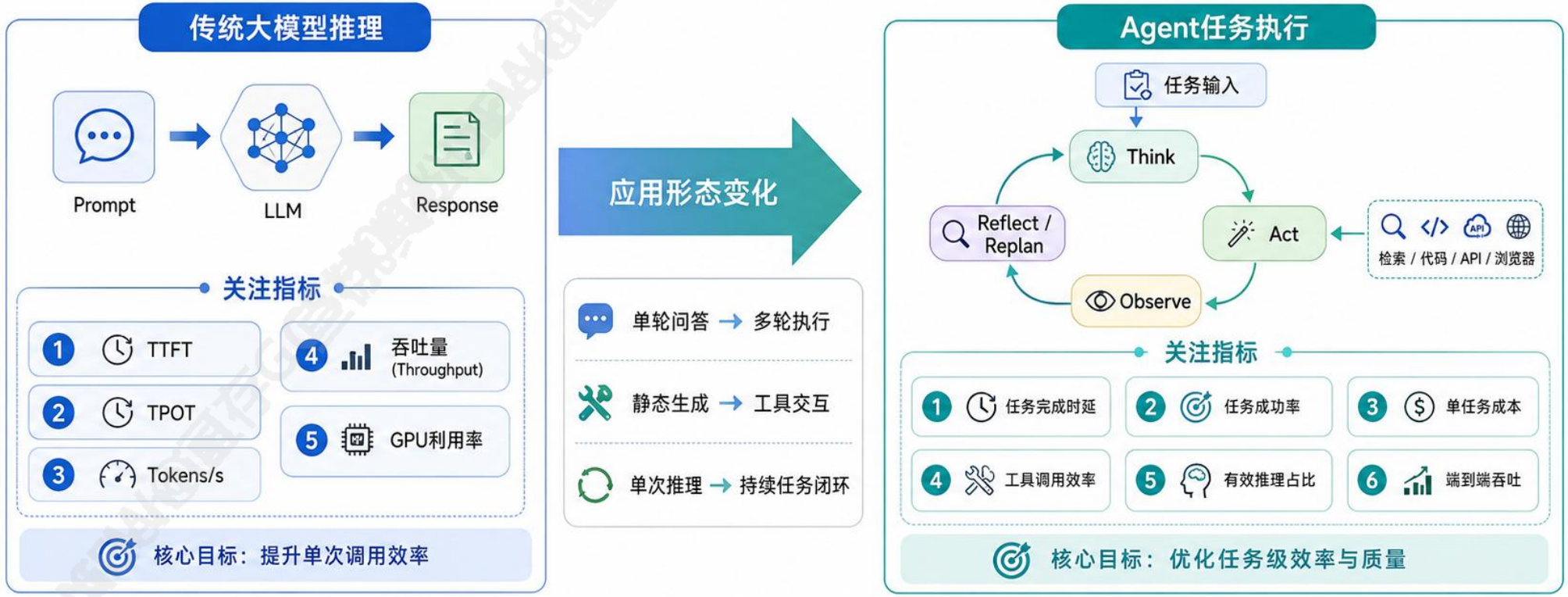
Agent时代的模型—算法—系统协同设计

——从单点优化到端到端智能基础设施

华为 先进计算与存储实验室
林炜哲

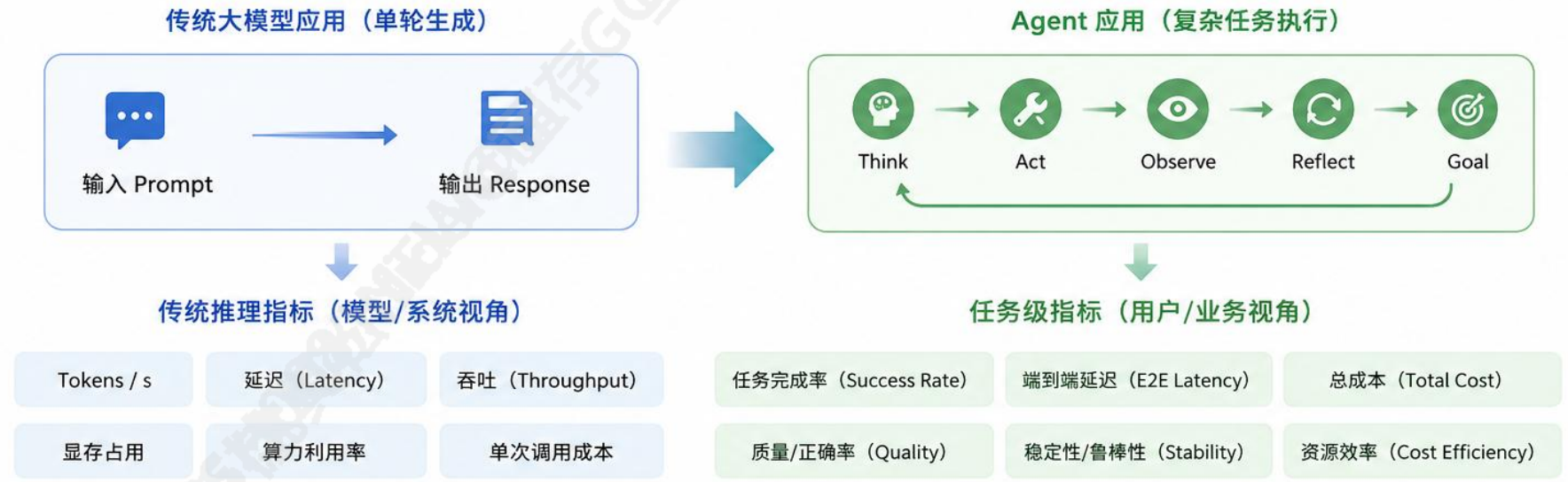
从传统推理指标到任务级指标

优化对象：从单次生成，转向端到端任务完成



单步更快 ≠ 任务更快 | Agent时代需要模型、算法、系统的协同优化

0.2 指标变化：从传统推理指标 → 任务级指标



0.3 单点优化失效：为什么需要协同设计

Table 1: **The Quantization Trap.** Comparison of FP16 baseline versus INT8 quantization on the HLE benchmark. Although quantization improves throughput (TPS), the loss in precision causes a sharp decline in success rate and triggers a 3.5× increase in retries, ultimately inflating the total time-to-solution.

Metric	FP16 (Baseline)	INT8 (Quantized)	Δ Relative	Impact
Throughput (Tokens/s)	42.5	61.6	+45.0%	Faster Inference
Task Success Rate	88.2%	61.7%	-30.0%	Degradation
Breakdown of Latency:				
Avg. Inference Time (s)	45.0	30.0	-33.3%	Step Speedup
Avg. Recovery Time (s)	5.0	55.0	+1000%	**Retry Loop**
Total Time-to-Solution (s)	**50.0**	**85.0**	+70.0%	**Slower End-to-End**

推理速度 ≠ Agent任务的端到端性能

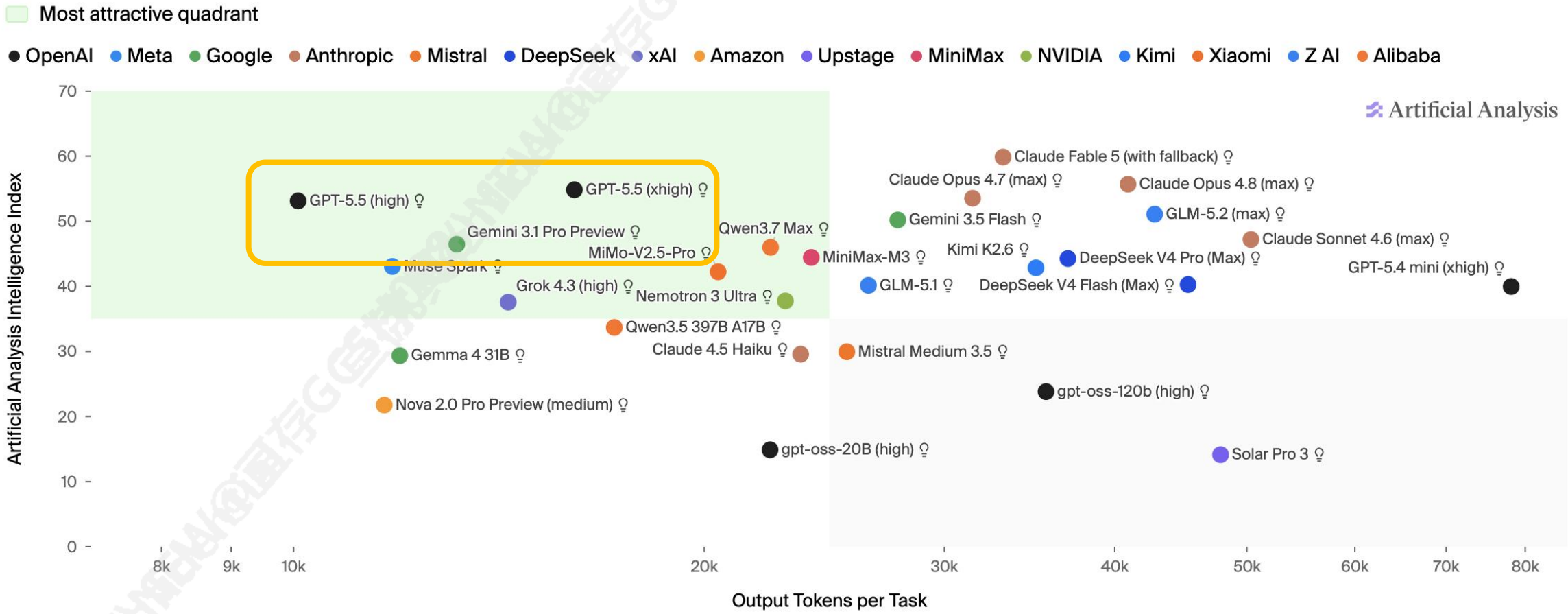
- Agent任务是长链任务，单步的速度虽然增加了，但是如果单步的质量下降，那么会引入更多的Retry loop，更多的轮次成本，最终得到更差的端到端表现
- 举例说明，在HLE任务上对Agent模型进行INT8量化后，虽然Token-by-token的生成速度提升了，**但是最终的端到端时延增加了~70%**

Table 2: **The Summarization Gap.** While summarization reduces the context window size per step, it introduces ambiguity that forces the agent to perform more turns to clarify information. This "Scissor Effect" neutralizes the efficiency gains.

Metric	Full Context	Summarized	Observation
Avg. Tokens per Step	8,500	2,100	4× Reduction
Avg. Turns to Solve	4.0	14.0	3.5× Increase
Total Token Consumption	≈ 34k	≈ 29.4k	Marginal Gain
Context Drift Rate	Low	High	**Cognitive Ambiguity**

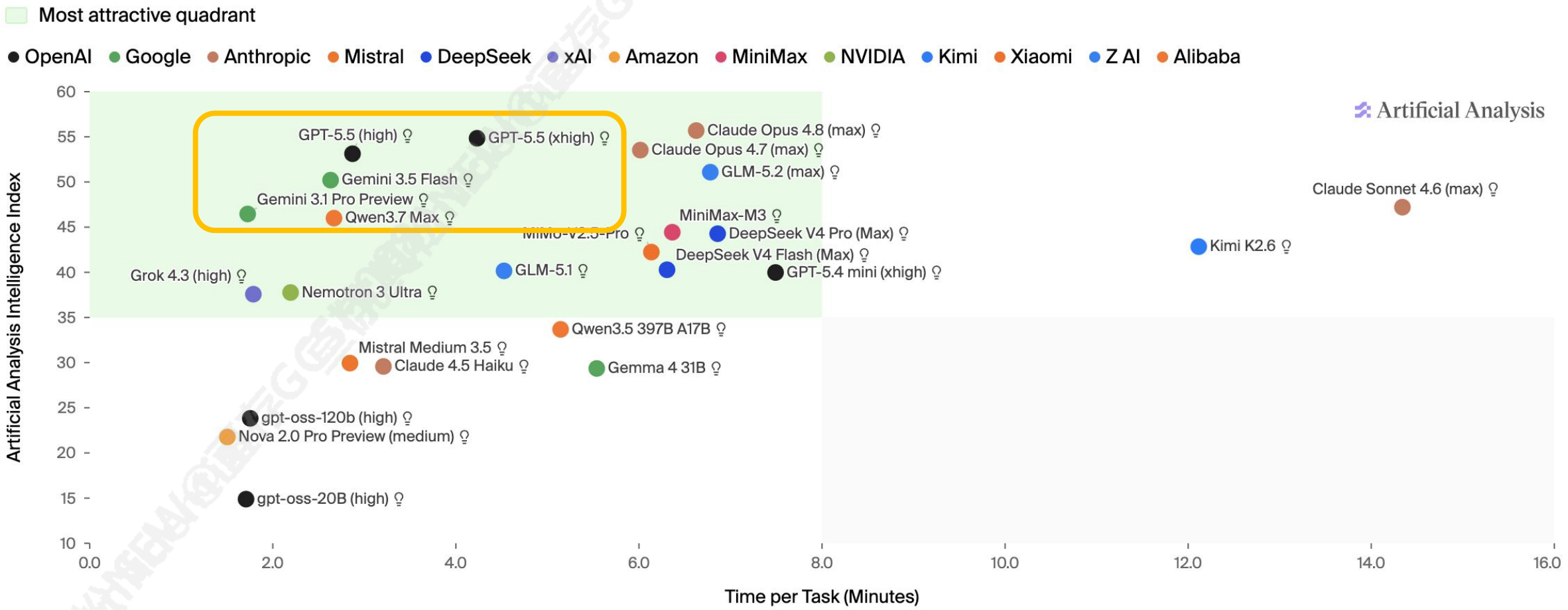
- 在另一个例子中，通过较为简单的上下文压缩手段，虽然减少了任务的上下文长度，提升了推理效率。但是**反向导致了轮次上升3.5倍**，用于重新检索、重新收集丢失的信息，以及重新构建求解路径。

- **信号协同：**模型、工具、推理过程、系统状态向控制器暴露信号
  例如：任务难度、是否卡住、上下文长度、KV命中率、工具等待时间。
- **决策协同：**调度器、路由器、控制器基于这些信号做选择
  例如：用大模型还是小模型，先跑哪个请求，要不要保留KV。
- **目标协同：**不再只优化单个模块指标，而是优化任务完成质量、时延和成本。



➢ 达到相同的任务目标，模型的Token消耗量体现单位智力成本

https://artificialanalysis.ai/evaluations/artificial-analysis-intelligence-index?utm_source=chatgpt.com&eval-token-usage=score-vs-output-tokens-per-task



- 同一水平线的模型，在时延上的排名与在Token消耗量上大相径庭
- 充分体现整系统优化的必要性

https://artificialanalysis.ai/evaluations/artificial-analysis-intelligence-index?utm_source=chatgpt.com&eval-token-usage=score-vs-output-tokens-per-task

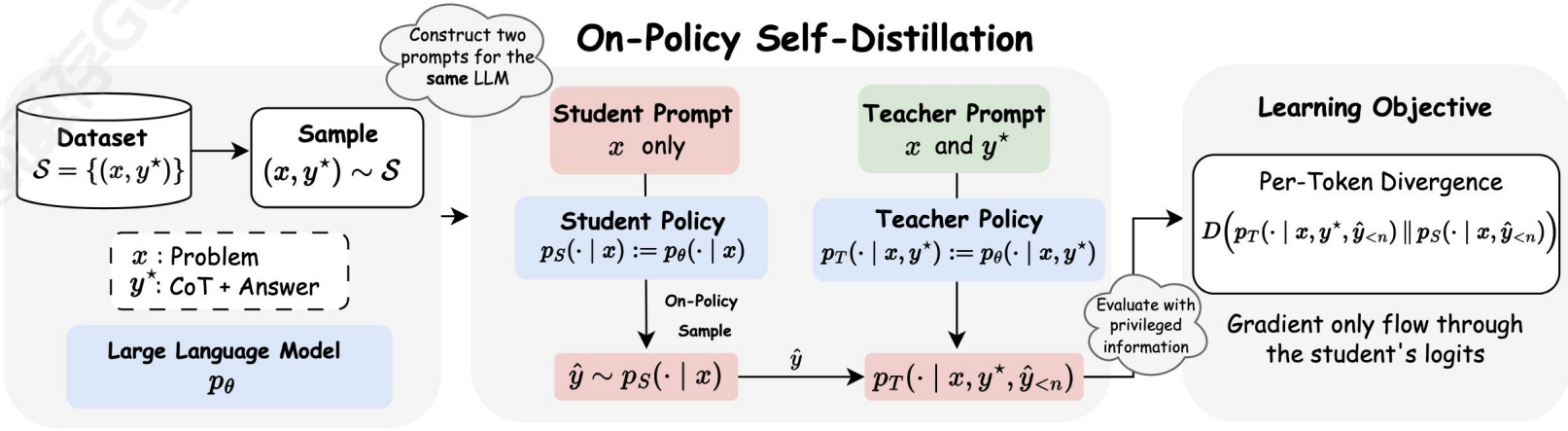
Agent时代的竞争力，不只是更大的模型，而是模型、算法、系统共同形成的端到端基础设施能力。

单点优化解决局部瓶颈，协同设计解决任务级效率

模型层优化

训练侧提升Token效率：让模型“少想、会想、想得值”

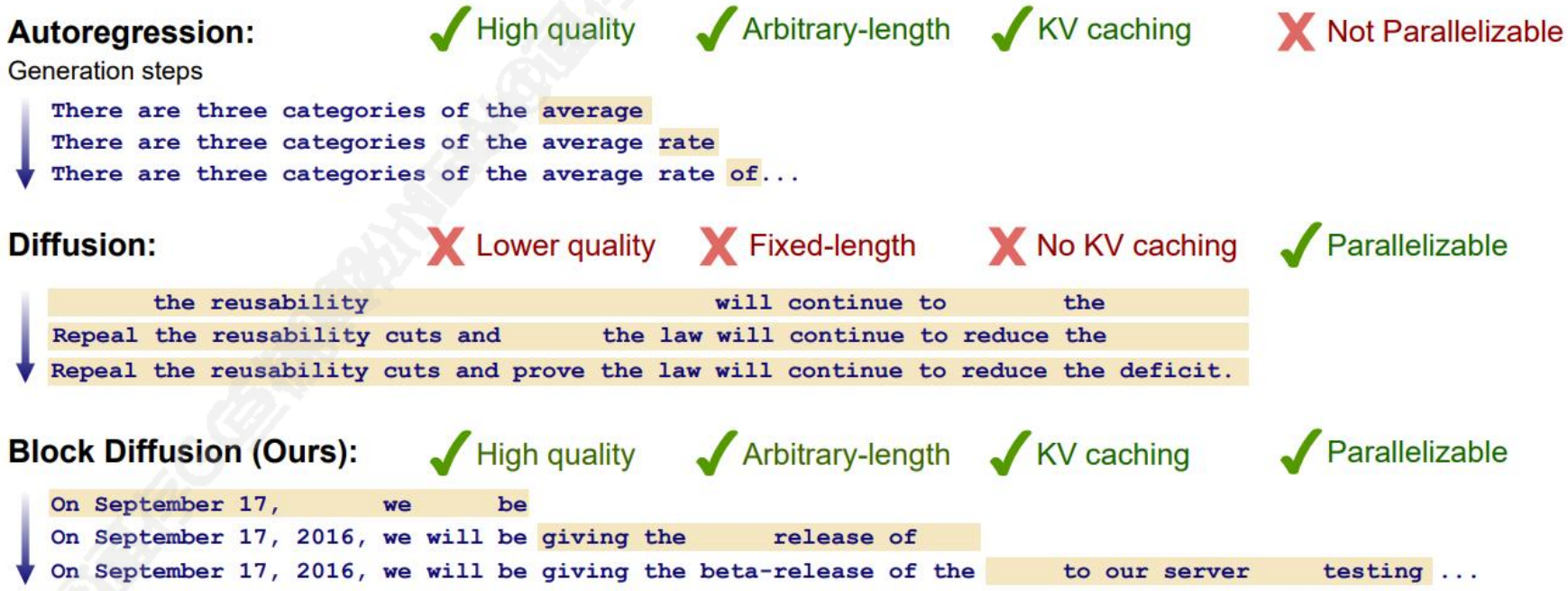
方法方向	解决的问题	核心思想
RL长度感知训练	过度推理、简单问题长思考	把正确性和推理成本共同纳入奖励
OPSD / OPD	冗余步骤，小模型RL Rollout质量低	给Teacher更多的信息，把原本需要多轮展开、长链思考或反复尝试才能完成的推理过程，压缩成更直接、更有效的决策路径
CoT压缩训练	推理链冗长、重复、低信息密度	学会保留关键推理步骤，减少冗余token
蒸馏/专项模型训练	大模型全程调用成本高	把昂贵推理能力迁移到小模型或专项模型



OPSD通过给Teacher更多的信息，指导Student模型用更短的路径达到相同效果

Self-Distilled Reasoner: On-Policy Self-Distillation for Large Language Models https://arxiv.org/abs/2601.18734v3

模型架构的优化：从自回归到扩散生成



核心想法：把序列切成块（block），在“块内”用扩散并行生成，在“块间”用更结构化的依赖（例如半自回归/分块推进）来控制计算与长度扩展。

效果：在保持 dLLM 并行优势的同时，缓解全长扩散的注意力与迭代成本。

Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models, ICLR 2025

整体轮次更少

Method	Accuracy (%) ↑	Tool Calls ↓	Turns Used ↓	Invalid Action Rate ↓
AR Agent	15.5	7.5	14.8	1.9%
DLLM Agent	15.5	6.7	13.0	6.4%
w/o context-clean corruption	14.5	6.8	15.0	6.2%
w/o span-aware attention alignment	14.5	7.1	14.6	7.1%

Table 1: Benchmarking evaluation on a 110-question subset sampled from BROWSECOMP-ZH [33], under a shared 32K context constraint and a maximum of $T_{\max} = 15$ rounds. "Tool Calls" and "Turns Used" are averaged per information seeker over episodes. "Invalid Action Rate" denotes the fraction of episodes containing at least one unparsable action span (e.g., missing delimiters, malformed tool call, or invalid schema). All methods share the same tool API, action parser, and budgets; the only difference is the backbone (AR vs. DLLM).

Seeker调用量减少

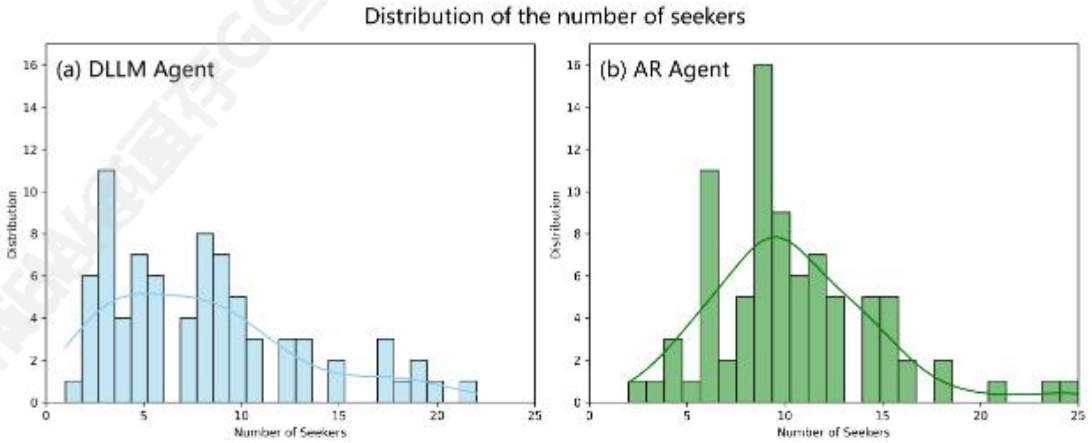


Figure 2: (a) and (b) show the distributions of the number of times the information seeker is invoked by the DLLM agent and the AR agent, respectively. The DLLM agent invokes the seeker significantly fewer times than the AR Agent.

实验结果

- 在可控实验环境下，对比完全相同数据训练的AR Agent和DLLM Agent
- 平均意义上讲，DLLM Agent 和 AR Agent可以达到相同的精度，但是端到端时间降低了 ≈ 30%
- 部分场景中，DLLM Agent 可以比AR Agent 快 8.18倍
- DLLM Agent 和 AR Agent对比，平均工具调用降低10.6%，平均轮次降低12.2%。

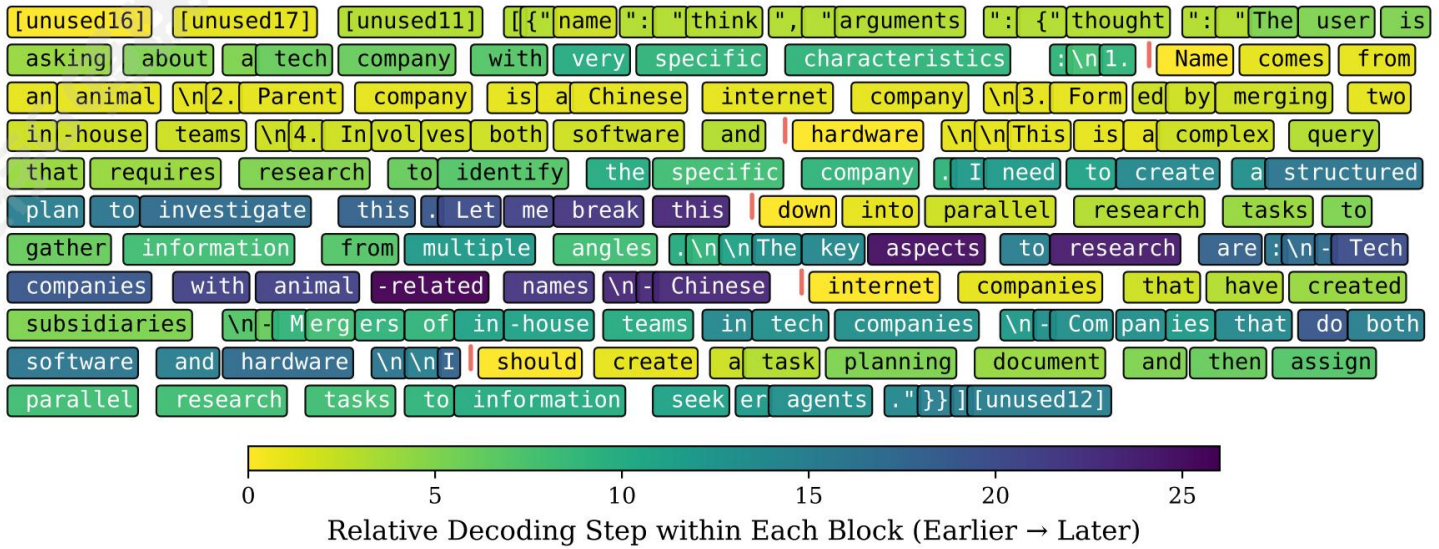
下图可视化了不同的block 在进行并行解码的结果。

给定下列用户请求，DLLM Planner 表现出明显的两阶段特征

- **用户意图理解**：DLLM展现出强大的并行处理能力，在 **1~2 个 diffusion step** 内同时识别多个核心约束
- **任务分解：**DLLM逐步填充关键 Token构建推理框架，展现了比AR模型更灵活的规划与推理过程

任务分解机制 （“逐词生成”到“骨架填充”）：

- 打破 AR模型逐 Token 生成的局限，DLLM 采用“先关键 Token 搭建框架后逐步完善细节”的策略



Heatmap showing the relative decoding order of each token within each block of an example Planner Agent response. Different blocks are separated by red lines.

Agent框架层优化

Agent系统总体架构

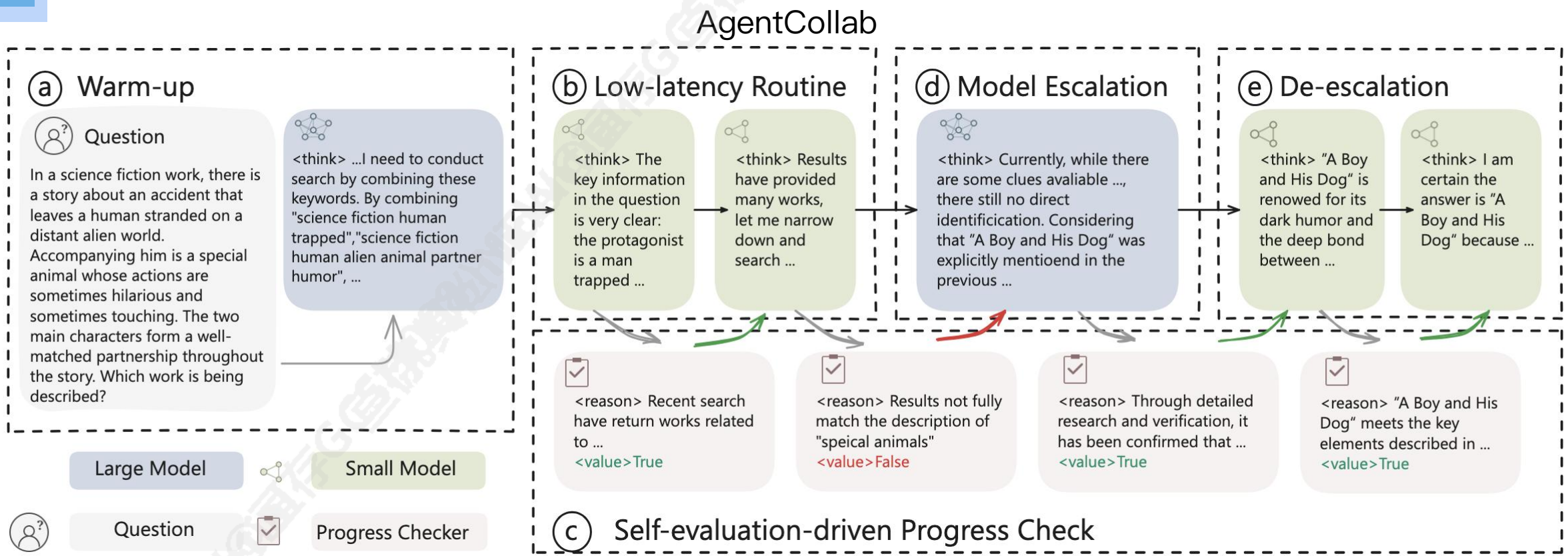
Agent框架层驱动任务执行，连接工具执行与推理系统能力

方案	优点	问题
全程大模型	稳定、推理强	成本高、延迟高
全程小模型	快、便宜	容易陷入局部搜索、错误循环
简单固定路由	易实现	不能感知任务状态变化

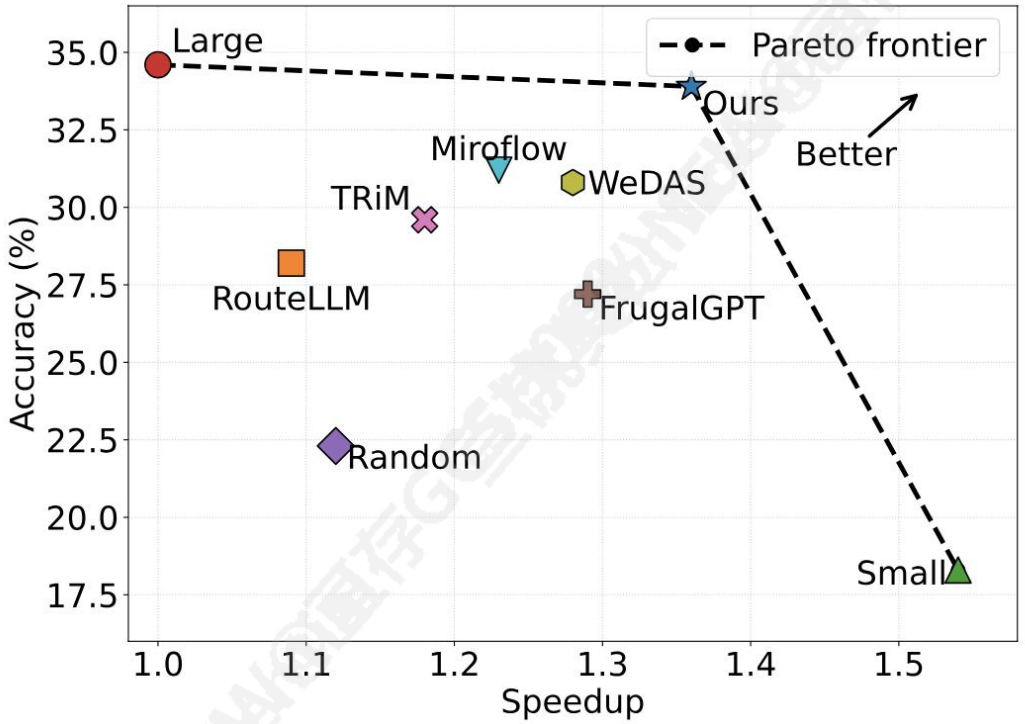
在单轮问答场景中，我们通常只需要在“用大模型”还是“用小模型”之间做一次选择。但在Agent任务中，一个任务会被拆成很多连续步骤，不同步骤对模型能力的需求并不相同。

很多中间步骤其实是低难度、模板化、重复性的。这些步骤如果全部交给大模型，会带来明显的成本和延迟浪费。但另一方面，Agent任务中也会出现关键决策点，例如计划失败、搜索方向错误、工具结果冲突、代码修改不收敛等，这时小模型往往容易陷入重复尝试或错误循环，需要更强模型进行纠偏

所以，大模型和小模型不是简单替代关系，而是互补关系：
小模型适合承担大部分常规执行，大模型适合承担复杂规划、关键判断和失败恢复。

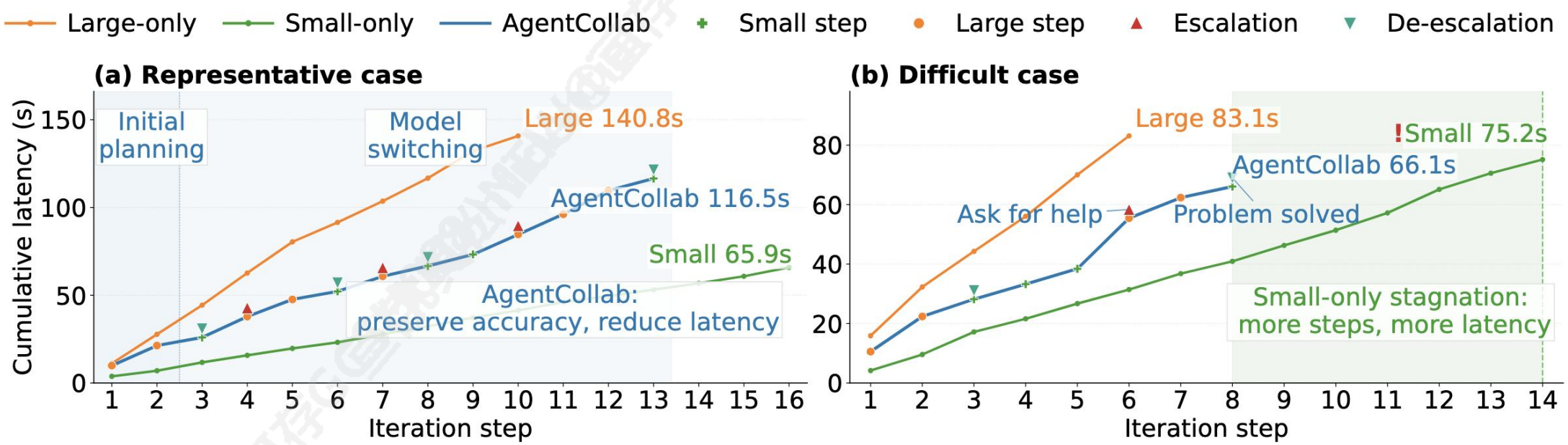


- 大模型与小模型通过自我进展审视，决策是否将控制权进行转移
- 小模型遇到难以解决的问题时主动进行升级，问题解决后大模型主动降级



模型：38B大模型、7B小模型

- 通过大小模型协作，在精度–时延帕累托前沿向优势区间前探
- 精度与大模型基本持平
- 性能逼近纯小模型



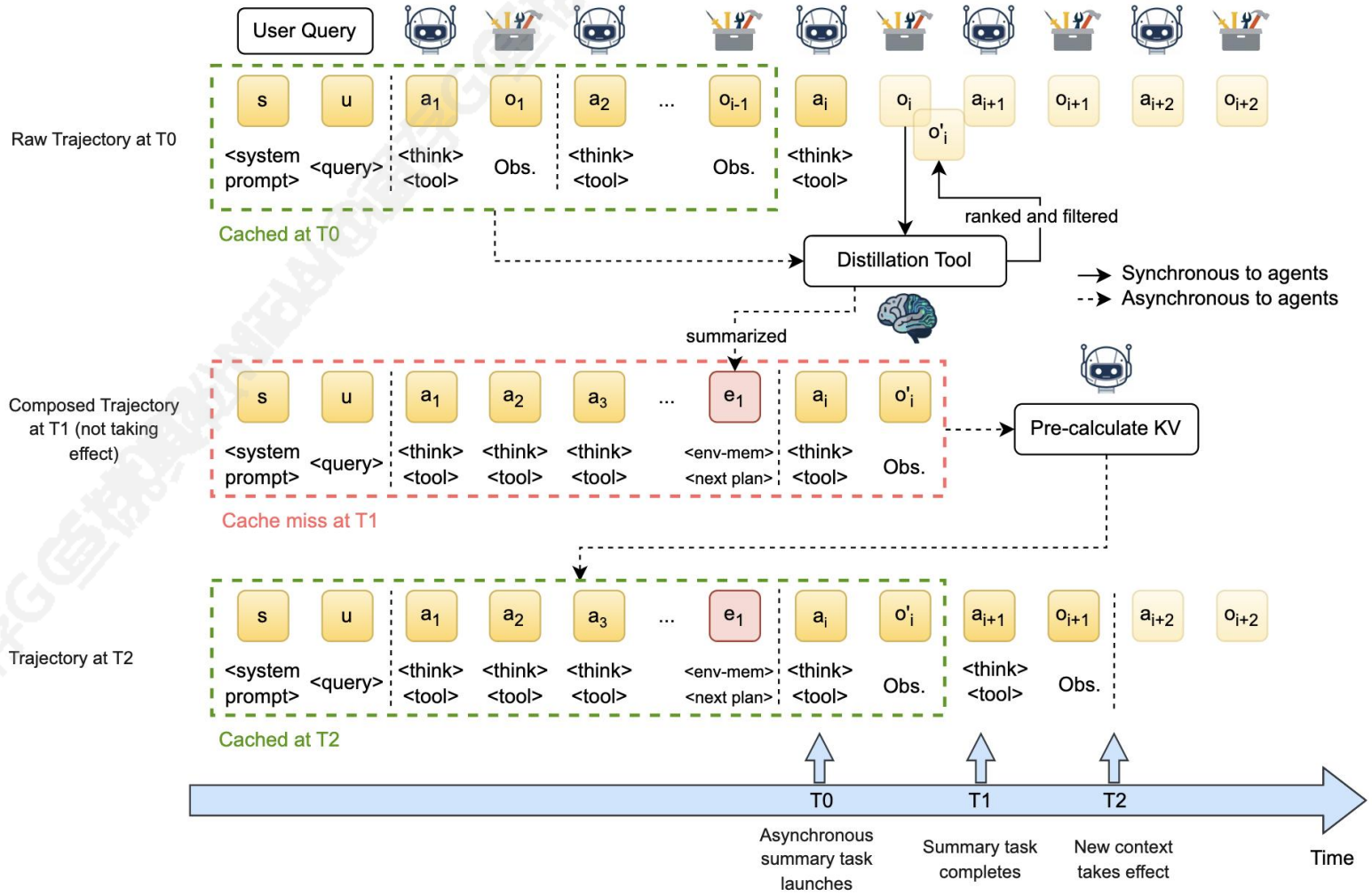
- 典型任务中，AgentCollab在大小模型间反复切换，在保持精度的前提下，时延介于大小模型之间
- 部分困难任务中，纯小模型遇到阻塞，用更多步数和时间完成任务，AgentCollab在阻塞点通过主动模型升级，高速解决问题，获得精度和时延双收益

Agent系统总体架构

Agent框架层驱动任务执行，连接工具执行与推理系统能力

核心权衡：语义保真 × Token 压缩 × KV 命中 × HBM 占用

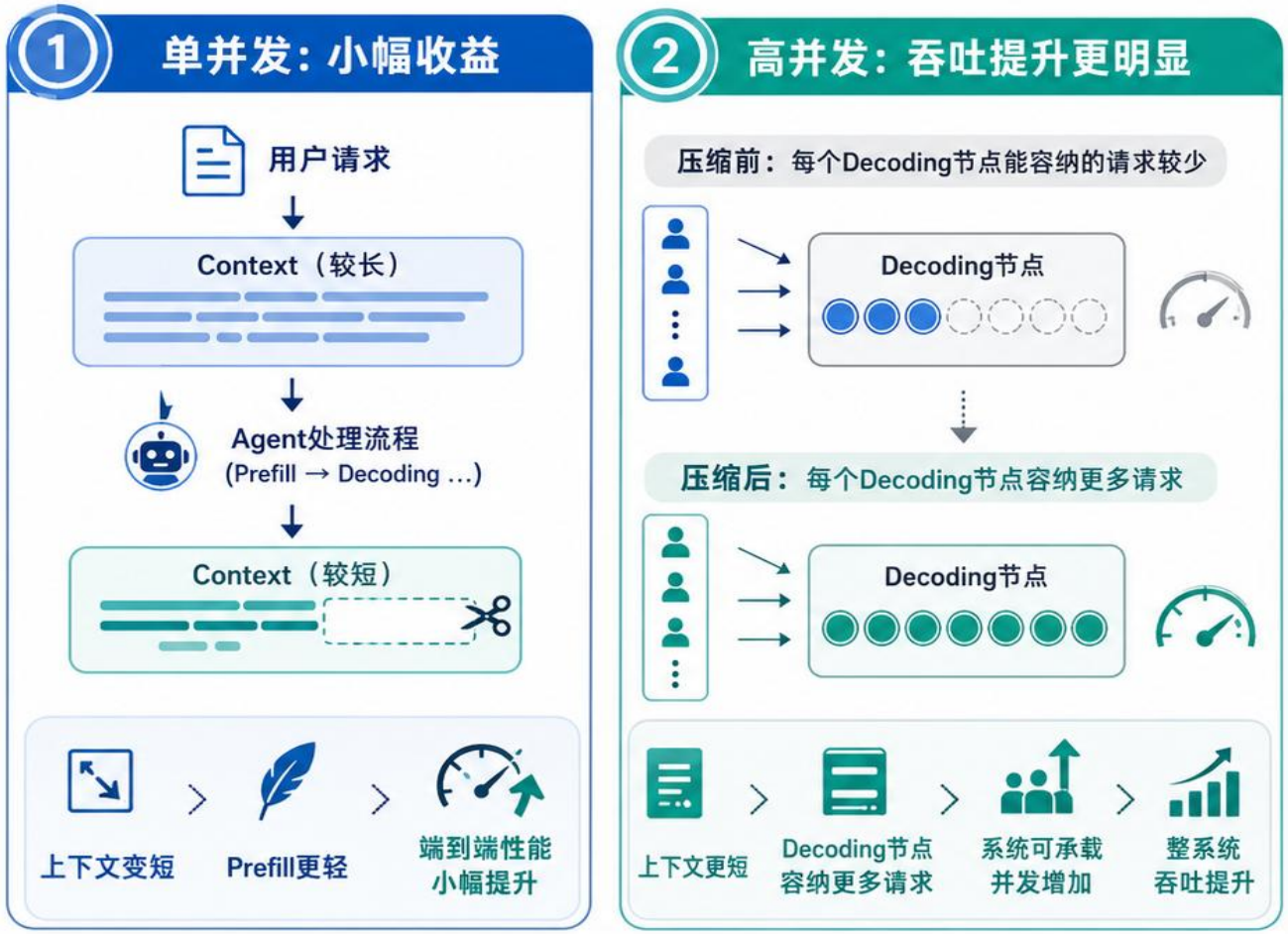
在多轮 Agent 任务中，最优压缩不是一味变短，而是在输入长度、缓存复用和系统时延之间找到平衡。



- AgentCompress保留最近轮次的交互，去除过往的工具返回结果，将思考内容进行压缩，重组上下文
- 异步发送重整后的上下文，预计算Prefill并锁定KV Cache，保证缓存命中

Table 8: Latency reduction and turn count with AgentCompress

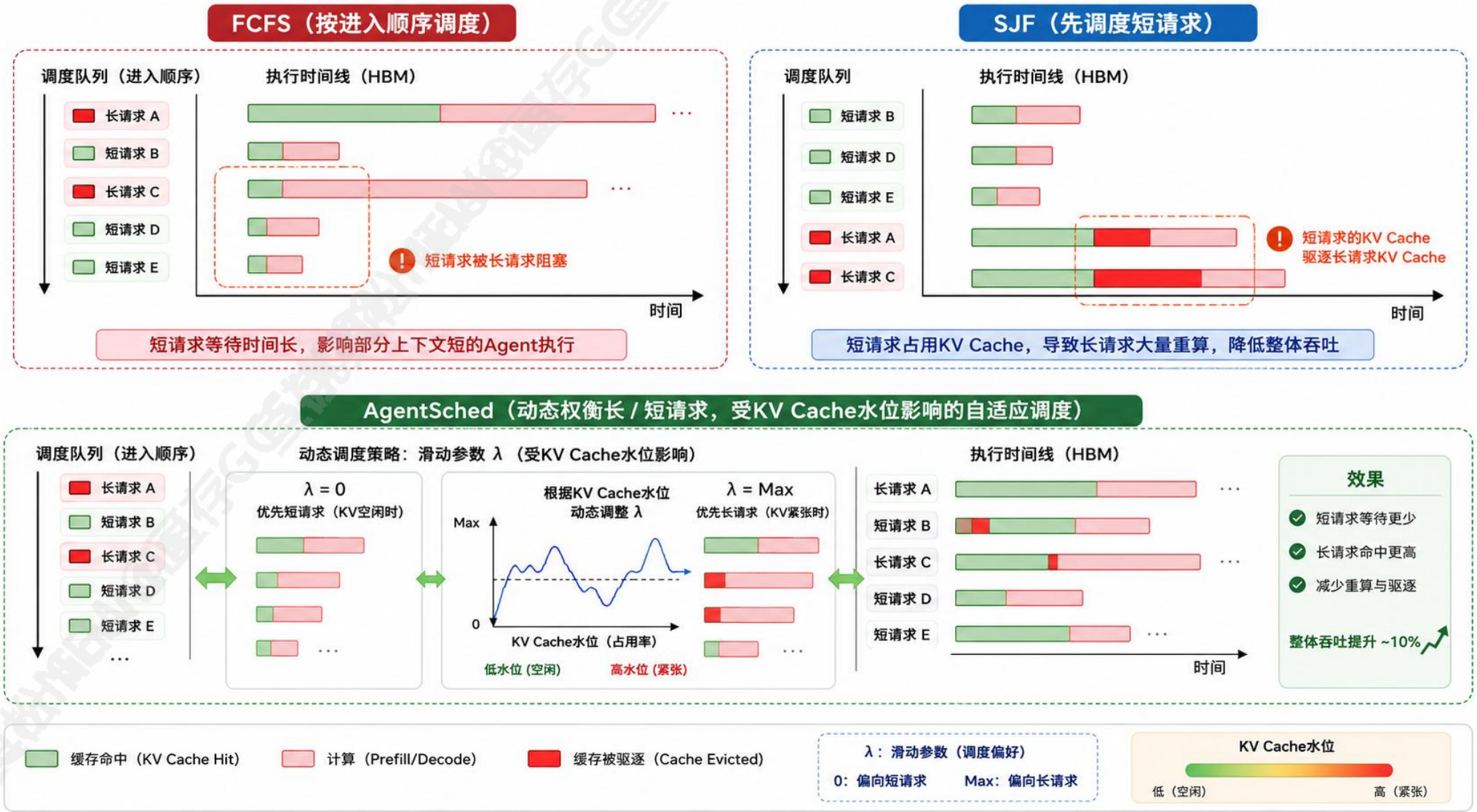
Model Configuration	Latency Reduction	# Turns
Baseline (TP=1, $N_{parallel}$=1)	0%	1.0×
w/ AgentCompress (Search Compression)	-9.8%	1.02×
w/ AgentCompress (Context Compression)	–6.1%	1.04×
Baseline (TP=1, $N_{parallel}$=4)	0%	1.0×
w/ AgentCompress (Search Compression)	-22.1%	0.97×
w/ AgentCompress (Context Compression)	–23.4%	1.03×
Baseline (TP=1, $N_{parallel}$=8)	0%	1.0×
w/ AgentCompress (Search Compression)	-13.2%	1.01×
w/ AgentCompress (Context Compression)	–42.1%	1.08×



推理系统层优化

Agent系统总体架构

Agent框架层驱动任务执行，连接工具执行与推理系统能力



Agent系统总体架构

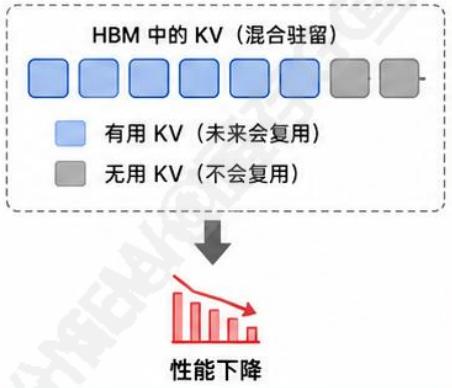
Agent框架层驱动任务执行，连接工具执行与推理系统能力

面向 Agent 工作流的 KV Cache 感知调度

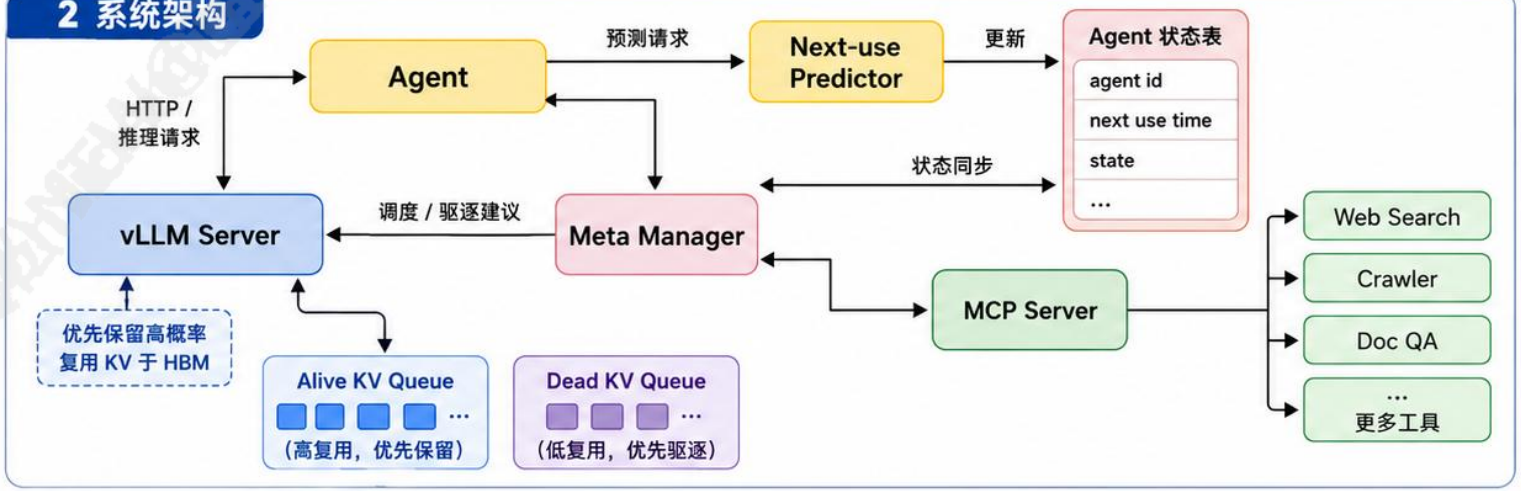
基于工作流状态与未来复用预测，优先保留高价值 KV 于 HBM，提升系统吞吐约 10%

1 传统问题

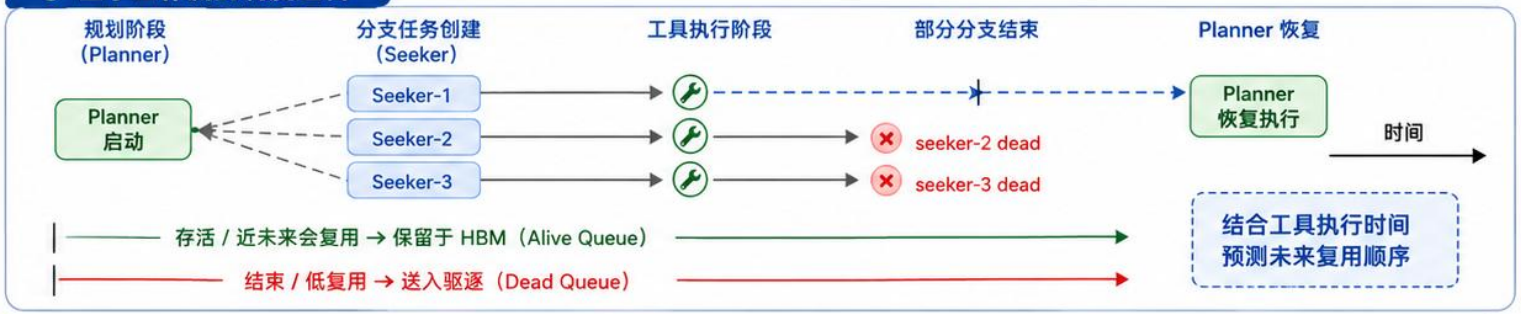
1. 传统调度无法感知 Agent 实际工作流与存留状态
2. 大量无关 KV 被迫驻留 HBM
3. 驱逐顺序无法匹配未来复用，导致重算 / 回拉



2 系统架构



3 基于工作流的调度逻辑



关键收益

- 减少无关 KV 驻留
- 避免重算与回拉
- 系统吞吐提升 ~10%

Agent系统总体架构

Agent框架层驱动任务执行，连接工具执行与推理系统能力

1 传统投机解码

小草稿模型（低成本）

低成本候选生成

T_1 T_2 T_3 … T_k

目标大模型（高精度）

目标大模型并行验证

T_1 T_2 T_3 … T_k ✓ ✓ ✓ ✗

更少逐Token自回归等待

T_1 T_2 T_3 …

- 低成本生成候选Token
- 目标模型并行验证
- 命中率决定加速收益

传统方法：主要依赖局部上下文 / 小草稿模型

2 Agent场景带来新的机会

Think → Act → Observe → Reflect

Agent任务中的重复Pattern

- 工具调用格式
- 搜索策略
- 代码修改模式
- 错误处理逻辑
- 结果总结模板

历史轨迹的跨轮与跨Session重复

会话 1（多轮） 会话 2（多轮） … 会话 N（多轮）

- 多轮任务结构更稳定
- 重复Pattern天然更多
- 可复用历史执行轨迹

Agent越长期运行；可积累的可复用模式越多

3 AgentSAM：全局记忆增强的投机推理

历史轨迹（跨轮）

跨轮上下文

工具模式

跨Session经验

…

后缀自动机上下文记忆（SAM记忆结构）

生成更接近目标模型的候选序列

T_1 T_2 T_3 T_4 … T_n

目标模型验证（并行验证）

T_1 T_2 T_3 T_4 … T_n ✓ ✓ ✓ ✓ ✓

低开销加速

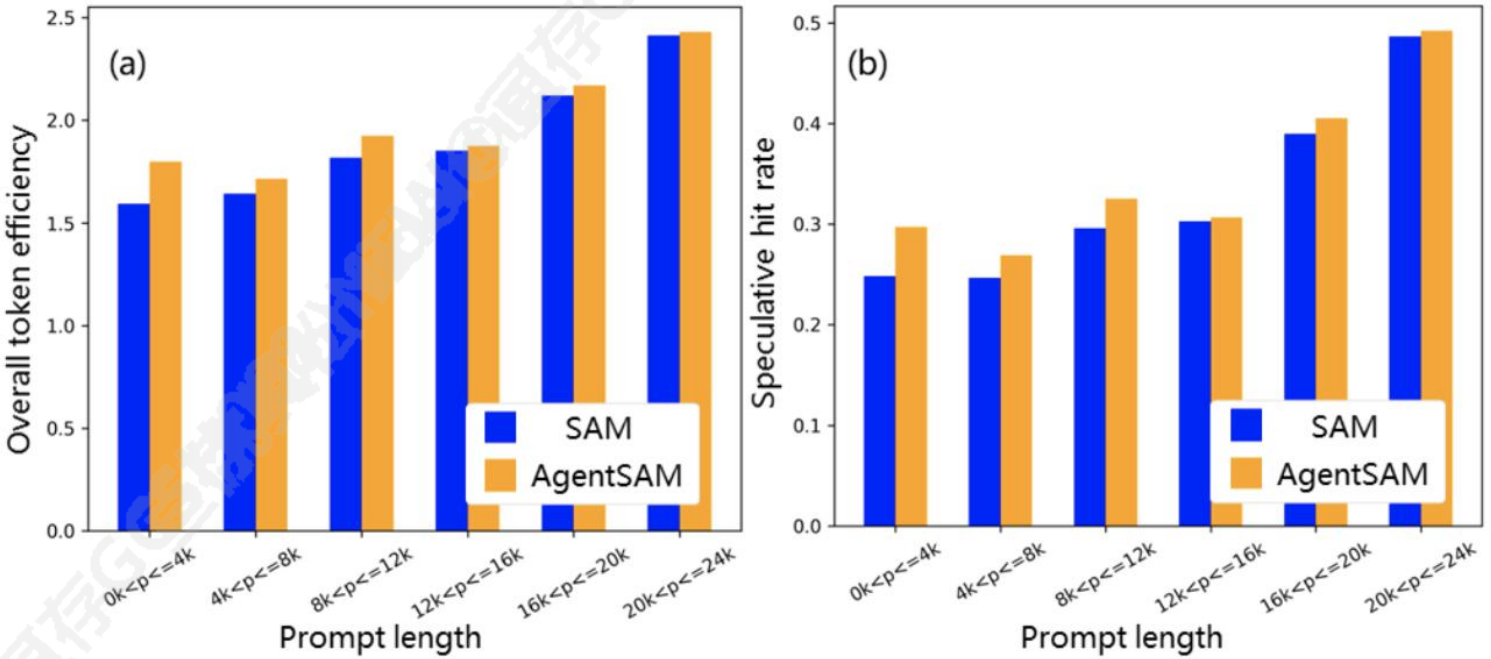
更高命中率，更低时延

- 利用跨轮与跨Session重复结构
- 候选生成不只依赖局部上下文
- 提高候选Token命中率
- 降低Decode时延

传统投机解码 = 局部草稿生成 →

核心价值：把Agent历史执行模式转化为推理加速信号 →

AgentSAM = 全局记忆增强加速

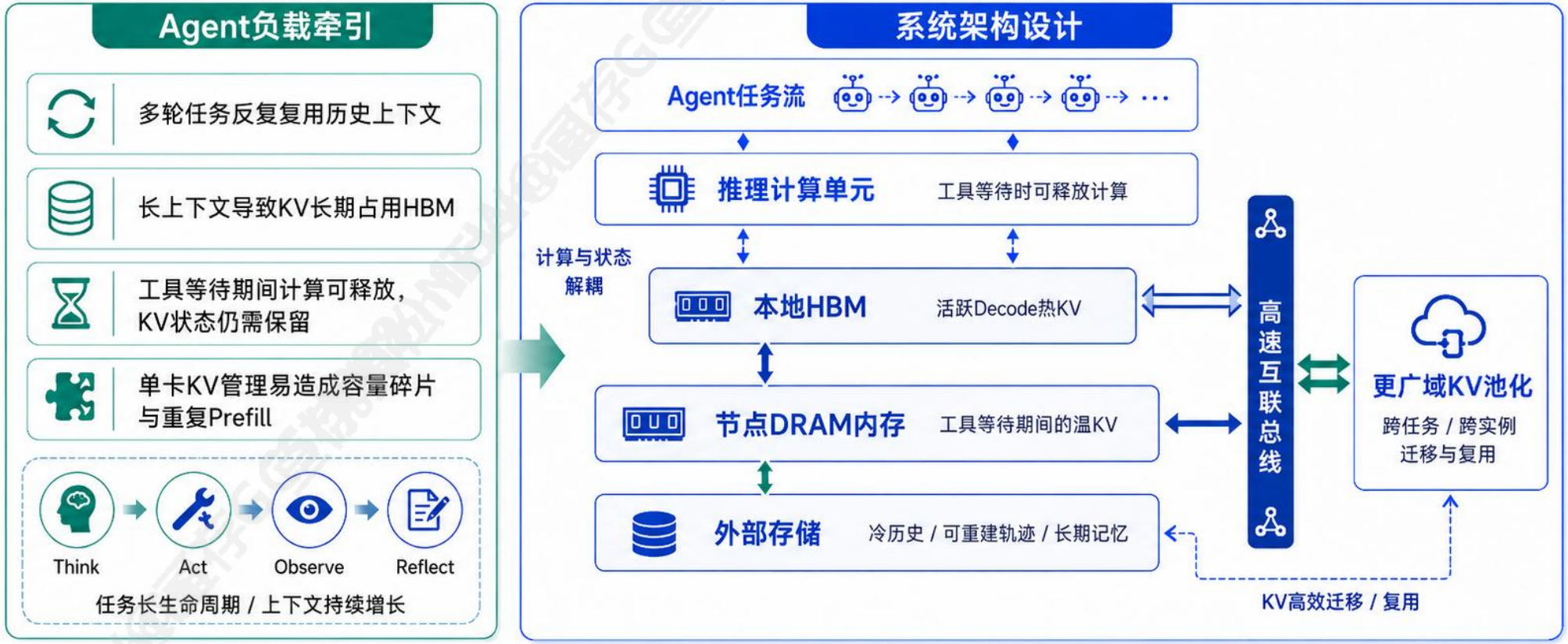


- 全局Agent信息动态提升场景接收率
- 可与更复杂的投机推理技术结合，进一步利用相似Session的内容辅助投机

系统架构优化

Agent系统总体架构

Agent框架层驱动任务执行，连接工具执行与推理系统能力



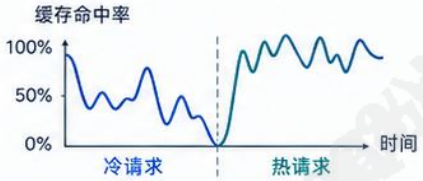
核心思想：Agent不是单次请求，而是任务流；系统架构需要围绕状态承载与缓存复用重新设计。

代表性应用：
LMCache/Mooncake
公开工作：
Predictive Multi-Tier Memory Management for KV Cache
Multi-tier dynamic storage of KV cache for LLM inference under resource-constrained conditions

Agent负载牵引

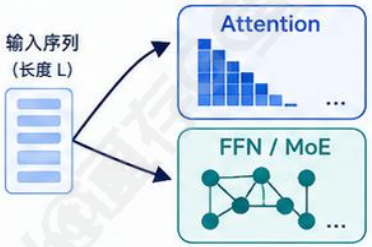
长上下文 | 多轮交互 | 缓存复用 | 强实时反馈

1 Prefill阶段特征



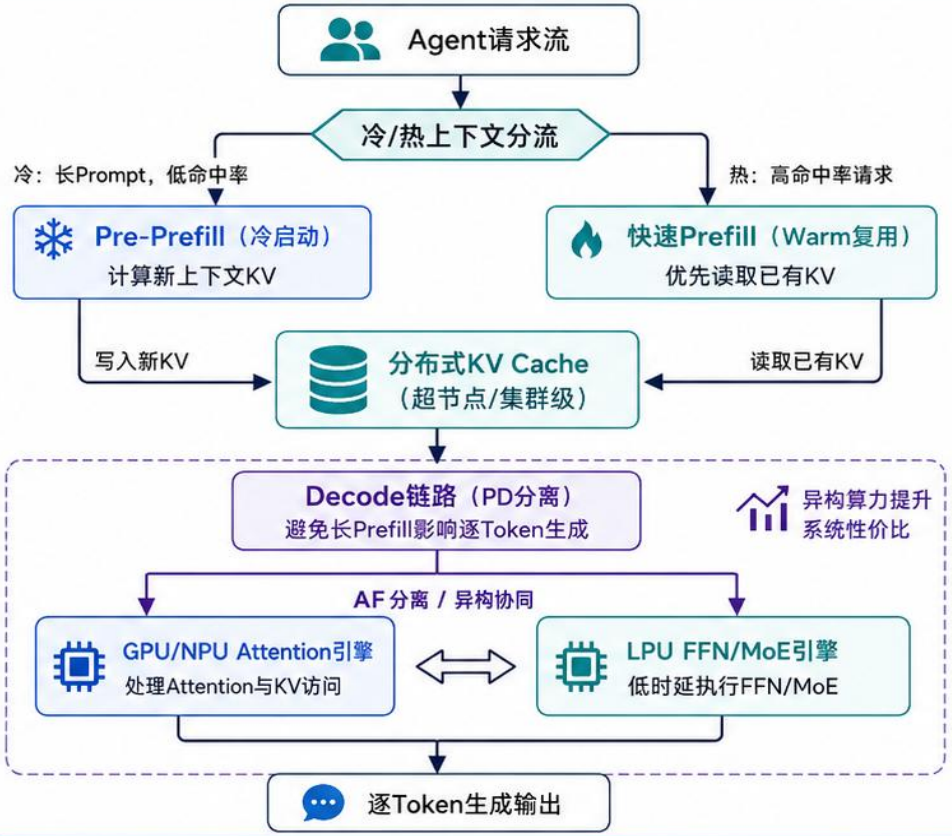
- 缓存命中率波动大
- 冷/热Prompt差异明显
- 算力模式与硬件需求不同

2 Decode阶段特征



- Attention随序列长度变化大
- FFN时延更恒定
- 需要更细粒度异构化

系统架构设计



Pre-Prefill：解决冷启动 | PD分离：隔离Prefill与Decode | GPU/NPU + LPU：实现Attention/FFN异构低时延

Agent负载越长程、越交互、越实时，推理架构越需要缓存感知与异构低时延设计。

Takeaways

1 Agent时代，优化对象从“单次推理”变成“任务完成”。

多轮交互　工具调用　环境反馈　自我修正

单次推理

任务完成（多轮循环）

Think → Act → Observe → Reflect

性能指标升级

传统指标（单轮视角）	任务级指标（端到端视角）
tokens/s	任务成功率
TFFT	端到端时延
TPOT	单任务成本
...	系统吞吐
	...

2 单点优化仍然重要，但真正收益来自模型–算法–系统协同。

模型层	算法层	系统层
RL、OPSD、蒸馏等	大小模型协同、上下文压缩、投机推理等	调度、缓存复用、资源管理等
提升 Token Efficiency	减少 无效推理	释放 任务级效率

协同优化，放大整体收益

更高吞吐 · 更低时延 · 更优成本

3 Agent负载正在反向牵引智能基础设施设计：缓存感知 + 异构低时延架构。

缓存感知的分层KV存储与生命周期管理（状态承载 + 缓存复用）

本地HBM	承载当前活跃Decode的热KV
节点DRAM内存	承载工具等待期间的温KV
外部存储	承载冷历史、可重建轨迹、长期记忆
更广域KV池化	支持跨任务、跨实例的KV迁移与复用

高速互联总线（NVLink / CXL / RoCE 等）
通过高速互联实现KV资源高效迁移与复用

异构低时延架构（Attention/KV 与 FFN/MoE 解耦）

GPU / NPU	LPU	高速互联网络
负责 Attention 与 KV 访问 适合高并行访存	负责 FFN / MoE 低时延执行 稳定、可预测	NVLink / Infinity Fabric 等 低时延、高带宽互联

更低时延　更高吞吐　更优性价比　更强稳定性

最终目标：更高并发 · 更低时延 · 更高成功率 · 更低成本，支撑规模化、可靠的Agent执行

团队：基础大模型部、诺亚方舟实验室、先进计算与存储实验室、计算技术开发部

Weizhe Lin, Hui-Ling Zhen, Shuai Yang, Xian Wang, Renxi Liu, Wenbo Gao, Hanting Chen, Wangze Zhang, Chuansai Zhou, Yiming Li, Chen Chen, Xi Chen, Xing Li, Zhiyuan Yang, Fang Guo, Xiaosong Li, Xianzhi Yu, Yaoyuan Wang, Zhenhua Dong, Mingxuan Yuan, Yunhe Wang

团队相关工作

- Towards Efficient Agents: A Co-Design of Inference Architecture and System：Agent推理架构与系统协同加速。
- AgentCollab: A Self-Evaluation-Driven Collaboration Paradigm for Efficient LLM Agents：大小模型动态协同提升Agent效率。
- DLLM Agent: See Farther, Run Faster：扩散模型并行生成加速Agent执行。
- TrimR: Verifier-based Training-Free Thinking Compression for Efficient Test-Time Scaling：验证器驱动压缩冗余思考。
- Scaling Up, Speeding Up: A Benchmark of Speculative Decoding for Efficient LLM Test-Time Scaling：系统评测投机解码加速慢思考。
- Revisiting Judge Decoding from First Principles via Training-Free Distributional Divergence：免训练Judge解码提升投机验证。
- EPD-Serve: A Flexible Multimodal EPD Disaggregation Inference Serving System on Ascend：多模态EPD分离推理系统。
- Unleashing Low-Bit Inference on Ascend NPUs: A Comprehensive Evaluation of HiFloat Formats：Ascend低比特推理格式评估。
- ……