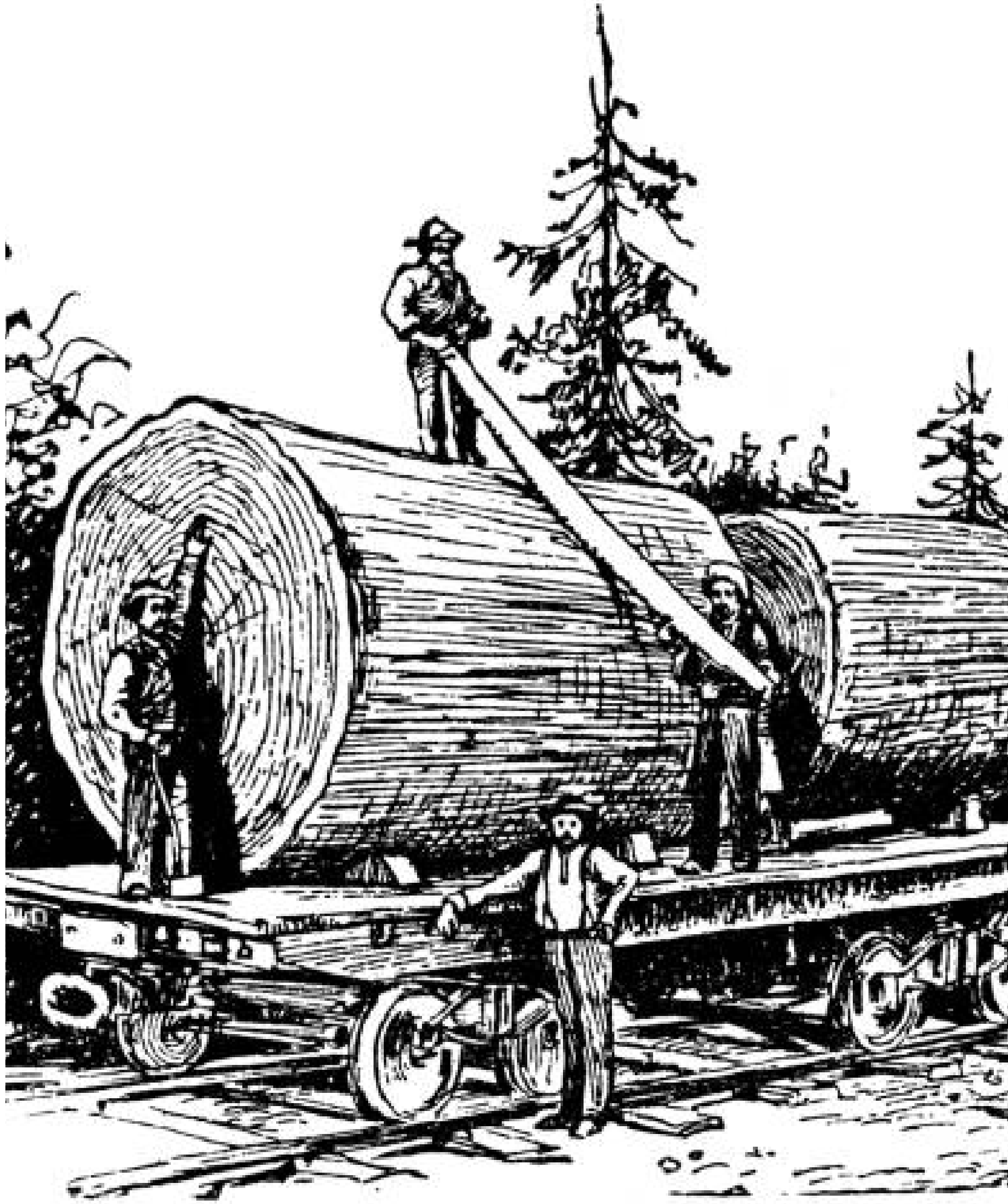
LogRules: Enhancing Log Analysis Capability of Large Language Models through Rules

Presented by Jason Gillette

University of Texas at San Antonio

September 15th, 2025



Introduction

Paper: *LogRules: Enhancing Log Analysis Capability of Large Language Models through Rules*
Authors: Xin Huang (Beijing Institute of Technology), Ting Zhang (Peking University), Wen Zhao (Peking University)
Problem Statement: "*smaller LLMs often lack the rule comprehension necessary to solve log tasks correctly or fail to follow the rules properly*" - large models can use rules but are costly, while small models are cheaper but fail to follow rules consistently.

Background

What is logging?

- the activity of felling trees and cutting and preparing the timber. 🪵
- the process of recording event messages generated by systems 📄

What is log parsing?

Log parsing is the conversion of raw logs to templates + parameters, e.g., `reading data from /etc/data` to template `reading data from <*>` and parameter `/etc/data`.

What is anomaly detection?

- Identifying log entries that deviate from normal system behavior, signaling a failure, misconfiguration, or security issue.
- Parsing enables signal (parameters), preventing noise from log constants (templates).

Background (cont.)

Why is parsing so difficult?

- Traditional REGEX extraction is brittle and fails to generalize as logs evolve.
- Case-based parsing (few-shot) tends to memorize examples, also doesn't generalize.

What is case-based parsing versus rule-based parsing?

- Case-based parsing: A model is given a few examples of logs and their templates, then asked to parse new logs by analogy.
- Rule-based parsing: The model is given explicit rules for how to abstract variables, then asked to apply them.

Background (cont..)

Case-based Prompt

```
Example: Jetty bound to port 62267 → Jetty bound to port <*>
Example: Got allocated containers 1 → Got allocated containers <*>
Now parse: Jetty bound to port 50322
```

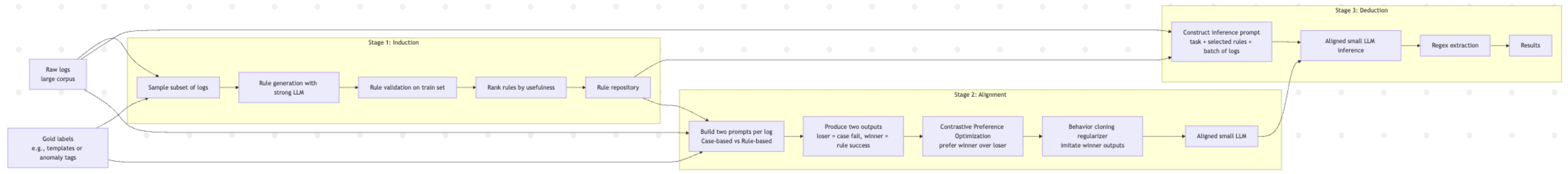
Rule-based Prompt

```
Rules:
1. Replace numbers with <*>
2. Replace file paths with <*>
Log: Jetty bound to port 50322
```

```
Response: Jetty bound to port <*>
```

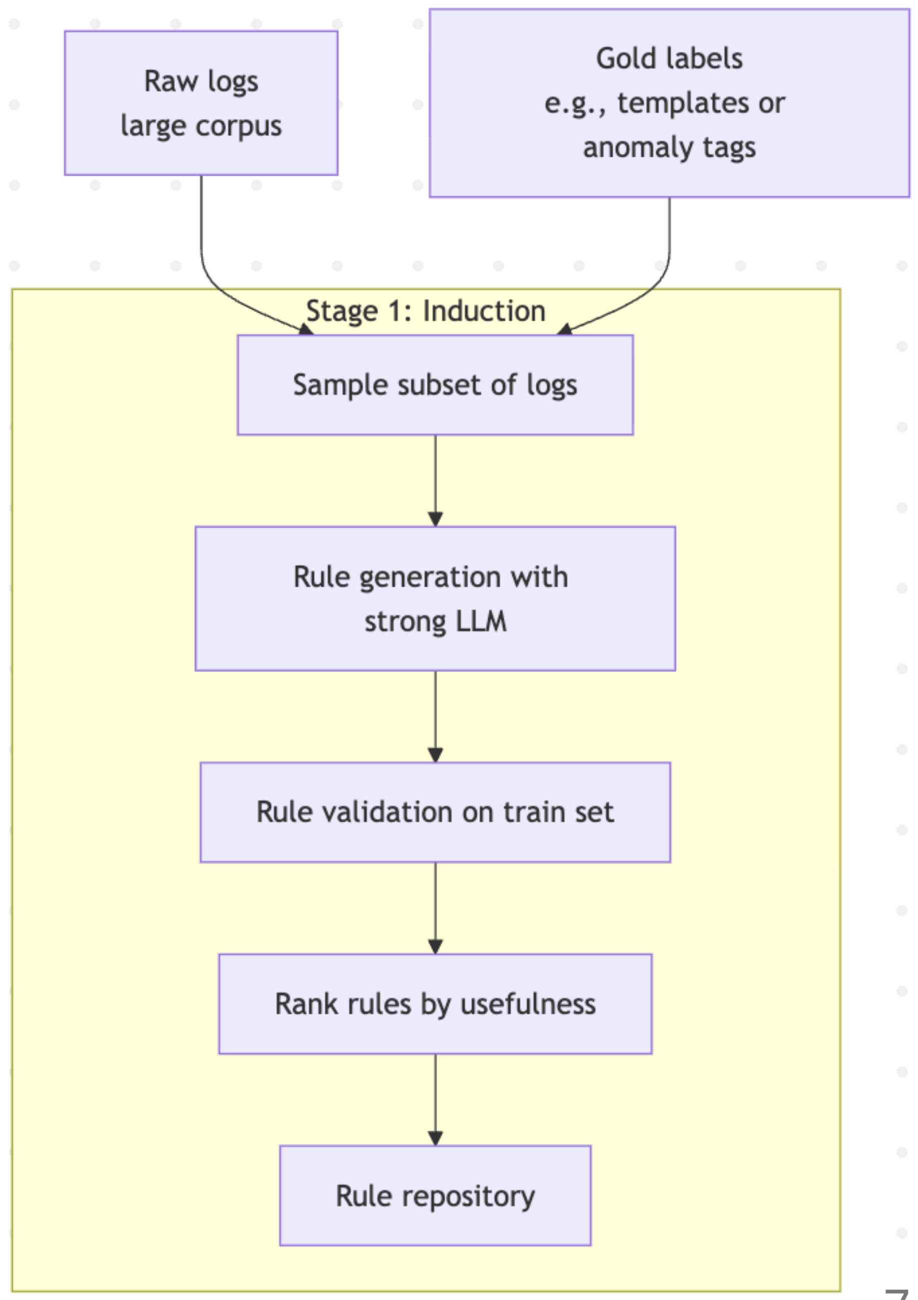
Architecture

1. Induction - Use strong LLM (GPT-4o-mini) for rule discovery
2. Alignment - Align smaller (cheaper) LLM (~7–8B) to prefer rules (fine-tuning)
3. Deduction - Generate results using fixed rule repository



Induction

- **Input**: sampled training logs
- **Rule generation**: GPT model proposes candidate rules, e.g., replace numbers, replace paths, replace ports
- **Validation**: keep only rules that consistently parse correctly
- **Ranking**: Rank occurrence in training set
- **Output**: rule repository

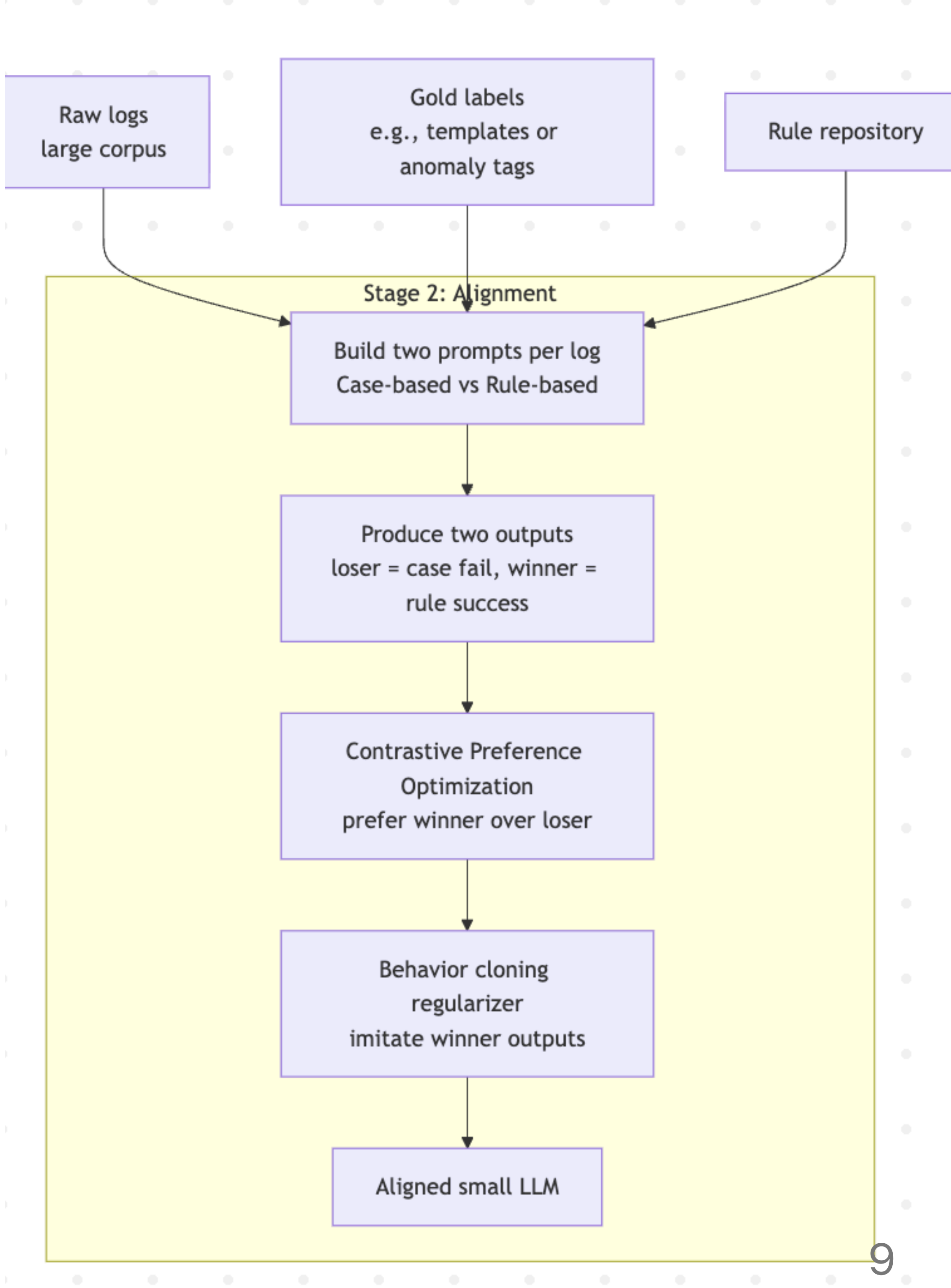

Raw logs large corpus
Gold labels e.g., templates or anomaly tags
Stage 1: Induction
Sample subset of logs
Rule generation with strong LLM
Rule validation on train set
Rank rules by usefulness
Rule repository

Rule-base Sample

Rank	Rule description
1	Replace sequences of numbers with `{{variable}}`
2	Replace IP addresses with `{{variable}}`
3	Replace host names with `{{variable}}`
4	Replace any occurrence of `size <number>` with `size {{variable}}`
5	Replace any occurrence of `blk_<number>` with `{{variable}}`
6	Replace any sequence indicating a user with `user={{variable}}`
7	Replace any `uid=<number>` and `euid=<number>` with `uid={{variable}}` and `euid={{variable}}` respectively

Alignment / Fine-tuning

- **Challenge**: small LLMs don't naturally follow rules
- **Dual prompting**:
  Case-based prompt → wrong output
  Rule-based prompt → correct output
- **Training pairs**: (log, gold, loser, winner)
- **Contrastive Preference Optimization (CPO)**: Increase winner probability, push down loser
- **Behavior cloning**: keeps outputs stable & on-task

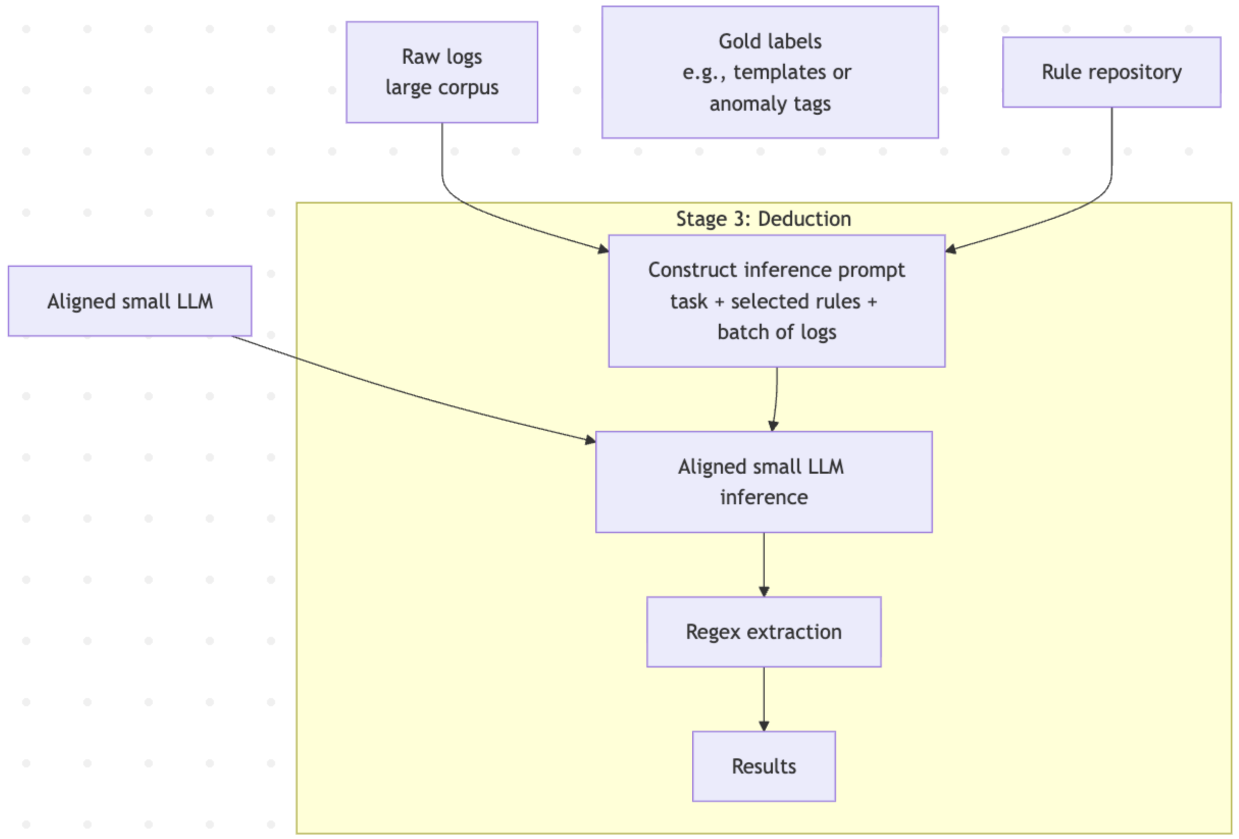


Raw logs large corpus
Gold labels e.g., templates or anomaly tags
Rule repository
Stage 2: Alignment
Build two prompts per log Case-based vs Rule-based
Produce two outputs loser = case fail, winner = rule success
Contrastive Preference Optimization prefer winner over loser
Behavior cloning regularizer imitate winner outputs
Aligned small LLM

Deduction

- **Testing workflow**: Prompt = {task + rules + logs} (Batching logs to avoid one-by-one bottleneck)
- Small aligned LLM outputs templates or anomaly labels
- Regex extracts final results

📝 No new rules are induced during tested, all inherited from induction.



Experiment

Research Question

Can explicit rules + a small aligned LLM (~7–8B) **parse logs** and **detect anomalies** better than case-based prompting and prior large LLM parsers—especially on unseen data.

Tasks & datasets

- **Log parsing**: 16 Loghub datasets, 2k logs each. Training uses **one log** from each of 10 datasets (for induction + alignment); testing uses the remaining 6 datasets (unseen systems/templates).

input: `workerEnv.init() ok /etc/httpd/conf/workers2.properties` + rule-base
output: `workerEnv.init() ok <*>`

- **Anomaly detection**: BGL and Spirit, 10,000 logs each, 4:1 train/test split, no overlap.

Loghub Data Sample

```json
{
  "log_samples": [
    {
      "line_id": 1,
      "timestamp": "Sun Dec 04 04:47:44 2005",
      "level": "notice",
      "content": "workerEnv.init() ok /etc/httpd/conf/workers2.properties",
      "event_id": "E2",
      "event_template": "workerEnv.init() ok <*>"
    },
    {
      "line_id": 2,
      "timestamp": "Sun Dec 04 04:47:44 2005",
      "level": "error",
      "content": "mod_jk child workerEnv in error state 6",
      "event_id": "E3",
      "event_template": "mod_jk child workerEnv in error state <*>"
    }
  ]
}
```

LogRules Models & Baseline Models

LogRules

- Induction: GPT-4o-mini to generate and rank rules.
- Alignment + Deduction: Qwen2.5-7B-Instruct, LLaMA-3-8B, LLaMA-3.1-8B

Baselines

- Parsing: Brain, DivLog, LILAC, LogBatcher (case-based LLaMA-3-8B), GPT-4-turbo.
- Anomaly detection: DeepLog, LogAnomaly, LogPrompt, LogGPT, OWL, plus OpenAI GPT models.

Log Parsing Metrics

- Grouping Accuracy (GA) - Whether logs of same template predicted in same cluster.
- Parsing Accuracy (PA) - Exact match between predicted template and gold template.
- Normalized Edit Distance (ED) - How similar predicted template is to gold, in terms of edit operations, normalized for length.
- F1 score of Grouping Accuracy (FGA) - Precision/Recall trade-off on grouping.
- F1 score of Template Accuracy (FTA) - Precision/Recall on exact template matches.

Anomaly Detection Metrics

- Precision (P) - Of the logs predicted anomalous, how many really are?
- Recall (R) - Of all real anomalies, how many did the model catch?
- F1 Score (F1) - Balance between precision and recall.

Grouping Accuracy Explained

Whether logs belonging to the same template are placed in the same cluster.

Example Logs:

1. Jetty bound to port 62267
2. Jetty bound to port 50322
3. Starting reading data from /etc/data

Gold clusters:
{1, 2} = Template: Jetty bound to port <>
{3} = Template: Starting reading data from <>

If the model groups {1, 3} together and {2} alone, GA is low because clustering is wrong even if individual templates look okay.

Normalized Edit Distance Explained

Minimum number of edits (insert, delete, substitute tokens) to turn the predicted template into the gold template, normalized by length

📝 Each word/placeholder in a template is treated as a token.

Example:

Gold template = `Starting reading data from <*>`

(5 tokens: Starting | reading | data | from | <*>)

Prediction = `Starting reading from <*>` (4 tokens).

Edit distance = 1 (delete “data”).

Normalized ED = 1 / 5 = 0.2 = 20% different.

F1 Score of Grouping Accuracy Explained

Template-level metric measuring how well logs are grouped into the right template clusters

Let N_T = # of ground-truth templates

Let N_P = # of predicted templates

Let N_C = # of templates that match correctly

Precision = N_C / N_P

Recall = N_C / N_T

FGA = harmonic mean of precision and recall.

F1 Score of Template Accuracy Explained

Similar to FGA, but stricter: a template is only counted correct if all tokens match the gold template and all its logs are grouped correctly

Ground truth templates = 100 ($N_T = 100$)

Model predicts 90 templates ($N_P = 90$)

70 of them are exact matches ($N_C = 70$)

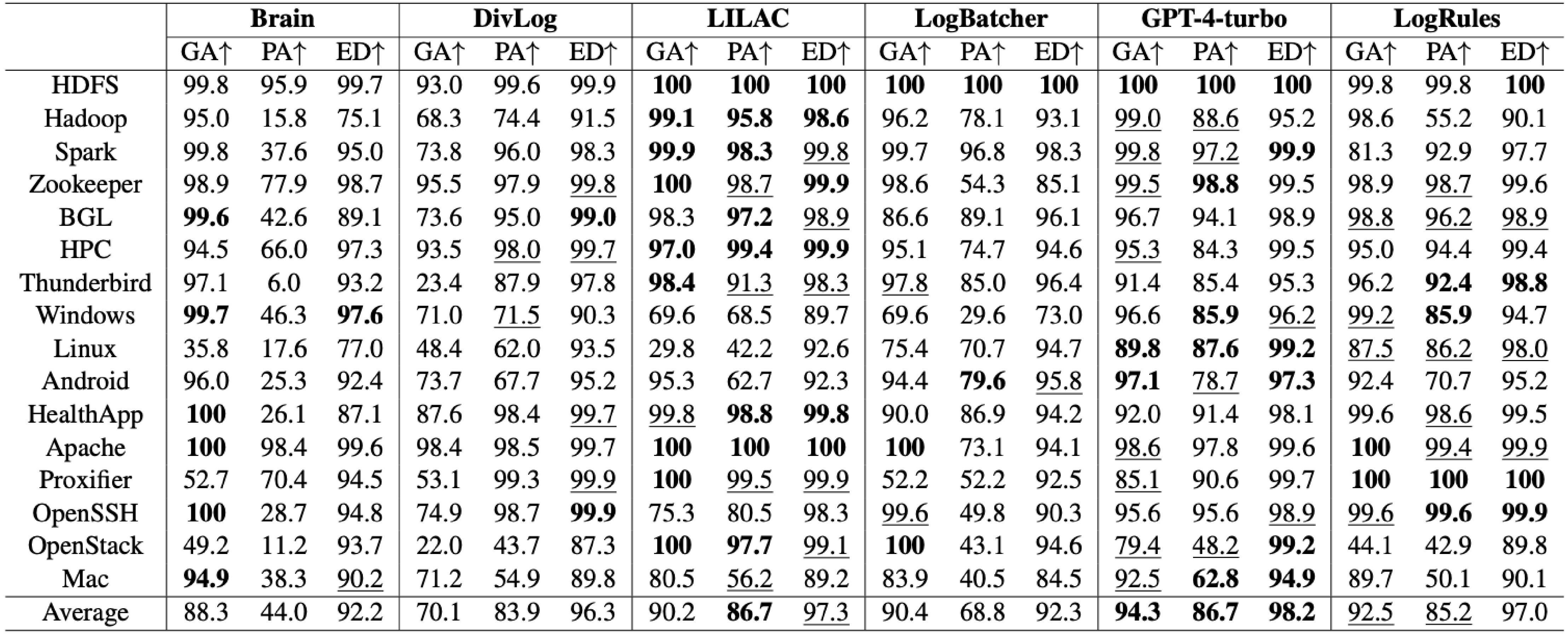
Precision = 70/90 = 0.78

Recall = 70/100 = 0.70

FTA ≈ 0.74

Results - Parsing across 16 datasets

- LILAC and GPT-4-turbo lead overall; LogRules is next best despite using a 7B model.
- LogRules still performs strongly on Windows and Linux, where others struggle.



	Brain			DivLog			LILAC			LogBatcher			GPT-4-turbo			LogRules		
	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑
HDFS	99.8	95.9	99.7	93.0	99.6	99.9	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	99.8	99.8	**100**
Hadoop	95.0	15.8	75.1	68.3	74.4	91.5	**99.1**	**95.8**	**98.6**	96.2	78.1	93.1	99.0	88.6	95.2	98.6	55.2	90.1
Spark	99.8	37.6	95.0	73.8	96.0	98.3	**99.9**	**98.3**	99.8	99.7	96.8	98.3	99.8	97.2	**99.9**	81.3	92.9	97.7
Zookeeper	98.9	77.9	98.7	95.5	97.9	99.8	**100**	98.7	**99.9**	98.6	54.3	85.1	99.5	**98.8**	99.5	98.9	98.7	99.6
BGL	**99.6**	42.6	89.1	73.6	95.0	**99.0**	98.3	**97.2**	98.9	86.6	89.1	96.1	96.7	94.1	98.9	98.8	96.2	98.9
HPC	94.5	66.0	97.3	93.5	98.0	99.7	**97.0**	**99.4**	**99.9**	95.1	74.7	94.6	95.3	84.3	99.5	95.0	94.4	99.4
Thunderbird	97.1	6.0	93.2	23.4	87.9	97.8	**98.4**	91.3	98.3	97.8	85.0	96.4	91.4	85.4	95.3	96.2	**92.4**	**98.8**
Windows	**99.7**	46.3	**97.6**	71.0	71.5	90.3	69.6	68.5	89.7	69.6	29.6	73.0	96.6	**85.9**	96.2	99.2	**85.9**	94.7
Linux	35.8	17.6	77.0	48.4	62.0	93.5	29.8	42.2	92.6	75.4	70.7	94.7	**89.8**	**87.6**	**99.2**	87.5	86.2	98.0
Android	96.0	25.3	92.4	73.7	67.7	95.2	95.3	62.7	92.3	94.4	**79.6**	95.8	**97.1**	78.7	**97.3**	92.4	70.7	95.2
HealthApp	**100**	26.1	87.1	87.6	98.4	99.7	99.8	**98.8**	**99.8**	90.0	86.9	94.2	92.0	91.4	98.1	99.6	98.6	99.5
Apache	**100**	98.4	99.6	98.4	98.5	99.7	**100**	**100**	**100**	**100**	73.1	94.1	98.6	97.8	99.6	**100**	99.4	99.9
Proxifier	52.7	70.4	94.5	53.1	99.3	99.9	**100**	99.5	99.9	52.2	52.2	92.5	85.1	90.6	99.7	**100**	**100**	**100**
OpenSSH	**100**	28.7	94.8	74.9	98.7	**99.9**	75.3	80.5	98.3	99.6	49.8	90.3	95.6	95.6	98.9	99.6	**99.6**	**99.9**
OpenStack	49.2	11.2	93.7	22.0	43.7	87.3	**100**	**97.7**	99.1	**100**	43.1	94.6	79.4	48.2	**99.2**	44.1	42.9	89.8
Mac	**94.9**	38.3	90.2	71.2	54.9	89.8	80.5	56.2	89.2	83.9	40.5	84.5	92.5	**62.8**	**94.9**	89.7	50.1	90.1
Average	88.3	44.0	92.2	70.1	83.9	96.3	90.2	**86.7**	97.3	90.4	68.8	92.3	**94.3**	**86.7**	**98.2**	92.5	85.2	97.0

Rule-based vs Case-based Parsing Results

- Rule-based prompting beats case-based on GA, FGA, and FTA (averaged over 16 datasets).
- **Shot sensitivity**: Among case-based settings, 2-shot > 1-shot & 4-shot, and FGA is worst in 4-shot, highlighting instability from example selection. - Rule-based is more stable because rules are validated and capture the essence.

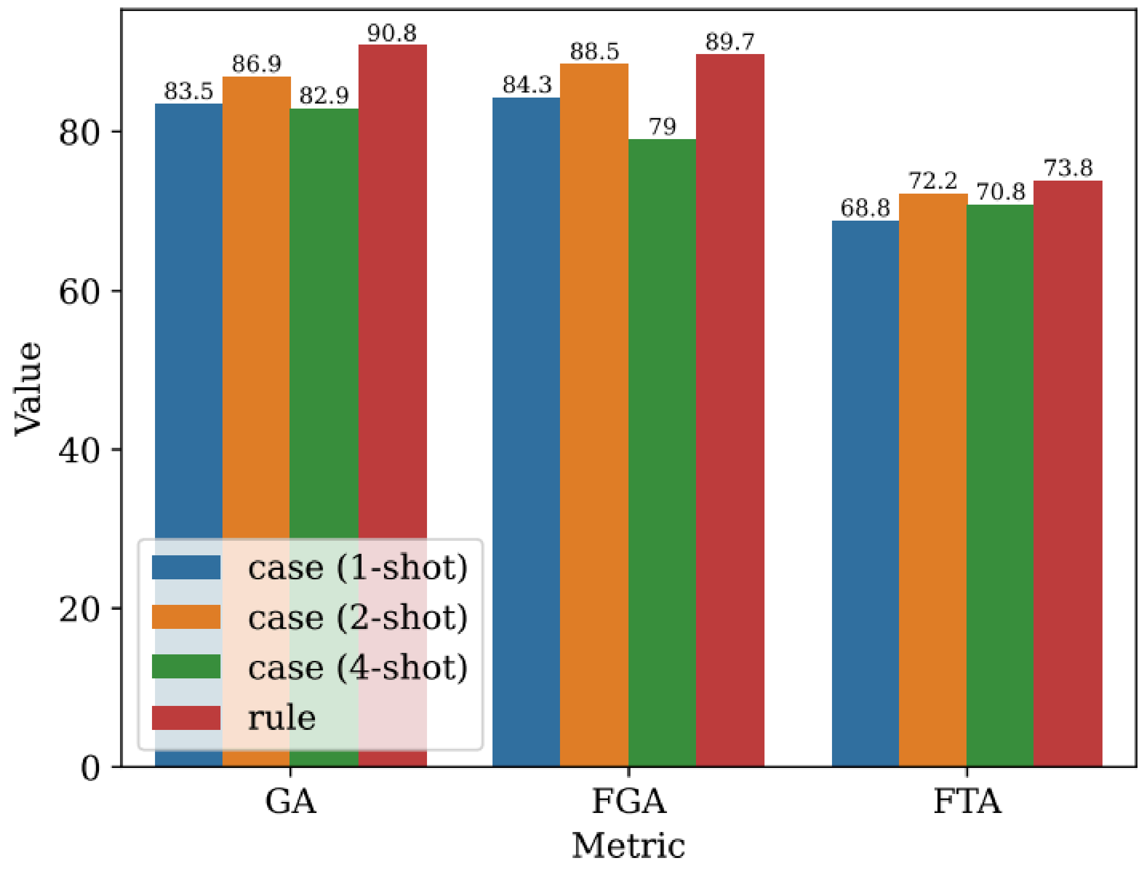
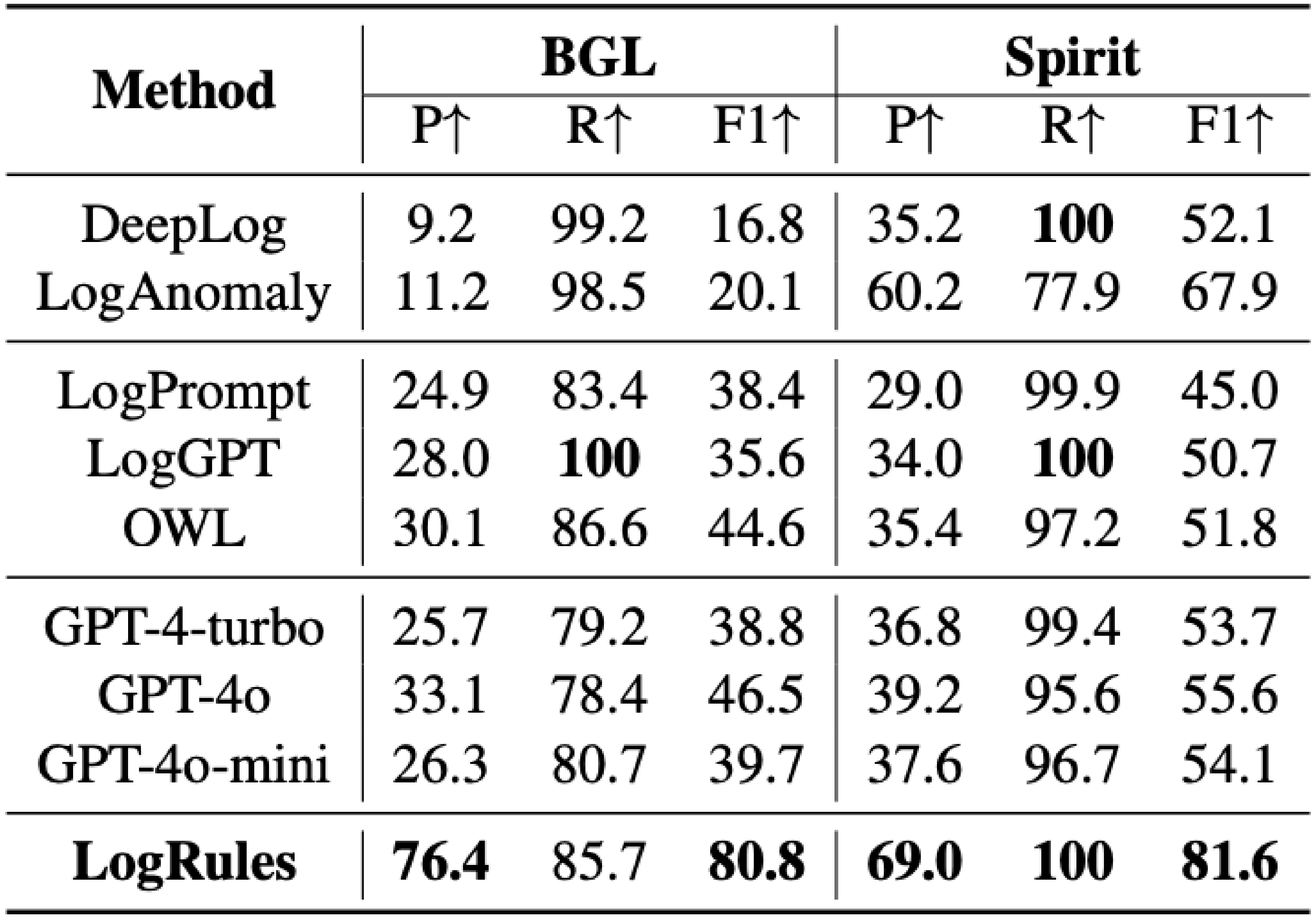


Figure 5: Comparison between case-based and rule-based reasoning for log parsing (%).

Anomaly Detection Results

- LogRules achieves the best F1 in the comparison, markedly improving over the LLM baseline OWL by +36.2% (BGL) and +29.8% (Spirit).
- Rule-based prompting outperforms case-based across P/R/F1 on both datasets.

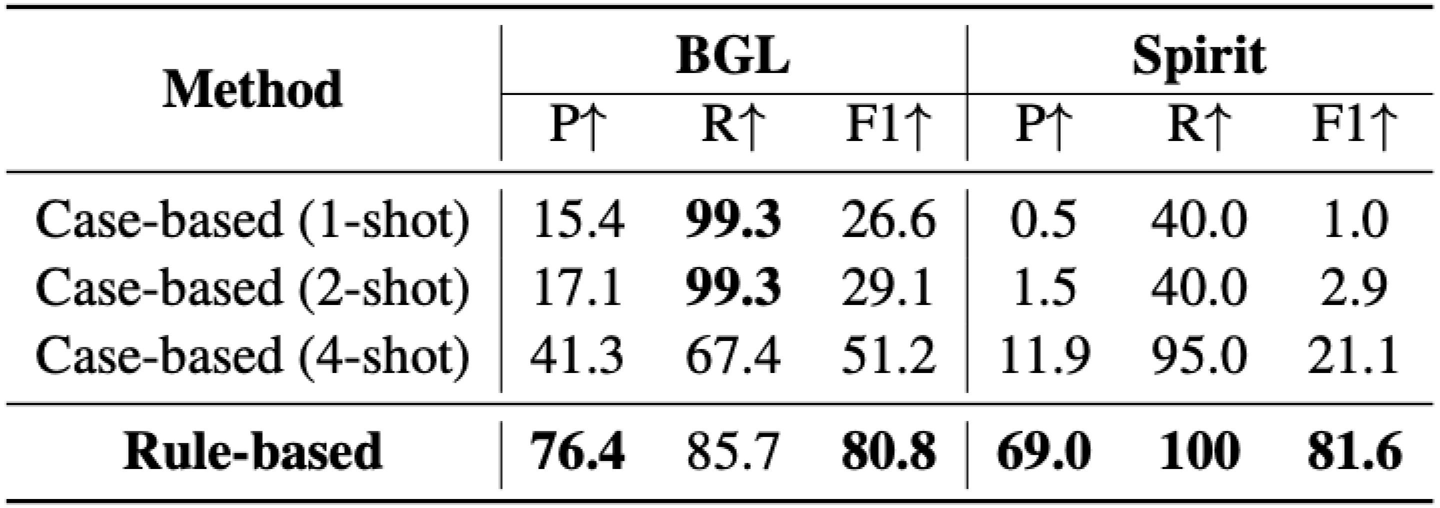
Table 2: Comparison with state-of-the-art log-based anomaly detection methods (%). The **bold** fonts indicate the best results among methods.



Method	**BGL**			**Spirit**		
	P↑	R↑	F1↑	P↑	R↑	F1↑
DeepLog	9.2	99.2	16.8	35.2	**100**	52.1
LogAnomaly	11.2	98.5	20.1	60.2	77.9	67.9
LogPrompt	24.9	83.4	38.4	29.0	99.9	45.0
LogGPT	28.0	**100**	35.6	34.0	**100**	50.7
OWL	30.1	86.6	44.6	35.4	97.2	51.8
GPT-4-turbo	25.7	79.2	38.8	36.8	99.4	53.7
GPT-4o	33.1	78.4	46.5	39.2	95.6	55.6
GPT-4o-mini	26.3	80.7	39.7	37.6	96.7	54.1
LogRules	**76.4**	85.7	**80.8**	**69.0**	**100**	**81.6**

Anomaly Detection Results (cont.)

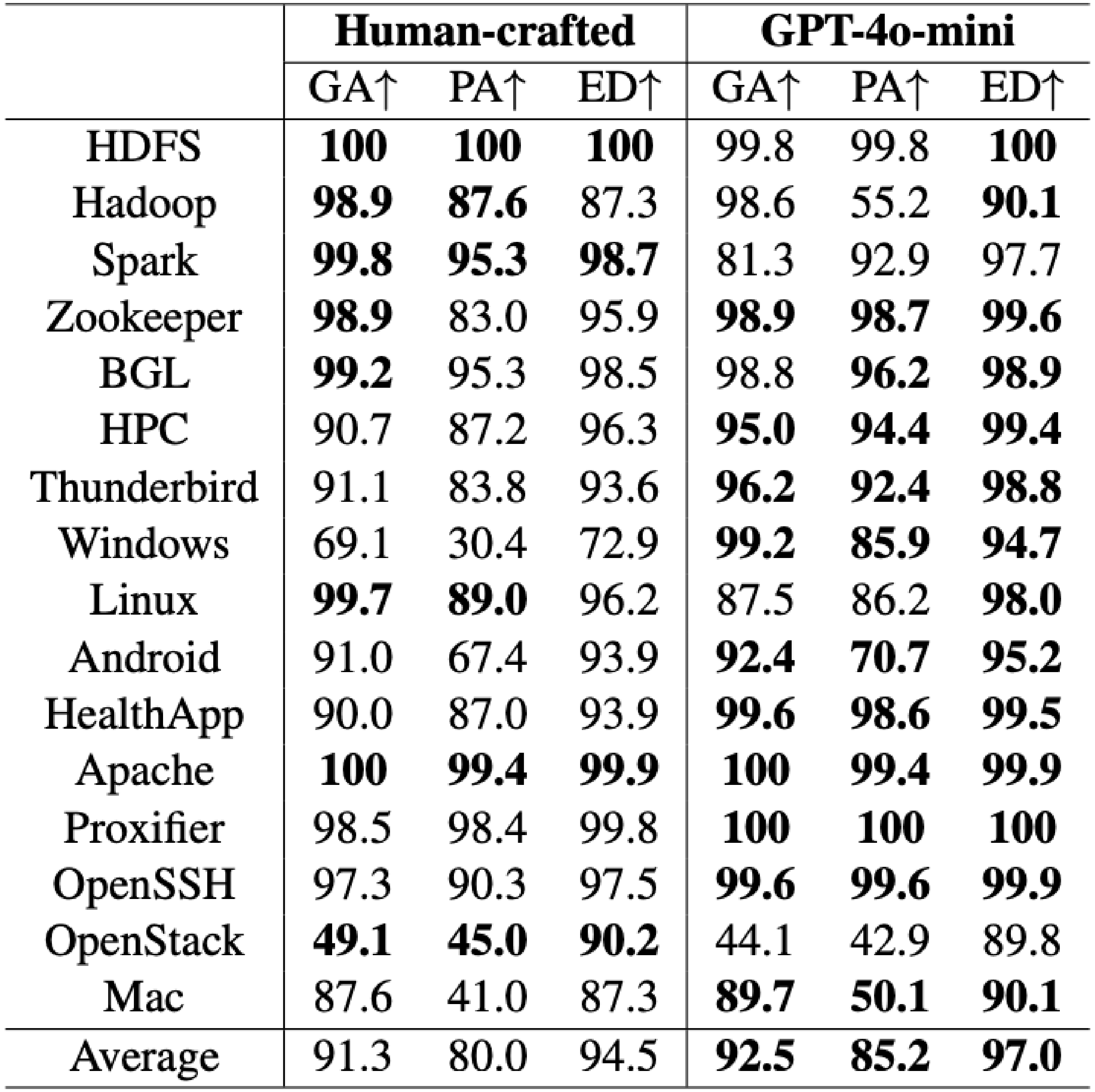
- Case-based performance varies with shot count, while rule-based is more stable.



Method	**BGL**			**Spirit**		
	P↑	R↑	F1↑	P↑	R↑	F1↑
Case-based (1-shot)	15.4	**99.3**	26.6	0.5	40.0	1.0
Case-based (2-shot)	17.1	**99.3**	29.1	1.5	40.0	2.9
Case-based (4-shot)	41.3	67.4	51.2	11.9	95.0	21.1
Rule-based	**76.4**	85.7	**80.8**	**69.0**	**100**	**81.6**

Human Rules Experiment

- **Human + model rules**: LogRules supports both human-crafted and GPT-generated rules.
- Rules generated by GPT-4o-mini outperform those crafted by humans, although the human-crafted rules have some advantages on certain logs.



	Human-crafted			**GPT-4o-mini**		
	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑
HDFS	**100**	**100**	**100**	99.8	99.8	**100**
Hadoop	**98.9**	**87.6**	87.3	98.6	55.2	**90.1**
Spark	**99.8**	**95.3**	**98.7**	81.3	92.9	97.7
Zookeeper	**98.9**	83.0	95.9	**98.9**	**98.7**	**99.6**
BGL	**99.2**	95.3	98.5	98.8	**96.2**	**98.9**
HPC	90.7	87.2	96.3	**95.0**	**94.4**	**99.4**
Thunderbird	91.1	83.8	93.6	**96.2**	**92.4**	**98.8**
Windows	69.1	30.4	72.9	**99.2**	**85.9**	**94.7**
Linux	**99.7**	**89.0**	96.2	87.5	86.2	**98.0**
Android	91.0	67.4	93.9	**92.4**	**70.7**	**95.2**
HealthApp	90.0	87.0	93.9	**99.6**	**98.6**	**99.5**
Apache	**100**	**99.4**	**99.9**	**100**	**99.4**	**99.9**
Proxifier	98.5	98.4	99.8	**100**	**100**	**100**
OpenSSH	97.3	90.3	97.5	**99.6**	**99.6**	**99.9**
OpenStack	**49.1**	**45.0**	**90.2**	44.1	42.9	89.8
Mac	87.6	41.0	87.3	**89.7**	**50.1**	**90.1**
Average	91.3	80.0	94.5	**92.5**	**85.2**	**97.0**

Discussion

- **Explicit rules = real generalization**: Turning implicit log patterns into a reusable rule repository is empirically more robust than case-based prompting.
- **Alignment matters**: Small LLMs use rules poorly; CPO + behavior cloning trains rule-following outputs over few-shot.
- LLaMA-3.1-8B-Instruct often emits Python code to “solve” parsing; without an interpreter this can slow or fail inference, contributes to Qwen2.5-7B performance.

Questions

1. Performance evaluation is dependent on regex extractions from model responses. Did the authors validate these extractions to ensure the model is not penalized for regex errors (false negatives)?
2. How does log-throughput impact model performance, i.e., prompting N logs relative to context window? Prompting with a single log likely to create bottleneck in production so the authors process an unspecified N logs per prompt, but this choice is unclear.
3. Can sampling a more log entries during induction lead to increased performance? Authors appear to sample 1 log per 10 training sets which may not be representative enough to generalize over evolving logs.
4. Would a hybrid approach like rule-based prompting for log parsing + case-based examples of variables improve performance on pre-trained or fine-tuned models?
5. Why did the authors consistently misspell the word "two" as "tow"? 💩

Reference

LogRules: Enhancing Log Analysis Capability of Large Language Models through Rules

(Huang et al., Findings 2025)