Flink Forward ASIA
Organized by Alibaba Cloud

AI Agent 的两朵云

流式与环境交互 · 自主从经验中学习

李博杰 | Bojie Li
https://01.me

ShenZhen
2026. 6.26 - 27

OVERVIEW

两朵云：当前 AI Agent 的两大难题

我们已习惯会写代码、做研究的 AI，却很少见到能像人一样实时对话、实时操作电脑的 AI。

流式与环境交互

语音与视频都是必须实时处理的流式数据。

难点：既要保证交互的实时性（几百毫秒接话、自然打断），
又要能深度思考。

自主从经验中学习

交互中积累的经验，如何被存储与复用。

难点：不靠每次重训，而是从一次次交互中持续学习、
沉淀知识。

THESIS

一个判断：交互就是流式事件处理

把「一轮接一轮」的交互，变成
流式事件处理 —— 这与 Flink 的设计思想同源。

Request / Response
= 微批 (microbatch)
一问一答，汇齐后计算

实时交互
= 真正的流 (streaming)
逐事件处理，event-time

接口设计得当
现有模型无需重训
瓶颈在数据的进出方式与表示形式

ROADMAP

本报告结构：一个 Agent，就是一个 Flink 作业

	Agent 的环节	对应的 Flink 概念	本报告的工作
感知	世界 → 模型	Source + event-time + 自定义序列化	**AOI 观测接口** Sema 语义传输
认知	实时 + 深度思考	批流一体 流=低延迟 / 批=高吞吐	**Interactive ReAct** Latent Bridge
记忆	经验存储与复用	有状态流处理 state backend + checkpoint	**UserAsCode · Engram** PreAct

PART 1 · 感知 / PERCEPTION

世界进入模型 = Flink 的 Source

让 Agent 像人一样实时地听与看，首要问题是世界数据如何进入模型——本质是数据源与序列化问题。

① 无法捕捉动态

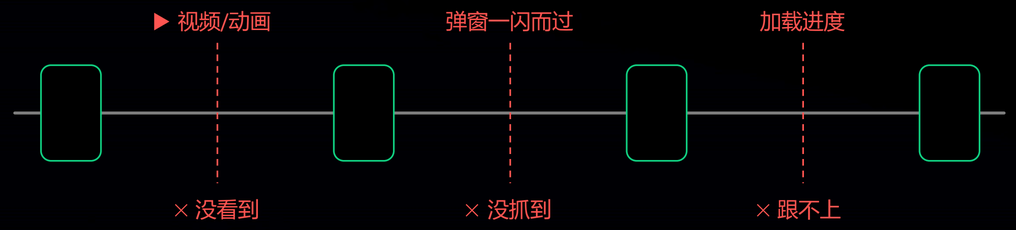
每 3–5 秒一张静态截图，截图之间发生的变化全部丢失。

② 缺乏听觉

无任何音频输入：会议发言、提示音、告警声均无法获取。

VideoWebArena 依赖视频的任务上最好模型仅 13.3%，人类 73.9%。

Agent 每3-5 秒看一眼屏幕；两眼之间的一切都被错过



完全没有音频输入：会议、提示音、报警都听不到

本节结构 感知含两层接口：观测接口（AOI，决定观测什么）与传输接口（Sema，降低传输开销），分别对应数据源与序列化。

PART 1 · 感知 / PERCEPTION

AOI · 把观测做成「事件触发」

核心：观测（连续、自适应）与动作（离散）解耦

- 三个 gated 组件——看 / 听 / 记：屏幕、声音无变化时关闭，静态页面零额外开销。
- Flink 类比：非定时轮询，而是变化触发（CEP / 动态窗口），仅在事件发生时付出代价。

反直觉的发现 让模型把当前画面写成一句话存入上下文，是单项增益最大的组件（+18pp）。

+17~48pp
动态任务 vs 纯截图

82%
最强 (Claude)，静态不降

12 / 12
音频任务全部完成

与记忆的联系 将易逝的视觉观测转写为持久文本，等价于为模型构建短期记忆；持久化记忆将在第三部分讨论。

PART 1 · 感知 / PERCEPTION

Sema · 接收方是模型，不是人

arXiv:2604.20940

下游消费者是模型时，两个假设失效

- 不需要感知级重建 —— 模型只要 task-relevant 的离散语义 token。
- 模型事件驱动、无物理时钟，对投递抖动不敏感。

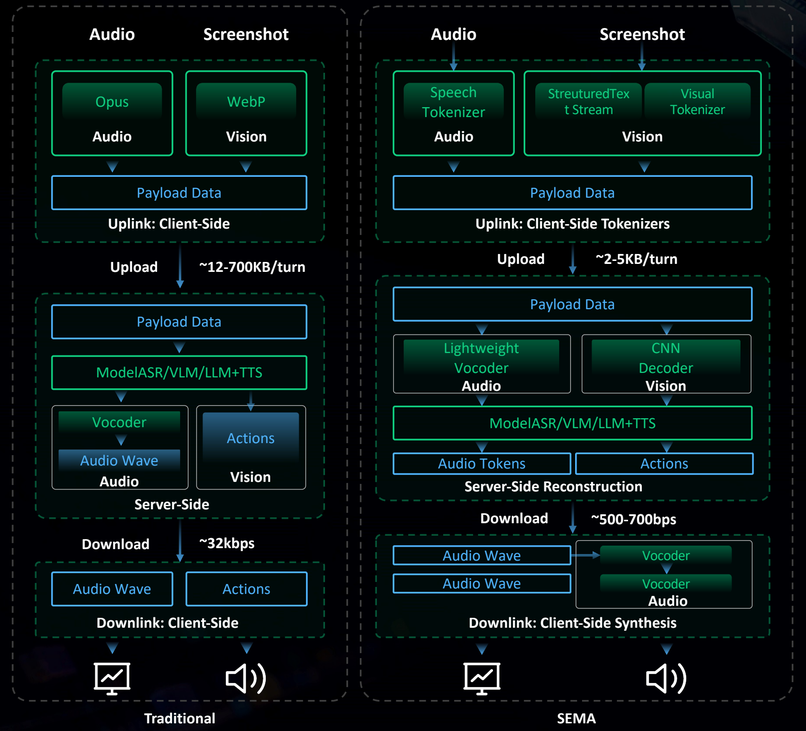
Shannon–Weaver 传输目标由信号保真 (Level A) 转向语义保真 (Level B)。

11–269×
视觉上行压缩

48.6×
语音上行压缩

≈ 无损
端到端任务精度

原则的普适性 「接收方是模型」在认知部分同样适用：快、慢模型之间的通道亦只需传输压缩的语义表示。



PART 2 · 认知 / COGNITION

实时与智能，近乎正交

感知把世界送进了模型；现在模型要在实时约束下，既快又深地思考。

实时但推理浅

- 实时交互每 ~200ms 需判断一次「是否发言、发言内容」。
- omni / 端到端语音可实时，但推理能力偏弱。

推理强但不实时

- SOTA 大模型单次响应往往需 300–500ms 以上。
- 传统 Observe-Think-Act：待输入完整后再长时间推理、最后作答。

解决思路：快慢分离 前台小模型负责快速响应，后台 SOTA 慢模型负责深度推理。

PART 2 · 认知 / COGNITION

快慢分离，就是批流一体

感知把世界送进了模型；现在模型要在实时约束下，既快又深地思考。

前台 · 流处理

- 小模型，逐事件、低延迟，约 200ms 维持话轮。
- 核心是 orchestration：决定哪些即时作答、哪些交由后台。

后台 · 批处理

- 直接采用 SOTA 模型，保证推理上限与吞吐。
- 后台执行深度规划，不阻塞前台交互。

Flink 类比 同一套语义下，流求低延迟、批求高吞吐——快慢两条 path 正是批流一体。

以动作游戏为例 仅有快系统则缺乏长期规划；仅有慢系统则无法及时反应、开局即败。

PART 2 · 认知 / COGNITION

Interactive ReAct · 边听边想（Think While Listening）

用户	候选人：我之前负责构建分布式系统……
思考	分布式系统——需评估深度，先搜一下……
工具	web_search("候选人 分布式系统 项目") 异步
用户	……我们用 Kafka+Redis 扛住 1000万 req/s 边说边查
思考	Kafka+Redis 适合高吞吐，继续听……
工具	GitHub：3 个开源项目，共 2000+ stars 工具结果
思考	结果印证经验！结合候选人所说……
助手	这个规模很厉害！说说最棘手的扩展挑战？ <0.5s

为什么能边听边想

- LLM 生成 ~200 token/s，远快于人说话 ~5–10 token/s。
- 观测 = 事件流；动作 = 交错的 token。
- 用说话间隙推理、异步触发工具。

效果 用户话音刚落，答案 / 工具结果往往已就绪。

PART 2 · 认知 / COGNITION

Interactive ReAct · 边说边想（Speak While Thinking）

用户	候选人：我准备好回答技术问题了。
思考	复杂问题，需要仔细组织。
助手	我来问你一道系统设计题。 快 <0.5s
思考	要覆盖扩展性、一致性、延迟…… 深思 ~5s
助手	设想你在构建一个全球 CDN。
思考	继续——点出缓存失效的难点……
助手	内容更新时，如何在 100+ 边缘节点做缓存失效?

为什么能边说边想

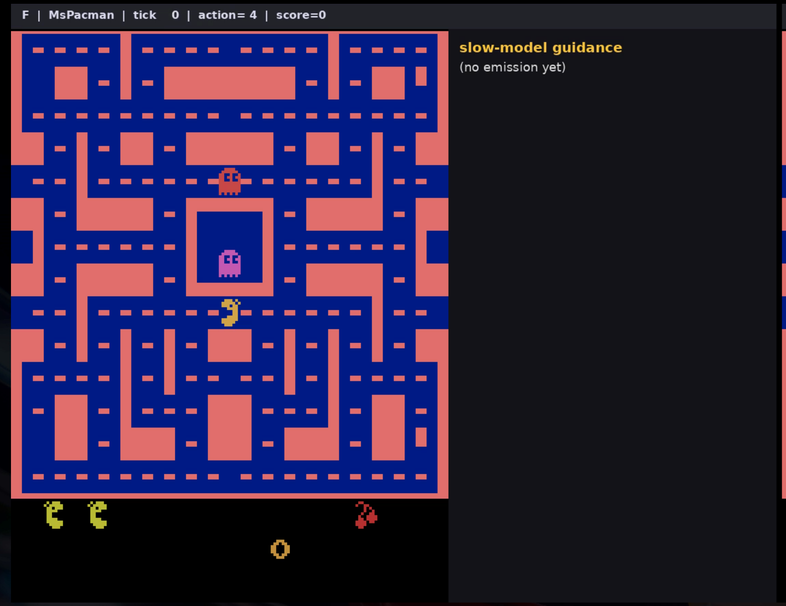
- Fast (<0.5s)：先出声占住话轮。
- Slow (~5s)：后台深思，顺势接续。
- 持续边想边说，问题逐句自然展开。

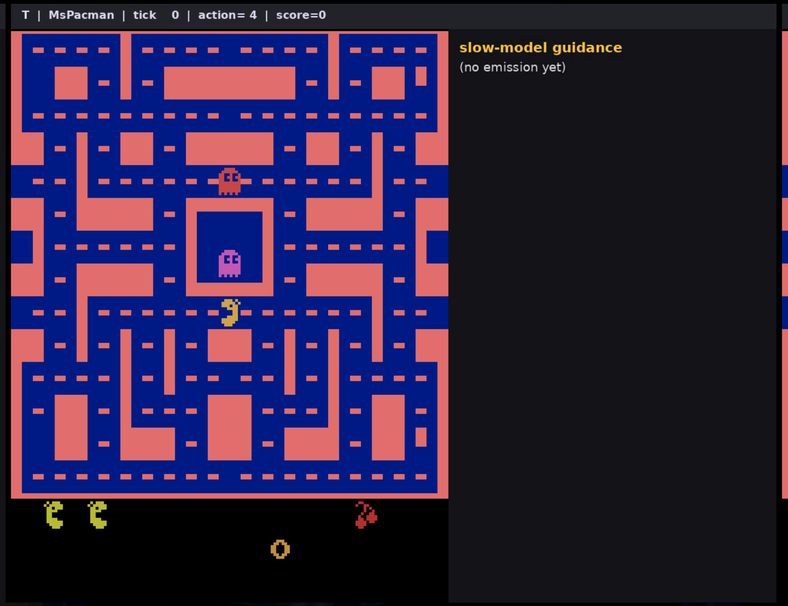
自然打断 说话中收到 interrupt 即停止、回到处理循环；首响应时间（TTFR）较轮流式降低约 67%。

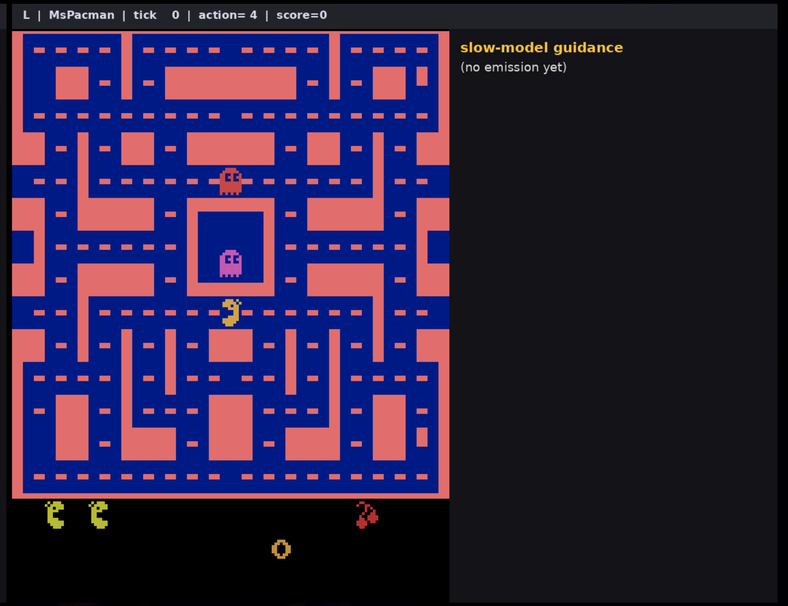
PART 2 · 认知 / COGNITION

Latent Bridge · 快慢之间用什么通道?

arXiv:2606.24470







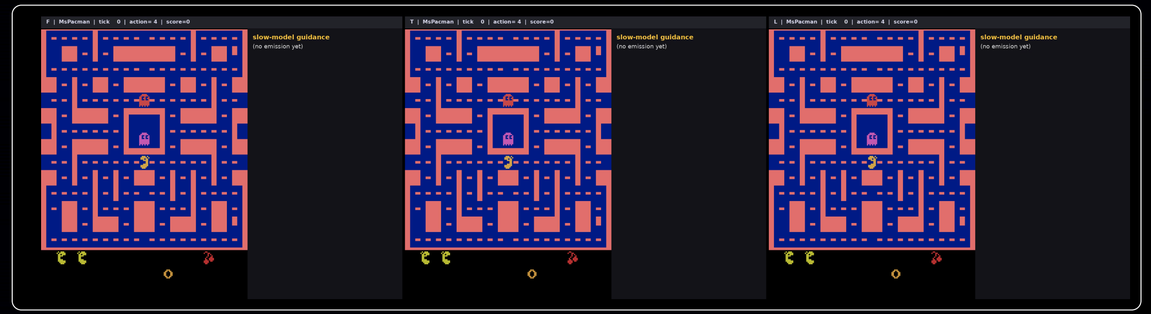
PART 2 · 认知 / COGNITION

Latent Bridge · 快慢之间用什么通道?

arXiv:2606.24470

既要快反射、又要长程规划的任务

- Ms. Pac-Man：快慢分离可视、可客观评分。
- 比较三种「快慢之间的通道」：
- 仅快模型 (F) / 文本通道 (T) / 连续 latent 通道 (L)。



“快慢之间的通道”就在这里：两个冻结模型都不变，只换中间这一条

慢模型（大）

长期规划
每隔K 帧才跑一次

通道（慢 快）

F：没有通道，快模型独自决策（只会被动躲）

T：文本通道 — 慢模型写一句话给快模型

L：连续 latent 通道——慢模型直接传一个隐向量

免去“压成文字"的损失，信息更密、更快

快模型（小）

快反射
每一帧都输出动作

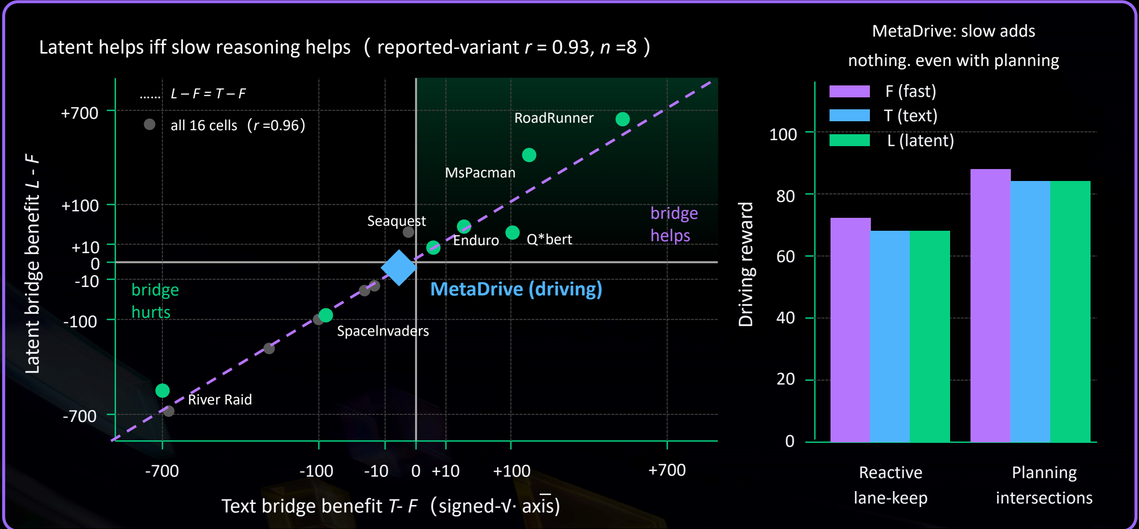
三种配置只差这一条通道，其余完全相同 得分L 628>T408>F

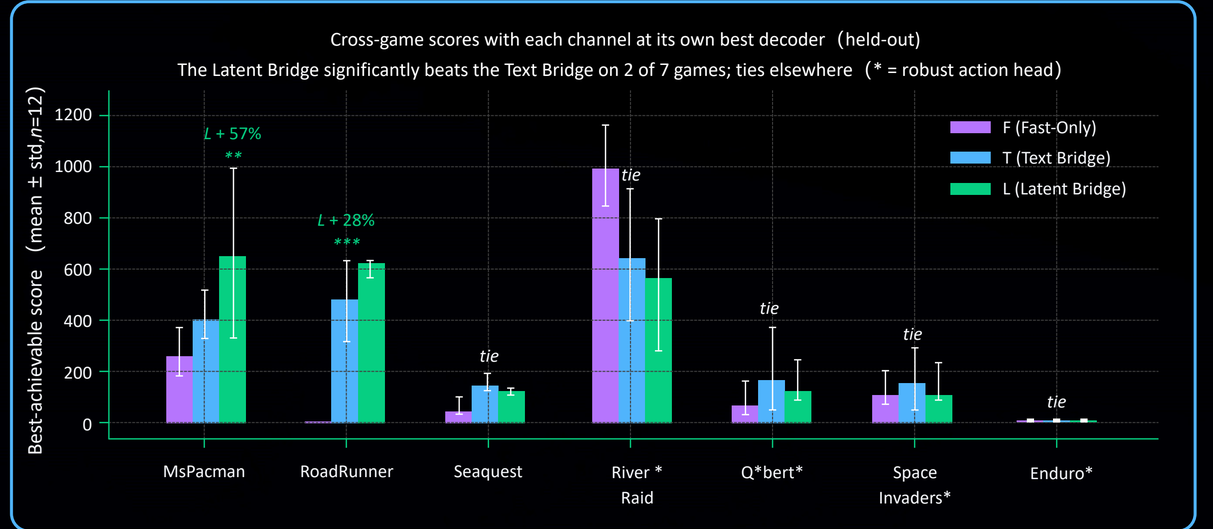
PART 2 · 认知 / COGNITION

Latent Bridge · 关键不是带宽，是慢思考是否有用

冻结两端、只学中间的桥（33M 参数），让通道成为唯一变量；快 MiniCPM-o 4.5（9B）/ 慢 Qwen3-VL-8B-Thinking。

arXiv:2606.24470





结论 latent 桥从不显著差于文本桥，并在 7 个游戏中的 2 个显著表现更好（Ms. Pac-Man +57%、RoadRunner +28%）

PART 3 · 记忆 / MEMORY

记忆的版图：经验存在哪、怎么取回

记忆 = Flink 的核心能力： 有状态流处理（state backend + checkpoint）——从交互中提取知识、存储并复用。

同一问题，先按「记什么」分两类——**声明式（用户是谁）** 与 **过程性（怎么做一件事）**。

声明式记忆 · 用户是谁 — 两种实现：外部代码 vs 模型参数

① UserAsCode + Programmable KV
≈ 外部结构化存储

事实 → 类型化代码 + 可执行约束；解释器在事实上计算
利用 KV Cache 的可组合性，缓存用户记忆和 Skills

② User as Engram
≈ 模型参数内

事实 → 写入模型参数的稀疏槽位；被模型内化、零上下文

过程性记忆 · 怎么做（技能）

③ PreAct
≈ 缓存

成功轨迹 → 可执行状态机；命中即跑、未命中回退

共同点 三者均在「自然语言记忆」基线之上提供更强结构，并将内容与计算 / 推理分离。

PART 3 · 记忆 / MEMORY

① UserAsCode · 把用户记忆表示成代码

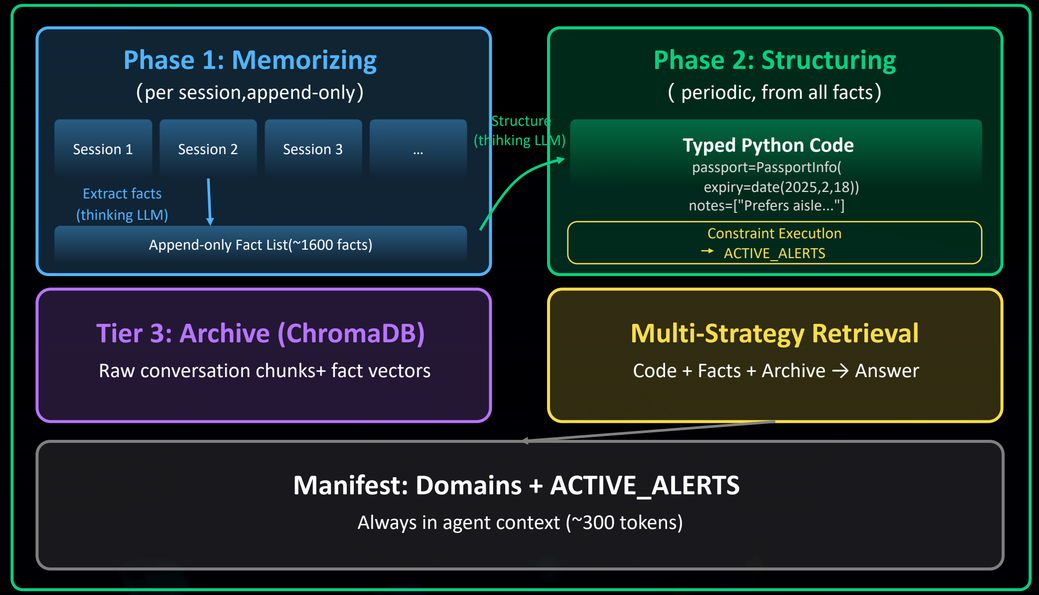
用户记忆 = 一组有类型的数据；两阶段流水线把对话变成可运行的代码。

arXiv:2606.16707

```python
# A user isn't a "bag of facts" — it's a typed Python object.
# Phase 2 of the pipeline structures each session into state:
user = UserProfile(
    name="Jessica Thompson",
    home_city="San Francisco",
    passport=Passport(number="AB1234567", expiry_date=date(2025, 2, 18)),
    trips=[Trip("Tokyo",       date(2025, 1, 15), international=True),
           Trip("Mexico City", date(2025, 3, 10), international=True),
           Trip("Portland",    date(2025, 4, 22), international=False)],
)
```

两阶段

① append-only 事实日志（只追加） ② 周期性 LLM 结构化为类型化代码（= checkpoint）。



① UserAsCode · 可执行约束（进阶）

bag-of-facts 擅长召回，但无法计算与约束

· 冲突消解 / 汇总计算 / 约束校验——均需在记录之上进行计算与判断。

```
# Aggregation is one line — not a top-k guess that drops records:
>>> sum(1 for t in user.trips if t.international)
2

for trip in trips:
    if not trip.is_international:
        continue
    days_remaining = (passport.expiry_date - trip.departure_date).days
    if days_remaining < 180:            # 6-month validity rule
        alerts.append({
            "severity": "critical", "domain": "travel",
            "message": f"Passport {passport.number} expires "
                f"{passport.expiry_date.isoformat()} -- only "
                f"{days_remaining} days before {trip.destination} "
                f"trip on {trip.departure_date.isoformat()} "
```

解释器确定性地执行

- 跨多条记录做计算与聚合，主动预先算出告警。
- 这是一堆零散事实做不到的。

Flink / DB 类比

append-only 日志 + 周期检查点 = changelog + checkpoint，是数据库系统的经典形态。

PART 3 · 记忆 / MEMORY

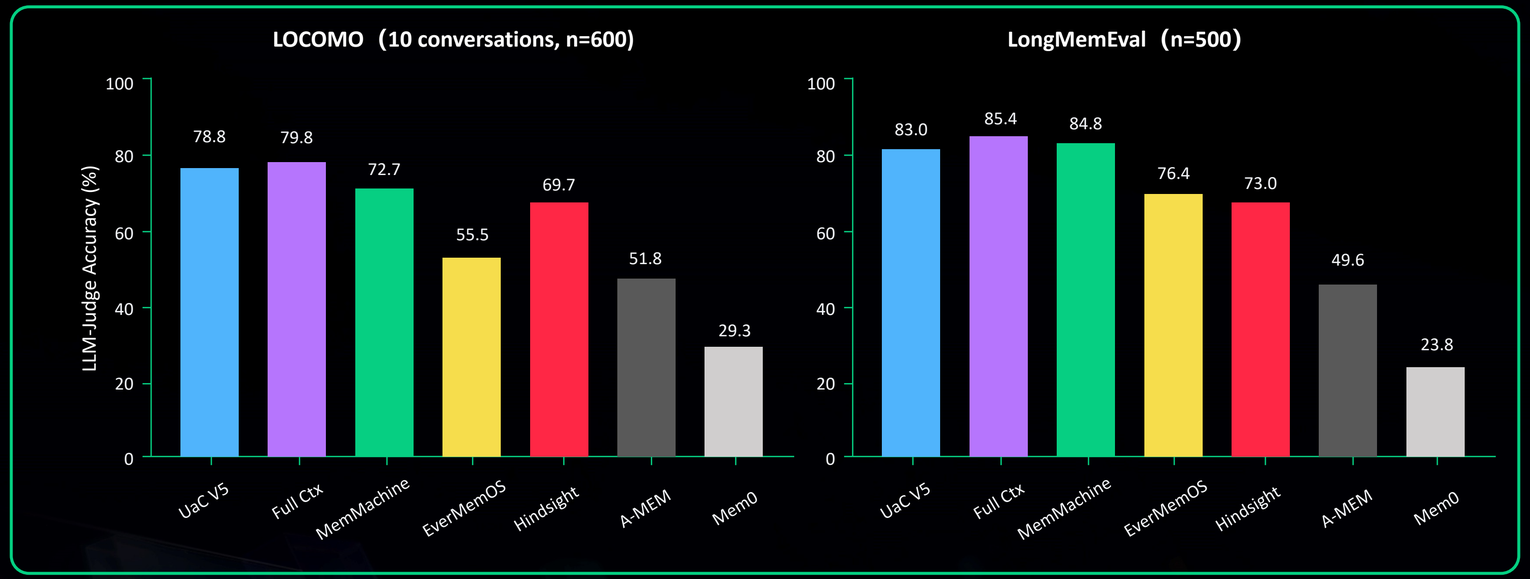
① UserAsCode · 评测

LOCOMO / LongMemEval 上对比检索类记忆系统与 full-context 上界。

78.8%
LOCOMO ≈ 上界 79.8%

99%
聚合推理（检索类 6–43%）

~15×
更省 · 告警检出 100%



PART 3 · 记忆 / MEMORY

① Programmable KV · 把记忆和 Skills 编译进 KV Cache

arXiv:2606.17107

UaC 把记忆编译成代码——但每轮对话都要重新 prefill 这段代码。能否预先编译成 KV Cache，直接拼接？

痛点 · 记忆每轮重读

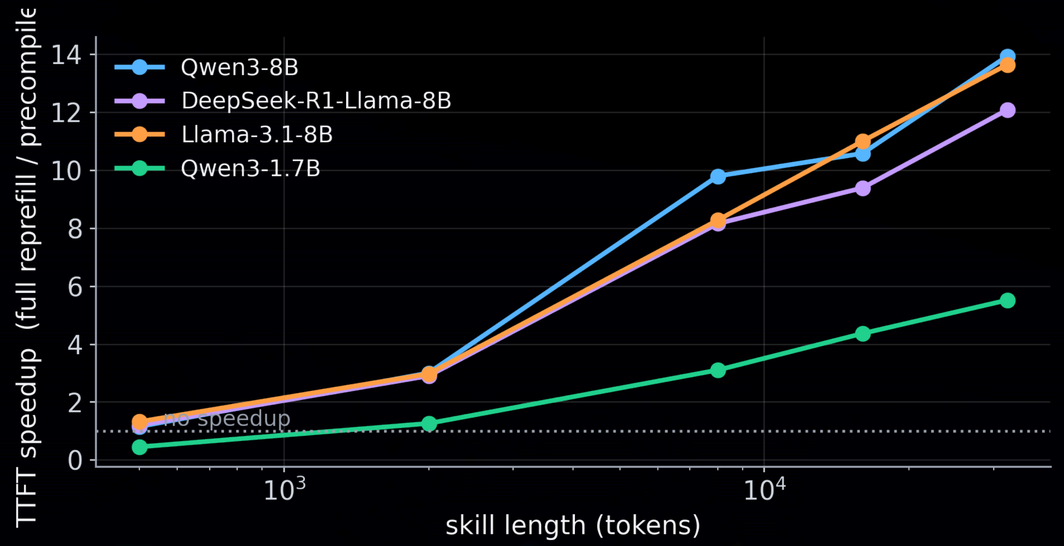
- 代码 / Markdown 记忆每轮重新 prefill，TTFT 随长度 **$O(L^2)$**。
- 前缀缓存只在「逐字符相同」时命中——记忆一变，整段失效。

做法 · 预编译 + RoPE 重定位

- 记忆块单独预编译一次，每轮 **RoPE 重定位**到 query 前拼入。
- **$O(L)$** 而非 $O(L^2)$，无需重新 prefill；行为与全量重算不可区分。

13.9×
更快 TTFT @32k

0.90–0.999
logit 余弦 · 与全量重算等价



预编译技能的 TTFT 加速 vs 技能长度（$O(L^2) \rightarrow O(L)$）

PART 3 · 记忆 / MEMORY

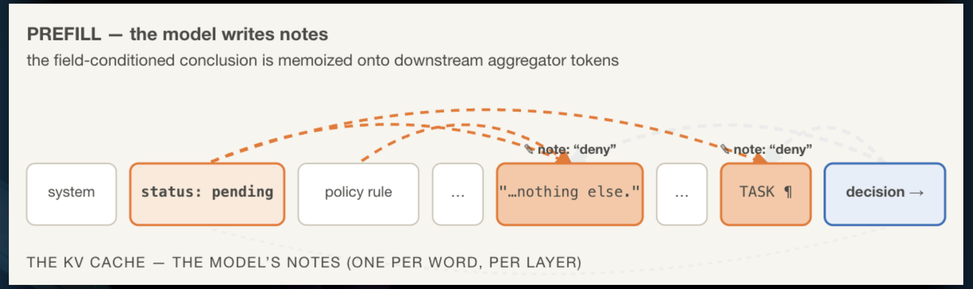
① Programmable KV · 贴便签修改 KV Cache 中的字段

arXiv:2606.17107

记忆里的字段更新（账号状态、时间、额度），不必整段重算——append-only 追加一条修订便签即可。

为何不直接改 KV Cache 里对应 token 的 KV

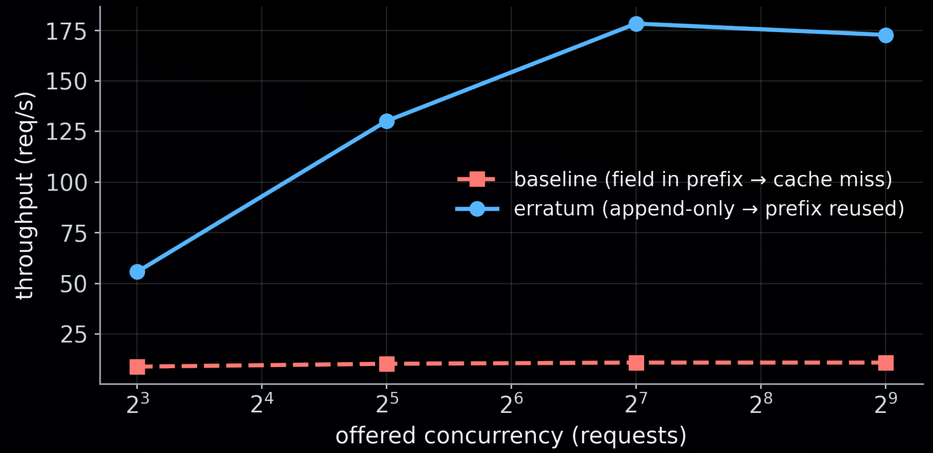
· 如果字段在规则之前，后续规则部分的 KV Cache 会记录基于字段内容的推理结论



便签式修订 · erratum

· 旧笔记留着，末尾追加一条显式「[更新] X 已变为…」。

· 只重算 ~6%；append-only，与前缀缓存对齐。



98.5% vs 1%
前缀缓存命中率

14.5×
吞吐

53x
更低 p90 TTFT

PART 3 · 记忆 / MEMORY

② User as Engram · 把用户事实写进模型参数

arXiv:2606.19172

先理解 Engram（DeepSeek）

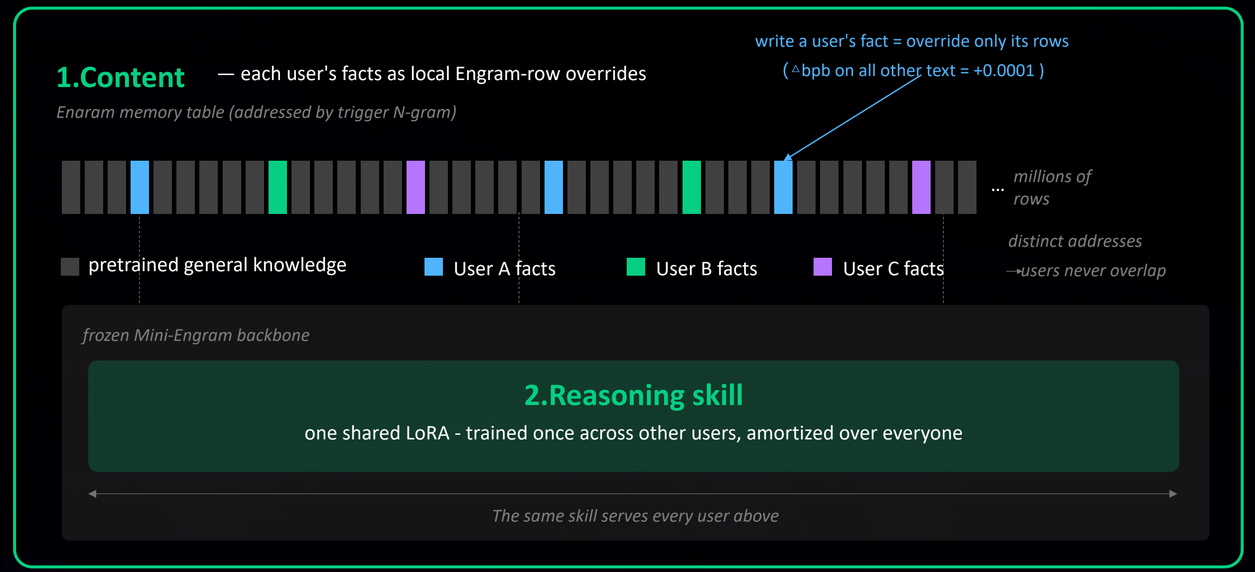
- 模型旁挂一张很大但稀疏的记忆表，作为可寻址的外置参数。
- 训练得到的门控决定何时、把哪条记忆注入模型。

问题：LoRA-as-memory

- per-user LoRA 是全局权重增量，污染无关文本；直接召回近满分，但间接推理下降。

思路：内容 / 技能解耦

- 把一条用户事实写入一个空槽（按用户隔离）；推理技能由共享 adapter 承担。

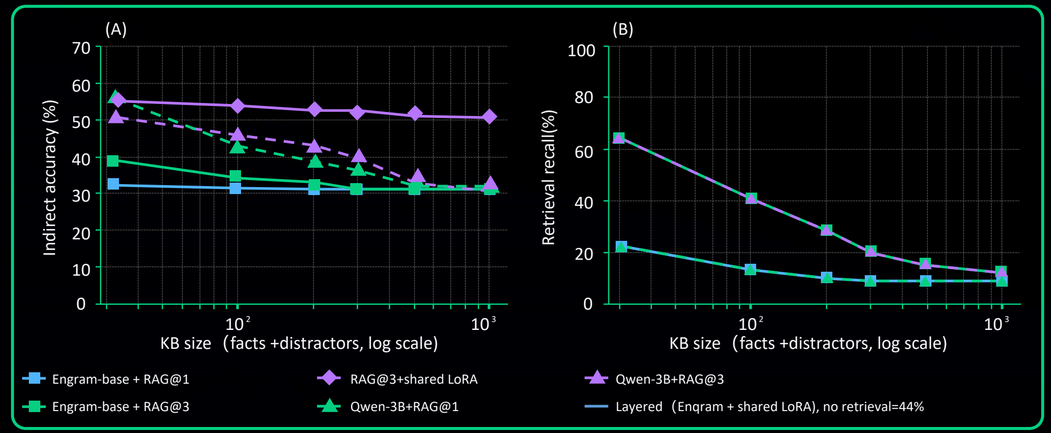


与 UserAsCode 的关系 二者均将内容与计算 / 推理分离（UaC：数据 vs 约束；Engram：事实行 vs 共享推理 adapter），区别仅在事实存于代码还是参数。

② User as Engram · 多租户隔离与评测

一个有寻址与隔离的存储系统

- per-user override 表：按请求加载、用后还原 。
- 零跨用户泄漏（由构造保证）。
- 地址不相交的 override 可交换、可加性叠加 。
- 类似 Stable Diffusion 的 LoRA 即插即用：企业 + 个人片段推理时叠加 。



5.6×
间接推理 vs LoRA

~33,000×
更少的无关文本污染

>100 facts
超过 2.5× 大模型的检索

PART 3 · 记忆 / MEMORY

③ PreAct · 把重复工作编译成缓存

arXiv:2606.17929

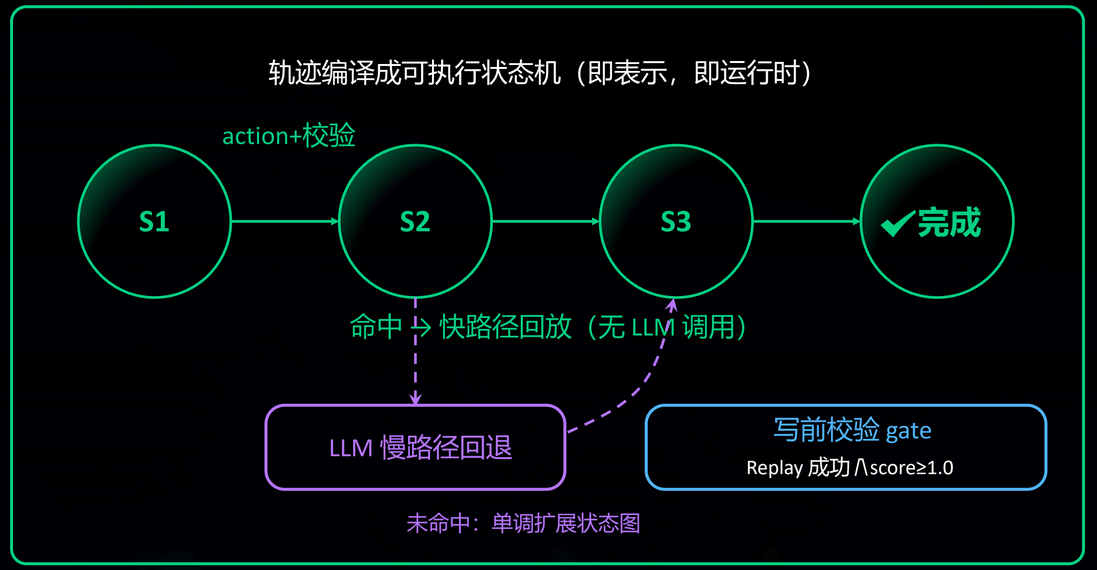
Computer Use 的 Rerun Crisis

- 每个任务都要多轮「看屏幕 → 推理 → 操作」（每步数秒）。
- 做过的任务仍以 O(M×N) 反复重新推导同一套操作序列。

过程性记忆 前两篇是声明式（用户的事实），PreAct 是过程性（怎么做一件事）。

设计：状态机即可执行

- 把成功轨迹编译成形式化状态转移图——既是表示，又是运行时。
- 轻量 XPath 状态校验；编译产物内自带条件分支。



PART 3 · 记忆 / MEMORY

③ PreAct · 把重复工作编译成缓存

任务：新建联系人 Anita Goodall

- 首次执行：agent 逐步「看屏幕 → 推理 → 操作」完成任务（右侧回放）。
- 成功后编译为状态机；再次遇到同类任务直接重放，无 per-step LLM 调用。
- 每步先用 XPath 校验页面，确认无误才执行动作。

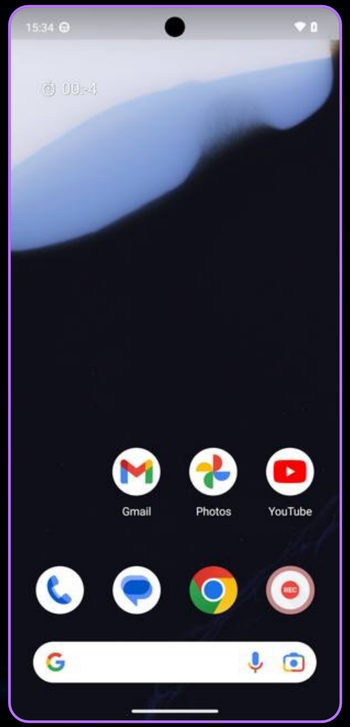
zzzzzzz

```
state permission_dialog
  verify  resource_id=com.android.permissioncontroller:id/permission_allow_button
  do      tap resource_id=com.android.permissioncontroller:id/permission_allow_button  →  contacts_main_screen

state contacts_main_screen
  verify  resource_id=com.google.android.contacts:id/floating_action_button
  do      tap resource_id=com.google.android.contacts:id/floating_action_button  →  new_contact_form_empty

state new_contact_form_empty
  verify  hint=First name&&class=android.widget.EditText
  do      type $first_name  →  new_contact_first_name_entered

state new_contact_first_name_entered
  verify  hint=Last name&&class=android.widget.EditText
  do      type $last_name  →  new_contact_last_name_entered
```



PART 3 · 记忆 / MEMORY

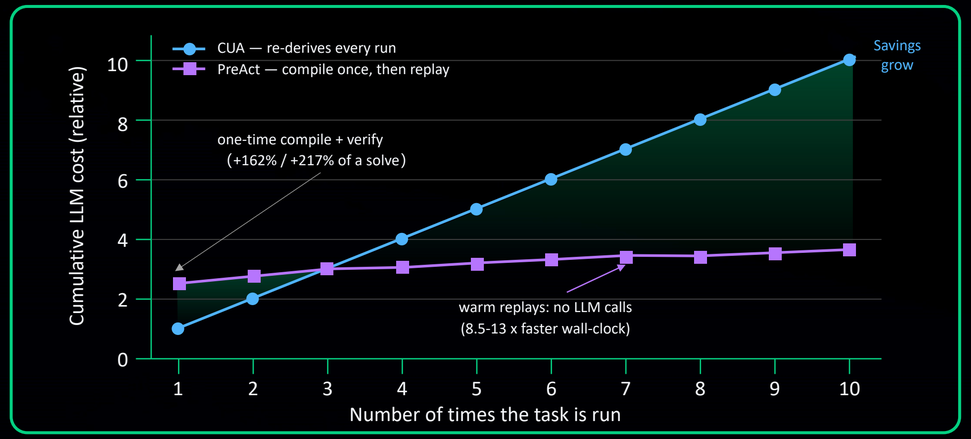
③ PreAct · 命中走快路径，未命中回退

命中 → 编译好的快路径（确定性、无 LLM 调用）

未命中 → LLM 慢路径回退（对应认知部分的快慢机制）

质量保证

- verify-before-store：仅 replay 成功且 score≥1.0 才入库。
- 单调精化：每次回退在已有状态图上扩展，而非重建。
- UI 变化时单周期精化，保持 LLM 级适应性。



8.5–13×
重放加速（无 per-step LLM）

73.3%
AndroidWorld（Gemini=Claude）

+1.75~2.6
写前校验净增任务/基准

与认知的一致性 命中走快路径、未命中回退慢路径，与认知部分的快 / 慢双路径机制一致。

CONCLUSION

回到 Agent 的两朵云

流式与环境交互

→ 感知 (Source) + 认知 (批流一体)

自主从经验中学习

→ 记忆 (有状态流处理)

决定系统表现的，往往是接口与表示，而非模型本身。

多模态观测 (AOI) • 网络传输 (Sema) • 思维通道 (Latent Bridge)

状态与记忆 (User as Code / Programmable KV / User as Engram / PreAct)

这正是 Flink 的核心命题。

相关论文与交互式演示： 01.me/research

Sema 2604.20940 · Latent Bridge 2606.24470 · UserAsCode 2606.16707 · Programmable KV 2606.17107

User as Engram 2606.19172 · PreAct 2606.17929 · AOI · Interactive ReAct（即将发表）

Flink Forward ASIA

Organized by Alibaba Cloud

THANK YOU

- 谢 谢 观 看 -