西湖大學
WESTLAKE UNIVERSITY

自主进化智能体：从固定工作流到动态架构的演进

张驰· 西湖大学 AGI 实验室

西湖大学 WESTLAKE UNIVERSITY

Agent 变强时，究竟是哪里在变？

"模型更大"并不是唯一的路径。

1 · 知识

先理解
领域

2 · 经验

再学会
操作

3 · 动作

然后提炼
重复动作

4 · 架构

最后
自我改写

这四篇工作构成一条完整的进化路径——从低层到高层。

西湖大學 WESTLAKE UNIVERSITY

同一条主线，四个进化层级

1	2	3	4
SQLAgent	**AppAgent**	**AppAgentX**	**Doctor**
知识 进化	经验 进化	动作 进化	架构 进化

→ → →

抽象层级逐步升高 —— 越往右，Agent 改变的越不是"内容"，而是"自身"。

用同一套问题来读每一篇工作

01
任务是什么
先让听众明白它到底要做什么

02
难点在哪
只讲与方法直接相关的难点

03
为什么这样做
讲清背后的动机

04
怎么做到
用论文原图 + 具体例子

05
结果如何
最后回到实验成绩

这样讲，方法不会悬空；听众也更容易记住为什么"需要进化"。

西湖大學 WESTLAKE UNIVERSITY

四篇工作其实都在回答同一个问题

当任务不再一次性完成，**Agent** 能否把过去变成未来的优势？

先探索	→	再沉淀	→	后复用	→	最终自改
SQLAgent / AppAgent		知识 / 文档 / 历史		示例 / shortcut / workflow		Doctor

探索 → 沉淀 → 复用 → 自我优化

06

层级 1 · SQLAgent

先熟悉数据库，再开始写 SQL

让 Agent 在写查询之前先探索数据库 schema。

西湖大學 WESTLAKE UNIVERSITY

先说说：什么是 SQL？

核心概念

SQL（结构化查询语言）是向数据库**提问**的标准方式——数据以行和列组成的表格存储。

你只需描述**想要什么**；数据库会自动决定**怎么取**。

表	由行 × 列组成，像电子表格
声明式	只说要什么，不写步骤
连接（Join）	一条查询可组合多张表

小例子 · 表 "orders"

id	merchant	amount
1	Acme	$120
2	Globex	$80
3	Acme	$200

Q：「每个商户各有多少订单？」

```
SELECT merchant, COUNT(*)
FROM orders  GROUP BY merchant;
```

→ **Acme: 2 · Globex: 1**

SQL 本身就这么简单——向数据库提出一个精确的问题。难的是足够了解数据，才能问对问题。

任务：把自然语言问题翻译成正确的 SQL

用户只问一句话——Agent 要自己找到表、字段、连接关系和聚合逻辑。

"过去 30 天里，哪个商户的退款率最高？"

4 张表 · 多次 join · 口径选择

- orders
- payments
- refunds
- merchants

一句短短的问题，背后可能藏着 4 张表、多个 join 和口径选择。

> Text-to-SQL 的本质不是"翻译"，而是先理解数据库。

三个问题例子

收入

"2024 年每个月的净收入是多少？"

需要区分 gross / refund / tax 的口径。

留存

"哪些客户连续三个月都下单？"

需要跨月窗口和去重。

质量

"哪些供应商的异常率最高？"

需要知道哪个字段才代表"异常"。

问题不难写，难的是先知道"数据库里的世界长什么样"。

挑战 1：同名字段，未必同义

order.status

payment.status

refund.status

它们都叫 **status**，但分别代表订单、支付、退款的状态。

拿错字段

可能取错列。

join 错表

可能连错表。

看着对，其实错

SQL 看似合理，语义却是错的。

真正缺的不是 SQL 语法，而是对数据库语义的熟悉度。

挑战 2：在陌生数据库里，schema 本身就很大

把整库 DDL 全塞进去，信息太多；只检索几张表，又容易漏掉关键关系。

全量 schema

上下文爆炸——一次性信息太多。

Top-k 检索

容易漏表、漏关系。

人工熟悉

准确，但成本太高。

所以不能只在"查询时推理"，而要在"查询前进化"。

人的做法：先熟悉数据库，再写查询

浏览 schema
先知道有哪些表

→

试写查询
确认字段含义

→

沉淀经验
记住可复用的模式

→

正式回答
更快、更准

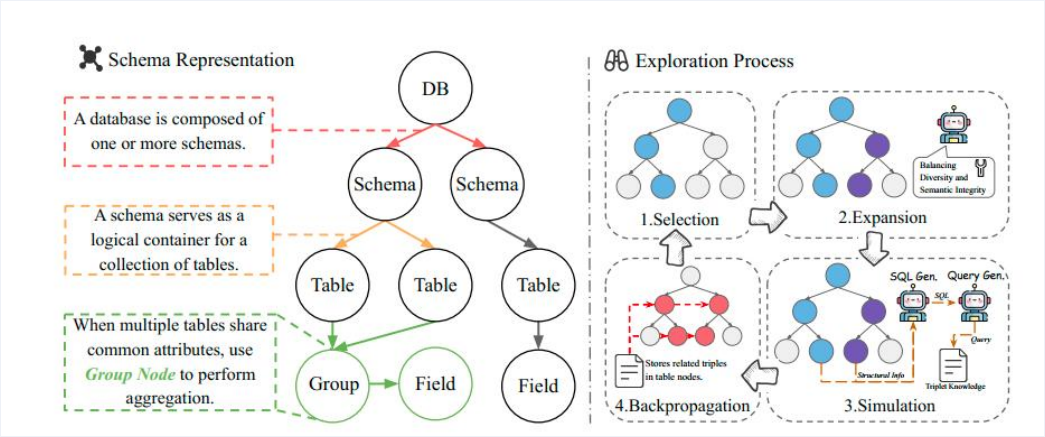
SQLAgent 的核心直觉：让 Agent 也经历一次"上手数据库"的过程。

两阶段框架：先探索，再部署

论文 · Figure 1 / Figure 2

左：如何把数据库表示成可探索的树。

右：如何把探索结果用于正式问答。

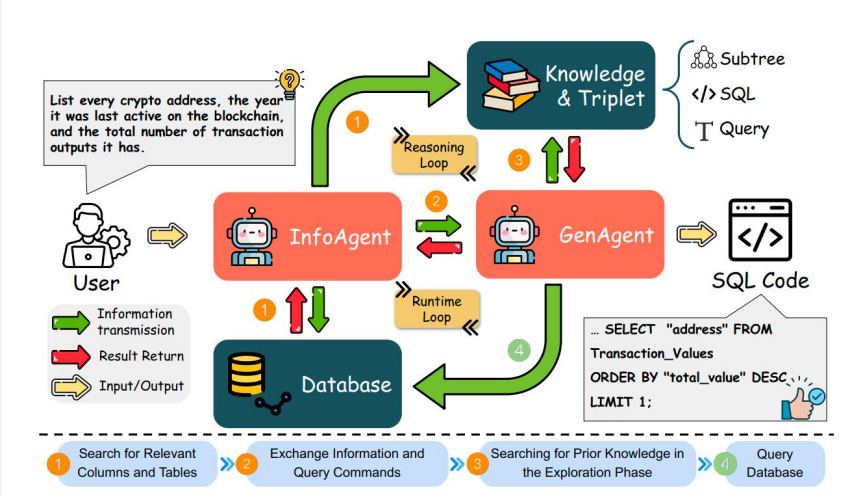


两阶段框架：先探索，再部署

论文 · Figure 1 / Figure 2

左：如何把数据库表示成可探索的树。

右：如何把探索结果用于正式问答。



探索阶段，到底"学"到了什么？

不是乱逛——它会生成可复用的三元组知识。

Schema 片段

相关的表与列。

SQL 查询

一条能跑通的查询。

自然语言

它所回答的问题。

示例： "高退款率商户" → 涉及哪些表 → 如何 join

部署时，这些会变成 in-context 示例——而不必每次重新摸索。

西湖大學 WESTLAKE UNIVERSITY

一个三元组到底长什么样

一条被存下来的知识 = 一个问题、它涉及的 schema，以及回答它的查询。

第 1 部分
自然语言

"过去 30 天里，哪个商户的退款率最高？"

第 2 部分
Schema 片段

```
merchants ( id, name )
orders    ( id, merchant_id, created_at )
refunds   ( id, order_id, amount )
```

第 3 部分
SQL 查询

```sql
SELECT m.name, COUNT(r.id)*1.0 / COUNT(o.id) AS refund_rate
FROM merchants m  JOIN orders o ON o.merchant_id = m.id
LEFT JOIN refunds r ON r.order_id = o.id
WHERE o.created_at >= NOW() - INTERVAL '30 days'
GROUP BY m.name  ORDER BY refund_rate DESC  LIMIT 1;
```

探索时存一次 → 部署时作为现成的 in-context 示例直接取用。（例如 top 结果：Globex，18.4%）

西湖大學 WESTLAKE UNIVERSITY

正式回答时，是两个 Agent 在配合

InfoAgent

检索相关 schema 与已积累的经验。

GenAgent

生成 SQL，并在需要时修正。

为什么不是单个 Agent？

找对上下文

一个负责找对上下文。

写对 SQL

一个负责写对 SQL。

结果 1：加入探索后，整体 EX 明显提升

25.78%

整体 EX

+12.80

vs. Spider-Agent

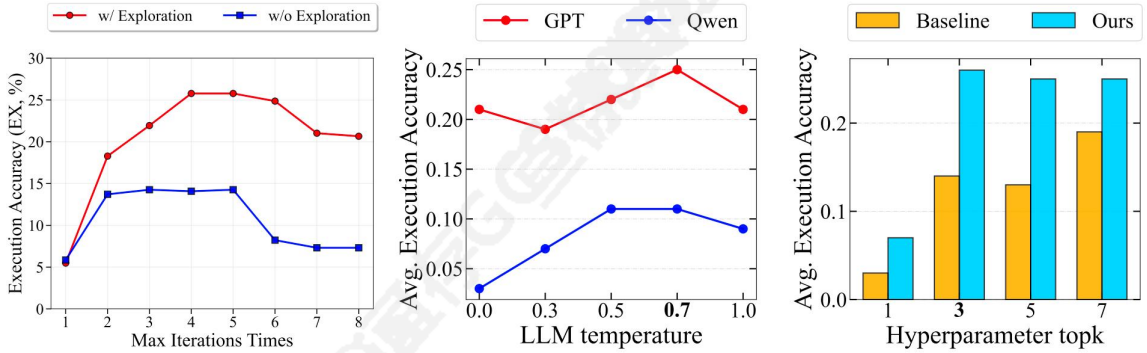
稳定的增益

在 Easy 与 Hard 子集上均有提升。

有探索 > baseline。

知识不是副产品——它直接转化为任务成功率。

结果 2：知识积累越多，表现越稳



(a) Performance During Iteration. **(b)** Analysis of Two Key Hyperparameters

LLM	Cost	EX@8 (%)		Δ
		Baseline	Full Method	
Qwen2.5-7B	20.8k	7.12	14.99	+7.87
GPT-4o	15.2k	21.57	31.99	+10.42
GPT-5	12.5k	22.49	32.90	+10.70
Claude-Sonnet-4.0	11.6k	29.97	39.12	+9.15

积累的知识会复利——后续回答既更准又更稳。

西湖大學 WESTLAKE UNIVERSITY

SQLAgent 的"进化"是什么？

从

不熟悉这个数据库

→

到

先探索，后复用

它进化的不是动作，也不是架构，而是——

领域知识

领域知识：Agent 在回答前先熟悉数据。

18

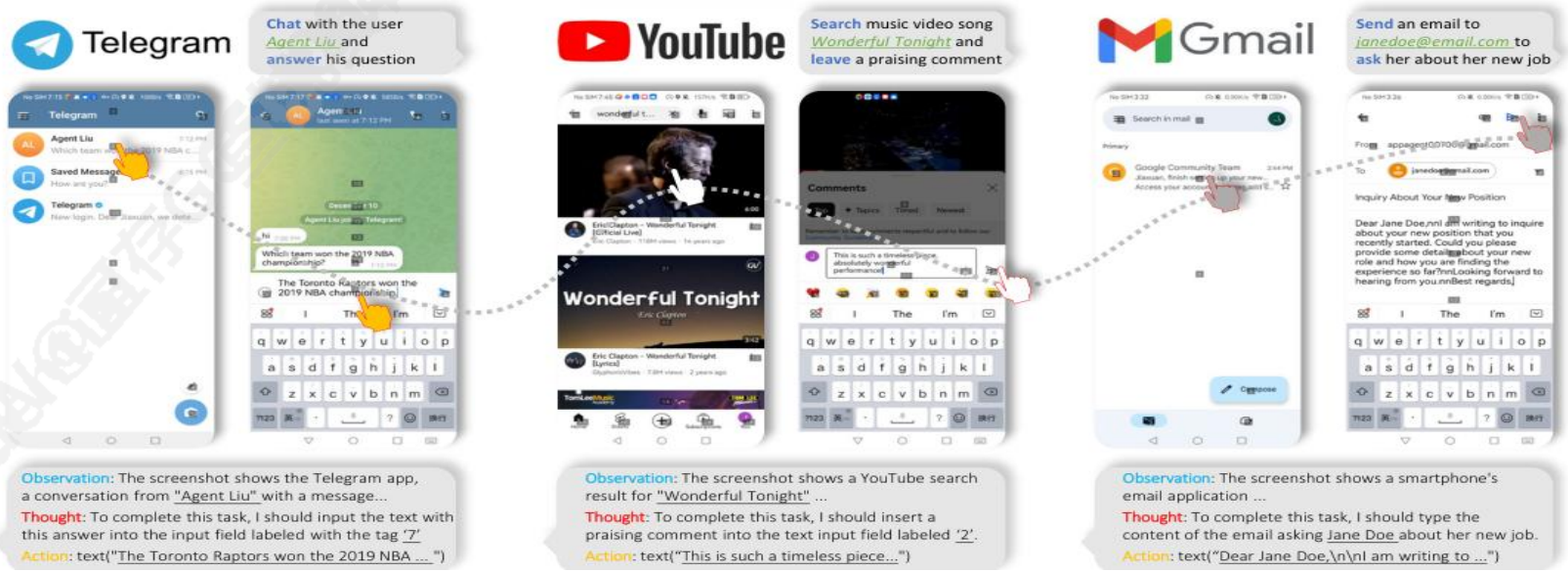
层级 2 · AppAgent

像人一样学会使用 App

像真实用户那样，靠看屏幕来操作 GUI。

任务：不用 API，只看屏幕来操作手机 App

像真实用户一样：看截图、理解界面、点按钮、输入文字。



这些任务和人类日常操作几乎一样。

几个任务，一看就懂它要解决什么

Gmail

给 Jane Doe 发一封邮件。

Clock

每周五和周日 12:30 设一个闹钟。

Lightroom

把照片调得更好看。

任务本身不复杂——复杂的是每个 App 的界面规则都不同。

挑战 1：GUI 世界没有统一的"语言"

同一个目标：
"搜索"

→

不同 App

按钮位置千差万别。

不同控件

图标、文案、手势都可能变。

如果每一步都从零理解屏幕，Agent 会又慢又脆。

西湖大學 WESTLAKE UNIVERSITY

挑战 2：原始动作空间太难用

原始坐标

"点击 x = 241, y = 818"

脆弱、难懂，也难以学习。

拟人化动作

Tap · Swipe · Text · Back

可读、可复用、更稳定。

成功率：　2.2%　→　48.9%

先把动作说成"人话"，Agent 才能稳定学会。

人的做法：先自己摸索，或先看别人怎么做

自己探索
亲自上手试用

→

观看示范
看别人怎么做

→

形成文档
把有用的写下来

→

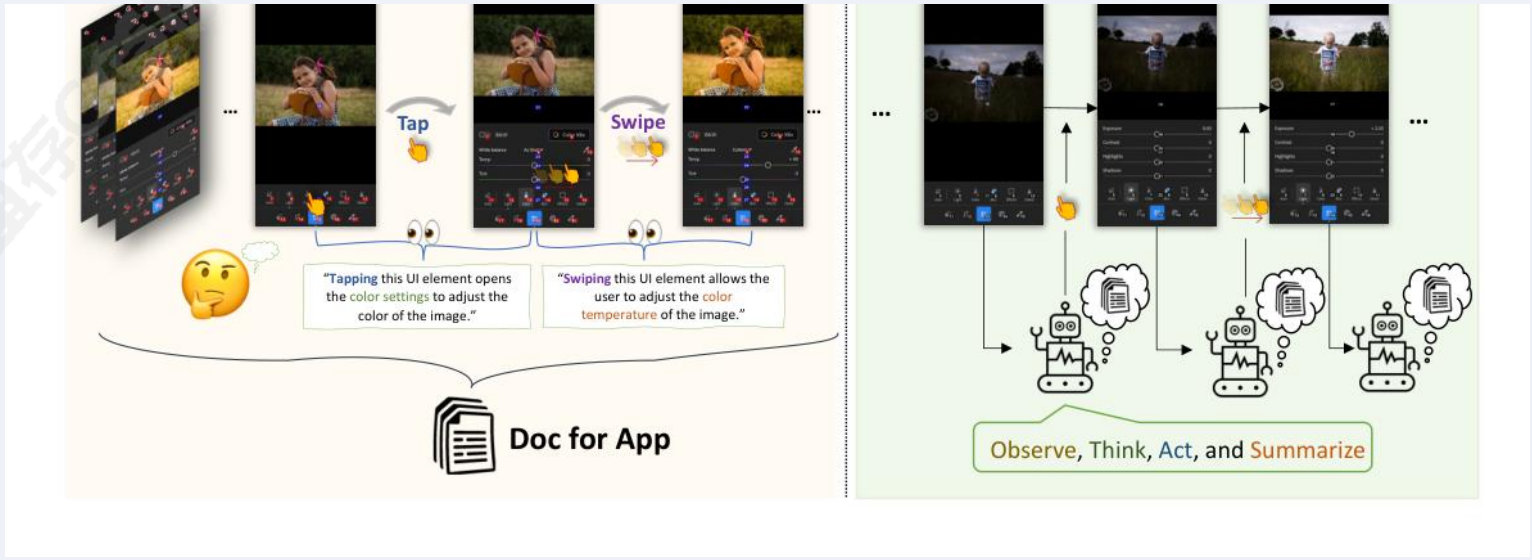
任务中复用
用到真实任务上

AppAgent 想让 GUI Agent 也拥有自己的"使用说明书"。

西湖大學 WESTLAKE UNIVERSITY

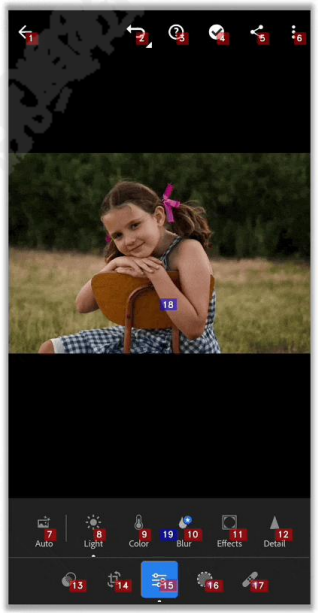
探索 — 文档 — 部署

论文 · **Figure 2**



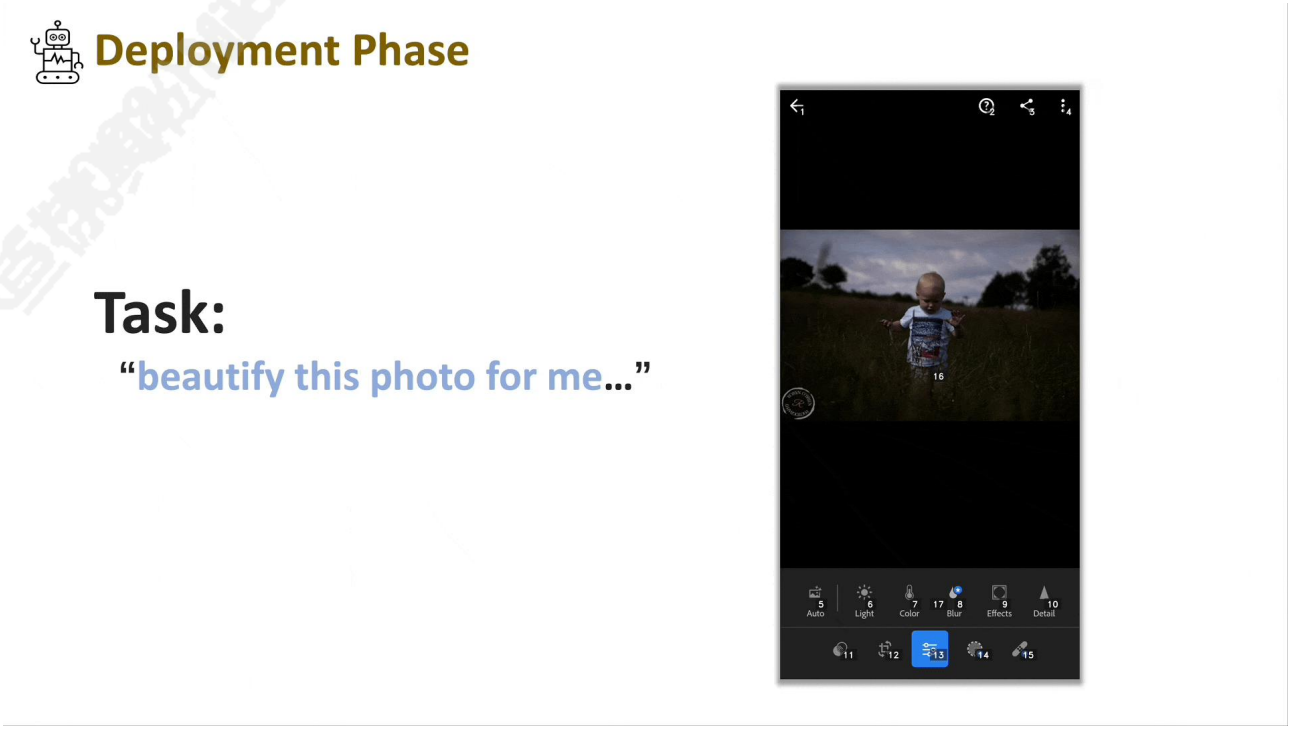
探索阶段，Agent 到底记录了什么？

Exploration Phase 👀



这一步让"经验"第一次变成可复用的外部记忆。

部署时：Observe → Think → Act → Summarize



一个紧密的循环：文档指导每一步动作，每一步动作又丰富文档。

结果 1：经验文档显著提升任务成功率

自动探索	观看示范	人工文档
73.3%	84.4%	95.6%

自动学到的经验，已经非常接近人工编写的说明书。

结果 2：不只是点按钮——也能做视觉任务

Lightroom · 案例研究

Agent 不仅完成了任务，还能产出更好的视觉效果——像熟练的修图师那样调整照片参数。

Agent 不仅完成了任务，还能产出更好的视觉效果——像熟练的修图师那样调整照片参数。

[论文配图]

西湖大學 WESTLAKE UNIVERSITY

AppAgent 的"进化"是什么？

从

每次都从零现学

→

到

积累操作经验

它进化的是——

经验

经验：关于每个 App 如何运作的可复用知识。

下一个问题：经验足够多时，Agent 能否把重复动作压缩成一个"捷径"？

30

层级 3 · AppAgentX

从"会做"到"熟练地做"

把重复的低层动作压缩成专家级的捷径。

挑战 1：每一步都完整推理，代价太高

感知

读取屏幕

→

思考

规划下一步

→

执行

执行一步

× **每一步** → 又慢，又消耗大量 token。

经验如果不能压缩，就只是"记得更多"——还算不上"更熟练"。

挑战 2：基础动作空间太低层

低层：手指怎么动

Tap · Swipe · Text

这些只描述了机械动作。

高层：用户想做什么

Search · Change Theme

这些描述的是意图。

要真正进化，Agent 必须把低层动作抽象成高层行为。

西湖大學 WESTLAKE UNIVERSITY

任务：同样操作 App——但要更快、更省、更像专家

此前：Agent 每一步都要完整推理。

每步都调用 LLM

每步都重新看屏幕

对反复出现的套路毫无记忆。

AppAgentX 想学会"熟练动作"——像不再逐步思考的专家用户。

人的熟练，本质上是"动作压缩"

初学者

看键盘 → 找到键 → 按下去。

一串低层动作序列。

熟练者

直接打出一个词。

一个高层捷径。

AppAgentX 把"经验"再蒸馏一步——变成动作空间的进化。

熟练用户不会每次都从零推理

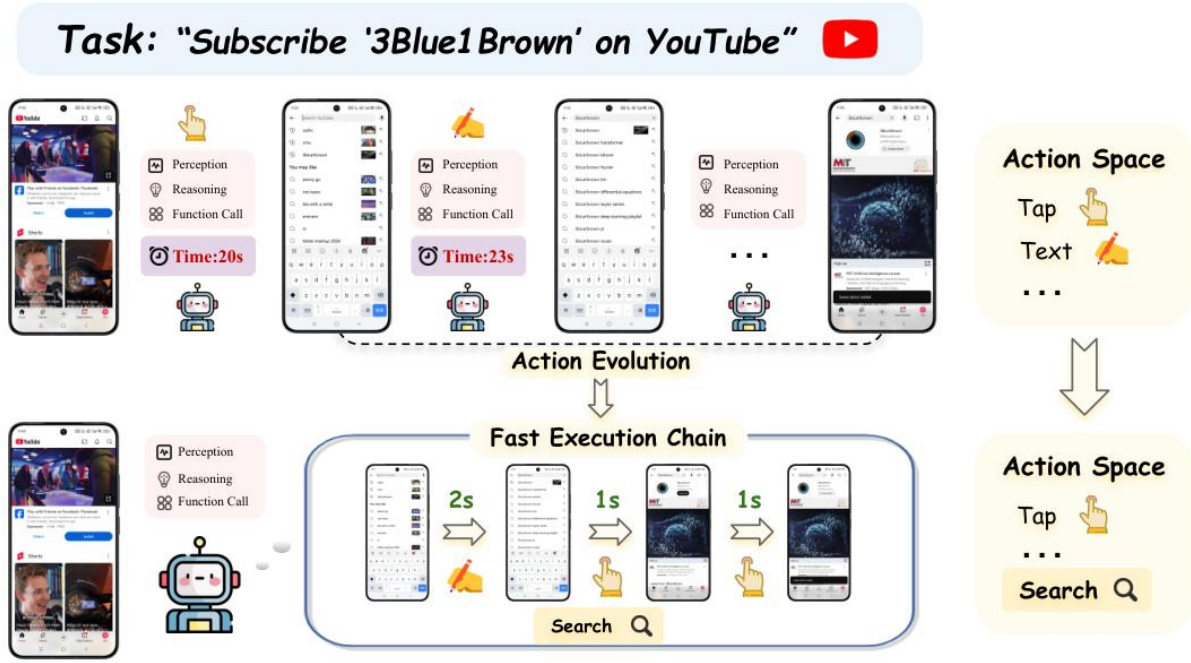
一串低层动作

tap → swipe → tap → tap …

↓

压缩成一个捷径

"搜索某某东西"



熟练的关键在于压缩：把常做的事变成一个动作。

历史 → 记忆 → 捷径 → 动态执行

历史

过去动作的轨迹

→

记忆

结构化的页面/元素图

→

捷径

蒸馏出的高层动作

→

动态执行

调用捷径，或回退到逐步执行

论文 · Figure 3——不是简单缓存，而是把反复出现的模式蒸馏成高层动作。

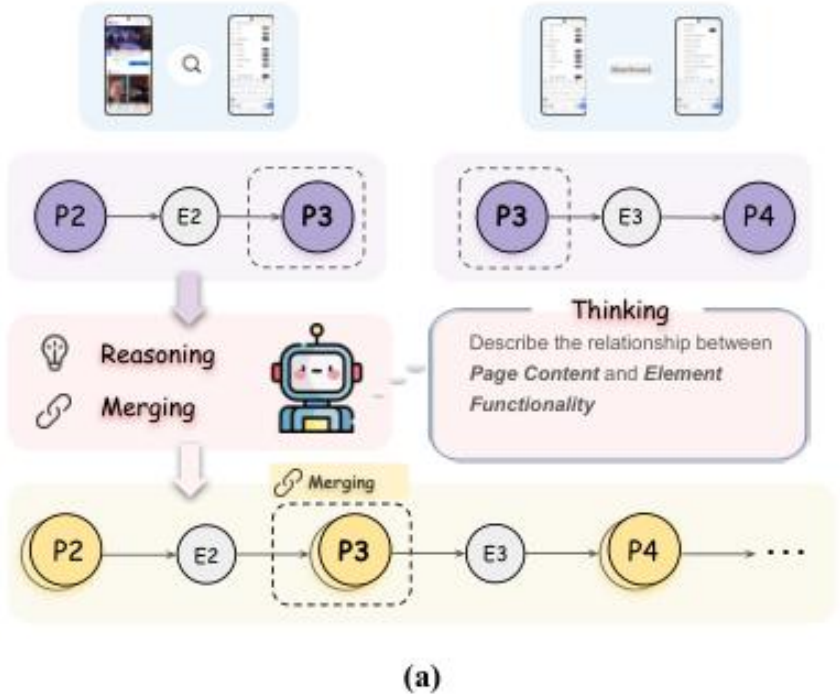
先把历史组织起来：Page Node → Element Node

页面之间

记录跳转关系——一个界面如何到达另一个。

页面内部

记录每个元素的功能——每个控件能做什么。

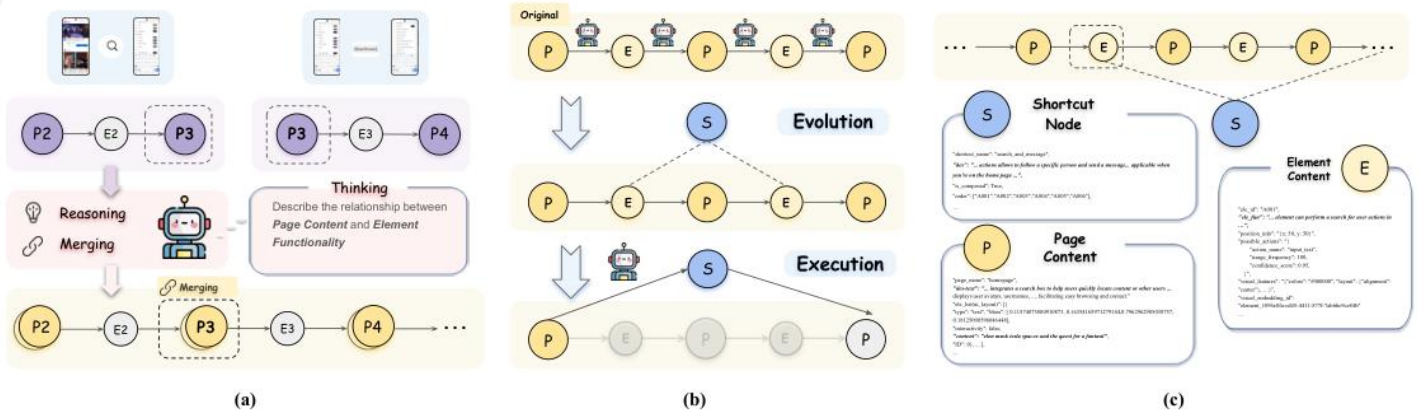


(a)

现在 Agent 记录的不只是"做过什么"，而是"这里能做什么"。

再把重复的序列压成一个捷径

tap → swipe → tap → tap → 搜索



压缩，正是把"记忆"变成"熟练"的机制。

结果 1：步骤更少、时间更短、token 更省

9.1→5.7

步骤

23s→16s

每步耗时

9.26k→4.94k

Token

而且成功率仍保持在 71.4%——更快更省，也没有牺牲准确率。

结果 2：在大规模基准上同样更强

46.3→88.2

DroidTask (%)

41.7→62.5

AndroidWorld (%)

这种提升不是小规模的偶然——在广泛的 GUI 基准上都成立。

西湖大學 WESTLAKE UNIVERSITY

AppAgentX 的"进化"是什么？

从

低层操作

→

到

高层动作

它进化的是——

动作空间

动作空间：把重复的低层动作抽象成可复用的捷径。

下一个问题：如果动作能自我改写，整个 workflow 能不能也自己长出来？

42

层级 4 · Doctor

连自己的 workflow 都能改写

从反馈中搜索并进化诊断 workflow。

任务：做医学图像诊断，而且要持续变得更准

输入

一张皮肤图像 + 一个诊断 workflow。

输出

病种预测，并给出置信度。

关键不在于它能否回答——而在于流程本身是否合理。

同样是诊断，不同的错误需要不同的修复

图像理解错误

看错了病灶特征。

修复：增补工具 / 视觉节点。

诊断推理错误

特征看对了，病推错了。

修复：增补会诊 / 分支。

一致性不足

多次运行不稳定。

修复：增加循环 / 投票。

真正需要进化的不是单个 prompt——而是整套 workflow 结构。

挑战 1：静态 workflow 太脆弱

固定流程

不管什么病例都是同一套步骤。

固定 prompt

完全不随输入调整。

固定输出

所有情况都是同一种格式。

不会自我纠正

即使错了也不会改流程。

一套套路打天下

不同病因却用同样的处理。

在高风险场景里，"固定"本身就是瓶颈。

挑战 2：只调 prompt 远远不够

有些错误需要加节点，有些需要加分支，有些则需要换一整套协作范式。

节点级

加一个工具，或改一个专家节点。

结构级

增加分支 / 循环 / 并行。

框架级

从 CoT 换成 Round Table。

真正的修复发生在三个不同层级——而不只是措辞上。（论文 · Figure 2）

像 NAS 那样，搜索更好的 Agent workflow

初始 workflow

从一个基础流程开始

→

诊断错误

找出失败的地方

→

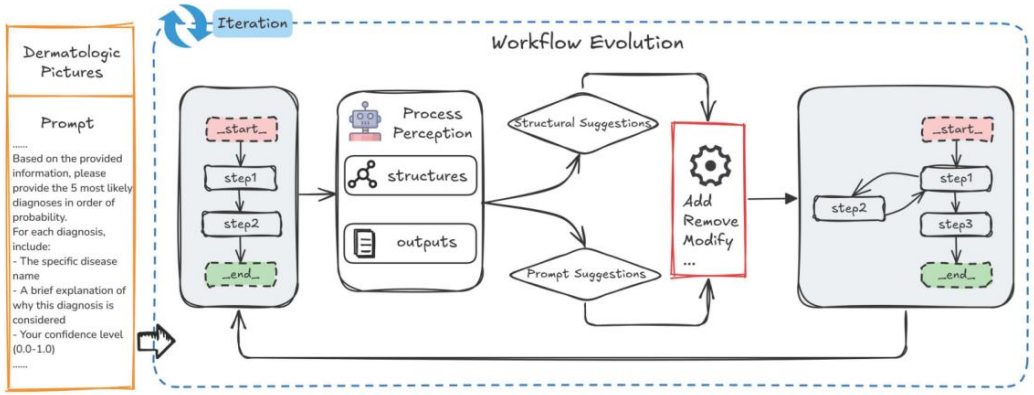
提出修改

给出结构性的改动建议

→

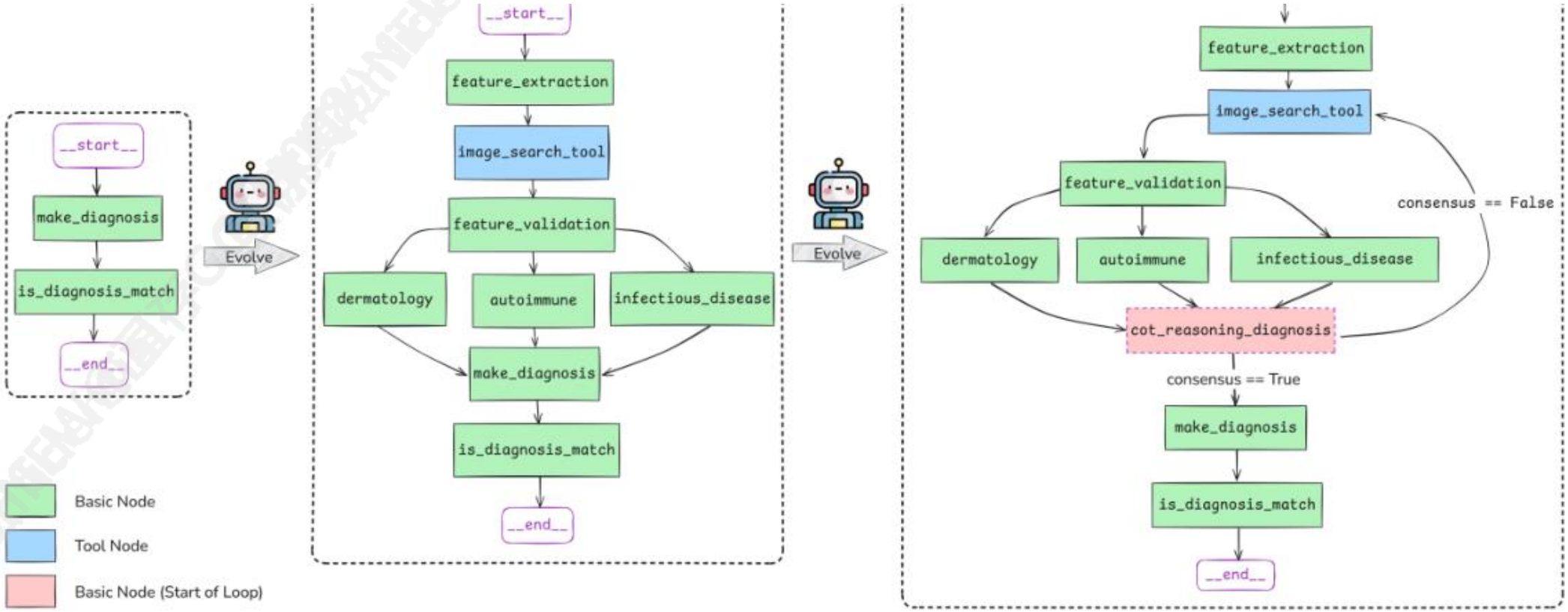
验证并保留

留下能带来提升的



关键转变：workflow 从"人工设计"变成"反馈驱动的搜索"。

错误诊断 → 提出修改 → 验证 → 进化



特别之处：搜索空间是分层的

节点级

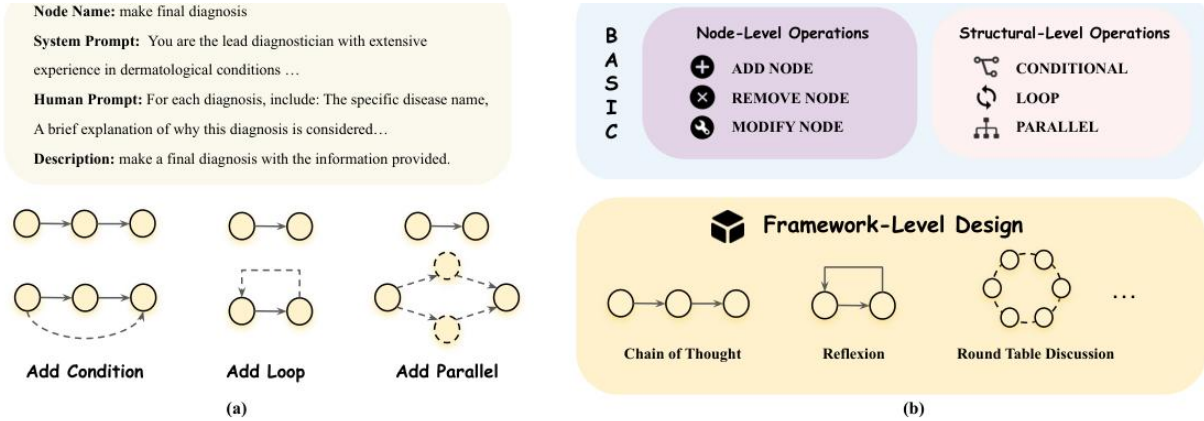
增 / 删 / 改 一个节点。

结构级

分支 / 循环 / 并行。

框架级

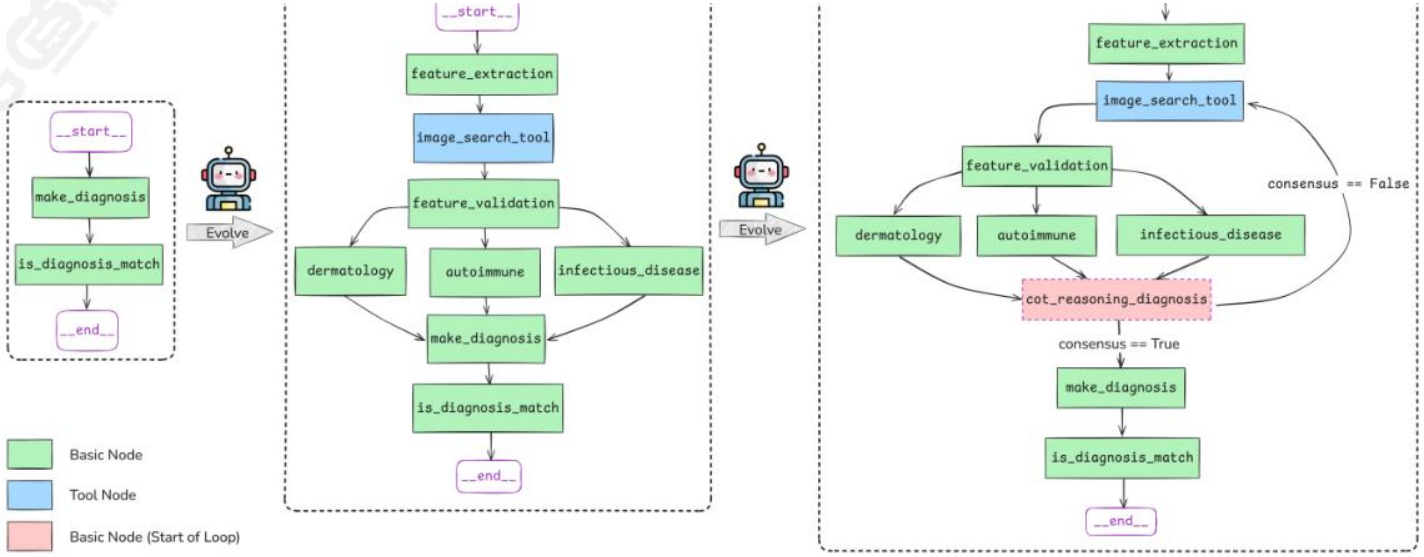
CoT / Reflexion / Round Table.



不是单点调参——而是在三个层级上同时搜索。

workflow 会真正"长出来"

Iteration 0	Iteration 3	Iteration 6
单一的线性链路	出现分支	并行与循环



从单链路长成分支、并行和循环——结构在自我进化。

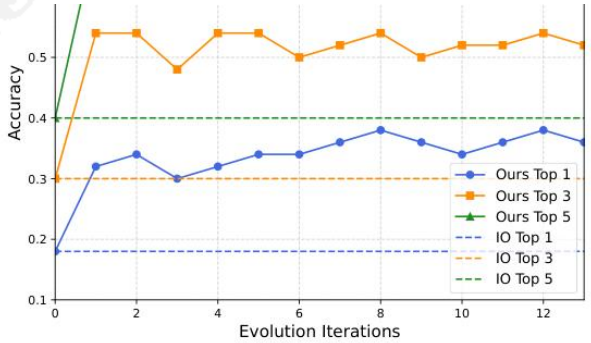
结果 1：跨模型、跨数据集都有显著提升

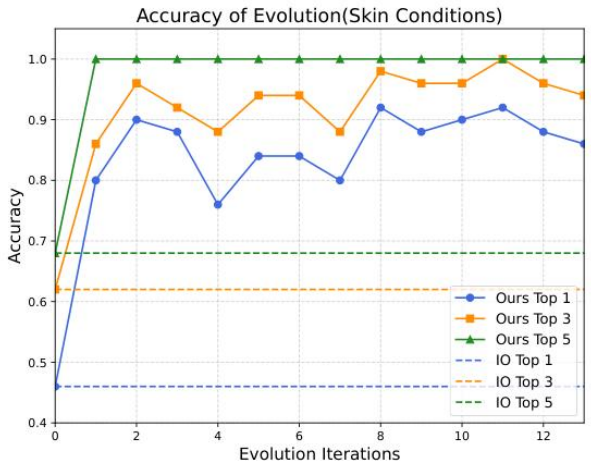
Table 1: Top-k diagnostic accuracy (%) of different methods using GPT-4o, GPT-4o-mini and Claude 3.5 Sonnet on Skin Concepts and Skin Conditions.

LLM	Method	Skin Concepts Accuracy (%)			Skin Conditions Accuracy (%)		
		Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
GPT-4o	IO	20.27	30.63	36.04	50.83	78.33	86.67
	CoT (Wei et al., 2022)	18.47	28.83	33.78	55.83	76.67	82.50
	Round Table (Chen et al., 2024c)	21.17	27.93	32.43	45.83	75.83	80.83
	Ours	**29.28**	**40.09**	**50.45**	**90.83**	**95.00**	**100.00**
GPT-4o-mini	IO	11.71	20.72	23.87	27.50	69.17	80.83
	CoT (Wei et al., 2022)	6.31	15.32	24.32	22.50	65.00	84.17
	Round Table (Chen et al., 2024c)	10.81	19.82	23.42	25.83	70.00	78.33
	Ours	**13.51**	**21.62**	**24.77**	**45.83**	**74.17**	**85.83**

在 GPT-4o、GPT-4o-mini、Claude 3.5 Sonnet 上，我们的方法都领先。

结果 2：进化不是乱改——它会收敛





搜索是有方向的：改动不断累积，趋向更好、更稳的 workflow。

Doctor 的"进化"是什么？

从

人工设计的 workflow

→

到

自动搜索的 workflow

它进化的是——

系统架构

系统架构：Agent 搜索并改写自己的 workflow。

这已经不只是"更会做任务"——而是"更会设计自己"。

统一视角

把它们拼起来

四篇工作，拼出一张 Agent 进化地图

SQLAgent	AppAgent	AppAgentX	Doctor
知识	经验	动作	架构
先懂领域	学会操作	提炼捷径	自我改写

西湖大學 WESTLAKE UNIVERSITY

共同设计哲学：先探索，后利用

SQLAgent	AppAgent	AppAgentX	Doctor
探索 探索数据库	探索 探索 App	探索 收集历史	探索 诊断错误
↓	↓	↓	↓
利用 用知识库	利用 用文档	利用 用捷径	利用 用更好的 workflow

真正在各篇之间复用的不是某个 trick——而是一种思路：先积累，再利用。

它们到底差在哪里？

论文	进化对象	主要机制	应用域
SQLAgent	知识库	MCTS 式探索	数据库
AppAgent	经验文档	探索 + 示范	手机 App
AppAgentX	动作空间	历史模式抽象	手机 App
Doctor	Workflow 架构	反馈驱动搜索	医学诊断

把四篇论文放在同一坐标系里

知识 → 经验 → 动作 → 架构

西湖大學 WESTLAKE UNIVERSITY

接下来——Agent 还能往哪些方向继续进化？

学习算法

它能否改写自己的学习规则？

跨域迁移

经验能否跨领域迁移？

多 Agent 协同进化

多个 Agent 能否互相塑造？

安全与可控

如何保证进化的方向是可靠的？

谢谢聆听

西湖大学 WESTLAKE UNIVERSITY

西湖大学 AGI 实验室