BLARE: EXPLOITING STRUCTURE IN REGULAR EXPRESSION QUERIES

UW-Madison: Ling Zhang, Jignesh M. Patel

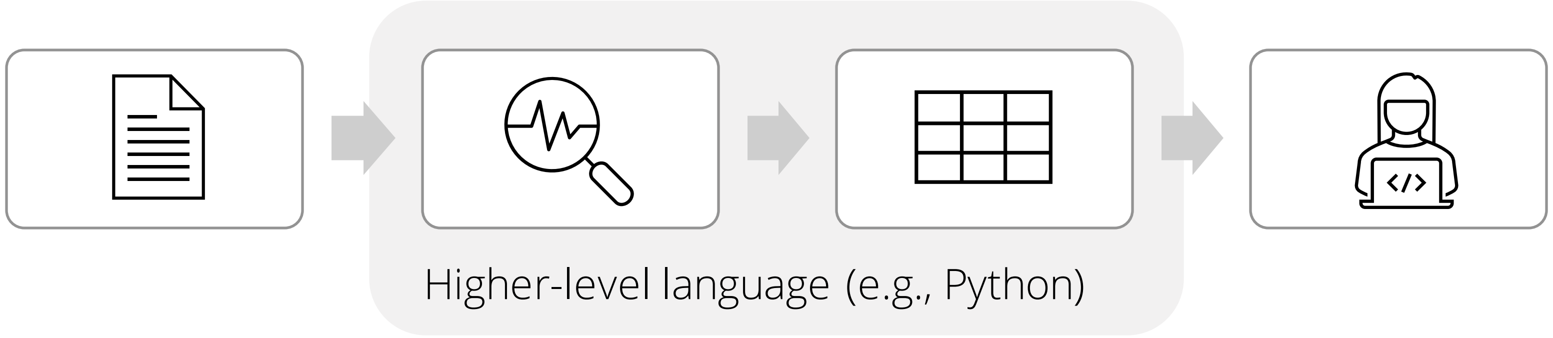
MS GSL: Shaleen Deep, Avrilia Floratou, Anja Gruenheid, Yiwen Zhu

INTRODUCTION

- Analyzing large volume of log data is crucial for large-scale system management

① Security, ② Customer Support, ③ Understanding system usage

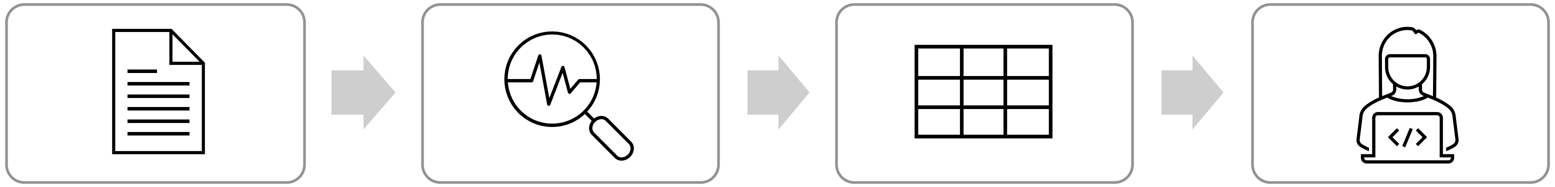
- Log analysis extract structured information from, schema-less, semi-structured logs
  - Unsuitable for relational DBMS
  - Done in ad-hoc data science approach



INTRODUCTION

- Identify the frequency and pattern of VMs' redeployment due to resizing within clusters in US-East-X region.
  - Obtain the relevant logs
  - Inspect sample logs to construct appropriate regular expression (regex)

    `replacing VM (VmId=([a-z0-9\-]+), VmName=us-east-X-([a-z0-9]+)-vm`
  - Extract the VM IDs using the regex
  - Deeper analysis of the specific VMs



Alice

PRIOR WORKS

- State-of-art regex evaluation under the hood
- Existing state-of-art regex libraries in several analytics and RDBMSs
  - Google's **RE2** is used in spam filtering, Google Sheets, MS Azure Data Explorer, etc.
  - **PCRE2** is widely used in intrusion detection, packet filtering, and spam filtering
  - MySQL uses **ICU Regex** for Unicode regex support
  - C++ version Lucene and C++ standard library uses **Boost Regex**

REGEX EVAL BASICS

- NFA
  - Each character requires O(m) memory lookups, where m = # states in automata
- DFA
  - Special case of NFA when input can transit to only one state
  - O(1) lookup per character, but larger state graph compared to NFA
- Existing Optimization Example

  Prefix literal
  - Prefiltering some irrelevant inputs `Read on \"(.+)\" failed: (.+)`

OBSERVATION:

Regex engines use DFAs/NFAs and need to do bookkeeping.

Expensive even for the simplest task of string matching.

REGEX CHARACTERISTICS

- 14.5 million public notebooks on GitHub authored between 2017-2020
  - 35% out of 200, 000 unique regexes contain at least 1 literal
- Our collected workloads

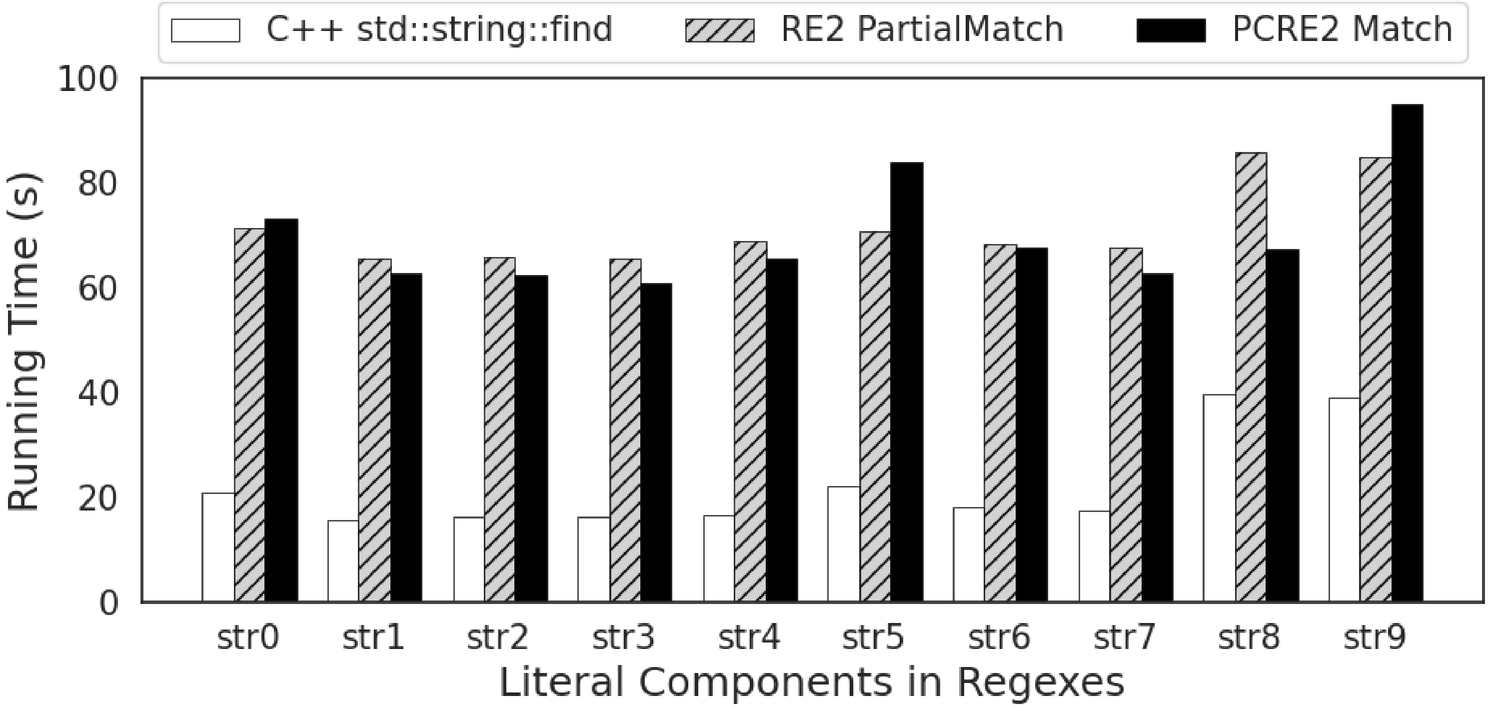
		SQL Server	*Azure Data Explorer*	*US-Accident*
# literal per regex	mean	3.2	1.4	1.8
	median	3	1	2
Mean # char in literal		39.9	12.2	5.1

OBSERVATION:
Most regexes contain literal components;
Regexes used in log analysis contain long literals and simple regex components.

INSIGHT

- Move **string matching** related computation outside regex engine

  Gap between evaluating string matching using a regex library vs string matching in code is ~3x



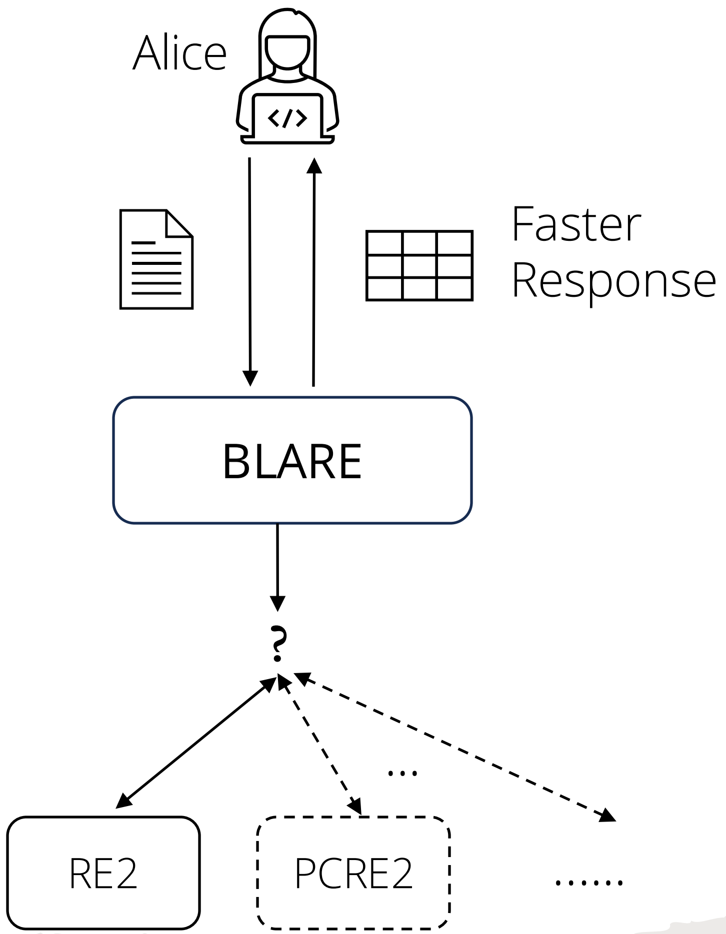
BLARE: OUR CONTRIBUTIONS
A FRAMEWORK FOR REGEX EVALUATION

Framework Design

- Implemented as a module on top of a regex engine that is used as a “blackbox”
  - (R1) engine-agnostic
  - (R3) no prior statistics needed
  - (R4) no specialized SW & HW dependence
- BLARE uses lightweight mechanisms to identify whether our new evaluation strategy is better than running the entire regex as-is on the regex engine
  - (R2) no large regressions

Framework Performance

- We implement BLARE on 4 regex engines (RE2, PCRE2, ICU Regex, Boost Regex)
  - 1.6x to 168x improvement over two production workloads and an open-source workload



REGEX DECOMPOSITION

- Split regex R to (prefix S suffix) where prefix and suffix are strings of literals
- R = `replacing VM (VmId=([a-z0-9\-]+), VmName=us-east-X-([a-z0-9]+)-vm`

  Prefix Regex Suffix
- We call **3-way-split** of the regex

  X-way-split: split a regex to a maximum of X literal-regex alternating components
- Recursively continue decomposing the regex gives us the **multi-way-split**.
- R = `replacing VM (VmId=([a-z0-9\-]+), VmName=us-east-X-([a-z0-9]+)-vm`

  Literal0 Regex0 Literal1 Regex1 Literal2

WHAT SPLITTING STRATEGY IS BEST?

- Cost Model (k: number of literals)

$$\mathtt{SMCost}(r,k) = \underbrace{|\ell| + \sum_{i=1}^{k} f \cdot \underbrace{\left(c \cdot (1-\sigma_i) \cdot \prod_{j=1}^{i-1} \sigma_j\right.}_{\text{part one}} + \underbrace{\left. i \cdot \prod_{j=1}^{i} \sigma_j \cdot \mathrm{lsize}\right)}_{\text{part two}}}_{\text{string matching cost}} + \underbrace{\underbrace{2 \cdot \sigma \cdot \sum_{i=1}^{k-1} |\ell'_i|}_{\text{substring extraction cost}} + \underbrace{\Theta(r) \cdot \sum_{i=1}^{k-1} |\ell'_i| \cdot \sigma}_{\text{running on engine}}}_{\text{total regex evaluation cost}}$$

1. The more we decompose a regex, the higher the string-matching cost
2. The lower the selectivity's of the string literals, the lesser the advantage of doing regex decomposition + higher the substring extraction cost
3. If string literals are selective, we often get to ignore the log line at early stage

OBSERVATION: k=2 (3-Way-Split) is most beneficial in majority of the time.

WHAT SPLITTING STRATEGY IS BEST?

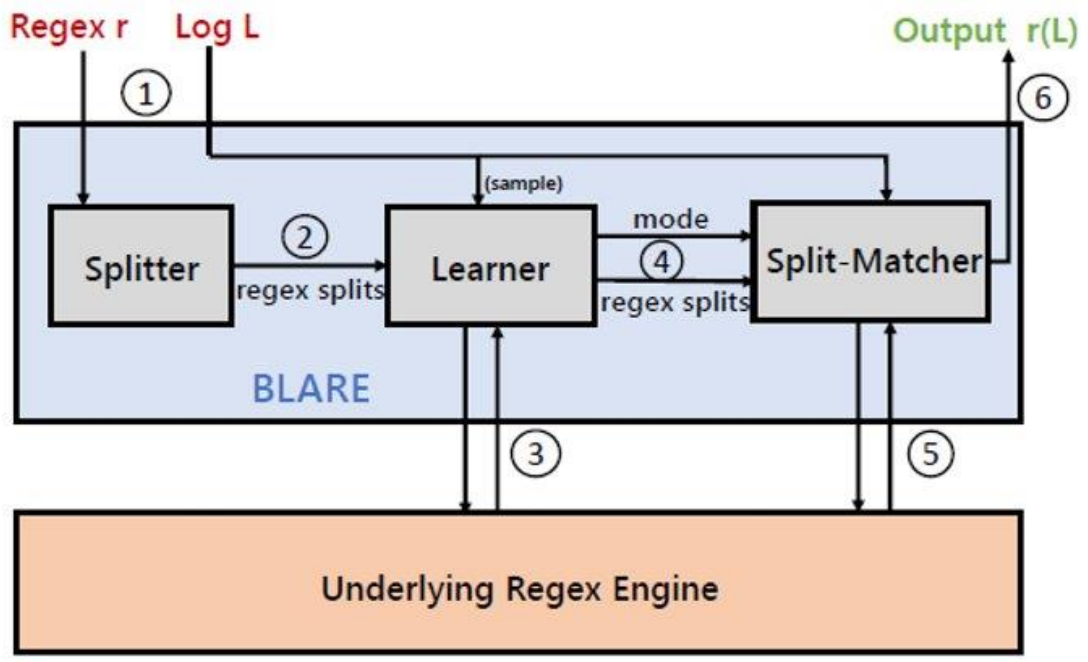
- Experimental Verification

Regexes	Run Time (s)				
	$k = 1$	$k = 2$	$k = 4$	$k = 6$	$k > 6$
A	2.03	1.99	2.27	2.21	2.22
B	2.03	1.99	2.27	2.21	2.22
C	2.10	2.04	2.30	2.24	2.25
D	2.00	1.94	2.22	2.16	2.20
E	1.84	1.80	2.05	2.00	2.00
F	2.09	2.03	2.32	2.24	2.26
G	2.08	2.02	2.31	2.24	2.26
H	2.40	1.99	2.18	2.08	1.99

NOTE: regex-specific best strategy may still vary depending on engine & selectivity.

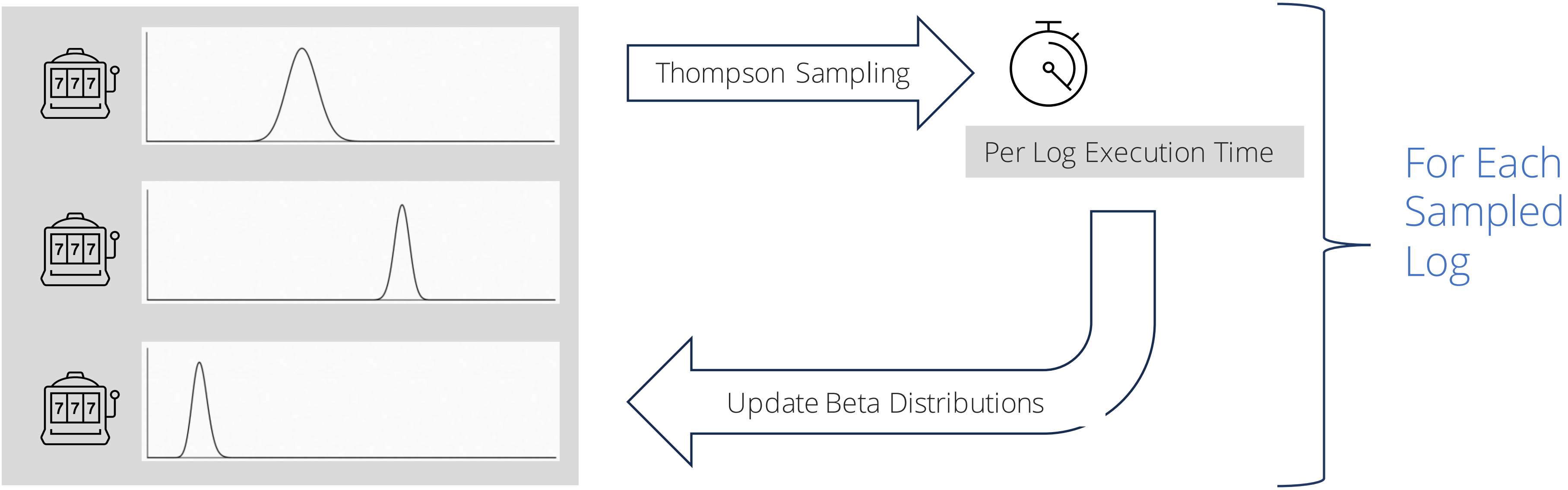
BLARE ARCHITECTURE

- Splitter
  - Construct different regex decompositions of interest
- Learner
  - Use a learning component to identify which split is likely to give the best performance.
- Split-Matcher
  - Execute the best strategy identified for the regex



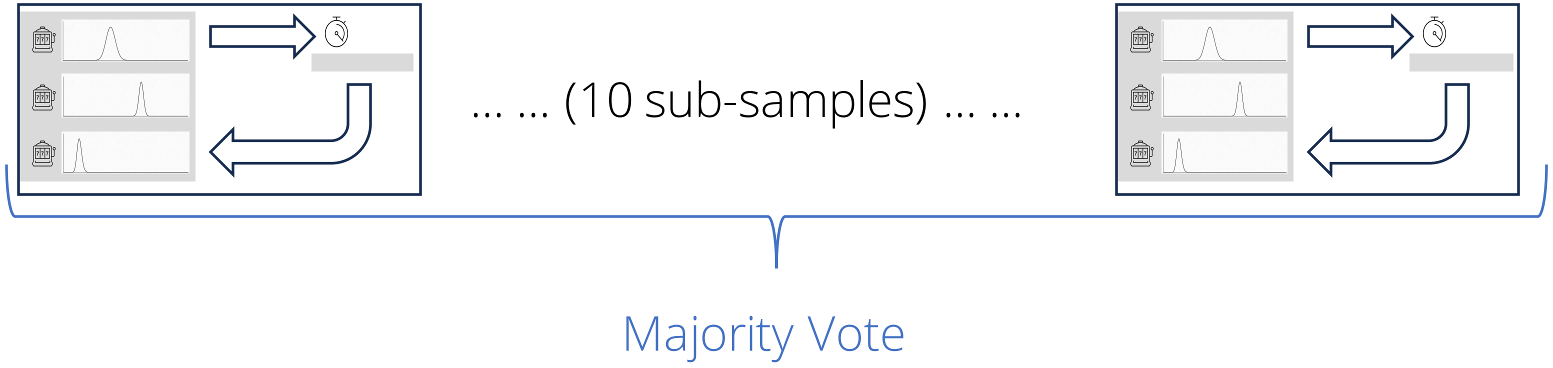
LEARNING

- Learn to choose strategy *on the fly*. No prior statistics needed
- Multi-Armed Bandit (MAB)
  - 3 arms: 3-Way-Split, Multi-Way-Split, Direct
  - Thompson Sampling addressing the exploration-exploitation dilemma
  - Ensemble Method dealing with noisy measured data



LEARNING

- Learn to choose strategy *on the fly*. No prior statistics needed
- Multi-Armed Bandit (MAB)
  - 3 arms: 3-Way-Split, Multi-Way-Split, Direct
  - Thompson Sampling addressing the exploration-exploitation dilemma
  - Ensemble Method dealing with noisy measured data



DESIGN CONSIDERATIONS

Extensibility & Simplicity

- Implement BLARE as a layer on top, calling underlying regex engine
- Easy to adopt, benefit from advancement of underlying engine
- Easily extended by adding arm(s)
- Small codebase (<1000 LOC) aids explainability

Minimize Learning Overhead

- Since learning is an overhead (proportional in the number of strategies), we deliberately keep the number of modes in BLARE to be small.
- Early stopping in MAB
- Thresholding number of log fed to learner

Prefix and Suffix Sizes

- Since selectivity is most important, and it is not directly connected to the length of the literal, we do not discard short prefix/suffix

EXPERIMENTAL EVALUATION

Experiment Setup

- We use 4 SOTA regex libraries: RE2, PCRE2, Boost Regex, and ICU Regex.
- All experiments on a machine running Intel Xeon@2.8GHz, 256 GB RAM.
- Sample size for learning is max {0.001% of the log, 200 lines}

Query Result Reporting

- Regex matching is performed 10 times and we record the trimmed mean.
- We store the extracted content of the <u>first match</u> result in a local variable.

Workloads

- 132 regexes used for SQL Server log analysis over 100M+ log lines
- 18 regexes used for log analysis over 890M+ log lines sourced from Kusto
- Open-source datasets on traffic accident in US with 2.8M+ log lines and 4 relevant regex

EXPERIMENTAL EVALUATION
- OVERALL PERFORMANCE

- Speedup obtained from BLARE w.r.t. running the workload on the underlying engine

	SQL Server	*Azure Data Explorer*	*US-Accident*
Google RE2	3.7x	3.3x	1.6x
PCRE2	3.2x	3.1x	168.3x
ICU Regex	1.6x	--	61.7x
Boost Regex	7.9x	4.9x	3.4x

- Nearly every query experienced a performance improvement across all the engines
- For queries that did not, the gap to the best strategy was < 2%

EXPERIMENTAL EVALUATION

- LEARNING OVERHEAD

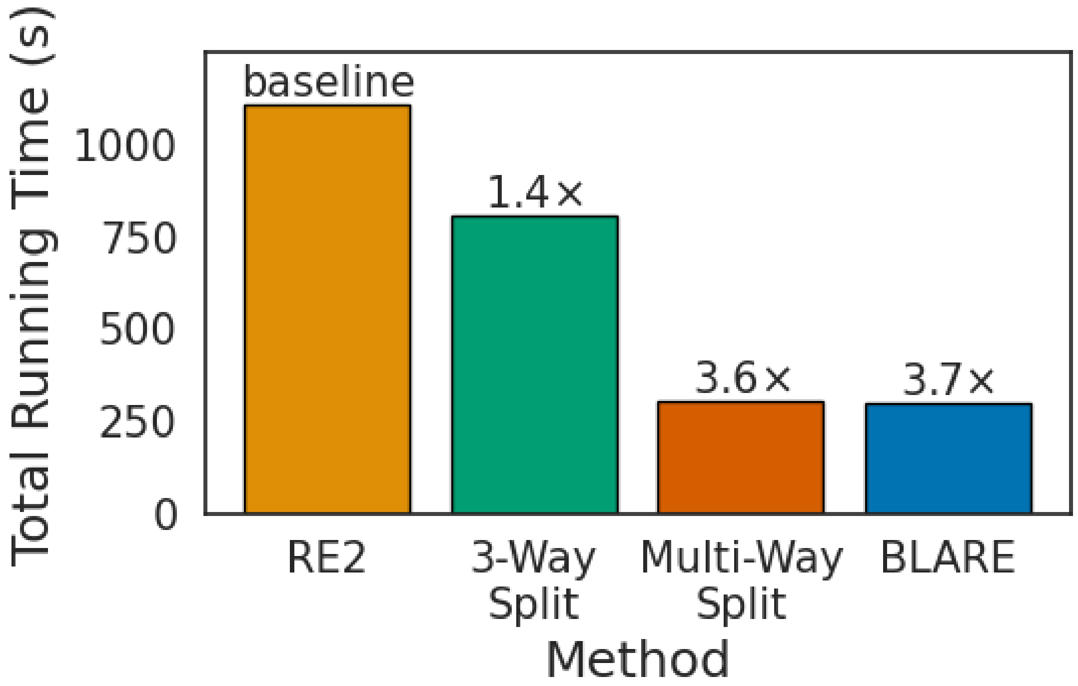
- Mean % of time spent in learning in BLARE

	SQL Server	*Azure Data Explorer*	*US-Accident*
BLARE-RE2	5.1%	6.7%	16.5%
BLARE-PCRE2	9.1%	--	23.8%
BLARE-ICU Regex	--	8.1%	28.1%
BLARE-Boost Regex	10.7%	6.1%	27.4%

- Note: US-Accident is consistently higher because the log size is small, lower threshold number of logs for learning takes a larger proportion compared to other workloads
- The cost of learning is justified by the overall gains made by BLARE

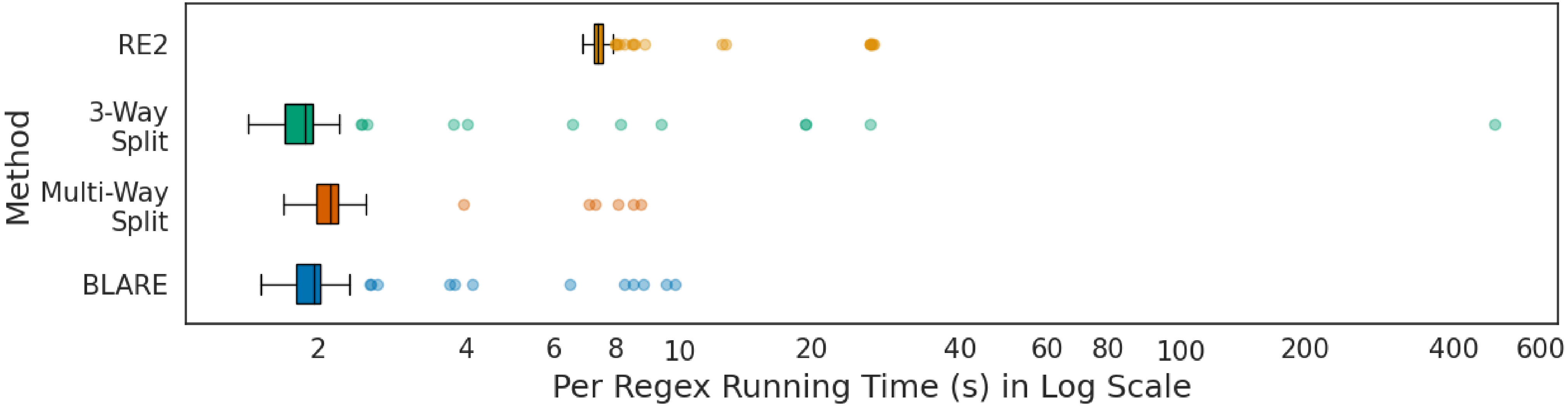
EXPERIMENTAL EVALUATION
- SPLITTING STRATEGIES

- **Overall performance** and distribution of per-regex running time for BLARE vs. 3-Way-Split vs. Multi-Way-Split
- Using **Google-RE2 on SQL Server**



EXPERIMENTAL EVALUATION
- SPLITTING STRATEGIES

- Overall performance and **distribution of per-regex running time** for BLARE vs. 3-Way-Split vs. Multi-Way-Split
- Using **Google-RE2 on SQL Server**

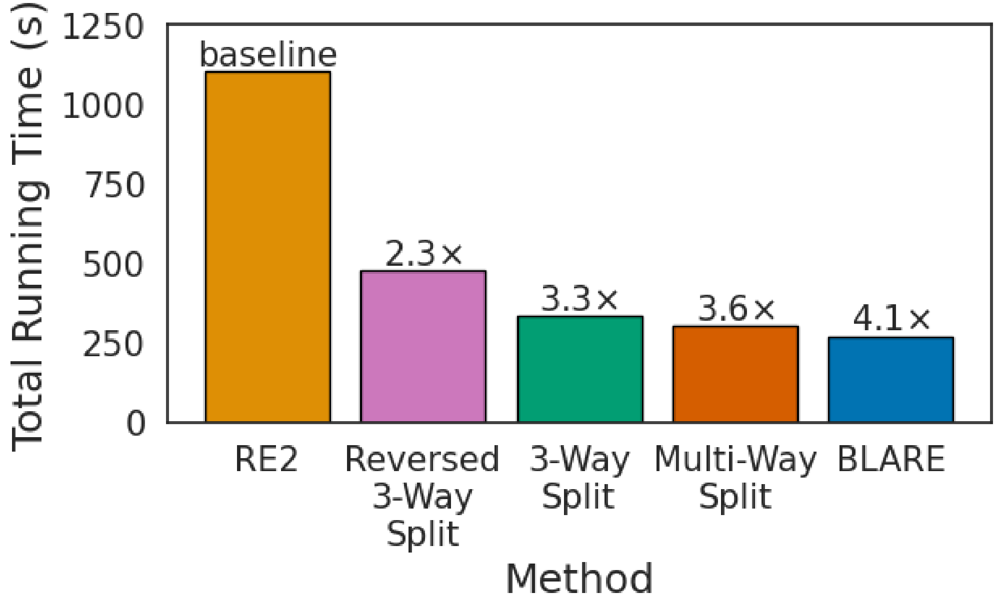


EXPERIMENTAL EVALUATION
- OTHER QUERIES

- **Overall performance** in terms of workload running time in seconds for 3 types of queries.
- Using **Google-RE2 on SQL Server**

	Running Time (s)		
	FirstMatch	*CountAllMatches*	*LongestMatch*
Google RE2	1105.7	1148.7	1128.5
BLARE - RE2	301.0	299.8	306.1
Improvement	**3.67x**	**3.83x**	**3.68x**

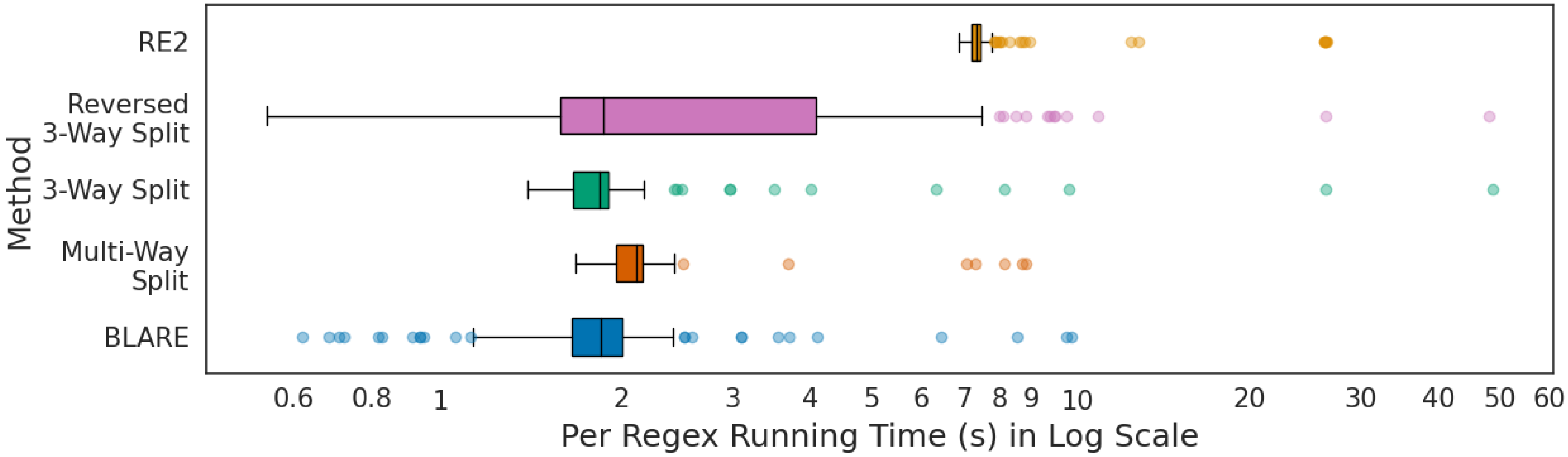
EXPERIMENTAL EVALUATION
- EXTENSIBILITY



- Add an additional **Reversed 3-Way-Split arm**
  - Instead of doing string containment checks left to right, we can also add another strategy that does right to left
- **Overall performance** and distribution of per-regex running time for BLARE vs. 3-Way-Split vs. Multi-Way-Split vs. Reversed 3-Way-Split
- Using **Google-RE2 on SqlServer**

EXPERIMENTAL EVALUATION
- EXTENSIBILITY

- Add an additional **Reversed 3-Way-Split arm**
  - Instead of doing string containment checks left to right, we can also add another strategy that does right to left
- Overall performance and **distribution of per-regex running time** for BLARE vs. 3-Way-Split vs. Multi-Way-Split vs. Reversed 3-Way-Split
- Using **Google-RE2 on SQL Server**

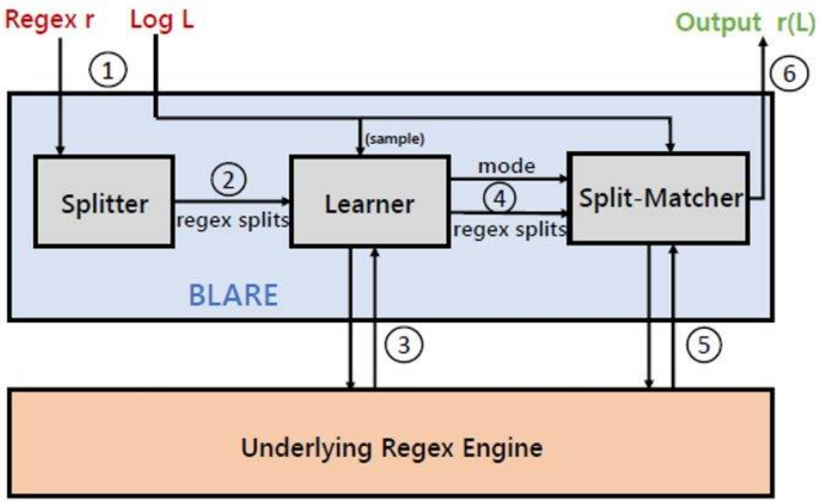


CONCLUSION

- We presented BLARE, a framework for faster regex evaluation for large volume log analysis.
- BLARE is engine-agnostic, does not make any assumptions on the hardware, statistics, etc.
- Experimental evaluation demonstrates speedups ranging from 1.6x to 168x over real-world datasets and workloads.
- Code: github.com/mush-zhang/Blare
- Future work:
  - Incorporate indexing, light-weight statistics collection, add more evaluation operators and build a regex query optimizer.

SUMMARY

BLARE ARCHITECTURE



BLARE PROPERTIES

(R.1) BLARE is engine-agnostic

(R.2) BLARE is extensible with no long-term dependency on specialized hardware or software.

(R.3) BLARE introduces no large regressions for any specific query in the workload

(R.4) BLARE requires no prior statistics or catalogs about the workloads