Flink Forward ASIA
Organized by Alibaba Cloud

AI Agent: Two Clouds

Streaming Interaction with the Environment · Autonomously Learning from Experience

Bojie Li | https://01.me

ShenZhen
2026. 6.26 - 27

OVERVIEW

Two Clouds: today's AI Agents — two major challenges

We're already used to AI that writes code and does research. But we rarely see an AI that can converse and operate a computer in real time, like a human.

Streaming Interaction with the Environment

Voice and video are streaming data that must be processed in real time.

The challenge: keep interaction real-time (respond within a few hundred ms, allow natural interruption) while still thinking deeply.

Autonomously Learning from Experience

How the experience accumulated during interaction is stored and reused.

The challenge: not retraining every time, but continually learning and distilling knowledge from each interaction.

THESIS

A claim: interaction is streaming event processing

Turning "turn-by-turn" interaction into streaming event processing — the same design philosophy as Flink.

Request / Response
= micro-batch
one question, one answer; compute once the batch is complete

Real-time interaction
= true streaming
per-event processing, event-time

Interfaces designed right
existing models need no retraining
the bottleneck is how data flows in/out and how it is represented

ROADMAP

Talk structure: one Agent is one Flink job

	Agent Stage	Corresponding Flink concept	Work in this talk
Perception	**World → Model**	Source + event-time + custom serialization	**AOI observation interface** Sema · semantic transport
Cognition	**real-time + deep thinking**	unified batch & stream stream = low latency / batch = high throughput	**Interactive ReAct** Latent Bridge
Memory	**store & reuse experience**	stateful stream processing state backend + checkpoint	**UserAsCode · Engram** PreAct

PART 1 · Perception

The world enters the model = Flink is the Source

Making an Agent listen and see in real time like a human, the first question is how world data enters the model——— fundamentally a data-source and serialization problem.

① Cannot capture dynamics

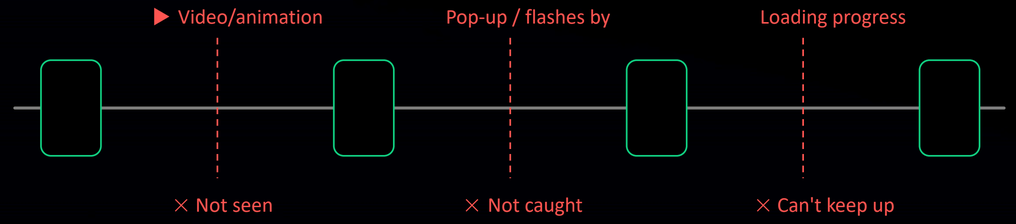
One static screenshot every 3–5 s; everything that changes in between is lost.

② No hearing

No audio input at all: meeting speech, notification sounds, and alerts are all unavailable.

VideoWebArena On video-dependent tasks the best model scores only 13.3%; humans 73.9%.

Agent looks at the screen once every 3-5 s; everything between two glances is missed



No audio input at all: meetings, notification tones, and alerts all go unheard

Section structure Perception has two interfaces: an observation interface (AOI, deciding what to observe) and a transport interface (Sema, cutting transport cost), corresponding to the data source and serialization.

PART 1 · Perception

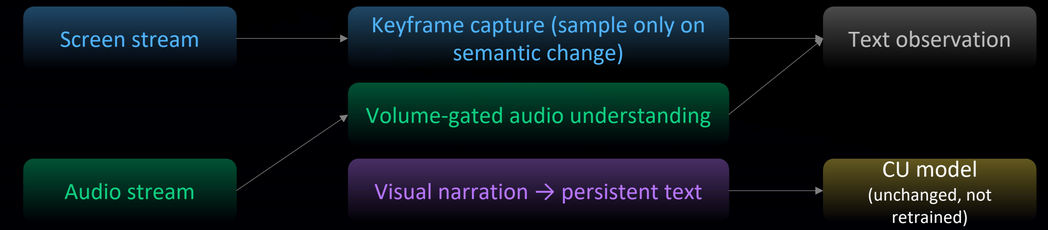
AOI · make observation “event-triggered”

arXiv: 2606.29472

Core: decouple continuous observation from discrete action

- Three gated components — see / hear / note: turn off when the screen or sound is unchanged; zero extra cost on static pages.

A counter-intuitive finding Having the model write the current frame into one sentence stored in context is the single highest-gain component (+18pp).



+17~48pp
dynamic tasks vs pure screenshots

82%
best (Claude), no drop on static

12 / 12
all audio tasks completed

Link to memory Transcribing fleeting visual observations into persistent text is equivalent to building short-term memory for the model; persistent memory is discussed in Part 3.

PART 1 · Perception

Sema · the receiver is the model, not a human

arXiv:2604.20940

When the downstream consumer is the model, two assumptions break

- No perceptual-level reconstruction needed —— the model only needs task-relevant task-relevant discrete semantic token.
- The model is event-driven, has no physical clock, and is insensitive to delivery jitter.

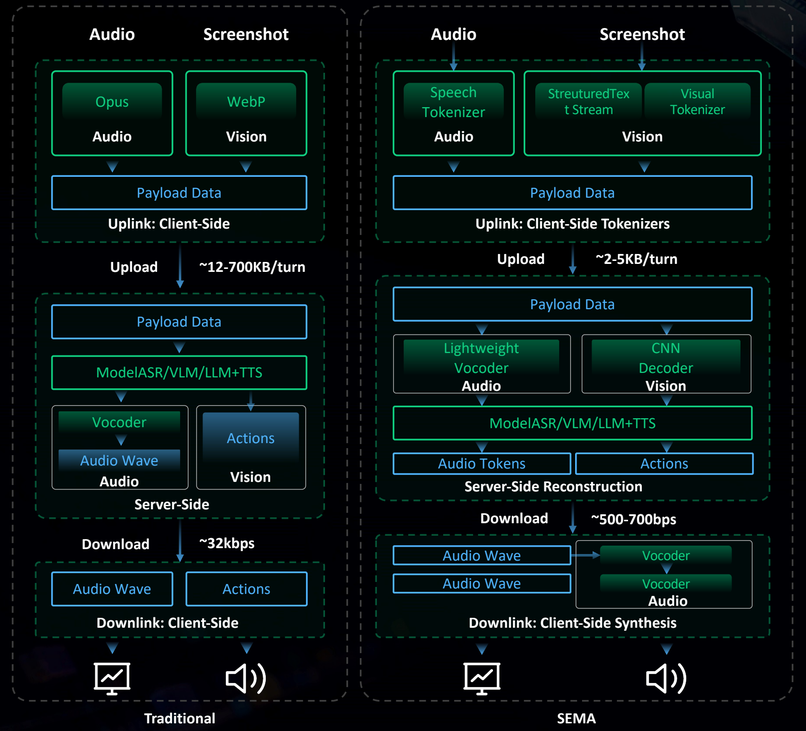
Shannon–Weaver The transport goal shifts from signal fidelity (Level A) to semantic fidelity (Level B).

11–269×
visual uplink compression

48.6×
voice uplink compression

≈ lossless
end-to-end task accuracy

Generality of the principle "The receiver is the model" applies equally in Cognition: the channel between the fast and slow models likewise needs only a compressed semantic representation.



PART 2 · Cognition

Real-time and intelligence are nearly orthogonal

Perception delivered the world into the model; now, under real-time constraints, the model must think both fast and deep.

Real-time but shallow reasoning

- Real-time interaction must decide ~200ms whether to speak and what to say roughly every
- omni / omni / end-to-end voice can be real-time, but reasoning is weak.

Strong reasoning but not real-time

- SOTA a single response from an SOTA model often takes 300–500ms or more.
- Classic Observe-Think-Act: wait for full input, then reason at length, then answer.

Approach: split fast and slow a small foreground model handles fast responses, a background SOTA slow model handles deep reasoning.

PART 2 · Cognition

Splitting fast and slow is unified batch & stream

Perception delivered the world into the model; now, under real-time constraints, the model must think both fast and deep.

Foreground • stream processing

- a small model, per-event, low-latency, holding the turn at about 200ms .
- the core is orchestration: deciding what to answer immediately and what to hand to the background.

Background • batch processing

- use an SOTA model directly, ensuring the reasoning ceiling and throughput.
- the background does deep planning without blocking foreground interaction.

Flink analogy Under one set of semantics, the stream seeks low latency and the batch seeks high throughput — the fast and slow path paths are exactly unified batch & stream.

An action game as example With only the fast system there is no long-term planning; with only the slow system it cannot react in time and loses from the start.

PART 2 · Cognition

Interactive ReAct · Think While Listening (Think While Listening)

User	Candidate: I previously led building distributed systems…
Think	Distributed systems — need to gauge depth, let me search first…
Tool	web_search("candidate distributed systems project") async
User	…we used Kafka+Redis to handle 10M req/s searching while they talk
Think	Kafka+Redis fits high throughput, keep listening…
Tool	GitHub: 3 open-source projects, 2000+ stars total tool result
Think	The result confirms the experience! Combined with what the candidate said…
Assistant	That scale is impressive! Tell me about the trickiest scaling challenge? <0.5s

Why it can think while listening

- LLM generates ~200 token/s, far faster than a human speaks at ~5–10 token/s.
- Observation = event stream; action = tokens.
- Reason in the gaps between words; trigger tools asynchronously.

Effect By the time the user stops speaking, the answer / tool result is often already ready.

PART 2 · Cognition

Interactive ReAct · Speak While Thinking (Speak While Thinking)

User	Candidate: I'm ready to answer technical questions.
Think	A complex question, needs careful organization.
Assistant	Let me give you a system-design question. fast <0.5s
Think	Need to cover scalability, consistency, latency… deep think ~5s
Assis tant	Imagine you're building a global CDN.
Think	Go on — raise the difficulty of cache invalidation…
Assistant	When content updates, how do you invalidate caches across 100+ edge nodes?

Why it can speak while thinking

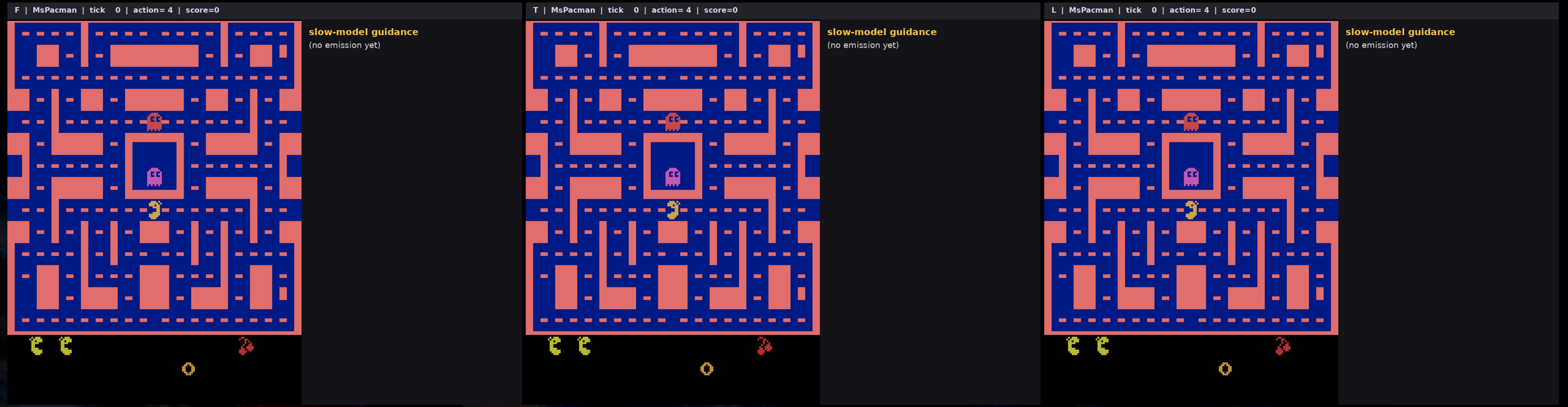
- Fast (<0.5s): speak first to hold the turn.
- Slow (~5s): think deeply in the background, then continue smoothly.
- Keep thinking while speaking; the question unfolds naturally, sentence by sentence.

Natural interruption receiving an interrupt while speaking stops it and returns to processing

PART 2 · Cognition

Latent Bridge · which channel between fast and slow?

arXiv:2606.24470



PART 2 · Cognition

Latent Bridge · which channel between fast and slow?

arXiv:2606.24470

A task needing both fast reflexes and long-horizon planning

- Ms. Pac-Man: fast/slow split is visible and objectively scorable.
- Compare three “channels between fast and slow”:
- fast model only (F) / text channel (T) / continuous latent channel (L).

The “channel between fast and slow” lives here: both frozen models stay fixed, only this middle piece changes

Slow model (large)

Long-term planning
runs only every K frames

Channel (slow → fast)

F: no channel, the fast model decides alone (only passive dodging)

T: text channel — the slow model writes one sentence to the fast model

L: continuous latent channel — the slow model passes a latent vector directly

avoids the loss of “compressing to text”; denser, faster information

Fast model (small)

Fast reflexes
outputs an action every frame

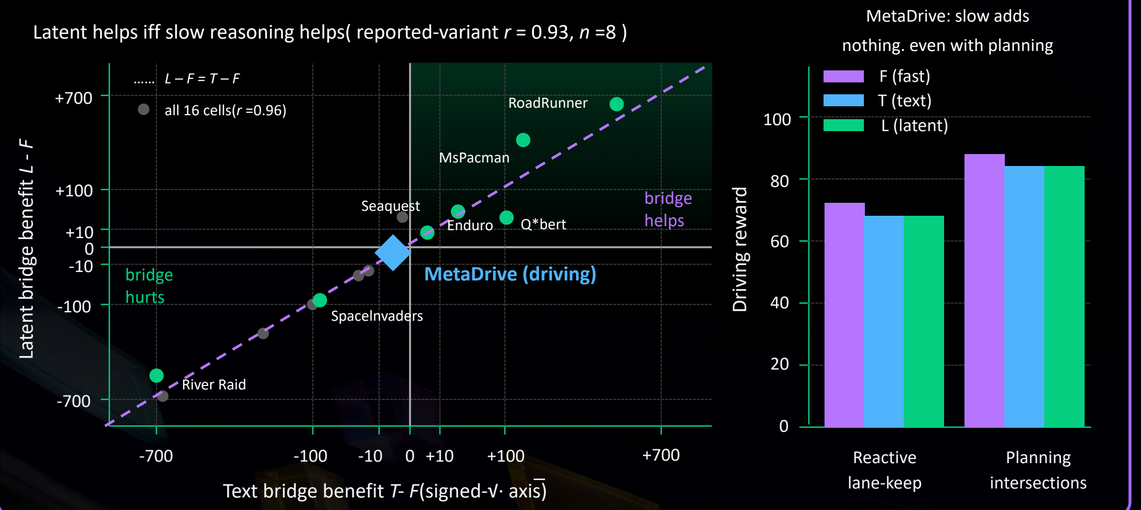
The three configs differ only in this one channel, else identical. ScoreL 628>T408>F

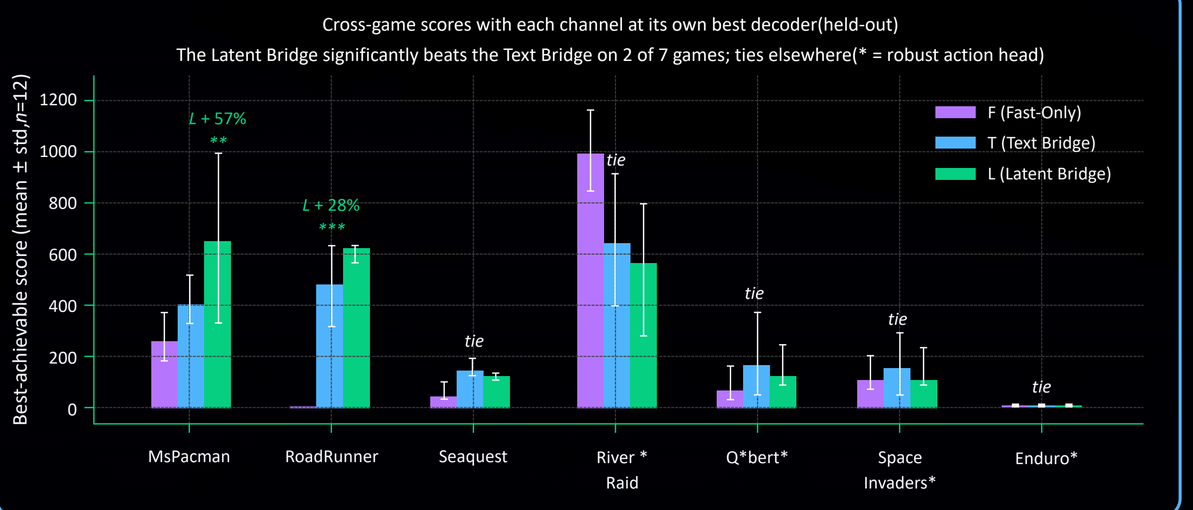
PART 2 · Cognition

Latent Bridge · the point isn't bandwidth, it's whether slow thinking helps

Freeze both ends, learn only the middle bridge (33M params), making the channel the only variable; fast MiniCPM-o 4.5(9B)/ slow Qwen3-VL-8B-Thinking.

arXiv:2606.24470





Conclusion latent bridge is never significantly worse than the text bridge; across 7 games it wins significantly on 2 (Ms. Pac-Man +57%, RoadRunner +28%)

PART 3 · Memory

The memory landscape: where experience lives and how it is retrieved

Memory = Flink's core capability: stateful stream processing (state backend + checkpoint) — extract knowledge from interaction, store it, and reuse it.

For the same problem, first split by "what to remember" into two kinds——**declarative (who the user is)** and **procedural (how to do something)**.

Declarative memory · who the user is — two implementations: external code vs model parameters

① UserAsCode + Programmable KV

≈ external structured storage

facts → typed code + executable constraints; interpreter computes over facts

cache memory & Skills via KV Cache composabilitySkills

② User as Engram

≈ inside model parameters

facts → written into sparse slots in model parameters; internalized by the model, zero context

Procedural memory · how to do it (skills)

③ PreAct

≈ cache

successful trajectory → executable state machine; run on hit, fall back on miss

In common All three add stronger structure on top of the "natural-language memory" baseline and separate content from computation / reasoning.

PART 3 · Memory

① UserAsCode · represent user memory as code

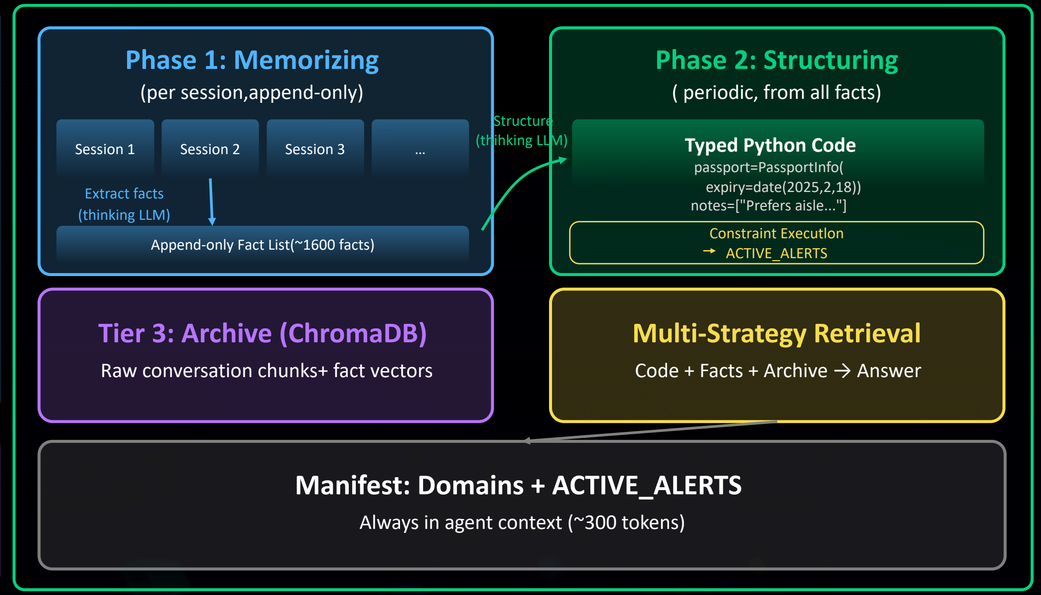
User memory = a set of typed data; a two-phase pipeline turns conversation into runnable code.

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)],
)
```

Two phases

① append-only fact log (append-only) ② periodic LLM structuring into typed code(= checkpoint).



① UserAsCode · executable constraints (advanced)

bag-of-facts is good at recall but cannot compute or constrain

· conflict resolution / aggregate computation / constraint checking: all require computation and judgment over the records.

```
# 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()} "
```

The interpreter executes deterministically

- Compute and aggregate across many records, proactively pre-computing alerts.
- A pile of scattered facts cannot do this.

Flink / DB analogy

append-only log + periodic checkpoint = changelog + checkpoint = a classic form in database systems.

PART 3 · Memory

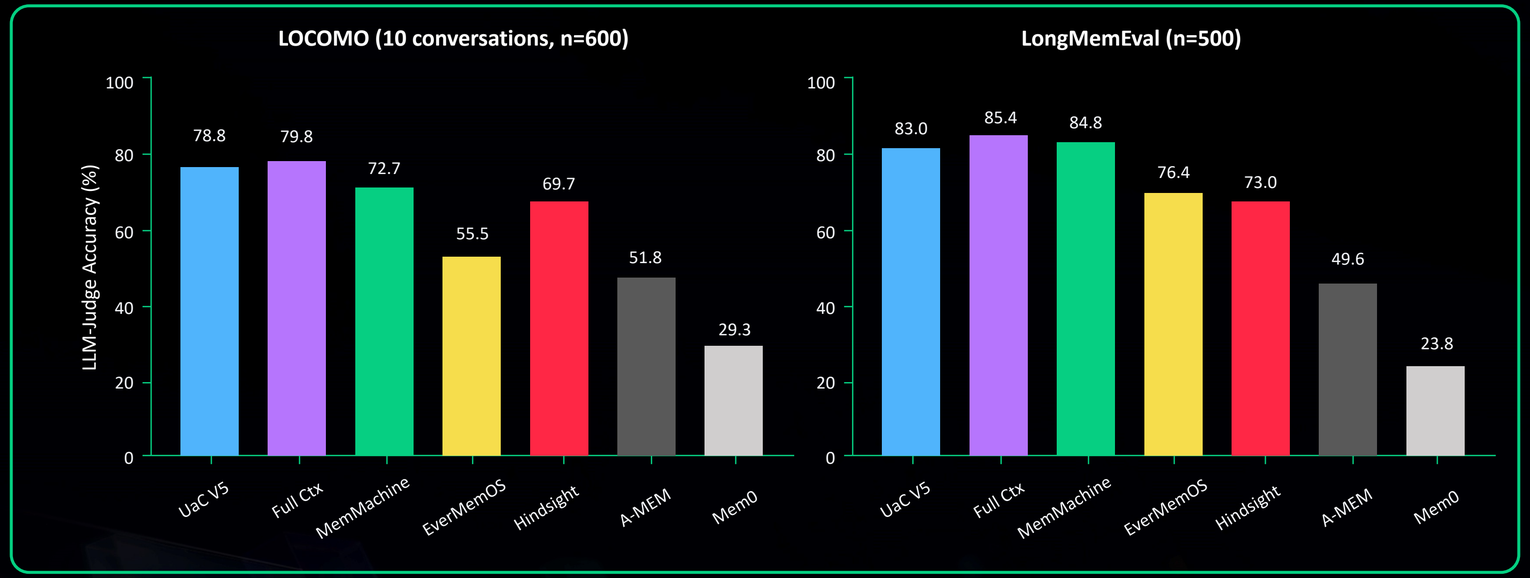
① UserAsCode · evaluation

LOCOMO / LongMemEval comparing retrieval-based memory systems against the full-context upper bound.

78.8%
LOCOMO ≈ upper bound 79.8%

99%
aggregate reasoning (retrieval-based 6–43%)

~15×
cheaper · alert detection 100%



① Programmable KV · compile memory and Skills into KV Cache

arXiv:2606.17107

UaC compiles memory into code, but every turn re-prefills this code. Can we pre-compile it into a KV Cache and just splice it in?

Pain point · memory re-read every turn

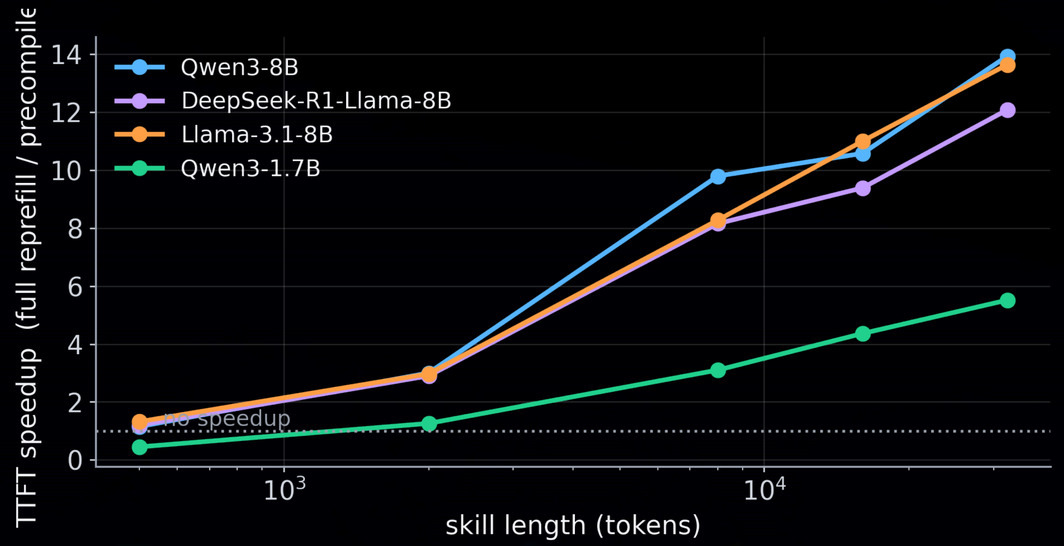
- code / Markdown memory re-prefilled every turn prefill, TTFT grows with length **O(L²)**.
- prefix cache hits only on “byte-for-byte identical”: any change invalidates the whole segment.

Method · pre-compile + RoPE relocation

- pre-compile each memory block once, then **RoPE-relocate** it before the query each turn.
- **O(L)** instead of O(L²), no re-prefill; behavior indistinguishable from full recompute.

13.9×
faster TTFT @32k

0.90–0.999
logit cosine · equivalent to full recompute



pre-compiled skills' TTFT speedup vs skill length (O(L²) → O(L))

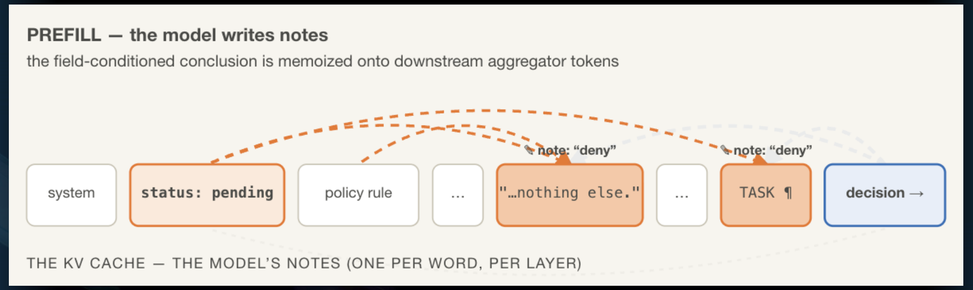
① Programmable KV · edit a field in the KV Cache with a sticky-note

arXiv:2606.17107

Field updates in memory (account status, time, quota) need not recompute the whole segment: append-only just append one revision sticky-note.

Why not directly edit the token's KV?

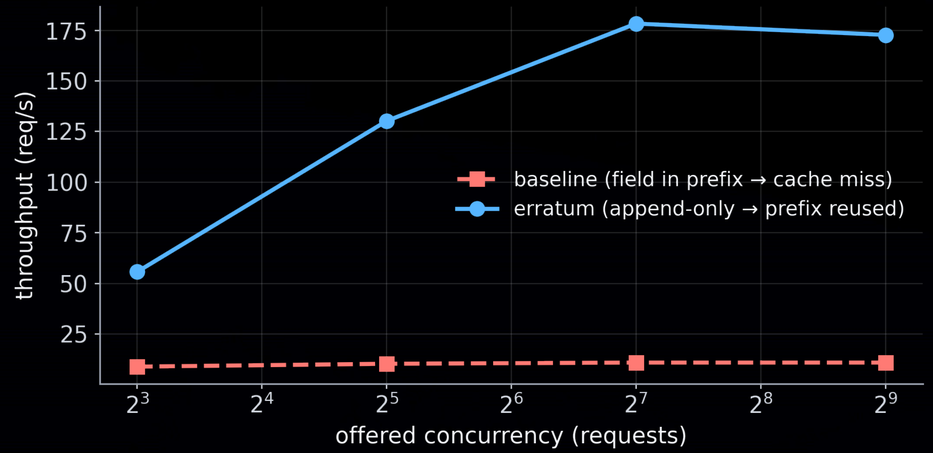
· if the field precedes the rule, the KV Cache of the later rule records reasoning conclusions based on the field's content



Sticky-note revision · erratum

· keep the old note, append an explicit "[update] X is now…" at the end.

· recompute only ~6%; append-only, aligned with prefix caching.



98.5% vs 1%
prefix cache hit rate

14.5×
throughput

53x
lower p90 TTFT

PART 3 · Memory

② User as Engram · write user facts into model parameters

arXiv:2606.19172

First, understand Engram (DeepSeek)

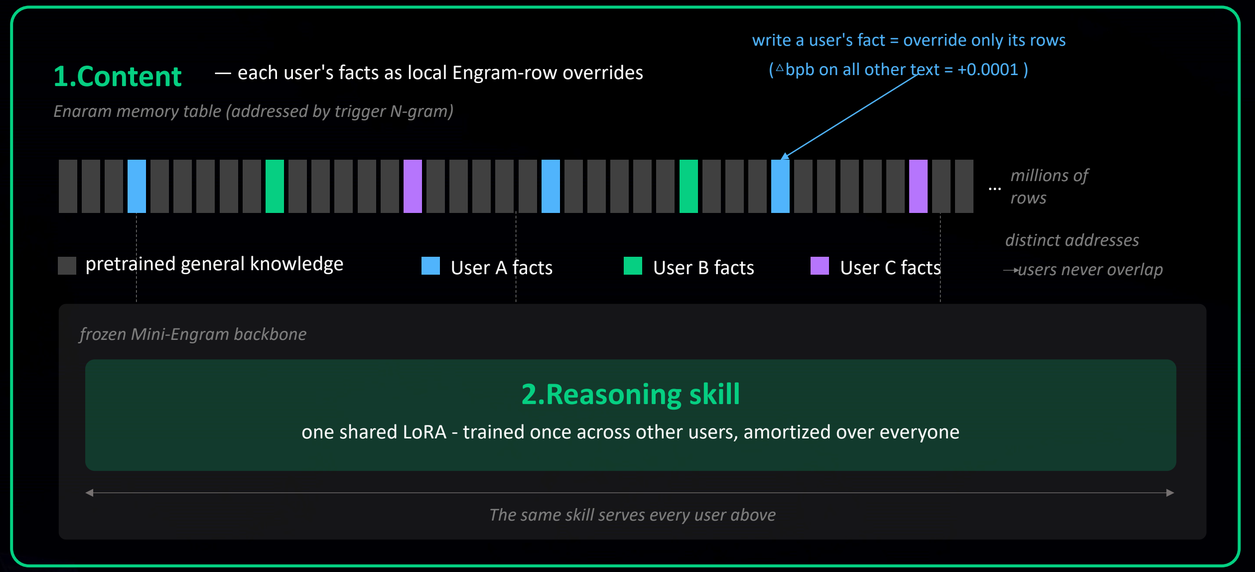
- a very large but sparse memory table hangs beside the model as addressable external parameters.
- a trained gate decides when, and which memory row, to inject into the model.

Problem: LoRA-as-memory

- per-user LoRA is a global weight delta that pollutes unrelated text; direct recall is near-perfect, but indirect reasoning drops.

Idea: content / skill decoupling

- write one user fact into an empty slot (isolated per user); reasoning skill is carried by a shared adapter.

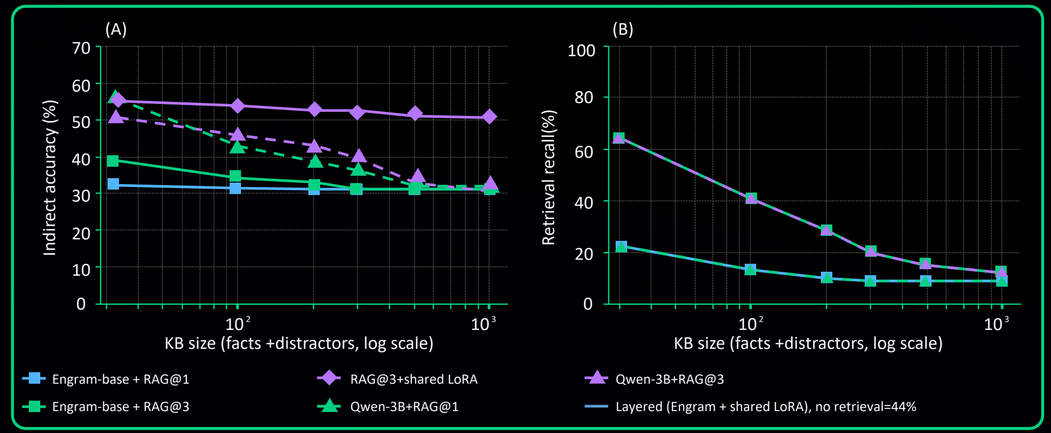


Relation to UserAsCode both separate content from computation / reasoning (UaC: data vs constraints; Engram: fact rows vs shared reasoning adapter); the only difference is whether facts live in code or in parameters.

② User as Engram · multi-tenant isolation and evaluation

A storage system with addressing and isolation

- per-user override table: loaded per request, restored after use .
- zero cross-user leakage (guaranteed by construction).
- address-disjoint overrides commute and stack additively .
- plug-and-play like Stable Diffusion LoRA: enterprise + personal fragments stack at inference .



5.6×
indirect reasoning vs LoRA

~33,000×
less pollution of unrelated text

>100 facts
beats a 2.5× larger model's retrieval

③ PreAct · compile repeated work into a cache

arXiv:2606.17929

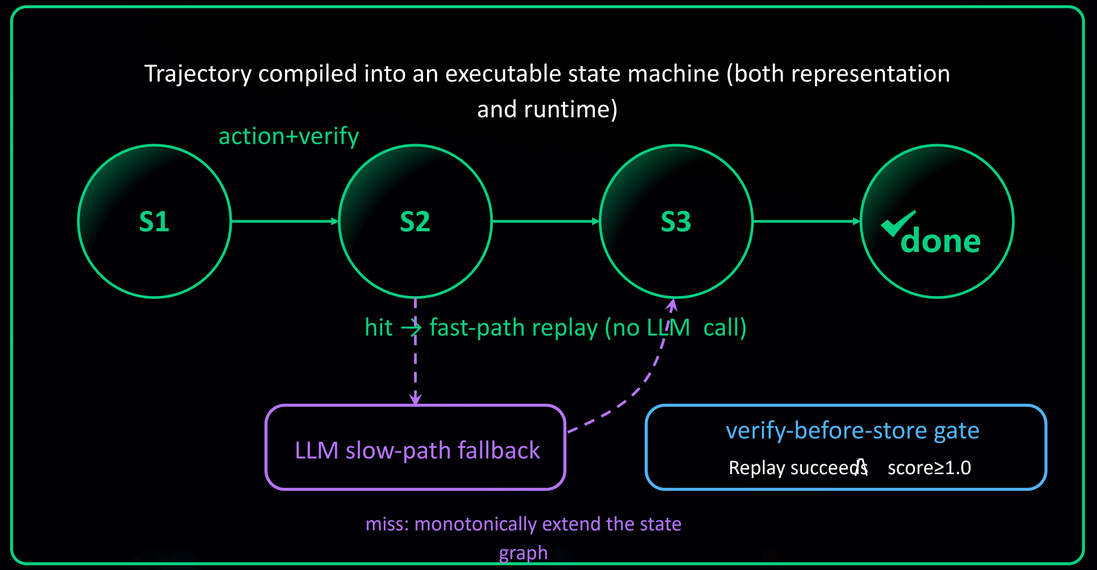
The Rerun Crisis of Computer Use

- every task needs many rounds of "look at screen → reason → act" (seconds per step).
- already-done tasks still re-derive the same action sequence at O(M×N) repeatedly.

Procedural memory the first two are declarative (the user's facts); PreAct is procedural (how to do something).

Design: the state machine is executable

- compile a successful trajectory into a formal state-transition graph: both representation and runtime.
- lightweight XPath state checks; compiled output carries conditional branches.



③ PreAct · compile repeated work into a cache

Task: create contact Anita Goodall

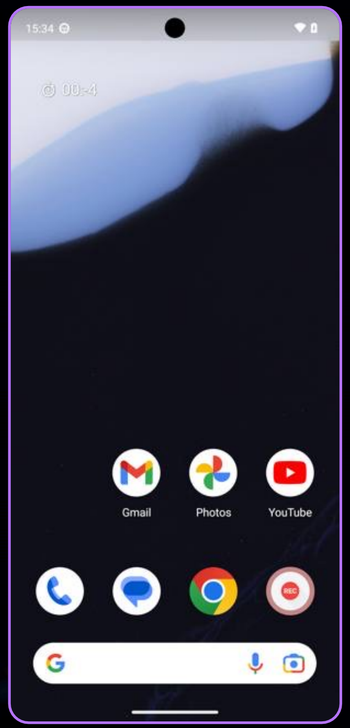
- first run: the agent steps through “look at screen → reason → act” to finish the task (replay on the right).
- once successful, compile into a state machine; a similar task later replays directly, with no per-step LLM call.
- each step first uses XPath to verify the page, acting only when confirmed correct.

```
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 · Memory

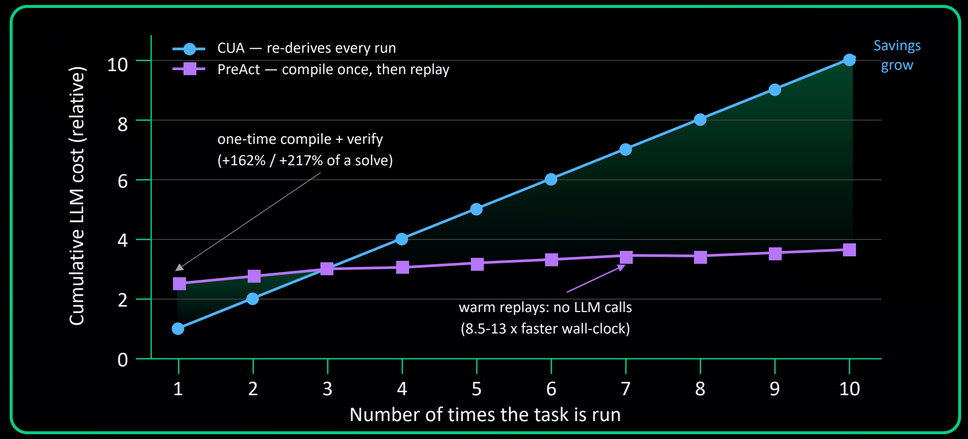
③ PreAct · hit takes the fast path, miss falls back

hit → compiled fast path (deterministic, no LLM call)

miss → LLM slow-path fallback (matching the fast/slow mechanism in cognition)

Quality assurance

- verify-before-store: store only if replay succeeds and score≥1.0 .
- monotonic refinement: each fallback extends the existing state graph rather than rebuilding it.
- UI single-cycle refinement on UI change, keeping LLM -level adaptability.



8.5–13×
replay speedup(no per-step LLM)

73.3%
AndroidWorld (Gemini=Claude)

+1.75~2.6
net task gain from verify-before-store//benchmark

Consistency with Cognition hit takes the fast path, miss falls back to the slow path — consistent with the fast / slow dual-path mechanism in Cognition.

CONCLUSION

Back to AI Agent's two clouds

Streaming Interaction with the Environment

→ Perception (Source) + Cognition (unified batch & stream)

Autonomously Learning from Experience

→ Memory (stateful stream processing)

What decides a system's performance is often the interface and representation, not the model itself.

Multimodal observation (AOI) • network transport (Sema) • thought channel (Latent Bridge)

State & memory (User as Code / Programmable KV / User as Engram / PreAct)

This is exactly Flink 's core thesis.

Related papers & interactive demos: 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 2606.29472 · Interactive ReAct (coming soon)

Flink Forward ASIA

Organized by Alibaba Cloud

THANK YOU

Thank you