

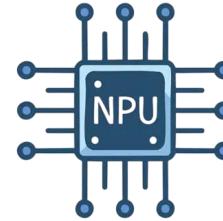


Lumina-Agent：基于数据中心范式与显存感知优化的 端到端小艺语音指令系统



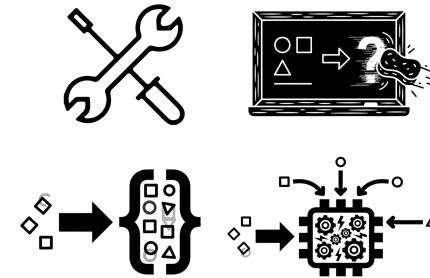
赛题关键痛点

NPU存在显存墙



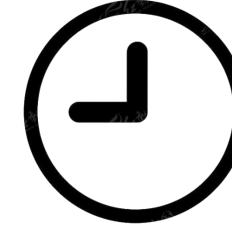
5GB显存硬性限制+NPU对动态长序列的性能敏感性，
限制了 Agent 的推理精度和任务复杂度

逻辑复杂度高

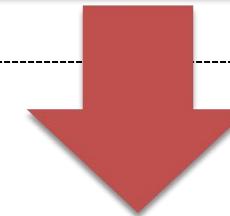


100+ 工具库，多轮对话中的状态遗忘、参数补全、多指令并发，
制约了 Agent 的推理鲁棒性与任务可靠性

时间计分机制



决赛添加推理时延作为评分标准，
设计的 Agent 不仅要准确率高，还要速度快

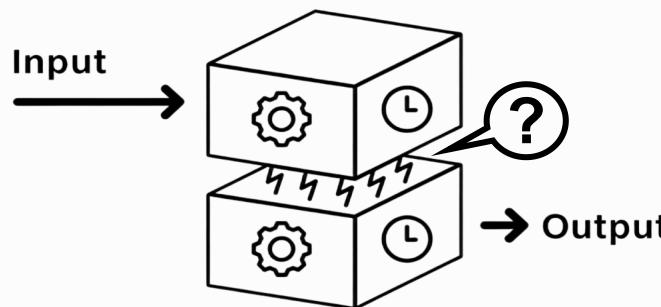


摒弃分层架构，选用 **Direct Agent** 作为 Lumina-Agent 的设计架构



分层架构

- 存在**双重推理延迟和错误级联风险**

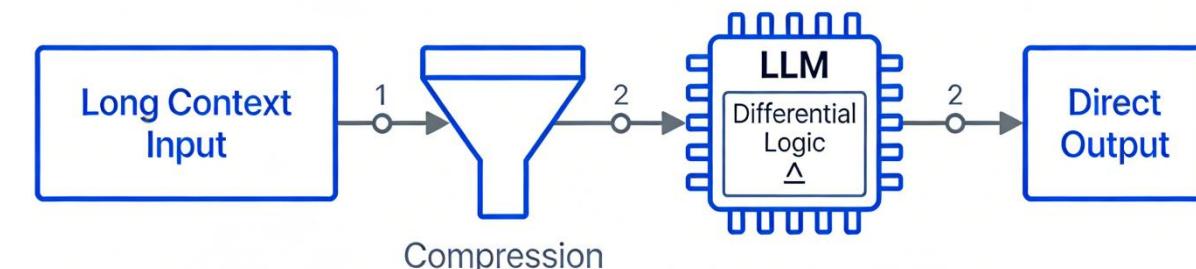
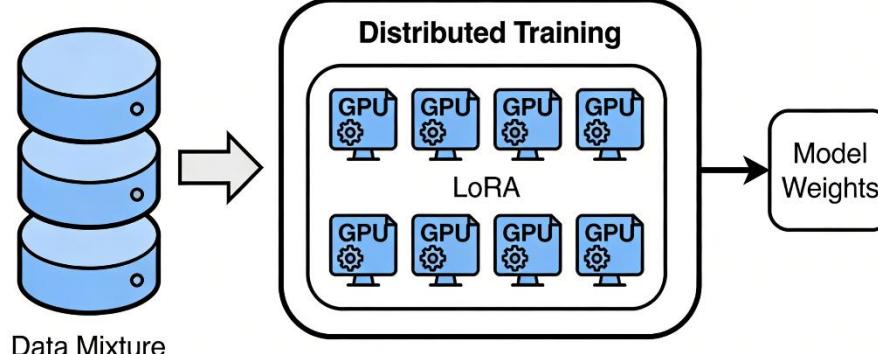


Lumina-Agent (Direct)

- 端到端直出，单次推理即可完成，全局注意力视野，鲁棒性更强**

决策原因

- 为了极致的**推理速度和容错率**，我们选择挖掘 Direct 架构的极限，而非堆砌复杂模块。
- 奥卡姆剃刀原则：如无必要，勿增实体**





创新点1：面向NPU的显存感知压缩

赛题架构

核心创新

实验效果

压缩示例

```
intent_name": "BlueToothOnOff",
"intent_description": "用于识别用户想要控制设备蓝牙功能开启或关闭的语音指令。该意图能够理解用户通过自然语言表达的蓝牙操作需求，包括但不限于打开蓝牙、关闭蓝牙、启用蓝牙连接、禁用蓝牙等相关操作指令。",
"slots": [{"ActionType": "数据类型: Boolean, 取值范围: True|False, 用途: True--打开蓝牙, False--关闭蓝牙"}]
```



BlueToothOnOff(ActionType:[BoolTrue|False])
控制设备蓝牙功能开启或关闭

背景问题

99个工具原始描述导致Prompt长度爆炸，显存占用大且推理慢

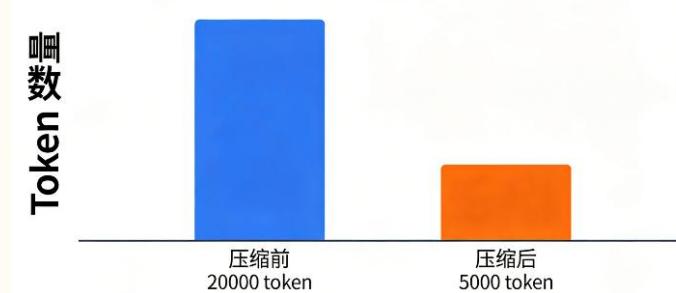
我们的解决方案：语义无损压缩算法

1. 语义蒸馏

提取 `intent_description` 的“动词+宾语”核心结构。

2. 参数剪枝

可将参数示例截断为 Top-3 等等。



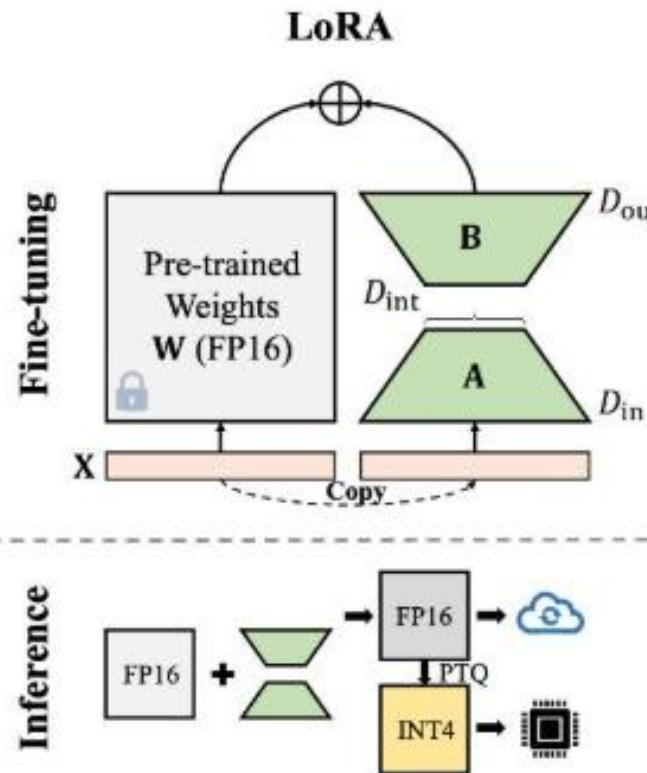
效果：最大上下文长度 **20000->5000**, 推理速度提升，显存占用显著下降



创新点2：消费级显卡上的长序列微调策略

赛题架构 | 核心创新 | 实验效果

LoRA微调示意图



全量微调(Full FT) vs LoRA微调

-
- 比赛是 Text-to-API 任务，输出格式固定
• LoRA 的低秩矩阵足以承载这种格式映射
• 仅训练插入的低秩矩阵（参数量 <2%），显存占用极低

我们的训练方案



1. 梯度检查点

“以计算换空间”。在前向传播中不保存中间层的激活值，显著降低显存占用。

2. 分布式并行DDP

8卡并行 + 梯度累积, 模拟大 Batch Size, 稳定收敛

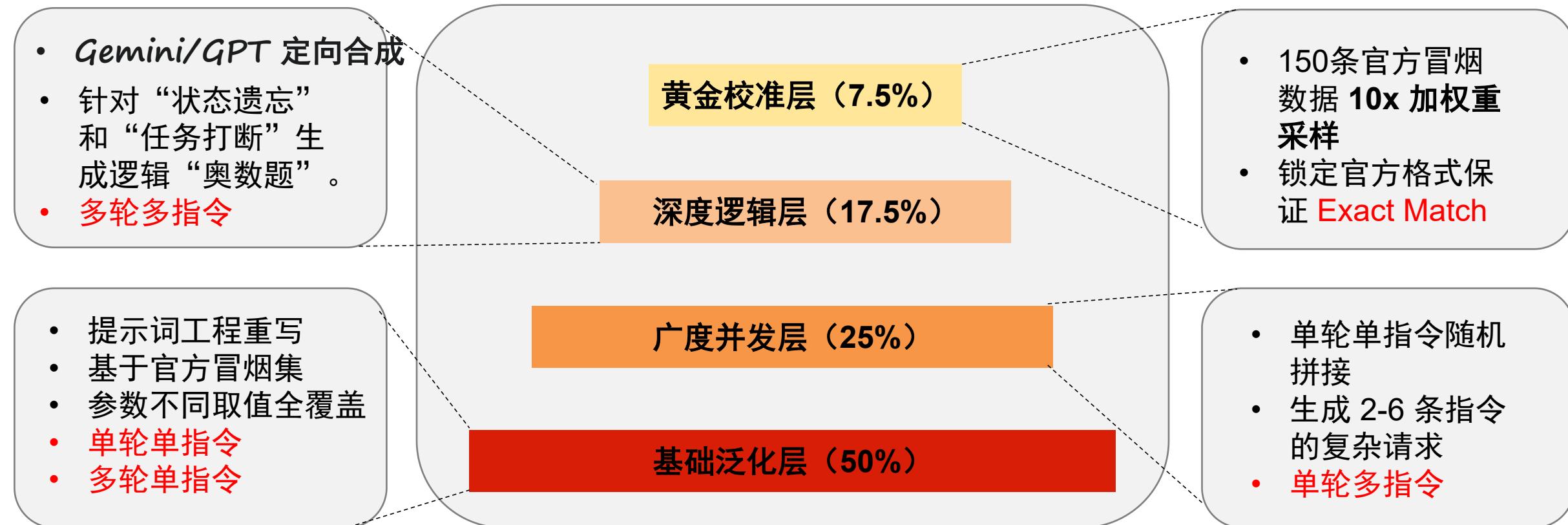
LoRA 通过冻结底座，在学会复杂工具调用的同时，较好保留了模型的通用语义理解能力，且能大幅度提高训练效率，降低训练成本



创新点3：混合专家合成管线的数据工程

赛题架构 | 核心创新 | 实验效果 |

针对语义泛化、结构并发、时序逻辑三大能力短板，构建全场景覆盖的训练集20000+条



在模型结构同质化的情景下，数据质量决定性能天花板。该管线有效解决了单轮的准确度与泛化性，多轮对话中的状态遗忘与参数幻觉问题



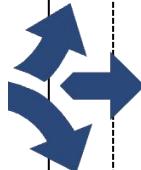
创新点3：混合专家合成管线的数据工程

赛题架构 | 核心创新 | 实验效果

全流程的数据反幻觉清洗机制，修剪模型的认知边界



训练数据清洗



时空语义修正

模型死记硬背
训练时的绝对日期

- Before
StartTime="2025-12-08 10:00"
- After
StartTime="明天 10:00"

稀疏参数清洗

模型过度生成无效参数
可能导致判分失败

- Before
SavePath=""
Time="None"
List=[]
- After
※

特殊参数保留

对于特殊的函数和
参数值需保留特例

ConnectWlan(Password="None")
无需提供密码的情景

“帮我把系统日期
调整为2024-1-1”

净化数据噪声，规避逻辑陷阱，以高质量数据基座筑牢模型的精准推理能力



创新点4：基于提示词的状态感知推理

团队介绍 | 赛题架构 | **核心创新** | 实验效果

针对决赛多轮多指令对话，容易重复输出第一轮已经执行过的指令，导致逻辑冗余

破局思路：In-Context 状态过滤流

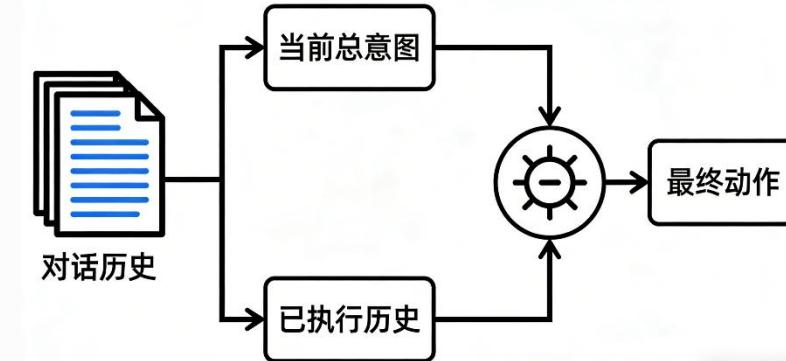
摒弃臃肿的**外部状态机**；利用 LLM 强大的上下文学习能力，在 System Prompt 中植入一套**逻辑筛**

Prompt 逻辑拆解与实现

思维链四步法

- 1 全量扫描：**提取所有用户需求集合 A
- 2 已完成核销：**检索历史回复，识别已执行集合 B
- 3 差量执行：**计算 $A - B$ 仅输出差集。
- 4 增量补全：**结合前文意图 + 本轮新参数
→合成完整指令。

实现示意



必须严格执行以下逻辑：

1. 全链路分析：不要仅关注最后一句。审视整个对话，提取用户意图和相关的参数细节（参数可能分散在不同对话轮次中）。
2. 已执行过滤（关键）：检查历史对话中的 Assistant 回复。如果某个需求在之前已经明确输出过函数指令，则本轮绝对不要重复输出，除非用户要求修改。
3. 待执行合并：
 - 对于本轮新提出的需求：立即生成指令。
 - 对于前文正在澄清的需求（如用户补全了缺失参数）：结合前文意图和本轮参数，生成完整指令。
4. 多指令输出：将所有计算出的“当前待执行”指令，用竖线 | 连接。

将上述思维链编写进**System Prompt**, 配合LoRA 在深度逻辑层数据上的训练，将这种“状态感知”内化为模型的直觉



04 实验验证与总结——泛化边界探索

赛题架构 | 核心创新 | **实验效果**

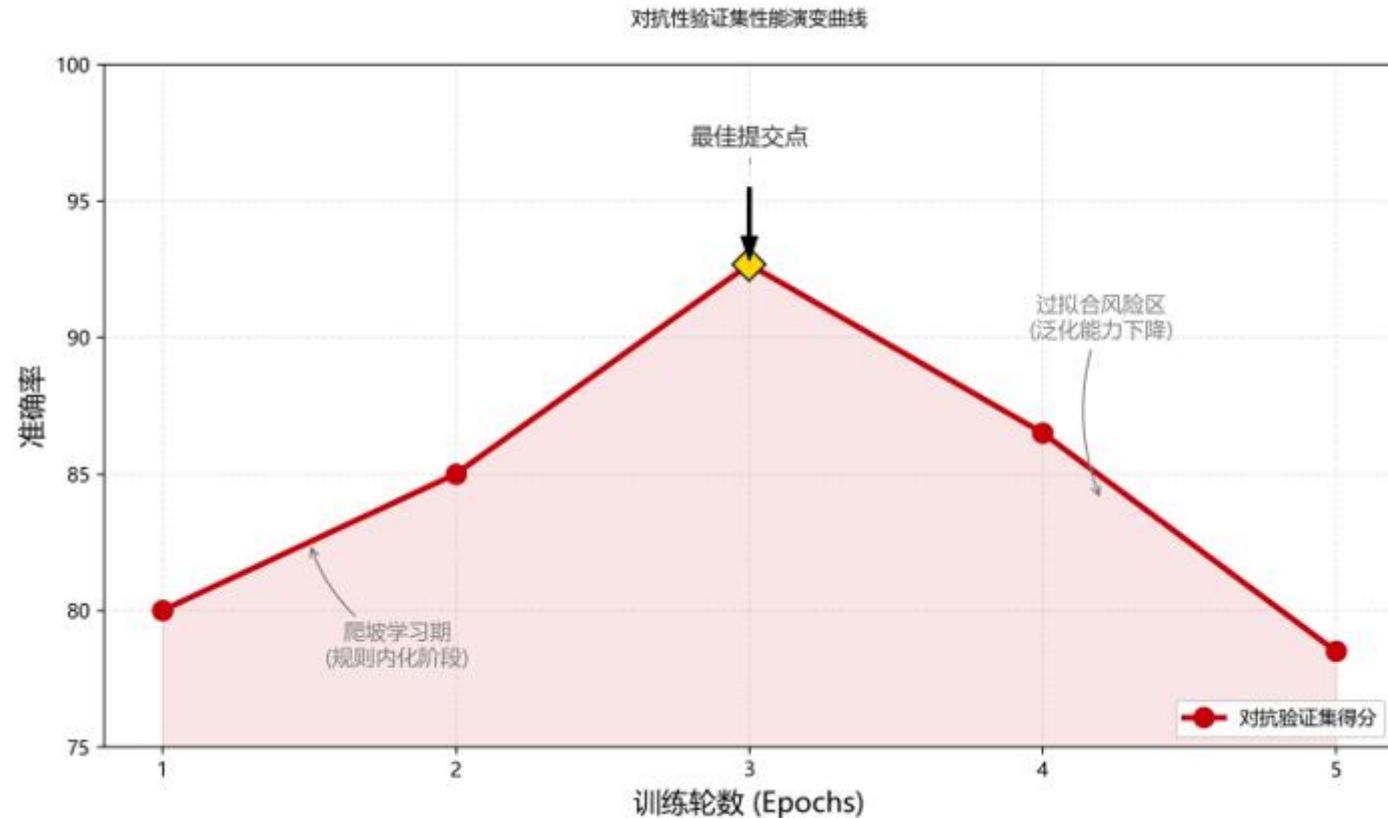
实验设置

- 针对基于高难度对抗验证集（100条逻辑陷阱题）进行评估
- 评估指标：人工判定模型回答是否正确

实验发现

- 爬坡期 (Epoch 1–2)：模型正在学习，分数稳步上升
- 甜蜜点 (Epoch 3)：达到峰值。此时模型对新指令的逻辑判断力最强，泛化能力达到巅峰
- 退化期 (Epoch 4–5)：分数滑落。模型陷入过拟合，导致在对抗测试中丢分

决策

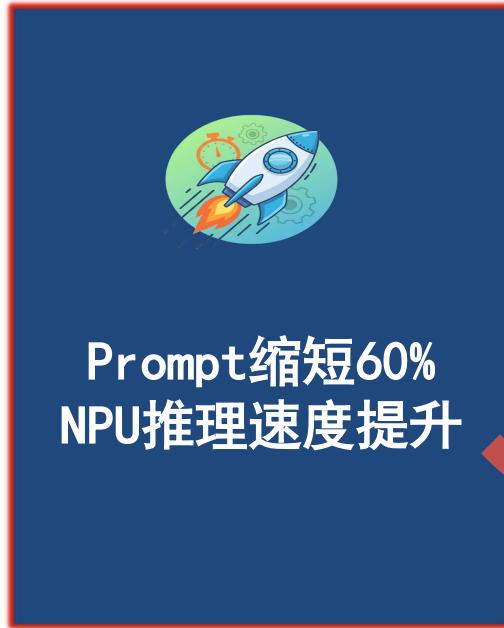


训练轮数影响实际泛化能力，在本地测试集上得到的结论，与最终提交的得分规律一致



04 实验验证与总结——核心价值

赛题架构 | 核心创新 | **核心价值**



高效率

低资源

高性能

模型的迭代精益求精，技术的探索永无止境