QCon
全球软件开发大会

InfoQ 极客传媒

把 50% 值班时间抢回来：字节 SRE Agent 从 0 到 1 的降噪与排障实战

董善东

20251024

个人&团队简介

董善东 博士
字节跳动Dev Infra-观测平台算法负责人

深耕 AIOps & 可观测行业多年， 在异常检测、根因分析、Agent 应用等领域有比较深入的行业认知和产品功能 Build 经验。 曾就职于腾讯云、阿里云。

团队面向整个字节内部开发者提供观测平台，与多个团队协同构建Metrics/Traces/Logs/Events等数据埋点&加工链路&存储，并基于此提供一站式的监控、报警、日志、链路追踪、根因分析等产品化能力

目录

InfoQ 极客传媒

北京

QCon 全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

北京

AiCon 全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

上海

QCon 全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月 — 5月 — 6月 — 8月 — 10月 — 12月

上海

AiCon 全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

深圳

AiCon 全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

北京

AiCon 全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

01

现状与痛点

现状简介

现状和挑战：
- 需服务内场最大的用户量、业务量， 业务多样性大， 很多业务处于狂奔
- 观测平台和业务部门的稳定性平台的合作与“竞争”
- AI时代更需要直面业务的最终需求来解决问题

双线并行、相辅相成、增强LUI/Agent化

保持产品演进长期方向，建设好基础能力，同时为AI铺路

提高AI投入，通过AI应用降低使用成本，同时辅助用户理解产品设计

增强LUI的全新交互能力、逐步将各类应用Agent化、通过Agent实现发现-分析-处理的闭环

产品演进	数据标准	覆盖度	平台能力	场景化应用	LUI
抄近路 - AIOps + LLM	自动化	智能化	内置RCA	各处集成	Agent化

告警值守的4大环节和痛点

告警噪音突出、业务分析难、
个性化知识加载困难

发现异常和分析

- 业务问题的分析比较难。
- 业务排查经验无法沉淀和复用。

告警接手和处理

- 平台存在大量注入规则，大量告警无人查看和处理
- 噪音和重要告警传统手段区分不明显。

告警协同和复盘

- 告警上下游排查和协同成本高
- 告警和故障复盘总结消耗太大

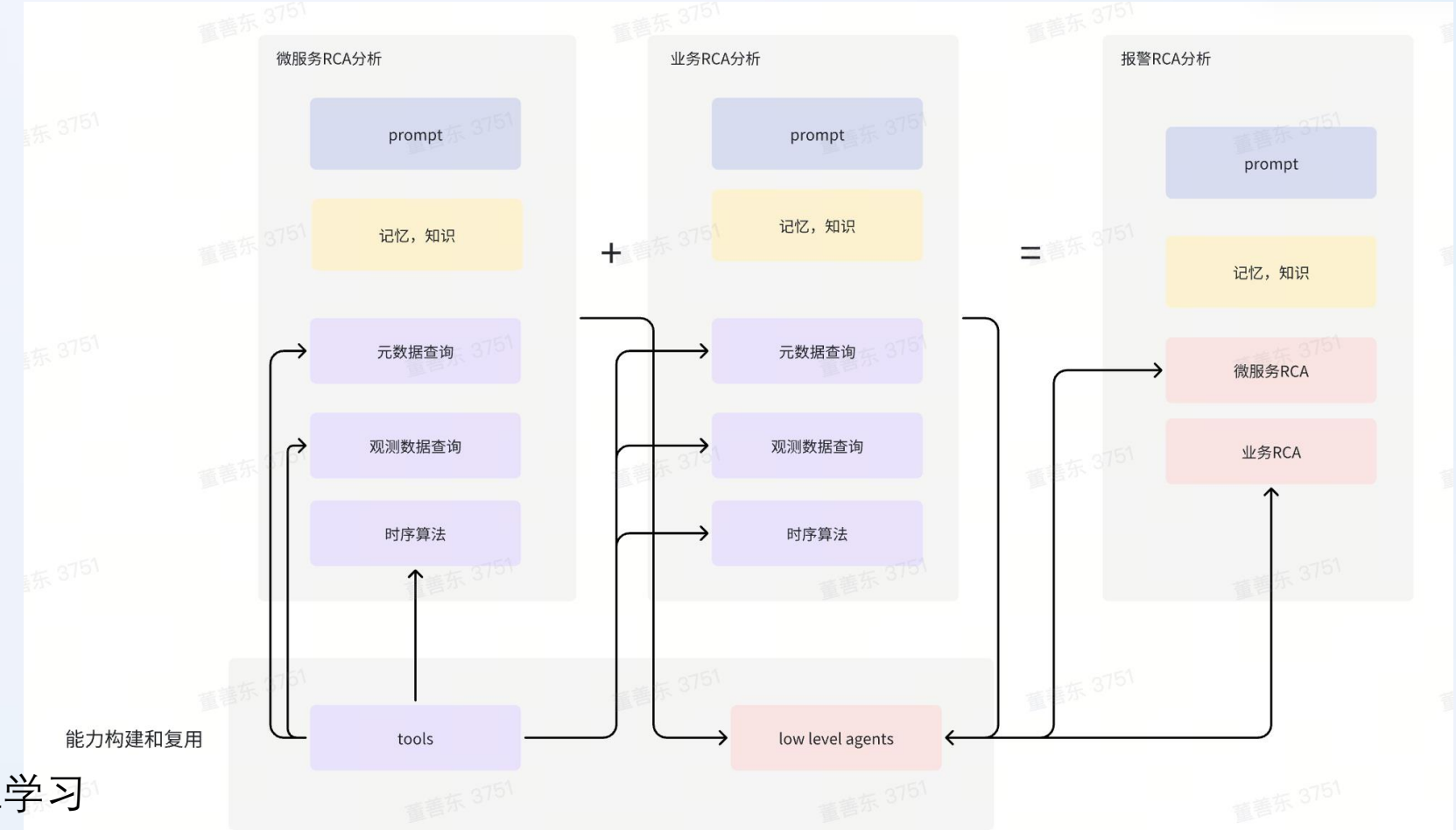
告警优化和预防

- 有些告警明显规则不合理， 但是人没有精力和能力进行规则优化调整

Agent引入来解决这些痛难点，有哪些优势?

1. 优秀的观测平台+AIOps基础:
- 观测数据标准化、全面准确的元数据
- 异常检测、RCA分析、告警聚合面向微服务的检测分析做了很好的积累和内部落地
- 打造了low/mid/high level 的tool service

2. LLM 作为一个通用大脑，已经展现出语义理解、复杂任务规划与拆解、工具调用、自主决策与学习方面又得到了增强。

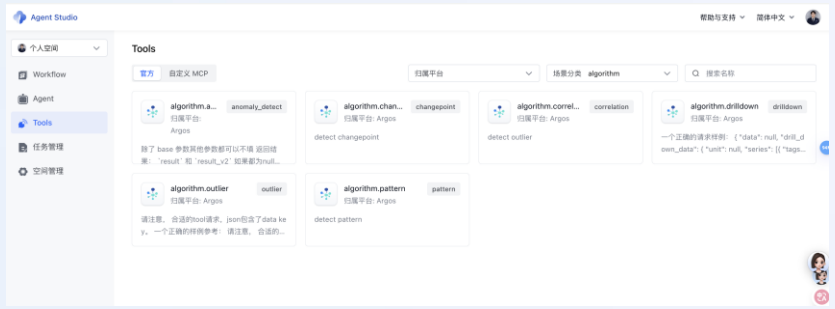


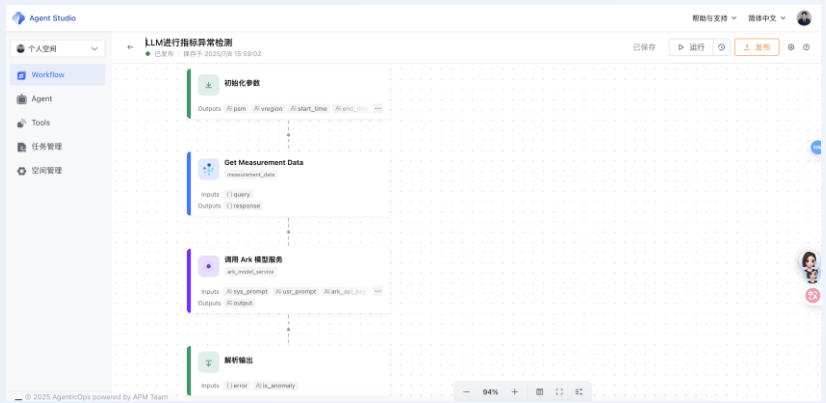
LLM + Agent范式 + 领域Tools + 知识&学习

02

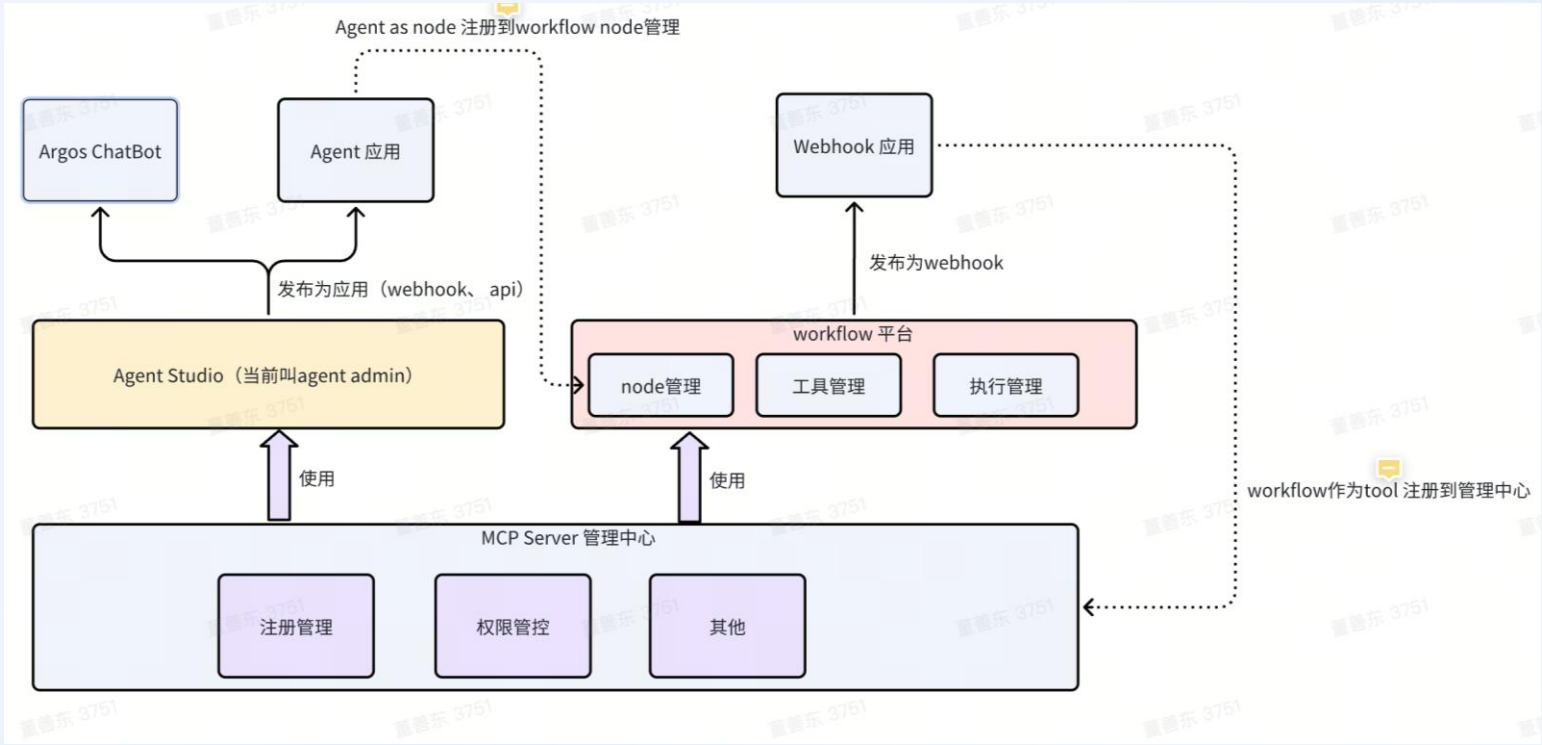
产品架构演进

产品层：Tools-->Workflow-->Agent/Agent Studio

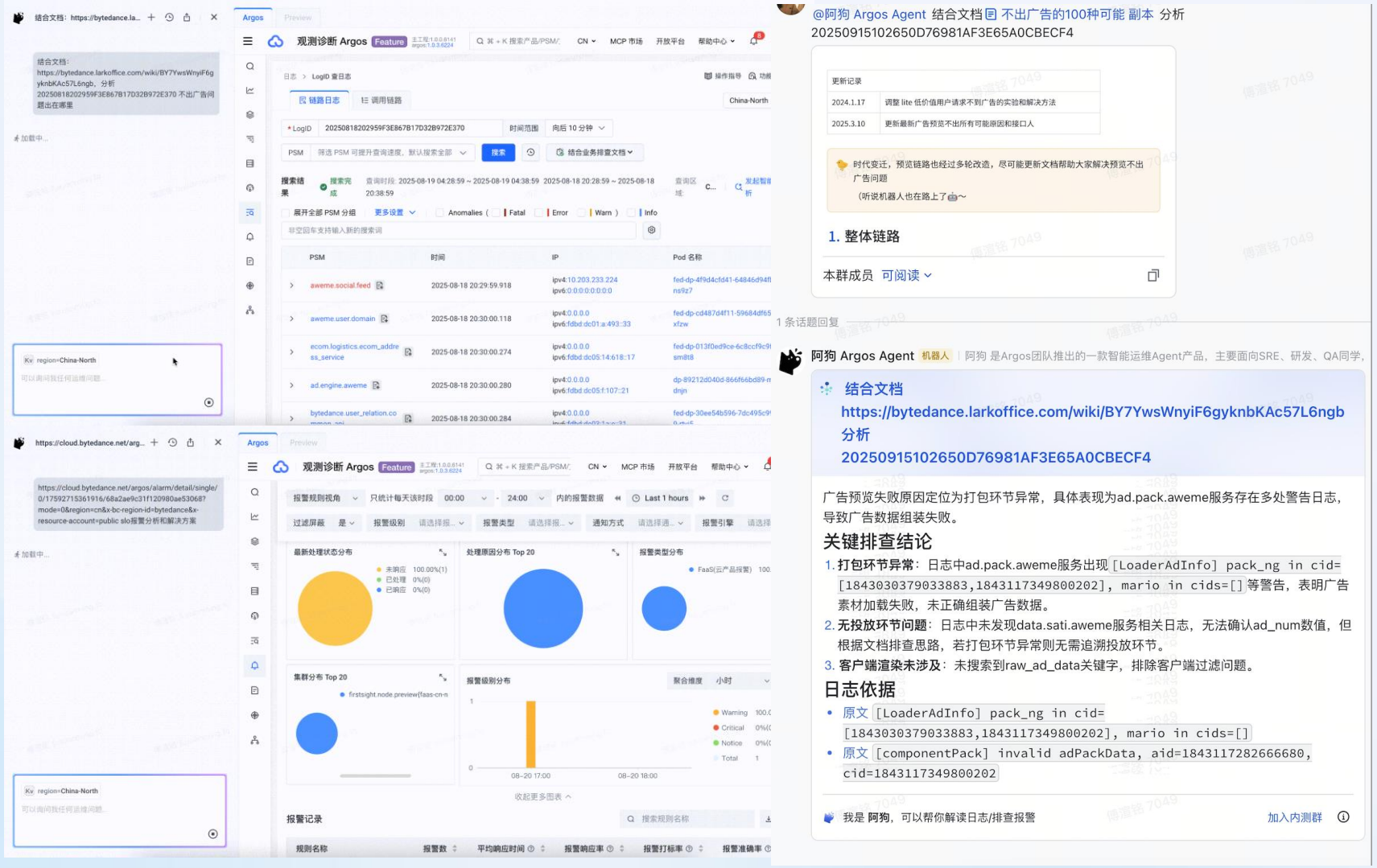




1. APM as a service->MCP Tools 工具集合
2. 流程编排的 Workflow
3. Agent Studio



Web & Lark LUI



web端：
- 结合页面context，进行进一步的分析

移动端：
- 结合告警的context，进行分析

SRE Agent平台架构

业务场景

发现：大盘巡检 | 异常准入 | 自动ACK

分析：指标分析 | 根因分析 | 告警建议

解决：告警处理 | 告警协同

总结&风险预防：日报 | 故障复盘 | 告警规则优化

Agent执行架构

通用Agent

场景agent

Plan Agent | 反思Agent | 报警值守 Agent | RCA Agent | 指标查询Agent | LogID解读Agent | trace分析agent

Agent执行与记忆

AgentPattern
- Basis Agent | Classifier
- Supervisor | Workflow

Plan&Exe执行器
- Plan | ToolCalls
- CodeAct | DAG

Agent 通信
- Agent IO | 多层级Agent
- 引用语法糖 | Agent As Tool

Context 管理
- 短-中-长记忆
- 压缩 | 记忆管理

学习机制

知识管理
- 知识提取
- 知识管理

Agent center

接入层
- 飞书BOT
- Argos LUI

Config
- Agent Config
- Trigger Config

MCP Tools
- data tools | analysis tools
- algorithm | 开放集成

调度
- Human Chat | Trigger
- Schedule

观测平台

APM as a service

原子算法 | 单实例分析 | 流量来源分析 | panic 分析 | 可用性分析 | 延迟上涨分析

观测数据：Metrics | Logs | Traces | Profilings | Events

元数据：服务技术栈 | 组件元数据 | 拓扑关系

LLM相关平台

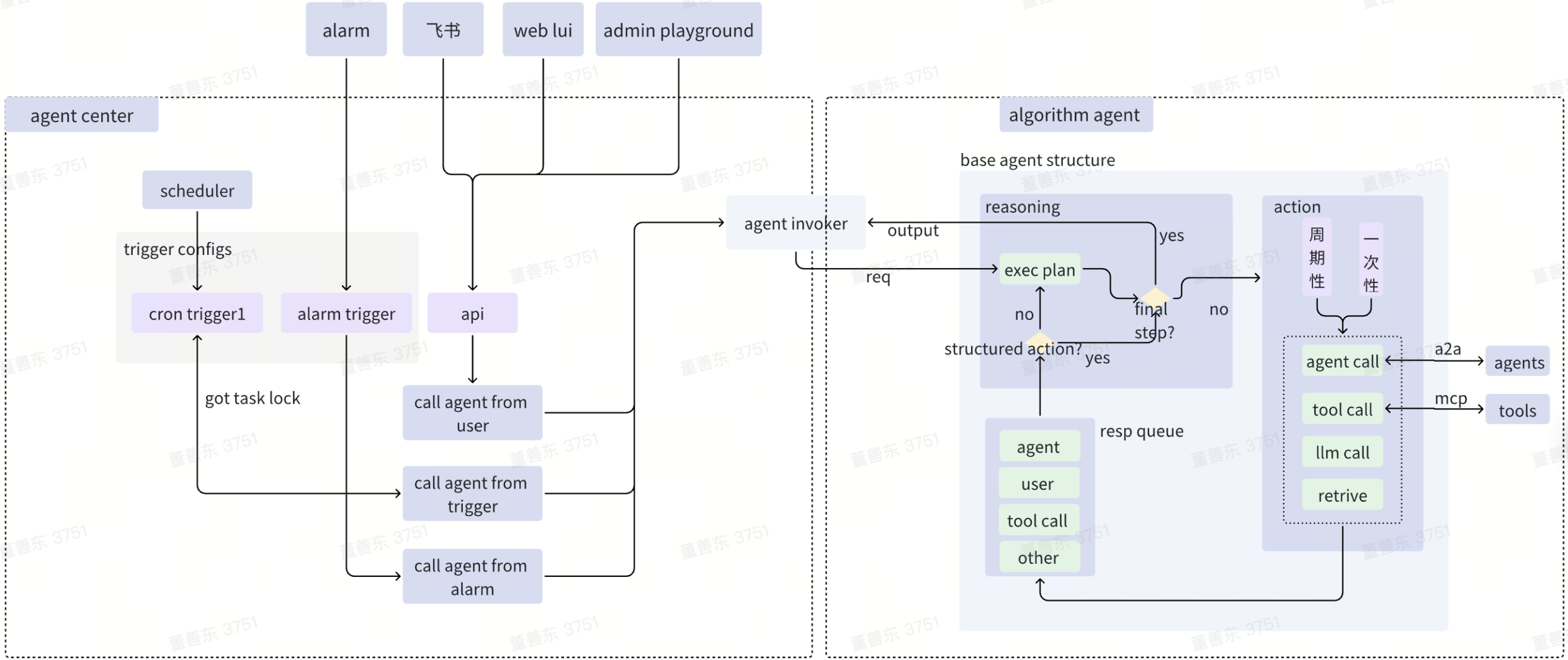
LLM 评估

场景评估 | Prompt优化

model service

模型基座 | SFT

Agent执行请求



SRE Agent的探索和踩得坑

定位： 打造SRE Agent， 解决繁琐、重复的劳动

总结： 踩过的坑

Agent关键点： Agent 执行的准确率、如何提供个性化能力（Memory）、超长上下文（Context Engineering）

多轮自动化执行（如流程类React）的准确率不稳定

尝试自动化执行业务排障 SOP、多轮流程（如 React 类操作）时，复杂场景下执行准确率不足，无法可靠替代人工步骤。

单一场景（RCA Copilot）多轮交互的精准度与实用性不足

初期聚焦 RCA Copilot 单一场景时，多轮交互式分析的准确率、场景覆盖度未达预期，难以真正替代人工高效定位问题。

踩坑经验

场景tool在封装粒度上的tradeoff

随着值守、巡检等场景增多，工具的“组织粒度”（过粗缺乏灵活性，过细则维护成本剧增）成为瓶颈，难以高效支撑多场景复用。

其他坑

观测数据过期，辛苦打标的case 样本失效

历史经验如何承载， 选择了告警规则上有的处理SOP，以及从历史的处理（对话）记录中学习出SOP

03

噪音告警识别和处理

噪音告警的定义与持续学习

噪音告警的定义

噪音告警通常是指对系统稳定运行无实际影响、可通过自动化手段快速处理的告警，如重复告警、低级别无关告警、快速恢复的毛刺告警等。

传统手段一般通过： 时序异常程度的方式来判定是否为噪音。

噪音判定的方式

告警的基础特征： 指标异常程度、pattern、发送频次、级别等

历史知识：告警历史的对话记录、处理经验

从对话、处理历史中持续学习

随着处理记录的越来越丰富， 整个知识系统也越来越丰富

噪音判定的特征工程

分类	策略名称	定义	适用场景	示例
客观指标	指标突刺	指标数据在短时间内迅速满足触发条件并快速恢复的瞬时波动	网络抖动、硬件临时故障等导致的短暂异常。	某接口响应时间突然升至500ms（阈值300ms），但10秒后恢复正常，触发指标突刺降噪。
	持续报警	指标数据长时间持续触发报警或循环触发/恢复，形成冗余通知。	慢性故障（如磁盘空间持续接近阈值）、周期性环境波动。	某服务器CPU使用率连续8小时超过90%，后续报警每小时仅通知1次。
	重复报警	同一类报警在短时间内高频次触发（基于规则或标签分组）。	接口频繁超时、日志重复错误等场景。	某数据库连接失败报警每分钟触发20次，超出阈值（15次/分钟），后续报警每5分钟发送1次。
	未知报警	历史上极少触发的规则或标签组合报警（避免漏报新异常）。	新上线服务、罕见故障模式。	某日志监控规则首次触发error级异常，即使频次低也不降噪。
	其他特征： 过于细粒度的报警、信息过载、低阅读率、用户打标数据、 指标的异常程度得分...			
主观感知	报警级别与用户耐受度	不同优先级报警的噪声判定阈值不同，用户对同一级别的耐受度存在差异。	生产环境与预发环境的报警分级处理。	某金融交易系统中，Critical级别的支付失败报警始终不降噪，而Warning级别的日志异常在夜间自动屏蔽。
	报警时段敏感性	用户对不同时间段的报警敏感度不同（如夜间仅接受高优报警）。	夜间运维值班、敏感时段（如会议期间）的报警管理。	某医院HIS系统在凌晨1:00-5:00仅允许服务器宕机报警（Critical），其余报警暂缓通知。
	个性化降噪	用户自定义噪声规则，覆盖特定场景下的差异化需求。	多租户环境、复杂业务场景的定制化管理。	某游戏公司运维团队自定义规则： 1. 游戏高峰期（20:00-24:00）放宽接口响应时间阈值 2. 屏蔽非战斗相关的日志报警。

知识提取与经验应用

历史对话积累和解析

批量将历史报警中的对话、处理记录进行获取和清洗等预处理。

从历史对话提取出知识

通过 LLM 提取关键信息，并封装为知识检索的tool， 为后续的知识沉淀和经验学习提供基础。

将非结构化文本记录转化为标准化格式的判定规则，使得处理记录更加规范、易于管理和分析。

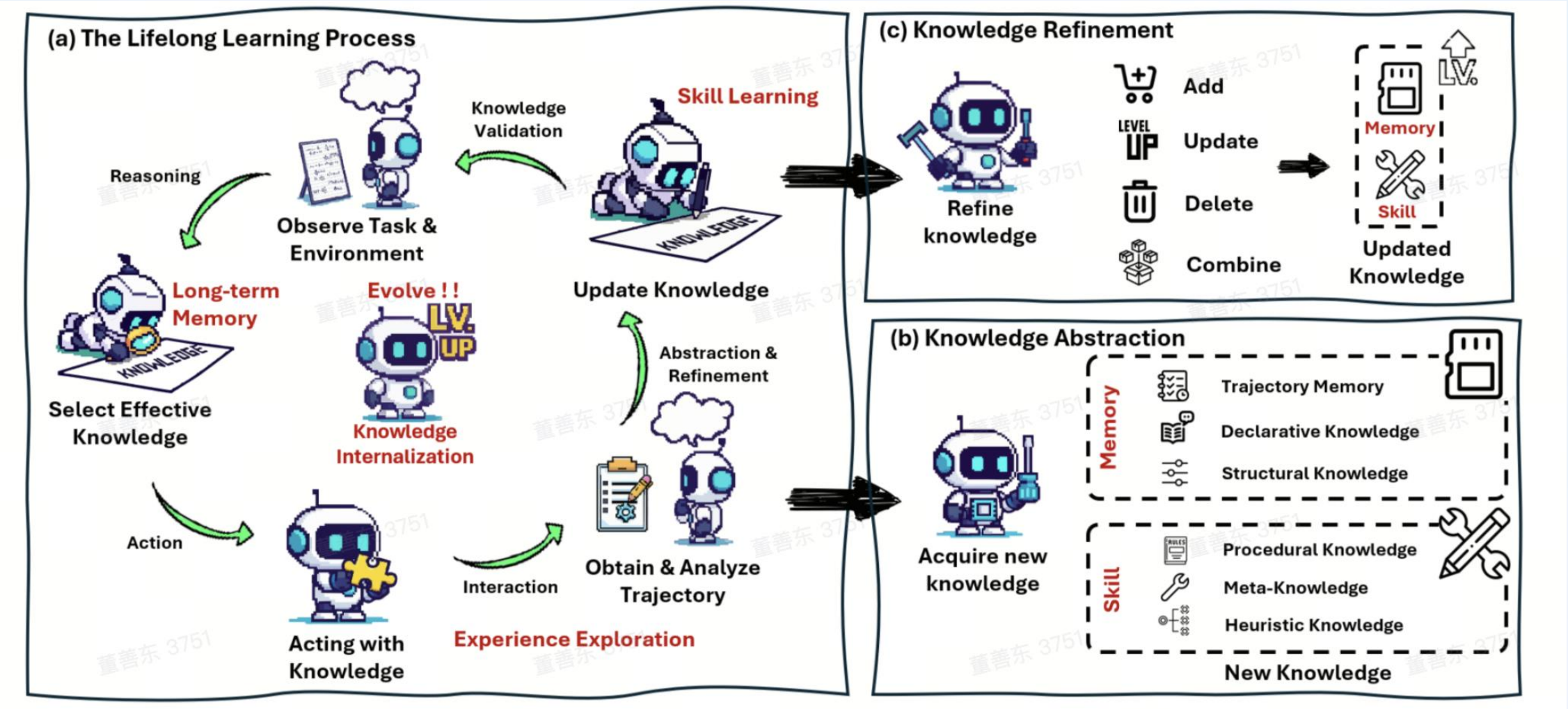
知识更新与规则优化

新的告警的对话和处理记录， 持续的从其中提取知识。

将结构化记录写入知识引擎的 “专家经验库”，实现知识的不断更新和规则的持续优化，提升 SRE Agent 的噪音识别能力。

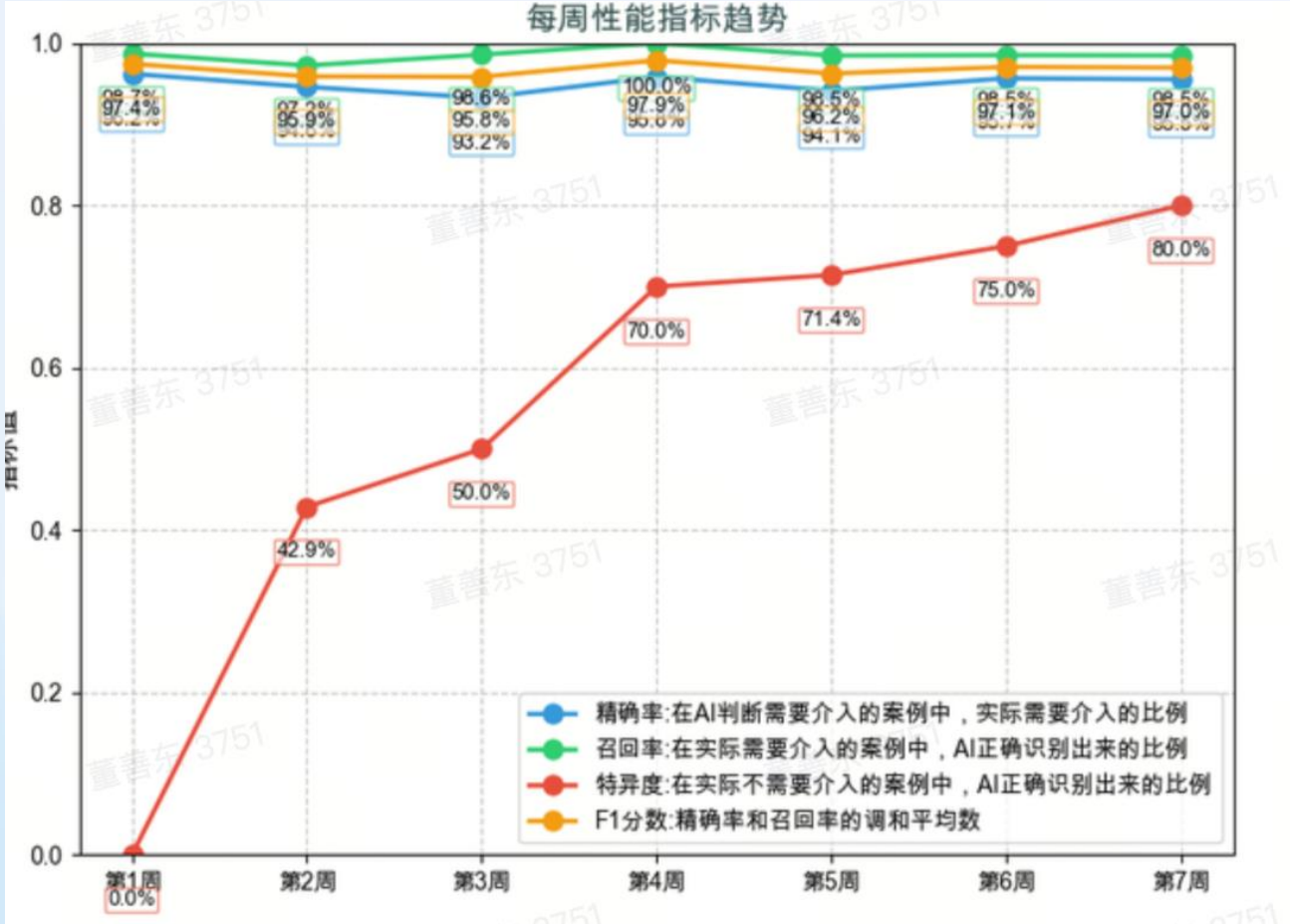
question	提取知识	llm_answer
针对报警规则[3518xx]【P1】【火山MongoDB】节点CPU使用率过高+时间信息+告警维度变量，我该怎么做?	@user_name: xx类的告警为周期性告警，进行持续关注即可，无需升级处理	结论： 需要人工介入 依据： 历史处理知识包含“xxx”,根据xx推导得出

持续学习机制



参考： https://arxiv.org/pdf/2508.19005

持续学习历史告警处理方式的效果

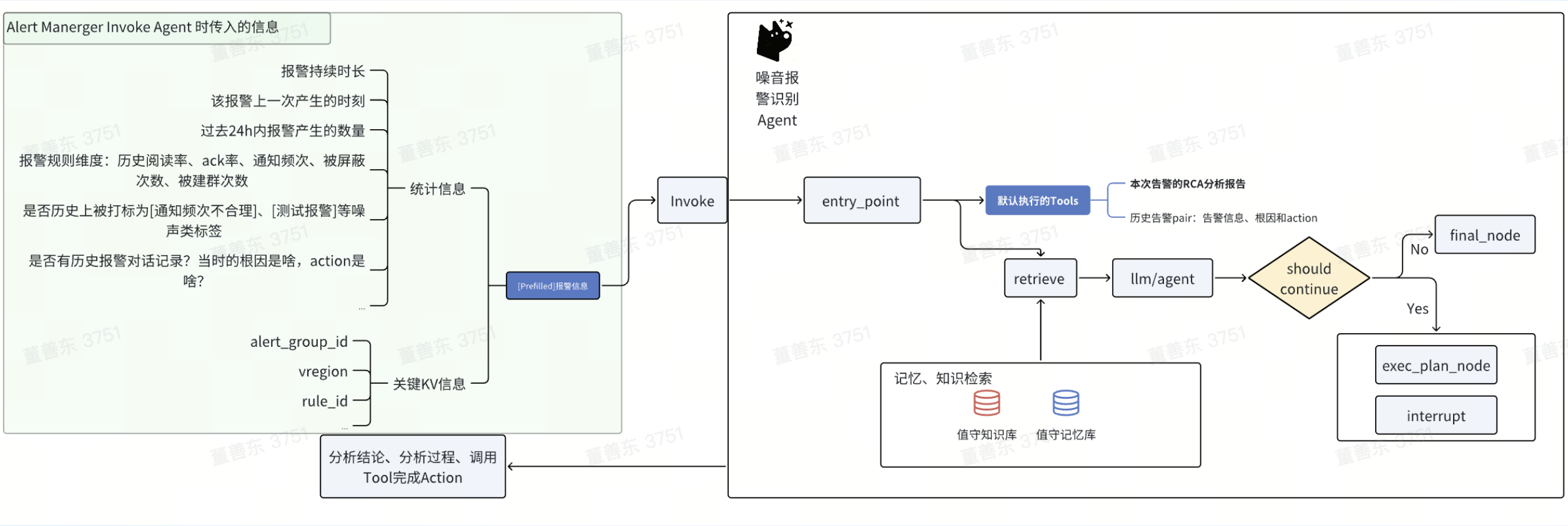


每周正负样本分类情况

周次	正样本数	正确分类数	召回率	负样本数	正确分类数	特异度
第1周	77	76	98.7%	3	0	0.0%
第2周	72	70	97.2%	7	3	42.9%
第3周	70	69	98.6%	10	5	50.0%
第4周	69	69	100.0%	10	7	70.0%
第5周	65	64	98.5%	14	10	71.4%
第6周	67	66	98.5%	12	9	75.0%
第7周	65	64	98.5%	15	12	80.0%
总计	485	478	98.6%	71	46	64.8%

正样本：需要干预的案例 (TP+FN)
负样本：不需要干预的案例 (TN+FP)
召回率：正样本分类准确率 = TP/(TP+FN)
特异度：负样本分类准确率 = TN/(TN+FP)

噪音判定Agent



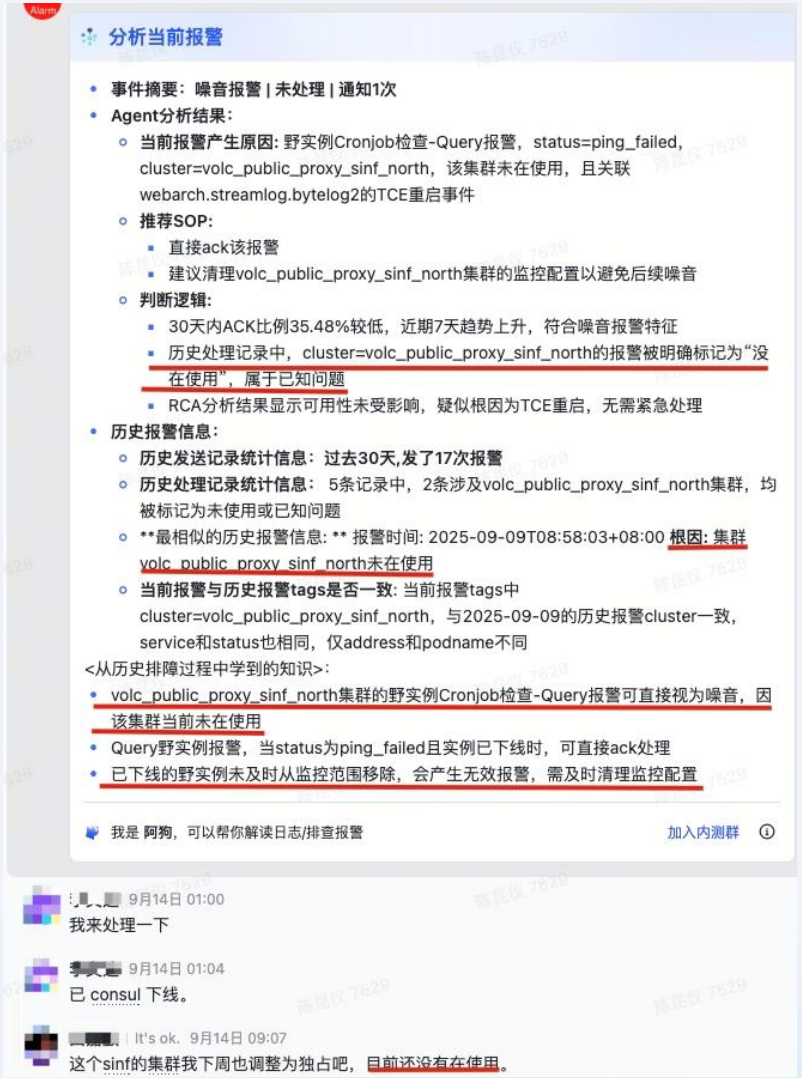
Agent Prompt 示例：

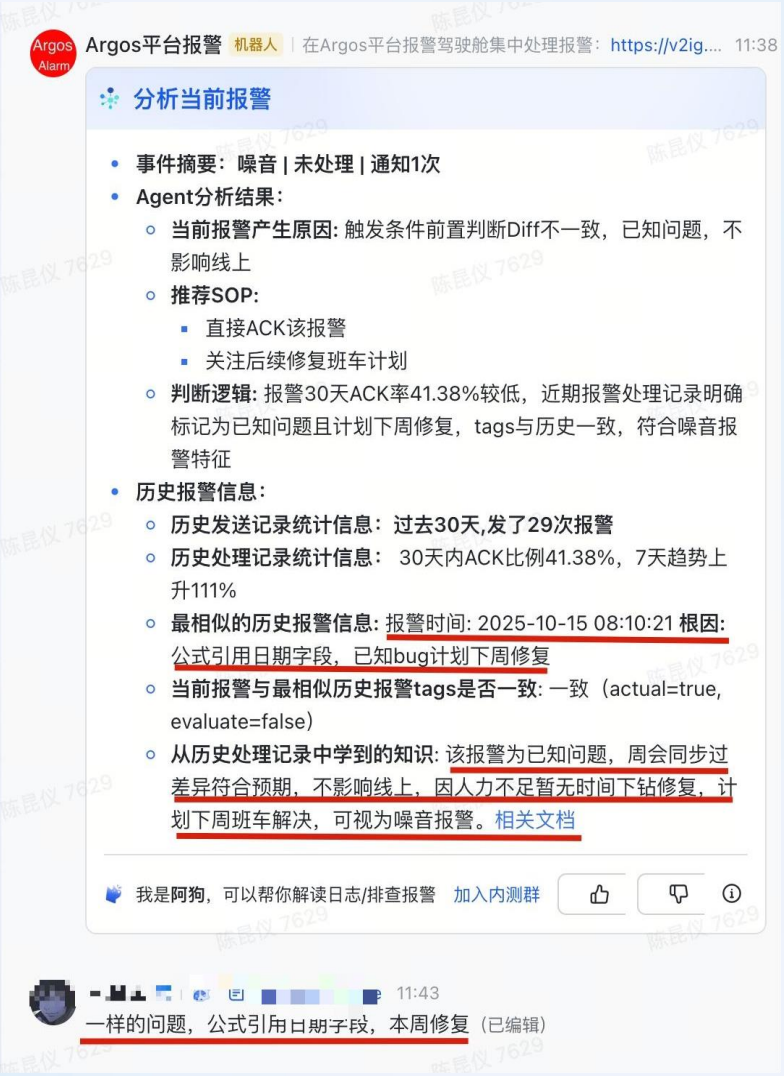
1. 角色定义： 噪音告警识别专家
2. 工作流程： 利用xx获取报警基础信息 --> 利用xx tool 获取获取报警历史处理经验 --> 根据用户给你的信息、rca报告、历史处理记录来综合判断，当前的报警是不是噪音报警

噪音告警分析样例

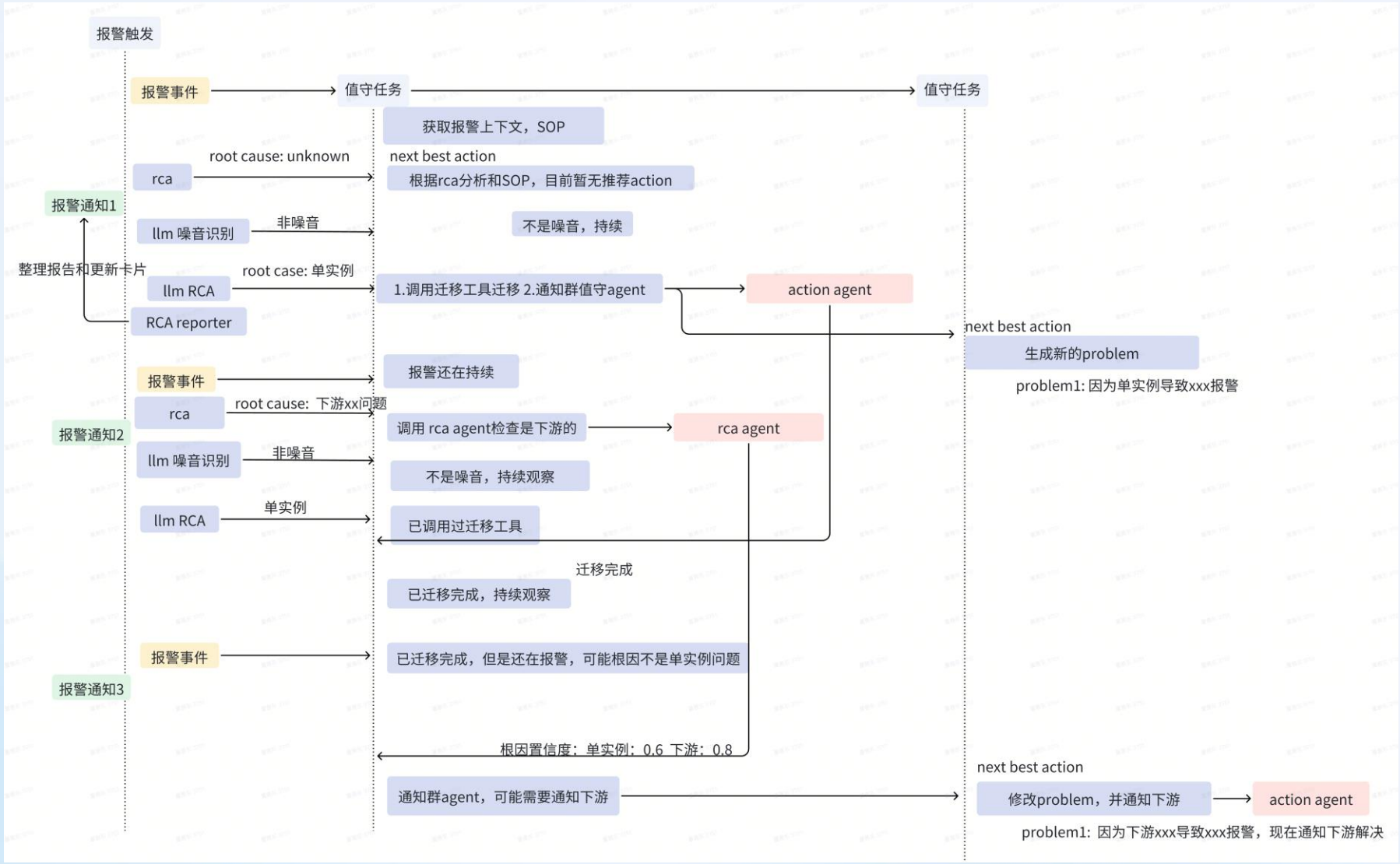
核心能力：
- ✓ 自动判断报警是否为噪音
- ✓ 智能分析报警产生根因
- ✓ 推荐标准化处理流程
- ✓ 展示相关历史报警记录
- ✓ 提炼历史处理对话精华

值守case示例：
根据对话可得，分析与最终人工排障结论一致





单次告警处理--> 告警值守的演进



- 多Agent协作：实现噪音分析、告警RCA、 告警Action的Agent

- 多Agent协作下的Context管理挑战

- 工程架构保证：长任务能力、告警状态管理、告警关联能力、action后的恢复校验机制

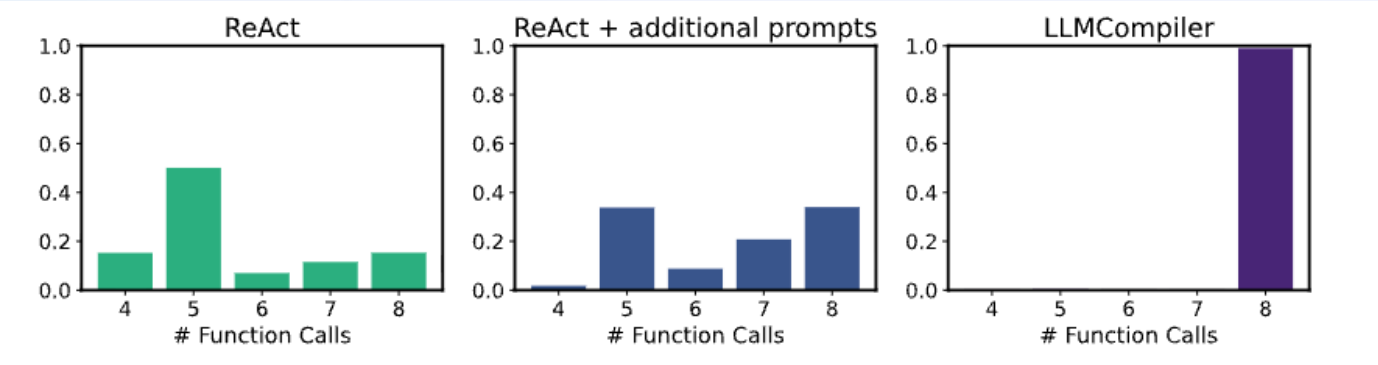
04

业务自定义分析

面临的挑战

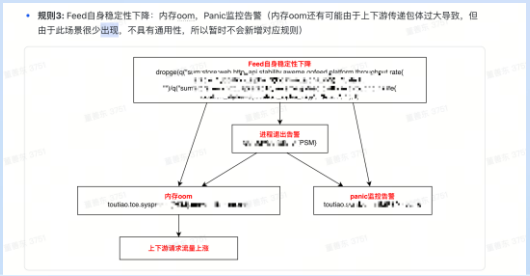
当前以大语言模型（LLM）为核心的智能系统在处理简单指令时表现优异，但在SRE复杂业务场景下面临以下问题

1. ReAct 架构的适用边界清晰: 1-3轮交互效果还符合预期， 超出则执行准确率明显下降。

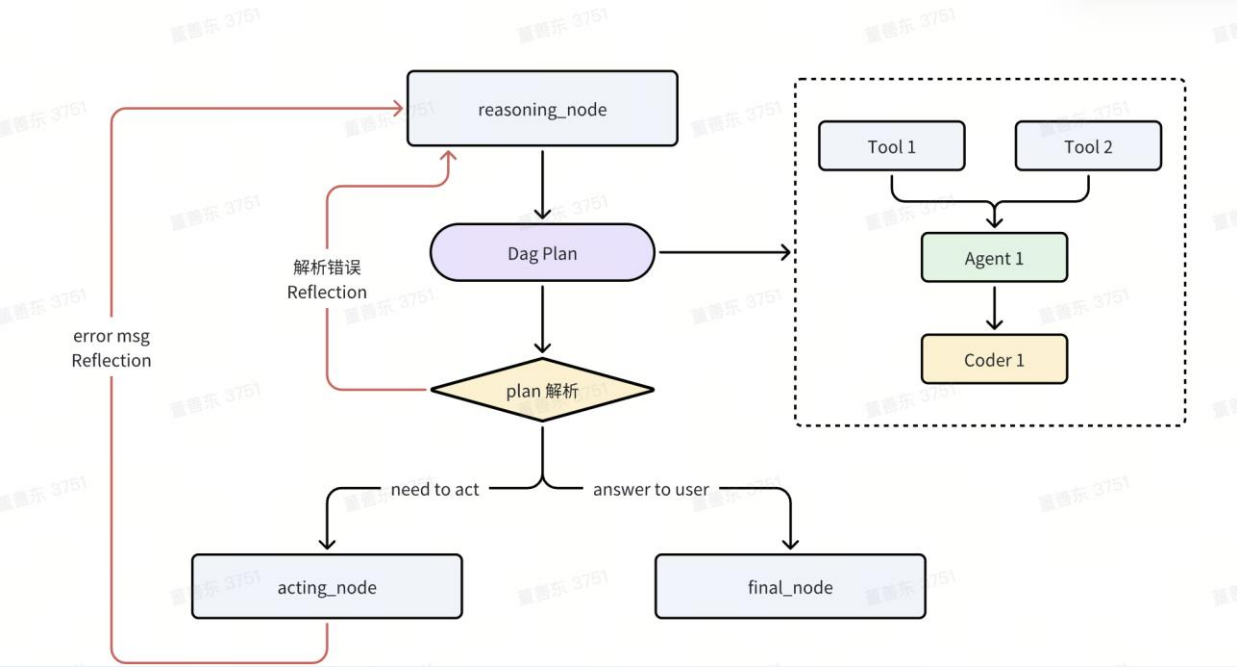


2. 对自定义的分析和排查的自然语言，执行的诉求明确：我们期望在工具完备的前提下，AI 能够支持自定义 SOP（标准作业程序）语言的执行。这一能力的实现，将有效解决自定义 RCA 分析及 Action（动作）执行的问题，让 AI 在更贴合实际业务需求的流程中发挥作用。

自定义分析和处理的真实示例

告警	自定义处理SOP	分析
TCE 实例进程退出报警	主进程发生了异常退出（ExitCode != 0），请查看服务日志或 systemd 日志，更多说明参考 TCE 业务进程异常退出追查思路[文档]	- 关联到相关日志 - 需要从文档中检索相关检查SOP
[P0] [OpenAPI-SLO] streamlog-OpenAPI查询SLO告警(>10%)	通常是-->访问Bytelog异常： 1. Check 是否 底层bytelog单节点问题？ Metrics:proxy.thrift_writer 的tag: server_pod 2. Check 是否 底层bytelog单节点OOM？(新指标)Metrics:tce oom_cnt 的tag: _cluster=* 3. Check 是否 底层bytelog单节点OOM？(旧指标)Metrics:tce oom.pod 的tag: paas_cluster=* 4. Check 错误码 底层bytelog返回错误码70x，Metrics: proxy.thrift_writer 的tag: thrift_resp_code 或 result	- 需要通过指标，进行像单实例、离群分析、错误码分析、等。
feed 成功率下跌告警		根据文档中的SOP文档， 提取分析SOP

一、 DagAct: 解决自定义问题的规划、执行架构



代码块 JSON 取消自动换行 复制

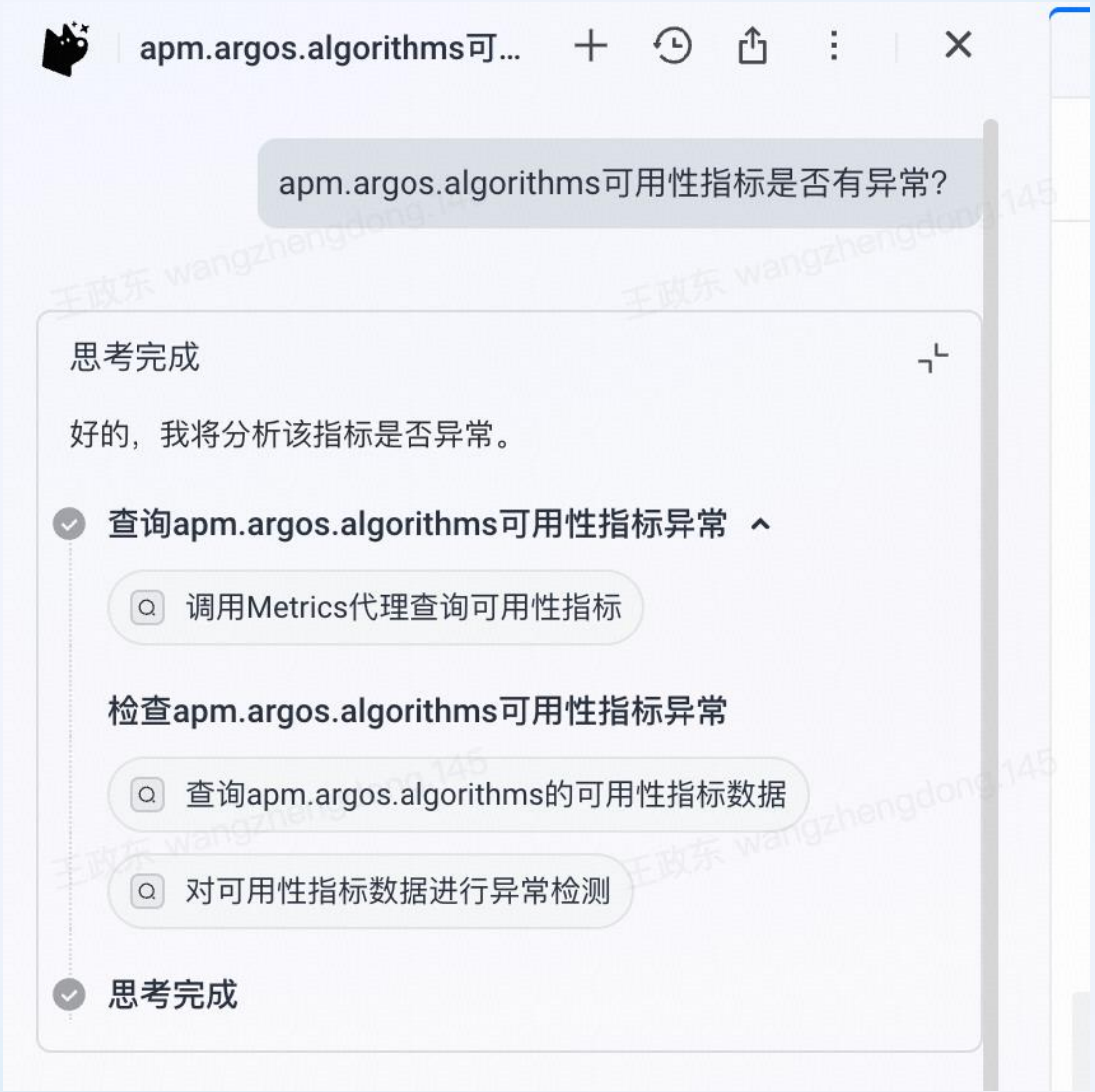
```
{
    "dag_name": "detect_success_rate_anomaly",
    "prepare": false,
    "title": "检测成功率异常点",
    "actions": [{
            "depends": [],
            "name": "get_total_data",
            "description": "调用工具获取总请求数指标数据",
            "tool_name": "metrics.data",
            "agent_name": "",
            "arguments": { ...
            }
    }, {
            "depends": [],
            "name": "get_success_data",
            "description": "调用工具获取成功请求数指标数据",
            "tool_name": "metrics.data",
            "agent_name": "",
            "arguments": { ...
            }
    }]
}
```

- actionDAG的格式规范
- 子Agent/tool 的使用规范
- 前后参数的引用传递

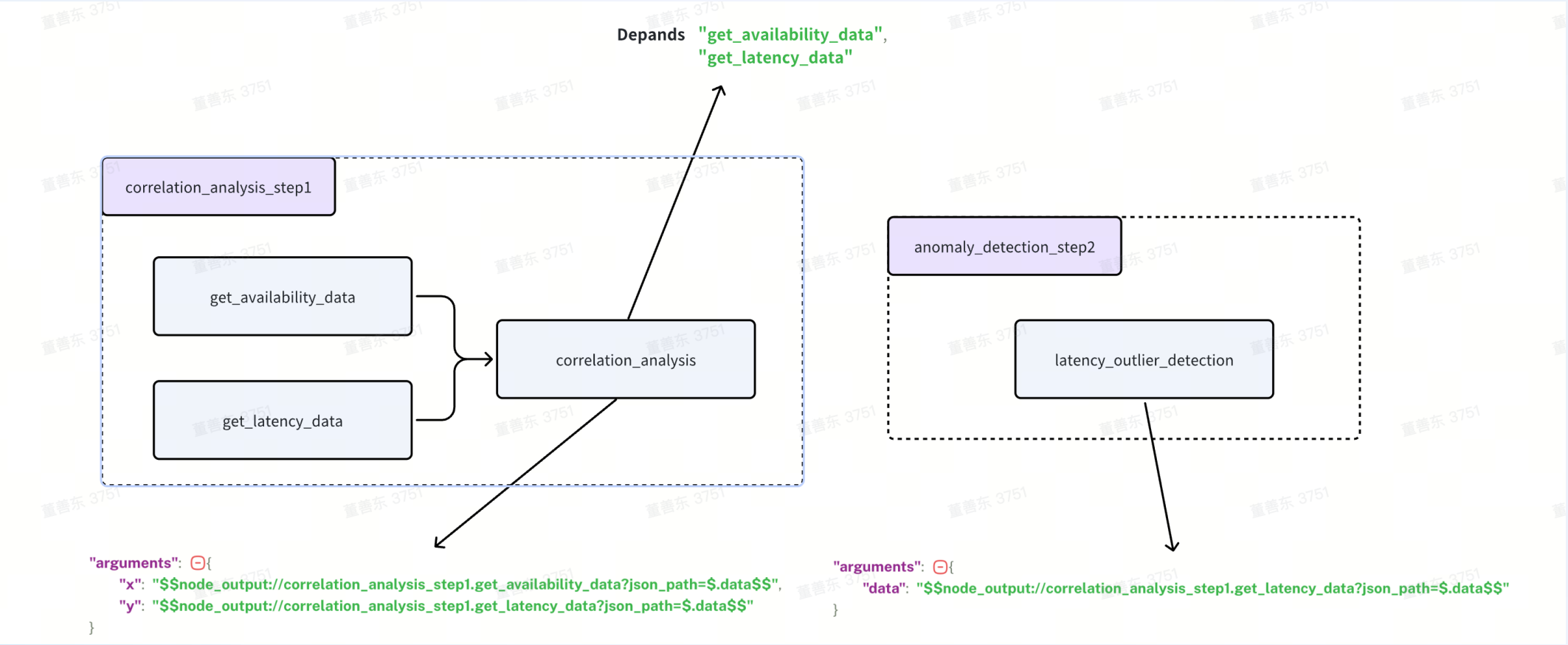
ActionDag 设计思考:
将原本泛化的、模糊的自然语言， 通过约束ActionDag生成的方式， 转变化规划一个有序、更加结构化、可复用、可验证的执行流程。 这样一个执行流程类似动态构建的workflow一样， 确定性更高。
在这样一个流程中， 可以标明在这个环节在做什么（dag_name、title）、 执行流程（actions）、前后的依赖项（depends）、 执行的工具 或者 agent（"name": "get_total_data"）， 以及工具所需要的入参（arguments）。

增强ActionDAG生成和执行准确率

在Reasoning节点中，我们会让LLM依次完成以下几个工作：
- ack：对用户任务的简单响应
- thought：对于本次任务的思考，LLM需要基于上下文进行以下几个方面的思考
  - 理解用户的需求
  - 思考使用的Tool/Agent
    - 参数的完整性校验
  - 对于历史Action的反思和总结
  - 判断是否满足用户的需求
- action_dag: 一个可执行的动作有向图
- todo：待办事件列表
- final_output：最终的回答
- interrupt：当前的任务无法执行，需要向用户进一步询问



定义语法糖， 实现参数传递的引用



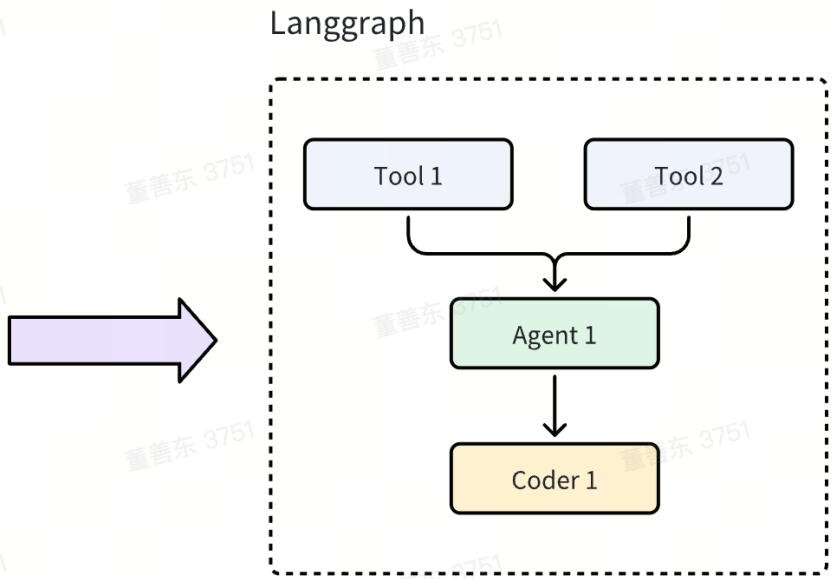
引用是DagAct环节中十分重要的一环，如果没有引用，那么LLM将会看到Node的所有输出，并且需要输出Node的所有输入，整个效果将会降级到ReAct模式

Reasoning和Acting解耦

Reasoning 节点负责生成待执行的actionDAG list。
Acting节点中，我们将会基于LLM生成的执行计划进行执行。首先我们会基于ActionDag生成一个LangGraph执行图。该图的每一个Node就是ActionDag中的每一个Action，而每一个Edge就是由depends参数来制定的。

代码块

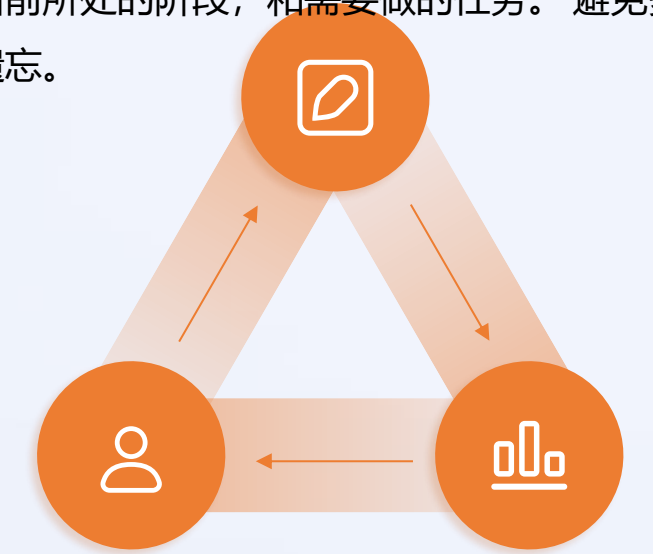
```
dag_name: str
title: str: 整个ActionDag的简要描述总结
actions:
 - name: action的名称
   description: 行为的描述
   tool_name: 依赖的tool name
   agent_name: 依赖的agent name
   arugemnts: dict|str 参数
   depends: list[str] 每一个item是依赖的node的名称，语法为 dag_name.action
```



DAGAct总结

引入全局规划

通过TODO list提供全局规划和任务，当前已经完成的任务。明确当前所处的阶段，和需要做的任务。 避免多轮task执行中遗忘。



think-tool增强执行准确率

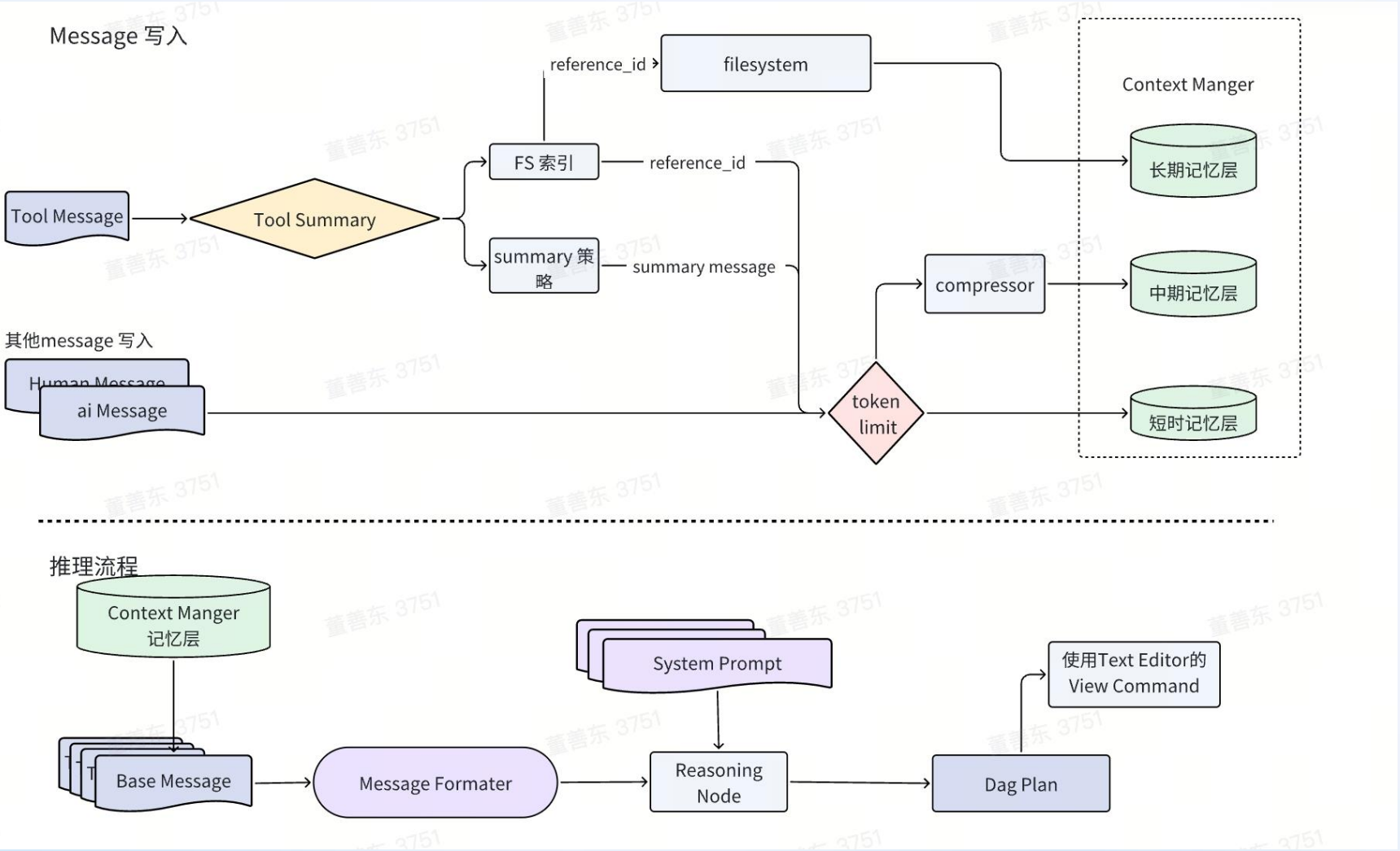
增强推理： 定义thought、可行性分析、DAG生成&校验、反思DAG等多个环节的拆分。 封装 think - tool，根据问题难易自动选择。

DAG约束：如主要步骤、结束条件、输出描述和条件分支等，提高问题解决的效率和准确性。

语法糖避免参数值传递

采取参数语法糖， 避免在多轮tool call中LLM复述观测数据或变量， 提高参数引用准确率，也提高LLM的输出效率

二、ContextEngineering：管理好你的context



1. tool result的总结、保存、检索

2. 超过model limit的context压缩

3. message list格式的重组和改写

4. subagent作为context 的隔离

Context Engineering实践的4个阶段

总结策略

直接截断
结果总结
RAG检索
rule+RAG

tool result的处理

构建文件系统

1. 底层支撑：FileSystem（内存文件操作层）
– 文件读写：
– 目录判断：
– 文本检索：
– 状态持久化：
2. 上层交互：TextEditorTool（长文本操作工具层）

filesystem + text-editor

不同的记忆策略

短期记忆层：存储最近的对话消息，包括用户输入、工具调用结果和系统响应。
中期记忆层：存储经过压缩的对话历史，作为短期记忆的补充。
长期记忆层 ： 长文本保存、跨session检索。

短-中-长memory机制

DAG Context Offloading

重组 message list。 保留DAG中必要的内容， 丢弃不必要的中间环节（thought\plan\dag\todolist）、dag output、ai-message(to user)

重组 message list

三、构建分析Agent，确定工作流程

1

定义Prompt

角色、目标、 工作流程、
输入输出要求、 约束

工作流程示例：
1. 查告警上下文
2. 进行RCA分析
3. 处理建议执行
4. 进一步分析Logid
5. 总结输出

2

定义model、执行架构

model 相关：
- model–name
- temperature等

执行架构：
- Plan & executor

3

定义subagent、tools

agents:
- metrics_agent
- dashboard_agent

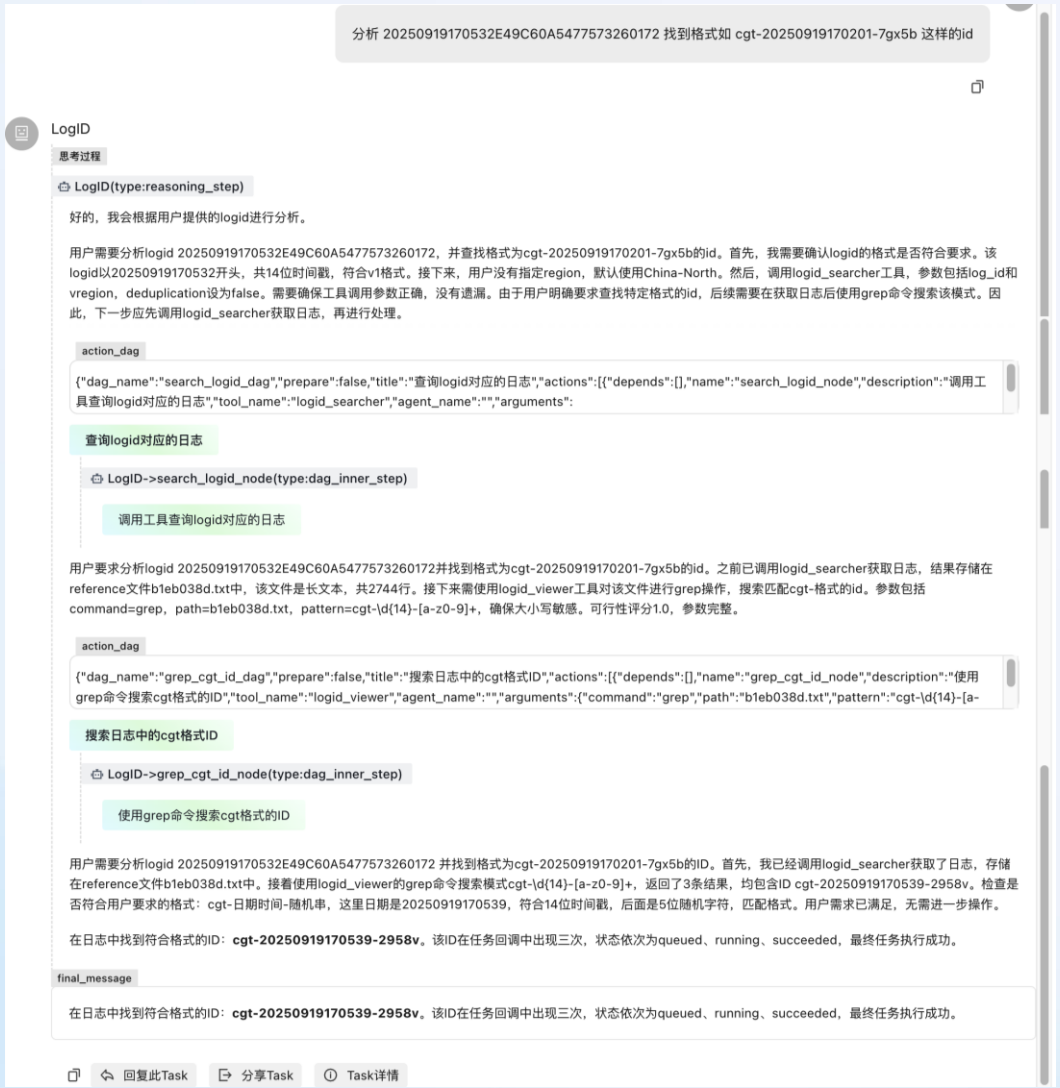
tools:
- alarm.get_alarm_data
- lark.file.fetcher

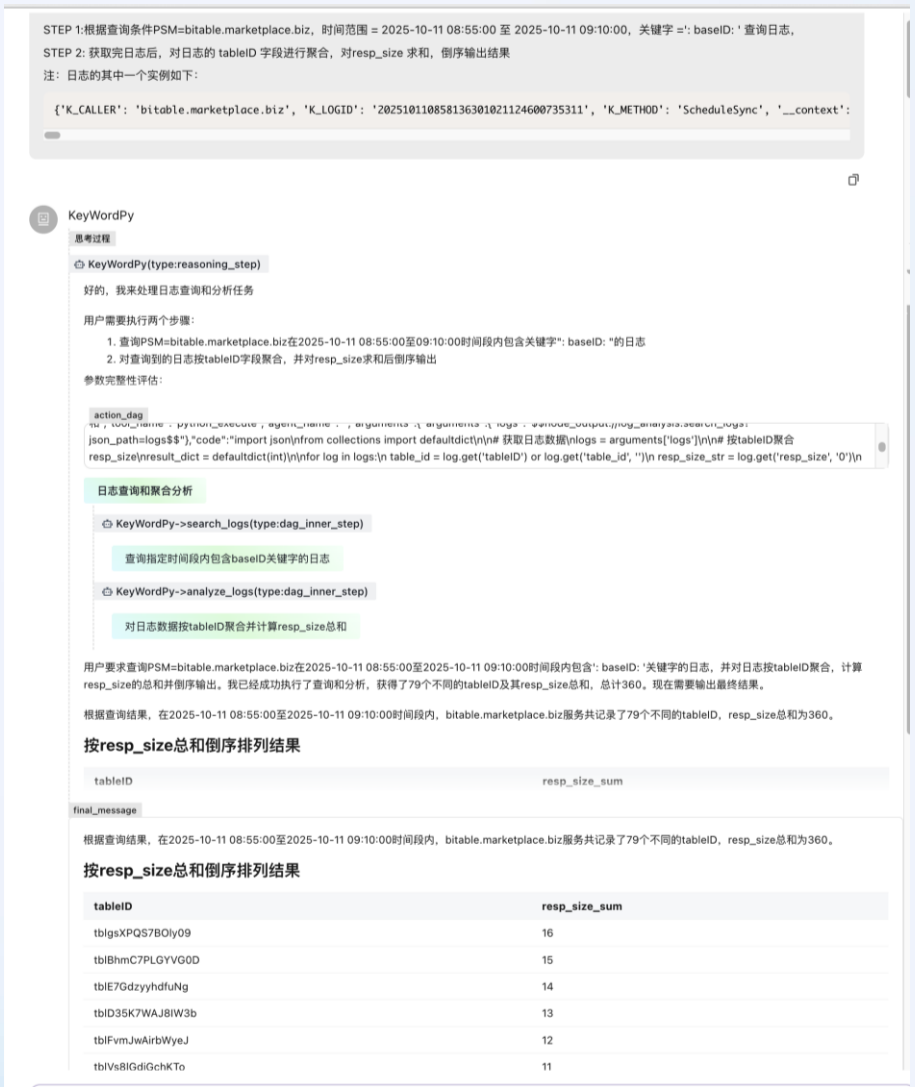
4

定义输入和输出

```
context_schema:
  t– ype: object
  – properties:
    alert_group_id:
      type: string
output_schema: null
```

自定义分析示例


分析 20250919170532E49C60A5477573260172 找到格式如 cgt-20250919170201-7gx5b 这样的id
LogID
思考过程
LogID(type:reasoning_step)
好的，我会根据用户提供的logid进行分析。
用户需要分析logid 20250919170532E49C60A5477573260172，并查找格式为cgt-20250919170201-7gx5b的id。首先，我需要确认logid的格式是否符合要求。该logid以20250919170532开头，共14位时间戳，符合v1格式。接下来，用户没有指定region，默认使用China-North。然后，调用logid_searcher工具，参数包括log_id和vregion，deduplication设为false。需要确保工具调用参数正确，没有遗漏。由于用户明确要求查找特定格式的id，后续需要在获取日志后使用grep命令搜索该模式。因此，下一步应先调用logid_searcher获取日志，再进行处理。
action_dag
{"dag_name":"search_logid_dag","prepare":false,"title":"查询logid对应的日志","actions":[{"depends":[],"name":"search_logid_node","description":"调用工具查询logid对应的日志","tool_name":"logid_searcher","agent_name":"","arguments":
查询logid对应的日志
LogID->search_logid_node(type:dag_inner_step)
调用工具查询logid对应的日志
用户要求分析logid 20250919170532E49C60A5477573260172并找到格式为cgt-20250919170201-7gx5b的id。之前已调用logid_searcher获取日志，结果存储在reference文件b1eb038d.txt中，该文件是长文本，共2744行。接下来需使用logid_viewer工具对该文件进行grep操作，搜索匹配cgt-格式的id。参数包括command=grep，path=b1eb038d.txt，pattern=cgt-\d{14}-[a-z0-9]+，确保大小写敏感。可行性评分1.0，参数完整。
action_dag
{"dag_name":"grep_cgt_id_dag","prepare":false,"title":"搜索日志中的cgt格式ID","actions":[{"depends":[],"name":"grep_cgt_id_node","description":"使用grep命令搜索cgt格式的ID","tool_name":"logid_viewer","agent_name":"","arguments":{"command":"grep","path":"b1eb038d.txt","pattern":"cgt-\d{14}-[a-
搜索日志中的cgt格式ID
LogID->grep_cgt_id_node(type:dag_inner_step)
使用grep命令搜索cgt格式的ID
用户需要分析logid 20250919170532E49C60A5477573260172 并找到格式为cgt-20250919170201-7gx5b的ID。首先，我已经调用logid_searcher获取了日志，存储在reference文件b1eb038d.txt中。接着使用logid_viewer的grep命令搜索模式cgt-\d{14}-[a-z0-9]+，返回了3条结果，均包含ID cgt-20250919170539-2958v。检查是否符合用户要求的格式：cgt-日期时间-随机串，这里日期是20250919170539，符合14位时间戳，后面是5位随机字符，匹配格式。用户需求已满足，无需进一步操作。
在日志中找到符合格式的ID：cgt-20250919170539-2958v。该ID在任务回调中出现三次，状态依次为queued、running、succeeded，最终任务执行成功。
final_message
在日志中找到符合格式的ID：cgt-20250919170539-2958v。该ID在任务回调中出现三次，状态依次为queued、running、succeeded，最终任务执行成功。
回复此Task
分享Task
Task详情


STEP 1:根据查询条件PSM=bitable.marketplace.biz，时间范围 = 2025-10-11 08:55:00 至 2025-10-11 09:10:00，关键字 =': baseID: ' 查询日志，
STEP 2: 获取完日志后，对日志的 tableID 字段进行聚合，对resp_size 求和，倒序输出结果
注：日志的其中一个实例如下：
{'K_CALLER': 'bitable.marketplace.biz', 'K_LOGID': '20251011085813630102112460073531l', 'K_METHOD': 'ScheduleSync', '__context':
KeyWordPy
思考过程
KeyWordPy(type:reasoning_step)
好的，我来处理日志查询和分析任务
用户需要执行两个步骤：
1. 查询PSM=bitable.marketplace.biz在2025-10-11 08:55:00至09:10:00时间段内包含关键字": baseID: "的日志
2. 对查询到的日志按tableID字段聚合，并对resp_size求和后倒序输出
参数完整性评估：
action_dag
json_path=logs$$"},"code":"import json\nfrom collections import defaultdict\n\n# 获取日志数据\nlogs = arguments['logs']\n\n# 按tableID聚合
resp_size\nresult_dict = defaultdict(int)\n\nfor log in logs:\n table_id = log.get('tableID') or log.get('table_id', '')\n resp_size_str = log.get('resp_size', '0')\n
日志查询和聚合分析
KeyWordPy->search_logs(type:dag_inner_step)
查询指定时间段内包含baseID关键字的日志
KeyWordPy->analyze_logs(type:dag_inner_step)
对日志数据按tableID聚合并计算resp_size总和
用户要求查询PSM=bitable.marketplace.biz在2025-10-11 08:55:00至2025-10-11 09:10:00时间段内包含': baseID: '关键字的日志，并对日志按tableID聚合，计算resp_size的总和并倒序输出。我已经成功执行了查询和分析，获得了79个不同的tableID及其resp_size总和，总计360。现在需要输出最终结果。
根据查询结果，在2025-10-11 08:55:00至2025-10-11 09:10:00时间段内，bitable.marketplace.biz服务共记录了79个不同的tableID，resp_size总和为360。
按resp_size总和倒序排列结果
tableID
resp_size_sum
final_message
根据查询结果，在2025-10-11 08:55:00至2025-10-11 09:10:00时间段内，bitable.marketplace.biz服务共记录了79个不同的tableID，resp_size总和为360。
按resp_size总和倒序排列结果
tableID
resp_size_sum
tblgsXPQS7BOly09
16
tblBhmC7PLGYVG0D
15
tblE7GdzyyhdfuNg
14
tblD35K7WAJ8lW3b
13
tblFvmJwAirbWyeJ
12
tblVs8lGdiGchKTo
11

05

总结与展望

经验总结

架构演进的经验

字节 SRE Agent 在从 0 到 1 的搭建过程中，经历了架构从 ReAct 到 CodeAct，最终到 DagAct 的迭代升级，积累了丰富的架构演进经验，为后续的技术发展提供了宝贵的借鉴。

降噪与排障的落地

通过引入基于 LLM 的 “告警值守 SRE Agent”，实现了告警降噪和自定义排障，将 RD 和 SRE 从繁琐的运维工作中解放出来，助力他们抢回 50% 的值班时间，目前在部分业务落地取得了不错的效果。

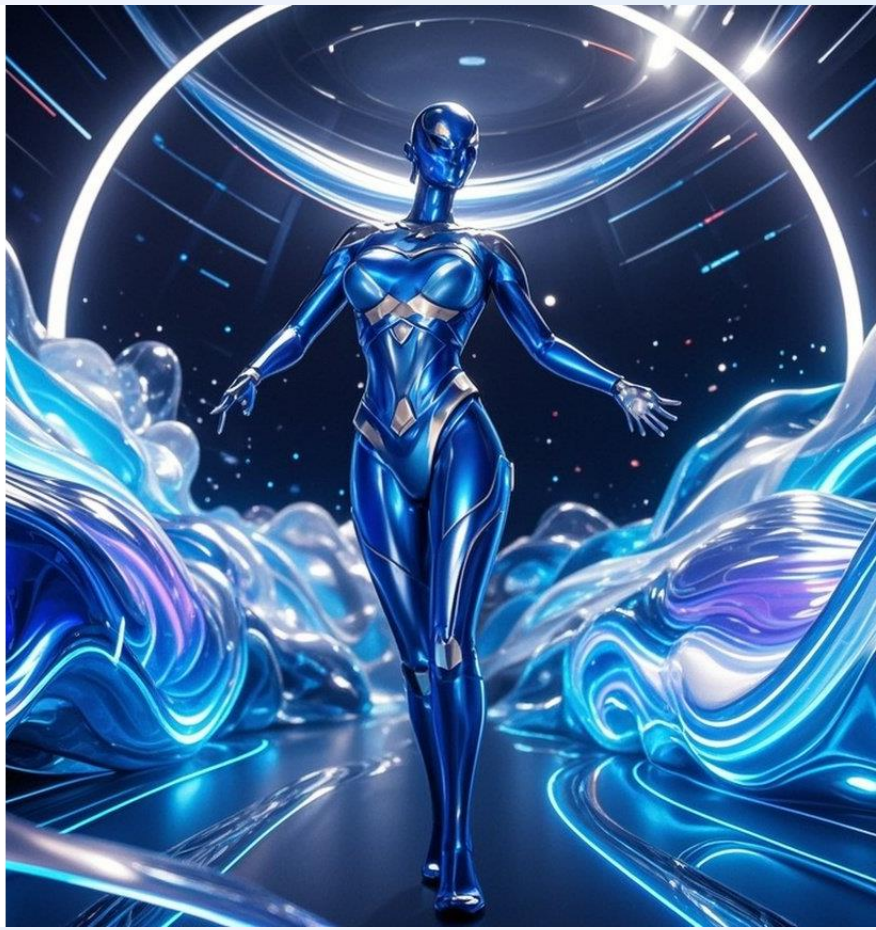
知识沉淀与学习

建立了有效的知识沉淀和学习机制，将历史处理记录、专家经验以及成功案例存储在知识引擎中，实现了 “一次执行，多次复用” 的学习效果，提高了团队的整体运维能力。

未来展望

SRE数字生命体：未来运维新伙伴

以“数字生命”为核心理念的智能运维专家，具备类人化思维与成长能力，将成为团队中可信赖的协作伙伴，重新定义未来运维模式。



类人化核心能力解析

1 **精准理解：**

SRE数字生命体通过场景目标捕捉与上下文分析技术，准确识别运维需求，理解复杂系统状态，实现智能决策支持。

2 **科学规划：**

基于多智能体协同架构，动态制定最优执行策略，实现资源调度、故障处理等运维场景的高效协同运作。

3 **持续成长：**

通过多维学习框架（数据驱动+经验沉淀）与闭环反思机制，持续优化决策模型，实现运维能力的自主进化。

最终价值与未来展望

1 **运维效率革命：从被动响应到主动预防**

SRE数字生命体通过智能监控与预测分析，将故障处理前置，实现运维效率质的飞跃，减少90%以上被动抢修场景。

2 **团队协作升级：人机协同的新工作范式**

数字生命体作为智能助手，7×24小时值守处理常规事务，释放人力聚焦战略创新，形成优势互补的新型协作模式。

3 **持续进化潜力：构建动态成长的运维生态**

基于机器学习框架持续吸收运维数据，使系统能力呈指数级增长，最终形成具备自我优化能力的活体运维体系。

InfoQ 极客传媒

北京

QCon 全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

北京

AiCon 全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

上海

QCon 全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月 — 5月 — 6月 — 8月 — 10月 — 12月

上海

AiCon 全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

深圳

AiCon 全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

北京

AiCon 全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

QCon
全球软件开发大会

InfoQ 极客传媒

谢谢大家

大模型正在重新定义软件

Large Language Model Is Redefining The Software