



Intent-Driven Network Management with Multi-Agent LLMs: The Confucius Framework

SIGCOMM' 25



JOHNS HOPKINS
UNIVERSITY



HARVARD
UNIVERSITY



Stony Brook
University

Zhaodong Wang[★] Samuel Lin[★] Guanqing Yan[★] Soudeh Ghorbani^{★†} Minlan Yu[‡] Jiawei Zhou[§]
Nathan Hu[★] Lopa Baruah[★] Sam Peters[★] Srikanth Kamath[★] Jerry Yang[★] Ying Zhang[★]

[★]Meta [†]Johns Hopkins University [‡]Harvard University [§]Stony Brook University

冯敏远
2025.9.18



- **研究背景与动机**

- 案例分析

- Confucius框架设计



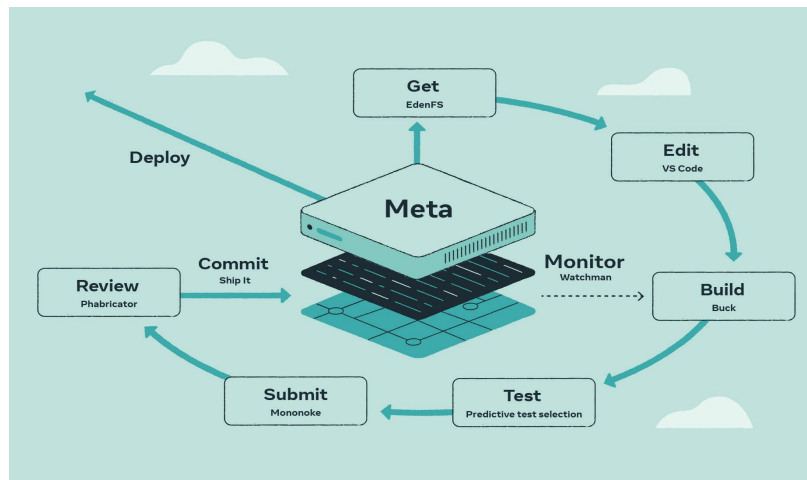
- 实验评估

- 经验与教训

研究背景与动机

传统网络管理方法的局限

- 管理像Meta这样的全球性网络是一项极其复杂的任务，对可靠性、性能和可扩展性有极高要求。
- 尽管有很多自动化工具，但工程师仍需投入大量精力进行手动操作和决策，这既耗时又容易出错。

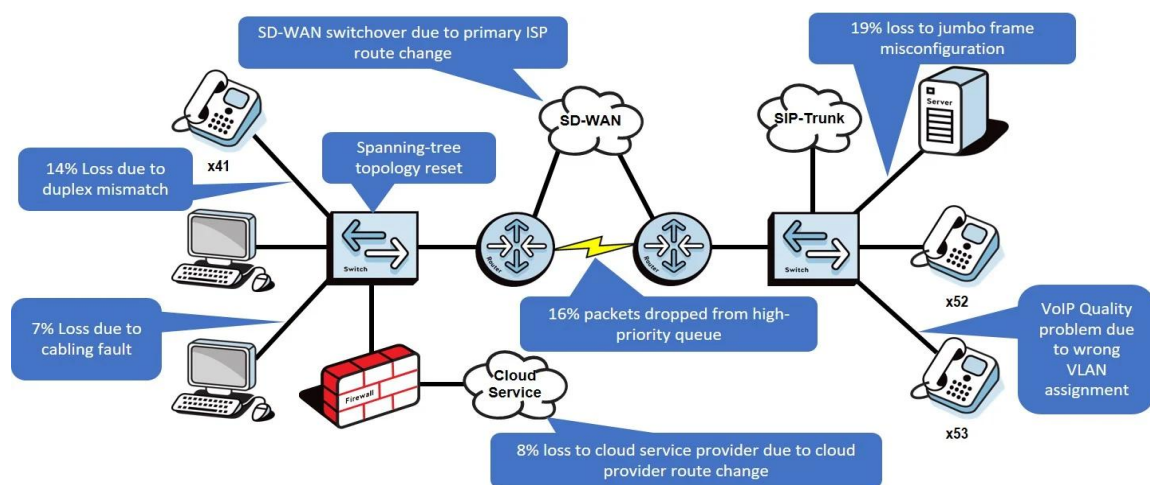
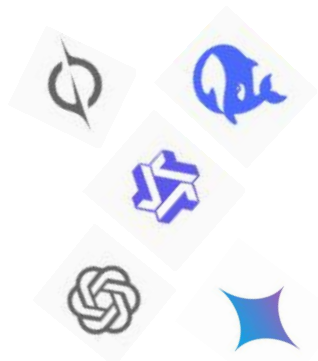


研究背景与动机

LLM的出现和局限

LLM作为一种变革性技术，为提升网络管理水平带来了新的可能性。

核心矛盾: 网络管理任务是复杂、多步骤的，需要调用各种专业工具，而一个简单的、单一的LLM无法胜任这种复杂性。例如，诊断一次服务失败，需要一系列连贯的排查、分析和定位操作。



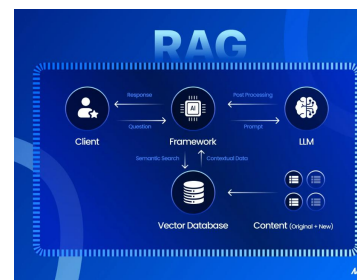
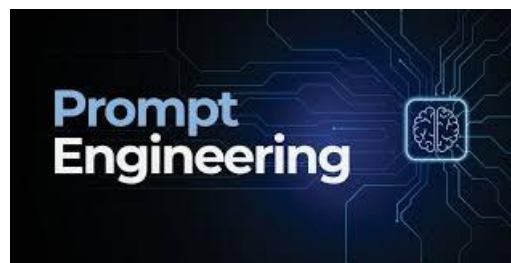
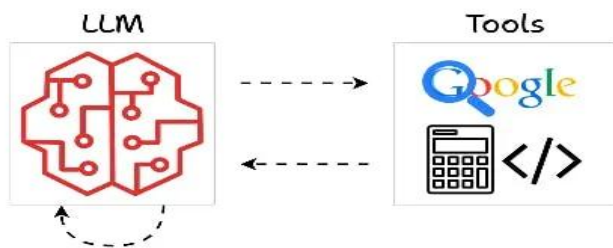
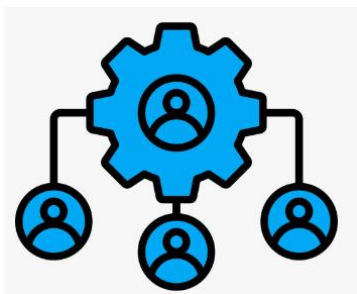
研究背景与动机

Confucius框架——一个新颖的多智能体 LLM 框架

网络管理中的复杂任务需要拆解为多个相互关联的子任务，每个子任务都依赖不同的专用工具和领域知识。若仅依靠单一 LLM 从头到尾完成整个任务，面对足够复杂的场景会导致提示词过长，且性能不稳定。Confucius 采用多智能体框架，将复杂 workflow 拆解为多个子任务，由不同的专用智能体分别处理

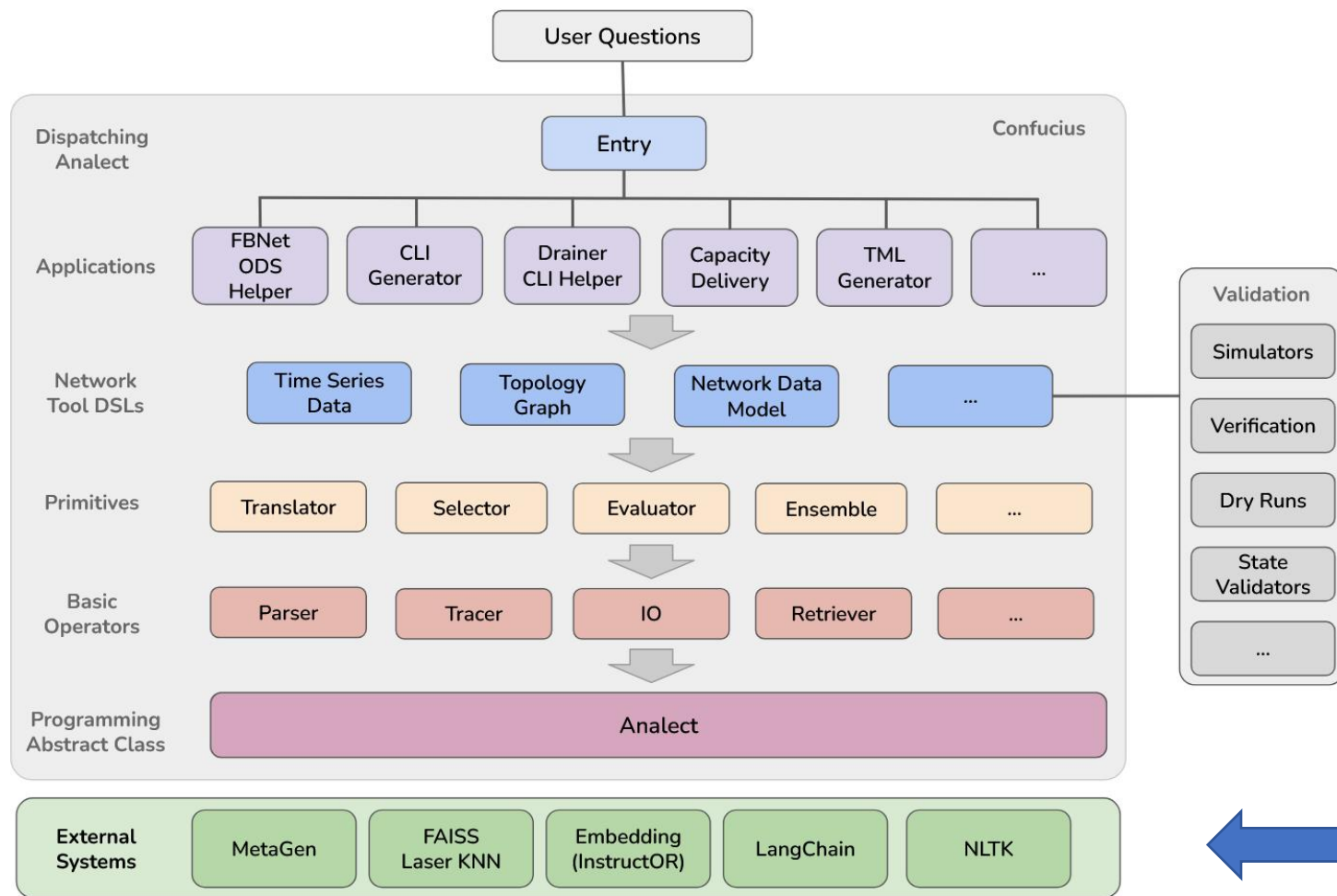
工作流程：

1. 用户提出问题 —— 用户用自然语言描述一个高层级的任务或问题。
2. 意图理解与规划 —— Confucius通过RAG检索与用户意图最匹配的工作流模板，生成一个包含所有执行步骤的计划草案（DAG）并交由用户确认。
3. 收集参数 —— Confucius激活Collector原语，通过对话逐一询问并收集执行DAG中每个节点所必需的输入参数。
4. 代码生成 —— Translator原语会将完整的计划（DAG）和所有收集到的参数“翻译”成最终可被机器执行的DSL代码。
5. 安全校验 —— 生成的代码在执行前会经过语法、功能和逻辑等多重安全校验，若校验失败则会反馈给LLM进行自我修正。
6. 结果呈现 —— 最终，经过安全校验的代码或执行结果会呈现给用户，并在必要时将之前隐去的敏感信息还原



研究背景与动机

Confucius框架——一个新颖的多智能体 LLM 框架



- 分层与可扩展: 采用从底层（外部系统）、中层（原语、DSL）到上层（应用）的分层设计，结构清晰，开发者可以轻松扩展以支持新的用例。

- 安全可靠: 内置验证和安全机制，所有由应用生成的操作在执行前都会经过严格的校验，确保生产环境的安全。

外置工具箱



- 研究背景与动机

- **案例分析**

- Confucius框架设计

- 实验评估

- 经验与教训

1. planning

2. tools

3. memory

4. validation

应用案例分析——网络容量 What-if 分析

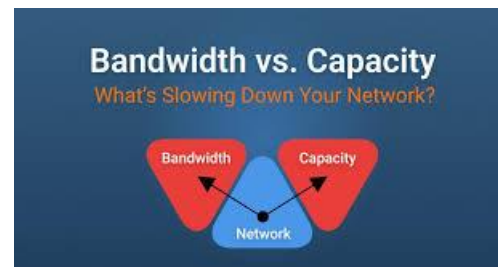
网络容量的“假设分析”

这是网络规划中一个非常常见但又极其繁琐的场景。

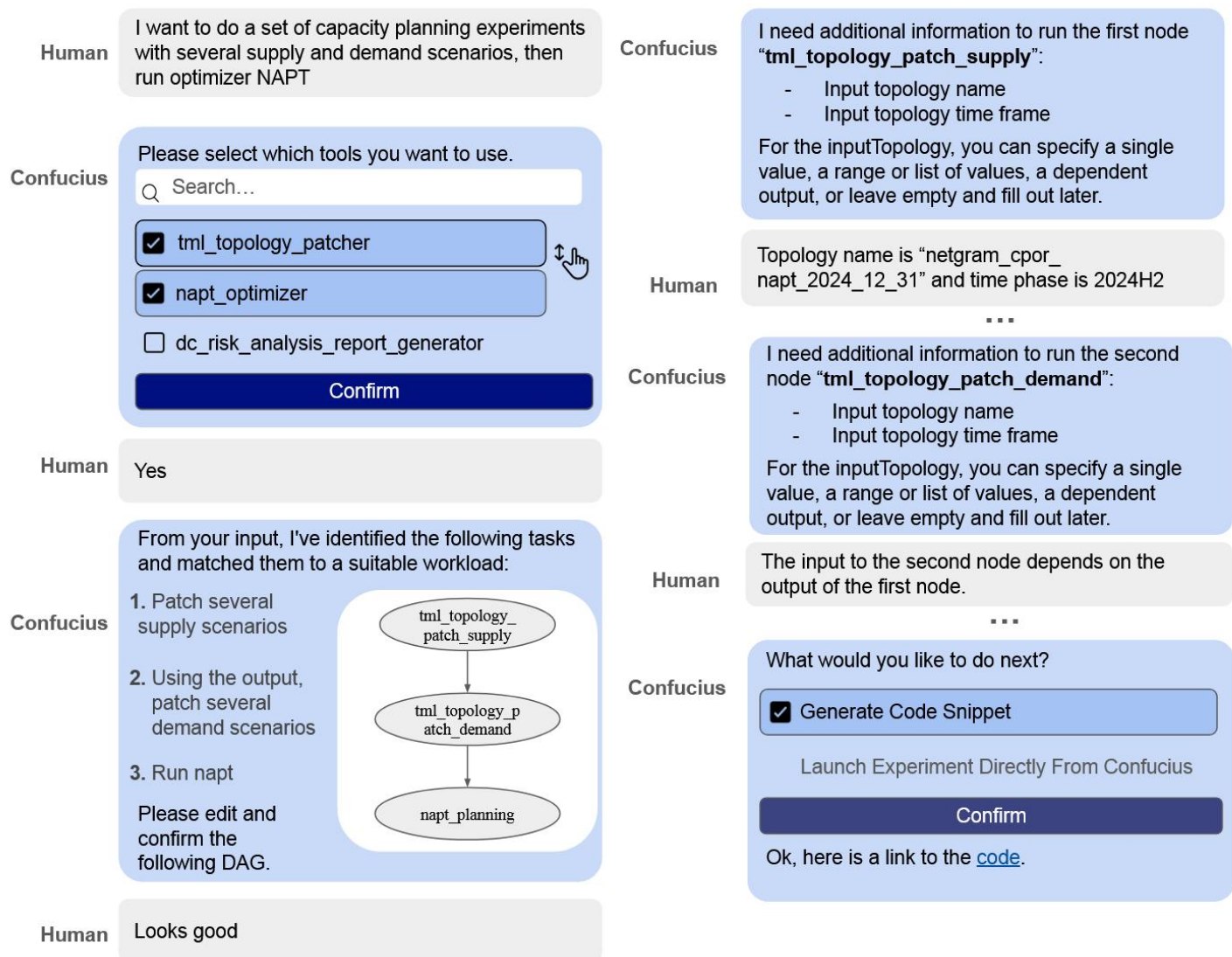
场景描述: 网络规划工程师需要进行复杂的实验来回答“如果.....会怎样”的问题。例如，“为了支持新的AI训练集群，我需要在主干网上增加多少容量？”

传统方法的痛点:

- **流程复杂:** 在Confucius出现之前，工程师需要手动完成一系列步骤：找到合适的工具、理解每个工具复杂的输入输出、然后手动编写脚本将这些工具调用串联起来。
- **重复性高:** 为了测试不同参数，工程师经常需要一遍遍地重复克隆和修改整个工作流，效率低下。
- **工具间互操作性差:** 工程师可能知道如何生成流量预测，但不知道如何将其作为输入喂给下一个网络优化工具。



应用案例分析——网络容量 What-if 分析



为了解决这些痛点，Confucius 提供了一个智能的、对话式的工作流生成和执行环境。我们来看一下具体流程。

用户描述意图

↓
confucius创建DAG

↓
confucius收集参数

↓
confucius生成代码

Confucius 将一个原来需要数小时甚至数天的手动、易错的过程，变成了一个流畅、高效、人机协作的对话过程

应用案例分析——网络故障诊断

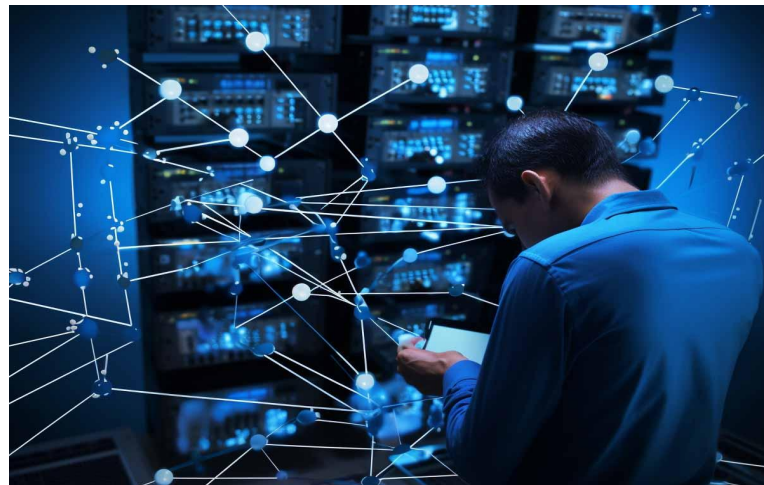
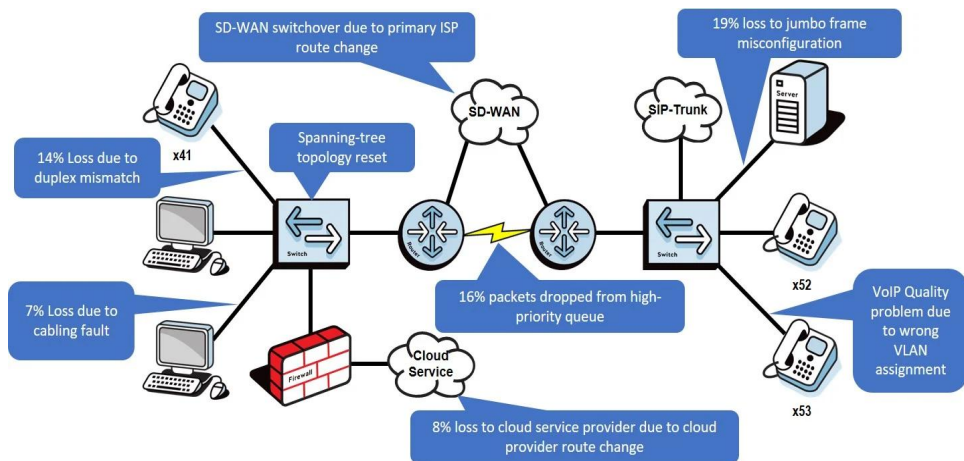
网络故障诊断

即使对资深工程师来说，这也是一项极具挑战性的任务。

•**场景描述:** 线上服务（如Scribe发布/订阅系统）出现了不明原因的延迟，需要快速定位问题根源。

传统方法的痛点:

工程师需要像侦探一样，从海量的监控数据、日志、指标中寻找线索，需要极高的专业经验和直觉。



应用案例分析——网络故障诊断

Human

Scribe read operations are experiencing long tail latency. Investigate whether this issue is caused by a network problem. If a network issue is identified, determine which region pair is affected.

Confucius

Thinking...

I will write a SQL query for the rtt_p99 column and calculate average RTT between DC pairs.

Invoke Tool Use: Scuba Engine

```
SELECT
  (SUBSTR(Sdc, 1, 3)) AS 'src_dc',
  (SUBSTR(Ddc, 1, 3)) AS 'dst_dc',
  AVG('rtt_p99', 'weight') / 1000 AS
  'rtt_p99_avg_ms' FROM 'network_latency'
WHERE
  'time' BETWEEN (NOW() - 86400) AND
  NOW()
  AND (('Ssiterole') IN ('DC'))
  AND (('Dsiterole') IN ('DC'))
GROUP BY 'src_dc', 'dst_dc'
```

Tool: Scuba Engine

	src_dc	dst_dc	rtt_p99_avg_ms
0	DC2	DC2	0.542145
1	DC2	DC3	24.811065
...
1153	DC4	DC4	0.146312

[1154 rows x 3 columns]

Confucius

Thinking...

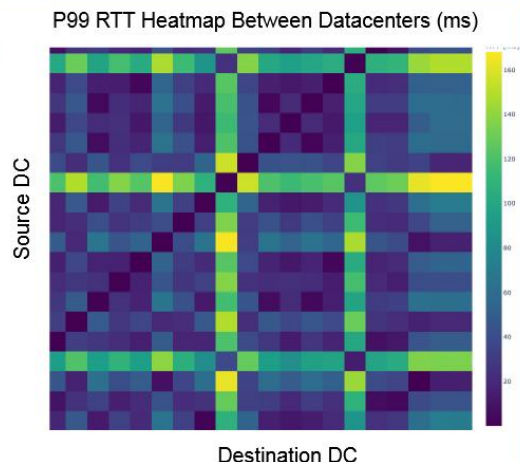
Let's visualize RTT patterns between DCs using a heatmap. I'll write a Python script.

Invoke Tool Use: Python Compiler

```
matrix_df = df.pivot(index='src_dc',
  columns='dst_dc', values='rtt_p99_avg_ms')

fig = go.Figure(data=go.Heatmap(
  z=matrix_df.values,
  x=matrix_df.columns,
  y=matrix_df.index,
  colorbar=dict(title='RTT (ms)'),
))
fig.show()
```

Tool: Python Compiler



Confucius

Thinking...

Let's analyze the key patterns from the heatmap and find the datacenter pairs with highest and lowest RTTs for Scribe reads.

Confucius

Invoke Tool Use: Python Compiler

```
top_5_highest = df.nlargest(5, 'rtt_p99_avg_ms')[['src_dc',
  dst_dc', 'rtt_p99_avg_ms']]
print(top_5_highest.to_string(index=False))
```

Tool: Python Compiler

src_dc	dst_dc	rtt_p99_avg_ms
DC1	DC5	168.057024
DC1	DC6	167.457915
DC1	DC7	166.755838
DC7	DC1	166.654504
DC1	DC8	162.874577

Confucius

We observe these highest latency pairs: (1) DC1 ↔ DC5: ~168ms, (2) DC1 ↔ DC6: ~167ms, (3) DC1 ↔ DC7: ~167ms. DC1 appears in most of the highest latency connections. Most high-latency pairs show RTTs ~165ms.

在这个案例中，Confucius不再仅仅是执行一个预设的流程，而是扮演了一位能够自主思考和行动的AI诊断专家

接收用户任务

思考

行动

向用户报告结果

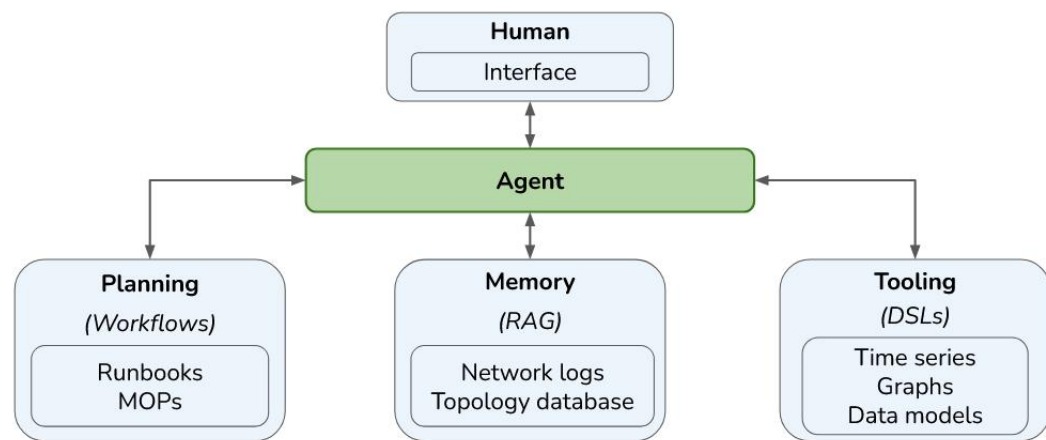
Confucius能够自主规划诊断步骤，并且无缝地在自然语言、SQL、Python代码、图表分析之间切换



- 研究背景与动机
- 案例分析
- **Confucius框架设计**
 - 1. planning
 - 2. tools
 - 3. memory
 - 4. validation
- 实验评估
- 经验与教训

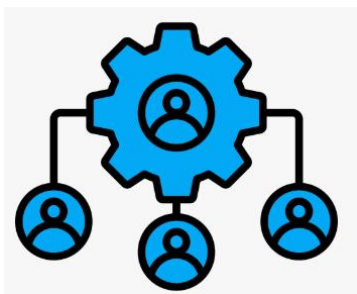
Confucius框架设计

Confucius框架——一个新颖的多智能体 LLM 框架

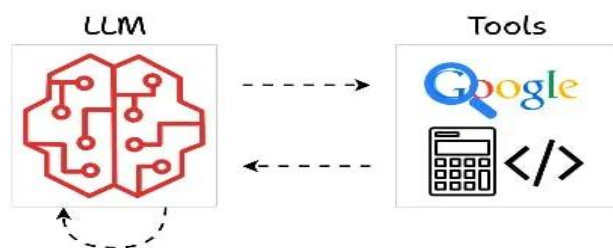


- 规划 (Planning): 利用现有的工作流(Workflows)和操作手册(MOPs)将任务分解成DAG。在关键节点引入人工确认, 并结构化地收集用户输入, 确保安全可靠。
- 记忆 (Memory): 使用RAG技术作为长期记忆, 从网络日志、拓扑数据库等获取知识。
- 工具 (Tooling): 通过将自然语言翻译成领域特定语言(DSLs), 与现有工具无缝集成。
- 验证 (Validation): 在指令执行前, 通过内置解析器、API试运行 (Dry Runs) 和外部校验工具进行多重安全检查, 确保操作的准确与安全。

Multi Agents



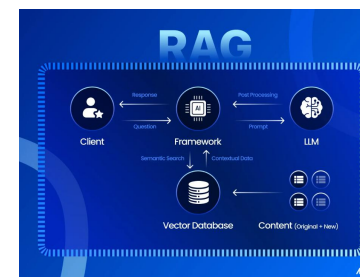
Tool Calling



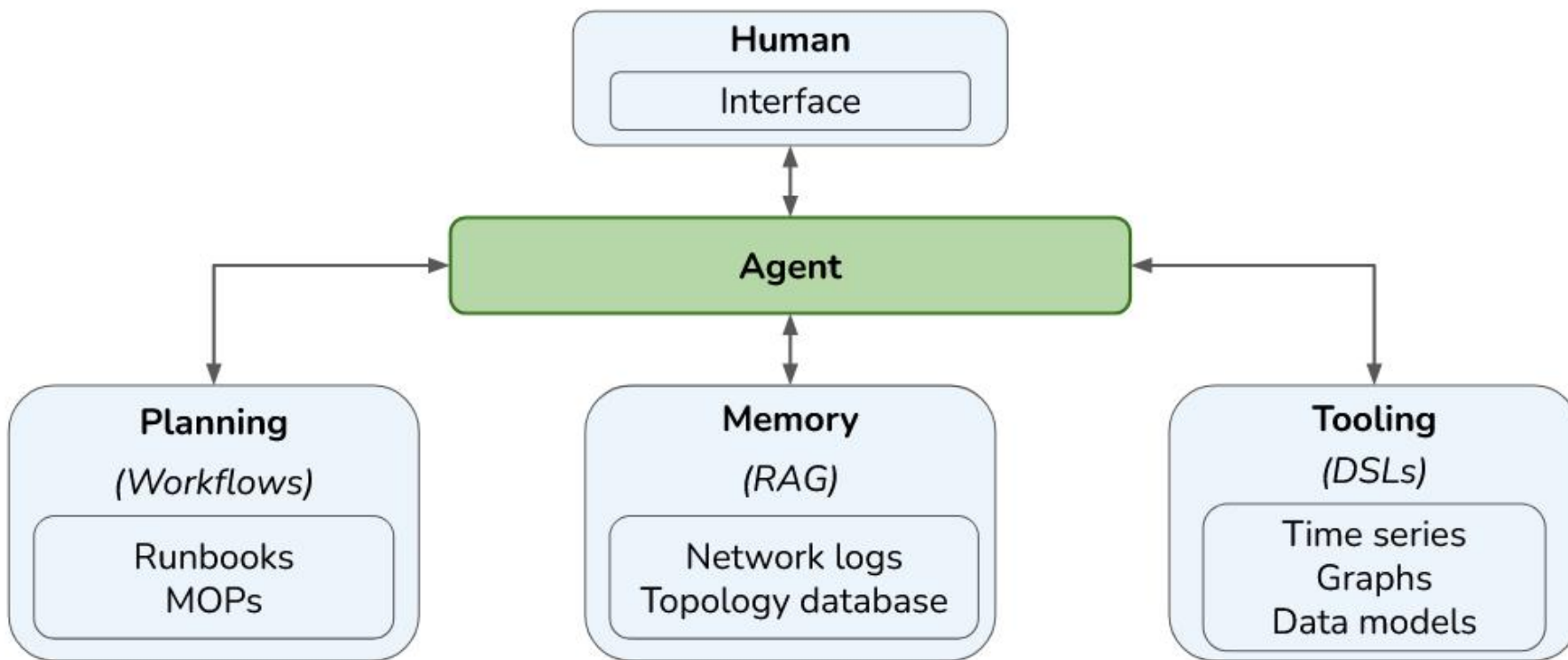
prompt Engineering



RAG



Confucius框架设计



四个模块: 规划 (planning) 工具 (tools) 记忆 (memory) 验证 (validation)



- 研究背景与动机

- 案例分析

- **Confucius框架设计**



- 实验评估

- 经验与教训

Confucius框架设计——planning

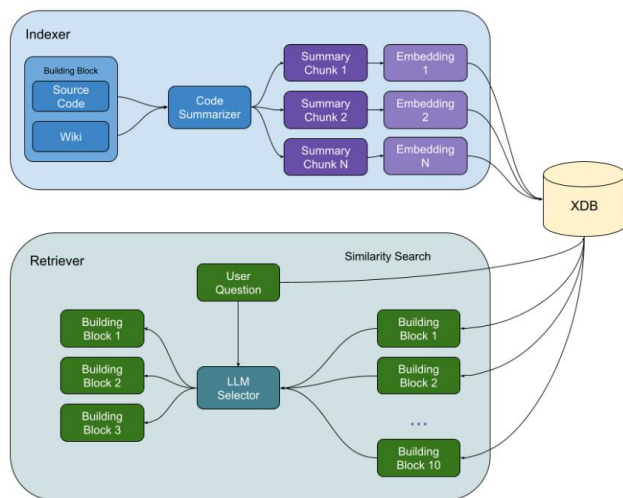
RAG检索

- Building Blocks:

像Meta这样的大公司，在长期的网络运维中，已经沉淀了大量的标准化操作流程（MOPs），这些是用自然语言撰写的逐步操作流程。为了便于自动化，这些MOPs已经被拆解成了更小、更模块化的构建模块（Building Blocks, BBs）。每一个BB都是一个可执行的脚本或配置片段，负责完成一个具体的子任务。

- 索引与检索

在生产环境中，我们面临工作流和 BB 的发现問題。由于存在数百个工作流和上千个 BB，工程师很难快速找到适合特定任务的组件。为此，我们开发了一种基于 RAG（检索增强生成）的方法来高效搜索工作流及 BB。



其体系结构包括两个核心组件：索引器（Indexer）和检索器（Retriever）。

- 索引器利用 LLM 分析每个工作流和 BB，提取关键信息并计算语义嵌入，存储到数据库中。
- 检索器分两步运行：首先进行粗粒度相似度搜索，筛选约 10 个候选工作流/BB；然后再进行细粒度选择，将结果缩小到不足 3 个。

Confucius框架设计——planning

DAG表示

用户输入问题后，通过RAG检索，系统找到了最可能符合用户意图的那个现有工作流，系统将这个被选中的工作流以DAG的形式呈现给用户。

在Confucius中，每一个工作流都被表示为一个DAG。

- 节点：DAG中的每一个节点都对应着一个BB（即一个子任务）。
- 边：节点之间的连接代表了数据的流向和任务的依赖关系最终生成过程

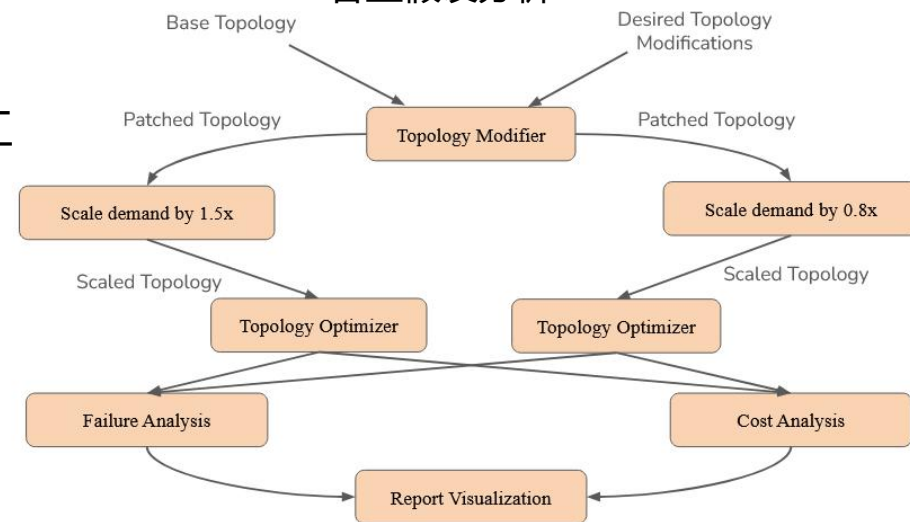
规划原语——规划时llm的执行方式

1. Ensemble：并行调用多个 LLM 执行同一任务，并将结果组合为单一输出。启用 Ensemble 时，用户提供一组 LLM 参数传递给支持的 API。系统支持四种组合模式：

- First-mode：选择最快代理的结果；
- Merge-mode：合并所有代理结果供人工审核；
- Filter-with-Validation：利用验证框架过滤错误结果；
- Return-all：返回所有答案集合。

2. Orchestrator：一种新型 AI 框架，使 LLM 能够通过逐步调用构建块来自主创建工作流。这种方法需要在提示中提供详细的示例和说明，指导 LLM 何时、如何使用哪些构建块。

容量假设分析



```
ensemble_out = await self.invoke_analect(
    AnalectEnsemble(),
    AnalectEnsemble.input(
        analect=analect_to_call,
        inputs=inputs,
        llm_params=[
            LLMParams(model="llama3-70b-instruct"),
            LLMParams(model="llama"),
        ],
        output_mode=AnalectEnsembleOutputMode.FIRST,
    )
)
```



- 研究背景与动机

- 案例分析

- **Confucius框架设计**



- 实验评估

- 经验与教训

Confucius框架设计——tools

• 工具原语——执行时llm的身份

网络工程师依赖复杂工具，而这些工具基于结构化数据运行。要想与现有的网络管理工具集成，我们必须将用户的自然语言翻译为这些工具可处理的结构化数据（DSLs）。为了支持翻译，Confucius 引入了三类原语：

1. Translator (核心翻译官):

职责: 这是团队的核心，负责将自然语言直接翻译成结构化的输出，如命令行、配置文件或DSL代码。

2. Selector (选项筛选专家):

职责: 当用户的指令存在多种可能性时（例如，“我想查一个模型”，但数据库里有10个相似的模型），Selector会介入，通过与用户交互或相似度搜索，帮助筛选出最正确的那一个选项。

3. Collector (信息追问专家):

职责: 当用户的指令信息不全或意图模糊时（例如，“查一下交换机的CPU”），Collector会主动提出追问（“请问是哪台交换机？时间范围是多久？”），以收集完整、精确的信息，避免因误解导致错误执行。

Confucius框架设计——tools

• 基础网络管理的三种常用 DSL (Domain-Specific Language)

虽然 Translator 可将自然语言转换为结构化数据，但仍需确定目标格式。现有网络工具依赖多种结构化数据格式。基于生产经验，我们识别出三类最常用的 DSL。

1. 网络图语言 (TML): 用于描述对网络拓扑的修改。
2. 时间序列数据语言 (ODS Query): 用于查询网络监控指标，如流量、CPU利用率等。
3. 网络数据模型语言 (Robotron): 用于操作和定义网络设备的精确配置。

"I want to add a new hose L3 node."



```
tm.add(  
  what="l3_nodes",  
  add=""  
    name="<name>",  
    l1_node_name="<l1_node_name>",  
    is_hose_entity=True,  
    lat=<lat>,  
    lon=<lon>,  
    node_role=NodeRole.DC,  
    ...  
)
```

TML

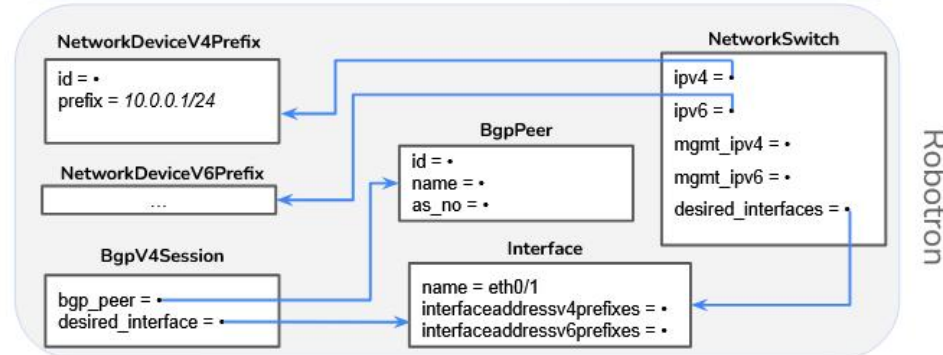
"Calculate the average CPU utilization for asw01.mia1 over the last 10 minutes, then add 5, and finally take the P95."



```
{  
  "key": {  
    "values": ["FBNet:device.cpu_util"]  
  },  
  "query": {  
    "time_range": "start = -600, end = 0",  
    "reduction": ["percentile(95)"],  
    "transformation": ["avg(10m)", "formula(+ $1 5)"],  
    ...  
  },  
  "entity": {  
    "values": ["fbnet_device(DEVICE_NAME=asw01.mia1)"]  
  }  
}
```

ODS

"Create a BGP session with AT&T using peer IP 10.0.0.1, reachable via interface eth0/1 on Switch 1."



Robotron

Confucius框架设计——tools

• Prompt Engeering

提升翻译准确性的有效方法之一是设计合适的提示工程策略。我们针对不同 DSL 任务定制最优组合：

Zero-shot chain of thought: 先思考后翻译。例如在 ODS 翻译时，要求 LLM 使用正则匹配解析字符串（如 “rsw1aa.*prn1” ），先理解网络上下文，再生成翻译。

Few-shot chain of thought: 提供示例帮助模型学习，如在 ODS 示例中教会模型理解 “entity” 字段。

Contrastive chain of thought: 提供错误示例并标注错误，例如在 ODS 翻译中构造错误示例 “在 prn2 区域包含交换机” 。

工具调用 (Tool calling) : 提供 API/CLI/库/数据库的访问能力。在 ODS 翻译中，直接调用 ODS Query API，并将返回消息加入后续提示，让 LLM 自我修正。

推理并执行 (Reason and act) : 使用 Orchestrator 让 LLM 自行规划复杂任务。例如在网络调查中，ODS 查询被拆分为 5 个子任务。

代码即推理 (Code as reasoning) : 让 LLM 生成代码而非直接修改拓扑结构。例如生成 TML 代码，由编译器和引擎验证与执行。这种方式不仅便于验证，还支持人工修改或合并代码片段。

• 内置验证

网络应用对安全性要求极高，因Confucius 内置多种验证机制，以减少人工检查负担，并能将错误消息自动反馈给 LLM 进行自我纠正。

- 语法检查: 使用内置解析器 (Parser) 检查生成的代码是否存在语法错误。
- 功能预演: 调用工具自带的API “试运行” (Dry Run)模式，在不实际执行的情况下，检查操作是否会引发功能性错误。
- 逻辑校验: 使用外部验证工具（如拓扑完整性检查器）来确保操作符合网络设计的逻辑规范。



- 研究背景与动机
- 案例分析
- **Confucius框架设计**
 - 1. planning
 - 2. tools
 - 3. memory**
 - 4. validation
- 实验评估
- 经验与教训

Confucius框架设计——memory

- **短期记忆——维持单个用户会话内的对话连续性。**

如果用户先询问“区域 A 的总出口流量”，然后再问“区域 B”，那么第一个问题的上下文应该能够高效复用到第二个查询中。

- 开发了一个专用的内存管理组件，它使用消息树作为内存抽象。在一个多步骤的复杂任务中，每个子任务都能访问其父任务乃至整个会话的记忆，从而全面理解对话的来龙去脉。

- **长期记忆——跨会话共享的外部领域知识，用于同一使用场景。**

解决的问题：让LLM能够利用其训练数据中不包含的、Meta内部的海量、专业且在不断更新的领域知识。

例如，让它知道公司内部数十万个网络数据模型的细节。

- Naive RAG (朴素RAG)：基础的相似度搜索，适用于小型数据集。
- Hybrid RAG (混合RAG)：采用“粗筛+精筛”的两阶段过滤方法。先通过相似度搜索找出大量候选项，再根据时效性、使用频率等规则或另一个LLM进行二次筛选，得到更精准的结果。
- Query Transformations (查询转换)：在检索之前，先用一个LLM来优化和重写用户的原始问题，提取关键术语，让后续的检索目标更明确，效果更好。

• 隐私保护

在赋予LLM强大记忆能力的同时，必须解决一个至关重要的问题——数据隐私。Confucius为此设计了一套严格的‘先脱敏、再处理、后还原’的安全机制。”

解决的问题：防止用户在对话中无意输入的个人身份信息（PII）或敏感数据（如IP地址、设备标识、姓名、邮箱等）被发送给第三方LLM模型。

实现方式：

1. 自动脱敏：在用户的请求被发送给LLM之前，一个名为“Identifier Redaction Service”的隐私处理服务会自动识别40种类型的敏感信息，并用虚拟占位符替换它们。
2. 安全处理：LLM处理的是已经脱敏过的数据。
3. 结果还原：在收到LLM的响应后，系统会再将占位符还原成原始的敏感数据，最终呈现给用户。



- 研究背景与动机
- 案例分析
- **Confucius框架设计**
 - 1. planning
 - 2. tools
 - 3. memory
 - 4. validation**
- 实验评估
- 经验与教训



• 基准测试框架

优化一个 Confucius 应用可能是复杂的任务，用户往往希望比较不同模型和提示技术在特定数据集上的性能表现。为此，我们开发了一个基准测试框架。

工作流程

评估流水线由三个关键组件构成：

- 用户提供的数据集
- Confucius 应用
- 一组评估标准

评估器首先读取输入-输出对的数据集，并在这些输入上调用应用程序。然后收集实际输出，并基于预定义的标准与期望结果进行对比评分。

三种内置评估标准：

- Exact Match（精确匹配）：适用于确定性和数值输出，例如验证集群中的设备数量；
- Regex Match（正则匹配）：适用于集合或字符串输出，用于验证是否包含特定关键词；
- LLM-as-a-Judge：适用于复杂输出，由 LLM 判断语义上的接近程度。



- 研究背景与动机
- 案例分析
- Confucius框架设计
 - 1. planning
 - 2. tools
 - 3. memory
 - 4. validation
- **实验评估**
- 经验与教训

实验设置

• 数据集：

1. TML：用于测试将自然语言翻译成拓扑修改语言
2. ODS transformations/ODS reductions：用于测试对时间序列数据（存储在Operational Data Store，即运营数据存储中）的查询和处理能力
3. Robotron：用于测试将自然语言翻译成Robotron数据模型的能力。一种网络数据模型的 DSL。
4. Netgram：由“描述网络任务的自然语言请求”及其对应的“代码块名称和URL链接”组成。这是 Meta 内部的 网络诊断 DSL，用于访问和组合不同的网络功能块
5. Wiki Q&A：维基百科风格的知识问答数据集，用于评估在大规模知识库上的检索和问答能力。

• 指标：

1. Netgram：采用 regex match，若模型输出同时包含真实代码块名称和正确的源码 URL，则得分 = 1；若仅正确识别一项，则得分 = 0.5。
2. 其他数据集：采用 LLM-as-a-Judge，即由 LLM 给出 0 ~ 1 的整体质量评分，以衡量生成的 DSL 或答案的正确性、完整性和格式规范性。



实验设置

• 模型:

1. 通用基础模型:

- a. Llama 3.1、Llama 3.3
- b. Claude 3.5 Sonnet
- c. Gemini 1.5 Pro、Gemini 2.0 Flash
- d. GPT-4o

2. 内部微调模型: 基于领域特定数据 (代码数据、维基文档、内部 Q&A) 进行训练, 用于对比评估。

• Baseline:

1. 单模型生成: 使用 Llama 3.1 直接输出结果, 不做集成。

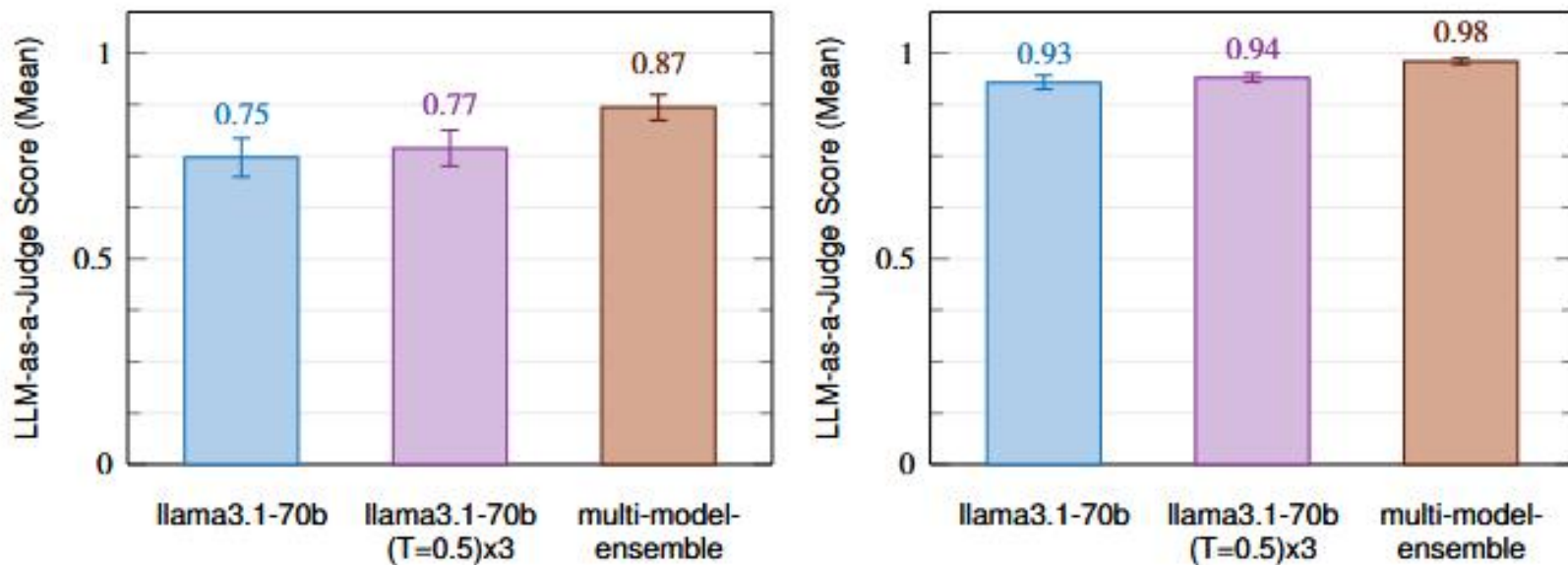
2. 微调模型: 一个在特定领域数据上微调的内部模型, 代表 “传统微调范式”。

3. 对比思路:

a. 用 Llama 3.1 单模型作为 baseline, 与 同质/异质集成 做对比。

b. 用 领域微调模型作为 baseline, 与 Confucius + prompt engineering 或 Confucius + RAG 做对比。

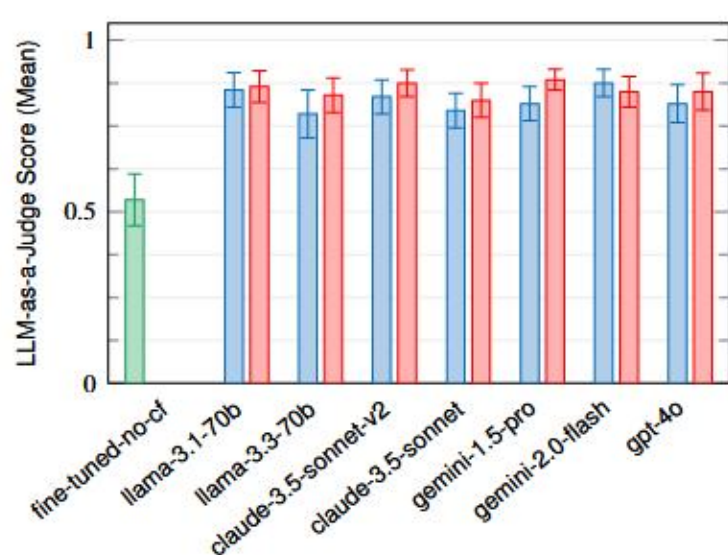
多智能体推理中的集成



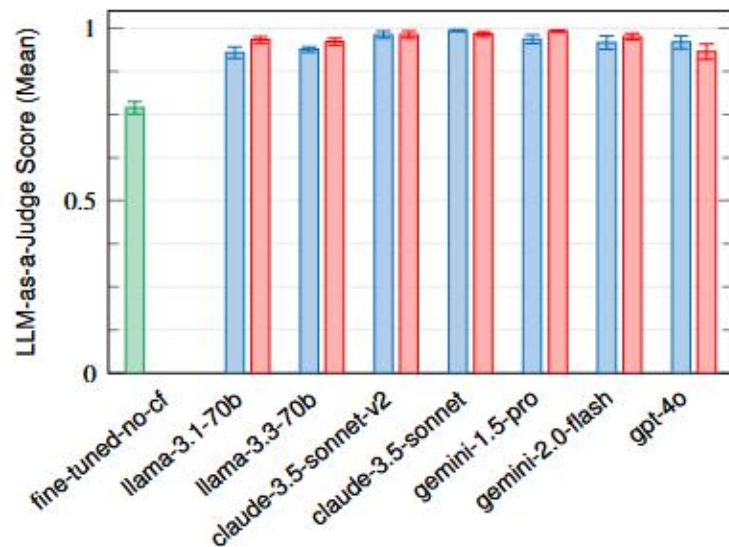
我们评估了其在 ODS transformations/ODS reductions 上的影响，实验设置包括：

- 1.单模型生成 (Llama 3.1) ；
- 2.同质集成：由三个 Llama 3.1 模型 (温度=0.5) 组合；
- 3.异质集成：组合 Llama 3.1、Claude 3.5 Sonnet 和 Gemini 2.0 Flash。

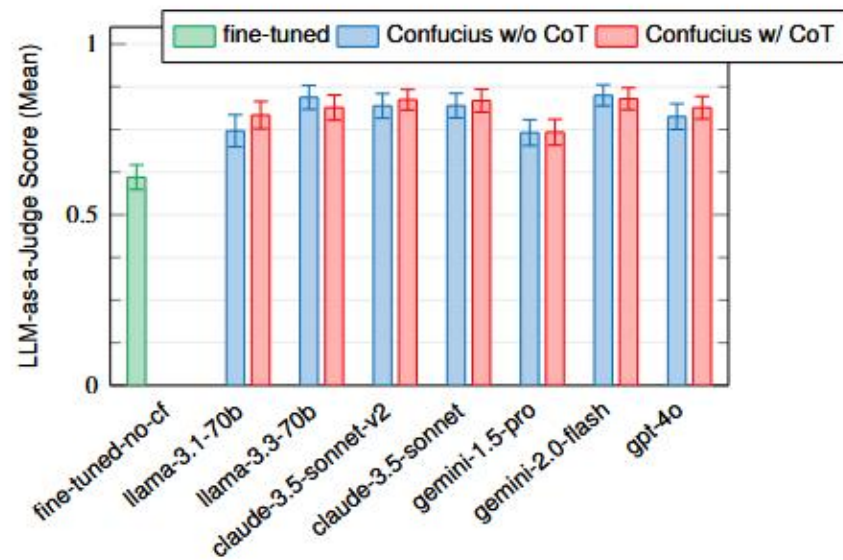
DSL 翻译——将用户的自然语言指令翻译成机器能够执行的领域特定语言



(a) TML ($N = 20$)



(b) ODS Transformation ($N = 91$)



(c) ODS Reduction ($N = 48$)

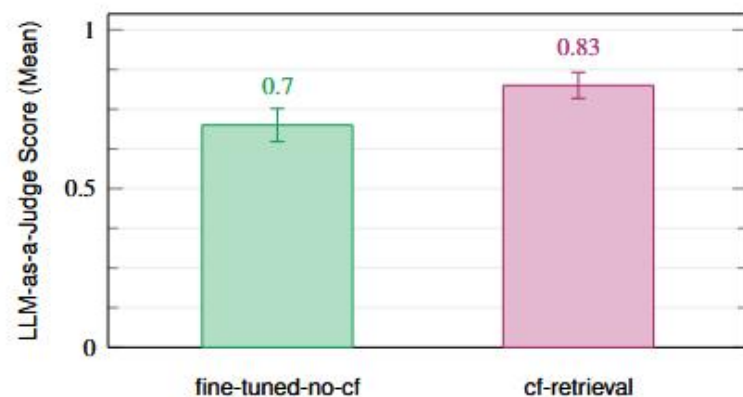
fine-tuned-no-cf: 用Meta内部大量的网络数据进行过训练的、针对网络任务的专用模型，不使用Confucius框架

绿色 (fine-tuned): 基线模型，即在内部数据上专门微调过的模型。

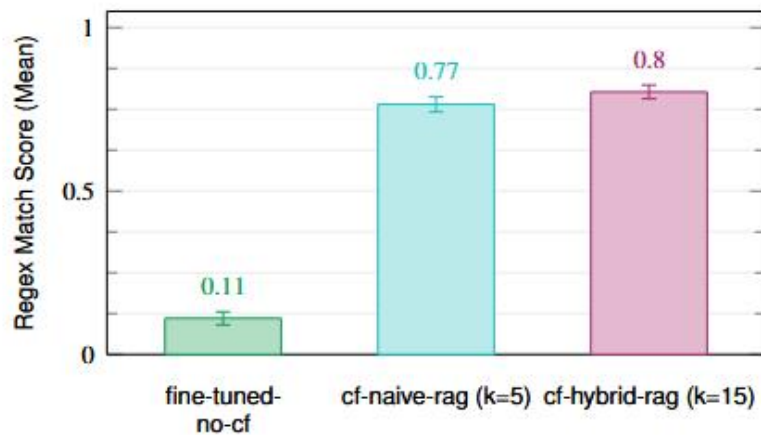
蓝色 (Confucius w/o CoT): 使用Confucius框架，但不加“思维链” (Chain-of-Thought) 提示。

红色 (Confucius w/ CoT): 使用Confucius框架，并且加入了“思维链”提示。

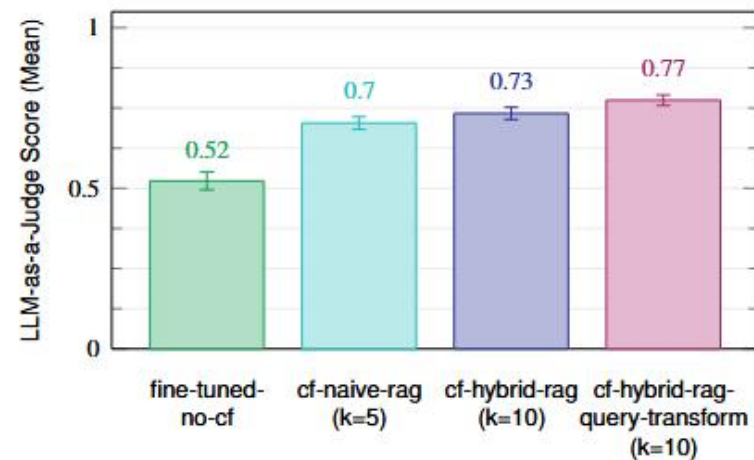
RAG 在知识检索中的应用



(a) Robotron ($N = 12$)



(b) Netgram ($N = 226$)



(c) Wiki Q&A ($N = 250$)

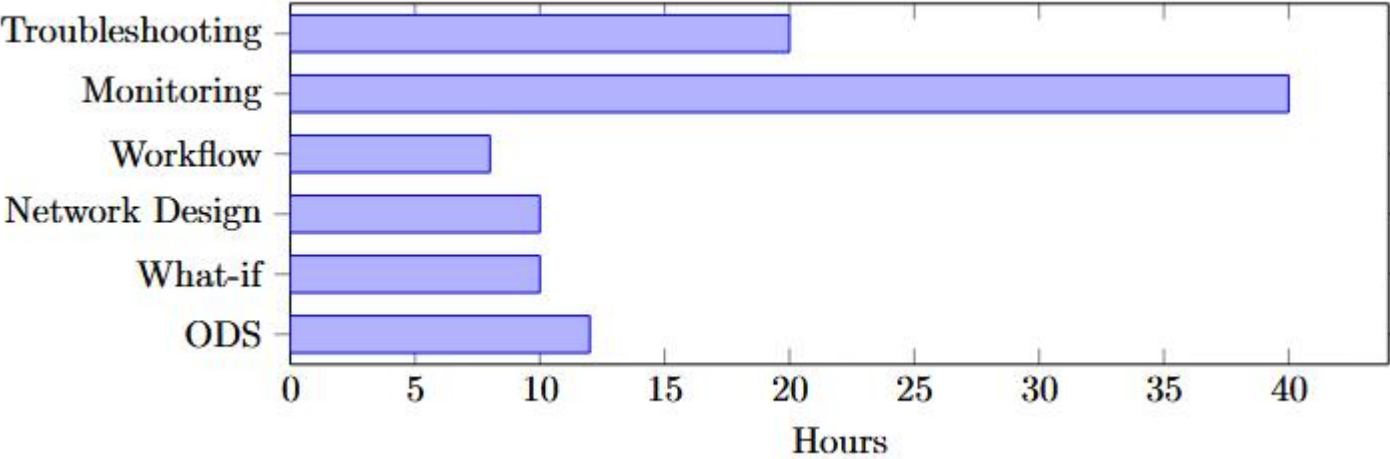
Robotron: Confucius 通过上下文学习从 400 多个目标模型中检索

Netgram: Confucius 从一个包含 11.86 万向量、约 1.24 GB 的嵌入库中检索

Wiki 问答: Confucius 从一个包含 330 万向量、约 33.5 GB 的嵌入库中检索。

采用与使用统计

Metric		Metric	Performance Diagnosis	What-if Planning
Total users	4.16K	Total users	121	605
Monthly active users	2.63K	Total sessions	1.86K	2.66K
Total sessions	241.38K	Total time spent (hr)	835.58K	207.33K
Total messages	31.62M	Total messages	4.17M	2.36M
Use cases onboarded	64	AI to human msg ratio	2.46	7.65
AI to human msg ratio	20.54			



去一年中，Confucius 实现了显著的增长与普及：共有 4160 名独立用户，24.138 万次会话，3162 万条消息。

AI 生成消息与人工消息的比例很高：所有用例平均 **20.54 倍**，性能诊断 **2.46 倍**，容量规划 **7.65 倍**。这表明大多数交互几乎无需人工干预，而性能诊断中的较低比例则反映了故障排查任务的复杂性

Confucius在各应用每周节省的工程师工时



- 研究背景与动机
- 案例分析
- Confucius框架设计
 - 1. planning
 - 2. tools
 - 3. memory
 - 4. validation
- 实验评估
- 经验与教训



• What Worked Well

1. RAG是解决网络问题的强大范式

许多网络任务的模式都是“根据现有项进行修改”，比如参考一个旧配置生成新配置。RAG的“检索-生成”模式完美契合了这类需求。

2. 与现有工具集成是推广的关键

通过与现有自动化系统集成，LLM 能生成结构化数据并嵌入日常流程，从而实现高频稳定应用。

Confucius成功的关键在于它无缝地融入了现有的自动化系统和工具，成为了工程师工作流的一部分，从而获得了广泛采纳



• 教训与挑战

1. 复杂的故障排查对LLM仍是难题:

- LLM擅长遵循固定的流程来解决已知问题。
- 但对于那些没有固定模式、需要深厚经验和直觉才能解决的疑难杂症，LLM仍难以替代顶尖的人类专家。

2. LLM的固有缺陷依然存在:

- 上下文丢失: 在长对话的后期，模型可能会“忘记”早期的重要约束（比如特定的时间范围）。
- 模型幻觉: LLM倾向于在找不到答案时“编造”一个，而不是承认“我不知道”。这在要求确定性的工程系统中是极其危险的。
- 隐私风险: LLM在处理不同密级的数据时，如果因权限问题出错，其报错信息并不明确，存在潜在风险。



- 基于此：在此之上，顺势思维

- 从“上下文记忆”到“上下文精炼”：论文提到了在长对话中存在上下文丢失的风险。虽然短期记忆的“消息树”结构对此有所缓解，但能否引入一个“上下文管理智能体”？该智能体专门负责在对话进行中，周期性地对长篇的上下文进行“精炼”，自动提取出最重要的约束条件和中间结论，从而保持上下文窗口的简洁和高效，让核心智能体始终能聚焦于最关键的信息。

- 与此类似：与此平行，发散思维

在我们的面向移动端资源的智能原生调度场景下，

- rag效果也许也大于微调。
- 采用异构llm ensemble的效果 > 同构多llm ensemble > 单一llm,采用异构llm ensemble进行尝试
- 本文的框架与我们的场景类似，是否可以沿用

- 由此需要：在此之下，逆向思维

- Confucius的架构本质上是一个将专家知识（以 workflow、工具、文档的形式存在）与LLM的通用推理能力相结合的平台。它可以被泛化用于任何符合“知识密集型、流程标准化、工具专业化”特点的领域，可以泛化到其他专业场景。



Q&A

2025年9月18日

冯敏远