LOG.title("

Log-driven Automated Software Reliability Engineering

")

Yintong Huo

Ph.D. Oral Defense

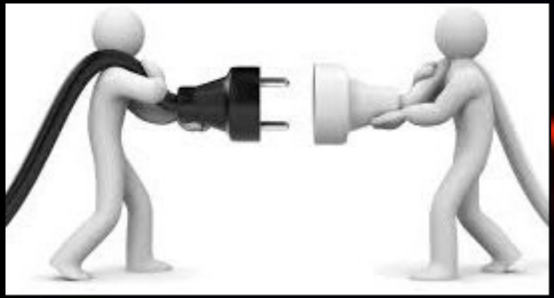
Supervisor: Prof. Michael R. Lyu

3 July, 2024

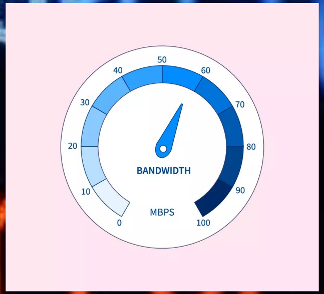
香港中文大學
The Chinese University of Hong Kong

Introduction

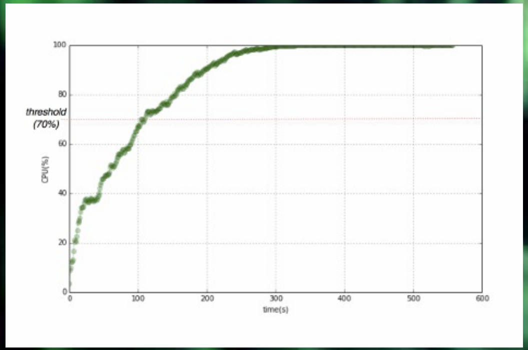
Software failure can happen…



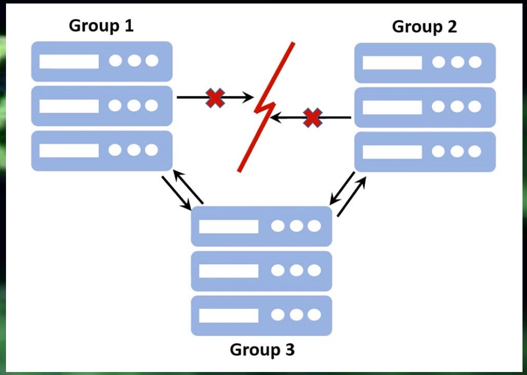
Power Outage



Network Bandwidth Limited



CPU Saturation/ Memory Saturation



Network Partition

More and More ……

Real-world revenue loss

The 15 Biggest Cloud Outages Of 2023

BY WADE TYLER MILLWARD

DECEMBER 13, 2023, 3:37 PM EST

THE CHANNEL CO. CRN

In fact, service outages have become so commonplace and preparation so essential that

Software Reliability Engineering (SRE) is in urgent need.

[RELATED: The 10 Hottest Networking Products Of 2023]

A report this year from Parametrix Insurance ...ded that a 24-hour outage of mission-critical services from AWS us-east-1 – the ... with the largest number of Fortune 500 companies relying on it – could cost **3.4 billion** ...irect revenue. A 48-hour outage could cost $7.8 billion.

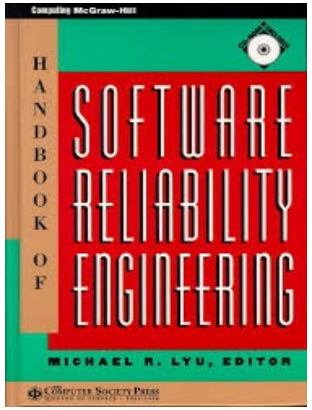
A 24-hour loss of east-1 and west-2 AWS services could cost $8.2 billion, $17.5 billion if lost for 48 hours, according to the report.

Hard to mitigate

Reputation

24h-hour outage:
Loss $3.4 billion

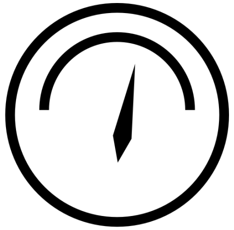
Automated SRE (data-driven)



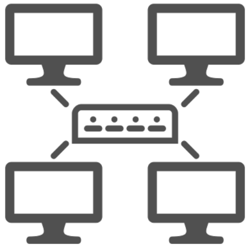
Code

Open Forum

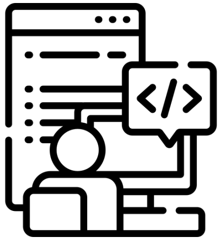
Log



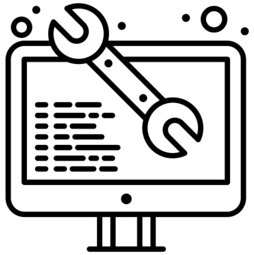
Meter Data



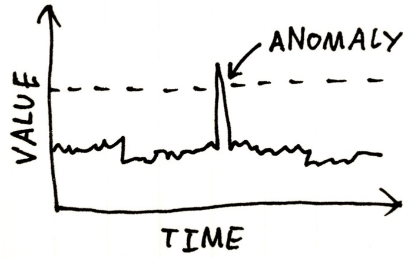
Topology



Code Production



Code Repair

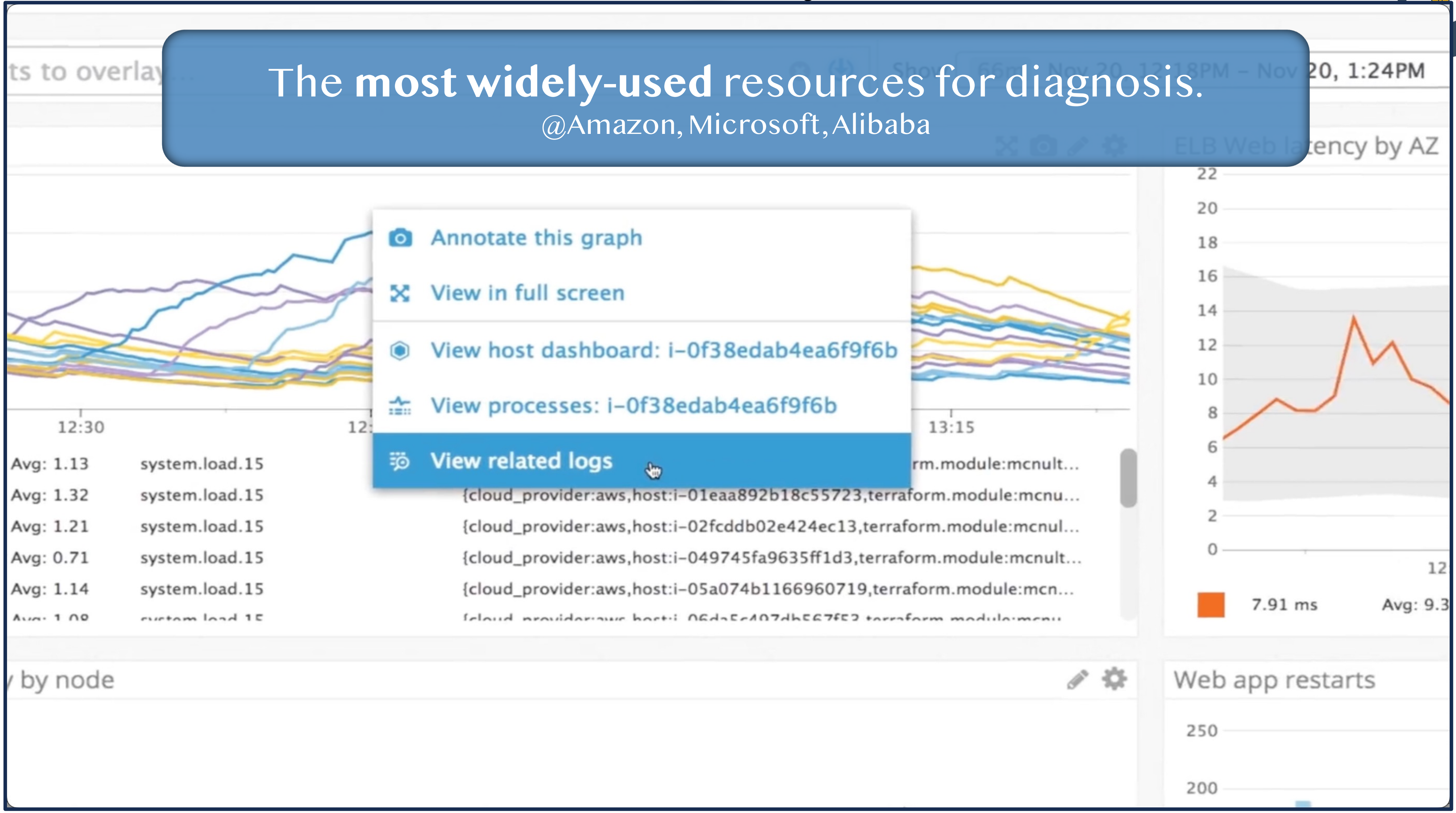


Software Operations

How do we monitor run-time systems?

The **most widely-used** resources for diagnosis.

@Amazon, Microsoft, Alibaba



Log-driven automated SRE: What are logs?

The lifecycle of logs

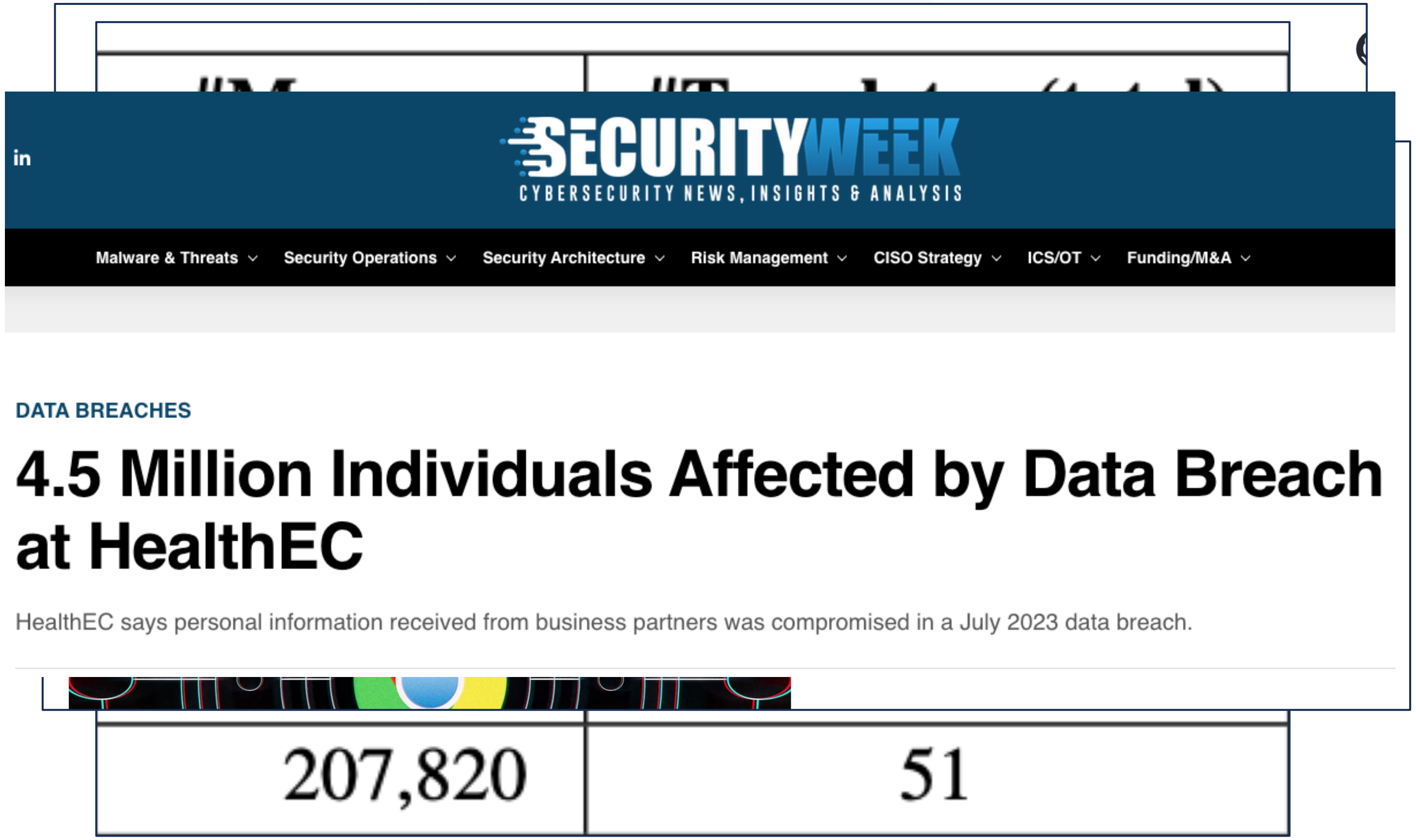
Logging statements

Log files

```
# Logging statements from Spark
(spark/storage/BlockManager.scala)
logError(s"Failed to report ${blockId} to master; giving up.")
logDebug(s"Putting block ${blockId} with replication took
${usedTimeMs}")
logInfo(s"Writing block ${blockId} to disk")
```

17/08/22 15:50:46 ERROR BlockManager **Failed to report** rdd_0_1 **to master; giving up**.
17/08/22 15:51:18 DEBUG BlockManager **Putting block** rdd_1_1 **with replication took** 0
17/08/22 15:51:55 INFO BlockManager **Writing block** rdd_1_1 **to disk**
…

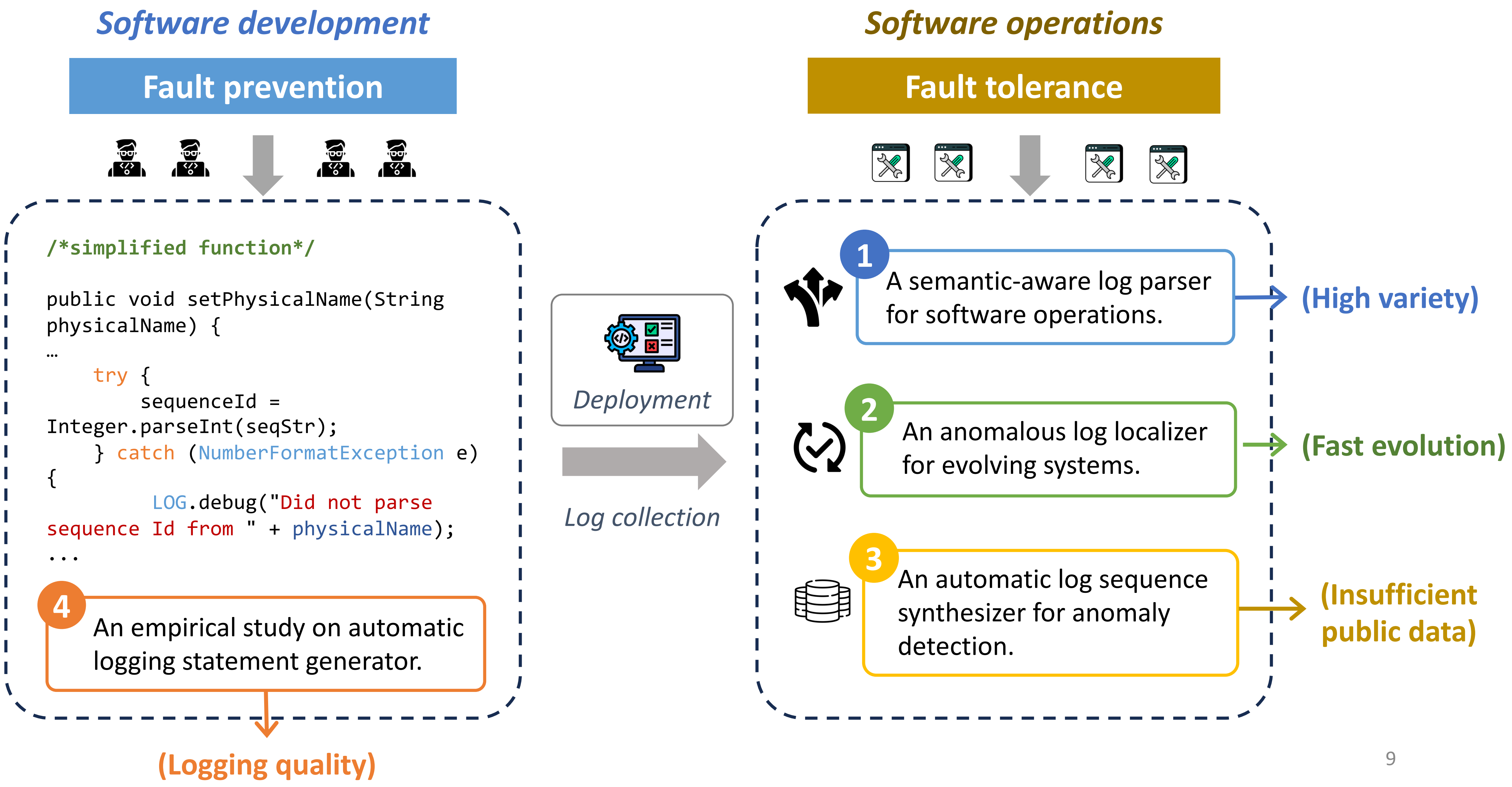
Log-driven automated SRE: Challenges



How to automate log analysis to monitor systems?

- Big volume
- High variety
  - Semi-structured language
- Fast evolution
  - Evolving log events
- Lacking open dataset

Contributions



Outline

Log-driven automated SRE

Reactive Log analysis

[ICSE'23, ASE'23, ISSRE'23, ICSE'24, FSE'24, ISSTA'24]

- Topic1: A semantic-aware log parser for software operations — [ICSE'23] (Chapter 3)
- Topic2: A log localizer for evolving software systems — [ISSRE'23] (Chapter 4)
- Topic3: An automatic log sequence synthesizer for anomaly detection — [ASE'23] (Chapter 5)

Proactive Logging practice

[FSE'24, TSE'24 Major]

- Topic4: An empirical study on automatic logging statement generator — [TSE'24 Major] (Chapter 6)

- Conclusion and Future Work

Outline

Log-driven automated SRE

- **Reactive Log analysis** [ICSE'23, ASE'23, ISSRE'23, ICSE'24, FSE'24, ISSTA'24]
  - Topic1: A semantic-aware log parser for software operations [ICSE'23] (Chapter 3)
  - Topic2: A log localizer for evolving software systems [ISSRE'23] (Chapter 4)
  - Topic3: An automatic log sequence synthesizer for anomaly detection [ASE'23] (Chapter 5)
- **Proactive Logging practice** [FSE'24, TSE'24 Major]
  - Topic4: An empirical study on automatic logging statement generator [TSE'24 Major] (Chapter 6)

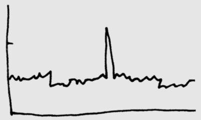
- Conclusion and Future Work

Preliminary: The workflow of log analysis

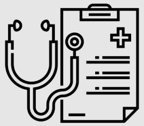
Log messages

Raw Log Messages	
1	**2008-11-11 03:40:58** BLOCK* NameSystem.allocateBlock: /user /root/randtxt4/_temporary/_task_200811101024_0010_m_000011_0/part-00011.blk_904791815409399662
2	**2008-11-11 03:40:59** Receiving block blk_904791815409399662 src: /10.251.43.210:55700 dest: /10.251.43.210:50010
3	**2008-11-11 03:41:01** Receiving block blk_904791815409399662 src: /10.250.18.114:52231 dest: /10.250.18.114:50010
4	**2008-11-11 03:41:48** PacketResponder 0 for block blk_904791815409399662 terminating
5	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.250.18.114
6	**2008-11-11 03:41:48** PacketResponder 1 for block blk_904791815409399662 terminating
7	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.251.43.210
8	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.210:50010 is added to blk_904791815409399662 size 67108864
9	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.18.114:50010 is added to blk_904791815409399662 size 67108864
10	**2008-11-11 08:30:54** Verification succeeded for blk_904791815409399662

Downstream applications

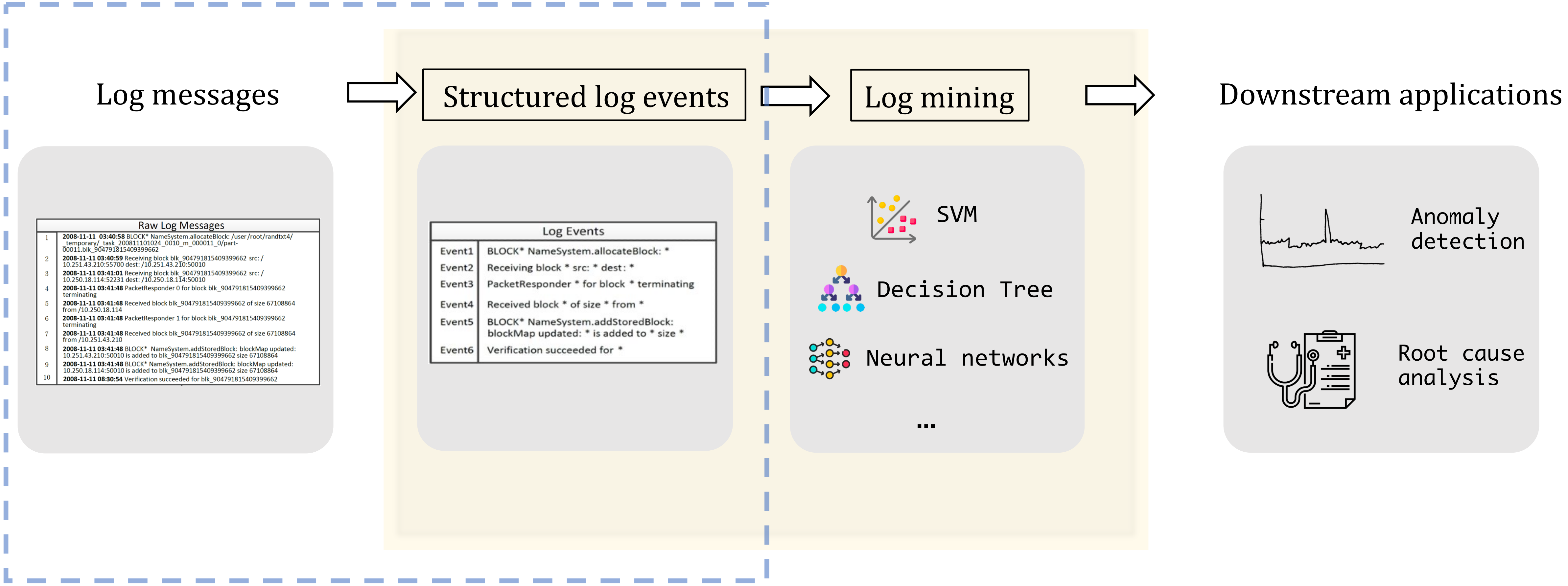


Anomaly detection



Root cause analysis

Preliminary: The workflow of log analysis



Raw Log Messages	
1	**2008-11-11 03:40:58** BLOCK* NameSystem.allocateBlock: /user/root/randtxt4/_temporary/_task_200811101024_0010_m_000011_0/part-00011.blk_904791815409399662
2	**2008-11-11 03:40:59** Receiving block blk_904791815409399662 src: /10.251.43.210:55700 dest: /10.251.43.210:50010
3	**2008-11-11 03:41:01** Receiving block blk_904791815409399662 src: /10.250.18.114:52231 dest: /10.250.18.114:50010
4	**2008-11-11 03:41:48** PacketResponder 0 for block blk_904791815409399662 terminating
5	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.250.18.114
6	**2008-11-11 03:41:48** PacketResponder 1 for block blk_904791815409399662 terminating
7	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.251.43.210
8	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.210:50010 is added to blk_904791815409399662 size 67108864
9	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.18.114:50010 is added to blk_904791815409399662 size 67108864
10	**2008-11-11 08:30:54** Verification succeeded for blk_904791815409399662

Log Events	
Event1	BLOCK* NameSystem.allocateBlock: *
Event2	Receiving block * src: * dest: *
Event3	PacketResponder * for block * terminating
Event4	Received block * of size * from *
Event5	BLOCK* NameSystem.addStoredBlock: blockMap updated: * is added to * size *
Event6	Verification succeeded for *

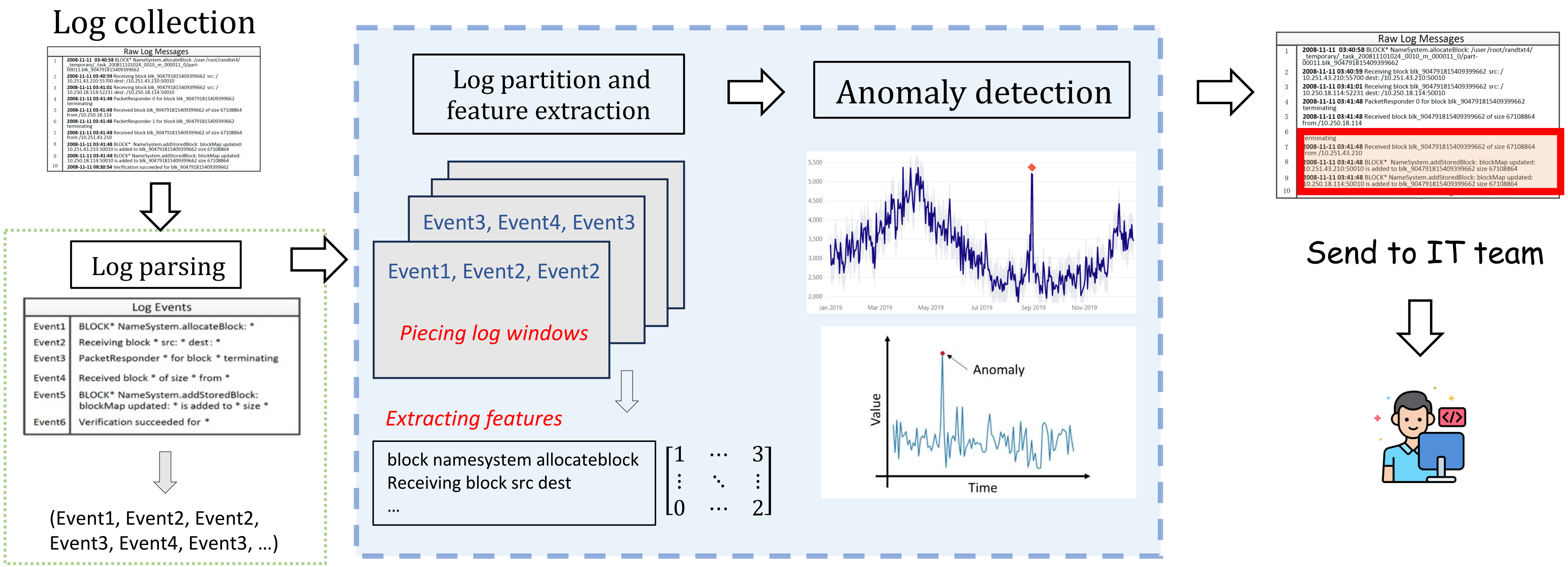
Preliminary: Log-based anomaly detection

- The *most widely-studied* task in log analysis
- Purpose: Detect if a system has run-time anomalies in a period of time via analyzing log files
  - ✓ Network error, CPU saturation, power outage etc..

Unexpected behaviors

Preliminary: Log-based anomaly detection

- The *most widely-studied* task in log analysis
- Purpose: Detect if a system has run-time anomalies in a period of time



Existing log parser

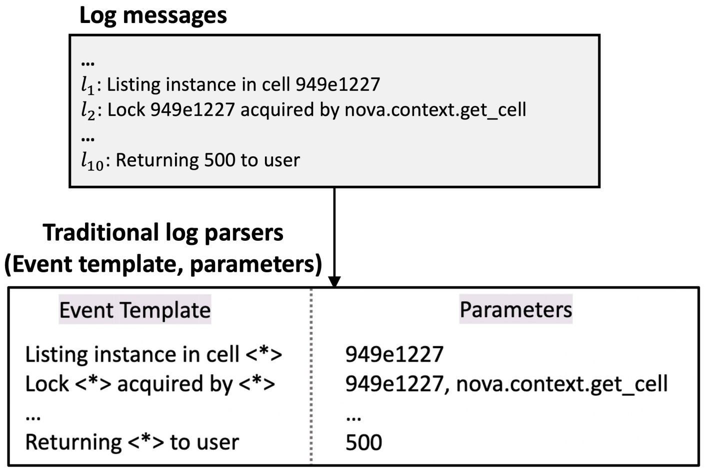
Log messages

...
l_1: Listing instance in cell 949e1227
l_2: Lock 949e1227 acquired by nova.context.get_cell
...
l_{10}: Returning 500 to user

Traditional log parsers
(Event template, parameters)

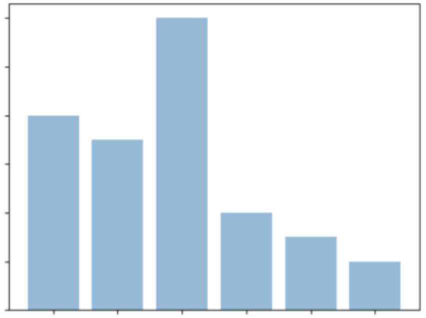
Event Template	Parameters
Listing instance in cell <*>	949e1227
Lock <*> acquired by <*>	949e1227, nova.context.get_cell
...	...
Returning <*> to user	500

Existing log parsers



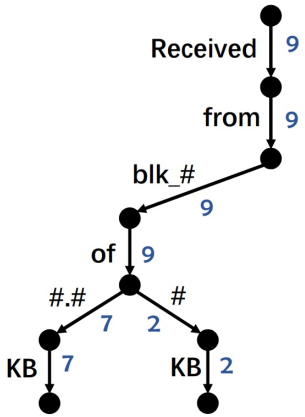
Statistical-based

- SLCT: word frequency.
- Logram: n-gram patterns.
- SHISO: word length.



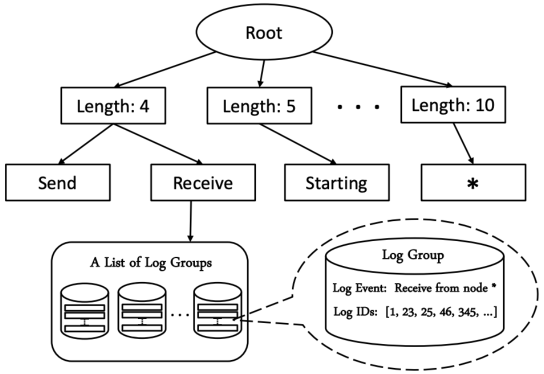
Graph-based

- Prefix-graph: probabilistic graph.



Tree-based

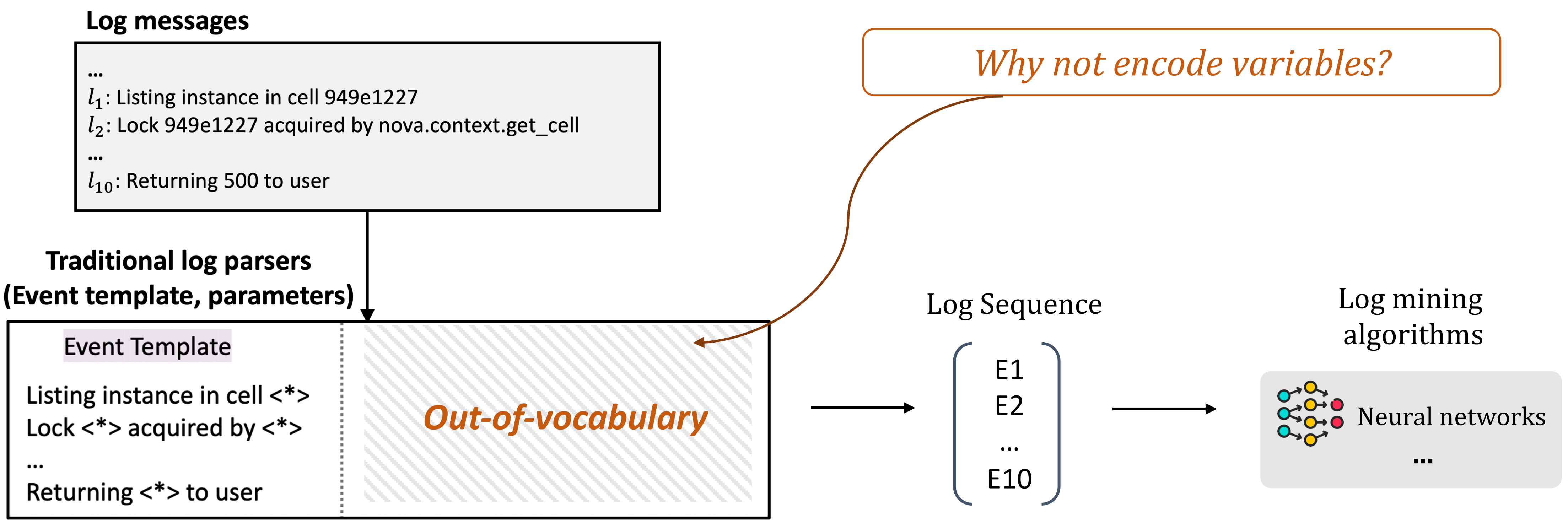
- Drain: traverse leaf nodes in a tree structure.



Current parsers are working with syntax-based superficial features, which cannot capture semantics.

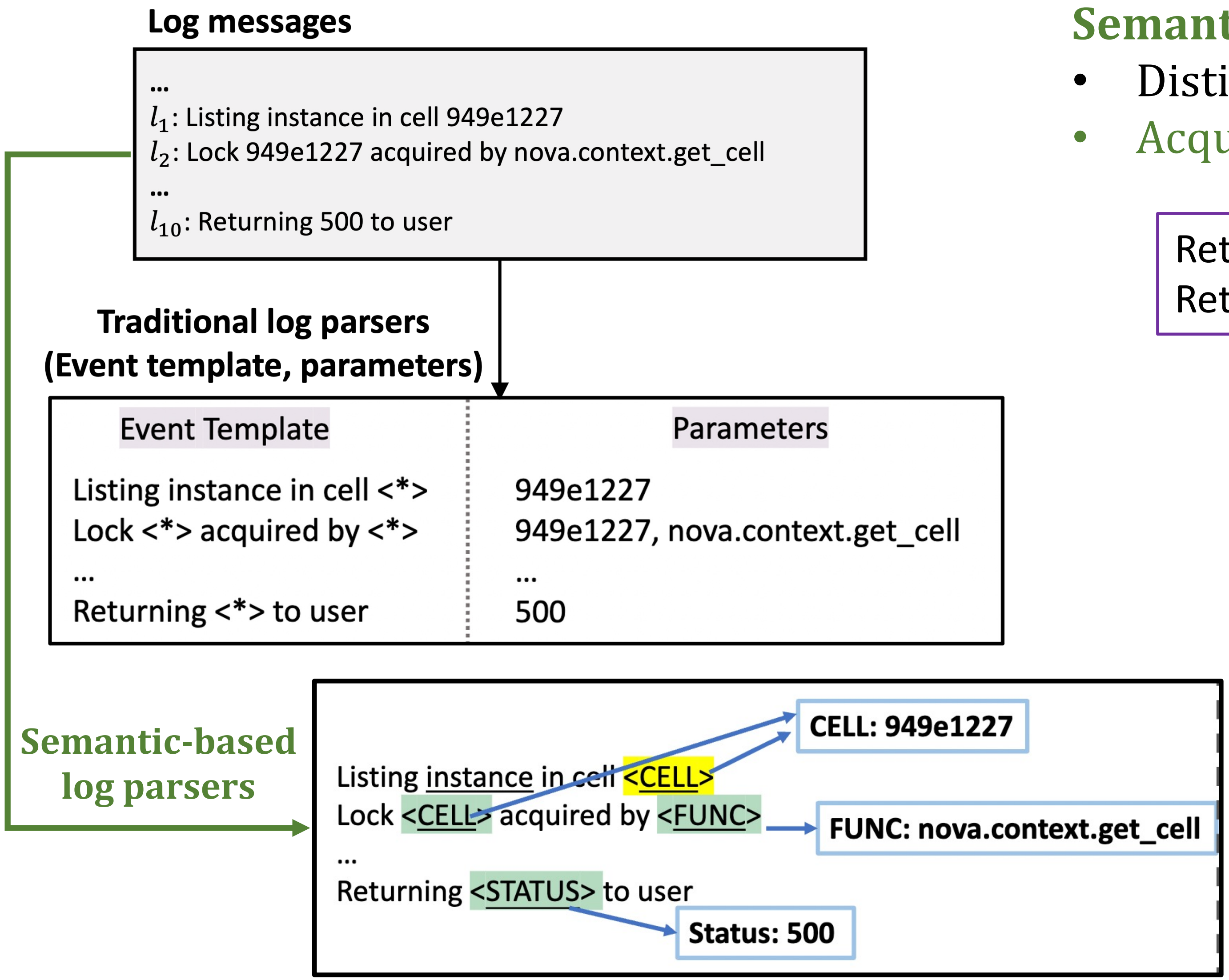
[IPOM'03] SLCT
[TSE'20] Logram
[SCC'13] SHISO
[ICDE'22] Prefix-graph
[ICWS'17] Drain

Existing log parsers



Insufficient semantic information obstructs subsequent analysis.

Motivation: We care semantics



Semantic Parser

- Distinguish event templates and parameters.
- Acquire semantics of parameters (variables).

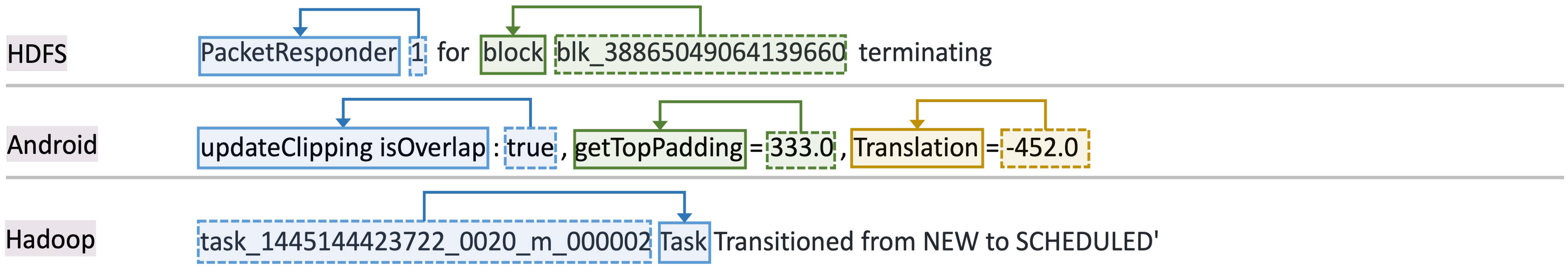
Returning 200 to user	Connection OK
Returning 500 to user	Connection timed out!

Semantics of variables:

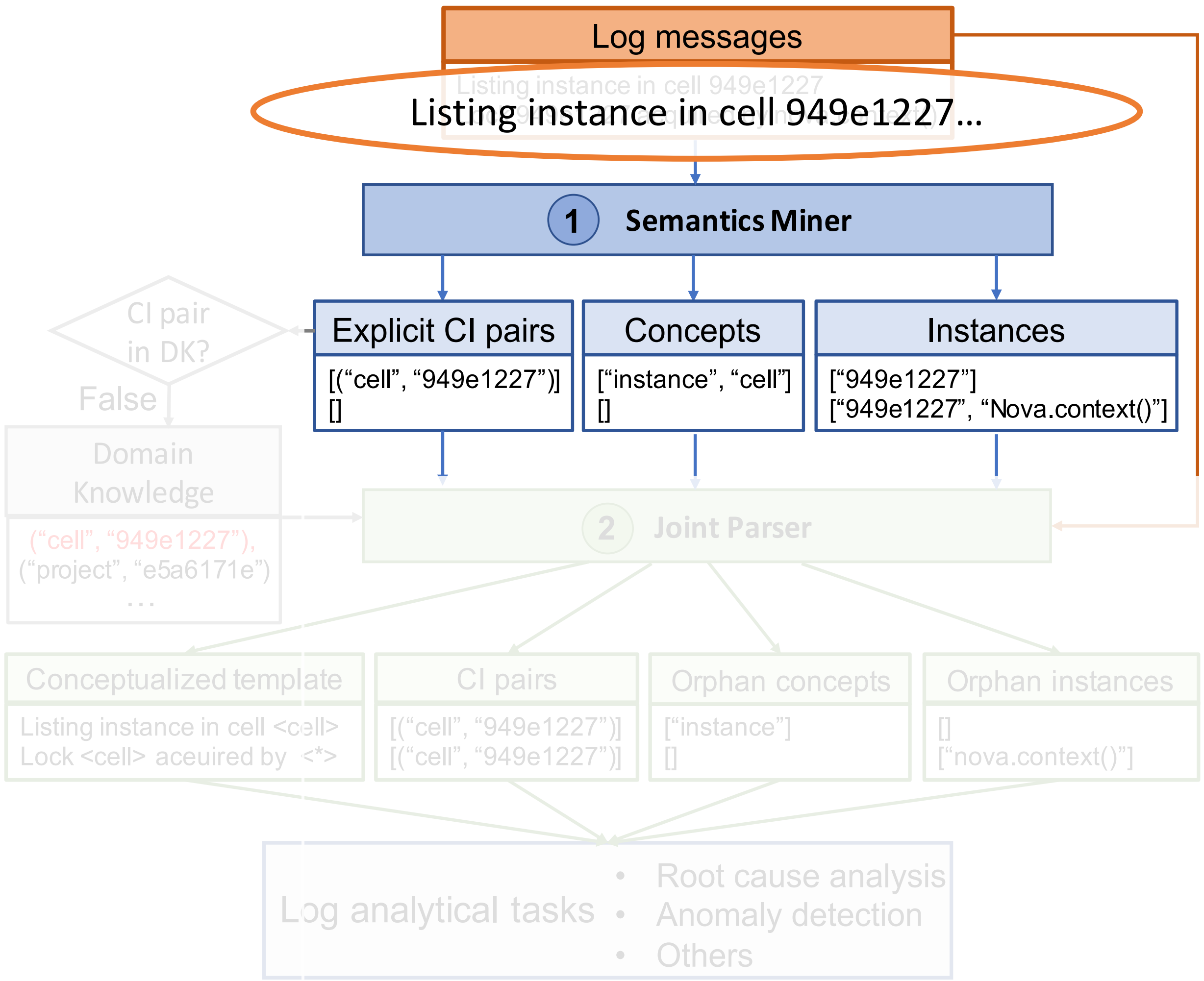
- What does 949e1227 mean?
  - Cell ID

Preliminary

- Terminologies
  - Semantic roles
    - Concepts: Technical terms in the log message (e.g., block).
    - Instances: Variables in the log message (e.g., blk_38865049064139660).
  - Variable semantics
    - Concept-Instance pairs (*CI pairs*), *describing the concept that the instance refers to*.



SemParser



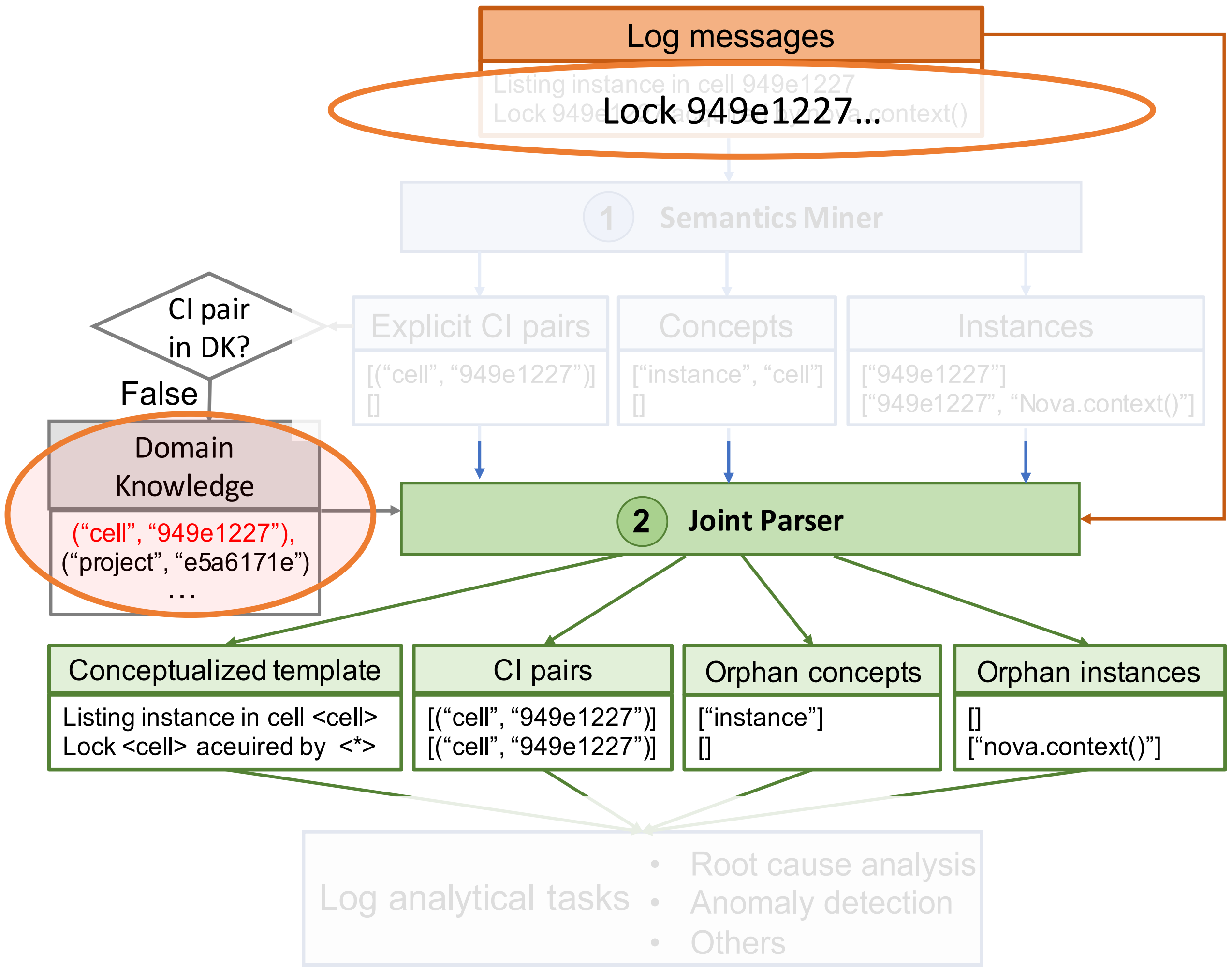
Overview (2 steps)

- Input: Log messages

> *Observation#1:*
> *CI pairs can appear in the same log.*

- Step1: Semantics Miner
  - Mine *explicit* variable semantics within single log message.

SemParser

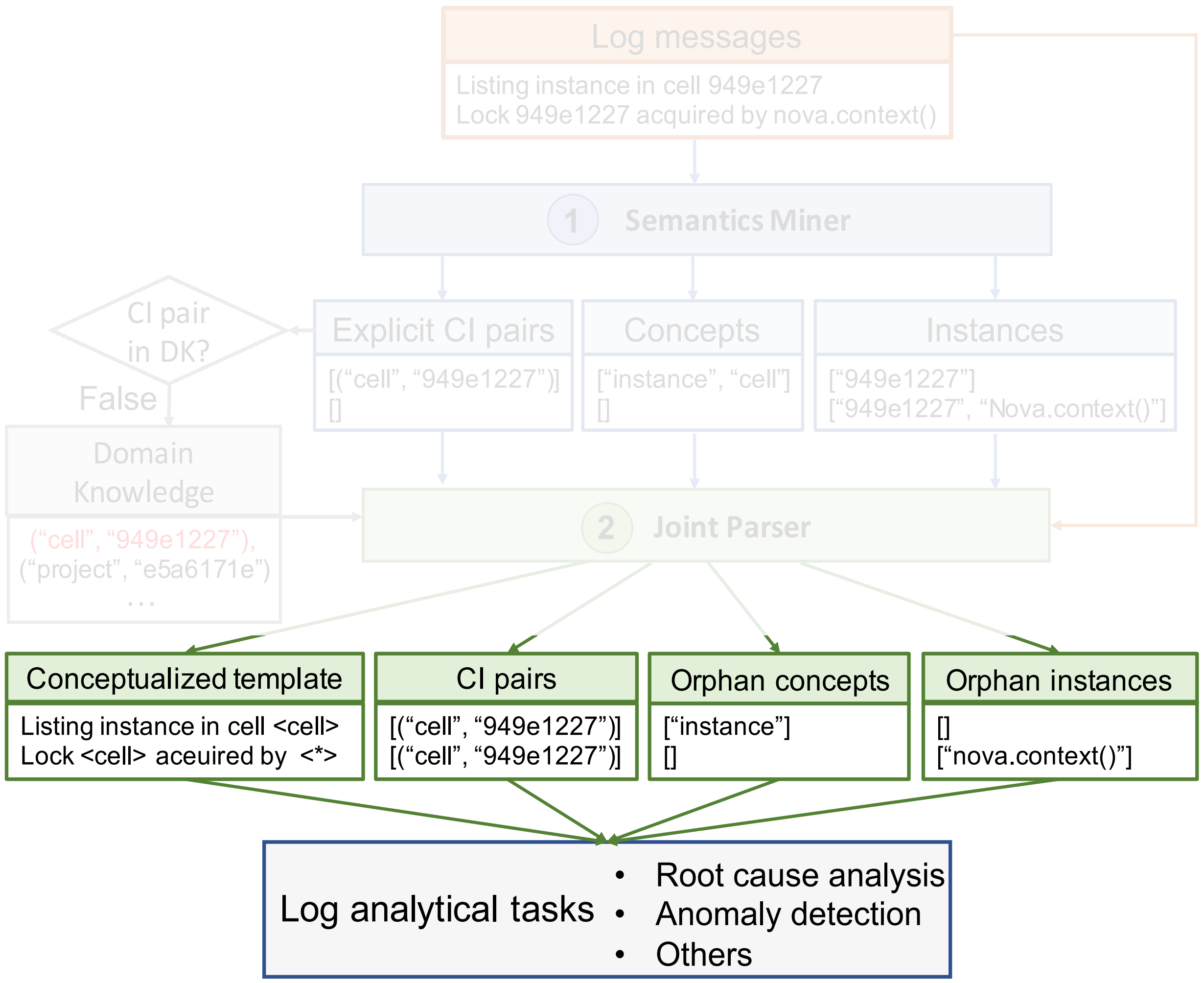


Overview (2 steps)

- Input: Log messages
- Step1: Semantics Miner
  - Mine *explicit* variable semantics within single log message.
- Step2: Joint Parser
  - Conduct *implicit* variable semantics inference across log messages.

Observation#2:
CI pairs can also occur in different logs.

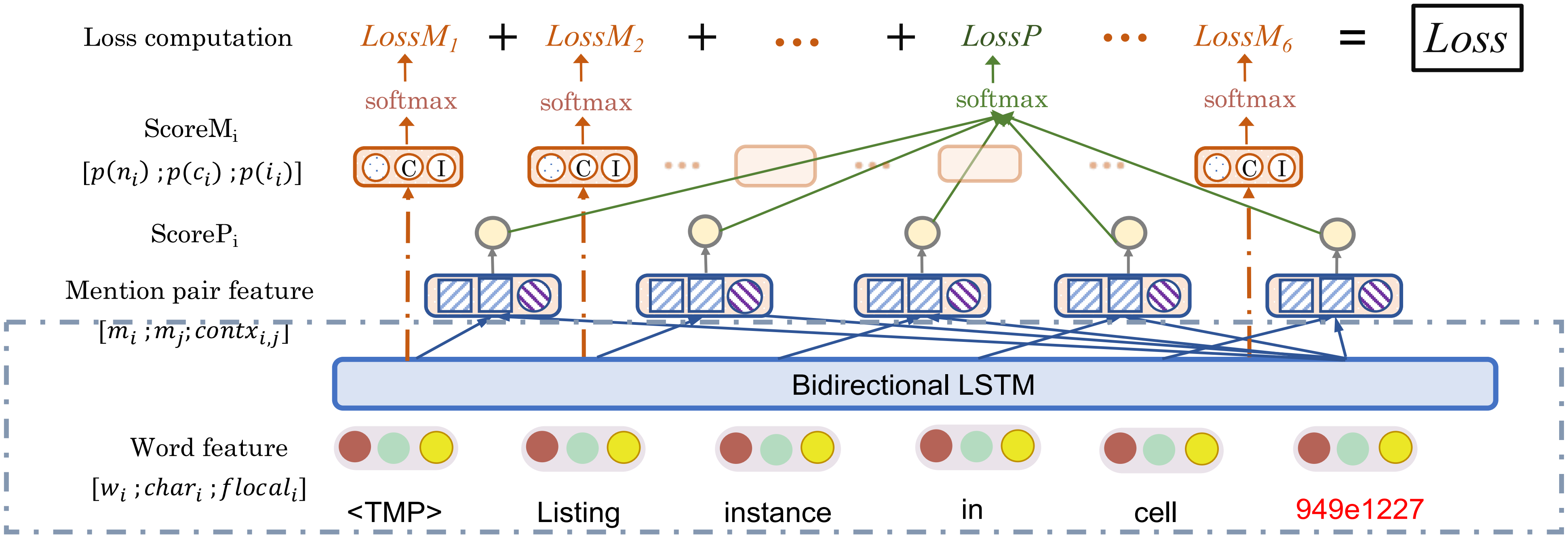
SemParser



Overview (2 steps)

- Input: Log messages
- Step1: Semantics Miner
  - Mine *explicit* variable semantics within single log message.
- Step2: Joint Parser
  - Conduct *implicit* variable semantics inference across log messages.
- Output: Log events (C-Template), semantic pairs (CI pairs), etc..

Step 1: Semantics miner



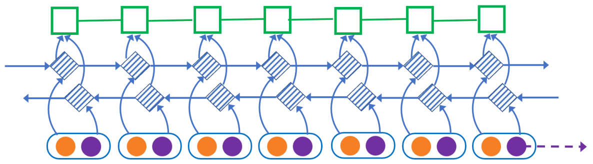
Acquire *contextual representation* for each token:

Token representation

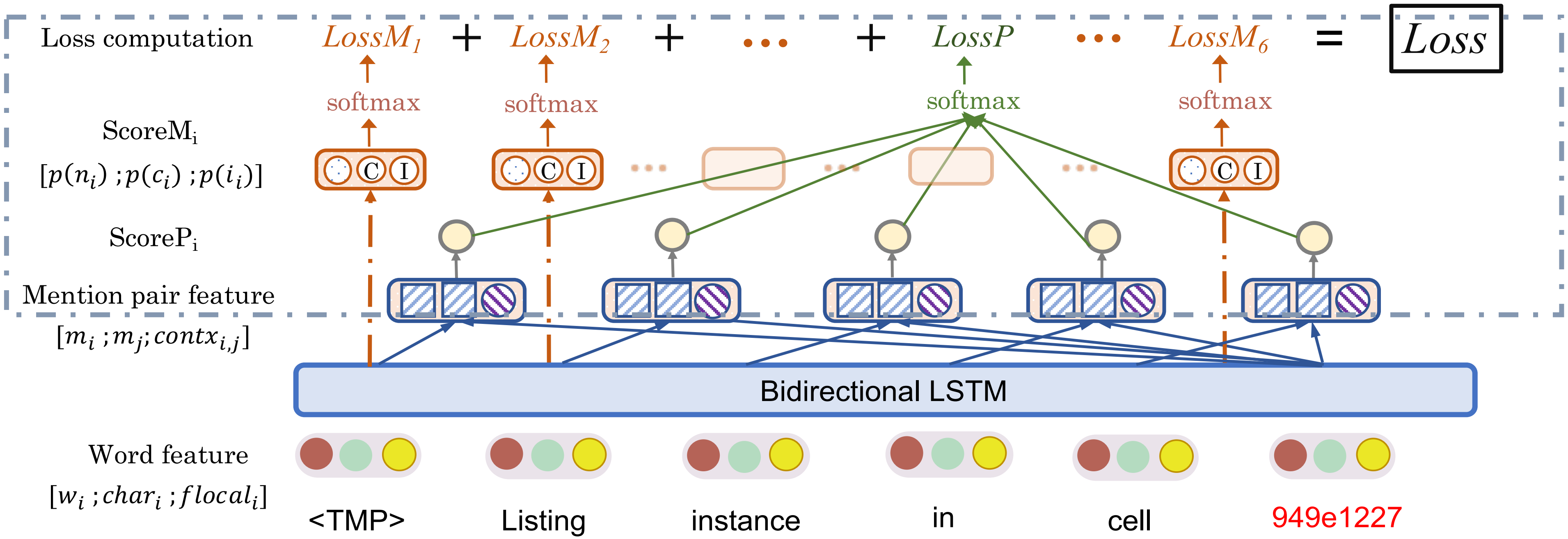
- Word-level: Word2vec.
- Character-level: CNN encoding.
- Local-level: One-hot encoding.

Contextual encoder

- Bi-LSTM to capture interactions and dependencies between words.



Step 1: Semantics miner



Use contextualized word representations for two sub-tasks

- *Word scoring*: determine the semantic roles.
- *Pair matching*: extract CI pairs.

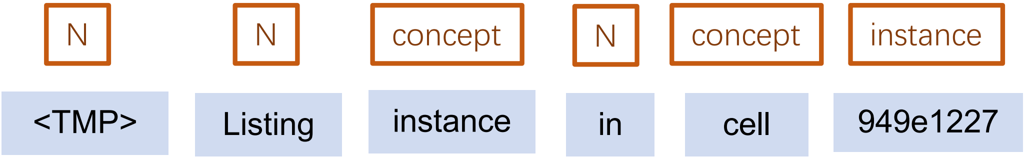
Multi-task learning

- Optimized simultaneously.

Step1: Semantics miner

Word scoring

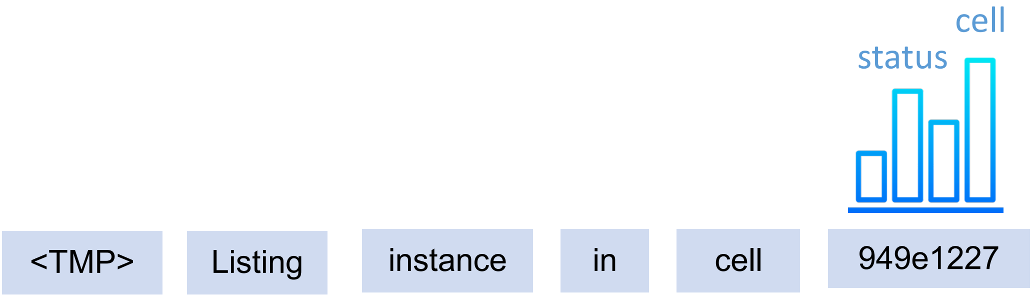
- Goal: determine the semantic role
  - Concept? Instance? Neither of both (N)?
  - Multi-classification problem solved by one feed-forward neural network.



Semantic role for each word?

Pair matching

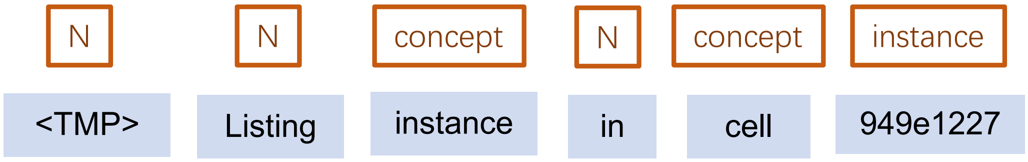
- Goal: discern the CI (concept, instance) pairs



Step1: Semantics miner

Word scoring

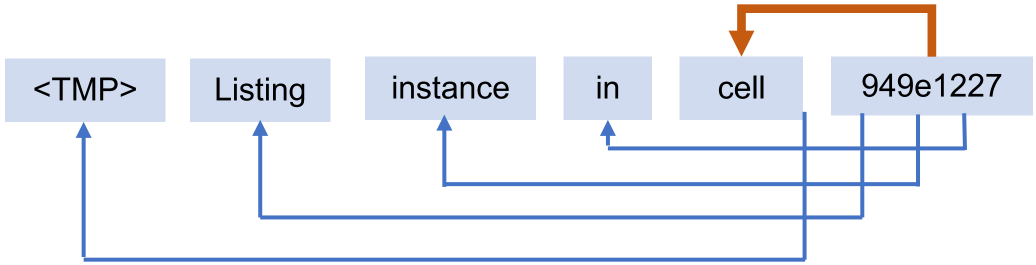
- Goal: determine the semantic role
  - Concept? Instance? Neither of both (N)?
  - Multi-classification problem solved by one feed-forward neural network.



Semantic role for each word?

Pair matching

- Goal: discern the CI (concept, instance) pairs
  - Paring problem: combat the close-world assumption.



Best description for 949e1227?

For each word:

1. *Form* pairs.
2. *Rank* the probability score for each pair (by a feed-forward neural network).
3. *Compute* loss function.

Step2: Joint parser

- Resolve *implicit* variable semantic.
- Sharing knowledge for multiple log messages.

1. ***Incorporate*** newly discovered CI pairs from the semantics miner.
2. ***Update*** the domain knowledge module.
3. ***Match*** with orphan variables.

Algorithm 1 Implicit instance-level semantics discovery

Input: Log message $M = m_0, ..., m_n$, instance indices $I = [i_0, ...i_j]$, concept indices $C = [c_0, ...c_k]$, explicit CI pair indices $P = [(s_0, t_0), ..., (s_u, t_u)]$

Output: Instances I', Concepts C', CI pairs P'

$P' = []$
$C' = []$
for all p such that $p \in P$ **do**
    if p contains 1 instance cur_I and 1 concept cur_C **then**
        DomainKnowledge.add($M[cur_C]$,$M[cur_I]$)
        I.REMOVE(cur_I)
        C.REMOVE(cur_C)
    end if
end for;
for all i such that $i \in I$ **do**
    if FINDCONCEPTFROMDOMAINKNOWLEDGE($M[i]$) **then**
        P'.APPEND([newfound concept, $M[i]$])
        C'.APPEND(newfound concept)
        I.REMOVE(i)
    end if
end for
I' = INDEXTOWORD(I)
C' += INDEXTOWORD(C);
P' += INDEXTOWORD(P)

Experimental settings

Can SemParser effectively extract semantics? (RQ1)

- Dataset for evaluating semantics mining
  - 6 representative system logs from Loghub.
  - Finetune : test = 50 : 1950.
- Metrics
  - Precision
  - Recall
  - F1

System type	System	#Logs	#Pairs	#Temp.	Unseen
Mobile system	Android	2,000	6,478	166	82.8%
Operating system	Linux	2,000	2,905	118	86.8%
Distributed system	Hadoop	2,000	2,592	14	84.6%
	HDFS	2,000	3,105	30	47.0%
	OpenStack	2,000	4,367	43	52.3%
	Zookeeper	2,000	1,189	50	75.9%

Can such semantics benefit operation tasks? (RQ2, RQ3)

- Dataset for downstream task evaluation
  - Contain labeled anomalies.
  - HDFS Dataset.
  - F-Dataset (from OpenStack).

Dataset	#Message	Anomaly rate
HDFS dataset	11,175,629	3%
F-Dataset	1,318,860	0.22%

Experimental results

- Variable semantics mining ability
  - ✓ *94.3% - 99.5%* in accuracy
  - ✓ Each component is beneficial

Framework	Andriod P	Andriod R	Andriod F1	Hadoop P	Hadoop R	Hadoop F1	HDFS P	HDFS R	HDFS F1	Linux P	Linux R	Linux F1	OpenStack P	OpenStack R	OpenStack F1	Zookeeper P	Zookeeper R	Zookeeper F1
SemParser	0.951	0.935	**0.943**	0.993	0.978	**0.985**	1.000	1.000	**1.000**	0.998	0.977	**0.987**	0.999	0.998	**0.999**	1.000	0.989	**0.995**
- w/o F_{char}	0.981	0.909	**0.943**	0.988	0.953	0.970	1.000	0.998	0.999	0.995	0.957	0.976	0.995	0.989	0.992	0.993	0.987	0.990
- w/o F_{local}	0.979	0.858	0.915	0.993	0.880	0.933	1.000	0.999	0.999	0.992	0.947	0.969	0.994	0.989	0.992	0.997	0.940	0.968
- w/o $LSTM$	0.979	0.858	0.915	0.993	0.879	0.932	1.000	0.999	0.999	0.995	0.909	0.951	1.000	0.963	0.981	0.966	0.953	0.959
- w/o F_{contx}	0.977	0.060	0.113	0.984	0.253	0.403	0.999	0.289	0.449	0.999	0.242	0.389	1.000	0.256	0.407	0.842	0.197	0.319

Experimental results

- Enhance subsequent anomaly detection
  - ✓ *0.82% - 2%* in HDFS
  - ✓ *8.27% - 16.58%* in OpenStack

(a) HDFS Dataset.

Baseline	DeepLog P	DeepLog R	DeepLog F1	LogRobust P	LogRobust R	LogRobust F1	CNN P	CNN R	CNN F1	Transformer P	Transformer R	Transformer F1
LenMa	.897	.994	.943	.914	.995	.953	.924	.995	.958	.872	.908	.890
AEL	.896	.994	.943	.935	**.996**	.964	.922	.995	.958	**.893**	.904	.898
Drain	.908	.994	.949	.934	.994	.963	.925	.995	.959	.886	.871	.878
IPLoM	.898	.994	.944	.940	.994	.966	.926	**.996**	.960	.889	.904	.896
SemParser	**.940**	**.995**	**.967**	**.954**	.995	**.974**	**.931**	.995	**.962**	.881	**.954**	**.916**
Δ%	+1.86%			+0.82%			+0.21%			+2.00%		

(b) F-Dataset

Baseline	DeepLog P	DeepLog R	DeepLog F1	LogRobust P	LogRobust R	LogRobust F1	CNN P	CNN R	CNN F1	Transformer P	Transformer R	Transformer F1
LenMa	.717	**.938**	.813	.714	**.924**	.806	.793	.815	.804	.685	.896	.776
AEL	.738	.934	.824	.791	.877	.832	.747	.924	.826	.503	**.962**	.660
Drain	.824	.867	.845	.810	.886	.846	.737	**.943**	.827	.693	.919	.790
IPLoM	.863	.833	.848	.808	.877	.841	.834	.834	.834	.929	.683	.787
SemParser	**.971**	.927	**.948**	**.952**	.913	**.932**	**.907**	.899	**.903**	**.938**	.904	**.921**
Δ%	+11.80%			+10.17%			+8.27%			+16.58%		

Experimental results

- Enhance subsequent failure identification
  - ✓ *3.81% - 12.5%* in Recall@1
  - ✓ -0.42% – 2.65% in Recall@2

Baseline	LSTM Rec@1	LSTM Rec@2	LSTM Rec@3	Atten-biLSTM Rec@1	Atten-biLSTM Rec@2	Atten-biLSTM Rec@3	CNN Rec@1	CNN Rec@2	CNN Rec@3	Transformer Rec@1	Transformer Rec@2	Transformer Rec@3
LenMa	0.839	0.924	0.953	0.858	0.943	0.957	0.877	0.962	0.967	0.919	0.934	0.948
AEL	0.844	0.919	0.953	0.853	0.915	0.962	0.810	0.905	0.929	0.858	0.929	0.953
Drain	0.844	0.919	**0.972**	0.863	0.938	0.953	0.867	0.948	0.967	0.853	0.919	0.943
IPLoM	0.848	0.943	0.957	0.863	0.948	0.962	0.867	**0.967**	**0.986**	0.839	0.910	0.948
SemParser	**0.954**	**0.968**	0.968	**0.954**	**0.968**	**0.972**	**0.945**	0.963	0.972	**0.954**	**0.958**	**0.968**
Δ%	+12.50%	+2.65%	-0.41%	+10.54%	+2.11%	+1.04%	+7.75%	-0.42%	-1.44%	+3.81%	+2.46%	+2.11%

API error	server add volume
Log message	... Cannot 'attach_volume' instance 853cfe1b ...
C-Template	... Cannot 'attach_volume' instance <*server*> ...
CI Pairs	[(server, 853cfe1b)]

Summary of Topic 1

- SemParser: a semantic-aware log parsing techniques for subsequent analysis
  - ✓ Motivation: Existing syntax-based parsers ignore semantics within logs.
  - ✓ Building the *first semantic-based log parser*, which can actively capture *intra-log* and *inter-log* semantics.
  - ✓ Reveal the *contribution of log semantics* on software operation tasks.

Outline

- Log-driven automated SRE
  - Reactive Log analysis [ICSE'23, ASE'23, ISSRE'23, ICSE'24, FSE'24, ISSTA'24]
    - Topic1: A semantic-aware log parser for software operations [ICSE'23] (Chapter 3)
    - Topic2: A log localizer for evolving software systems [ISSRE'23] (Chapter 4)
    - Topic3: An automatic log sequence synthesizer for anomaly detection [ASE'23] (Chapter 5)
  - Proactive Logging practice [FSE'24, TSE'24 Major]
    - Topic4: An empirical study on automatic logging statement generator [TSE'24 Major] (Chapter 6)
- Conclusion and Future Work

Finer-grained log analysis

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

Anomaly detection

- **Coarse-grained approach**
- Detect anomalies in a session
- Provide limited information

line100-200 **: Anomalous!**

Anomalous log identification (root cause analysis)

- **Fine-grained approach**
- Identify specific “anomaly” logs
- Provide detailed information

line104 **: Anomalous!**

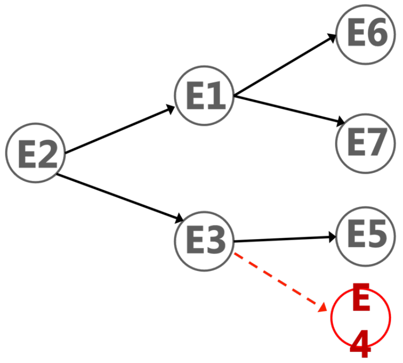
Existing solutions

Got assigned task 5
Found block 46
Dropping block 46 from ...
Ignored message: Heartbeat
5 blocks selected for drop
...
Got assigned task 7
Got assigned task 8

Raw logs (training phase)

E1: Got assigned task <*>
E2: Found block <*>
E3: Dropping block <*> from ...
E4: Ignored message: Heartbeat
E5: <*> blocks selected for drop
...
E1: Got assigned task <*>
E1: Got assigned task <*>

Parsed log events
(A sequence of log event ID)



Reference model

Raw logs (testing phase)

Simple assumption

Model all possibilities of the normal log sequences

Strict,
rigorous,
Close-world assumption

Any *violation* of the reference model will be considered as the *anomalous log*.

Software keeps evolving…

Version 1.1

[DATE] [TIME] INFO Started reading broadcast variable 9
[DATE] [TIME] INFO Got assigned task 5
[DATE] [TIME] INFO Found block 46
…

Software evolution

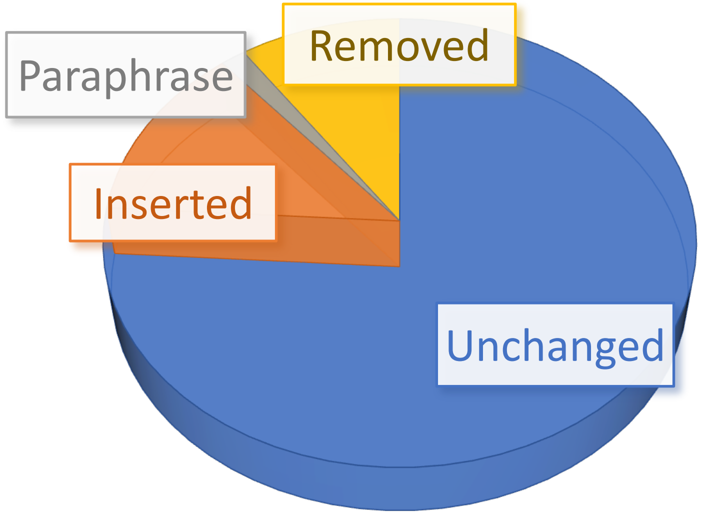
```
  public void handleEvent(Event event){
     String path = event.getProperty(PATH);
-   log.info(”Started reading broadcast variable", variable);
+   log.info(“Started reading broadcast variable”, variable, “with pieces in total size”, size)
     if (PATH != null) {
        String includePath = PATH
                   …
```

Version 1.2

[DATE] [TIME] INFO Started reading broadcast variable 5 with pieces in total size 1024
[DATE] [TIME] INFO Got assigned task 5
[DATE] [TIME] INFO Found block 46
…

The evolution of logging statements

- Spark2 and Spark3
- Evolving logging statements
  - ✓ Insert (12.9%)
  - ✓ Paraphrase (1.49%)
  - ✓ Remove (9.7%)



CASE I. ***Insert*** *a log logging statement in Spark3:*

Discovering resources for <*> with script:<*>

CASE II. ***Paraphrase*** *a log logging statement in Spark3 from Spark2:*

Started reading broadcast variable <*>

Started reading broadcast variable <*> with <*> pieces (estimated total size <*> MiB)

CASE III. ***Remove*** *a log logging statement from Spark2:*

Scala <*> cannot get type nullability correctly via reflection, thus Spark cannot add proper input null check for UDF.

Observation#1: A large amount of logging statements (24.9%) change over software evolution

The evolution of logging statements

- Spark2 and Spark3
- Evolving logging statements
  - ✓ Insert (12.9%)
  - ✓ Paraphrase (1.49%)
  - ✓ Remove (9.7%)

CASE I. ***Insert*** *a log logging statement in Spark3:*

Discovering resources for <*> with script:<*>

CASE II. ***Paraphrase*** *a log logging statement in Spark3 from Spark2:*

Started reading broadcast variable <*>

Started reading broadcast variable <*> with <*> pieces (estimated total size <*> MiB)

CASE III. ***Remove*** *a log logging statement from Spark2:*

Scala <*> cannot get type nullability correctly via reflection, thus Spark cannot add proper input null check for UDF.

1.49% paraphrased logging statements

Account for

8.75% log messages

Observation#2: developers always change the frequently-used logging statements.

How does the evolution affect reference models?

Raw logs → Parsed log events → Reference model

#1 Challenge. Parsing error

Parsing result:
Connecting to ResourceManager at sp2sl1 / 172.17.0.3:8030

Ground truth:
Connecting to ResourceManager at <*>

Parsing result for Spark2:
Changing <*> acls <*> <*>

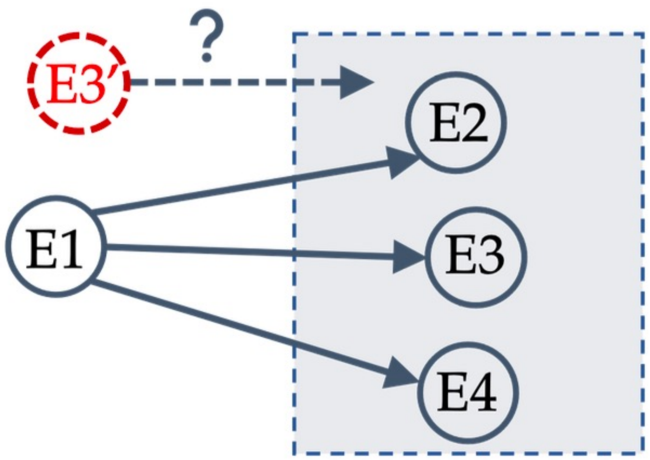
Parsing result for Spark3:
Changing <*> acls groups to:
Changing <*> acls to: root

- Log parsers can introduce errors.
- The evolution of logs over time can make parsing even more challenging.

How does the evolution affect reference models?

Raw logs → Parsed log events → Reference model

#2 Challenge. Evolving events



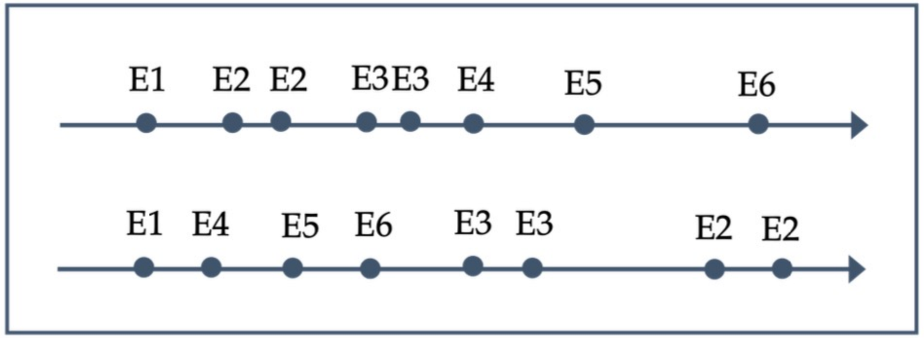
E1: Running task <*> in stage <*> (TID <*>)
E3 (in Spark2): Started reading broadcast variable <*>
E3'(in Spark3): Started reading broadcast variable <*> with <*> pieces (estimated total size <*> MiB)

- Event-matching.
- A paraphrased logging statement can *mislead* the reference model.

How does the evolution affect reference models?

Raw logs → Parsed log events → Reference model

#3 Challenge. Unstable sequences



Spark2:	**Spark3:**
E1 -> E3	E1 -> E2 -> E3

E1: Connecting to driver: <*>
E2: Successfully registered with driver
E3: Resources for <*>:

A new logging statement E2 can *alter* previously collected log sequences.

Our approach: EvLog

Two Insights

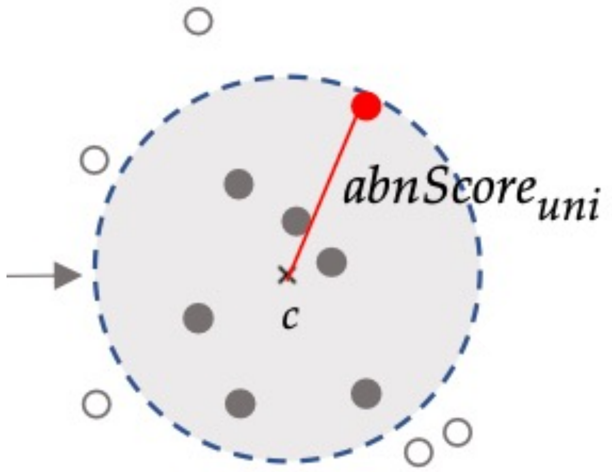
- The majority of logs are *normal* in a healthy system (*normal* >> *abnormal*).
- The anomalous logs are unknown *a priori* because we *cannot inject all kinds of failures.*

Intuition

- *~~Happy families~~ are all alike; every ~~unhappy famility~~ is unhappy in its own way.*

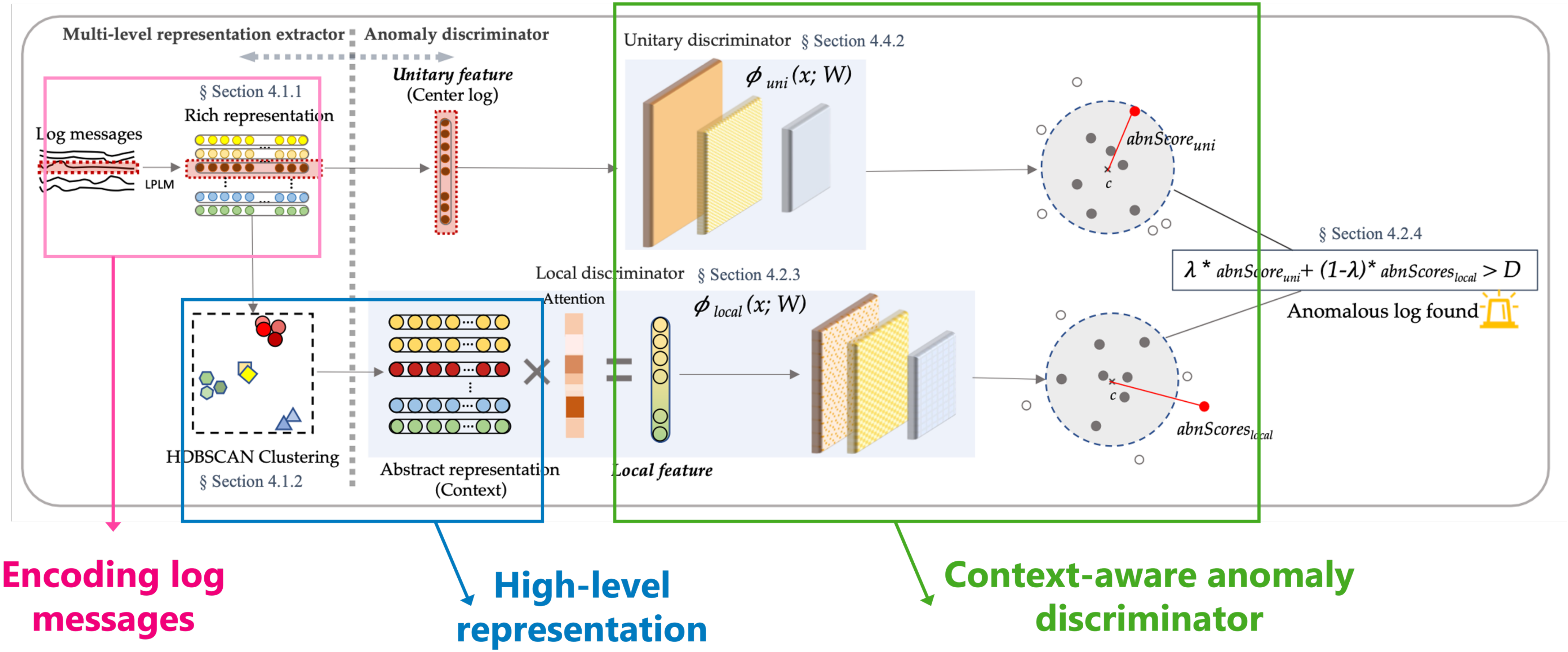
  Normal logs *anomalous log* ---<<Anna Karenina>>

Basic idea



Our approach: EvLog

- Our goal: Identifying anomalous logs over software evolution
- Our challenge: *Parsing error*, *evolving events*, *unstable sequences*



Multi-level representation extractor

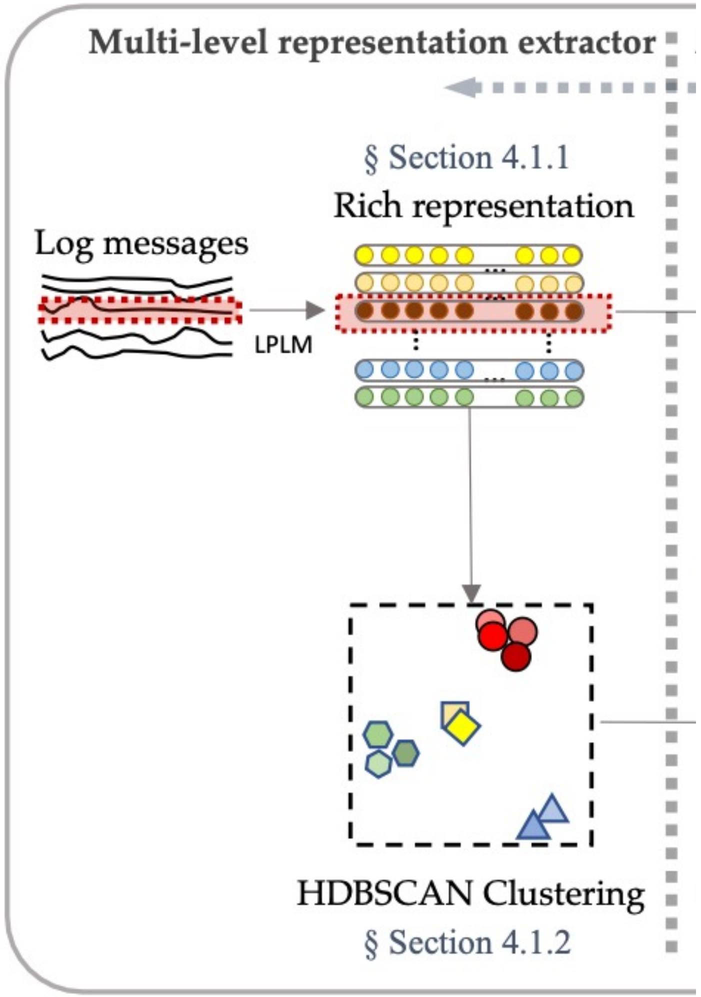
Rich representation

- Using a pre-trained language model to obtain log semantics.
- Eliminating log parsing errors.

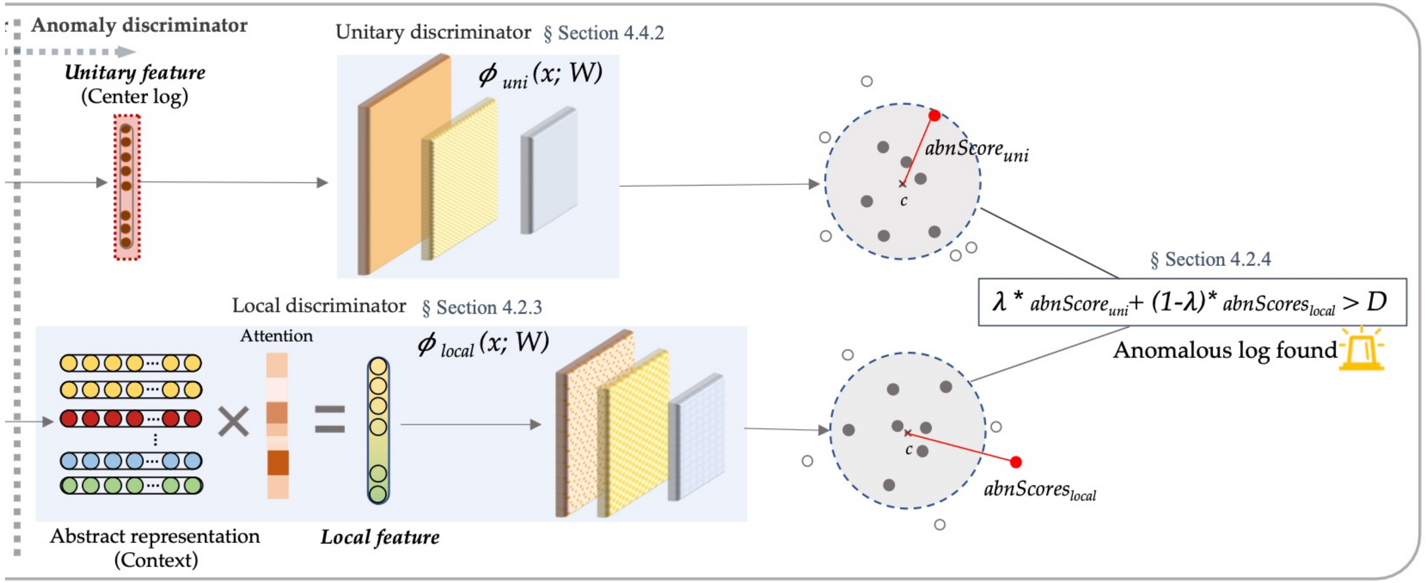
Abstract representation

- Goal: extract a high-level semantic representation
- Cluster the rich representation by HDBSCAN
- Each log is represented by the centroid of its cluster

Paraphrased logs will not change their abstract representation → *stable over evolution*



Anomaly discriminator



Basic idea

- Learn the "normality" of normal logs

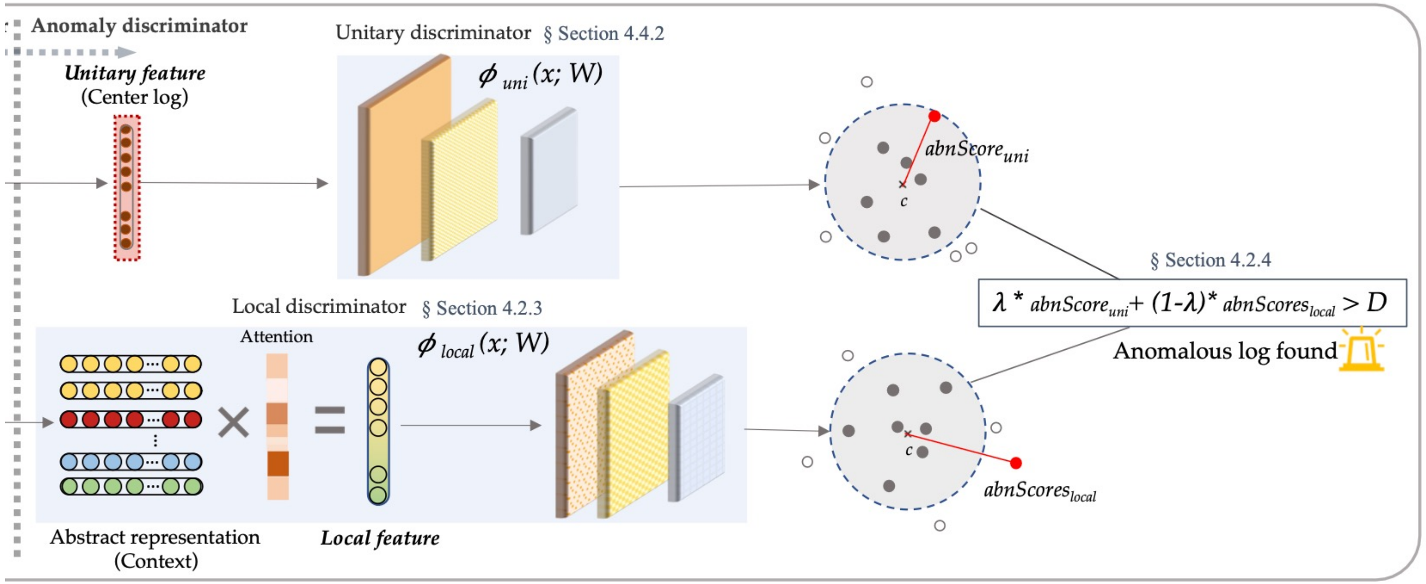
How to measure normality?

- Two aspects
  - Unitary
  - Local (context)

Unitary discriminator

- Learn the single log feature
- The individual log with negative words ("failure") usually be anomalous.

Anomaly discriminator



Basic idea

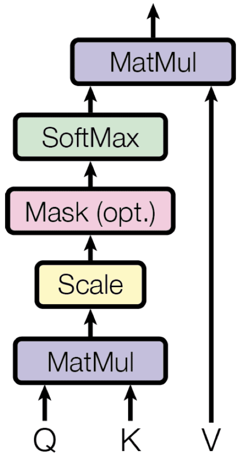
- Learn the "normality" of normal logs

How to measure normality?

- Two aspects
  - Unitary
  - Local (context)

Local discriminator

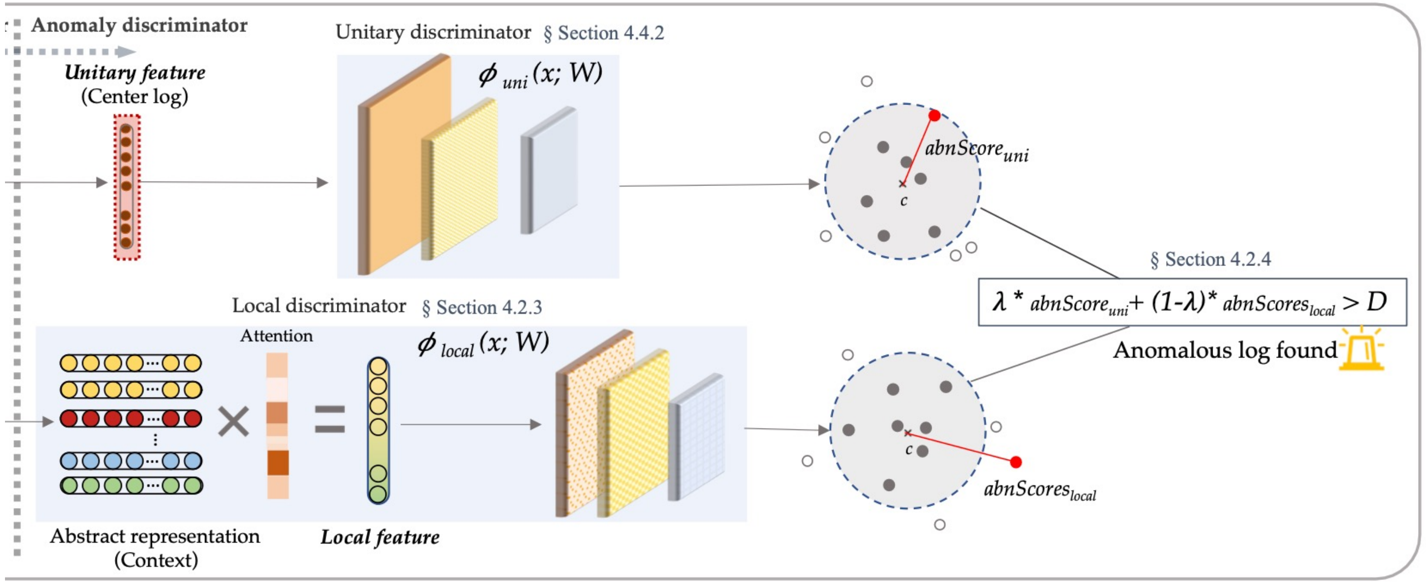
- Different logs, different importance
- Asynchronized log collection -> unstable sequences
- Apply the "attention mechanism" to learn the surrounding log context



Anomaly discriminator

Basic idea

- Learn the "normality" of normal logs



How to measure normality?

- Two aspects
  - Unitary
  - Local (context)

How to integrate?

- Consider the two discriminator jointly

$$J = \min_{W} \frac{1}{n} \sum_{i=1}^{n} \|\phi(x_i; W) - c\|^2 + \frac{\alpha}{2}\|W\|^2$$

$$abnScore = \lambda * abnScore_{uni} + (1-\lambda) * abnScore_{local},$$
$$abnScore_i = \|\phi_i(x; W_i) - c_i\|^2, i \in \{uni, local\}.$$

Experiments

Dataset

- 2 widely-studied systems
  - Hadoop and Spark
- 2 version for each system
- 22 different workloads
- 18 typical failures

Categories	Workloads
Micro task	Sort, Wordcount, etc.
Machine learning	Bayes Classification, Gradient Boosted Trees, etc.
SQL	Aggregation, Join, Scan etc.
Websearch	Pagerank
Graph	NWeight, Graph Pagerank
Streaming	Repartition

In total, 6,703,460 log messages with recognized 69,513 anomalous logs.

Experiments

Metrics

- (binary classification) Precision, Recall, and F1

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$\text{F1} = 2 \times \frac{\textit{Precision} * \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

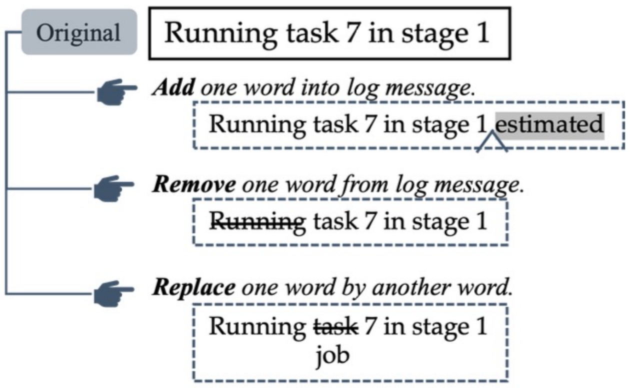
Experiment results

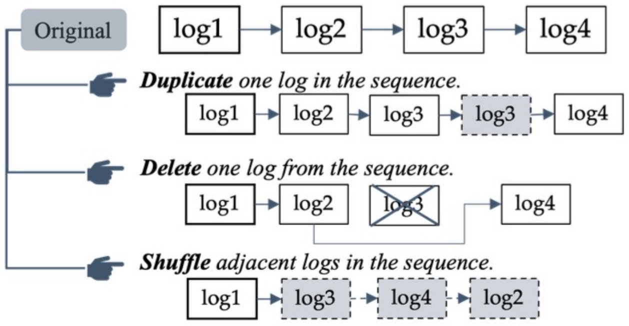
- Effectiveness in localizing anomalous logs
  - ✓ *91.5% - 97.2%* in F1 for intra-version
  - ✓ 79.5%-88.4% in F1 for inter-version

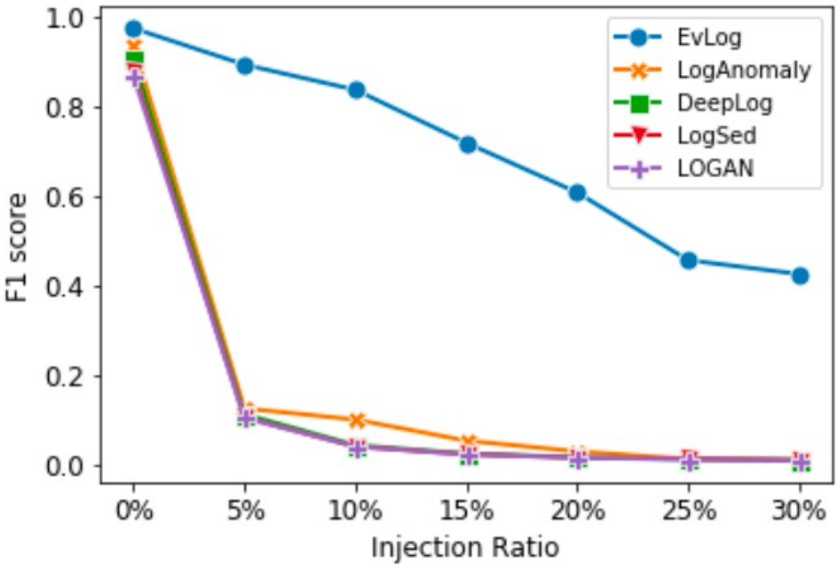
LOGEVOL-HADOOP												
	Intra-version						**Inter-version**					
	Hadoop2 → Hadoop2			Hadoop3 → Hadoop3			Hadoop2 → Hadoop3			Hadoop3 → Hadoop2		
Baseline	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
LOGAN	0.894	0.995	0.942	0.899	0.988	0.942	0.360	0.988	0.528	0.376	0.995	0.546
LogSed	0.910	0.995	0.951	0.925	0.986	0.955	0.371	0.988	0.540	0.390	0.993	0.560
DeepLog	0.913	0.985	0.947	0.926	1.000	0.961	0.386	0.999	0.556	0.410	0.971	0.576
LogAnomaly	0.926	0.994	0.958	0.939	0.988	0.963	0.389	0.998	0.560	0.407	0.995	0.578
EvLog	0.945	0.982	**0.963**	0.952	0.988	**0.970**	0.770	0.941	**0.847**	0.857	0.913	**0.884**
LOGEVOL-SPARK												
	Intra-version						**Inter-version**					
	Spark2 → Spark2			Spark3 → Spark3			Spark2 → Spark3			Spark3 → Spark2		
Baseline	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
LOGAN	0.798	0.943	0.865	0.967	0.870	0.916	0.016	0.943	0.032	0.012	0.943	0.024
LogSed	0.842	0.914	0.877	0.907	0.923	0.915	0.013	0.917	0.026	0.010	0.914	0.020
DeepLog	0.862	0.952	0.905	0.858	0.976	0.914	0.017	0.947	0.032	0.014	0.909	0.026
LogAnomaly	0.931	0.939	0.935	0.898	0.947	**0.922**	0.020	0.923	0.038	0.017	0.948	0.034
EvLog	0.970	0.974	**0.972**	0.944	0.888	0.915	0.922	0.700	**0.795**	0.920	0.812	**0.863**

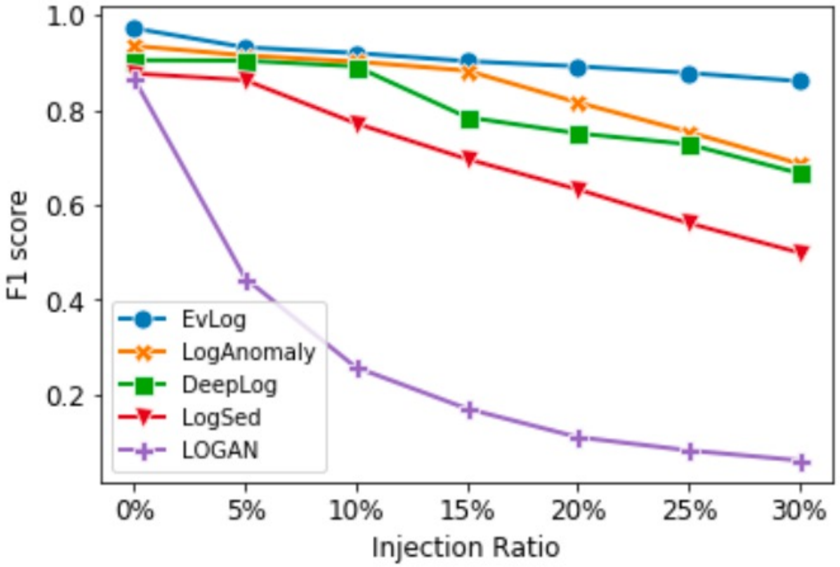
Experiment results

- Robust for evolution types (blue line)
  - ✓ *Evolving log events*
  - ✓ *Unstable log sequences*







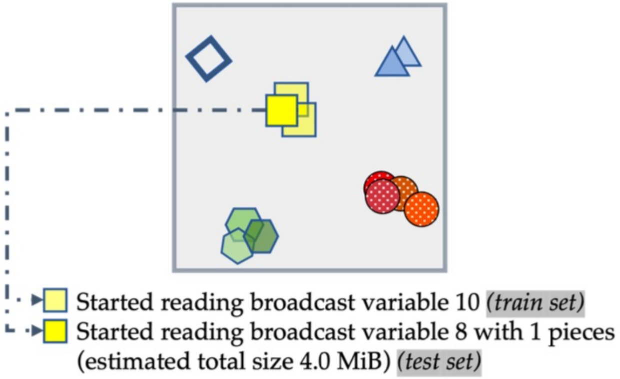


Case study

Prediction of each model

Log sequence	DeepLog	LogAnomaly	LOGAN	LogSed	EvLog	GT
Running task 6.0 in stage 5.0 (TID 71)						
Started reading broadcast variable 8 with 1 pieces (estimated total size 4.0 MiB)						
Started reading broadcast variable 6 with 1 pieces (estimated total size 4.0 MiB)						
Started 1 remote fetches in 2 ms						
Started 1 remote fetches in 4 ms						
I/O error constructing remote block reader.						
Failed to connect to /172.17.0.2:50010 for block, add to deadNodes…						

EvLog: Cluster for abstract representation



Summary of Topic 2

- EvLog: an anomalous log localization framework for evolving software systems
  - ✓ Motivation: Existing approaches rely on unchanged log events
  - ✓ Revealing *three challenges from log evolution*.
  - ✓ Building the *first evolution-adaptive log localizer via one-vs-all classification techniques.*

Outline

- Log-driven automated SRE
  - Reactive Log analysis
    [ICSE'23, ASE'23, ISSRE'23, ICSE'24, FSE'24, ISSTA'24]
    - Topic1: A semantic-aware log parser for software operations — [ICSE'23] (Chapter 3)
    - Topic2: A log localizer for evolving software systems — [ISSRE'23] (Chapter 4)
    - Topic3: An automatic log sequence synthesizer for anomaly detection — [ASE'23] (Chapter 5)
  - Proactive Logging practice
    [FSE'24, TSE'24 Major]
    - Topic4: An empirical study on automatic logging statement generator — [TSE'24 Major] (Chapter 6)
- Conclusion and Future Work

Inspired from the training process of LLMs

ChatGPT: Intelligent chatbot

Copilot: Smart programming assistant

Bing AI: Searching with AI

...

Large language models (ChatGPT as an exampe)

Continuing developing

Foundation: training with **a large amount of high-quality** dataset.

Dataset is the core of data-driven models

What do we have for intelligent **log analysis**?

- Collecting logs from **real-world service providers**:
  + Rich log events
  -- Privacy issues

- Collecting logs from **laboratory environments**:
  + Publicly available
  -- Simplified log events

Collecting logs for open-source research is demanding yet challenging!

Existing log datasets for anomaly detection

One of the most widely-used log datasets, LogHub.

Dataset	# Log Event	# Workload	# Failure Type	# Message	Collection Time
$\mathcal{D}$-HDFS	30	NA	11	11,175,629	38.7 Hours
$\mathcal{D}$-Hadoop	242	2	3	394,308	NA
$\mathcal{D}$-BGL	619	NA	NA	4,747,963	214.7 days
$\mathcal{D}$-Zookeeper	77	NA	NA	207,820	26.7 days

#1 ***Comprehensiveness***
of log events

- Limited number of workloads and failures.
- Unrealistic to simulate all kinds of system behaviors.

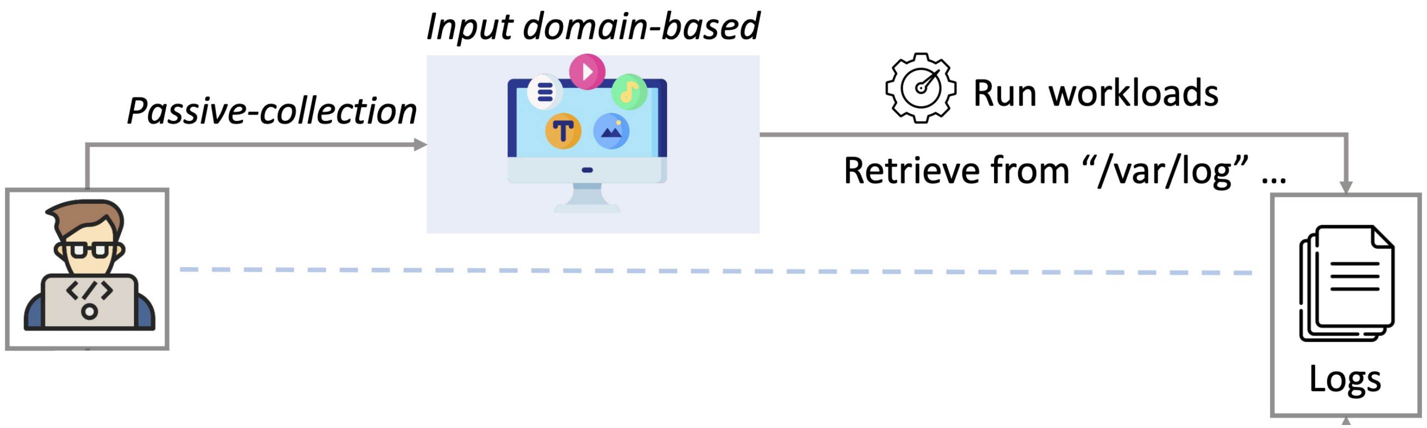
#2 ***Scalability***
over diverse systems

- Require human efforts to deploy new systems.
- Limited in system diversity.

#3 ***Flexibility***
of log utility

- Not controllable for imitating different scenarios.

The idea of AutoLog



How logs are generated?

```
public void setTemperature(Integer temperature) {
  // ...
  logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);
  if (temperature.intValue() > 50) {
    logger.info("Temperature has risen above 50 degrees.");
  }
}
```

⇓

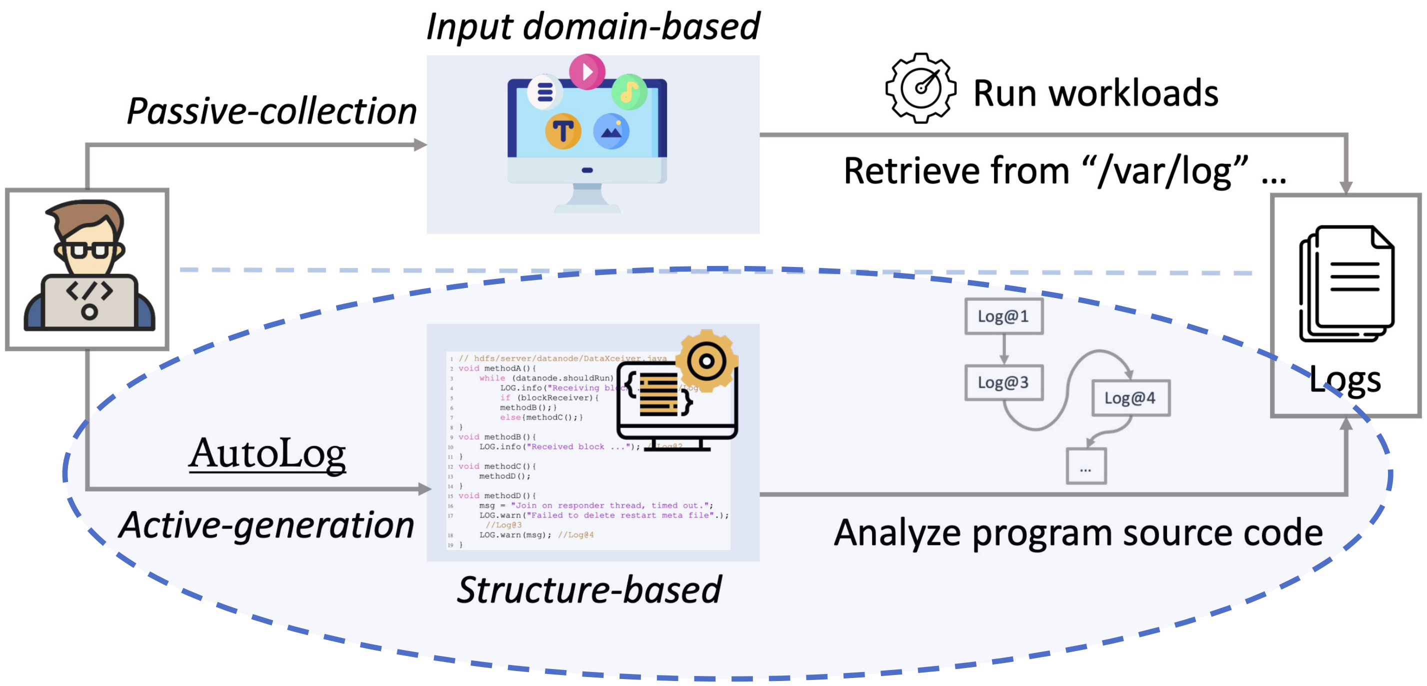
```
0 [setTemperature] DEBUG Wombat - Temperature set to 61. Old temperature was 42.
0 [setTemperature] INFO Wombat - Temperature has risen above 50 degrees.
```

Logging statements in the source code

Collected logs

Logs are generated during the execution of logging statements in the source code.

The idea of AutoLog

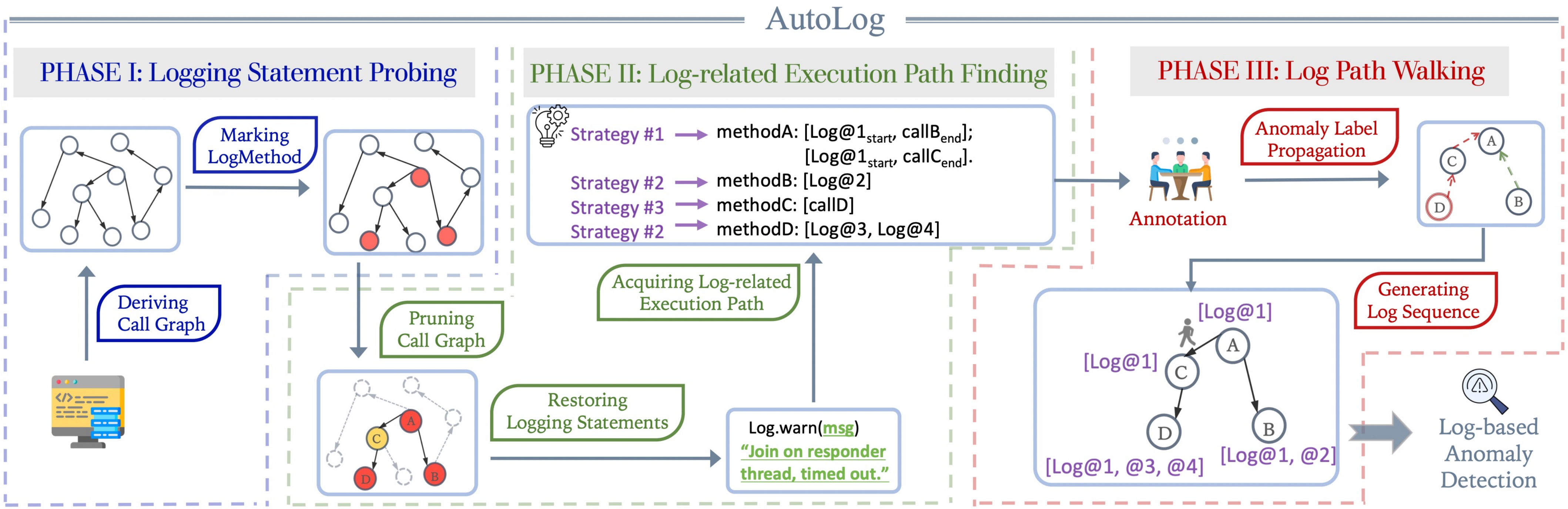


Log collection problem → Constructing ***log sequences*** based on the ***execution order*** of logging statements

Logs are generated during the execution of logging statements in the source code.

AutoLog framework

Goal: Constructing execution paths related to logging statements in a program.

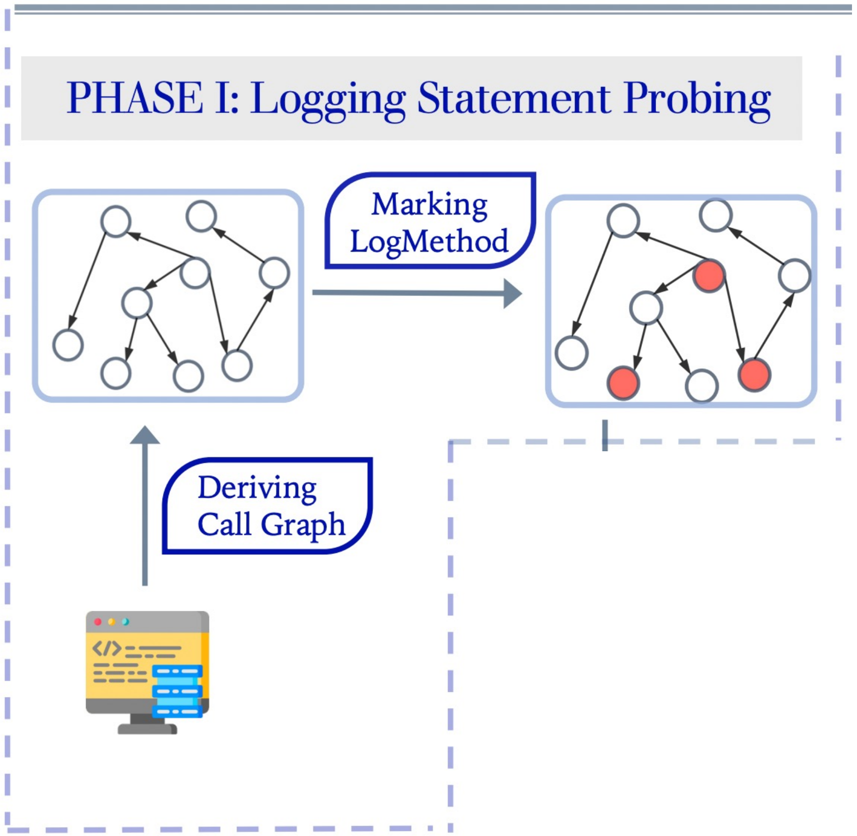


1. Exploring the methods with logging statements and their calling relationships.

2. Finding log-related execution paths over the program.

3. Traversing the execution paths to obtain normal log sequences and abnormal ones.

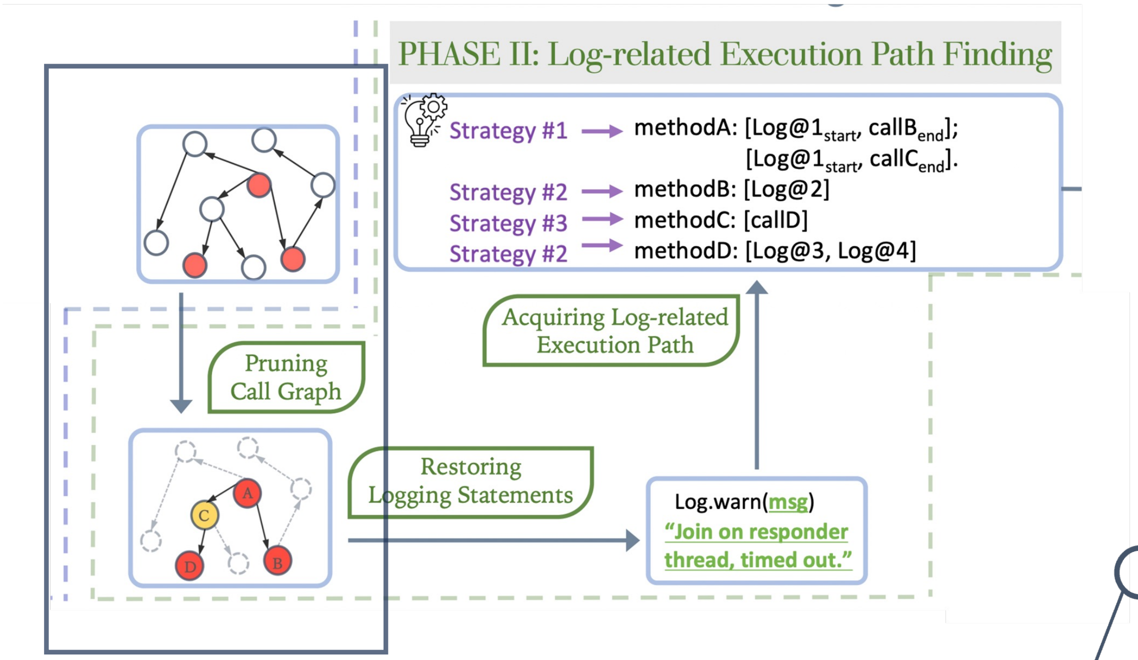
Phase 1: Logging statement probing



Exploring logging statements in the whole program.

- Deriving call graphs.
- Marking methods containing logging statements (LogMethod).

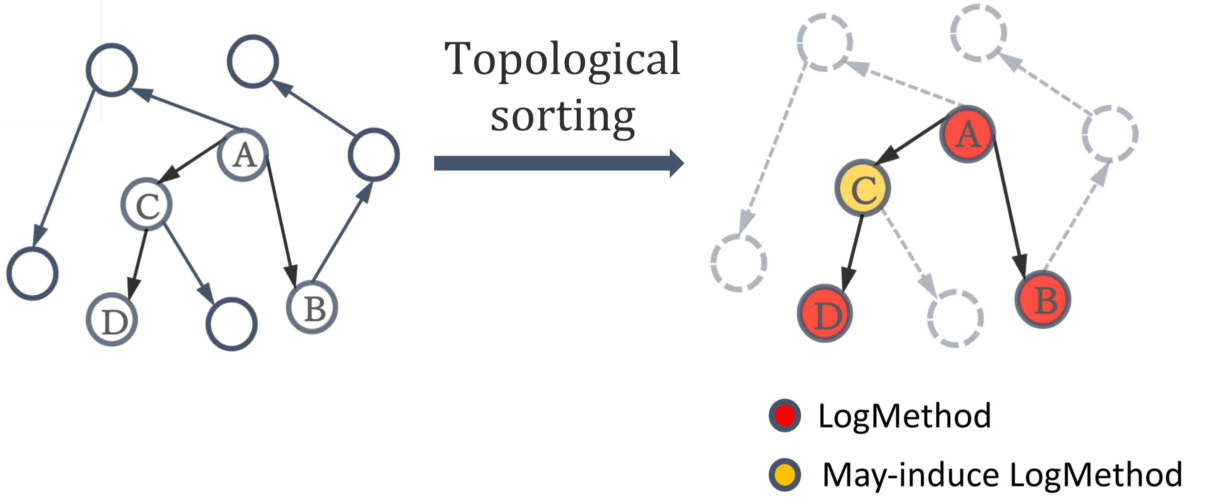
Phase2: Log-related execution path finding



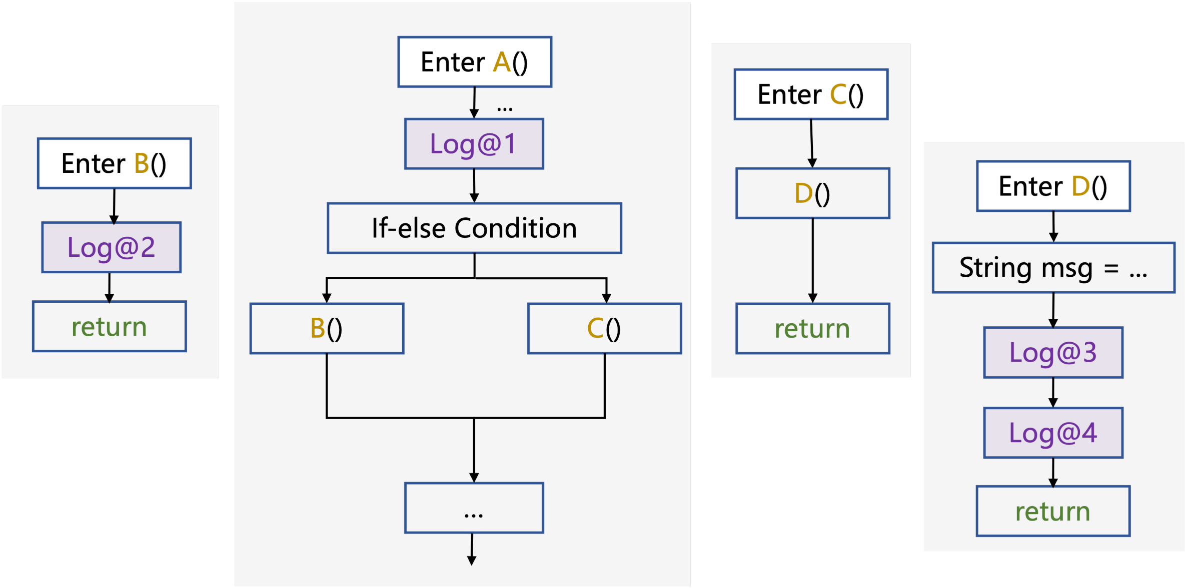
Constructing log-related execution path.

Challenge: Enumerating the paths in large-scale software is impractical.

Step1: Pruning call graph.



Phase2: Log-related execution path finding



Constructing log-related execution path.

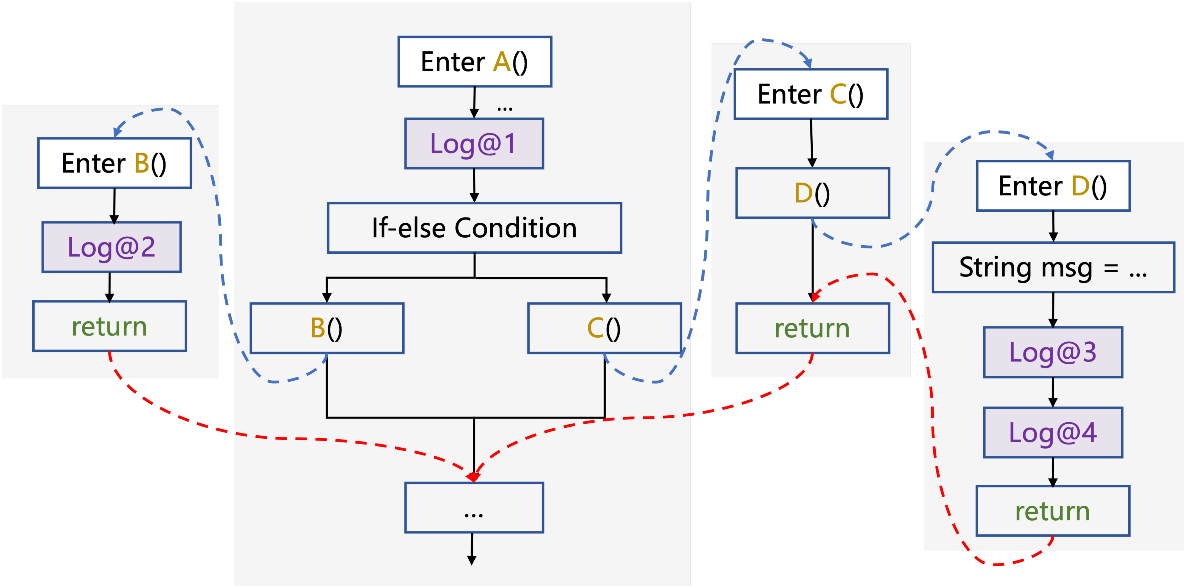
Challenge: Enumerating the paths in large-scale software is impractical.

Step2: Acquiring log-related execution paths (LogEPs).

Getting LogEPs

1. Constructing control flow graphs for intra-methods.

Phase2: Log-related execution path finding



Constructing log-related execution path.

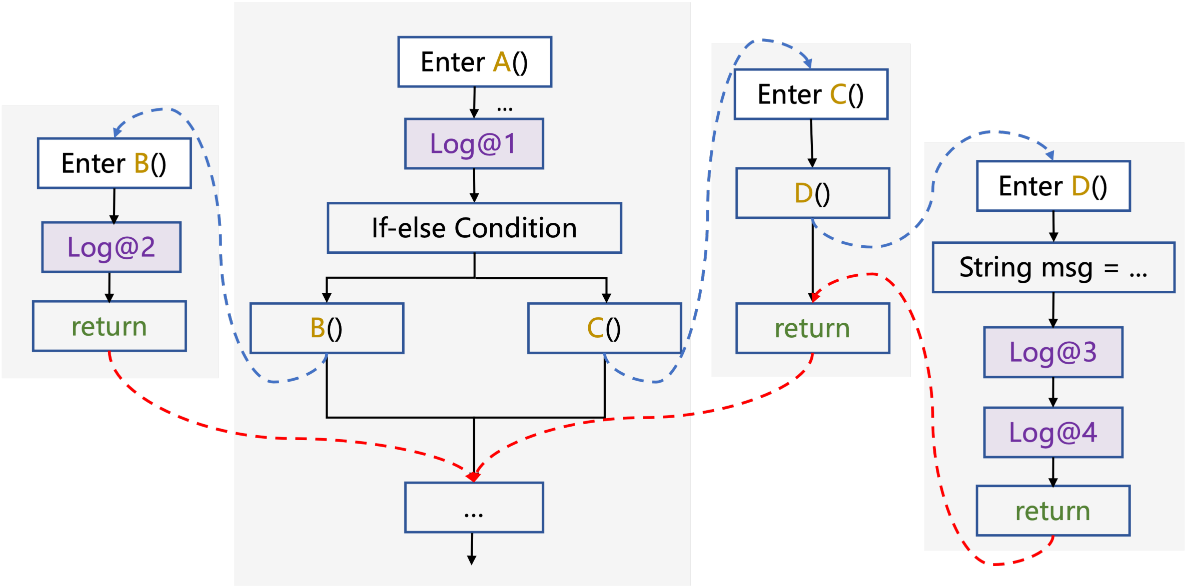
Challenge: Enumerating the paths in large-scale software is impractical.

Step2: Acquiring log-related execution paths (LogEPs).

Getting LogEPs

1. Constructing control flow graphs for intra-methods.
2. Linking the invocations.

Phase2: Log-related execution path finding



- methodA: [Log@1, callB]; [Log@1, callC]
- methodB: [Log@2]
- methodC: [callD]
- methodD: [Log@3, Log@4]

Constructing log-related execution path.

Challenge: Enumerating the paths in large-scale software is impractical.

Step2: Acquiring log-related execution paths (LogEPs).

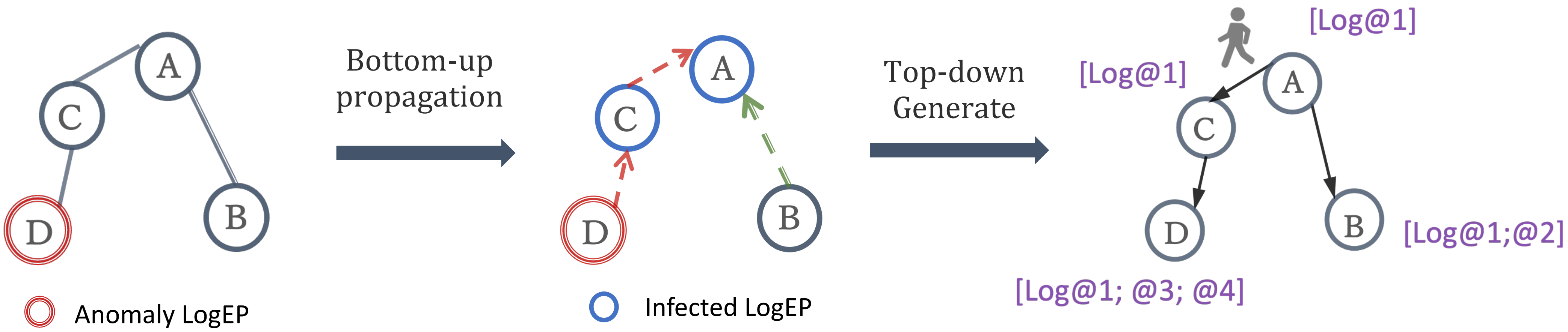
Getting LogEPs

1. Constructing control flow graphs for intra-methods.
2. Linking the invocations.
3. Recording the *invocations* and *logging statements*.

Phase3: Log path walking

Generating normal log sequences and anomaly ones. *Efficient* annotation: *seed-propagation*

1. Annotating “seed” anomaly LogEPs.
2. Propagating labels to all LogEPs.
3. Generating log sequences by walking over LogEPs and their invocations.



Experimental settings

Can AutoLog generate quality log sequence? (RQ1, RQ2)

- Dataset
  - Same system as in LogHub
  - 50 most-popular Java projects from Maven
- Metrics
  - Coverage of all logging statements
  - Execution time

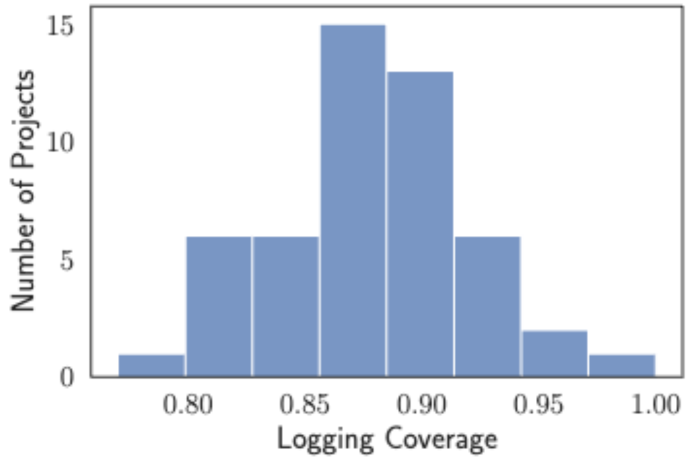
Can such sequence benefit anomaly detection? (RQ3)

- Training resource
  - Train in AutoLog VS. Train in LogHub
- Benchmarking resource
  - Evaluating by AutoLog
- Metrics
  - Precision
  - Recall
  - F1

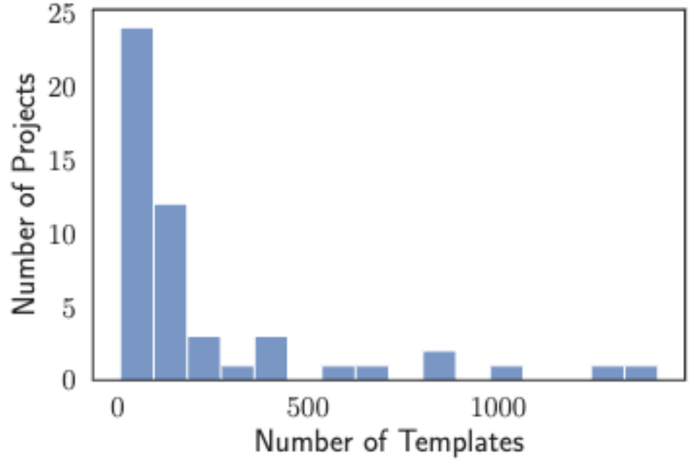
RQ1: Comprehensiveness?

- Simulate comprehensive system behavior
  - ✓ *9x – 58x* on the number of log events
  - ✓ *Covers 87.8%* logging statements on average

System	Dataset	# Log Event	Logging Coverage	$\mathcal{D}$-Coverage	Increment (↑)
Hadoop	$\mathcal{D}$-Hadoop AUTOLOG-Hadoop	242 2879	242/3426 (7.1%) 2879/3426 (84.0%)	219/242 (90.5%)	12x
HDFS	$\mathcal{D}$-HDFS AUTOLOG-HDFS	30 1367	30/1700 (1.8%) 1367/1700 (80.4%)	27/30 (90.0%)	58x
Zookeeper	$\mathcal{D}$-Zookeeper AUTOLOG-Zookeeper	77 740	77/758 (10.2%) 740/758 (97.6%)	77/77 (100%)	9x
Apache Storm Flink Kafka	AUTOLOG-Apache Storm AUTOLOG-Flink AUTOLOG-Kafka	1754 1574 847	1754/1887 (93.0%) 1574/1711 (92.0%) 847/1002 (84.5%)	- - -	- - -



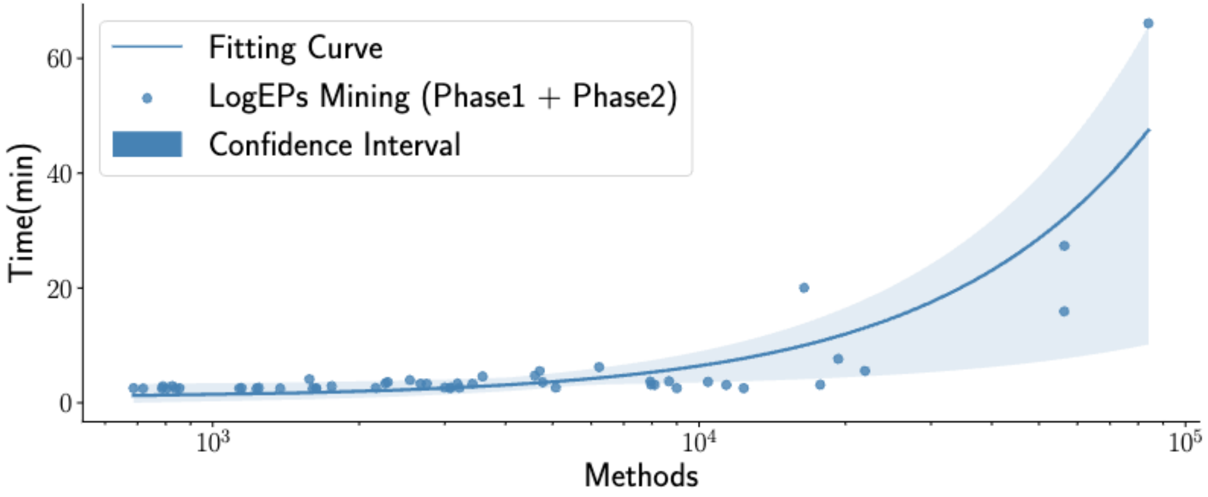
(a) Logging coverage histogram



(b) # Log event histogram

RQ2: Scalable?

- Efficient and scalable approach
  - ✓ Shortens generation time *(15x)*
  - ✓ Execution within *60* mins for *50* projects

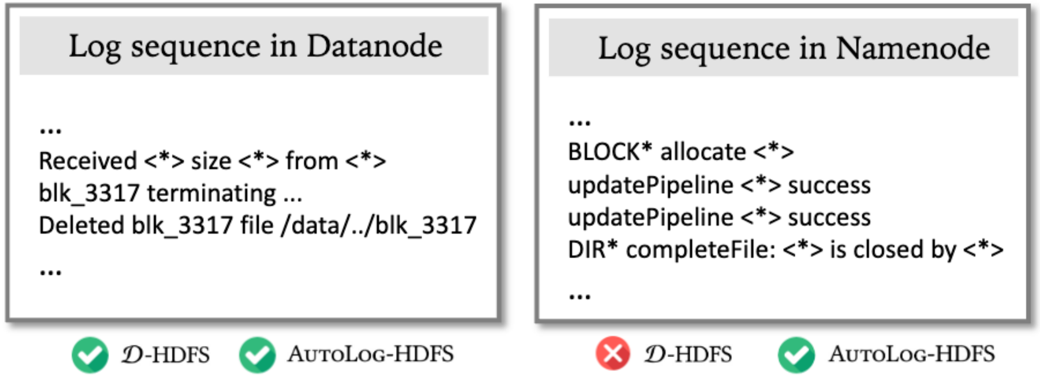


System	Dataset	# Message	Execution Time	# Messages/min (speed)	Acceleration (↑)
Hadoop	$\mathcal{D}$-Hadoop AUTOLOG-Hadoop	394,308 392,427	NA 3.41 hours	NA 1,918	–
HDFS	$\mathcal{D}$-HDFS AUTOLOG-HDFS	11,175,629 11,376,233	38.7 hours† 2.62 hours	4,813 72,367	15x
Zookeeper	$\mathcal{D}$-Zookeeper AUTOLOG-Zookeeper	207,820 211,425	26.7 days† 17 mins	6 12,436	2072x
Apache Storm Flink Kafka	AUTOLOG-Apache Storm AUTOLOG-Flink AUTOLOG-Kafka	1,001,245 1,003,416 1,002,629	1.28 hours 1.21 hours 39 mins	13,037 13,821 25,708	- - -

RQ3: Benefit anomaly detection?

- Benefit anomaly detectors
  - ✓ Consistently improve *(1.93%)* performance consistently

	Train set	$\mathcal{D}$			AutoLog		
Test set	Approach	P	R	F1	P	R	F1
$\mathcal{D}$	Transformer	0.889	0.904	0.896	0.892	0.996	**0.941**
	CNN	0.936	0.995	0.965	0.959	0.997	**0.978**
	LogRobust	0.942	0.994	**0.967**	0.947	0.988	**0.967**
AutoLog	Transformer		-†		0.723	0.755	0.739
	CNN		-†		0.697	0.790	0.741
	LogRobust		-†		0.673	0.875	0.761



Summary of Topic 3

- AutoLog: a code-guided log sequence synthesizer for anomaly detection
  - ✓ Motivation: Existing public log datasets fall short of comprehensive events, scalability, and flexibility.
  - ✓ Formulating the log sequence generation problem as an execution order acquisition task.
  - ✓ Applying program analysis to automatically simulate log sequences.

Outline

Log-driven automated SRE

Reactive Log analysis

[ICSE’23, ASE’23, ISSRE’23, ICSE’24, FSE’24, ISSTA’24]

- Topic1: A semantic-aware log parser for software operations [ICSE’23] (Chapter 3)
- Topic2: A log localizer for evolving software systems [ISSRE’23] (Chapter 4)
- Topic3: An automatic log sequence synthesizer for anomaly detection [ASE’23] (Chapter 5)

Proactive Log analysis

[FSE’24, TSE’24 Major]

- Topic4: An empirical study on automatic logging statement generator [TSE’24 Major] (Chapter 6)

- Conclusion and Future Work

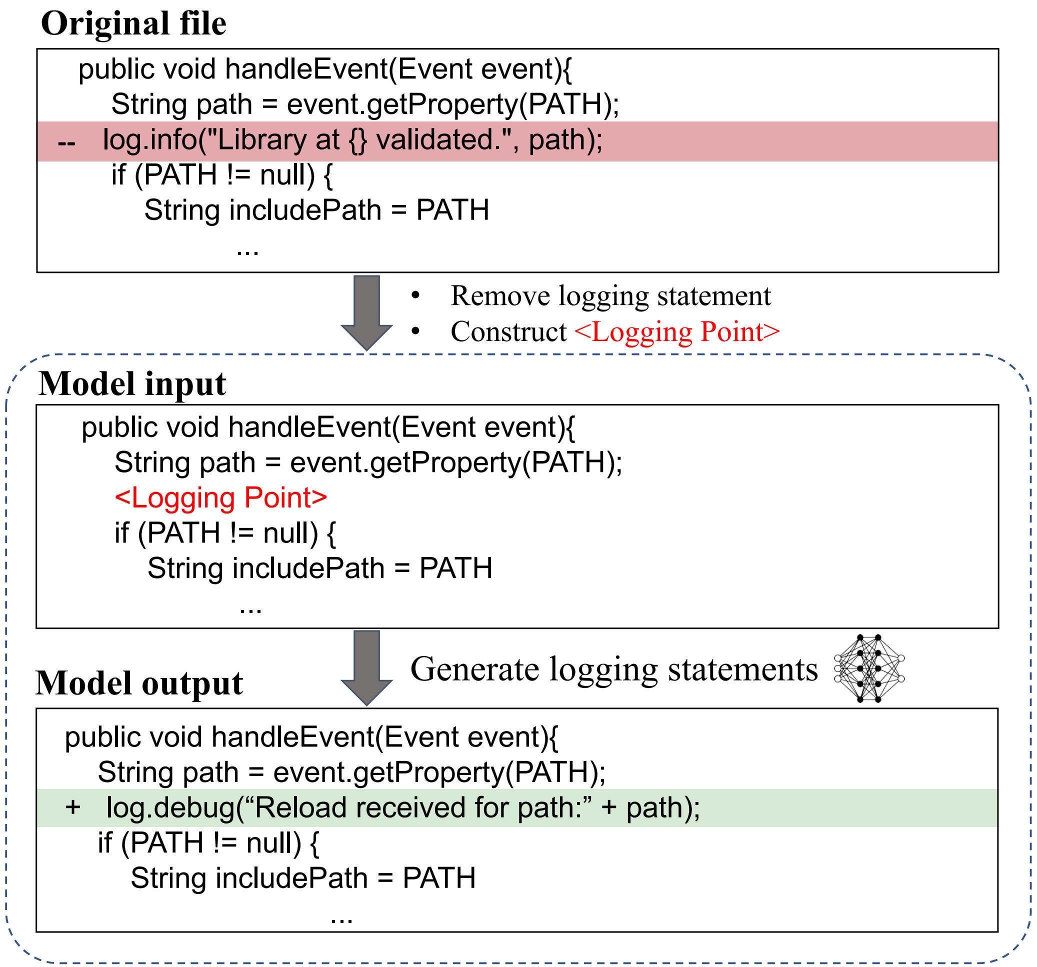
Logging statement generation

- Logging statements
  - Natural language descriptions
  - Program variables

Generating human language

Generating programs

Generating logging statements?



Study subjects

- 11 top-performing LLMs
  - ✓ General-purpose LLMs
  - ✓ Logging-specific LLMs
  - ✓ Code-based LLMs
- 3 traditional logging models

Model	Access	Description	Pre-trained corpus (Data size)	#Params	Year
General-purpose LLMs					
Davinci	API	Davinci is derived from InstructGPT [27] is an "instruct" model meant to generate texts with clear instructions. We access the Text-davinci-003 model by calling the official API from OpenAI.	-	175B	2022
ChatGPT	API	ChatGPT is an enhanced version of GPT-3 models [28], with improved conversational abilities achieved through reinforcement learning from human feedback [29]. It forms the core of the ChatGPT system [30]. We access the GPT3.5-turbo model by calling the official API from OpenAI.	-	175B	2022
Llama2	Model	Llama2 [31] is an open-sourced LLM trained on publicly available data and outperforms other open-source conversational models on most benchmarks. We deploy the Llama2-70B model provided by the authors.	Publicly available sources (2T tokens)	70B	2023
Logging-specific LLMs					
LANCE	Model	LANCE [15] accepts a method that needs one logging statement and outputs a proper logging statement in the right position in the code. It is built on the T5 model, which has been trained to inject proper logging statements. We re-implement it based on the replication package [32] provided by the authors.	Selected GitHub projects (6M methods)	60M	2022
Code-based LLMs					
InCoder	Model	InCoder [18] is a unified generative model trained on vast code benchmarks where code regions have been randomly masked. It thus can infill arbitrary code with bidirectional code context for challenging code-related tasks. We	GitHub, GitLab, StackOverflow (159GB code, 57GB StackOverflow)	6.7B	2022
			GitHub code (158.7B tokens)	13B	2022
			The Stack (1T tokens)	15.5B	2023
			Publicly available code (500B tokens)	34B	2023
			-	-	2022
			-	-	2021
			-	-	2022

Ingredient	Model	Description	#Params	Venue	Year
Logging levels	DeepLV	DeepLV [11] leverages syntactic context and message features of the logging statements extracted from the source code to make suggestions on choosing log levels by feeding all the information into a deep learning model. We reimplement the model based on the replication package provided by the authors*.	0.2M	ICSE	2021
Logging Variables	WhichVar	WhichVar [13] applies an RNN-based neural network with a self-attention mechanism to learn the representation of program tokens, then predicts whether each token should be logged through a binary classifier. We reimplement the model based on its paper due to missing code artifacts*.	40M†	TSE	2021
Logging Text	LoGenText-Plus	LoGenText-Plus [38] generates the logging texts by neural machine translation models (NMT). It first extracts a syntactic template of the target logging text by code analysis, then feeds such templates and source code into Transformer-based NMT models. We reproduce the model based on the replication package provided by the authors.	22M	TOSEM	2023

Experiment preparation

- LogBench-O
  - Crawling from GitHub
  - 2,420 files
  - 3,870 methods
  - 6,849 logging statements
- LogBench-T
  - Transforming from LogBench-O

- ✓ RQ1: How do different LLMs *perform* for logging statements generation?
- ✓ RQ2: How do LLMs compare to *conventional logging models* in logging ability?
- ✓ RQ3: How do the *prompts* for LLMs affect logging performance?
- ✓ RQ4: How do *external factors* influence the effectiveness in generating logging statements?
- ✓ RQ5: How do LLMs perform in logging *unseen code*?

Selected experiment results & Findings

- ✓ Existing models correctly predict levels for 74.3% of logging statements
- ✓ There is significant room for improvement in producing logging variables and logging texts.

Model	Logging Texts						
	BLEU-1	**BLEU-2**	**BLEU-4**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**	**Semantics Similarity**
General-purpose LLMs							
Davinci	0.288	0.211	0.138	0.295	0.127	0.286	0.617
ChatGPT	0.291	0.217	0.149	0.306	0.142	0.298	0.633
Llama2	0.235	0.168	0.102	0.264	0.116	0.261	0.569
Logging-specific LLMs							
LANCE†	0.306	0.236	0.167	0.162	0.078	0.162	0.347
Code-based LLMs							
InCoder	0.369	0.288	0.203	0.390	0.204	0.383	0.640
CodeGeex	0.330	0.248	0.160	0.339	0.149	0.333	0.598
TabNine	0.406	0.329	0.242	0.421	0.241	0.415	0.669
Copilot	**0.417**	**0.338**	0.244	**0.435**	0.247	**0.428**	**0.703**
CodeWhisperer	0.415	0.338	**0.249**	0.430	**0.248**	0.425	0.672
CodeLlama	0.216	0.146	0.089	0.258	0.103	0.251	0.546
StarCoder	0.353	0.278	0.195	0.378	0.195	0.369	0.593

† Since LANCE decides logging point and logging statements simultaneously, we only consider its generated logging statements with correct locations.

Selected experiment results & Findings

- Comment or non-comments?
  - ✓ Ignoring code comments results in an average 2.43% decrease in recommending logging texts.

Model	Logging Levels	Logging Variables	Logging Texts		
	AOD	F1	BLEU-4	ROUGE-L	Semantics Similarity
Davinci	0.834 (0.0%-)	0.587 (3.1%↓)	0.133 (3.6%↓)	0.283 (1.0%↓)	0.608 (1.5%↓)
ChatGPT	0.833 (0.2%↓)	0.592 (2.0%↓)	0.149 (0.0%-)	0.294 (1.3%↓)	0.614 (3.0%↓)
Llama2	0.789 (1.3%↓)	0.574 (1.2%↓)	0.099 (2.9%↓)	0.255 (2.3%↓)	0.544 (4.4%↓)
InCoder	0.789 (1.4%↓)	0.674 (1.2%↓)	0.201 (1.0%↓)	0.377 (9.2%↓)	0.622 (2.8%↓)
CodeGeex	0.848 (0.8%↓)	0.617 (6.1%↓)	0.149 (6.9%↓)	0.306 (8.1%↓)	0.578 (3.3%↓)
TabNine	0.876 (0.5%↓)	0.690 (1.1%↑)	0.239 (1.2%↓)	0.412 (0.7%↓)	0.655 (2.1%↓)
Copilot	**0.878** (0.5%↓)	0.696 (2.2%↓)	0.241 (1.2%↓)	**0.419** (2.1%↓)	**0.689** (2.0%↓)
CodeWhisperer	0.877 (0.7%↓)	**0.718** (0.7%↓)	**0.244** (2.0%↓)	0.418 (1.6%↓)	0.661 (1.6%↓)
CodeLlama	0.804 (1.2%↓)	0.581 (2.0%↓)	0.087 (2.2%↓)	0.247 (1.6%↓)	0.544 (0.3%↓)
StarCoder	0.823 (0.7%↓)	0.647 (0.9%↓)	0.193 (1.0%↓)	0.369 (2.4%↓)	0.591 (**0.3%**↓)
Avg.Δ	0.835 (0.8%↓)	0.638 (2.1%↓)	0.173 (2.2%↓)	0.338 (3.0%↓)	2.1%↓

Selected experiment results & Findings

- Function-level or file-level?
  - ✓ Incorporating file-level programming contexts leads to a great improvement
  - ✓ More helpful than comments

Model	Logging Levels	Logging Variables	Logging Texts		
	AOD	F1	BLEU-4	ROUGE-L	Semantics Similarity
Davinci	0.854 (2.6%↑)	0.638 (5.3%↑)	0.156 (13.0%↑)	0.318 (11.2%↑)	0.635 (2.9%↑)
ChatGPT	0.858 (2.8%↑)	0.650 (7.6%↑)	0.253 (51.5%↑)	0.389 (30.5%↑)	0.704 (11.2%↑)
Llama2	0.832 (4.1%↑)	0.617 (6.2%↑)	0.149 (46.1%↑)	0.392 (50.2%↑)	0.669 (17.6%↑)
InCoder	0.815 (1.9%↑)	0.745 (9.2%↑)	0.307 (51.2%↑)	0.521 (35.3%↑)	0.734 (11.7%↑)
CodeGeex	0.869 (1.6%↑)	0.696 (5.9%↑)	0.241 (50.6%↓)	0.395 (18.6%↑)	0.644 (7.7%↑)
TabNine	0.912 (3.6%↑)	0.767 (9.9%↑)	0.375 (55.0%↑)	0.530 (27.7%↑)	0.783 (17.0%↑)
Copilot	**0.916** (3.9%↑)	0.742 (4.2%↑)	0.346 (41.8%↑)	0.522 (22.0%↑)	**0.816** (16.1%↑)
CodeWhisperer	0.913 (3.6%↑)	**0.792** (9.6%↑)	**0.401** (61.0%↑)	**0.559** (31.5%↑)	0.811 (20.7%↑)
CodeLlama	0.817 (0.4%↑)	0.607 (2.4%↑)	0.144 (61.8%↑)	0.378 (50.6%↑)	0.642 (17.6%↑)
StarCoder	0.847 (2.2%↑)	0.714 (9.3%↑)	0.314 (61.0%↑)	0.517 (40.1%↑)	0.679 (14.5%↑)
Avg.Δ	2.7%↑	6.9%↑	49.3%↑	31.8%↑	13.7%↑

Summary of Topic 4

- An empirical study on LLM-powered logging statement generation
  - ✓ Motivation: To what extent can LLMs produce correct and complete logging statements for developers?
  - ✓ Two benchmarks for evaluating logging statement generation.
  - ✓ Eight findings and five implications
    - ✓ File-level context incorporation
    - ✓ Generalizability for unseen code

Outline

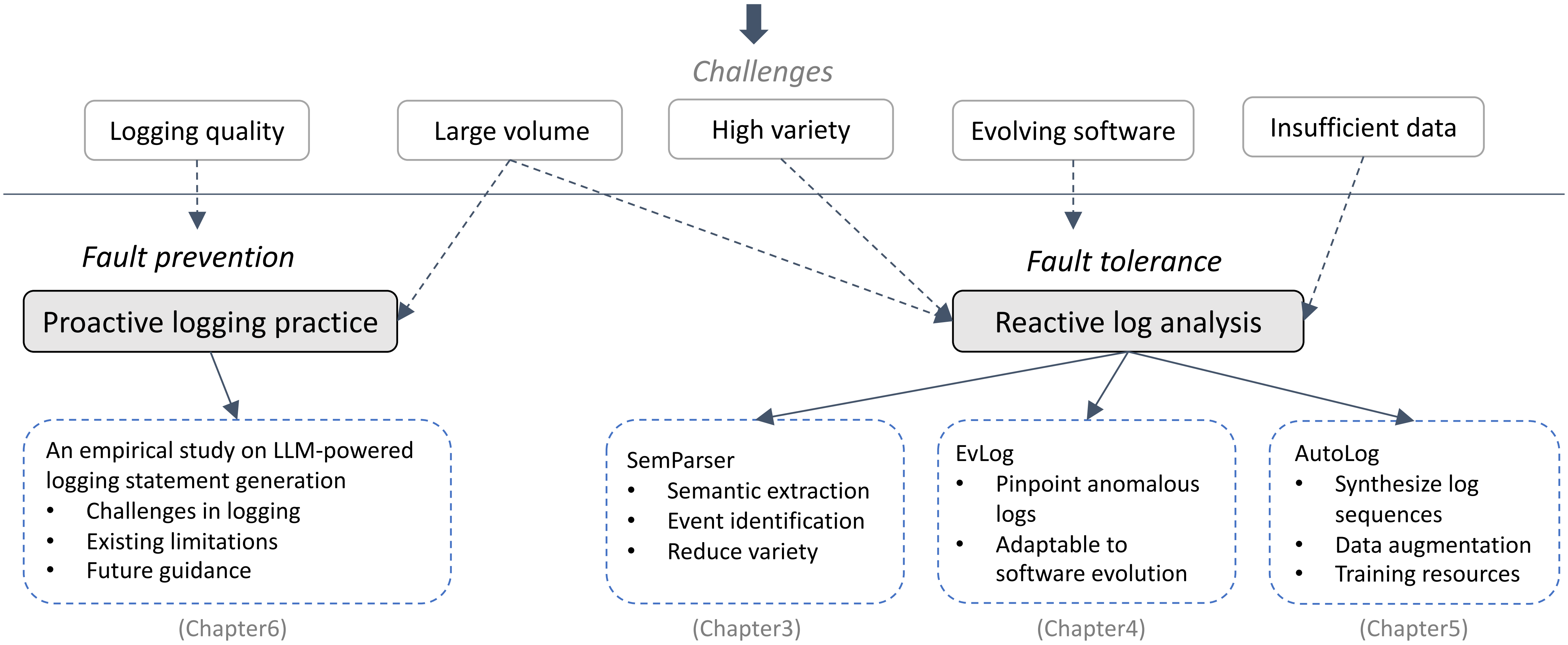
Log-driven automated SRE

- **Reactive Log analysis**
  [ICSE'23, ASE'23, ISSRE'23, ICSE'24, FSE'24, ISSTA'24]
  - Topic1: A semantic-aware log parser for software operations — [ICSE'23] (Chapter 3)
  - Topic2: A log localizer for evolving software systems — [ISSRE'23] (Chapter 4)
  - Topic3: An automatic log sequence synthesizer for anomaly detection — [ASE'23] (Chapter 5)
- **Proactive Logging practice**
  [FSE'24, TSE'24 Major]
  - Topic4: An empirical study on automatic logging statement generator — [TSE'24 Major] (Chapter 6)

- Conclusion and Future Work

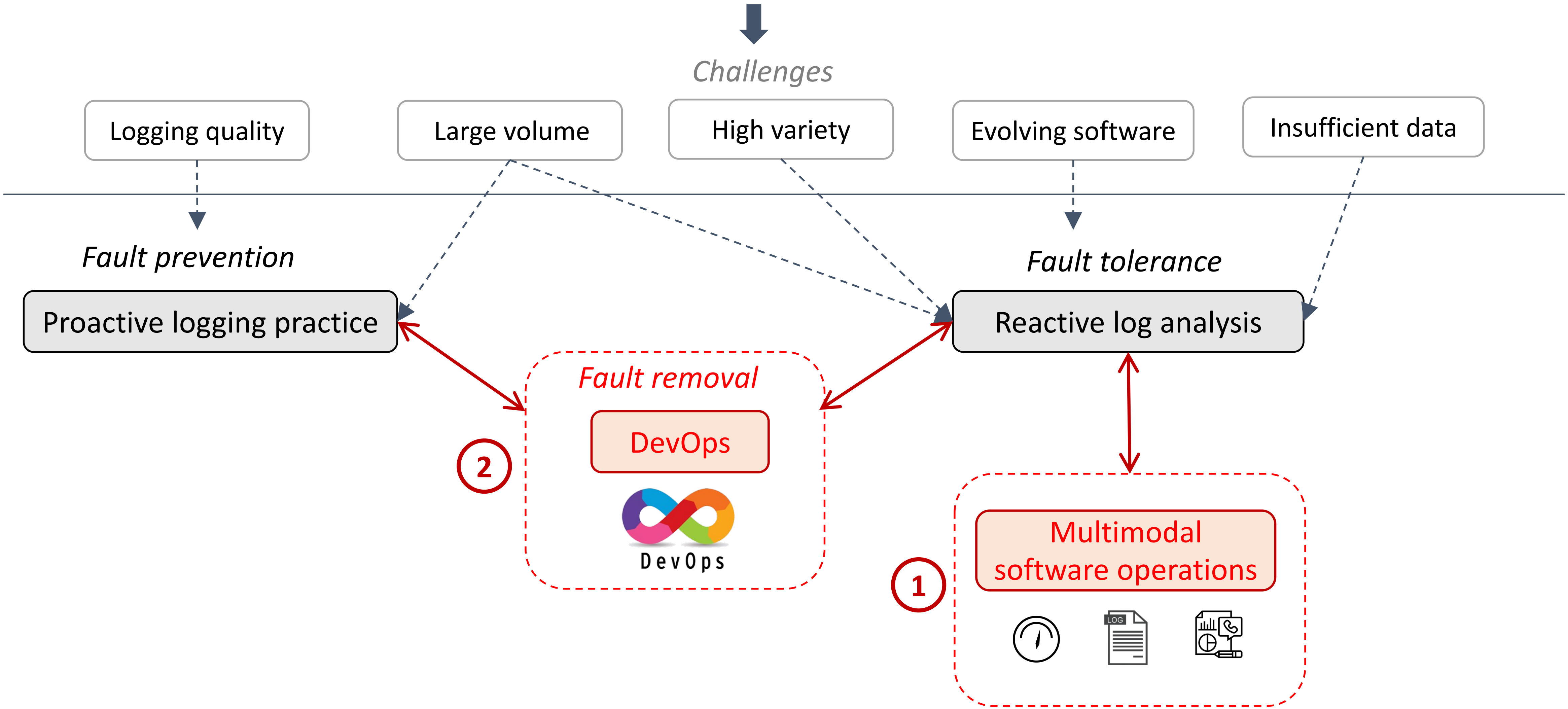
Conclusion

Log-driven automated software reliability engineering



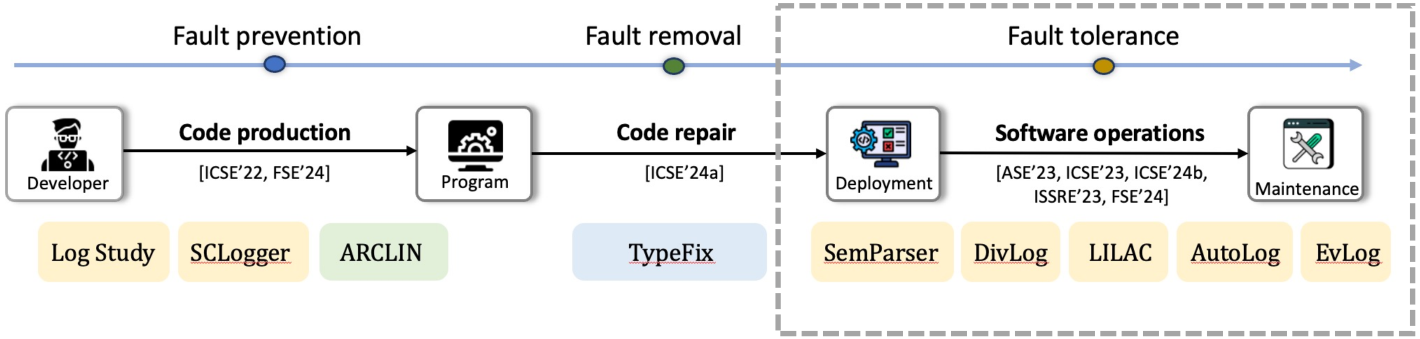
Future work

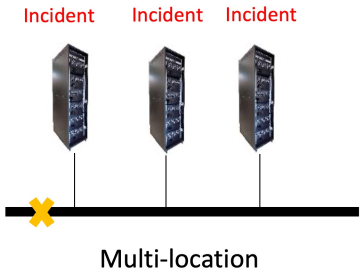
Log-driven automated software reliability engineering

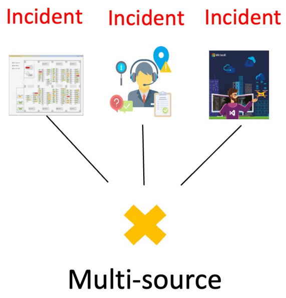


Multi-modal software operations

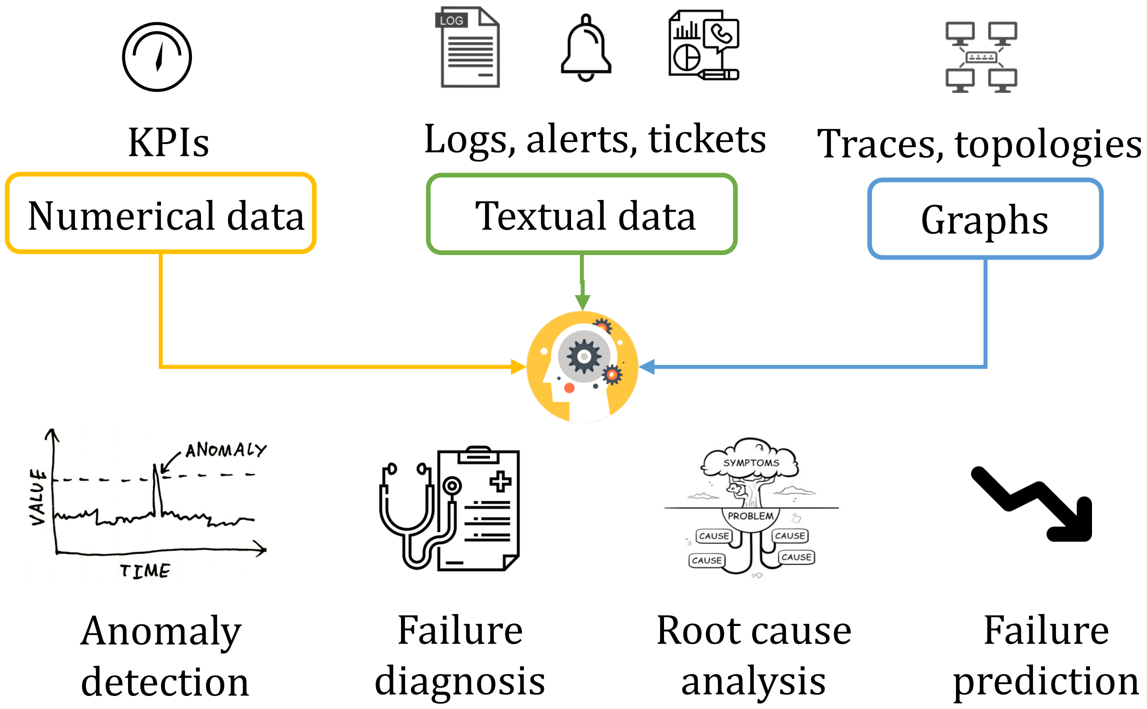
Reliable Operations



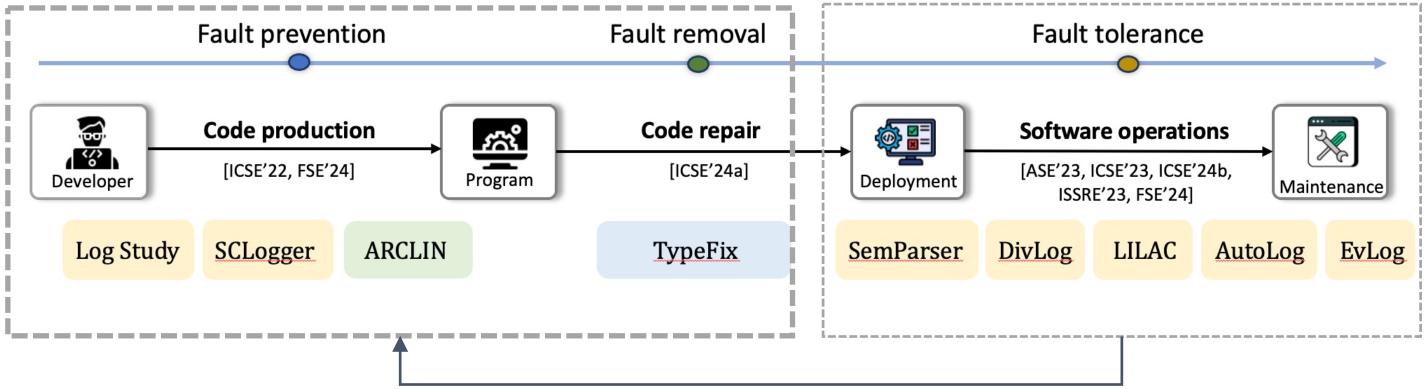




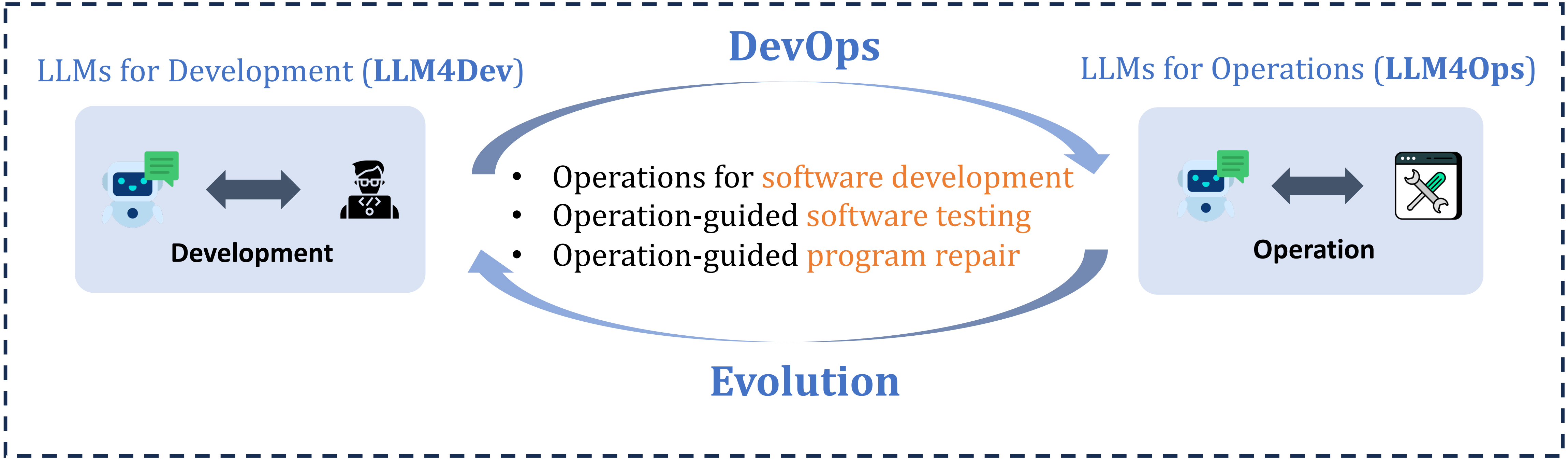
- Software becomes more complicated
- Incidents are highly-correlated
- But they are separately resolved



Operation-guided software development



Continuous Integration



Publications (1)

1. ARCLIN: Automated API Mention Resolution for Unformatted Texts
Yintong Huo, Yuxin Su, Hongming Zhang, Michael R. Lyu. *Proceedings of 44th International Conference on Software Engineering (ICSE), 2022*.
2. LogVM: Variable Semantics Miner for Log Messages
Yintong Huo, Yuxin Su, Michael R. Lyu. *Proceedings of 33rd International Symposium on Software Reliability Engineering (ISSRE), 2022*
3. SemParser: A Semantic Parser for Log Analytics
Yintong Huo, Yuxin Su, Baitong Li, Michael R. Lyu. *Proceedings of 44th International Conference on Software Engineering (ICSE), 2023*
4. EvLog: Identifying Anomalous Logs over Software Evolution
Yintong Huo, Cheryl Lee, Yuxin Su, Shiwen Shan, Jinyang Liu and Michael R. Lyu. *Proceedings of 34th IEEE International Symposium on Software Reliability Engineering (ISSRE), 2023*
5. AutoLog: A Log Sequence Synthesis Framework for Anomaly Detection
Yintong Huo#, Yichen Li#, Yuxin Su, Pinjia He, Zifan Xie, and Michael R. Lyu. *Proceedings of 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2023.*
6. Domain Knowledge Matters: Improving Prompts with Fix Templates for Repairing Python Type Errors
Yun Peng, Shuzheng Gao, Cuiyun Gao, **Yintong Huo**, Michael R. Lyu. *Proceedings of 44th International Conference on Software Engineering (ICSE), 2024*

Co-first author
* Corresponding author (mentorship)

Publications (2)

7. Enhancing LLM-based Coding Tools Through Native Integration of IDE-Derived Static Context
Yichen Li, Yun Peng, **Yintong Huo***, and Michael R. Lyu. *To appear in the IEEE/ACM International Conference on Software Engineering Workshop on Large Language Model for Code* (ICSE-LLM4Code), 2024

8. LILAC: Log Parsing using LLMs with Adaptive Parsing Cache
Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, **Yintong Huo**, Pinjie He, Jiazhen Gu, and Michael R. Lyu. *To appear in the ACM International Conference on the Foundations of Software Engineering* (FSE), 2024

9. Go Static: Contextualized Logging Statement Generation
Yichen Li, **Yintong Huo***, Renyi Zhong, Zhihan Jiang, Jinyang Liu, Junjie Huang, Jiazhen Gu, Pinjie He, and Michael R. Lyu. *To appear in the ACM International Conference on the Foundations of Software Engineering* (FSE), 2024

10. A Large-scale Evaluation for Log Parsing Techniques: How Far are We?
Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, **Yintong Huo**, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R. Lyu. *To appear in the ACM International Symposium on Software Testing and Analysis* (ISSTA), 2024

11. *(Preprint)* Exploring the Effectiveness of LLMs in Automated Logging Generation: An Empirical Study
Yichen Li#, **Yintong Huo**#, Zhihan Jiang, Renyi Zhong, Pinjia He, Yuxin Su, Lionel C. Briand, and Michael R. Lyu.

12. *(Preprint)* Automatically Generating UI Code from Screenshot: A Divide-and-Conquer-Based Approach
Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, **Yintong Huo***, and Michael R. Lyu.

Co-first author
* Corresponding author (mentorship)

Thank you
Q&A