Splash: a Semi Structured Parser for Logs Assisted by Human Feedback

Sichao Yang

Splash: a Semi Structured Parser for Logs Assisted by Human Feedback

Sichao Yang[1], Ye Yang[1]

X-Epic, Chengdu, China
sichaoy@x-epic.com

Abstract. During the runtime of a program, system statuses are normally documented in logs. Log parsing is a crucial initial step for subsequent analysis. While existing parsers have shown notable enhancements in accuracy when applied to public datasets, their performance is hindered notably when deployed in real-world settings due to two primary challenges. Firstly, the accuracy of these parsers is heavily reliant on hyperparameters, the optimal values of which vary significantly across different datasets. Secondly, the continuous software evolution along with its logging statements results in a rapid degradation of parsing accuracy after parser development. Although various mitigation strategies have been proposed, their efficacy remains limited. We propose a human-in-the-loop framework, named Splash, which integrates human expertise with data-driven methodologies. It operates on top of set of carefully designed parsing models and aims to minimize human effort by self-identifying potential parsing errors and actively querying for human's assistance. Additionally, it employs proxy measures to monitor performance degradation and can request human assistance when necessary. Experimental evaluations conducted on 16 benchmark datasets demonstrate that our model outperforms all baselines in terms of accuracy. Furthermore, its effectiveness in mitigating model deterioration is exemplified through experiments conducted on a newly labeled dataset simulating a dynamic data environment.

Keywords: Log parsing · Human-in-the-loop · Active learning

1 Introduction

Logs play a pivotal role in documenting crucial system events and runtime status within the realm of software. The widespread adoption of log analysis techniques, which leverage data mining methods to glean insights into system behaviors, is evident. Research encompasses various areas such as anomaly detection, user behavior analysis, failure prediction, log compression, among others [1]. These techniques spread further into hardware design domain recently, including failure triage, root cause tracing [2], and visualization of system event flow [3]. The majority of data mining models utilized in the analysis require structured input, like an event list or a matrix. However, log messages typically present in an unstructured format. Consequently, the demand for log parsing techniques capable

of converting semi-structured logs into structured data is substantial and has been a subject of extensive study in recent years.

Log parsing involves extracting log templates from raw log messages. Logs are created from logging statements, as illustrated in Fig. 1. Each log typically consists of a free-text body and a structured header containing fields such as timestamp, verbosity level, and component name. The body of a log message comprises two main components: constants and variables. Only variables dynamically change based on the system's status during runtime. In the example depicted in Fig. 1, the body of each log message is transformed into a template where constants are retained, and variables are substituted with wildcard tokens (<*>). Each template corresponds to a unique event ID.

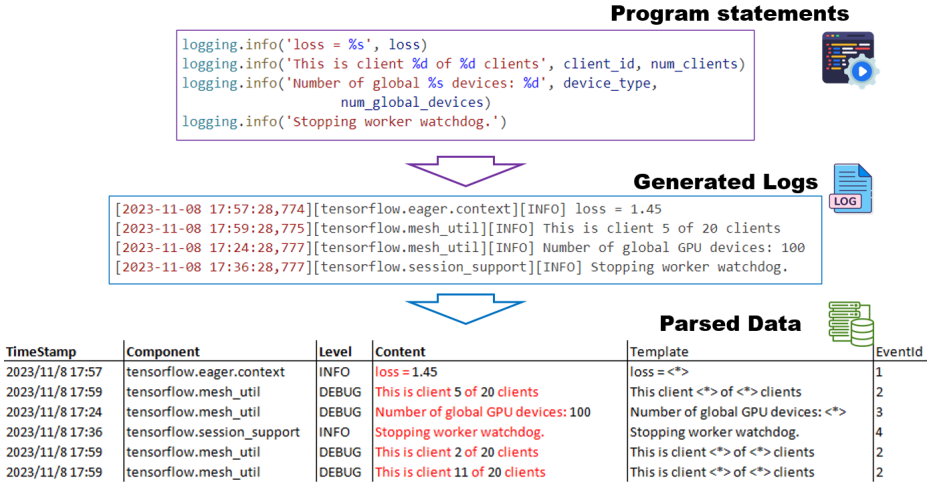

Fig. 1. Log parsing process: the top box contains printing statements written in python, the middle box contains the log messages recorded from runtime and the bottom shows the output from a typical log parser

Log parsing has traditionally relied on text regularization as a primary method. However, this approach has become increasingly inadequate for modern systems that undergo frequent updates, necessitating substantial manual effort to review and revise regularization sets. Moreover, it requires interdisciplinary collaboration between engineering teams, which is prone to errors. The dependence on customized regularization sets also hinders tool suppliers' efforts to develop automated log analyzers that could better serve their customers, as illustrated in Fig. 2.

Proposals have emerged for methods that automatically generate regular expressions through source code analysis [4]. Yet, the diverse logging formats, tools and source code accessibility render this direction more challenging than initially perceived. Consequently, recent studies have shifted focus towards data-driven

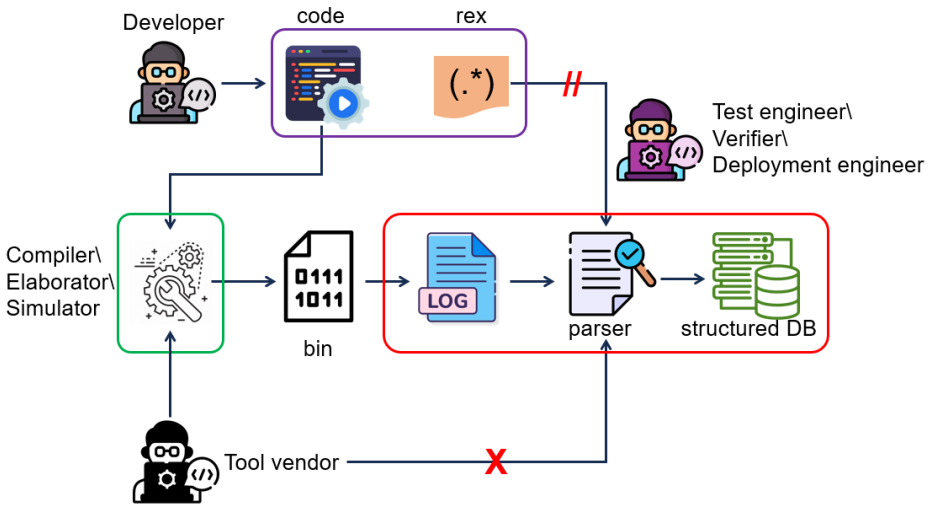

Fig. 2. Limitation on parsing with regular expression

approaches. The parsers can be classified into two categories: deep learning-based and machine learning-based methods. Although deep learning-based methods like neural-networks in [6] [7] and large-language-model used in [8] have shown significant improvements in accuracy, particularly at the token level (defined in section 2). The substantial requirements for extensive training data, labeling, and computational time still hinder the widespread application of these models in industrial settings. Machine learning-based approaches include n-gram frequency [9] matching, frequent itemset mining [10], tree [5], hierarchical clustering [11], etc..

Although many the aforementioned log parsers have shown promising accuracy on public log datasets [12], they face several challenges when applied in practical settings. Firstly, the reported parsing accuracy for each model largely depends on the selection of hyperparameters. Therefore, there is a strong implicit assumption that sufficient labels are available for hyperparameter tuning. This is an unrealistic assumption, particularly when deployed on-premise where the data collection time window is very limited. Additionally, the labeling process demands comprehensive program knowledge and is labor-intensive and error-prone. Thus, a good cold-start accuracy is often a necessity rather than an option. Secondly, log parsers require continuous updates as the log-generating system evolves. The ability to adaptively adjust hyperparameters is essential to sustain performance. Unfortunately, the exsisting process of fine-tuning log parsers is intricate and resource-intensive, involving meticulous adjustments of hyperparameters based on a sizable validation set.

Considering these challenges, we are motivated to develop a novel log parser, named Splash, aimed at enhancing the accuracy and robustness of existing parsers while minizing human assistance effort. Specifically, in the Semi-structured

Parser for Logs Assisted by Human feedback (Splash), we introduce a Human-in-the-Loop (HITL) framework where Splash selects multiple candidates with auxiliary information from log or template examples to actively query human feedback for refining its initial predictions. We also propose a model-human co-learning strategy for scenarios where labels are unavailable, significantly enhancing the resilience of existing parsers. Finally, we introduce a proxy metric to monitor changes in model performance over time in the absence of labels. By tracking this metric and automatically triggering HITL interventions, model deterioration can be effectively prevented.

Extensive evaluations on Splash was conducted on 16 public datasets from [12]. With minimum number of human queries, Splash achieves an average parsing accuracy exceeding 0.9. Upon enabling advanced strategy of feedback iterations, the accuracy of Splash can be further enhanced to 0.98 surpassing the performance of existing state-of-the-art log parsers. All the above interventions are based on a few binary questions, demonstrating high efficiency in terms of human effort cost. Furthermore, with a newly labeled dataset, we simulated the log evolution environment and found that the gradual decline in parsing accuracy can be mitigated through smart triggering on human intervention.

In summary, the contributions of this paper are as follows:

- The design of Splash, an effective and efficient log parser that incorporates user feedback for model tuning and label collection.
- The design of proxy metric, which helps to monitor model performance when no label available and enables smart triggering on human intervention.
- The evaluation of Splash on 16 public log datasets demonstrating its superiority in terms of accuracy and robustness compared to existing state-of-the-art log parsers.

This paper is organized as follows: Section II provides formal definitions on concepts describing the problem. Section III provides motivation on the design of a new parser by empirical experiments. Section IV the overview on Splash is presented. Section V conducts experiments for evaluation and finally, Section VI concludes the paper.

2 Problem Definition

In this section, we provide a formal description on the log parsing problem.

Event sequence A log dataset is composed of line of logs, each line can be seen as a realized event e from a particular logging statement, together they form a sequence of events, E := (e_1, e_2, ..., e_m), where m is the length of log messages.

Token sequence As each event is a line of string, to better represent it, we break it into sequence of tokens via tokenization function g, $g(e) \rightarrow T$, T := (t_1, t_1, ..., t_n), where n is the length of the token sequence. Function g uses a set of delimiters D to split the input iteratively and output T. The default delimiter in D is the space.

Log template Given a token sequence, it could consists two types of token, namely constant s and variable v. The constant are fixed, whereas the variables can be realized to different values. Define a labeling function L which takes t_i as input and output the token type: $L(t_i) = x_i, x_i \in \{s, v\}$. The variable tokens are then replaced by wildcard token "*" and token sequence is converted to log template Y, Y := (y_1, y_2, ..., y_n), $x_i == s$? $y_i = t_i : y_i = *$. All templates form set S.

Log parsing Given log dataset E, delimiter set D, template set S and similarity measure ϕ, log parsing refers to the process of clustering the events in E into groups and generating a template for each group, ensuring that for each e_i and its corresponding template Y_j satisfies the relation: $\phi_D(e_i, Y_j) \geq \epsilon \wedge j = argmax_{Y \in S} \phi_D(e_i, Y)$. where $\phi_D(e_i, Y_j)$ is the similarity between the token sequence i and the template j, ϵ is a user defined similarity threshold, index template j is the most similar template from set S. There are generally many choices for similarity measure such as hamming distance [5], longest common subsequence [13] for which we shall discuss later in the next section.

After parsing, the event sequence i is linked with template j and its corresponding event ID (also called cluster id) C_j. The whole log messages are grouped into clusters $\hat{C}$:= {$\hat{c_1}$, $\hat{c_2}$, ..., $\hat{c_k}$}.

Evaluation There are generally three ways to evaluate parser accuracy: two on the event level, one on the token level.

The first is called F1 score which is the harmonic average of precision and recall. Precision is defined as:

$$P := \frac{TP}{TP + FP} = \frac{\sum_c (matched\_predicted\_pairs)}{\sum_c (predicted\_pairs)} \tag{1}$$

The predicted pairs can be calculated as counting how many different pairs in a predicted cluster $\hat{c_i}$: $\binom{|\hat{c_i}|}{2}$. The matched predicted pairs are those pairs in $\hat{c_i}$ that also can be found in true cluster c.

The second metric is called grouping accuracy which is more restricted than F1. It is defined as the summation on the size of the correctly matched clusters over total log size. A correctly matched cluster is defined as: every member in a predicted cluster matches with a true cluster and vise versa: $\hat{c_i} \subset c_j \wedge \hat{c_i} \supset c_j$. The last one is called token accuracy, in which it is considered correct when the predicted template is identical to the true template. This metric is more stringent compared to the above two metrics.

We choose grouping accuracy as the evaluation metric for two reasons. Firstly, high score in F1 score can not prevent error prediction on some rare but essential logs like error messages. Secondly, to achieve high token accuracy it requires more powerful token-by-token predicting methods such as language models, which is not the challenge to be solved in this study.

3 Empirical Study and Motivation

3.1 Accuracy instability

To demonstrate the accuracy instability issue, we conduct an experiment on the widely used model Drain [5] that grid-searches its two hyperparameters, namely similarity threshold st∈(0.0,0.7] and max depth d∈[3,7], and record the accuracy on each grid node. This is then repeated on all 16 datasets. As can be seen in Fig. 3, the parsing accuracy varies greatly across datasets. 6 out of 16 boxes has the box width (75 percentile - 25 percentile) more than 30% of the median. Similar results were found on other existing models, but were not demonstrated here due to space limit. It clear shows that without proper setting of the hyperparameters, existing parser's performance is significantly hindered.

Further experiment also shows the optimal hyperparameter values for each dataset are not concentrated in a small area of the search space that can be estimated easily. Hence its hard to set the hyperparameters without labeled dataset.

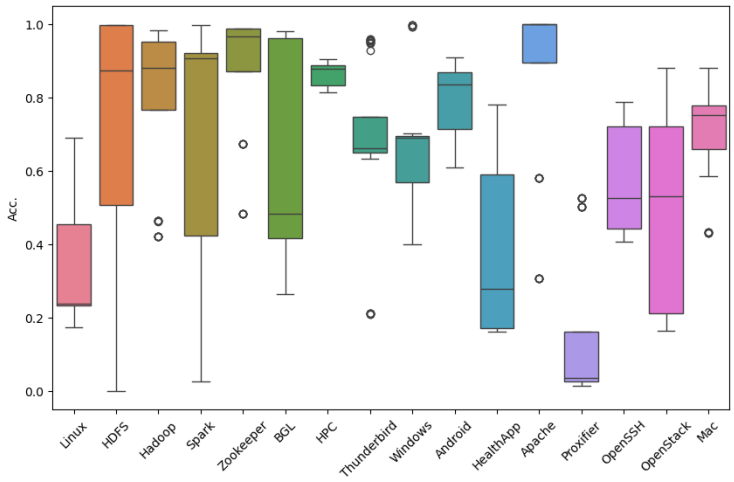

Fig. 3. Parsing accuracy variation of Drain - x axis represents 16 public dataset, y axis is the parsing accuracy defined in section 2, each box is calculated from statistics over 35 grid nodes

3.2 Performance deterioration

Performance deterioration happens in any machine learning application. But it is particularly severe in log parsing due to frequent system updates. To illustrate this issue, we constructed a new dataset BGL2 from the same program that

generated BGL [12]. BGL2 dataset contains 2k new log messages with their corresponding templates. A model is then hyperparameter searched on BGL and evaluated on the fixture of two datasets. The data drifting process is simulated by gradually increasing the mixing rate of BGL2 from 0% to 100% while reducing BGL's share. We record the model's evaluation accuracy and repeat the process for 10 times by random sampling from both datasets. As can be seen from the shaded blue line in Fig. 5, the average accuracy drops significantly during the process which clearly shows that without proper monitoring the parser can cease to work correctly rapidly.

Motivated by solving the above two essential issues in order to bring log parsing into industrial practice, we proposed Splash.

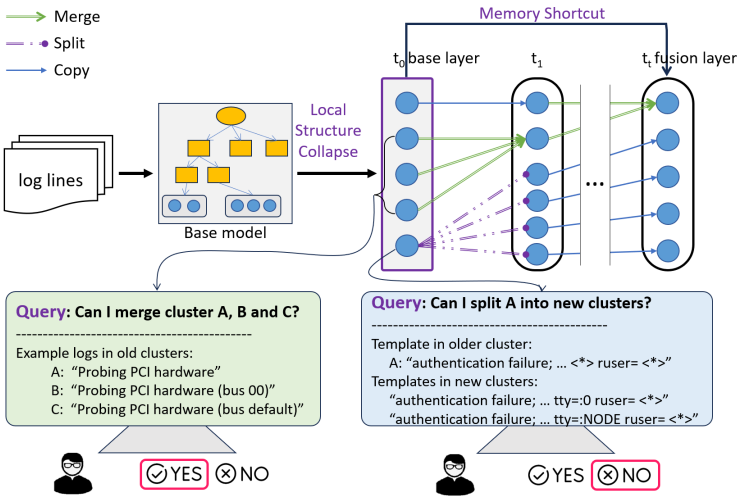

Fig. 4. Overview on Splash

4 Overview on Splash

The primary concept underlying Splash is the integration of models and human expertise. We utilize a base model to generate initial log clusters, examine the clusters from a different perspective, and engage human input to refine the mapping. Our hypothesis centers on the limitations of the base model, specifically its reliance on a) the local structure established during the learning process, and b) the associated similarity metric. Drawing inspiration from the concept of "Ensemble Modeling," we opt to flatten the local structure acquired by the base model, such as a heuristic tree with token length as the initial branching criterion, to promote a global perspective in subsequent analyses. Furthermore,

we intentionally select orthogonal similarity metrics to supplement those employed in the base model, aiming to yield query candidates that can potentially fix the error from base model. To facilitate efficient and active learning from human experts, we have devised a set of operations and a strategic querying approach. Additionally, in order to determine the optimal timing for human intervention throughout the parser's lifetime operation, we propose a proxy metric for monitoring the extent of data drift.

As is shown in 4, firstly the flattened clusters enters into the base layer at step t0, then Splash possesses three legal operations, namely 'merge', 'split' and 'copy'. Specifically, 'merge' propose to group candidates in t0 into a new cluster in the fusion layer t1, 'split' propose to break an old cluster into new clusters and 'copy' simply transport the old one to the new layer. Then Queries are formed with additional information included. For 'merge' operation, example logs in each old cluster are provided to help human making the yes-or-no decision. For 'split' operation, instead of logs, templates are provided in order to visualize how much potential increase on constant-count in each new cluster compared to the old one. Depending on human's decision, if it is a 'no', 'merge' or 'split' operation falls back to 'copy'. Lastly, since the querying process is sequential, multiple fusion layers can be created before the final layer and recorded for further analysis. However these intermediate fusion layers are not needed during prediction. Therefore, we add a 'memory shortcut' path from the base layer directly to the final layer.

4.1 Candidate searching

To obtain the right candidate for each query, a search for all possible mappings is conducted, then these are ranked, a subset of them forms the final query.

For 'merge' operation, we adopted the hierarchical clustering (HC) algorithm with the longest common sub-sequence (LCS) metric for the similarity measure on templates which is orthogonal to the Hamming distance used in the base model. The HC algorithm starts by forming each template as a cluster, then iteratively groups two most similar clusters. The similarity of two clusters is defined as the inverse of the largest distance between all element pairs in clusters. The similarity is obtained by dividing LCS of two templates by the smaller length of the two templates, $sim(x, y) = \frac{|LCS(x,y)|}{min(|x|,|y|)}$. Instead of choosing a cutoff threshold for stop criteria in classic HC, We stop it until k clusters are formed. The k clusters are then ranked by the order of which they are formed.

For 'split' operation, instead of a binary split [13], we choose to perform a multi-noded split to minimize the human effort. The splitting criteria is the information gain ratio (G) on the log cluster (C). As is shown in Eq.2, where $S(C)$ denotes the saturation ratio of the cluster, $|C|$ is the number of logs that are contained in a cluster, $\hat{C}_i$ is the new child cluster i. IV is the intrinsic value which penalize on larger number of nodes. The saturation ratio is defined as the number of constants over the template length. The whole searching procedure has two loops. For the inner loop, for a given cluster it searches through all token

position to obtain the max gain, $max_p(G)$. For the outer loop, it scans through all clusters and rank them by each $max_p(G_i)$.

$$G = (\frac{\sum_i S(\hat{C}_i)|\hat{C}_i|}{S(C)|C|} - 1) \cdot \frac{1}{IV}, \quad IV = -\sum_i \frac{|\hat{C}_i|}{|C|} \log \frac{|\hat{C}_i|}{|C|} \tag{2}$$

4.2 Query strategy

Depending on the human resouce budget, we designed three strategies to query, namely the 'single-max', 'iterate-till-false' and 'top-k'. The 'single-max' strategy is used to form a single query by the top 1 ranked candidates and this is used when the budget is tight. The 'iterate-till-false' iteratively samples candidates from ranked set and continues only when the previous query was accepted by the human expert. The 'top-k' means to form k queries without consideration of prior decisions. Since the main philosophy behind Splash is that model's ability is limited. We argue that it is likely that correct candidate hides below the false positive ones. Therefore, 'top-k' is the default strategy and k is set to 3 during the experiments.

4.3 Monitoring proxy

Unlike applications in which data naturally arrives with label, monitoring performance after deployment is difficult in log parsing since the template label is not given. An proper proxy is in great need. There are two categories of feature to be chosen from, one is derived from raw data, the other is from model's prediction. Features from raw data can be statistics on log message length, word length and vocabulary, etc. We choose message count from model prediction as the proxy feature. The message counts are aggregated per event ID and the Jensen-Shannon divergence is employed to measure distance between two count distributions sampled at different time points throughout Splash's service. Based on the distance between current distribution and the 'day 0' distribution, a decision on whether to call for human assistance is made.

5 Experiments

In this section, we conducted three experiments to evaluate Splash and ask three research questions:

- How accurate is Splash compared to other existing parsers?
- How effective is Splash's HITL framework given no label?
- Can Splash handle data drifting issue with monitoring proxy?

5.1 Full label scenario

16 datasets are provided by [12] with each log message labeled to the corresponding event ID and template, reader are referred to [12] for more details. We use the labels to perform hyperparameter search for all models listed in Table 1. Since labels are given, Splash calls a decision simulator instead of a real human in HITL. Notice, for Brain [14], We removed dataset-specific symbol replacements to align with the standard adopted by other models. As can be seen, Splash achieves the highest average accuracy and ranked first in 14 out of 16 datasets. Furthermore, the standard deviation of the accuracy across datasets is significantly lower than others (4 times smaller than SPINE), indicating its robustness on varies log distributions. By comparing to Splash's base model, Drain, we can see 12% accuracy improvement which demonstrates the design strength mentioned in previous section. We received similar results when changing base model from Drain to other baselines.

5.2 No label scenario

In this experiment, we randomly set model's hyperparameters as has been discussed in section 3 and compare the accuracy with and without HITL calls. The accuracy is ranked from low to high on x axis and plotted in Fig. 6. In 10 out of 16 cases, HITL calls give noticeable boosts on accuracy, followed by minor improvements in 4 of the rest 6 cases. Once the first HITL is done, pseudo labels can be used to perform second round of hyperparameter search and HITL calls. Further accuracy improvements on some datasets can be obtained by this iterative process. However, it is not guaranteed to converge to the optimal accuracy. We leave the design of a robust accuracy improving policy to future work.

5.3 Data drifting scenario

In this experiment, we first constructed an extended dataset named BGL2 based on the original BGL dataset. It simulates a twenty day software development scenario which contains newly coming events everyday and we manually labeled these new events. Then two model-updating strategies are compared with the baseline of no updating. One strategy is to update model everyday by calling HITL module, the other is to call HITL only when model performance falls below threshold. We conducted this experiment with randomly added new events for 10 repetitions and obtained the result shown in Fig. 5. It can be observed while the base model's accuracy is dropping significantly when the time progress, both updating strategies can effectively remove this negative effect. Furthermore, with proper monitoring proxy, much less effort can be saved by strategy 2 - 1 HITL call was made compared to 20 calls in the first strategy. On average, 5 (3 for merge, 2 for split) decisions was made by human user per HITL call demonstrating neglecting human effort compared to accuracy boosting.

Tools	Splash	Brain	SPINE	Drain	SLCT	...	Logram	ULP
Android	0.9215	0.942	0.932	0.911	0.882		0.795	0.838
Apache	1	1	1	1	0.731		0.313	1
BGL	0.982	0.9785	0.948	0.963	0.573		0.587	0.930
Hadoop	0.996	0.915	0.946	0.948	0.423		0.451	0.990
HDFS	1	0.9975	0.998	0.998	0.545		0.930	0.998
HealthApp	0.982	0.8765	0.988	0.78	0.331		0.267	0.902
HPC	0.957	0.9455	0.871	0.887	0.839		0.911	0.951
Linux	0.998	0.4805	0.676	0.69	0.297		0.361	0.364
Mac	0.905	0.942	0.789	0.787	0.558		0.568	0.814
OpenSSH	0.995	1	0.681	0.788	0.521		0.612	0.434
OpenStack	1	1	0.757	0.733	0.867		0.326	0.492
Proxifier	1	0.5265	0.967	0.527	0.518		0.504	0.024
Spark	0.998	0.9975	0.925	0.92	0.685		0.282	0.922
Thunderbird	0.970	0.971	0.964	0.955	0.882		0.554	0.676
Windows	0.999	0.718	0.99	0.997	0.697		0.694	0.410
Zookeeper	0.990	0.99	0.989	0.967	0.726		0.724	0.993
Avg. Acc.	0.981	0.892	0.901	0.866	0.630		0.555	0.733
Std.	0.029	0.168	0.112	0.136	0.189		0.21553	0.29853

Table 1. Accuracy of Splash and other models on 16 public datasets

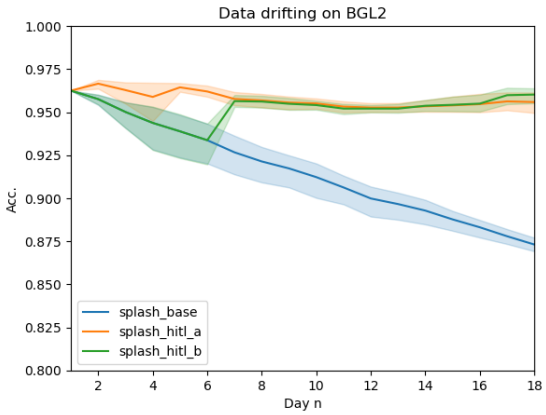

Fig. 5. Accuracy on data drifting scenario with and without human-in-the-loop calls, HITL is called with two strategies: a. human intervened every day, b. human intervened only on day6

6 Conclusion

In conclusion, the development of an accurate and reliable log parsing technique is essential for advancing further analysis. While current parsers demonstrate proficiency with public datasets, they encounter significant challenges in industrial environments, characterized by accuracy fluctuations in scenarios with limited or absent labels and sharp performance decline over time. To tackle these challenges, we introduce Splash, a novel approach that combines human expertise with data-driven methods. Splash identifies potential candidates from initial parsing outcomes and actively query for assistance with minimum human effort. Alongside mapping strategies and query formulation, we introduce a monitoring proxy to address the issue of performance degradation. Experimental results on 16 benchmark datasets demonstrate Splash's superior accuracy against baselines and its effectiveness in preventing deterioration in an ever-changing environment. Splash emerges as a model-agnostic, robust, accurate solution for log parsing in real-world scenarios.

Splash's parsing strategy primarily relies on template matching, with little consideration of the semantic meaning of events limiting its accuracy on certain cases. Currently, it functions as a single-line parser, lacking support for multi-line log statements (such as tables, diagrams). While knowledge from human feedback has been used for post-processing of model structure and hyperparameter tuning, we posit that deeper insights (such as keyword importance, delimiting pattern) can be extracted. Our future research will focus on exploring these aspects.

References

1. He, Shilin, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su and Michael R. Lyu. A Survey on Automated Log Analysis for Reliability Engineering. ACM Computing Surveys (CSUR) 54 (2020): 1 - 37.
2. Synopsys, Using Machine Learning to Automate Debugof Simulation Regression Results., white paper, retrieved Nov-2023
3. Anna M. Ravitzki, Vtool. The Big Data Revolution Beautiful Servant or Dangerous Monster? .
4. Xu, Wei et al. Detecting large-scale system problems by mining console logs. Symposium on Operating Systems Principles (2009).
5. He, Pinjia et al. Drain: An Online Log Parsing Approach with Fixed Depth Tree. 2017 IEEE International Conference on Web Services (ICWS) (2017): 33-40.
6. Nedelkoski, Sasho et al. Self-Supervised Log Parsing. ArXiv abs/2003.07905 (2020): n. pag.
7. Le, Van-Hoang and Hongyu Zhang. Log Parsing with Prompt-based Few-shot Learning. 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (2023): 2438-2449.
8. Xu, Junjielong et al. Prompting for Automatic Log Template Extraction. ArXiv abs/2307.09950 (2023): n. pag.
9. Dai, Hetong et al. Logram: Efficient Log Parsing Using nn-Gram Dictionaries. IEEE Transactions on Software Engineering 48 (2020): 879-892.

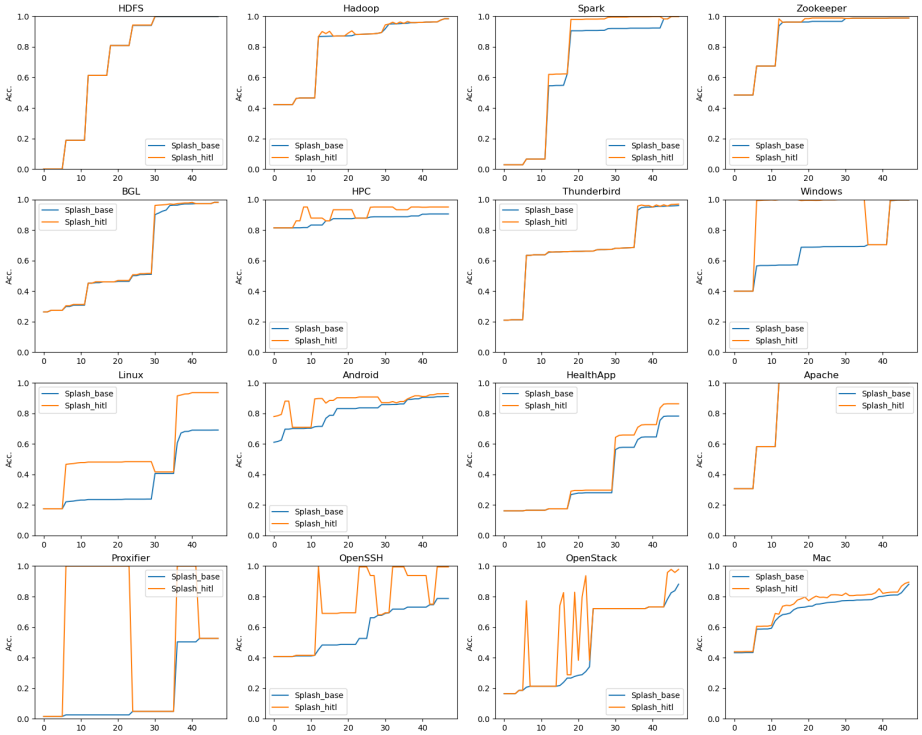

Fig. 6. Accuracy with different hyperparameter values with and without human-in-the-loop calls on 16 public datasets

10. Nagappan, Meiyappan and Mladen A. Vouk. Abstracting log lines to log event types for mining software system logs. 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010) (2010): 114-117.
11. Hamooni, Hossein et al. LogMine: Fast Pattern Recognition for Log Analytics. Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (2016): n. pag.
12. Zhu, Jieming et al. Tools and Benchmarks for Automated Log Parsing. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (2018): 121-130.
13. Wang, Xuheng et al. SPINE: a scalable log parser with feedback guidance. Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2022): n. pag.
14. Yu, Siyu et al. Brain: Log Parsing With Bidirectional Parallel Tree. IEEE Transactions on Services Computing 16 (2023): 3224-3237.