No More Labelled Examples? An Unsupervised Log Parser with LLMs

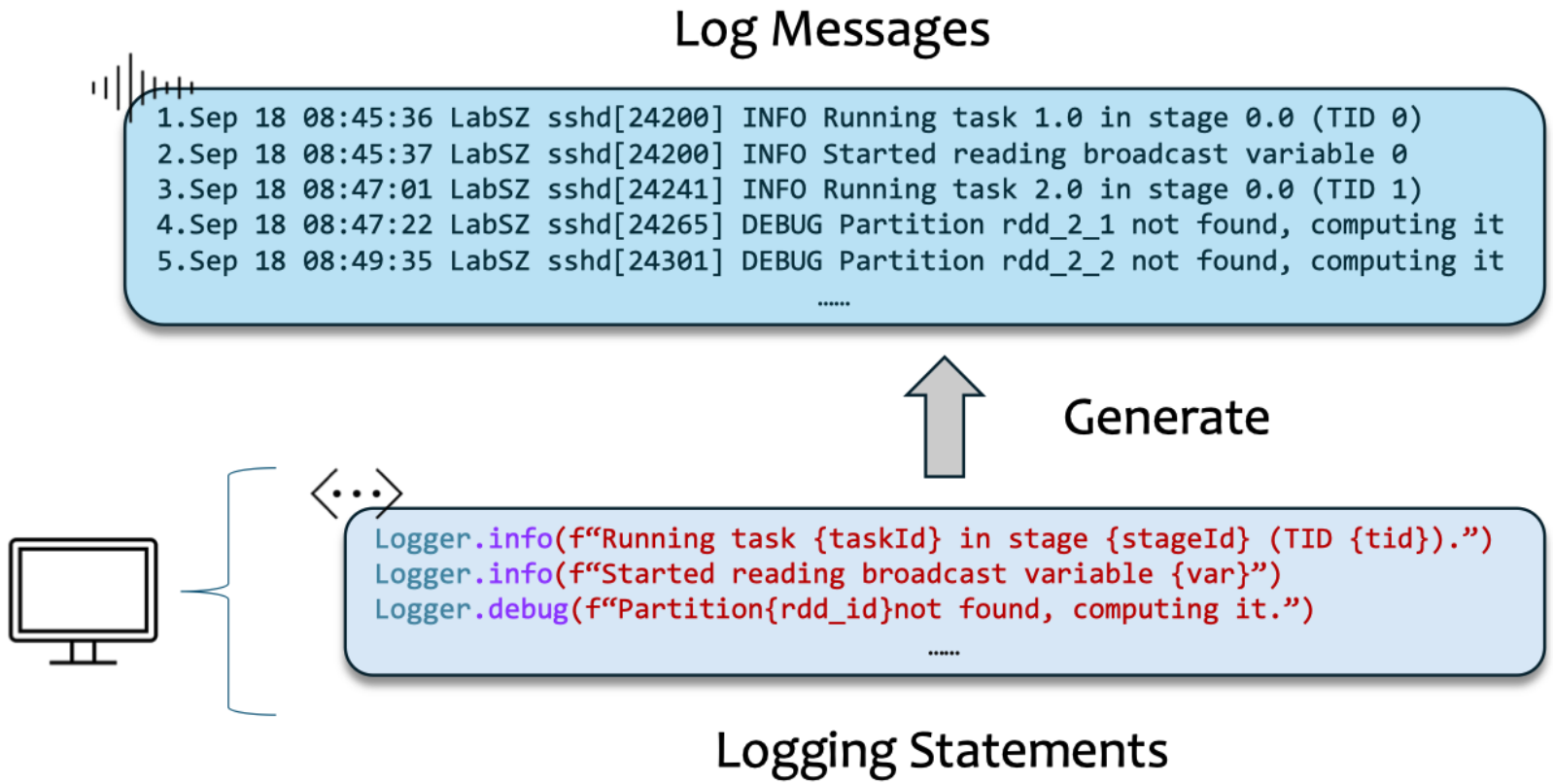
Junjie Huang[1], Zhihan Jiang[1], Zhuangbin Chen[2], Michael R. Lyu[1]

[1]The Chinese University of Hong Kong, [2]Sun Yat-sen University

Read the paper!

System logs

Logs are generated by logging statements and record runtime information of a system



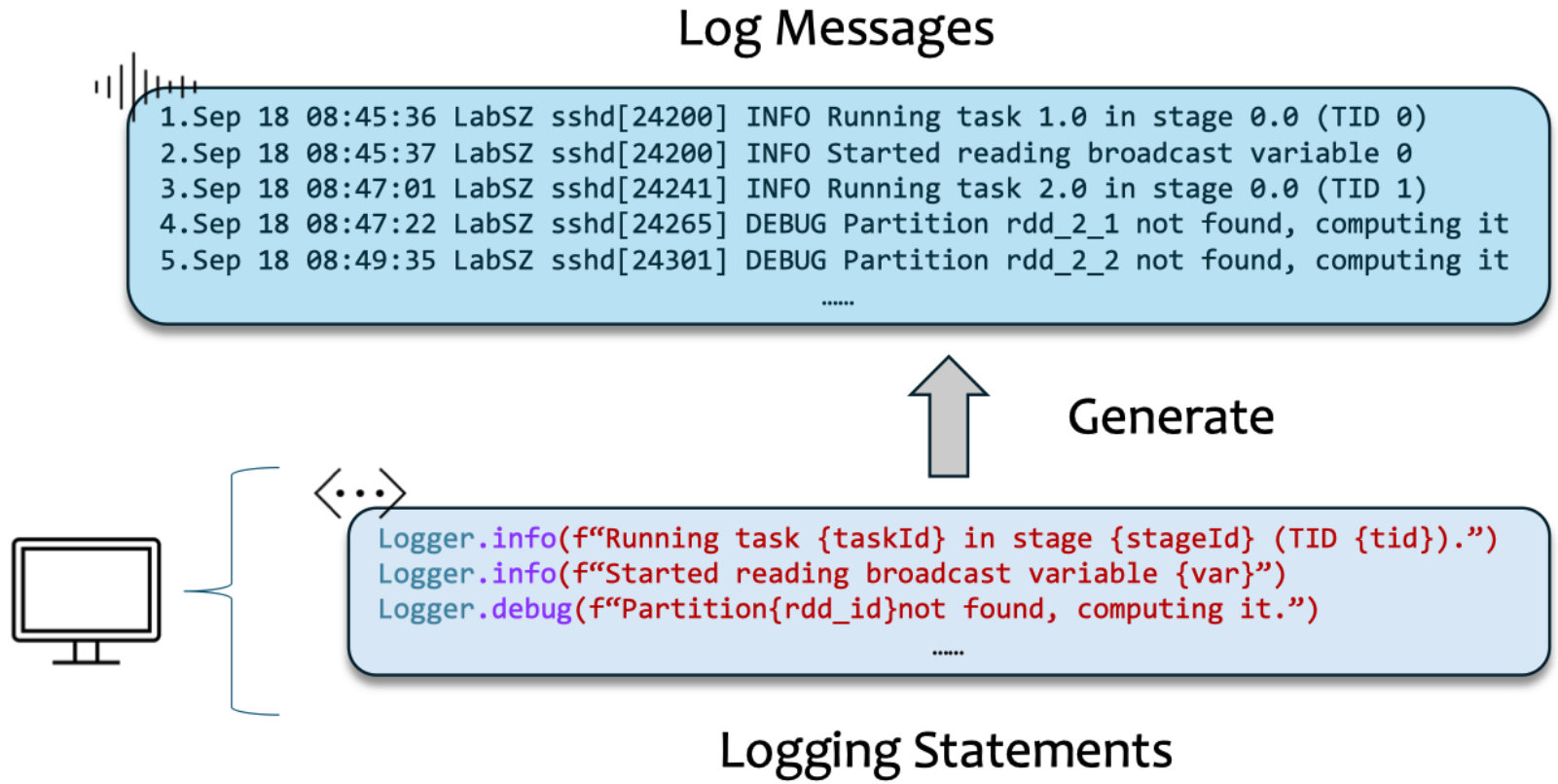
System logs

Automatic Log Analysis is important for:

Anomaly Detection

Failure Diagnosis

Root Cause Analysis



Log parsing

Raw logs are unstructured, which are not suitable for *understanding*, *analyzing* and *storing*

Transform raw log messages into structured format

Raw Logs

```
1.Sep 18 08:45:36 LabSZ sshd[24200] INFO Running task 1.0 in stage 0.0 (TID 0)
2.Sep 18 08:45:37 LabSZ sshd[24200] INFO Started reading broadcast variable 0
3.Sep 18 08:47:01 LabSZ sshd[24241] INFO Running task 2.0 in stage 0.0 (TID 1)
4.Sep 18 08:47:22 LabSZ sshd[24265] DEBUG Partition rdd_2_1 not found, computing it
5.Sep 18 08:49:35 LabSZ sshd[24301] DEBUG Partition rdd_2_2 not found, computing it
.....
```

How to understand and process these complex logs?

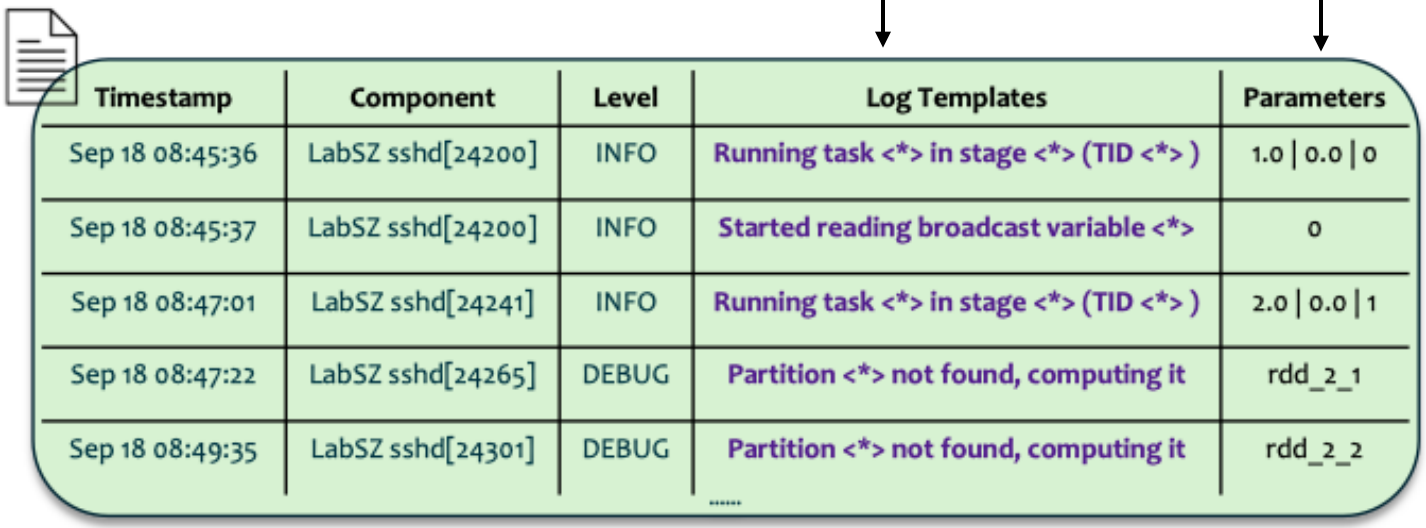
Structured Logs

Timestamp	Component	Level	Log Templates	Parameters
Sep 18 08:45:36	LabSZ sshd[24200]	INFO	Running task <*> in stage <*> (TID <*>)	1.0 | 0.0 | 0
Sep 18 08:45:37	LabSZ sshd[24200]	INFO	Started reading broadcast variable <*>	0
Sep 18 08:47:01	LabSZ sshd[24241]	INFO	Running task <*> in stage <*> (TID <*>)	2.0 | 0.0 | 1
Sep 18 08:47:22	LabSZ sshd[24265]	DEBUG	Partition <*> not found, computing it	rdd_2_1
Sep 18 08:49:35	LabSZ sshd[24301]	DEBUG	Partition <*> not found, computing it	rdd_2_2
.....				

Two same events happened on **two partitions** (*rdd_2_1*, *rdd_2_2*)

Log parsing

Goal: to distinguish between constant part and dynamic part.



Timestamp	Component	Level	Log Templates	Parameters
Sep 18 08:45:36	LabSZ sshd[24200]	INFO	Running task <*> in stage <*> (TID <*>)	1.0 | 0.0 | 0
Sep 18 08:45:37	LabSZ sshd[24200]	INFO	Started reading broadcast variable <*>	0
Sep 18 08:47:01	LabSZ sshd[24241]	INFO	Running task <*> in stage <*> (TID <*>)	2.0 | 0.0 | 1
Sep 18 08:47:22	LabSZ sshd[24265]	DEBUG	Partition <*> not found, computing it	rdd_2_1
Sep 18 08:49:35	LabSZ sshd[24301]	DEBUG	Partition <*> not found, computing it	rdd_2_2
.....				

Describing the main content of the logged event

The parameters associated with the event.

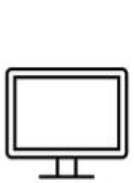
Log parsing

Why not map raw logs with their logging statements?

Raw Logs

```
1.Sep 18 08:45:36 LabSZ sshd[24200] INFO Running task 1.0 in stage 0.0 (TID 0)
2.Sep 18 08:45:37 LabSZ sshd[24200] INFO Started reading broadcast variable 0
3.Sep 18 08:47:01 LabSZ sshd[24241] INFO Running task 2.0 in stage 0.0 (TID 1)
4.Sep 18 08:47:22 LabSZ sshd[24265] DEBUG Partition rdd_2_1 not found, computing it
5.Sep 18 08:49:35 LabSZ sshd[24301] DEBUG Partition rdd_2_2 not found, computing it
……
```

Logging Statements



```
Logger.info(f"Running task {taskId} in stage {stageId} (TID {tid}).")
Logger.info(f"Started reading broadcast variable {var}")
Logger.debug(f"Partition{rdd_id}not found, computing it.")
……
```

Source code is not always accessible in practice!

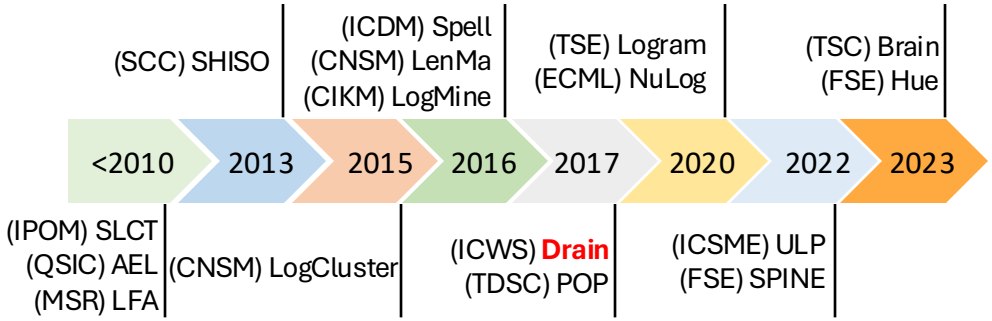
We need data-driven methods!

[SOSP'09] Large-scale system problem detection by mining console logs.

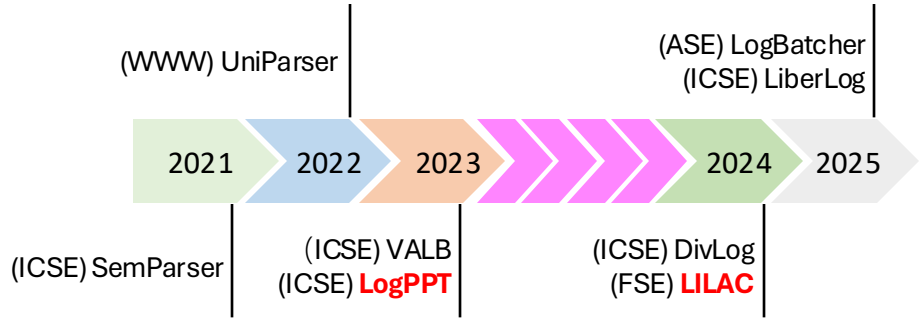
Existent log parsers

A wide range of data-driven parsers have been proposed

Syntax-based log parsers



Semantics-based log parsers



[ICSE'22] Guidelines for assessing the accuracy of log message template identification techniques
[ISSTA'24] A Large-scale Evaluation for Log Parsing Techniques: How Far are We?

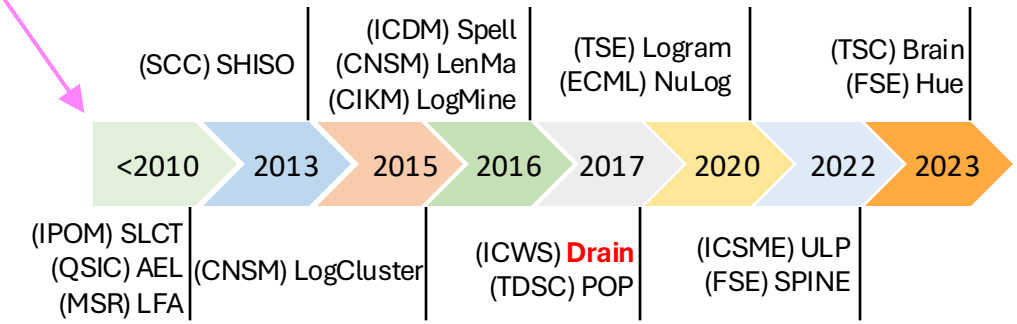
Existent log parsers

A wide range of data-driven parsers have been proposed

Syntax-based log parsers

Semantics-based log parsers

Using statistic features or heuristic rules, e.g., log length and word frequency.



Pros: fast, training-free

Cons: limited accuracy

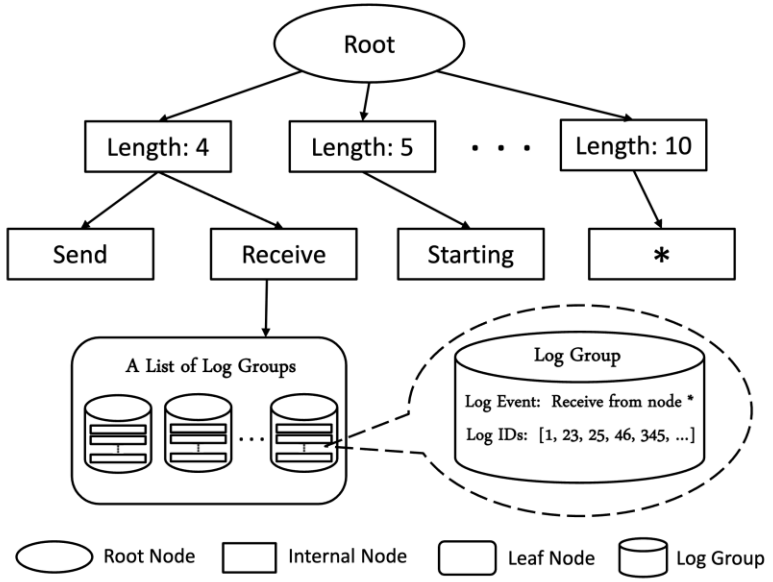


Fig. 2: Structure of Parse Tree in Drain ($depth$ = 3)

An example: Drain

[ICWS'17] Drain: An Online Log Parsing Approach with Fixed Depth Tree

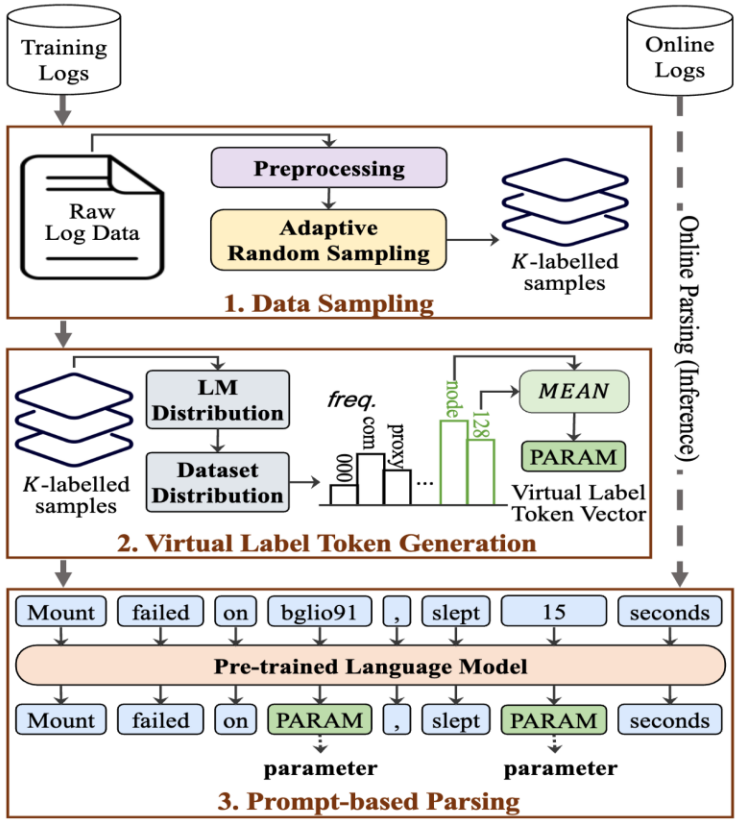
Existent log parsers

A wide range of data-driven parsers have been proposed

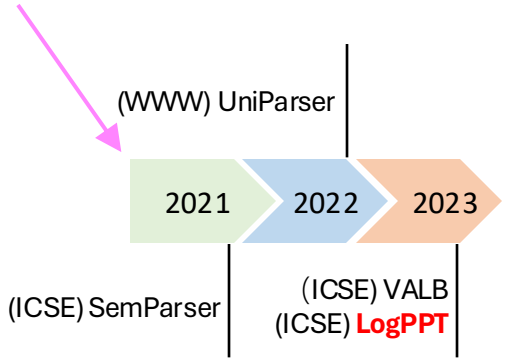
Syntax-based log parsers

Semantics-based log parsers

An example: LogPPT



Leveraging Pretrained Language Models (PLMs) to understand the semantic meaning of log messages.



Pros: accurate

Cons: require labels for training

[ICSE'23] Log Parsing with Prompt-based Few-shot Learning

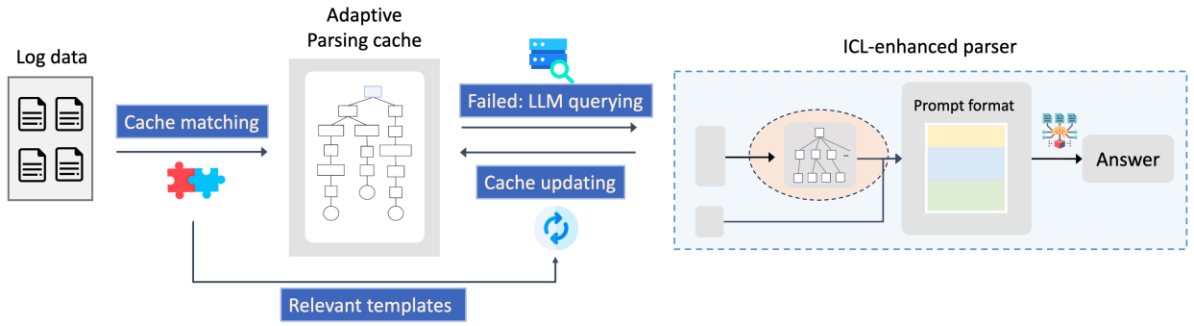
Existent log parsers

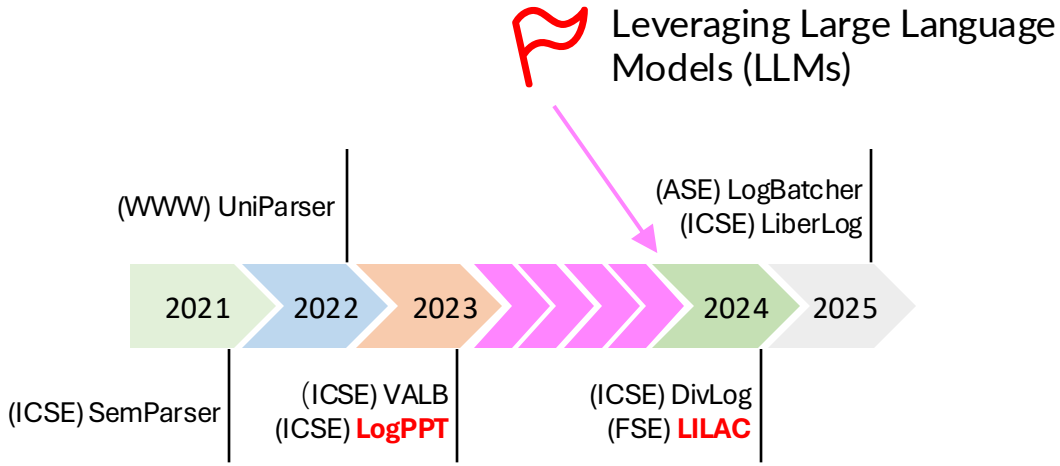
A wide range of data-driven parsers have been proposed

Syntax-based log parsers

Semantics-based log parsers

An example: LILAC



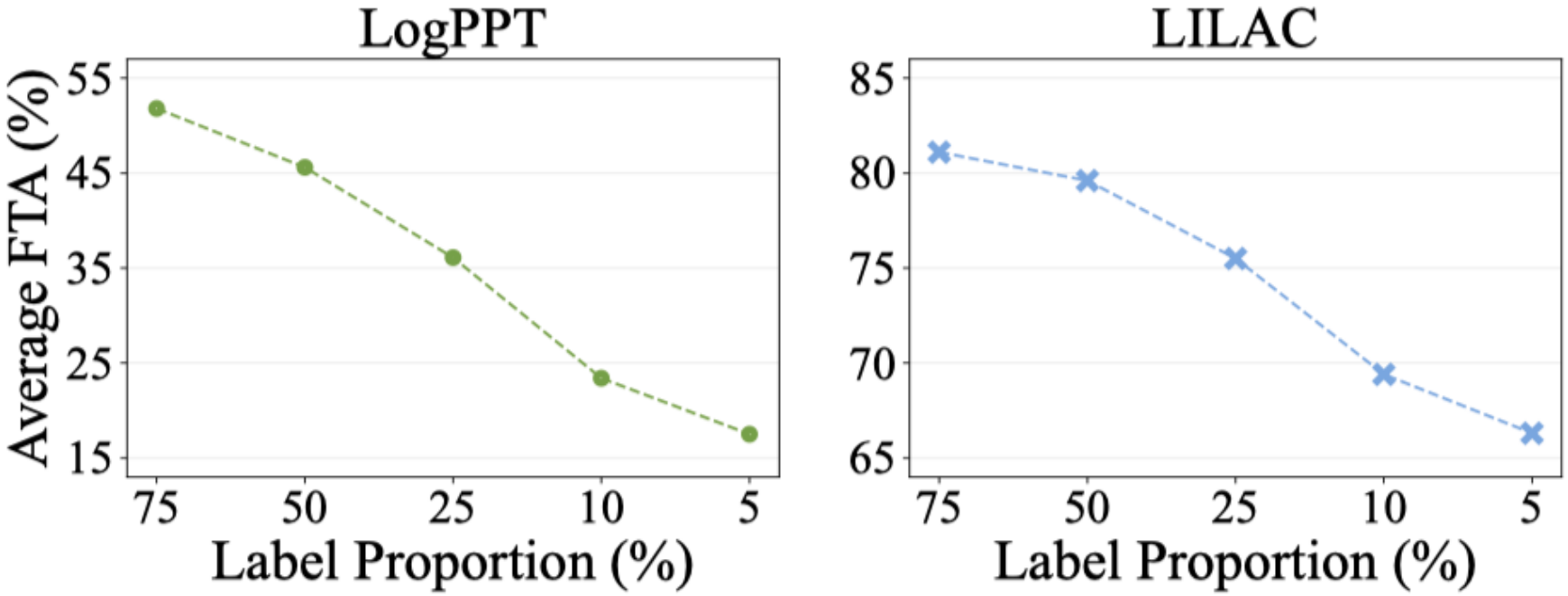


Pros: accurate

Cons: require labels for in-context learning

[FSE'24] LILAC: Log Parsing using LLMs with Adaptive Parsing Cache

Parsing accuracy declines when providing limited labels!



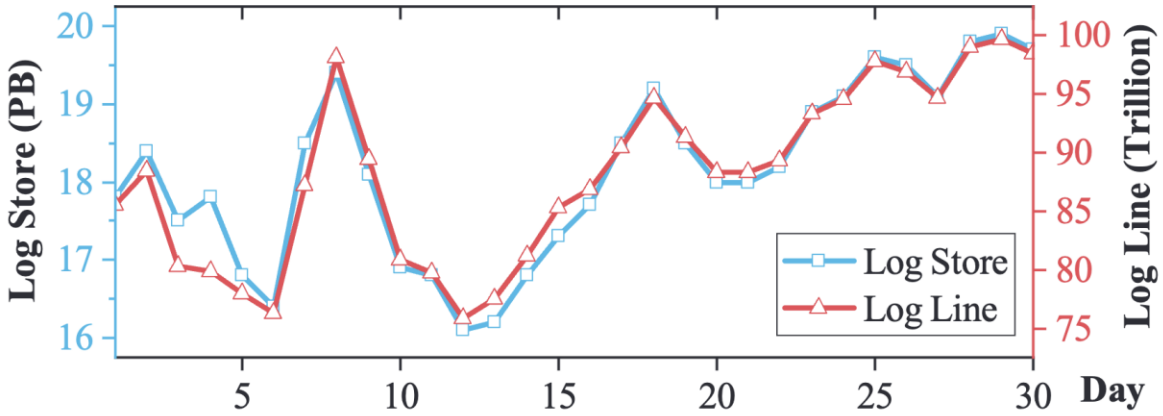
Enough high-quality labels are important for accurate parsers!

[ICSE'23] Log parsing with prompt-based few-shot learning.
[FSE'24] Lilac: Log parsing using llms with adaptive parsing cache.

High quality labels are hard to obtain

- Production log data are huge in volume
  - Annotating at scale is expensive
- Logs can change when software evolves
  - Templates require reannotation

Can we develop a log parser that is accurate but does not rely on labelled examples?



In April 2022, WeChat produced 16-20 PB of logs per day

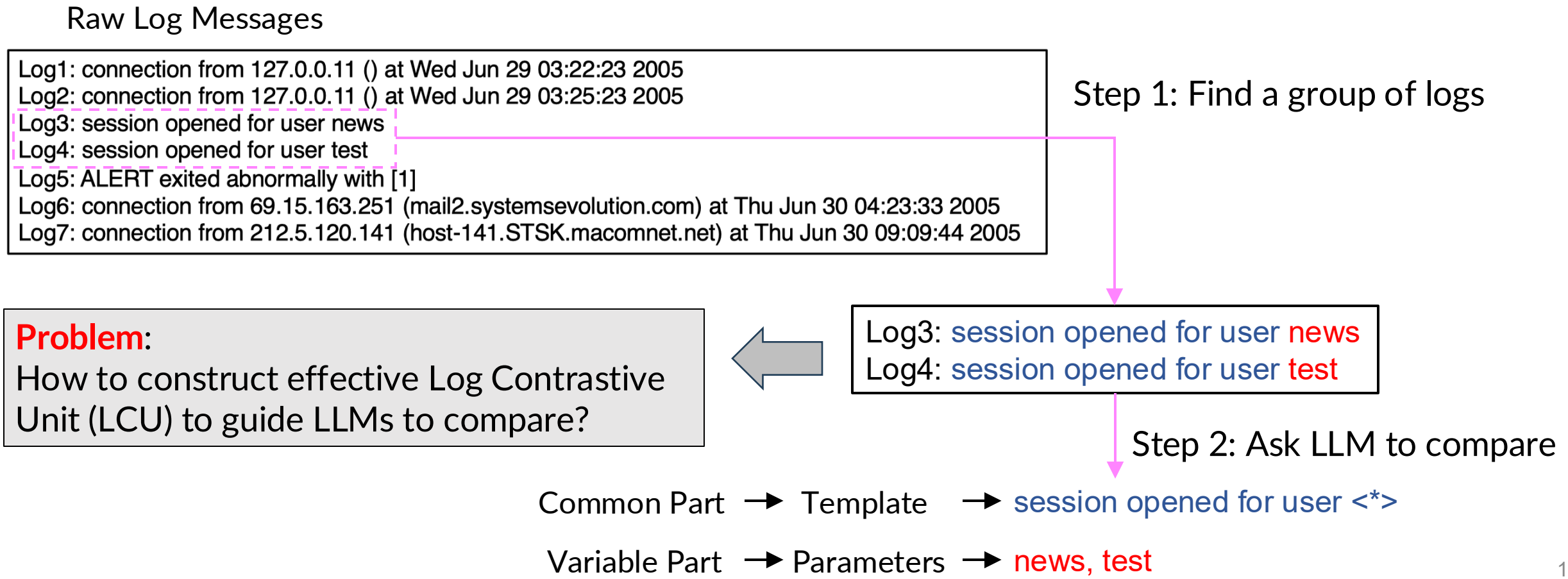
Percentage	Unchanged	Inserted	Paraphrased	Removed
Log message	91.16%	0.07%	8.75%	0.02%
Logging statement	76.12%	12.69%	1.49%	9.70%

Logging evolution ratio between Spark 2.4 and Spark 3.0

[ICSE'23] LogReducer: Identify and Reduce Log Hotspots in Kernel on the Fly
[ISSRE'23] EvLog: Identifying Anomalous Logs over Software Evolution.

Can LLMs compare logs to infer variables?

Core idea: leveraging LLM to perform comparative analysis on multiple log messages



Problem:
How to construct effective Log Contrastive Unit (LCU) to guide LLMs to compare?

Challenges

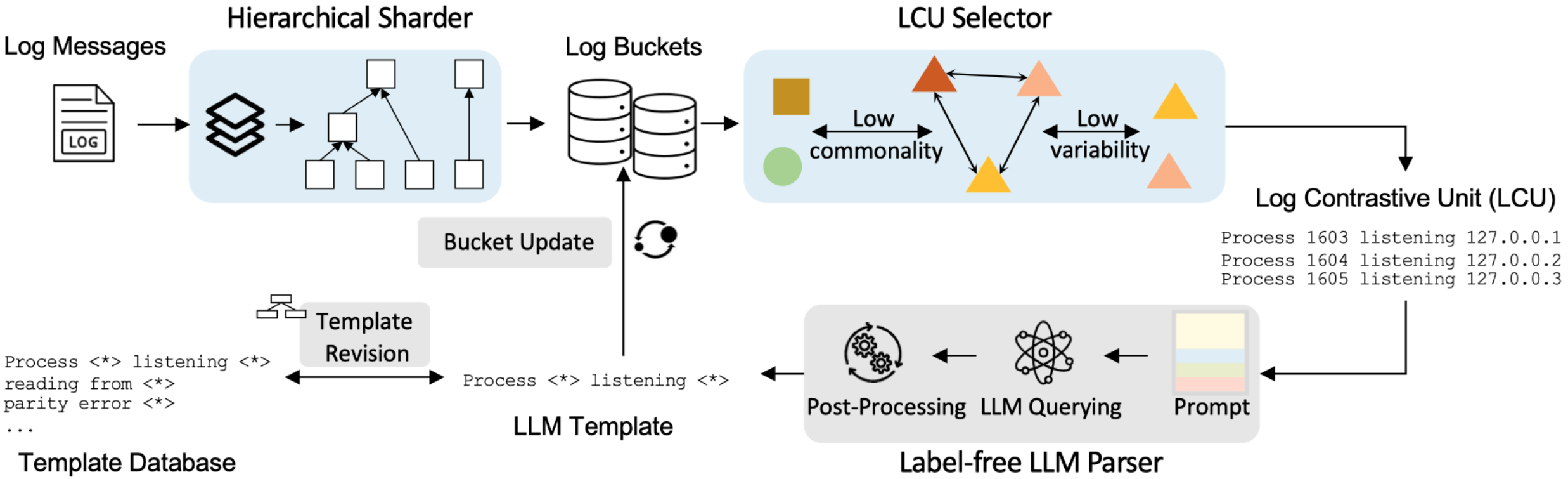
Problem: How to construct effective LCUs to guide LLMs to compare?

* A *Log Contrastive Unit* (LCU) is grouped log messages potentially sharing the same template, allowing LLMs to parse via comparison

- Challenge 1: LCU Volume Explosion

- Challenge 2: Balancing Commonality and Variability
  - Commonality: These log messages should share some common tokens
  - Variability: These log messages should differ in some of their tokens

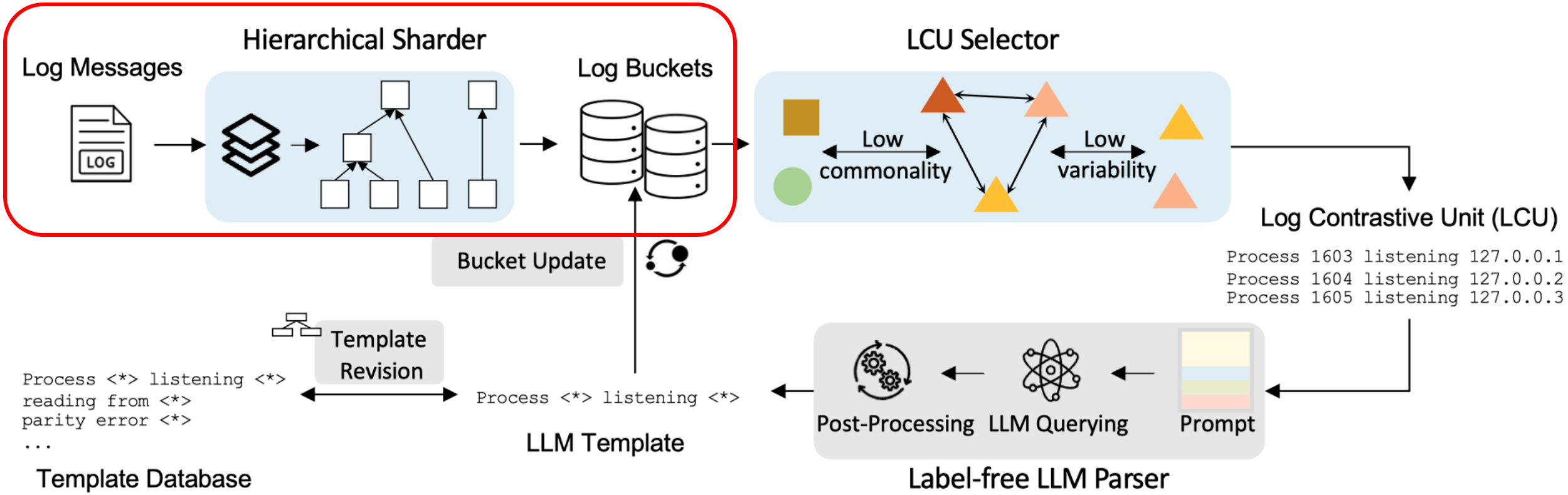
LUNAR: LLM-based Unsupervised Log Parser



Overview framework of LUNAR

LUNAR: LLM-based Unsupervised Log Parser

- Challenge 1: LCU Volume Explosion → **Hierarchical Log Sharding**
  - ✓ Reduce sampling overhead
  - ✓ Enable parallelization



Overview framework of LUNAR

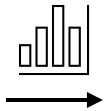
Hierarchical log sharding

- Step 1: Length-based Sharding

But logs of the same length can have different templates…

- Step 2: Top-k Token Agglomerative Clustering

```
Process <*>  listening <*>
Process 1603 listening 127.0.0.1
Process 1603 listening 127.0.0.2
Process 1605 listening 127.0.0.3
```



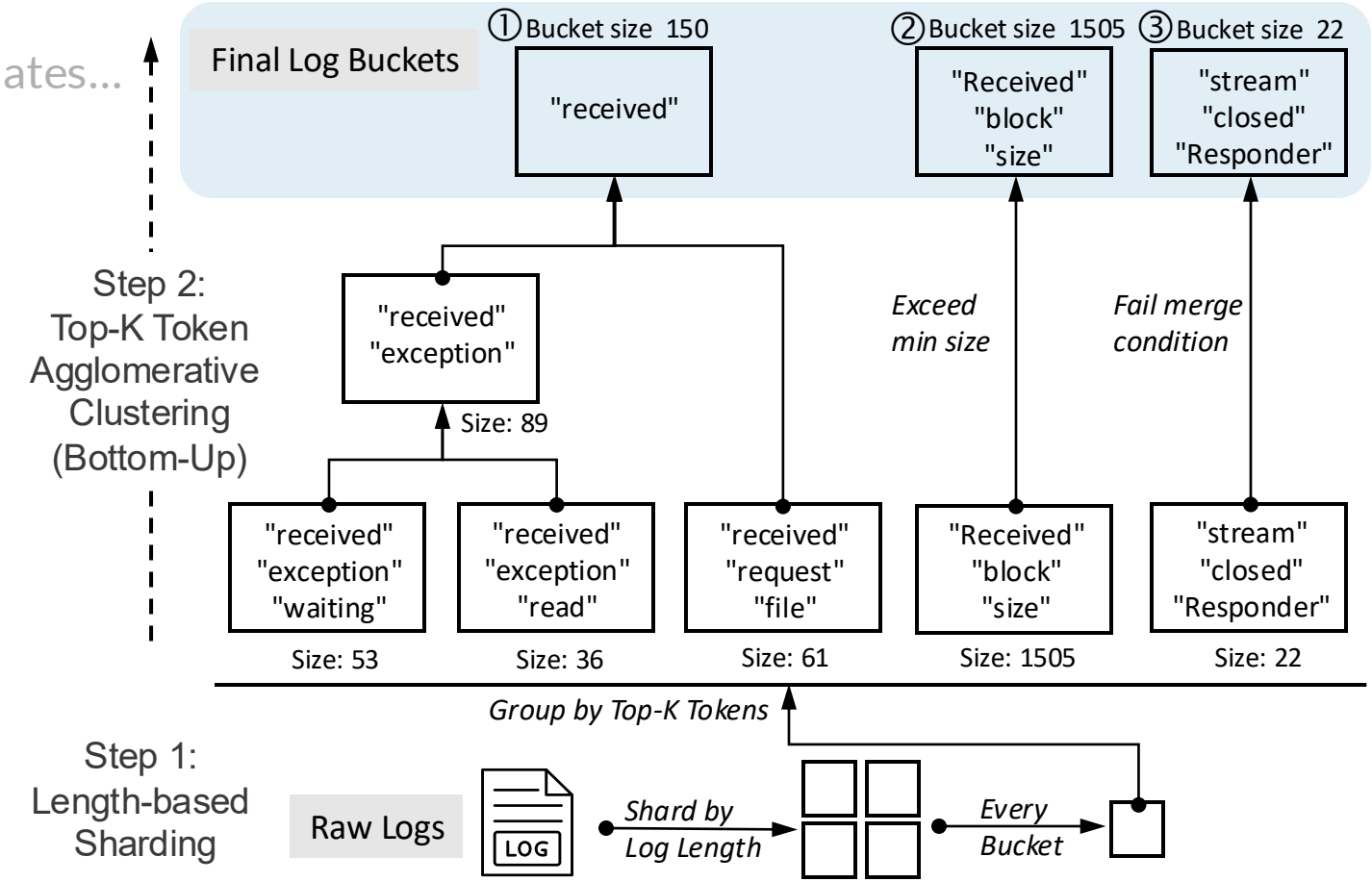
```
Process:    3
Listening:  3
1603:       2
1604:       2
……
```

Received user file upload request
Waiting data processing to complete
Exception waiting received file processing
Exception read received profile data
…

Token frequency

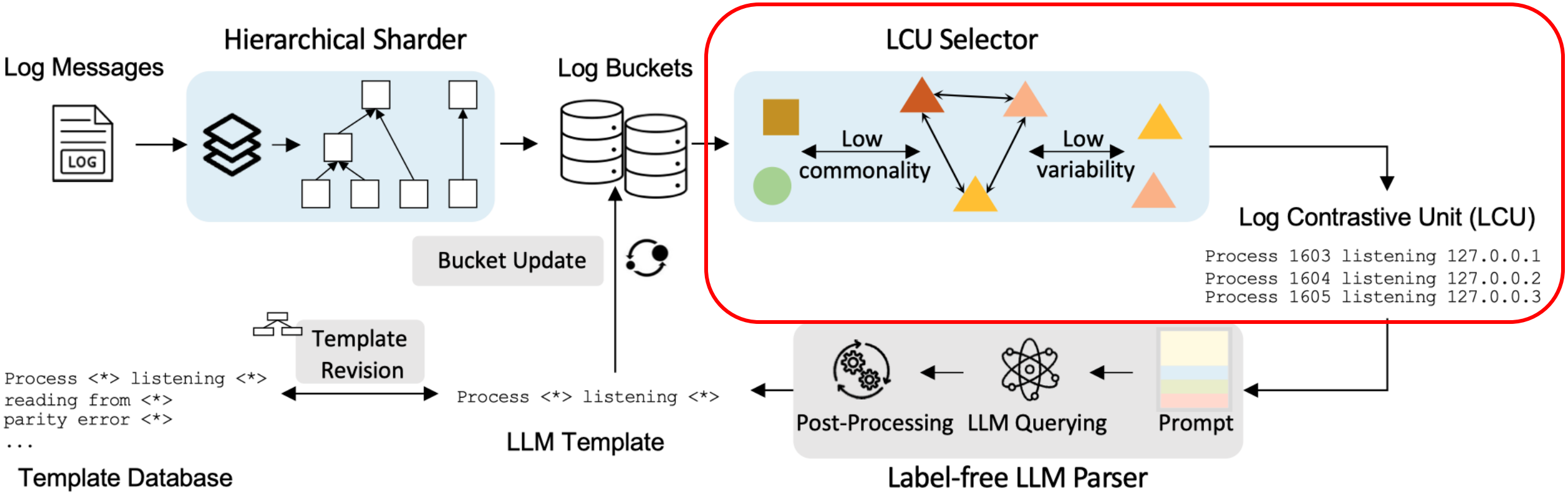
Top-3 Tokens

Received user file upload request	→ (received, request, file)
Waiting user data to complete	→ (data, waiting, user)
Exception waiting received file processing	→ (received exception waiting)
Exception read received profile data	→ (received exception read)



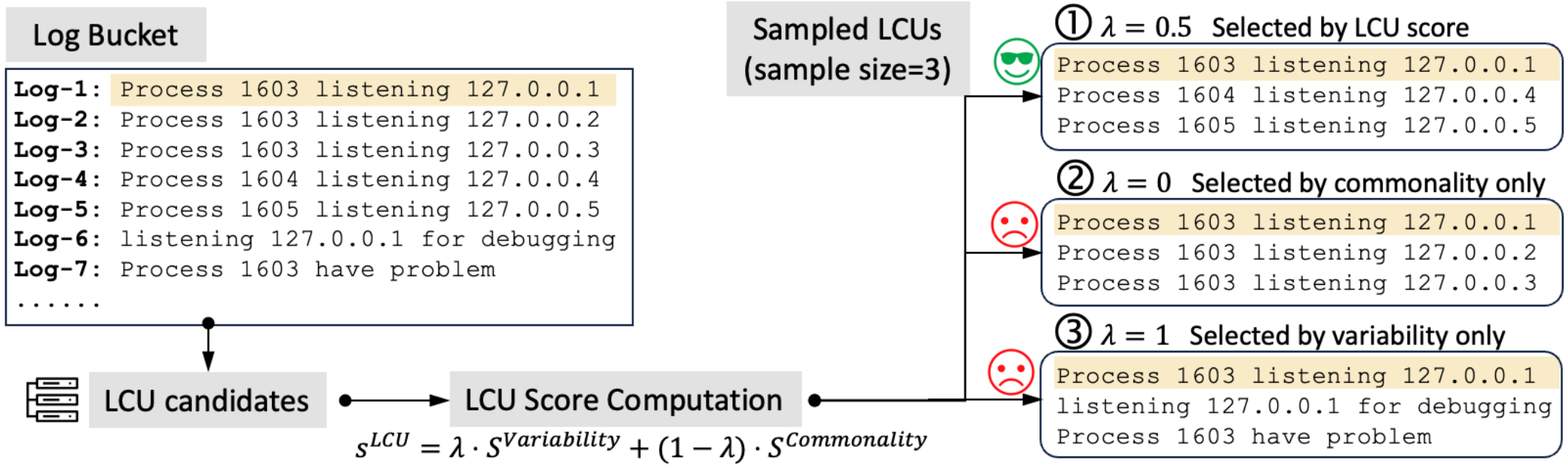
LUNAR: LLM-based Unsupervised Log Parser

- Challenge 2: Balancing Commonality and Variability → **Two-stage LCU Selector**



Overview framework of LUNAR

LUNAR: LLM-based Unsupervised Log Parser



Variability: Average pairwise distance of logs

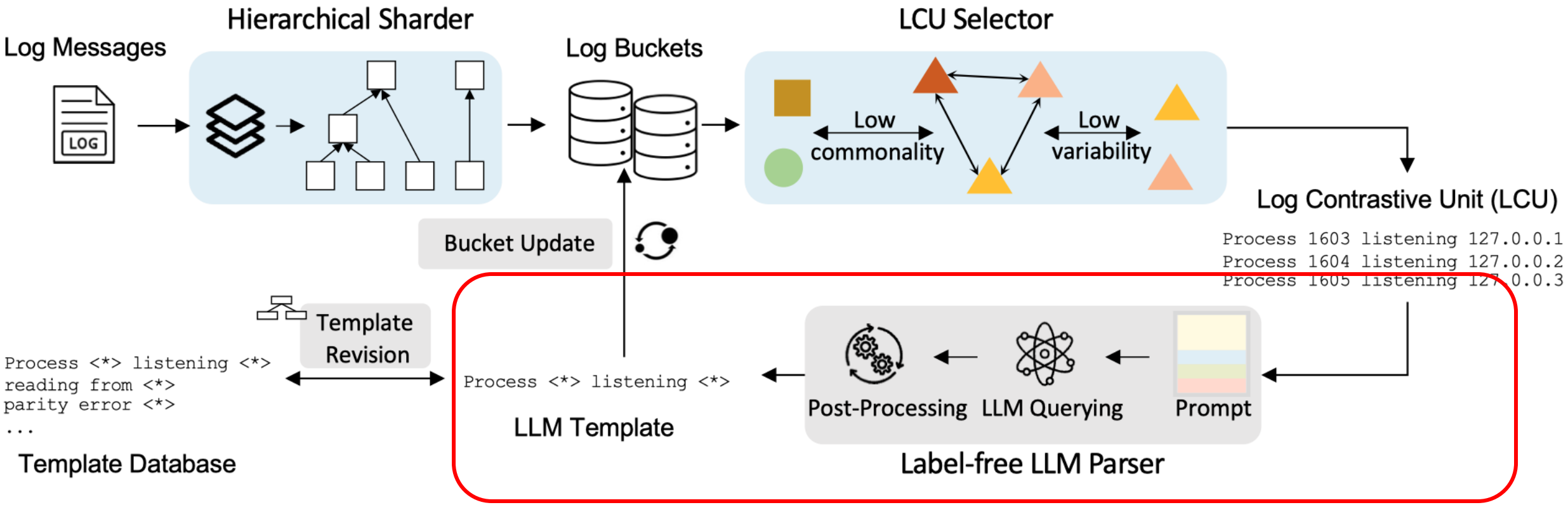
$$S^{Variability} = \frac{2}{L(L-1)} \sum_{i=1}^{L} \sum_{j=i+1}^{L} 1 - \text{Jaccard}(l_i, l_j)$$

Commonality: Average difference in similarity of all pairs

$$S^{Commonality} = \frac{2}{P(P-1)} \sum_{i=1}^{P} \sum_{j=i+1}^{L} 1 - |\text{Jaccard}(p_i) - \text{Jaccard}(p_j)|$$

LUNAR: LLM-based Unsupervised Log Parser

- Step 3: Extract log template for LCU → **Label-free LLM Parsing**



Overview framework of LUNAR

LUNAR: LLM-based Unsupervised Log Parser

Parsing Prompt

- Task Instruction
- Parameter Examples
- Output Constraints
- Queries LCUs

No more labelled examples needed!

Prompt

```
# Basic Requirements:
- I will provide multiple log messages, each delimited by backticks.
- You must identify and extract all dynamic variables in each log with {placeholder} and output static log templates.
- Identify the semantics of variables and compare the differences between logs to identify potential dynamic variables if they belong to the same template.
- Preserve any dynamic variables already marked by `<*>` or `{placeholder}`.
- Pay attention to the slightly different strings among logs, which have high possibility to be dynamic variable.
- Do not convert non-variables, especially when only one log is presented in the group.
# Advices on Parameters and Non-parameters:
- Common variables: numbers, IP addresses, URLs, file paths, directories, hex values, usernames, etc.
- Full directory with filename, complex url with server address or domain should be recognize as one variable.
- Error messages/types, java exceptions, or interrupted messages are NOT dynamic variables as they contain important information. Specific actions or status words are NOT dynamic variables.

# Parameter Examples:
- `/var/www/html/xxx` -> `{directory}`
- `192.168.0.1:8008` -> `{complex_ip}`
- `blk_-123456783` -> `{blk_id}`

# Output Constraints:
- For each log line, output corresponding log template starting with LogTemplate[idx], no other line break.
- Each input log's template is delimited by backticks.

Log[1]: `Received block blk_-160899 of size 91178 from 10.250.10.6`
Log[2]: `Received block blk_-434451 of size 6710 from 10.250.15.8`
Log[3]: `Received block blk_783186  of size 6864 from 10.251.26.198`
```

Prompting LLM

LLM Response

```
LogTemplate[1]: `Received block {blk_id} of size {size} from {ip}`
LogTemplate[2]: `Received block {blk_id} of size {size} from {ip}`
LogTemplate[3]: `Received block {blk_id} of size {size} from {ip}`
```

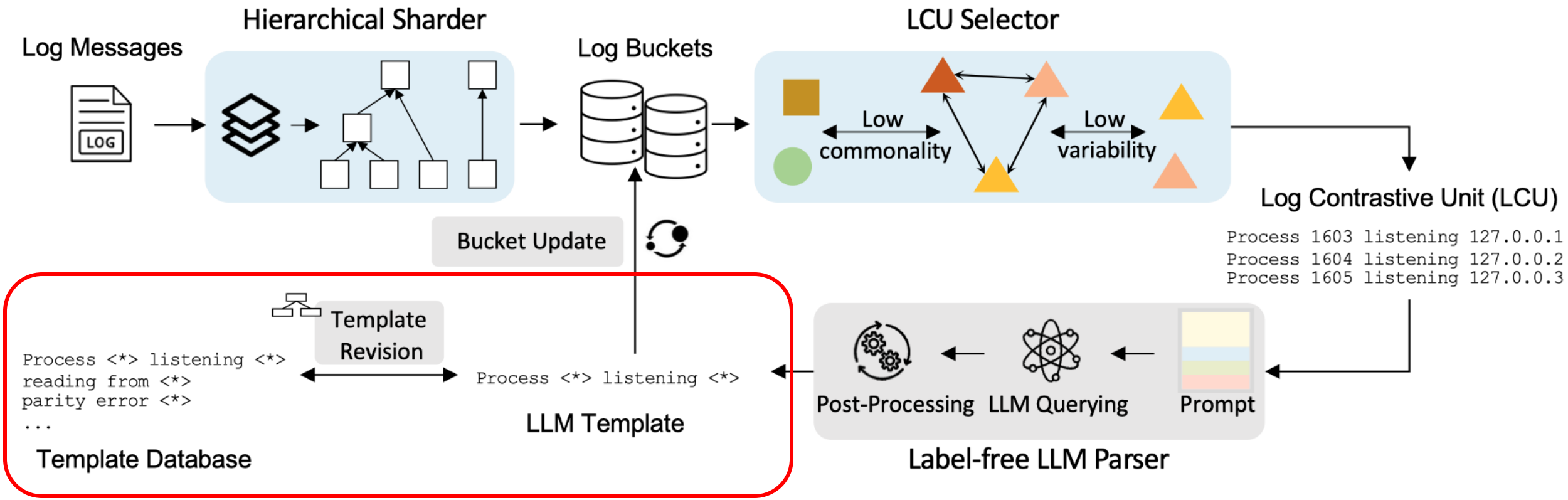
Template Acquisition

Template

```
Received block <*> of size <*> from <*>
```

LUNAR: LLM-based Unsupervised Log Parser

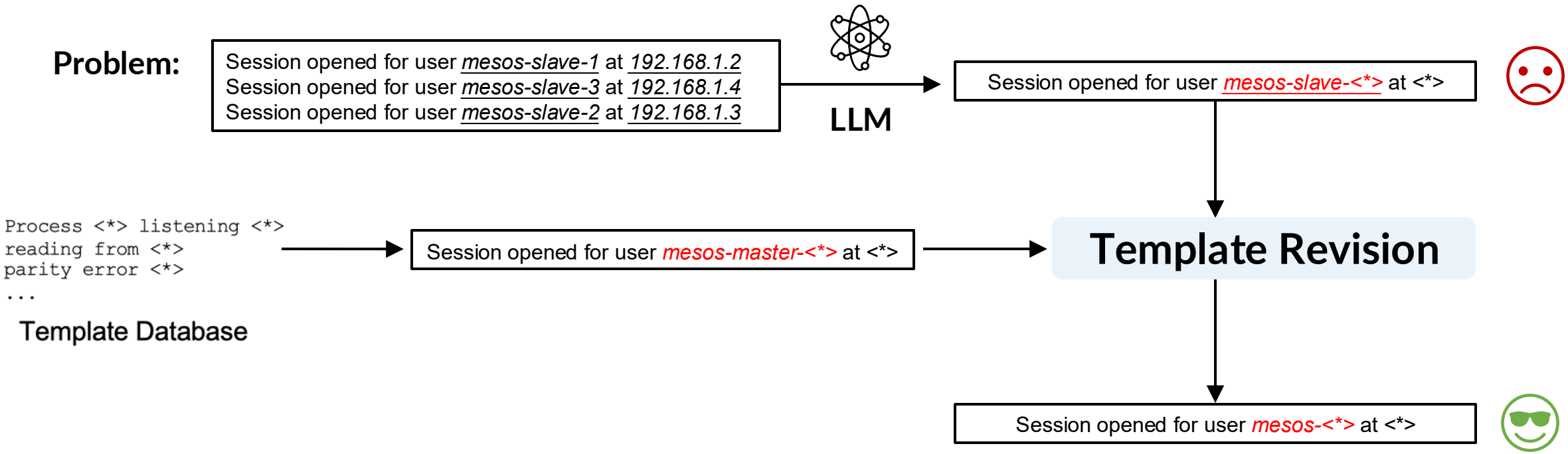
- Step 4: **Template Revision**



Overview framework of LUNAR

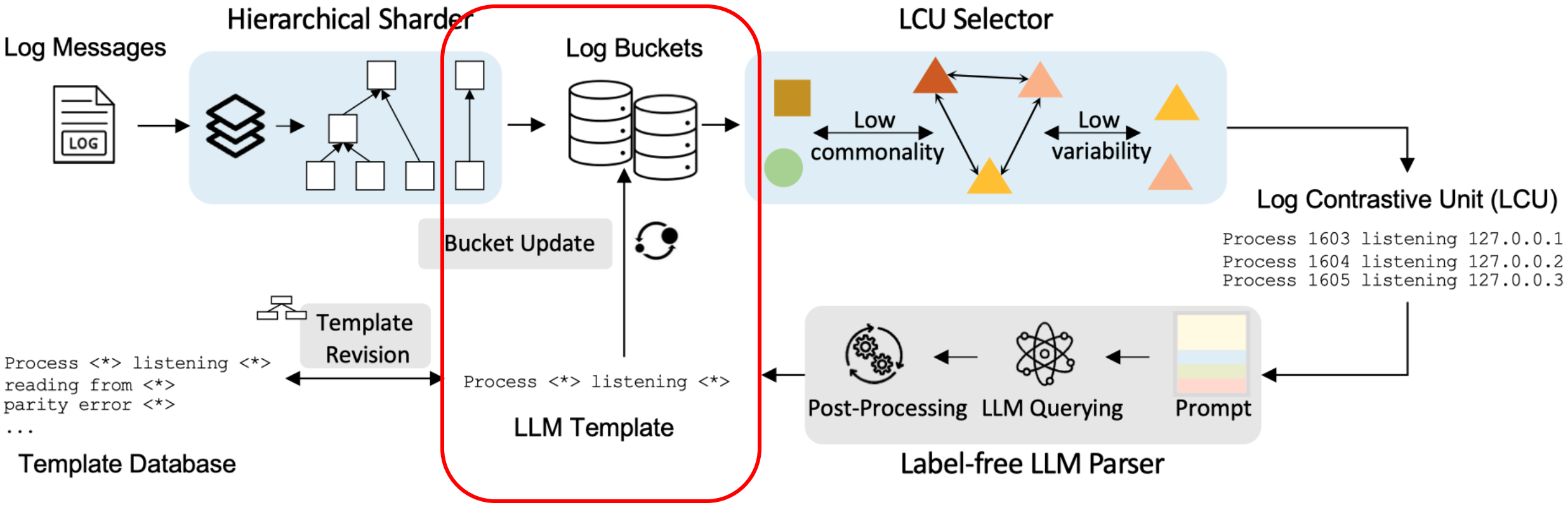
LUNAR: LLM-based Unsupervised Log Parser

- Step 4: **Template Revision**



LUNAR: LLM-based Unsupervised Log Parser

- Step 5: **Bucket Updating**



Overview framework of LUNAR

Evaluation

- RQ1: How effective is LUNAR in log parsing?
- RQ2: How do different components affect LUNAR?
- RQ3: How effective is LUNAR integrated with different LLM and prompts?
- RQ4: How efficient and cost-effective is LUNAR in parsing large-scale log data?

Baselines

Unsupervised methods:
- AEL, Drain, Brain, LogMine, LILAC w/o ICL

Label-required methods:
- UniParser, LogPPT, LILAC

Metrics

Grouping Accuracy (GA)
Parsing Accuracy (PA)
F1 score of Grouping Accuracy (FGA)
F1 score of Teamplate Accuracy (FTA)

Settings

LLM: gpt-3.5-turbo-0125

LCU size: 3 logs

	Dataset	# Logs	# Templates
Open-source Logs [27, 72]	Proxifier	21,320	11
	Apache	51,977	29
	OpenSSH	638,946	38
	HDFS	11,167,740	46
	OpenStack	207,632	48
	HPC	429,987	74
	Zookeeper	74,273	89
	HealthApp	212,394	156
	Hadoop	179,993	236
	Spark	16,075,117	236
	BGL	4,631,261	320
	Linux	23,921	338
	Mac	100,314	626
	Thunderbird	16,601,745	1,241
Industrial Logs [21, 63]	CTS [63]	264	78
	HiBench [63]	1,879	93
	LoFI [21]	2,080	453
Average		2,965,931.9	230.1

Statistics of Log Parsing Datasets

Never seen by LLMs!

RQ1: Effectiveness – compared to *unsupervised* parsers

Method	Metric	Proxifier	Apache	OpenSSH	HDFS	OpenStack	HPC	Zookeeper	HealthApp	Hadoop	Spark	BGL	Linux	Mac	Thunderbird	CTS	HiBench	LoFI	Avg.
Unsupervised Log Parsers																			
AEL	GA	97.4	**100.0**	70.5	99.9	74.3	74.8	99.6	72.5	82.3	—	91.5	**91.6**	79.7	78.6	78.8	92.1	84.5	85.5
	PA	67.7	72.7	36.4	62.1	2.9	74.1	84.2	31.1	53.5	—	40.6	8.2	24.5	16.3	62.5	56.7	30.4	45.2
	FGA	66.7	**100.0**	68.9	76.4	68.2	20.1	78.8	0.8	11.7	—	58.7	80.6	79.3	11.6	79.2	87.6	84.4	60.8
	FTA	41.7	51.7	33.3	56.2	16.5	13.6	46.5	0.3	5.8	—	16.5	21.7	20.5	3.5	59.7	48.6	25.0	28.8
Drain	GA	69.2	**100.0**	70.7	99.9	75.2	79.3	99.4	86.2	92.1	88.8	91.9	68.6	76.1	83.1	79.9	61.6	79.0	82.4
	PA	68.8	72.7	58.6	62.1	2.9	72.1	84.3	31.2	54.1	39.4	40.7	11.1	35.7	21.6	66.7	26.8	29.5	45.8
	FGA	20.6	**100.0**	87.2	93.5	0.7	30.9	90.4	1.0	78.5	86.1	62.4	77.8	22.9	23.7	69.3	46.0	78.5	57.0
	FTA	17.6	51.7	48.7	60.9	0.2	15.2	61.4	0.4	38.4	41.2	19.3	25.9	6.9	7.1	57.0	23.6	22.6	29.3
Brain	GA	52.1	99.7	66.3	96.0	**100.0**	80.0	99.3	97.9	56.3	97.2	94.0	79.0	83.4	79.2	80.3	79.1	64.3	82.6
	PA	70.3	28.7	48.1	92.9	14.1	66.3	82.2	17.5	14.3	39.3	40.2	1.0	32.5	26.1	70.1	62.2	25.3	43.0
	FGA	73.7	93.3	75.9	75.9	**100.0**	44.7	79.8	87.2	52.8	20.8	75.6	75.1	75.4	74.8	80.0	68.8	69.3	71.9
	FTA	73.7	46.7	34.5	62.1	29.2	21.3	60.1	33.9	20.0	1.0	19.7	27.5	29.4	27.4	69.3	47.7	21.7	36.8
LogMine	GA	50.4	**100.0**	—	—	—	—	69.7	—	82.7	—	64.4	73.6	85.1	—	87.5	79.7	81.0	77.4
	PA	0.0	26.2	—	—	—	—	45.6	—	52.9	—	9.7	3.5	28.3	—	69.7	43.3	29.6	30.9
	FGA	0.8	**100.0**	—	—	—	—	1.6	—	12.4	—	24.7	75.1	45.1	—	83.5	85.1	81.6	51.0
	FTA	0.0	41.4	—	—	—	—	0.8	—	6.3	—	3.7	19.5	12.2	—	65.8	43.6	24.0	21.7
LILAC w/o ICL	GA	52.1	99.7	74.6	**100.0**	52.4	**87.0**	99.8	**100.0**	85.8	90.0	86.2	77.9	78.8	66.7	86.4	69.4	59.8	80.4
	PA	67.1	97.0	43.4	94.7	37.4	93.3	58.1	56.2	71.0	71.5	78.2	78.6	50.1	43.2	76.9	23.1	33.9	63.2
	FGA	42.4	91.8	78.3	69.3	90.0	**88.9**	95.0	**97.1**	90.0	82.8	83.2	79.0	77.3	35.4	80.2	43.2	65.6	75.9
	FTA	48.5	72.1	49.3	56.0	68.0	75.0	74.0	71.7	66.7	60.0	66.3	57.4	46.7	21.9	68.9	22.2	30.0	56.2
Our proposed method																			
LUNAR	GA	98.9	**100.0**	**78.0**	**100.0**	**100.0**	86.4	99.3	**100.0**	**94.1**	97.5	**95.5**	83.0	**87.7**	**86.5**	**97.0**	93.0	**90.5**	**93.4**
	PA	**100.0**	**99.9**	72.2	**100.0**	90.4	99.0	85.1	96.2	83.7	**99.5**	**98.4**	73.5	58.3	59.5	**86.0**	73.5	**74.5**	**85.3**
	FGA	87.0	**100.0**	**92.3**	**96.8**	**100.0**	83.3	88.5	**97.1**	92.6	88.8	87.3	**87.4**	86.3	**86.9**	**96.1**	**88.4**	**92.6**	**91.3**
	FTA	95.7	**89.7**	**92.3**	**94.6**	**87.5**	**83.3**	81.2	**84.3**	**70.2**	**65.7**	**78.9**	**72.1**	**52.6**	**57.4**	**87.0**	**67.4**	**73.8**	**78.5**

LUNAR outperforms all *unsupervised* parsers!

RQ1: Effectiveness – compared to *label-required* parsers

Method	Metric	Proxifier	Apache	OpenSSH	HDFS	OpenStack	HPC	Zookeeper	HealthApp	Hadoop	Spark	BGL	Linux	Mac	Thunderbird	CTS	HiBench	LoFI	Avg.
Label-required Log Parsers																			
UniParser	GA	50.9	94.8	27.5	**100.0**	**100.0**	77.7	98.8	46.1	69.1	85.4	91.8	28.5	73.7	57.9	61.7	85.8	71.9	71.9
	PA	63.4	94.2	28.9	94.8	51.6	94.1	**98.8**	81.7	**88.9**	79.5	94.9	16.4	**68.8**	**65.4**	56.1	74.5	57.3	71.1
	FGA	28.6	68.7	0.9	**96.8**	96.9	66.0	66.1	74.5	62.8	2.0	62.4	45.1	69.9	68.2	54.5	73.7	77.8	59.7
	FTA	45.7	26.9	0.5	58.1	28.9	35.1	51.0	46.2	47.6	1.2	21.9	23.2	28.3	29.0	46.5	54.4	35.7	34.1
LogPPT	GA	98.9	78.6	27.7	72.1	53.4	78.2	96.7	99.8	48.3	47.6	24.5	20.5	54.4	56.4	95.1	29.6	8.7	58.3
	PA	**100.0**	94.8	65.4	94.3	40.6	**99.7**	84.5	**99.7**	66.6	95.2	93.8	16.8	39.0	40.1	**86.0**	9.0	6.5	66.6
	FGA	87.0	60.5	8.1	39.1	87.4	78.0	91.8	94.7	52.6	37.4	25.3	71.2	49.3	21.6	92.5	20.9	19.9	55.1
	FTA	95.7	36.8	10.5	31.2	73.8	76.8	80.9	82.2	43.4	29.9	26.1	42.8	27.4	11.7	85.0	9.0	5.4	45.2
LILAC	GA	**100.0**	**100.0**	74.8	**100.0**	**100.0**	86.9	**100.0**	**100.0**	92.6	**99.9**	91.1	82.5	81.4	79.4	85.6	**95.8**	78.2	91.1
	PA	**100.0**	97.2	**99.9**	94.7	**95.2**	93.7	68.5	60.4	73.2	68.9	90.4	**81.4**	56.4	53.3	77.7	**78.9**	48.6	78.7
	FGA	**100.0**	**100.0**	87.7	**96.8**	**100.0**	86.1	**96.7**	**97.1**	**94.1**	**89.7**	**88.5**	86.1	**87.1**	85.9	86.9	84.7	85.2	**91.3**
	FTA	**100.0**	86.2	84.9	71.0	83.3	76.4	**83.5**	77.5	69.5	63.6	72.1	64.4	48.7	55.7	77.2	61.4	53.6	72.3
Our proposed method																			
LUNAR	GA	98.9	**100.0**	**78.0**	**100.0**	**100.0**	86.4	99.3	**100.0**	**94.1**	97.5	**95.5**	83.0	**87.7**	**86.5**	**97.0**	93.0	**90.5**	**93.4**
	PA	**100.0**	**99.9**	72.2	**100.0**	90.4	99.0	85.1	96.2	83.7	**99.5**	**98.4**	73.5	58.3	59.5	**86.0**	73.5	**74.5**	**85.3**
	FGA	87.0	**100.0**	**92.3**	**96.8**	**100.0**	83.3	88.5	**97.1**	92.6	88.8	87.3	**87.4**	86.3	**86.9**	**96.1**	**88.4**	**92.6**	**91.3**
	FTA	95.7	**89.7**	**92.3**	**94.6**	**87.5**	**83.3**	81.2	**84.3**	**70.2**	**65.7**	**78.9**	**72.1**	**52.6**	**57.4**	**87.0**	**67.4**	**73.8**	**78.5**

LUNAR also outperforms all *label-required* parsers!

LUNAR can *generalize to unseen logs* effectively w/o labels!

RQ2: Ablation study of components

Metrics	GA	PA	FGA	FTA
LUNAR	93.4	86.8	91.0	79.0
Variants w.r.t Log Sharder				
w/o log length	89.1 (↓ 4.6%)	83.2 (↓ 4.2%)	90.1 (↓ 1.0%)	78.3 (↓ 0.9%)
w/o top-k tokens	89.1 (↓ 4.6%)	82.9 (↓ 4.5%)	90.4 (↓ 0.6%)	78.2 (↓ 0.9%)
Variants w.r.t LCU Selector				
w/ random selection	90.6 (↓ 3.0%)	84.3 (↓ 2.9%)	90.5 (↓ 0.5%)	77.7 (↓ 1.6%)
w/ minimum similarity	88.8 (↓ 4.6%)	83.9 (↓ 4.5%)	89.5 (↓ 1.9%)	80.2 (↓ 1.4%)
w/ maximum similarity	89.1 (↓ 4.6%)	82.8 (↓ 4.6%)	88.9 (↓ 2.3%)	75.8 (↓ 4.0%)
w/ consecutive selection	89.2 (↓ 4.5%)	82.6 (↓ 4.8%)	90.2 (↓ 0.9%)	77.5 (↓ 1.9%)

Average accuracy across all datasets

All designs of *log sharder* and *LCU selector* contribute to the overall performance

RQ3: Different LLMs and prompts

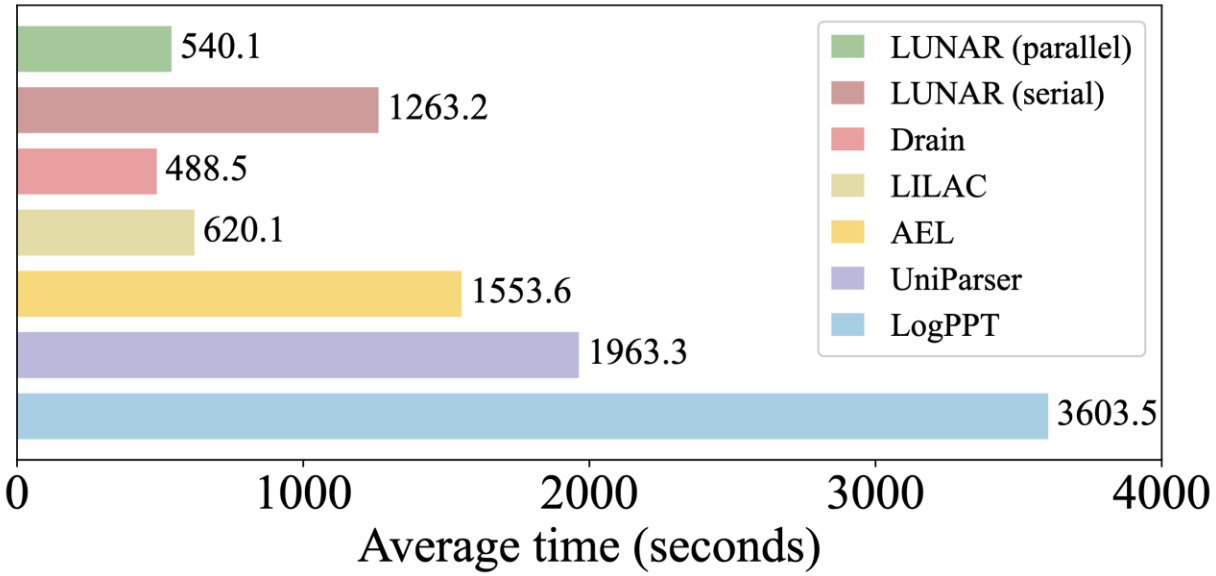
Metrics	GA	PA	FGA	FTA
LUNAR (GPT-3.5)	93.4	86.8	91.0	79.0
Variants w.r.t Parsing Prompt				
w/o parameter examples	91.9 (↓ 1.6%)	84.1 (↓ 3.1%)	90.6 (↓ 0.4%)	77.0 (↓ 2.5%)
w/o output constraints	87.3 (↓ 6.5%)	70.1 (↓ 19.2%)	63.5 (↓ 30.2%)	47.2 (↓ 40.3%)
w/o parameter advice	93.3 (↓ 0.1%)	84.9 (↓ 2.2%)	91.0 (↓ 0.0%)	76.9 (↓ 2.7%)
w/ simpler requirements	89.3 (↓ 4.4%)	82.9 (↓ 4.5%)	87.4 (↓ 4.0%)	76.2 (↓ 3.5%)
Variants w.r.t LLM				
GPT-4	93.4 (↑ 0.0%)	87.0 (↑ 0.2%)	92.1 (↑ 1.2%)	79.8 (↑ 1.0%)
GPT-4o	86.2 (↓ 7.7%)	79.5 (↓ 8.4%)	88.5 (↓ 2.7%)	75.1 (↓ 4.9%)
Claude	90.4 (↓ 3.2%)	84.1 (↓ 3.1%)	90.0 (↓ 1.1%)	78.1 (↓ 1.1%)
Llama-3.1-405B	92.9 (↓ 0.5%)	86.9 (↑ 0.1%)	90.8 (↓ 0.2%)	80.1 (↑ 1.4%)
DeepSeek-V2.5-236B	90.1 (↓ 3.5%)	82.3 (↓ 5.2%)	90.3 (↓ 0.8%)	78.2 (↓ 1.0%)
Qwen2-72B	89.6 (↓ 4.1%)	83.1 (↓ 4.3%)	89.0 (↓ 2.2%)	76.1 (↓ 3.7%)
Llama-3.1-70B	86.6 (↓ 7.3%)	80.6 (↓ 7.1%)	89.0 (↓ 2.2%)	77.4 (↓ 2.0%)
Llama-3.1-8B	90.8 (↓ 2.8%)	75.4 (↓ 13.1%)	88.2 (↓ 3.1%)	65.3 (↓ 17.3%)
Qwen2-7B	86.0 (↓ 7.9%)	68.1 (↓ 21.5%)	87.3 (↓ 4.1%)	59.3 (↓ 24.9%)

Average accuracy across all datasets

- All four components in the parsing prompt contribute to overall performance.
- Removing output constraints results in the biggest drop.

- LUNAR consistently achieves high performance with different LLMs.
- LUNAR's performance can be affected by the capabilities of LLMs.

RQ4: Efficiency and cost



- LUNAR (parallel) is faster than SOTA LLM-based methods (LILAC)
- LUNAR is comparable to the fastest syntax-based methods in efficiency

	Avg. Accuracy	# Input Tokens	# Output Tokens	# Invocation	$ Cost
LILAC w/o ICL	68.9	116.3K	26.8K	726.3	$0.098
LILAC	83.3	68.4K	4.5K	219.0	$0.041
LUNAR	87.1	131.7K	13.9K	269.4	$0.086

Token consumption of LLM-based parser

- LUNAR improves performance without introducing significant cost

Conclusion

- **LUNAR: an LLM-based Unsupervised Log Parser**

- **Comprehensive Evaluation**:
  - 14 public datasets and 3 industrial datasets (New LoFI Dataset!)
  - LUNAR outperforms both *unsupervised* and *label-required* parser
  - Both *accurate* and *efficient*!

- Merits from one reviewer:

> *"It also helps that this approach seems straight-forward enough that I could see companies adopt it without too much hassle (not something that can be said about all existing works in this area).* ***I'm excited about recommending this approach to some of my industrial partners."***

Check this work in:

Paper

Data & Code

Collected by LogPAI

LOGPAI

Log Analytics Powered by AI

Q & A