高精度日志规则生成框架 LogParserX

银枪修罗一枪爱死

2025 年 2 月 21 日

目录

1 设计思路

1.1 任务分析

根据附件和数据集中的数据格式分析，要求在提供日志原文 logText 的情况下，提供能够提取 logField 的高精度规则，其中 key 和 value 均来源于 logText。

首先对给出的开发集数据进行分析，一共四百条都带 logField 部分，根据人工手动分析，把部分不合理的提取结构和内容进行修改，比如“HTTP/S 响应码”应该作为 key，而不是：“HTTP”作为 key，剩余的部分“S/响应码 403”作为 value，这不符合常识。并且分类出来与 20 个检查点命中的一共有 11 种，其中未出现在检查点明确字段里的有 433 种，指 key-value 是唯一的数量。

通过对开发集的数据集人工分类，可以得到大概如下的 key-value 分类，在每一条日志种可能包含多个不同类别的 key-value 对，因此这里只提供最常见的字段分类：

表 1: 日志字段分类与正则规则表

分类	说明	值含义	示例	正则表达式
时间	无年份	月日时: 分: 秒	Jul 29 16:57:28	`date_p`
	带年份	年-月-日时: 分: 秒	2021-11-05 11:34:18+08:00	`date_p_3`
	简写日期	年-月-日	2017-06-22	`date_iso_p`
设备信息	Hostname	主机标识符	sco-12	`hostname_p`
	User-Agent	客户端信息	Chrome/75.0.3770.100	`user_agent_p`
	IP:Port	网络端点	10.207.94.231:52445	`ip_port_p_3`
	SessionID	会话标识	12222	`session_p`
	ProcessID	进程标识	sshd	`pid_p`
键值对	CID	连接标识	0x81d80420	`key_value_p`
	OID	对象标识	1.3.6.1.4.1.2011.6.8.2.2.0.3	`key_value_p`
	Domain	域信息	vlan3260	`key_value_p`
	Storage	存储类型	SSD	`key_value_p`
	JSON	结构化数据	”key”: ”value”	`json_str_p`
	Segment	特殊字段	类型: Host	`segment_p`
安全事件	WEB 攻击	攻击类型	php_GET 注入	`web_attack_p`
	系统告警	告警级别	NULL 高 55	`sys_attack_p`
	端口扫描	端口序列	90-09-10-20	`WebPort_p`
功能信息	函数调用	方法参数	func(param)	`function_p`
	路径分割	资源定位	key/value	`slash_pattern`
其他	方括号	端口	[123456]	`fangkuohao_p`
	电邮	提取邮件	mail*	`email_p`

由表 1 可以知道，不同的 key-value 对应不同的正则表达式，如果只使用一条通用正则是无法过滤出所有可能的信息的，因此分层过滤就很重要了，对日志分类后整理出人工初级的手工正则，再对日志进行逐条过滤，最后再整合到一起作为完整 logField。

1.2 任务分解

我们一共需要完成三个主要任务：正则生成，含正则代码生成，优化含正则代码。另外，还有一个验证任务用于验证优化正则代码和测试集匹配程度。

1.2.1 正则生成

提供 logText 和 logField 部分，要求生成正则表达式，实现用于 logText 能得到 logField 的要求。

1.2.2 含正则代码生成

使用上一步生成的正则表达式，嵌入 python 代码函数里，实现正则匹配过滤，最后返回生成代码。

1.2.3 优化含正则代码

使用上一步生成的代码，验证生成代码的准确性，若不匹配则优化代码，最后返回优化的代码以及覆盖率测试信息。在完成三个主要任务之后，需要对优化正则代码的结果进行测试集验证，测试集由大模型从开发集学习数据结构和内容进行派生得来，最后对比官方评价指标和覆盖率。

1.3 基本框架

设计智能体框架如下图1，主要包含三个代理上下文合作，正则检查员、python 代码生成专家、验证正则代码专家，默认都使用 qwen-2.5-72b-instruct 交互。

它们分别负责不同的工作，首先提供人工知识库给正则检查员，正则检查员对人工知识库种涉及到当前日志的正则进行提取并检查，若不满足匹配则调整该正则表达式，最后传递给 python 代码生成专家；python 代码生成专家要求使用这个调优过后的正则来生成匹配的 python 代码并检测运行效果，若运行效果不佳，允许进行正则调整和代码调整，最后返回使用正则表达式的代码；验证正则代码专家接收到上一步传来的 python 代码，直接验证其效果，若不匹配，直接再次优化 python 代码并计算匹配率，最后返回生成的报告，报告包含代码，优化措施，匹配率等信息。

2 实现原理

本项目主要使用了智能体开发框架 CrewAI version 0.102.0，在 Windows 11 和 Ubuntu 20.04.4 环境下，结合 Docker 进行实验。实验环境使用了 Python 3.12.9 和 qwen-2.5-72b-instruct 模型作为主要的技术栈。

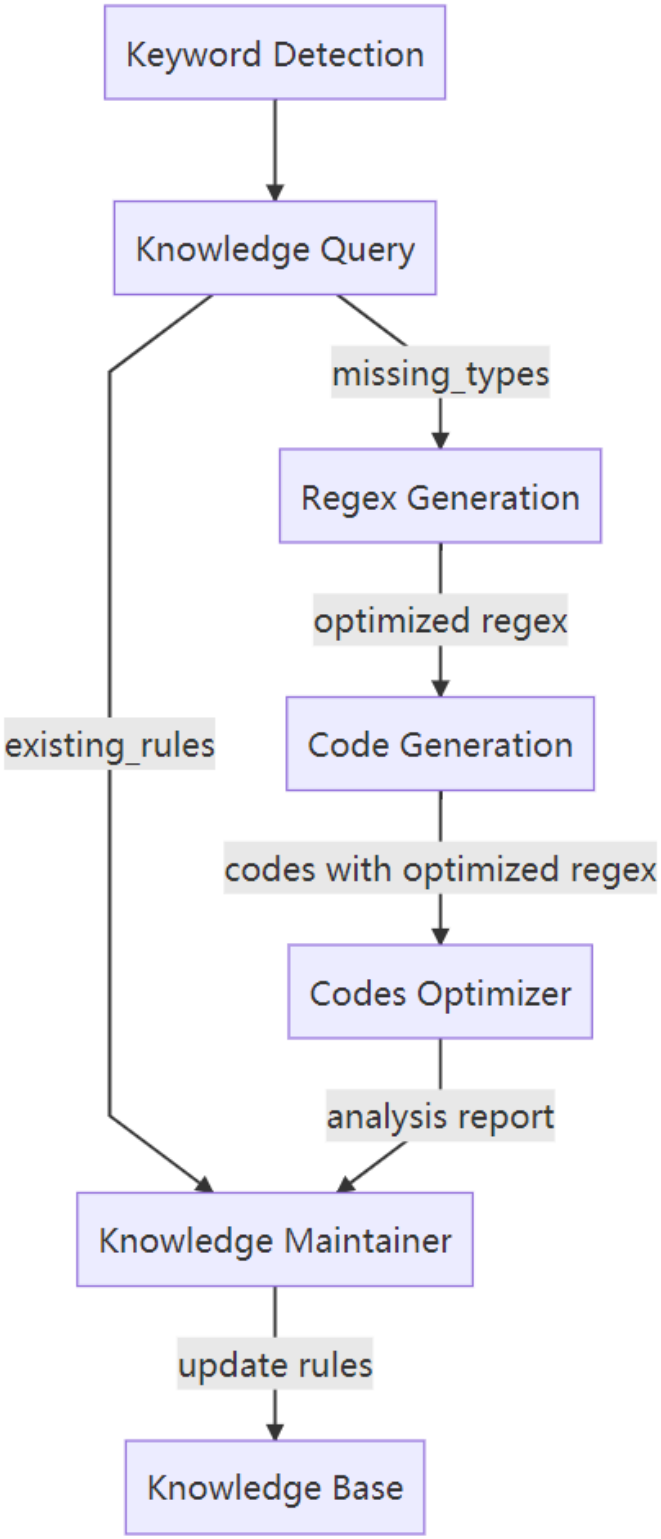

图 1: **LogParserX 框架**

2.1 CrewAI 智能体开发框架

CrewAI 是一个用于构建多智能体系统的框架，支持通过角色和任务的定义来实现复杂的智能体协作。CrewAI 提供了两种主要的构建方式：Crews 和 Flows。Crews 是一组具有自主性和协作能力的智能体，Flows 是事件驱动的工作流，用于精确控制复杂的自动化流程。具体代码原理见 List 1。

自定义代理和沟通上下文：在本项目中，我们自定义了智能体的代理和沟通上下文，以满足特定的任务需求。通过定义智能体的角色、目标和背景故事，可以实现智能体之间的自然协作和动态任务分配。

自动运行代码检测工具：为了确保代码的质量和稳定性，我们设置了自动运行代码检测工具。这些工具可以在代码运行过程中自动检测潜在的问题，并提供详细的错误报告和建议。

2.2 实验设置与系统实现

2.2.1 实验环境

- **操作系统**: Windows 11 & Ubuntu 20.04.4 双平台验证，确保跨平台兼容性与稳定性
- **容器化**: 为智能体使用代码检查工具提供环境
- **编程环境**: Python 3.12.9，依赖管理使用 anaconda 2020.07 和 pip3。

2.2.2 模型架构

采用 `qwen-2.5-72b-instruct` 模型作为核心智能体交互的大模型，关键参数配置如下表 2：

表 2: 模型参数配置

参数	值
上下文窗口	4096 tokens
温度系数	0.1

2.3 数据处理流程

2.3.1 开发集分段策略

按照日志结构划分如表 3, 分别分为大类的四种类型，每个类型 100 条训练数据，用于智能体学习阶段优化含规则的代码，这四类数据将会在测试集的生成阶段作为母本用于派生相同数目的日志。

表 3: 日志数据结构特征分类统计

分类	数据范围	字段占比	结构模式	关键分隔符	攻击标记
Class 1	0-100	84.6%	时间 + 主机 + 会话 ID	[ ] / space	无
Class 2	100-200	81.1%	应用层协议事务流	:/&=;	中
Class 3	200-300	66.9%	键值对配置操作	= ;	低
Class 4	300-400	71.0%	混合攻击向量	~<>	高

注：1) 字段占比指有效结构化字段比例；2) 攻击标记级别依据 CVE 评分标准划分：低 (0-4.0)、中 (4.1-7.0)、高 (7.1-10.0)；3) 分隔符采用正则表达式表示法：[] 表示可选符，/&= 表示交替出现

2.3.2 结构特征学习

根据上一个分段要求，编写提示词和例子，要求大模型生成相同结构和特征的日志返回，一次输入一个母本，生成一条测试日志，保证内容不相同但是包含特征和结构一致。下面的文字框图是代表一个生成任务和代理的提示词配置。

Data Generation Specification

Role Log Info Generator

Goal Generate the same format of given log, make them belong to the same source.

Backstory Expert in log extraction and generation. Through scanning logs and extracting keywords from given records, acquires sufficient features to generate target log info. Produces clean logs with identical structure but varying contexts, ensuring provenance consistency between original and generated logs.

Data Generation Task You are given a log: log, you should generate a log with the same format and different context and belong to the same source. For example, your given example is: .... Your generated result should only include the whole log without any explanation or other texts.

Input: Original log sample

Output: Generated log sample

Features:
- Structure pattern replication
- Keyword preservation
- Delimiter consistency

3 实现功能

在 LogParserX 框架中主要实现的是智能体学习和结果检验两个部分的功能。智能体学习阶段把输入的日志内容和手工知识库进行学习和优化更新，最后输出一个含优化结果和代码的报告文件，用于下游的结果检验部分进行验证和比较。结果检验阶段负责提取报告中的代码信息，并将测试集中的日志内容嵌入，运行代码进行筛选结果验证，计算匹配率和覆盖率，用于输出最后的验证结果。

整体框架的输入输出流见图 2。其中前四个部分主要由智能体学习部分负责，后面的代码提取和测试流程由结果检验部分承担。

3.1 主要功能

3.1.1 智能体学习

智能体学习部分包含三个智能体，正则检查员、Python 代码生成专家和验证正则代码专家，各司其职并参与团队协作。这三个智能体通过协作完成了从正则表达式检查、代码生成到代码验证的整个过程，确保了日志解析的准确性和效率。

- **正则检查员**：首先接收人工知识库，从中提取涉及当前日志的正则表达式并进行检查。如果正则表达式不满足匹配要求，正则检查员会对正则表达式进行调整，然后将其传递给 python 代码生成专家。
- **Python 代码生成专家**：收到调优后的正则表达式后，使用它生成匹配的 python 代码，并检测代码的运行效果。如果运行效果不佳，python 代码生成专家可以对正则表达式和代码进行调整，最后返回使用正则表达式的代码。
- **验证正则代码专家**：从上一步接收 python 代码，直接验证其效果。如果不匹配，验证正则代码专家会再次优化 python 代码并计算匹配率，最后返回包含代码、优化措施和匹配率等信息的报告。

3.1.2 结果检验

结果检验部分是整个 LogParserX 框架中对智能体学习成果进行验证和评估的关键环节。其主要任务是对智能体学习阶段输出的报告文件中的代码信息进行提取，并结合测试集中的日志内容，运行相应的代码来对筛选结果进行验证，进而计算出匹配率和覆盖率等重要指标，最终输出具有说服力和准确性的验证结果。

结果检验阶段首先会对报告文件中的代码信息进行精准提取。提取完成后，系统会将测试集中的日志内容嵌入到提取的代码中，通过运行代码来对日志进行筛选和处理。匹配率是指在测试集中，被正确筛选出的日志条目占总日志条目的比例，它能够直观地反映代码对于日志的提取精度。覆盖率指的是生成代码匹配日志得到的 logField 部分能够覆盖原始 logField 的比例，这代表代码对日志的提取宽度。

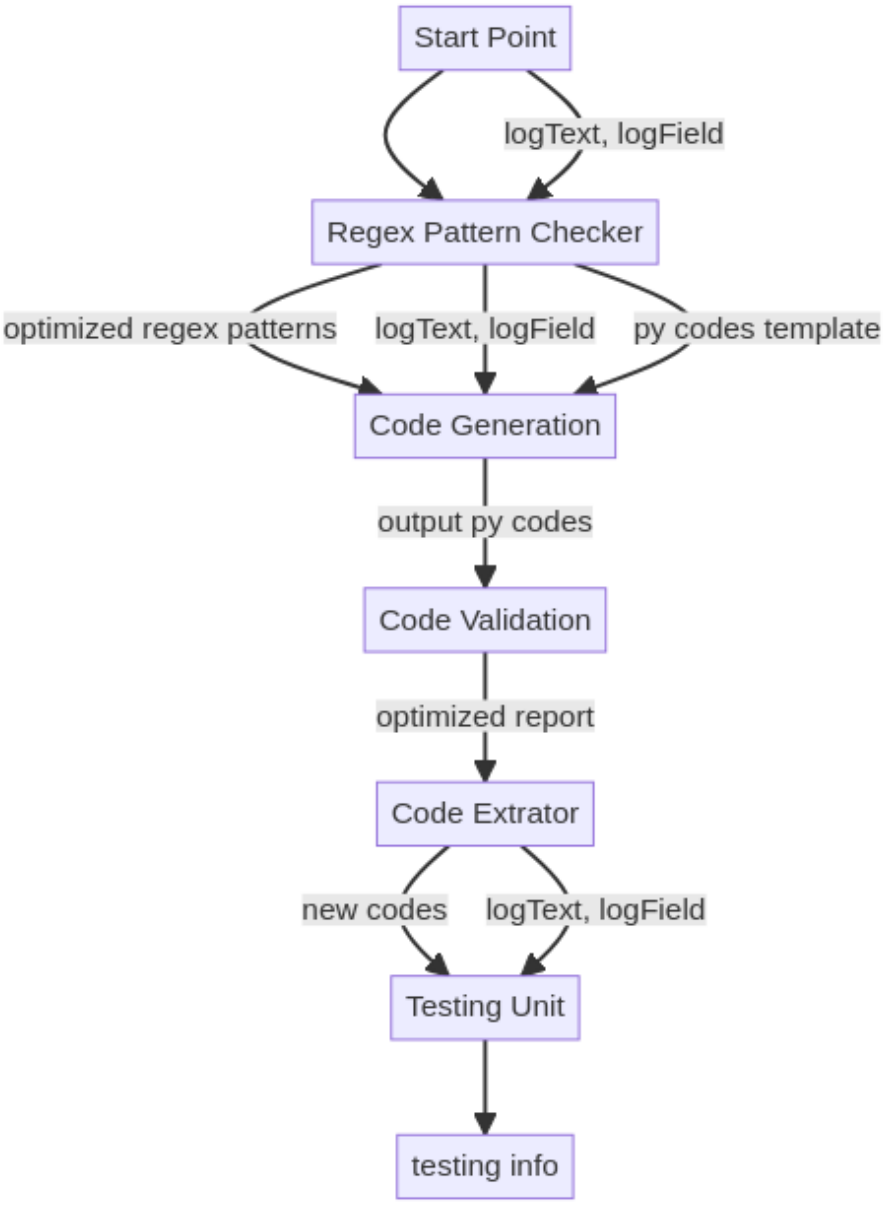

图 2: **LogParserX 输入输出流**

结果检验部分会将验证结果以清晰、直观的方式输出。输出的内容不仅包括匹配率和覆盖率等关键指标，还会对验证过程中发现的问题和不足进行详细说明，为 LogParserX 的优化和完善提供重要的参考依据。

3.2 实验对比

这里采用的主要评估指标覆盖率实际上是命中率，即生成的 logField 的字段能覆盖原始标签的比例，而剩下部分则为官方的比例计算。但是由于官方的完全匹配率非常严格，导致大部分的人工匹配分数不高。而采用大模型的实验部分也优先使用自测覆盖率进行比较，但是在提示词部分着重强调了尽可能的完全匹配结构。

3.2.1 人工知识库直接提取

该部分使用人工知识库直接进行提取，没有进行学习训练并直接使用开发集进行验证操作，包含所有部分正则的筛选，可以看到表4 的完全匹配率很低，几乎很难全部匹配，但是在命中率上表现良好。

表 4: 人工正则覆盖率与匹配率对比测试结果

日志索引范围	自测覆盖率	官方匹配率	官方完全匹配率	覆盖率 <70% 计数
0 - 5	91.0%	100.0%	0.0%	0
0 - 10	91.0%	100.0%	0.0%	0
0 - 100	84.6%	100.0%	1.0%	15
0 - 400	79.6%	98.2%	0.2%	106
100 - 200	81.1%	99.0%	0.0%	27
120 - 125	92.0%	100.0%	0.0%	0
200 - 300	66.9%	95.0%	0.0%	46
300 - 400	71.0%	99.0%	0.0%	39

3.2.2 LogParserX 提取

从开发集中生成了四种测试集用于后续的验证部分，经过 LogParserX 框架的智能体学习阶段，使用开发集中一半的比例用于训练（划分为四个种类后再取一半），验证部分也使用一半的数据集。

经过测试验证的结果如下表 5，可以看出在使用开发集进行验证的部分，官方匹配率和自测覆盖率较高，最好达到了 97% 和 81%；而在使用生成的测试集进行验证的时候，各个指标均有下降，其中原因可能是模型学习到的是局部特征，在对于一些特殊结构时未识别出来，但是尽可能输出多的 logField 用于覆盖原始的 logField 以保证较高命中率。

综上所述，在对比之后，计算得到 LogParserX 在开发集上的平均分数为 53.6，在生成测试集上的分数为 52.4，而人工正则最好结果为 41。LogParser 对于提升官方完全

匹配率效果好，而且官方匹配率并不低，覆盖率虽然没有人工正则高，但是整体评分高于人工正则。

表 5: LogParserX 覆盖率与匹配率对比测试结果

日志类别	验证集	自测覆盖率	官方匹配率	官方完全匹配率
Class 1 (0-50)	开发集	77.0%	96.0%	26.0%
	测试集	76.0%	96.0%	22.0%
Class 2 (100-150)	开发集	81.0%	97.0%	24.0%
	测试集	79.0%	96.0%	23.0%
Class 3 (200-250)	开发集	75.0%	95.0%	27.0%
	测试集	72.0%	95.0%	24.0%
Class 4 (300-350)	开发集	72.0%	92.0%	23.0%
	测试集	70.0%	92.0%	20.0%

4 创意说明

4.1 智能体合作

在智能体学习阶段，设计了三个智能体用于组织协作，后两个智能体的输入和输出都是上下文敏感的，因此可以接收到前两个智能体的输出作为自身工作的输入，极大的简化了代码的书写要求和设计复杂度，并且为每个智能体设置了记忆，可以更轻松地从之前的交互历史中学习到经验。在三个智能体的互相协作下，能够更好地学习样本标签并生成代码。

4.2 预处理知识库

在智能体学习阶段输入提示词部分的先验知识来源于预处理知识库，即使用人工正则表达式和代码模板来提示模型该如何编写适合当前优化的正则表达式的代码。在用于代码生成任务的前期准备，省略了过多重复学习的操作，只需要从人工提取的正则中寻找未命中的正则表达式，并生成新的适配正则，传递给代码生成代理用于生成完整代码即可。无须再次生成多个重复的正则表达式用于下游任务。

4.3 中间产物用于优化

在智能体学习阶段，中间产物有如下几项：优化的正则表达式，使用优化的正则表达式的生成代码，最后优化的代码，最终优化报告。其中一些产物可用于智能体自我学习的下游输入，减少重复错误，比如优化的正则表达式和初始生成的代码。此外，初始

生成的代码还可以用于中间截断验证部分，可以在中间部分提前验证代码匹配的正确性，对于未来进行结构优化有好处。

Listing 1: 智能体任务编排

```
# 多智能体协同配置示例:
class MyTest():
    def __init__(self):
        # 配置模型参数
        self.llm_config = {
            "model_name": os.getenv("MODEL_NAME"),
            "api_base": os.getenv("OPENAI_API_BASE"),
            "api_key": os.getenv("OPENAI_API_KEY"),
            "temperature": 0.3,
            "max_tokens": 4096
        }
    def _init_llm(self):
        """初始化大模型实例"""
        return ChatOpenAI(
            model=self.llm_config["model_name"],
            openai_api_base=self.llm_config["api_base"],
            openai_api_key=self.llm_config["api_key"],
            temperature=self.llm_config["temperature"],
            max_tokens=self.llm_config["max_tokens"],
            streaming=False  # 关闭流式避免中断
        )
    @agent
    def researcher(self) -> Agent:
        return Agent(
            config=self.agents_config['researcher'],
            llm=self._init_llm(),
            verbose=True,)
    @task
    def research_task(self) -> Task:
        return Task(
        config=self.tasks_config['research_task'],)
    @agent
    def reporting_analyst(self) -> Agent:
        return Agent(
        config=self.agents_config['reporting_analyst'],
        llm=self._init_llm(),
        verbose=True,)
    @task
    def reporting_task(self) -> Task:
        return Task(
            config=self.tasks_config['reporting_task'],
```

```
42              output_file='refs/mytest/output/report.md')
43      @crew
44      def crew(self) -> Crew:
45          return Crew(
46              agents=self.agents,
47              tasks=self.tasks,
48              process=Process.sequential,
49              verbose=True,)
```