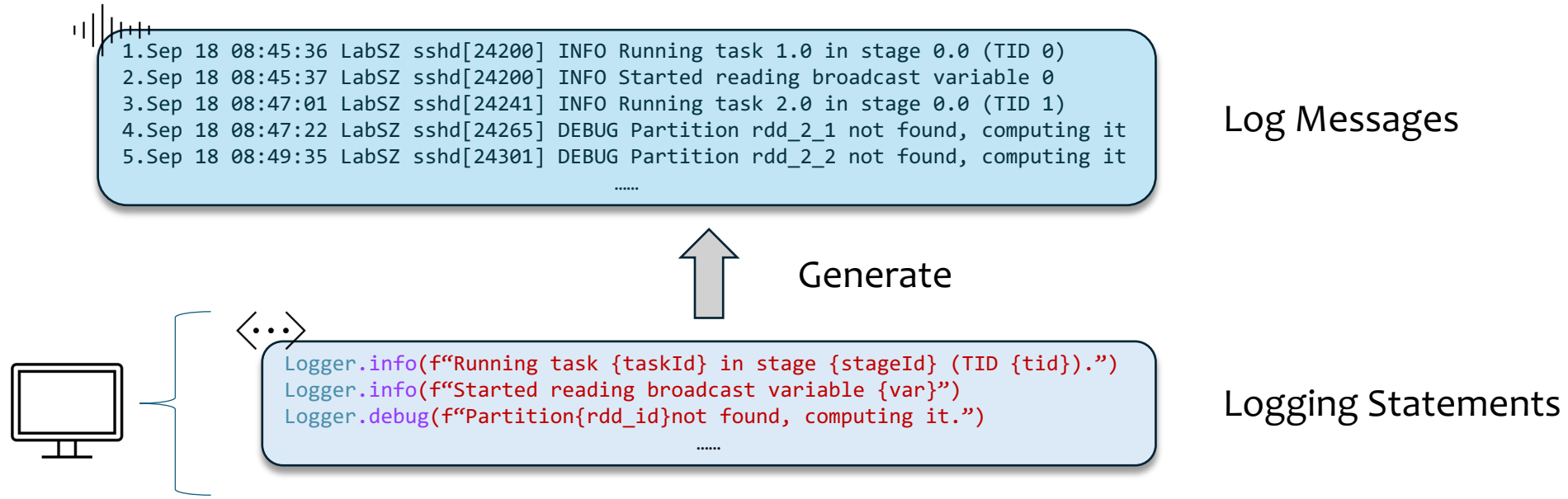
LILAC: Log Parsing using LLMs with Adaptive Parsing Cache

Zhihan Jiang[1], Jinyang Liu[1], Zhuangbin Chen[2], Yichen Li[1], Junjie Huang[1], Yintong Huo[1], Pinjia He[3], Jiazhen Gu[1] and Michael R. Lyu[1]

[1]*The Chinese University of Hong Kong,*
[2]*Sun Yat-sen University,*
[3]*The Chinese University of Hong Kong (Shenzhen)*

System Log

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



System Log

Automated Log Analysis is important for:

Anomaly Detection

Fault Localization

Root Cause Analysis

```
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
……
```

Log Messages

Generate

```
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.”)
……
```

Logging Statements

Log Parsing

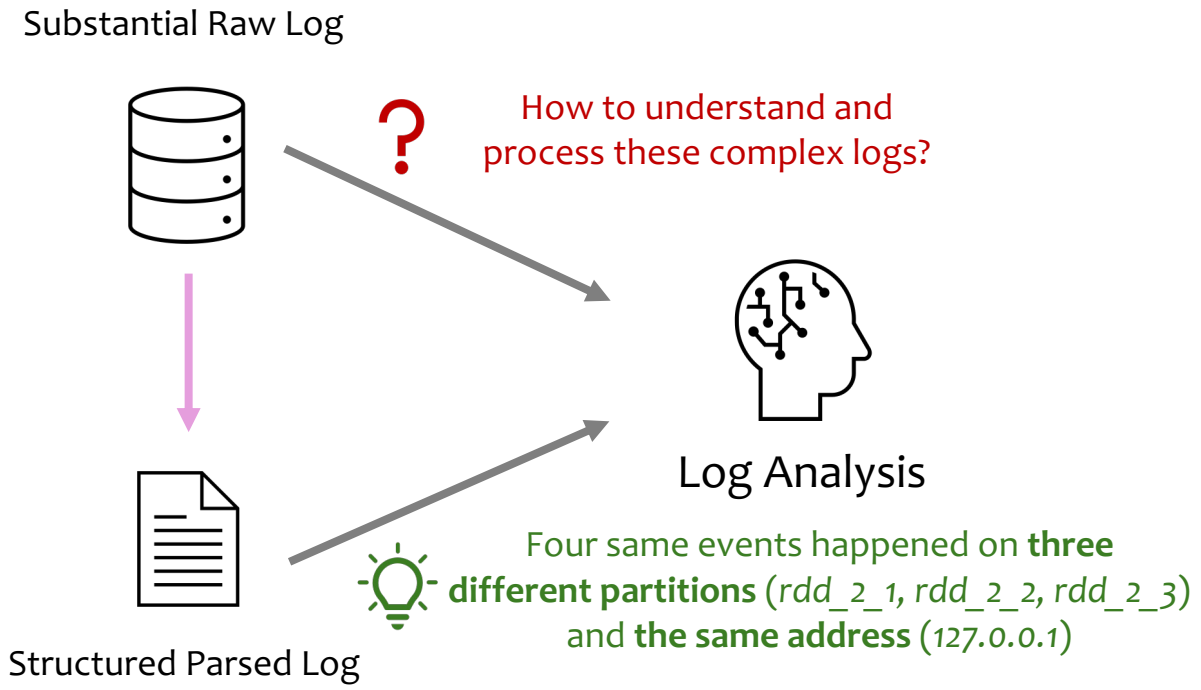
Logs are unstructured, which are not suitable for *understanding*, *analyzing* and *storing*.

Log Parsing

Converting raw logs into structed formats is a critical prerequisite step!

```
Sep 18 08:47:22 DEBUG Partition rdd_2_1 not found in 127.0.0.1
Sep 18 08:47:35 DEBUG Partition rdd_2_2 not found in 127.0.0.1
Sep 18 08:47:49 DEBUG Partition rdd_2_1 not found in 127.0.0.1
Sep 18 08:48:17 DEBUG Partition rdd_2_3 not found in 127.0.0.1
```

Timestamp	Event	Level	Log Templates	Parameters
Sep 18 08:47:22	e1	DEBUG	Partition <*> not found in <*>	rdd_2_1 | 127.0.0.1
Sep 18 08:47:35	e1	DEBUG	Partition <*> not found in <*>	rdd_2_2 | 127.0.0.1
Sep 18 08:47:49	e1	DEBUG	Partition <*> not found in <*>	rdd_2_1 | 127.0.0.1
Sep 18 08:48:17	e1	DEBUG	Partition <*> not found in <*>	rdd_2_3 | 127.0.0.1



Log Parsing

Goal: to distinguish between constant part and variable 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
.....				

Parsed Results

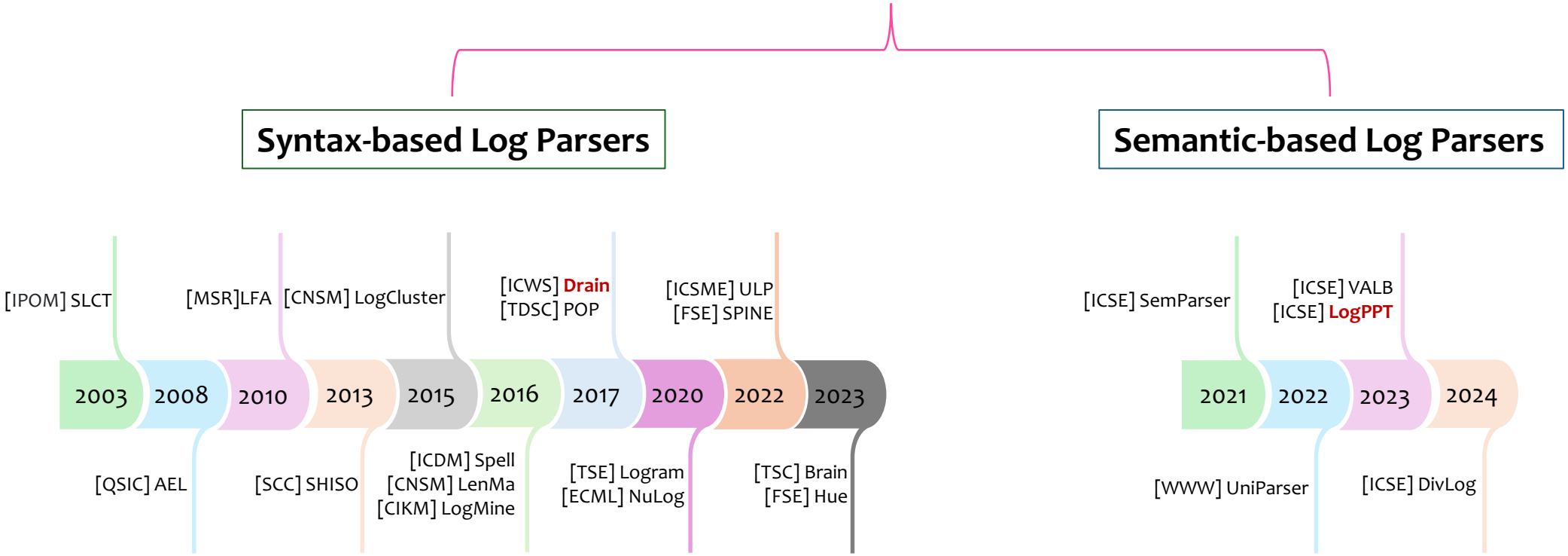
Log Parsing

```
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
.....
```

Raw Logs

Related Work

A wide range of data-driven methods have been proposed to parse logs.

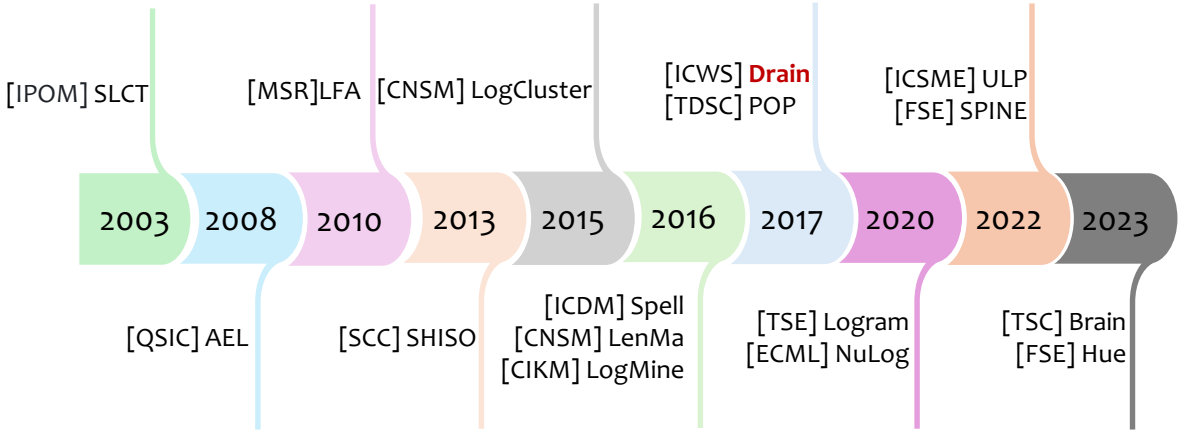


Limitation of Existing Parsers

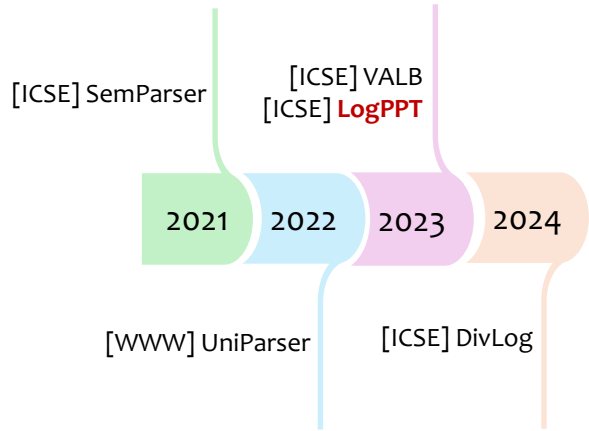
Logs are continously evolving!

The performance of existing log parsers in practice remains unsatisfatory.

Syntax-based Log Parsers



Semantic-based Log Parsers



"Guidelines for assessing the accuracy of log message template identification techniques", ICSE'22
"Investigating and improving log parsing in practice", FSE'22
"A Large-scale Evaluation for Log Parsing Techniques: How Far are We?", ISSTA'24

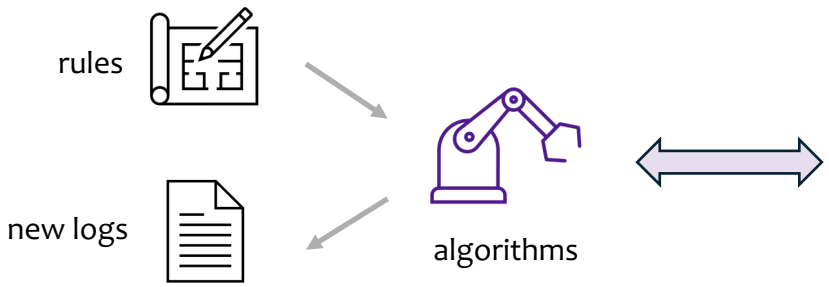
Limitation of Existing Parsers

Logs are continously evolving!

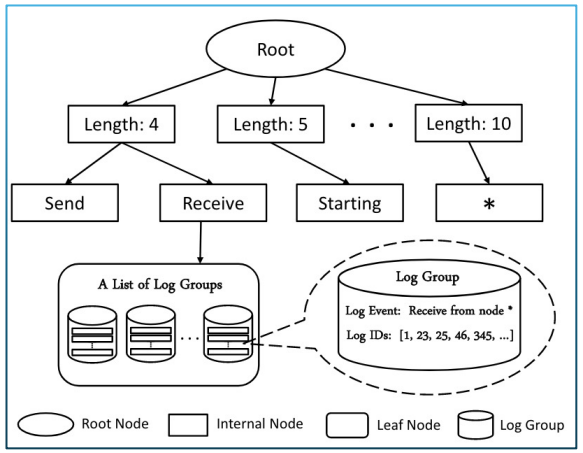
The performance of existing log parsers in practice remains unsatisfatory.

Syntax-based Log Parsers

Utilizing specific features or heuristics,
e.g., log length and word frequency.



heavily rely on manual-crafted rules!



Example: Drain

Limitation of Existing Parsers

Logs are continously evolving!

The performance of existing log parsers in practice remains unsatisfatory.

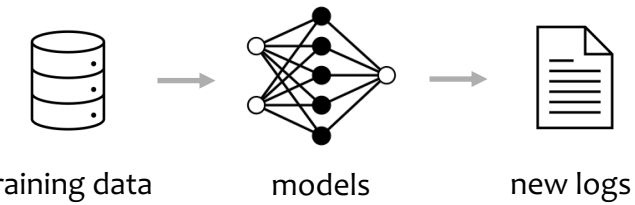
Syntax-based Log Parsers

Utilizing specific features or heuristics, e.g., log length and word frequency.

heavily rely on manual-crafted rules!

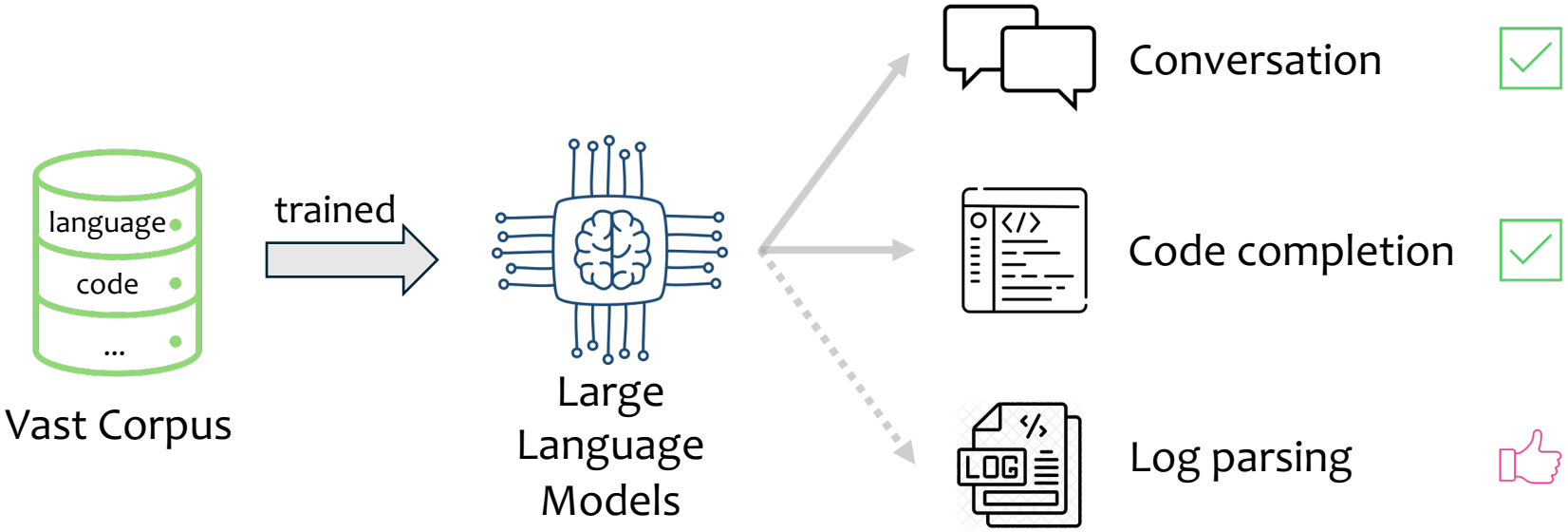
Semantic-based Log Parsers

Utilizing deep learning models to learn semantics and system-specific patterns.

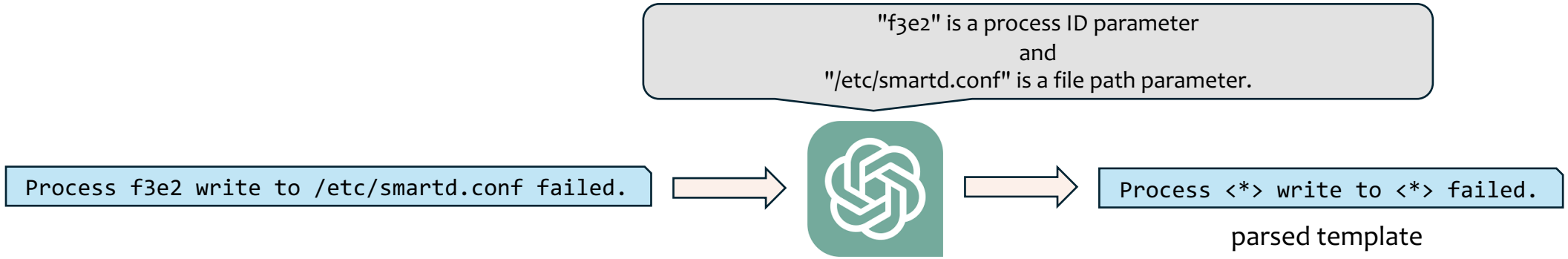


restricted by limited training data!

Opportunity: LLMs



Leveraging the power of LLMs to accurately parse logs is promising!



Challenges

Problem: How could we design a practical LLM-based log parsing framework?

Challenge 1: **Specialization**

- LLMs are not trained for the task of log parsing, resulting in compromised accuracy.

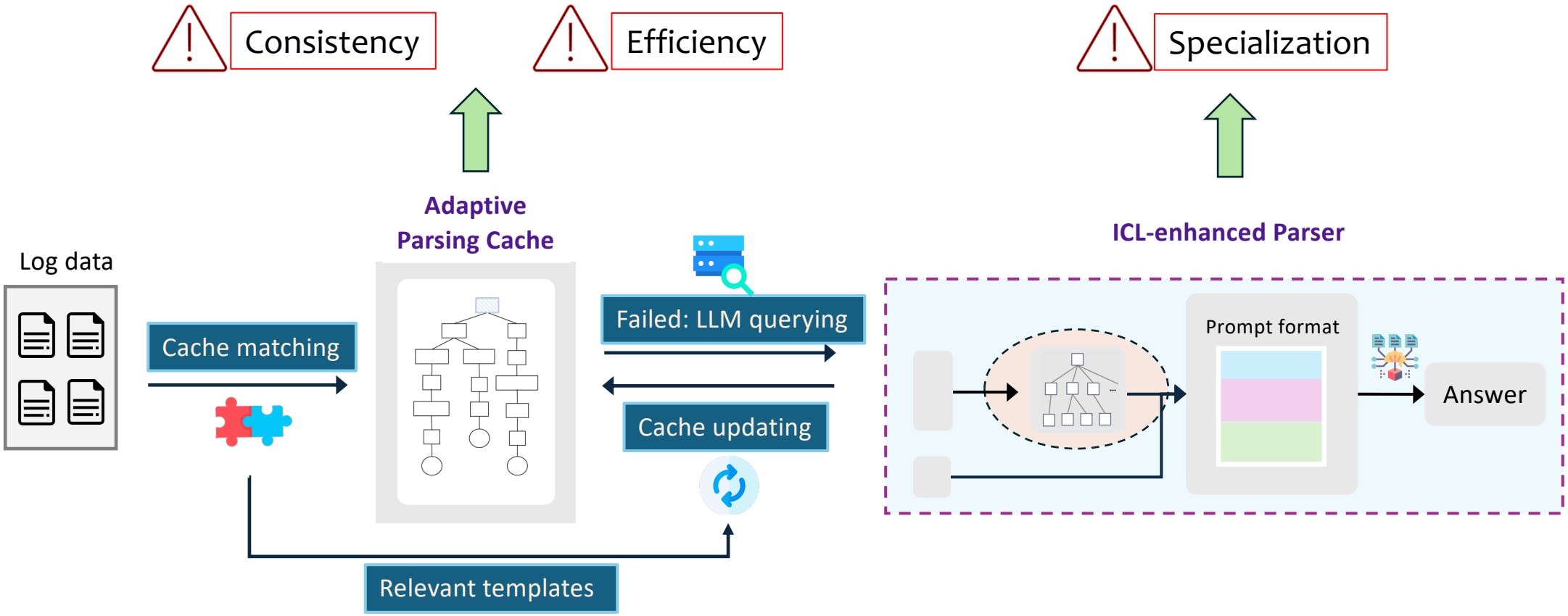
Challenge 2: **Consistency**

- The unstable responses of LLMs could lead to inconsistent log templates.

Challenge 3: **Efficiency**

- The latency and cost of querying LLMs fails to meet the requirements in large-scale scenarios.

Our Approach



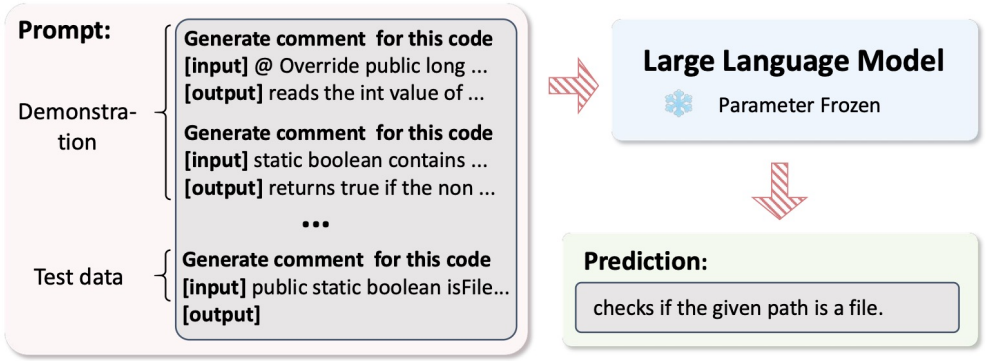
Overall framework of LILAC

ICL-enhanced Parser

Specialization

The **in-context learning (ICL)** paradigm has been proven to be effective in various downstream tasks.

Understanding log parsing task without model tuning.



Prompt format

I want you to act like an expert in log parsing. I will give you a log message wrapped by backticks. Your task is to identify all the dynamic variables in logs, replace them with {variables}, and output a static log template. Please print the input log's template wrapped by backticks.

Here are some examples:
Log message: `Finished spill process 7`
Log template: `Finished spill process {variables}`
Log message: `Task cleanup failed for 3 attempt`
Log template: `Task cleanup failed for {variables} attempt`
Log message: `MapCompletionEvents request from attempt_1895628352891_0043_r_001. startIndex 9 maxEvents 10000`
Log template: `MapCompletionEvents request from {variables}. startIndex {variables} maxEvents {variables}`

Please parse the following log message:
Log message: `Progress of TaskAttempt attempt_1445062781478_0018_m_007 is : 0.3811716 done`

Query LLM

LLM's output

Log template: `Progress of TaskAttempt {variables} is : {variables} done`

Postprocess

Generated template

Progress of TaskAttempt <*> is : <*> done

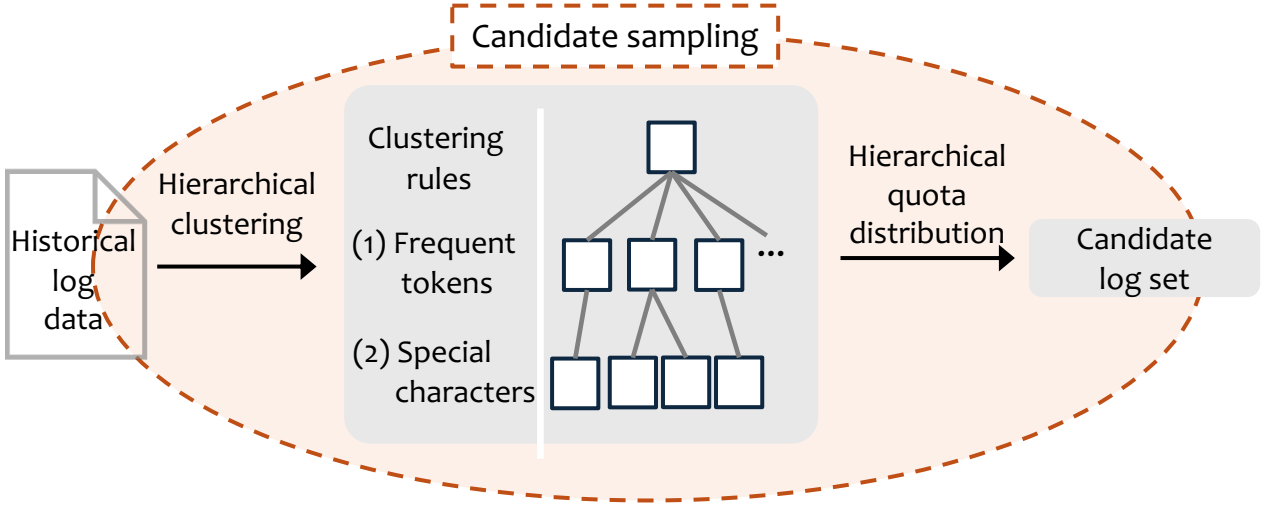
ICL-enhanced Parser

The quality of demonstrations can affect the ICL performance.

Problem: how to select **high-quality** demonstration examples from **substantial** log data?

Our Design

- *Hierarchical candidate sampling* from historical logs
- *Demonstration selection* from candidate log set



Diverse & Representative

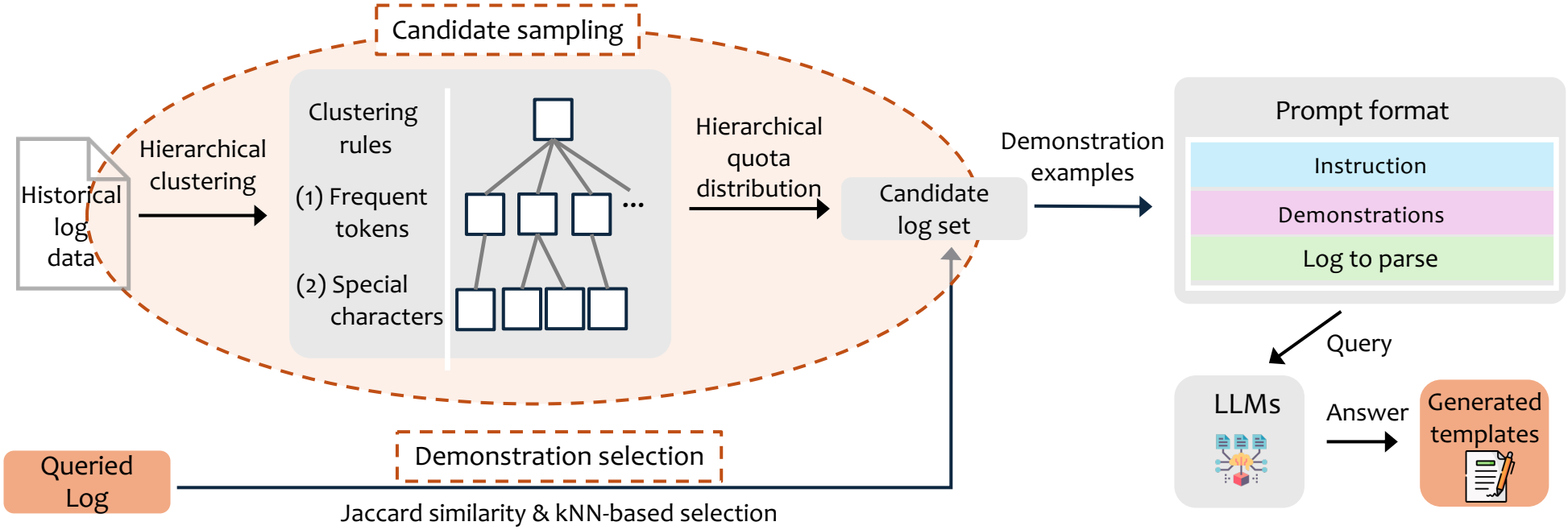
ICL-enhanced Parser

The quality of demonstrations can affect the ICL performance.

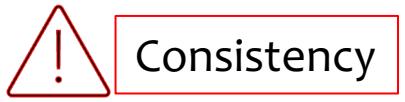
Problem: how to select **high-quality** demonstration examples from **substantial** log data?

Our Design

- *Hierarchical candidate sampling* from historical logs
- *Demonstration selection* from candidate log set



Adaptive Parsing Cache



Our Design

Tree-based adaptive parsing cache to store and refine the generated log templates from the ICL-enhanced Parser

Motivation: the number of *log templates* is significantly smaller than the number of *logs* in real-world systems.

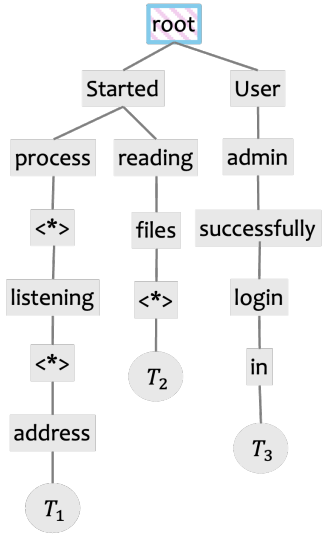
Adaptive Parsing Cache

Consistency

Efficiency

Our Design

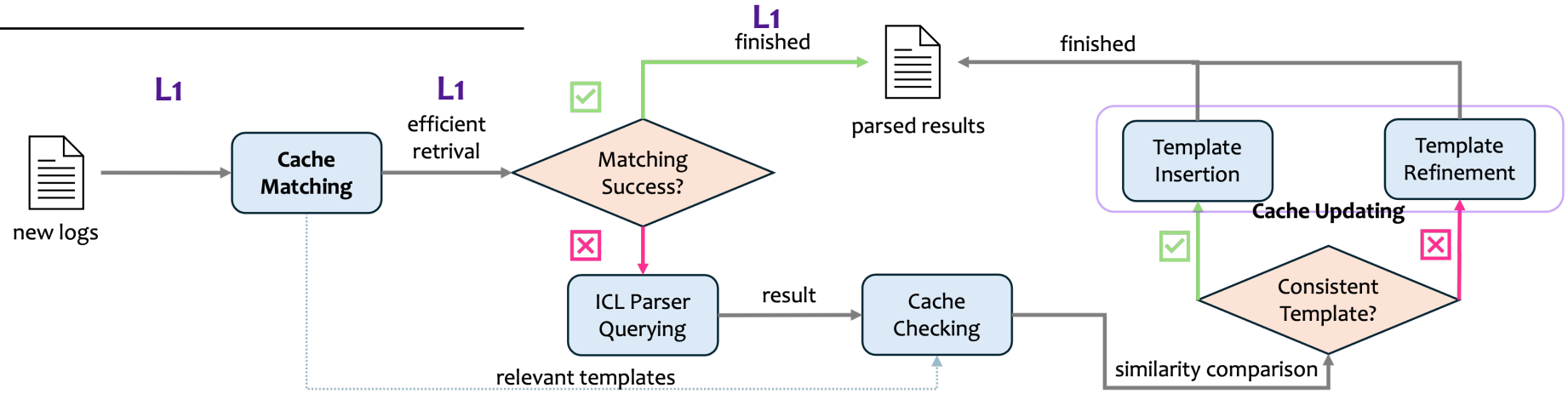
Tree-based adaptive parsing cache to store and refine the generated log templates from the ICL-enhanced Parser



Three Cached Templates:

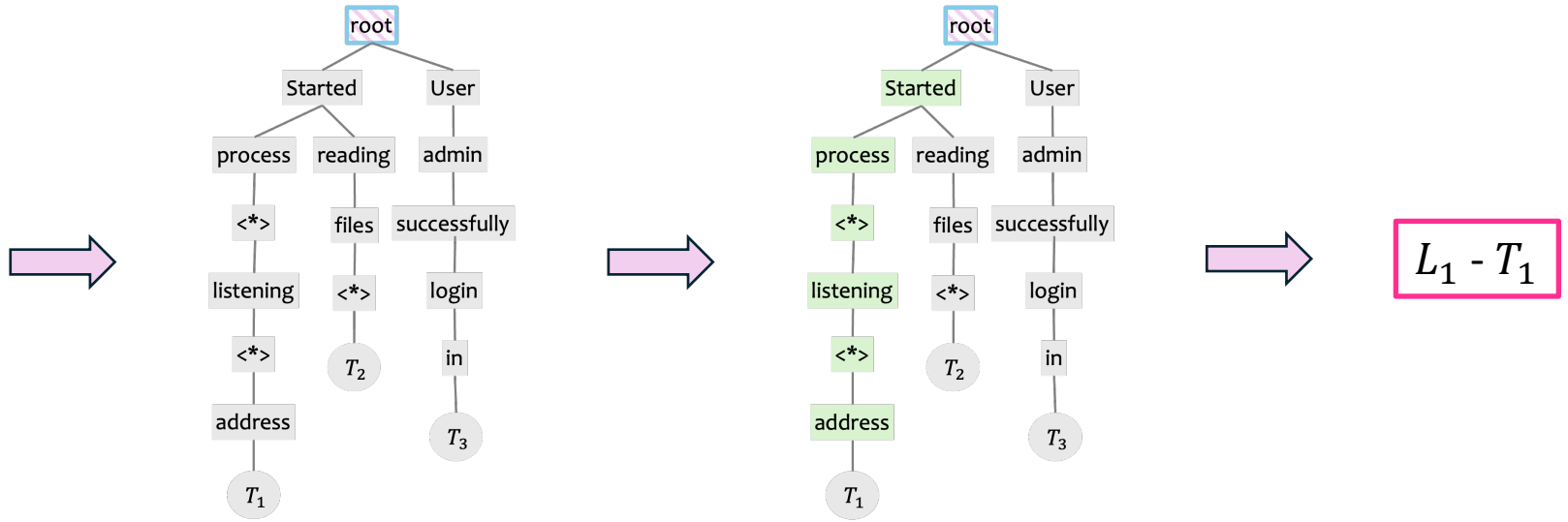
- T_1: Started process <*> listening <*> port
- T_2: Started reading <*> files
- T_3: User admin successfully log in

Adaptive Parsing Cache



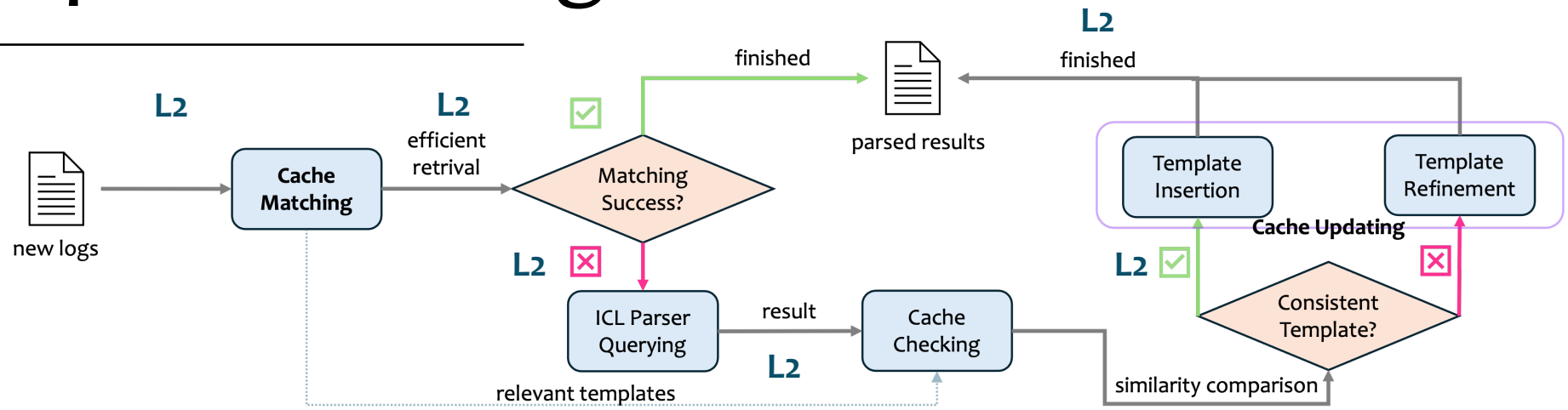
Avoid duplicate LLM queries!

L_1 : **Started process 1616 listening 127.0.0.1 address**



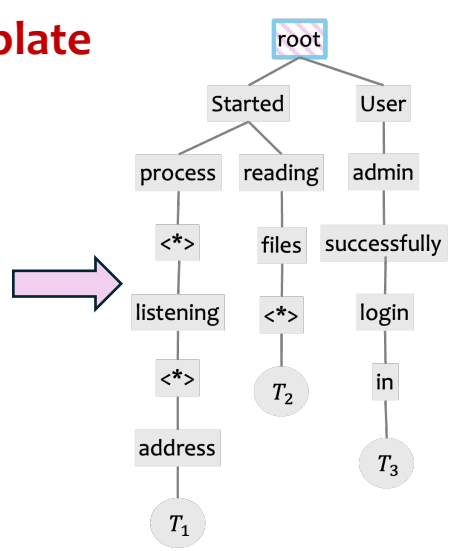
L_1 - T_1

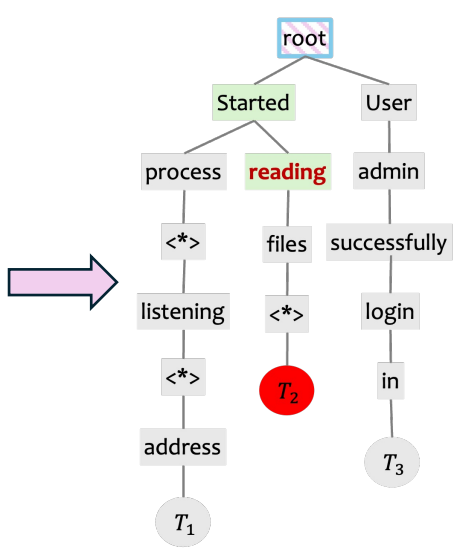
Adaptive Parsing Cache



Store new log template

L_2 : Started reading broadcast variable 1





Relevant Template: T_2

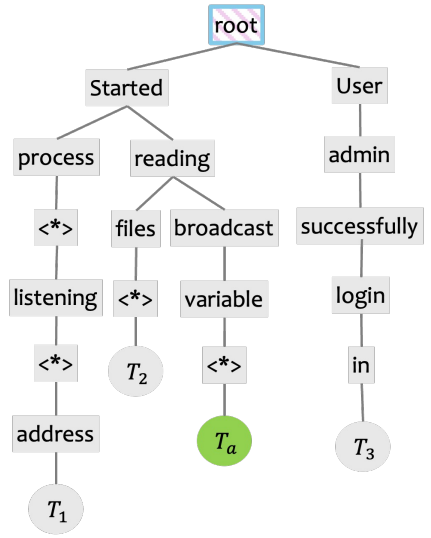
ICL-enhanced Parser

T_a

Compare Similarity

$\text{Sim}(T_a, T_2) < \theta$

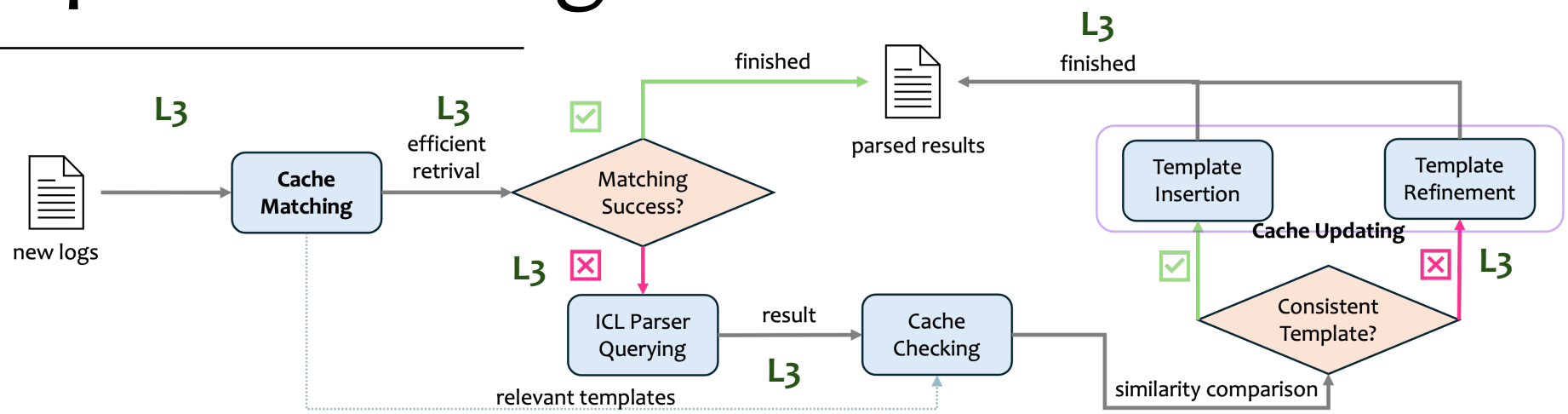
Consistent Template



Template Refinement

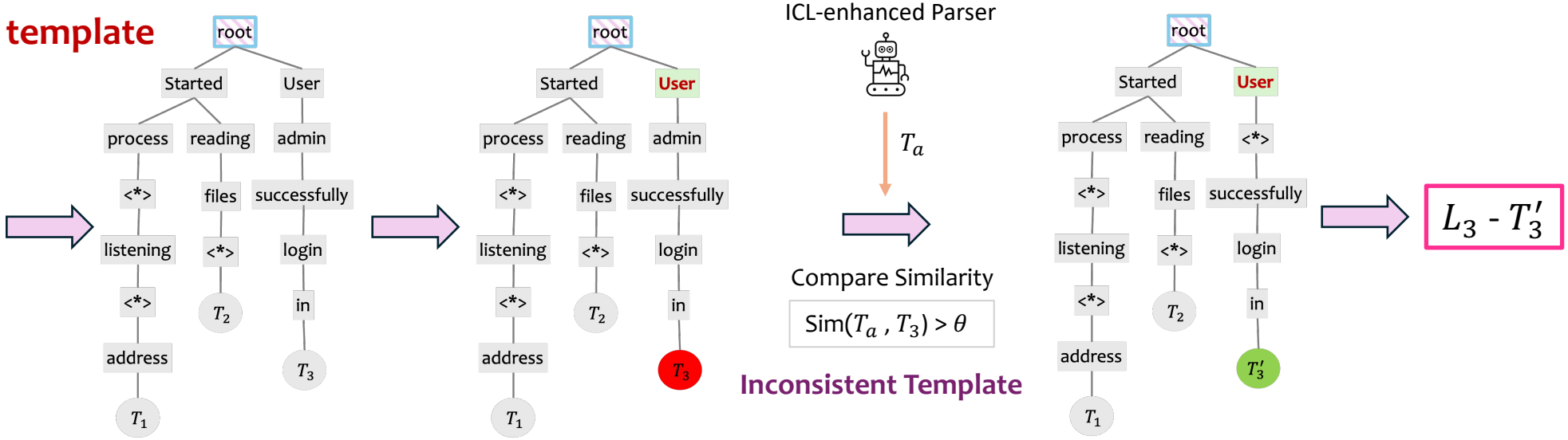
$L_2 - T_a$

Adaptive Parsing Cache



Refine inconsistent template

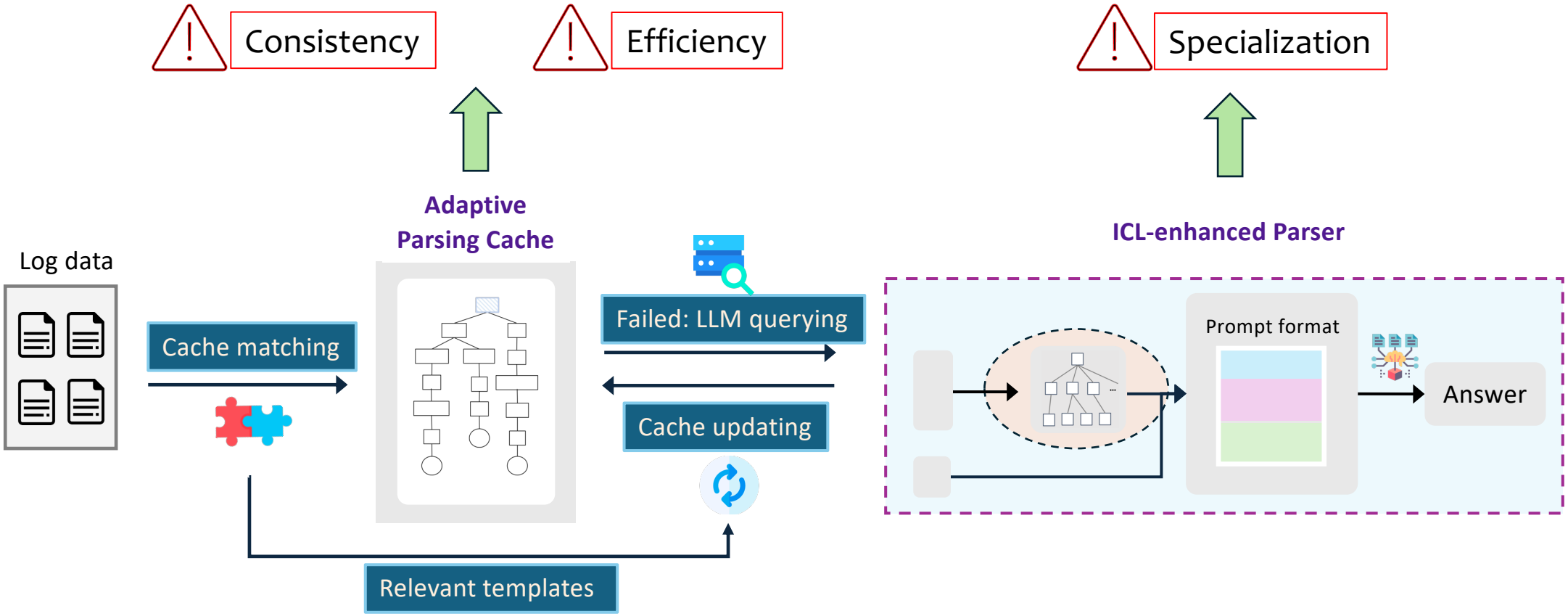
L_3 : **User user1 successfully login in**



Relevant Template: T_3

Template Refinement for T_3

Our Approach



The collaboration between the two components resulted in effective and efficient log parsing.

Evaluation

Research Questions

- RQ1: What is the **effectiveness** of LILAC?
- RQ2: What is the **contribution of each design**?
- RQ3: What are the **capabilities with different LLMs**?
- RQ4: What is the **efficiency** of LILAC?

Datasets

Loghub-2.0

System	Dataset	# Log Templates	# Log Messages
Distributed systems	Hadoop	236	179,993
	HDFS	46	11,167,740
	OpenStack	48	207,632
	Spark	236	16,075,117
	Zookeeper	89	74,273
Super-computer systems	BGL	320	4,631,261
	HPC	74	429,987
	Thunderbird	1,241	16,601,745
Operating systems	Linux	338	23,921
	Mac	626	100,314
Server application	Apache	29	51,977
	OpenSSH	38	638,946
Standalone software	HealthApp	156	212,394
	Proxifier	11	21,320
Average		**249.1**	**3,601,187**

Settings

- LLM: gpt-3.5-turbo-0613
- Historical data rate: 20%
- Candidate samples: 32
- Demonstration examples: 3

State-of-the-art Baselines

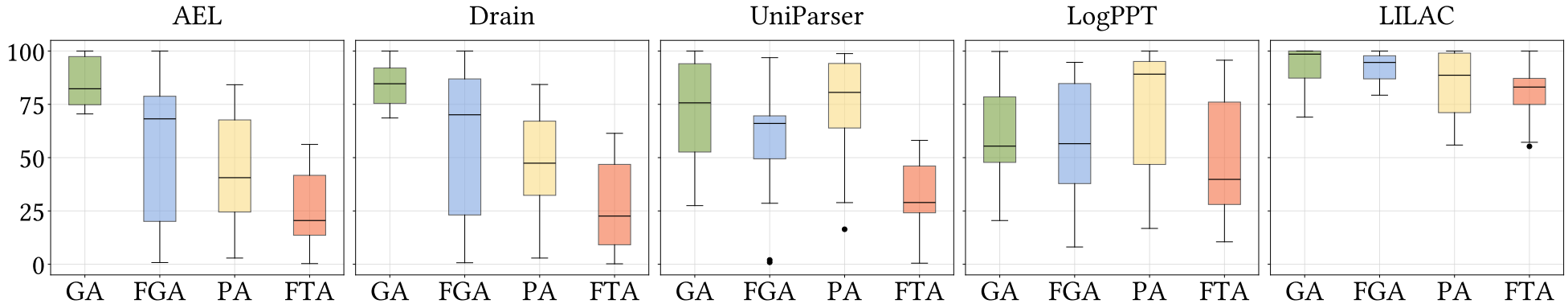
- AEL (*syntax-based*)
- Drain (*syntax-based*)
- UniParser (*semantic-based*)
- LogPPT (*semantic-based*)

Metrics

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

Evaluation

- **RQ1: Effectiveness**



- **RQ2: Ablation**

	GA	FGA	PA	FTA
LILAC	92.7	92.4	84.2	81.0
w/o ICL	83.5 (↓ 9.9%)	76.5 (↓ 17.2%)	62.6 (↓ 25.6%)	58.4 (↓ 27.9%)
w/ random selection	84.2 (↓ 9.2%)	80.6 (↓ 14.6%)	74.4 (↓ 11.6%)	66.7 (↓ 17.7%)
w/ random sampling	87.6 (↓ 5.5%)	80.9 (↓ 12.4%)	77.5 (↓ 8.0%)	68.3 (↓ 15.7%)
w/ LogPPT sampling	91.3 (↓ 1.5%)	84.8 (↓ 8.2%)	79.7 (↓ 5.3%)	74.9 (↓ 7.5%)

	GA	FGA	PA	FTA
LogPPT (original)	61.2	57.4	73.6	47.8
w/ parsing cache	90.8 (↑ 48.4%)	89.0 (↑ 55.1%)	78.0 (↑ 6.0%)	67.4 (↑ 41.0%)

- LILAC **outperforms all other state-of-the-art log parsers.**

- All designs of *ICL-enhanced parsers* and *adaptive parsing* cache contribute to the overall performance.

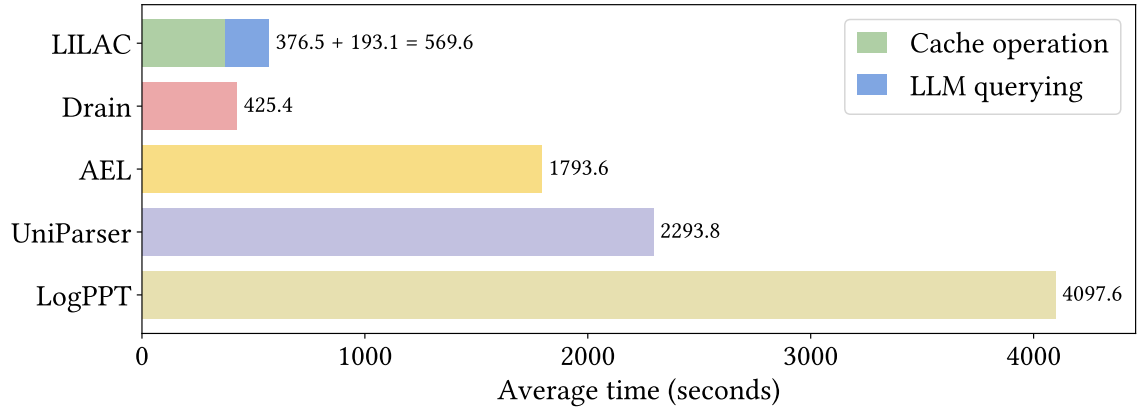
Evaluation

- **RQ3: Different LLMs**

	GA	FGA	PA	FTA
ChatGPT (175B)	92.7	92.4	84.2	81.0
Davinci (175B)	91.9 (↓ 0.9%)	92.9 (↑ 0.5%)	87.1 (↑ 3.4%)	81.5 (↑ 0.6%)
Curie (13B)	90.1 (↓ 2.8%)	87.6 (↓ 5.2%)	77.8 (↓ 7.6%)	71.2 (↓ 12.1%)

- LILAC consistently achieves high performance with different LLMs.
- The performance of LILAC can be influenced by the capabilities of LLMs.

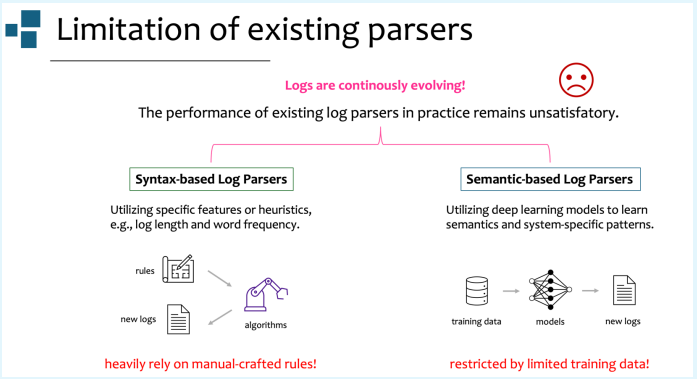
- **RQ4: Efficiency**



- LILAC successfully reduces millions of LLM queries to an average of 279.7.
- LILAC surpasses semantic-based methods and is comparable to the fastest syntax-based method in efficiency.

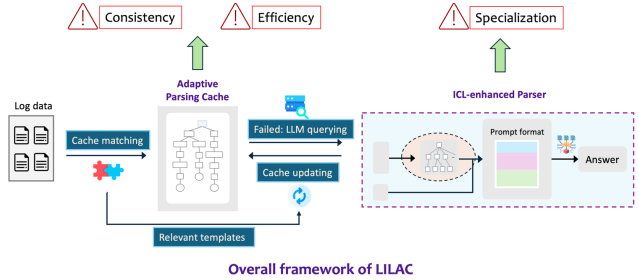
Conclusion

Limitation of existing parsers



Motivation

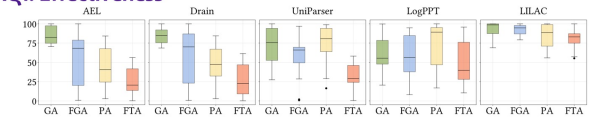
Our Approach



Approach

Evaluation

- **RQ1: Effectiveness**



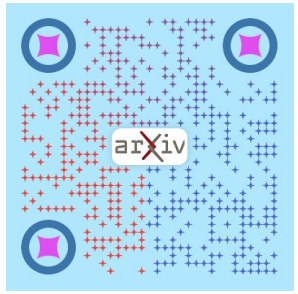
- **RQ2: Ablation**

	GA	FGA	PA	FTA
LILAC	92.7	92.4	84.2	81.0
w/o ICL	83.5 (↓ 9.9%)	76.5 (↓ 17.2%)	62.6 (↓ 25.6%)	58.4 (↓ 27.9%)
w/ random selection	84.2 (↓ 9.2%)	80.6 (↓ 14.6%)	74.4 (↓ 11.6%)	66.7 (↓ 17.7%)
w/ random sampling	87.6 (↓ 5.5%)	80.9 (↓ 12.4%)	77.5 (↓ 8.0%)	68.3 (↓ 15.7%)
w/ LogPPT sampling	91.3 (↓ 1.5%)	84.8 (↓ 8.2%)	79.7 (↓ 5.3%)	74.9 (↓ 7.5%)

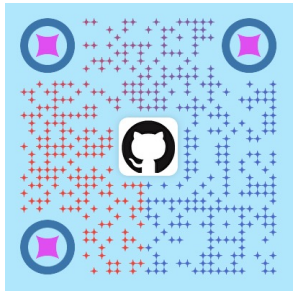
	GA	FGA	PA	FTA
LogPPT (original)	61.2	57.4	73.6	47.8
w/ parsing cache	90.8 (↑ 48.4%)	89.0 (↑ 55.1%)	78.0 (↑ 6.0%)	67.4 (↑ 41.0%)

- LILAC **outperforms all other state-of-the-art log parsers.**
- All designs of *ICL-enhanced parsers* and *adaptive parsing* cache contribute to the overall performance.

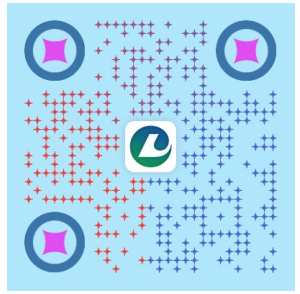
Experiment



Pre-print paper



Artifact



LogPAI website

Thank you for listening

Q & A

Contact: zhjiang@link.cuhk.edu.hk