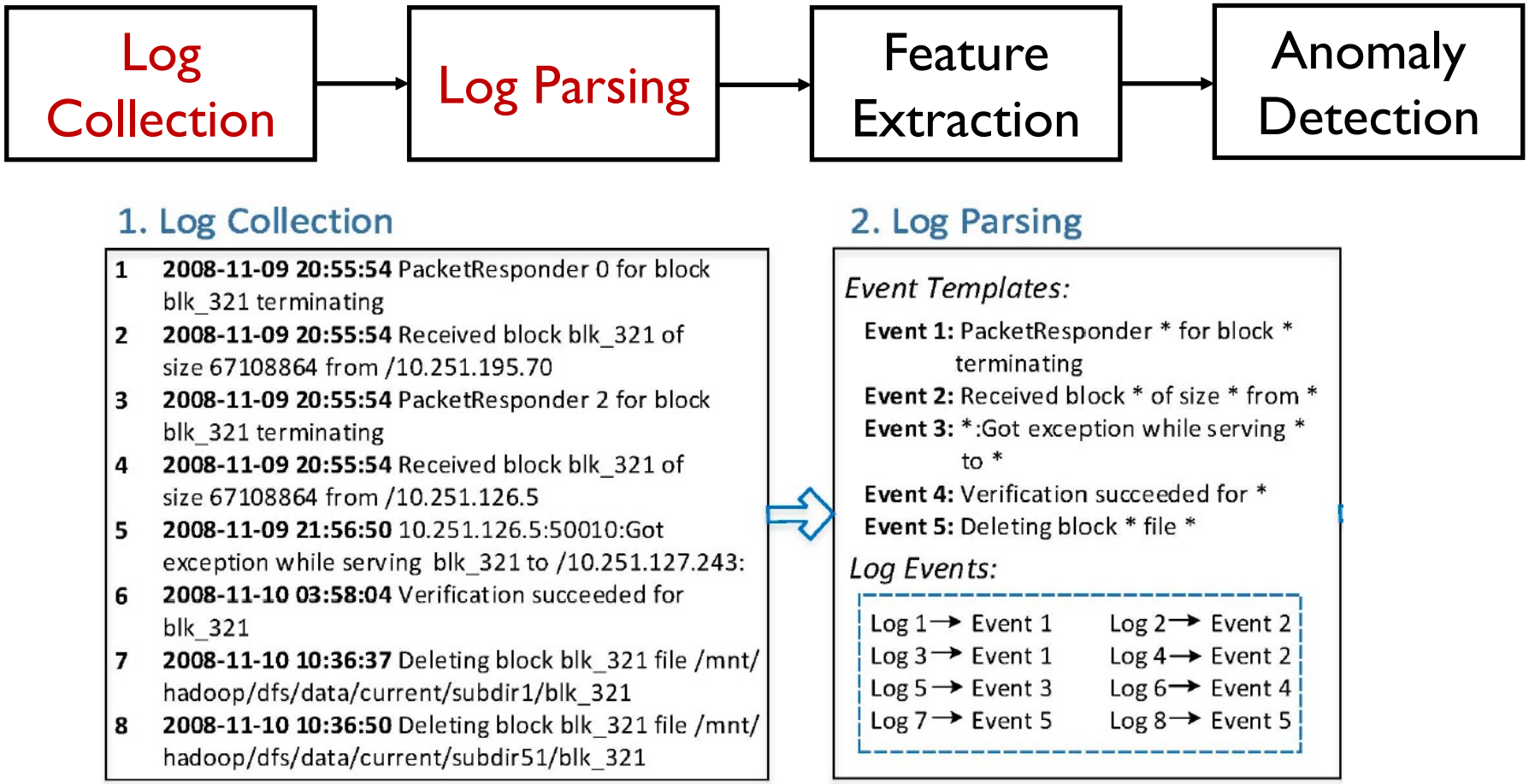


Figure 12: Running time with increasing log size

Finding 5: Most anomaly detection scale linearly with log size except Log Clustering and Invariants Mining.

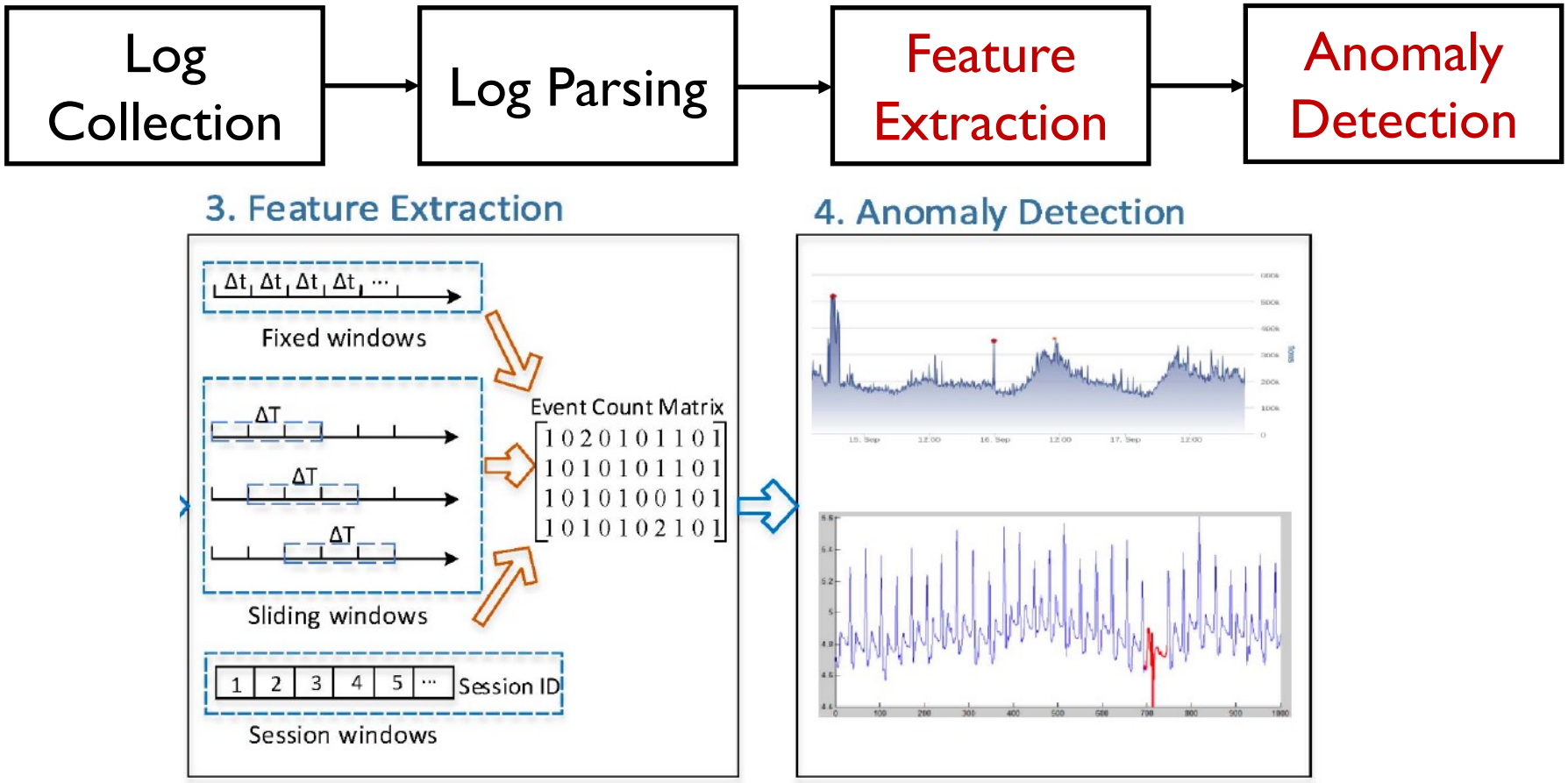
Demo

Demo: Analyzing System Logs for Anomaly Detection



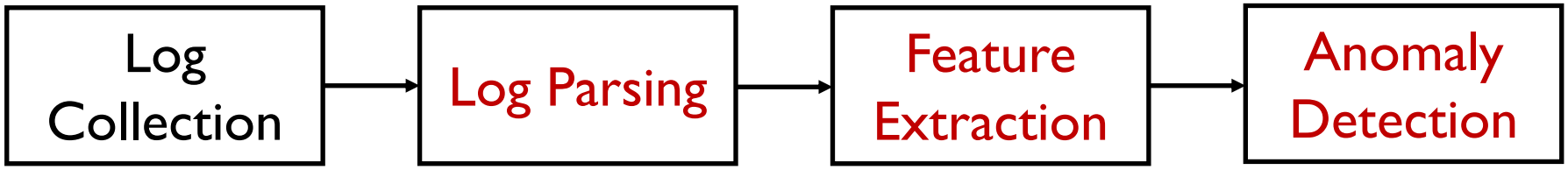
He et al., *Experience Report: System Log Analysis for Anomaly Detection*. ISSRE '16.

Demo: Analyzing System Logs for Anomaly Detection



He et al., *Experience Report: System Log Analysis for Anomaly Detection*. ISSRE '16.

Demo: Analyzing System Logs for Anomaly Detection



Google Colab Notebook:

https://colab.research.google.com/drive/1ClOZtGyk14D7a815IBgSgE-XeThwcAyj?usp=sharing

He et al., *Experience Report: System Log Analysis for Anomaly Detection*. ISSRE '16.

Group Assignments