Introduction to Modern LLM Agent Implementation

How modern agents are built

Houdong Liang

INTRODUCTION

Why Learn Agents?

- Agents are already part of everyday life
- A young field — still moving very fast

INTRODUCTION

Agents You May Already Use

- **Coding** — Cursor, Codex, Claude Code, Antigravity
- **Task-oriented** — Pine AI, Decagon, Intercom Fin, Harvey, Glean
- **Browser** — Browser Use, Claude in Chrome, Perplexity Comet
- **In-house** — QA bots, workflow agents, and domain-specific tools

INTRODUCTION

How We Will Study Agents

- History is short — core ideas since ~2022–2023
- Learn from **research** and **industry** together
- Terminology is still fluid — trace ideas, not buzzwords

INTRODUCTION

A Short History

INDUSTRY

Nov 30, 2022
ChatGPT
GPT-3.5 · chat interface
OpenAI

Jun 13 , 2023
Function Calling
Chat Completions API
OpenAI docs

Nov 25, 2024
MCP
Model Context Protocol
Anthropic

Feb 24, 2025
Claude Code
Agentic coding harness
Anthropic

Oct 16 , 2025
Agent Skills
Modular skill packages
Anthropic

Apr 2026
Cursor 3.0
Cloud & background agents
Cursor

Jul 28, 2026
MCP 2026-07-28
Stateless protocol core
Anthropic

2022	2023	2024	2025	2026
Oct 6, 2022 **ReAct** Reason + Act loop arXiv	**Feb 9, 2023** **Toolformer** Self-taught tool use arXiv	**May 6, 2024** **SWE-agent** Agent-computer interface arXiv	**Jan 20, 2025** **DeepSeek-R1** RL for reasoning arXiv	**Mar 2026** **ProRL Agent** Rollout-as-a-Service · multi-turn RL arXiv
	Mar 20, 2023 **Reflexion** Verbal reinforcement arXiv		**Aug 2025** **Agent RL** Multi-turn SWE + RL arXiv	**Aug 2026** **Lego-RL** Harness-native coding RL arXiv
	May 17 , 2023 **Voyager** Skill library · Minecraft arXiv			
	Oct 2023 **SWE-bench** Real-world SWE eval arXiv			

RESEARCH

LITERATURE REVIEW

Core Papers

Oct '22

ReAct

Reason + Act loop

arXiv

Feb '23

Toolformer

Self-taught tool use

arXiv

Mar '23

Reflexion

Learn from failure

arXiv

May '23

Voyager

Skill library

arXiv

LITERATURE REVIEW

ReAct

- Interleave **Reasoning** and **Acting**
- Thought → Action → Observation loop
- Even a strong "brain" must learn how to think

Yao et al., Oct 2022 — ICLR 2023

LITERATURE REVIEW

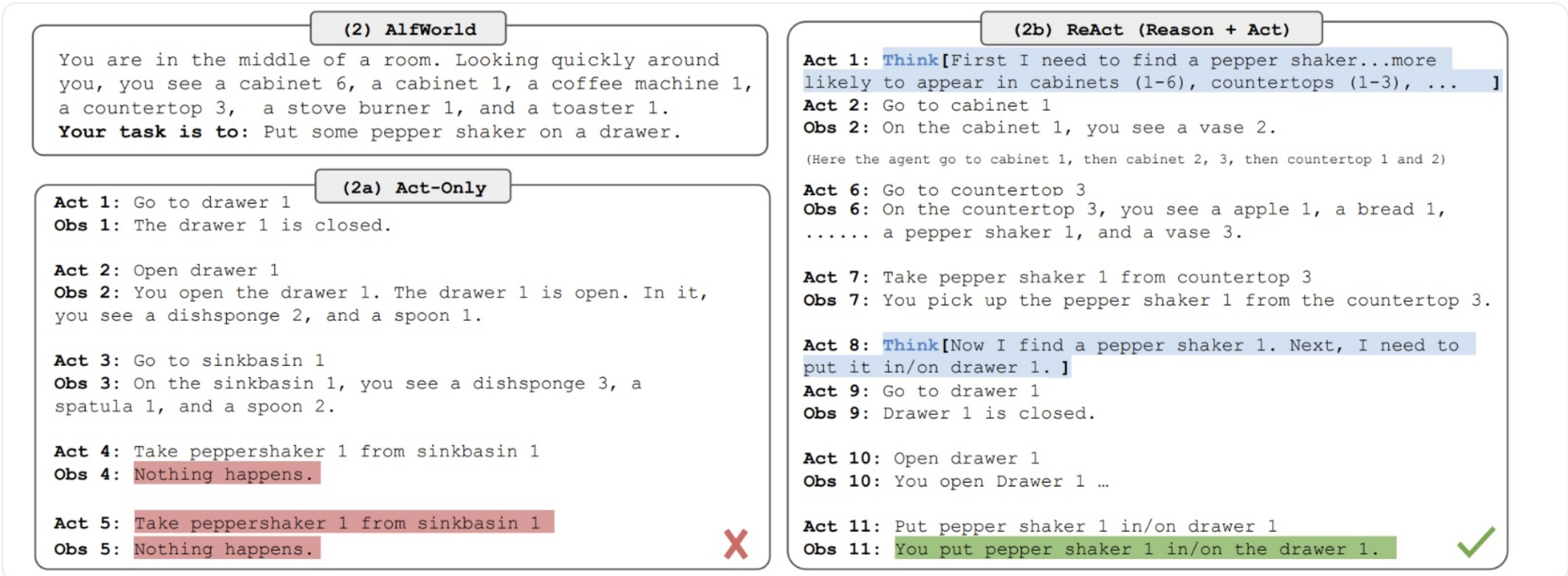
ReAct

$$\pi(a_t \mid c_t) \quad c_t = (o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t)$$

- Action depends on **full context** — not the latest input alone
- In LLM agents: o_t = observations, a_t = actions — all as messages

LITERATURE REVIEW

ReAct



- AlfWorld: **Think** before Act — plan the search, then execute

LITERATURE REVIEW

ReAct

- Implementation = pure **prompt engineering**
- No tool-calling API — actions & observations are plain text in context

Table 8: An ReAct prompt on the ALFWorld clean task.

```
You are in the middle of a room. Looking quickly around you, you see a cabinet
13, a cabinet 12, a cabinet 11, a cabinet 10, a cabinet 9, a cabinet 8, a cabinet
7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet
1, a coffeemachine 1, a countertop 1, a diningtable 1, a drawer 1, a fridge 1, a
garbagecan 1, a microwave 1, a shelf 3, a shelf 2, a shelf 1, a sinkbasin 1, a
stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: put a clean lettuce in diningtable.
> think: To solve the task, I need to find and take a lettuce, then clean it with
sinkbasin, then put it in diningtable.
OK.
> think: First I need to find a lettuce. A lettuce is more likely to appear in
fridge (1), diningtable (1), sinkbasin (1), stoveburner (1-3), cabinet (1-13). I
can check one by one, starting with fridge 1.
OK.
> go to fridge 1
The fridge 1 is closed.
> open fridge 1
You open the fridge 1. The fridge 1 is open. In it, you see a cup 3, a egg 2, a
potato 3, and a potato 2.
> go to diningtable 1
On the diningtable 1, you see a apple 1, a bread 1, a butterknife 2, a cup 2, a
fork 2, a knife 2, a knife 1, a ladle 1, a lettuce 1, a mug 2, a mug 1, a pan 2, a
peppershaker 1, a spatula 3, a tomato 2, and a tomato 1.
> think: Now I find a lettuce (1). Next, I need to take it.
OK.
> take lettuce 1 from diningtable 1
You pick up the lettuce 1 from the diningtable 1.
> think: Now I take a lettuce (1). Next, I need to go to sinkbasin (1) and clean
it.
OK.
> go to sinkbasin 1
On the sinkbasin 1, you see a apple 2, a ladle 2, a spoon 1, and a tomato 3.
> clean lettuce 1 with sinkbasin 1
You clean the lettuce 1 using the sinkbasin 1.
> think: Now I clean a lettuce (1). Next, I need to put it in/on diningtable 1.
OK.
> go to diningtable 1
On the diningtable 1, you see a apple 1, a bread 1, a butterknife 2, a cup 2, a fork
2, a knife 2, a knife 1, a ladle 1, a mug 2, a mug 1, a pan 2, a peppershaker 1, a
spatula 3, a tomato 2, and a tomato 1.
> put lettuce 1 in/on diningtable 1
You put the lettuce 1 in/on the diningtable 1.
```

LITERATURE REVIEW

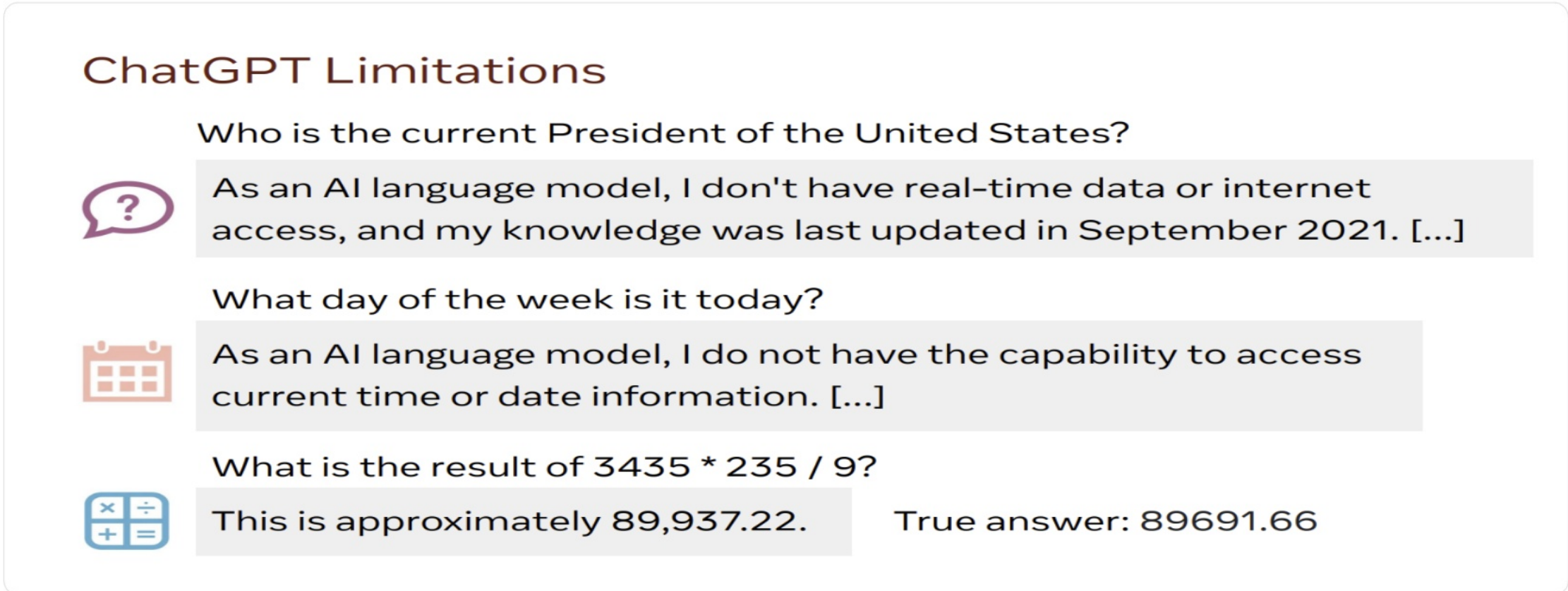
Toolformer

- Teach a model **when** to call APIs — self-supervised
- Collect cases where tools clearly improve the answer
- Fine-tune GPT-J → beats much larger models without tools

Schick et al., Feb 2023 — NeurIPS 2023

LITERATURE REVIEW

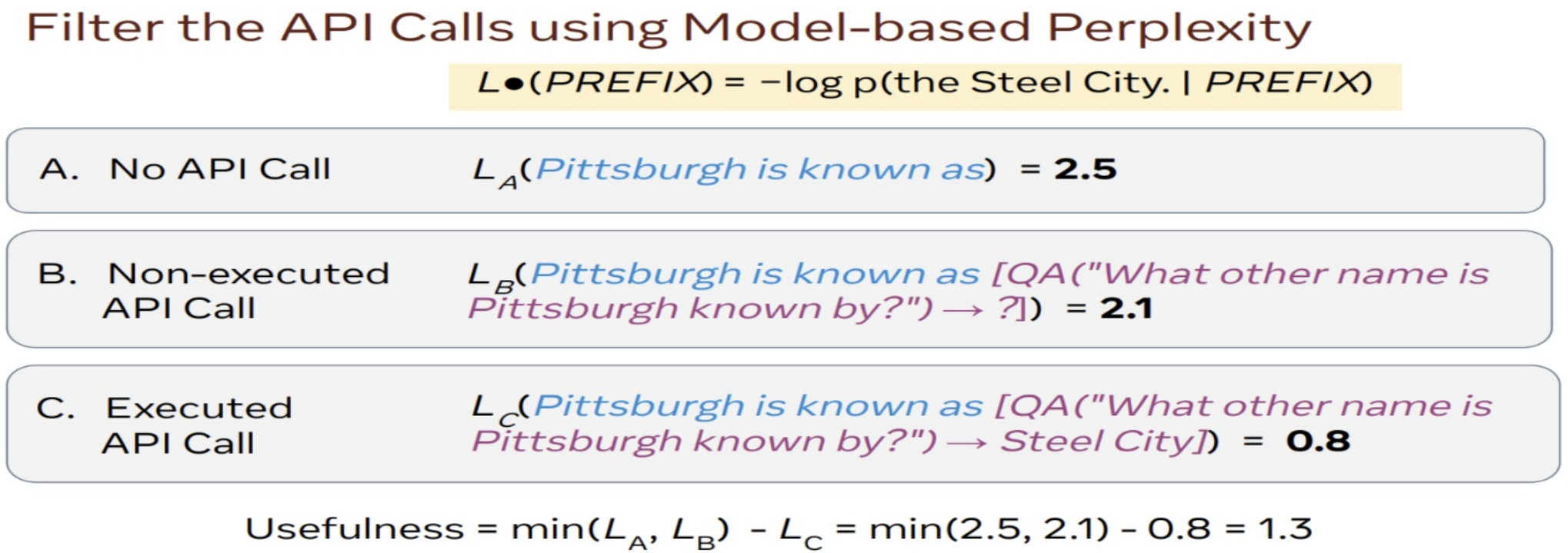
Toolformer



- Without tools, the model **guesses** — and often wrong

LITERATURE REVIEW

Toolformer

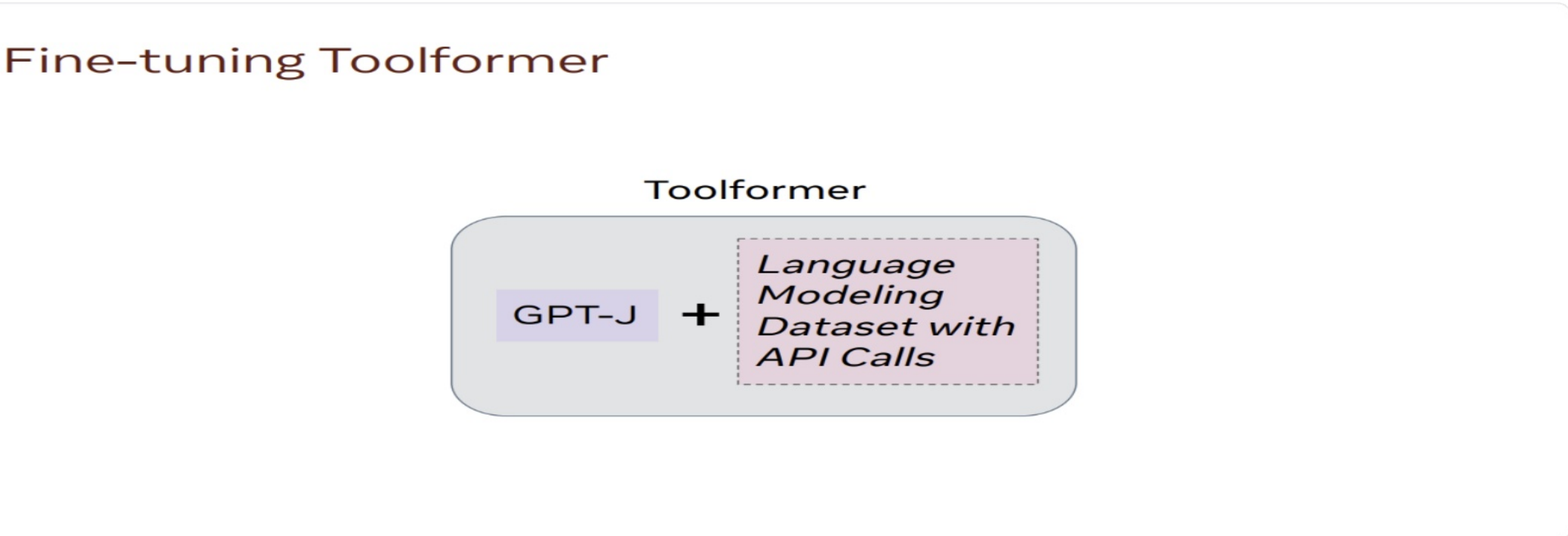


- Keep API calls where the result **measurably lowers loss** — collect those cases

LITERATURE REVIEW

Toolformer

- GPT-J + curated API-call dataset → **Toolformer**
- Fine-tuned small model **outperforms** larger base models



LITERATURE REVIEW

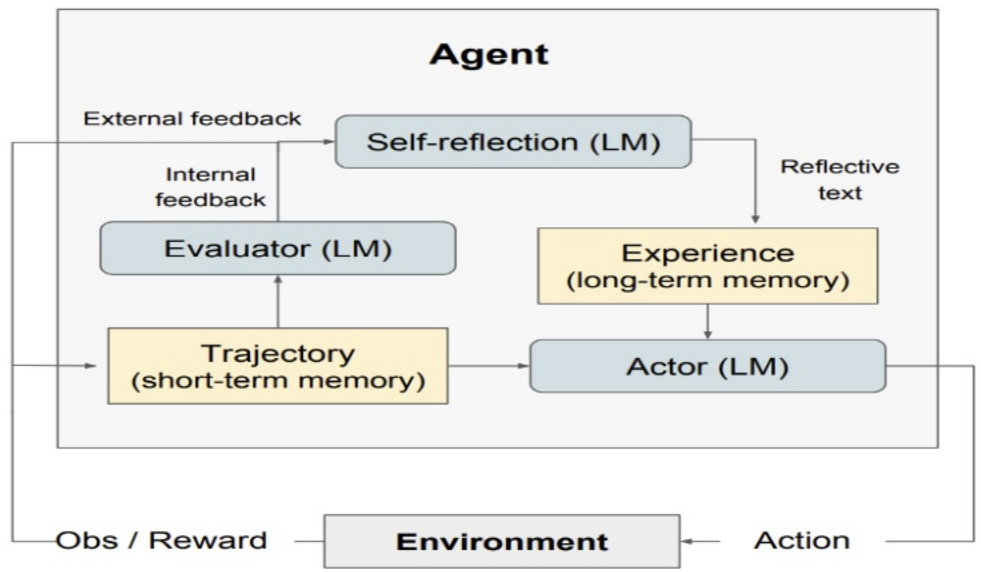
Reflexion

- Memory spans **trials** within one task — not across sessions
- Compress a failed trajectory into verbal feedback (cap $\Omega \approx 1$–3)
- Retry until success — no weight updates

Shinn et al., Mar 2023 — NeurIPS 2023

LITERATURE REVIEW

Reflexion



Algorithm 1 Reinforcement via self-reflection

Initialize Actor, Evaluator, Self-Reflection: M_a, M_e, M_{sr}
Initialize policy $\pi_\theta(a_i|s_i)$, $\theta = \{M_a, mem\}$
Generate initial trajectory using π_θ
Evaluate τ_0 using M_e
Generate initial self-reflection sr_0 using M_{sr}
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while M_e not pass or $t <$ max trials **do**
    Generate $\tau_t = [a_0, o_0, \ldots a_i, o_i]$ using π_θ
    Evaluate τ_t using M_e
    Generate self-reflection sr_t using M_{sr}
    Append sr_t to mem
    Increment t
end while
return

Figure 2: (a) Diagram of Reflexion. (b) Reflexion reinforcement algorithm

- Actor · Evaluator · Self-reflection — reflective text feeds the next trial

LITERATURE REVIEW

Voyager

- Create skills through exploration — **reuse** them on harder tasks
- Code is the action space — each skill is a callable program (a tool)
- Curriculum + iterative prompting + skill library

Wang et al., May 2023 — NeurIPS 2023

Voyager

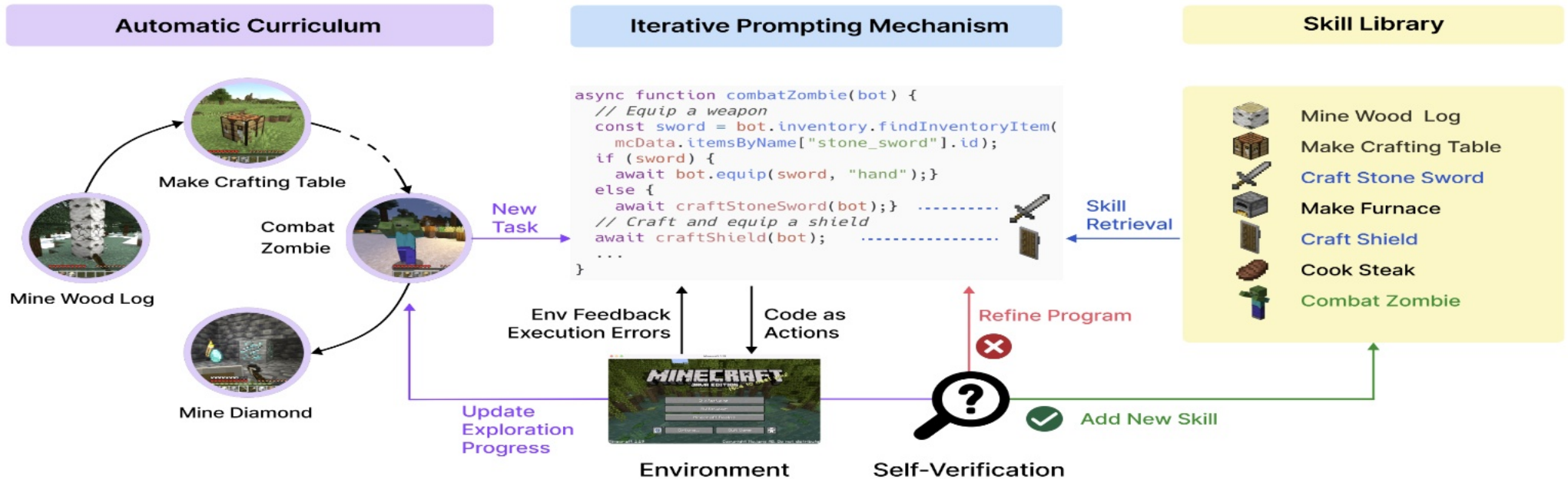


Figure 2: VOYAGER consists of three key components: an automatic curriculum for open-ended exploration, a skill library for increasingly complex behaviors, and an iterative prompting mechanism that uses code as action space.

- Curriculum explores · prompting writes code · library stores verified skills

Voyager

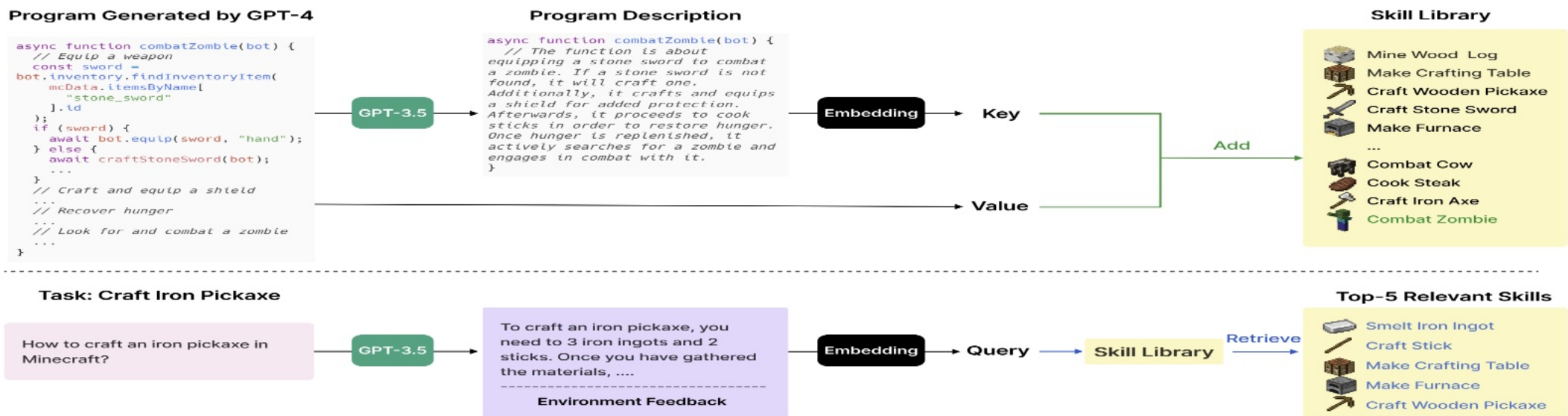


Figure 4: Skill library. **Top: Adding a new skill.** Each time GPT-4 generates and verifies a new skill, we add it to the skill library, represented by a vector database. The key is the embedding vector of the program description (generated by GPT-3.5), while the value is the program itself. **Bottom: Skill retrieval.** When faced with a new task proposed by the automatic curriculum, we first leverage GPT-3.5 to generate a general suggestion for solving the task, which is combined with environment feedback as the query context. Subsequently, we perform querying to identify the top-5 relevant skills.

- Add: program → description → embedding · Retrieve: query → top-k skills

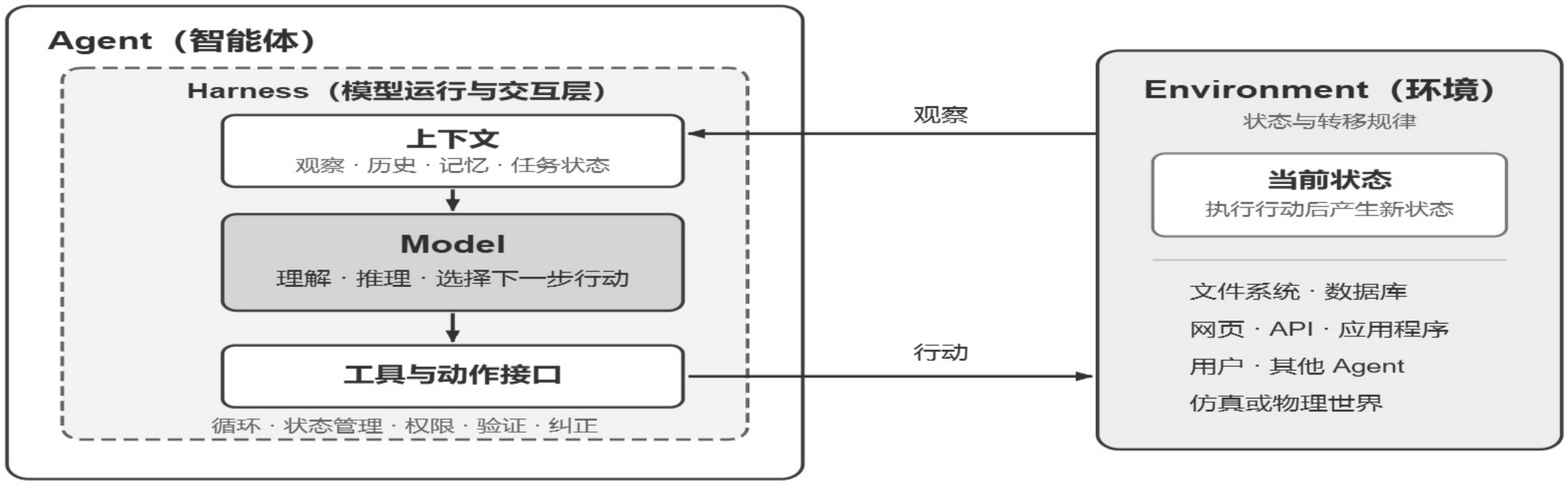
LITERATURE REVIEW

Research ↔ Industry

- Research validates ideas → absorbed into **models** and **products**
- Industry ships harnesses & APIs → feed back into **research**

AGENT ARCHITECTURE

LLM Agent Architecture



- Agent = **Model** + **Harness** ↔ Environment
- Observe → decide → act — the harness mediates every turn

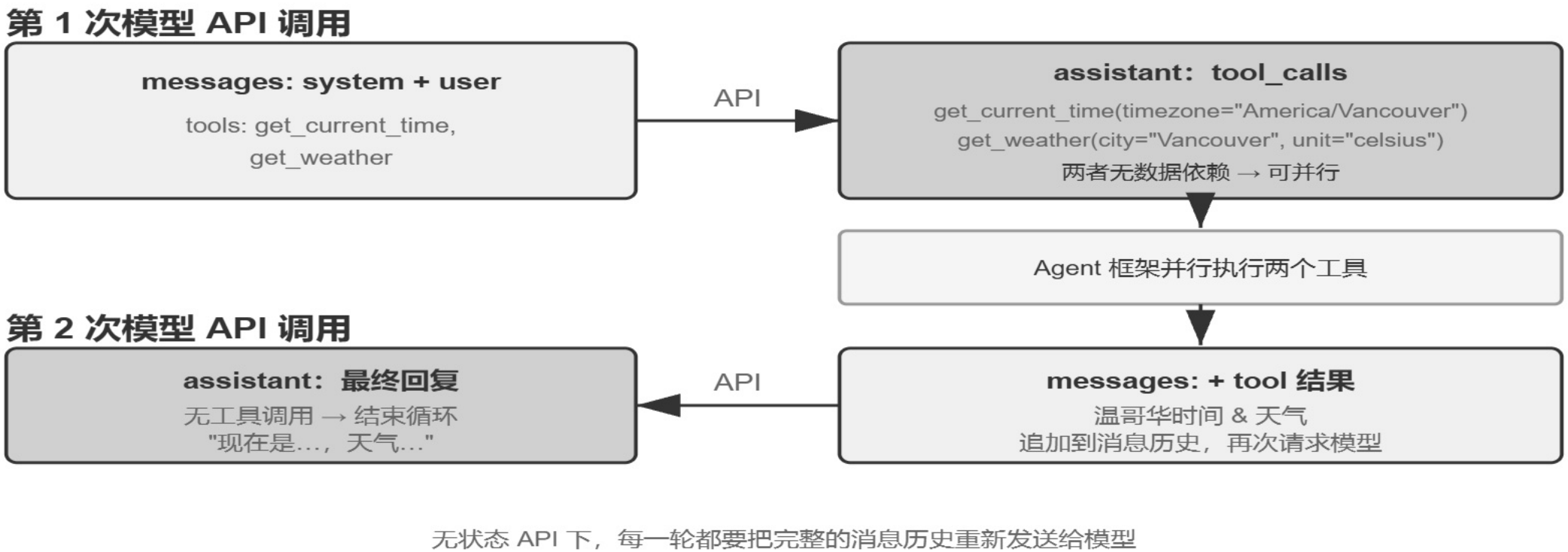
CONTEXT MANAGEMENT

Why Context Matters

- Agents need **background** the base model never saw — even a genius engineer needs context
- Context quality often beats raw **model size**

CONTEXT MANAGEMENT

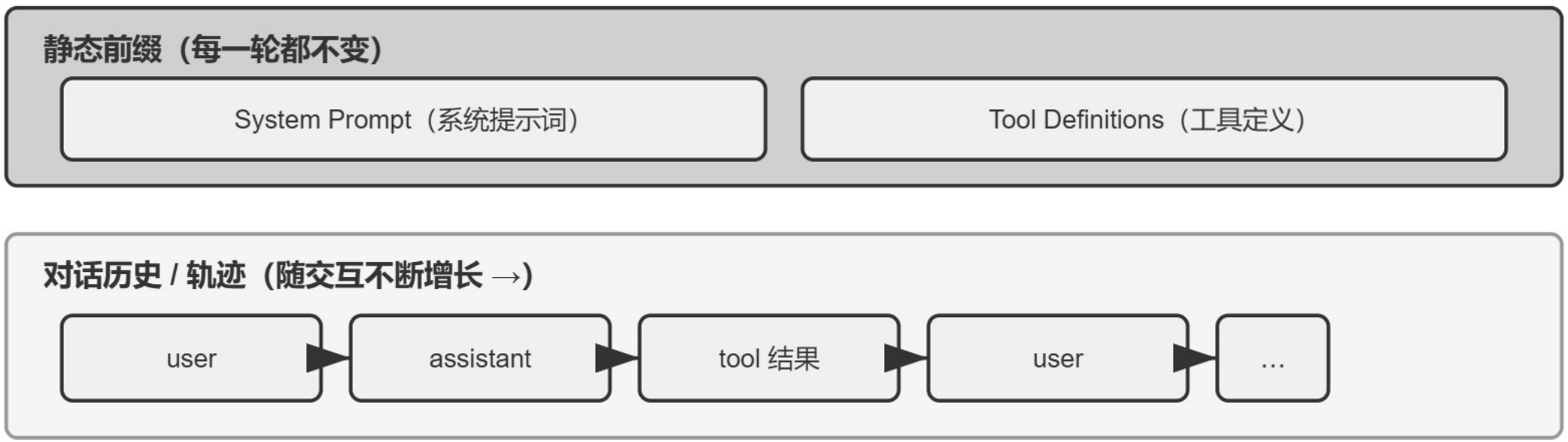
Multi-turn API Calls



- Stateless API — full message history resent every turn

CONTEXT MANAGEMENT

Context Structure



“静态前缀 + 轨迹”结构：前缀不能动（利于 KV Cache），轨迹可压缩

- Static prefix + growing **trajectory**

CONTEXT MANAGEMENT

The Context Window

- System · tools · history · reasoning · generation point

CONTEXT MANAGEMENT

Prompt Cache

- API-layer prefix reuse between calls — saves **cost** and latency
- **Stable prefix** (system + tool defs) is critical — recurs in RAG / JIT / compression

Anthropic
cache_control: { type: "ephemeral" }

OpenAI
On by default; explicit breakpoints on GPT-5.6+

Google Gemini
Implicit on 2.5+; explicit via cachedContents.create

DeepSeek
Automatic — watch prompt_cache_hit_tokens

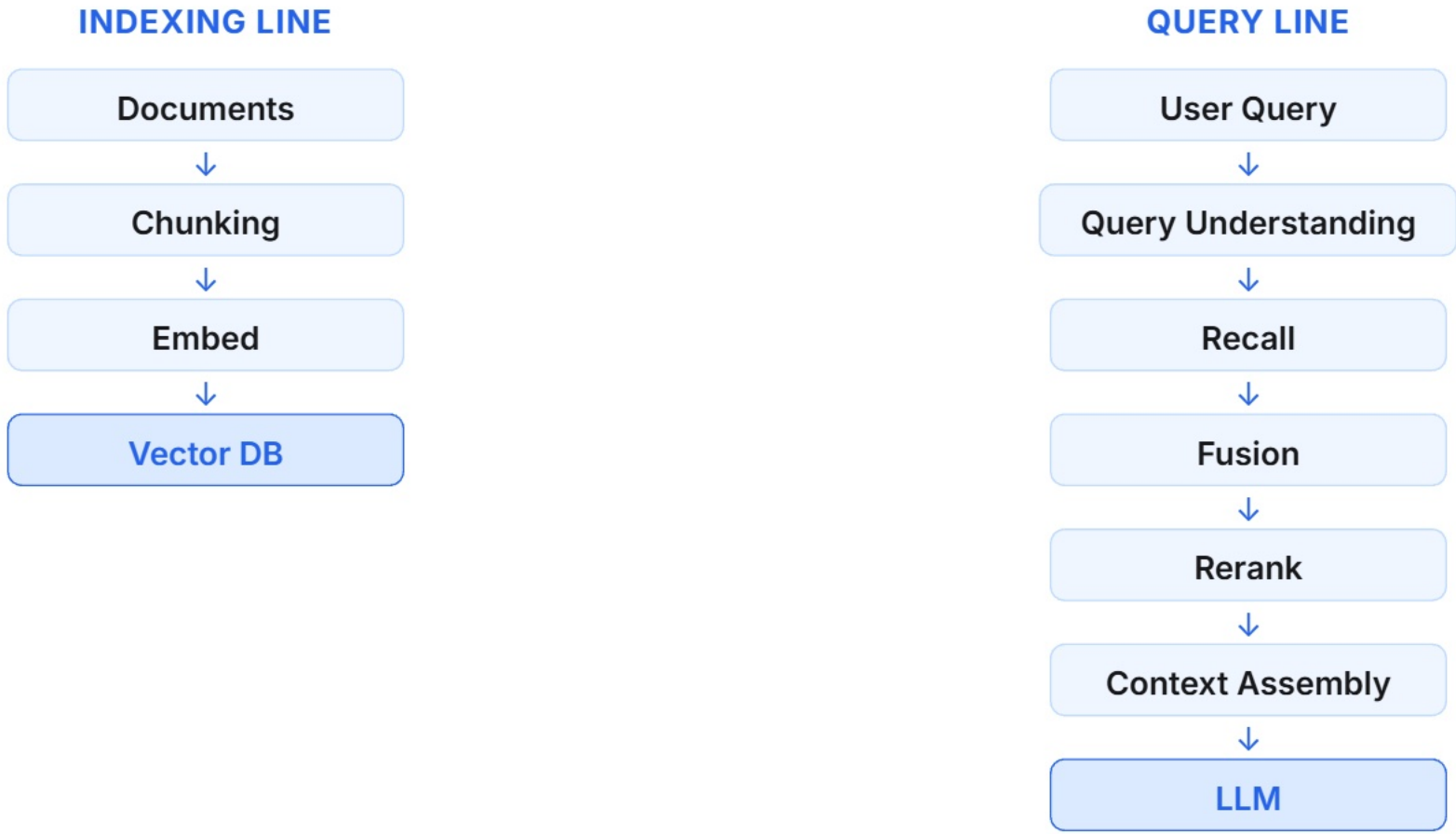
CONTEXT MANAGEMENT

Why Retrieval-Augmented Generation (RAG)?

- Millions of internal docs, finite context — surface what matters
- Example: QA bot over ~1000 Q&A txt files (not all problems need RAG)

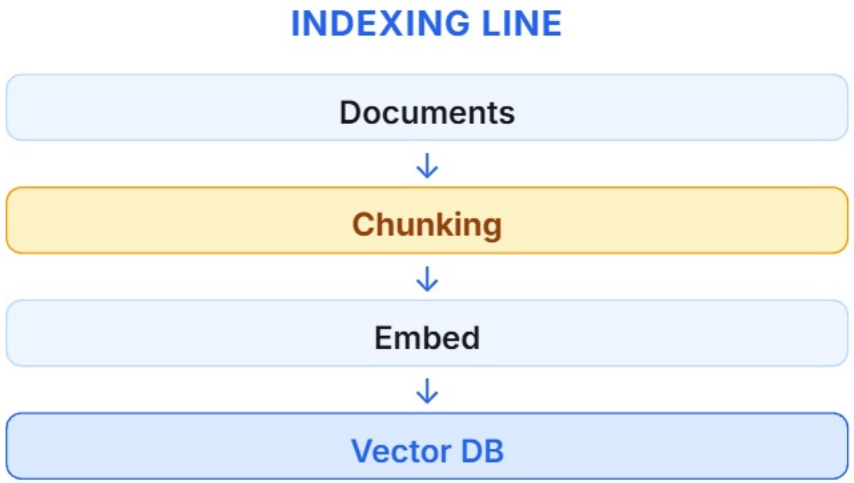
CONTEXT MANAGEMENT

Traditional RAG Pipeline



CONTEXT MANAGEMENT

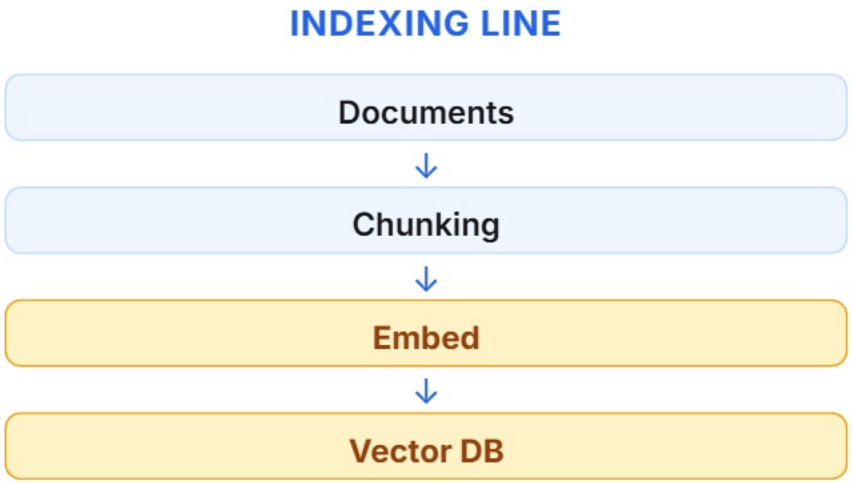
Chunking



- Fixed · structure-aware · semantic splits
- Size vs overlap: small chunks lose context; large chunks dilute retrieval

CONTEXT MANAGEMENT

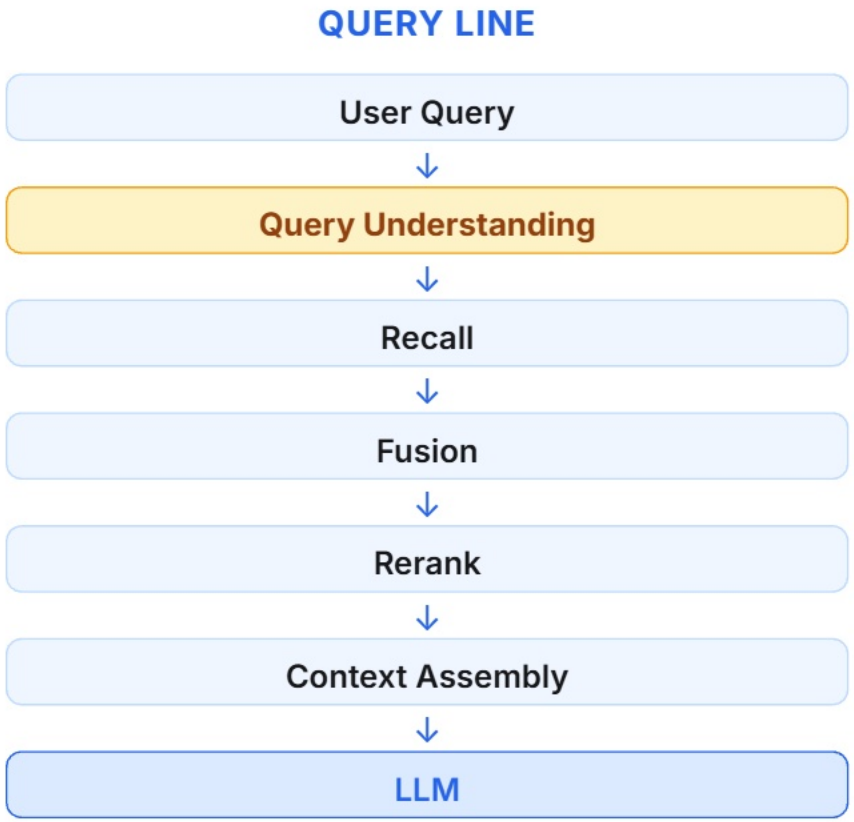
Embed & Vector DB



- Use an **embedding model** to vectorize chunks (e.g. Qwen3-Embedding, text-embedding-3-large)
- Store embeddings in a **vector DB** (e.g. pgvector, Chroma, Milvus) — offline indexing

CONTEXT MANAGEMENT

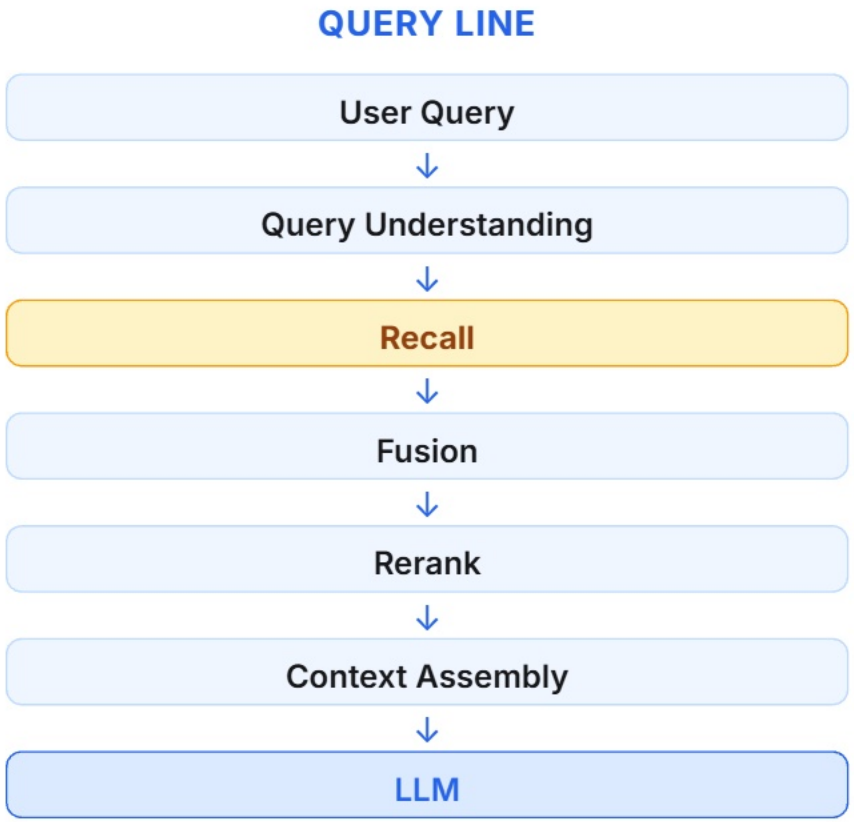
Query Understanding



- Resolve pronouns & vague tasks before recall — e.g. "finish this" → explicit task description

CONTEXT MANAGEMENT

Recall



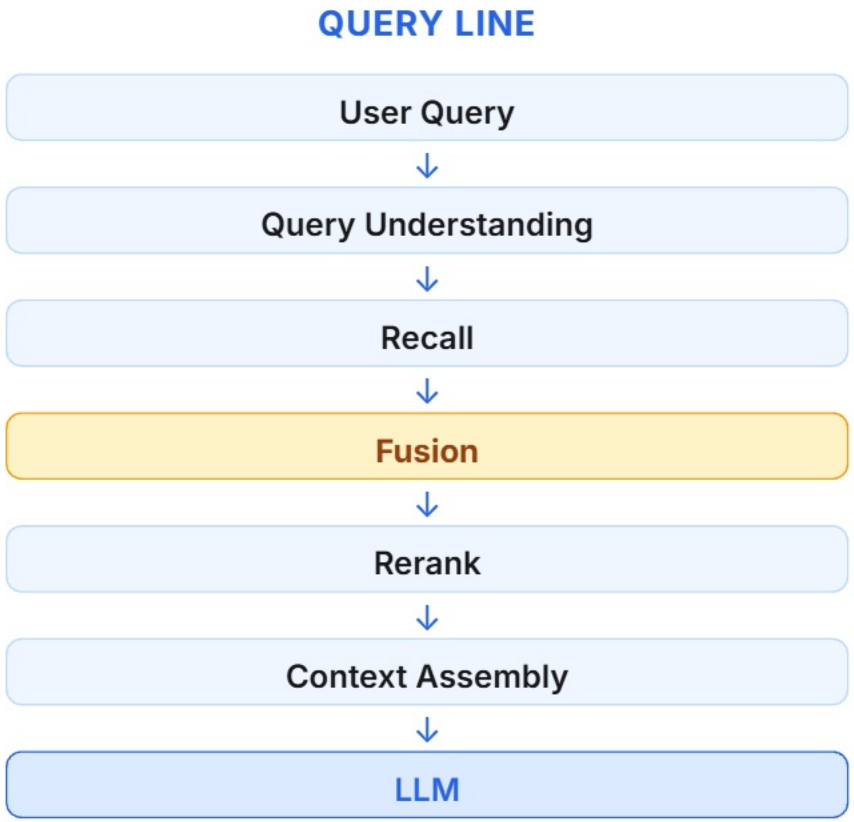
- **Dense recall** — semantic similarity via vector DB + ANN top-k; keyword-insensitive
- **Sparse recall (BM25)** — exact keywords via inverted index; weak on paraphrase

$$\text{Dense: } \text{sim}(q, d) = (E(q) \cdot E(d)) \,/\, (\|E(q)\| \, \|E(d)\|)$$

$$\text{Sparse: } \text{BM25}(q, d) = \Sigma \, \text{IDF}(t) \cdot (\text{tf}(k_1+1)) \,/\, (\text{tf} + k_1(1 - b + b|d|/\text{avgdl}))$$

CONTEXT MANAGEMENT

Fusion (RRF)

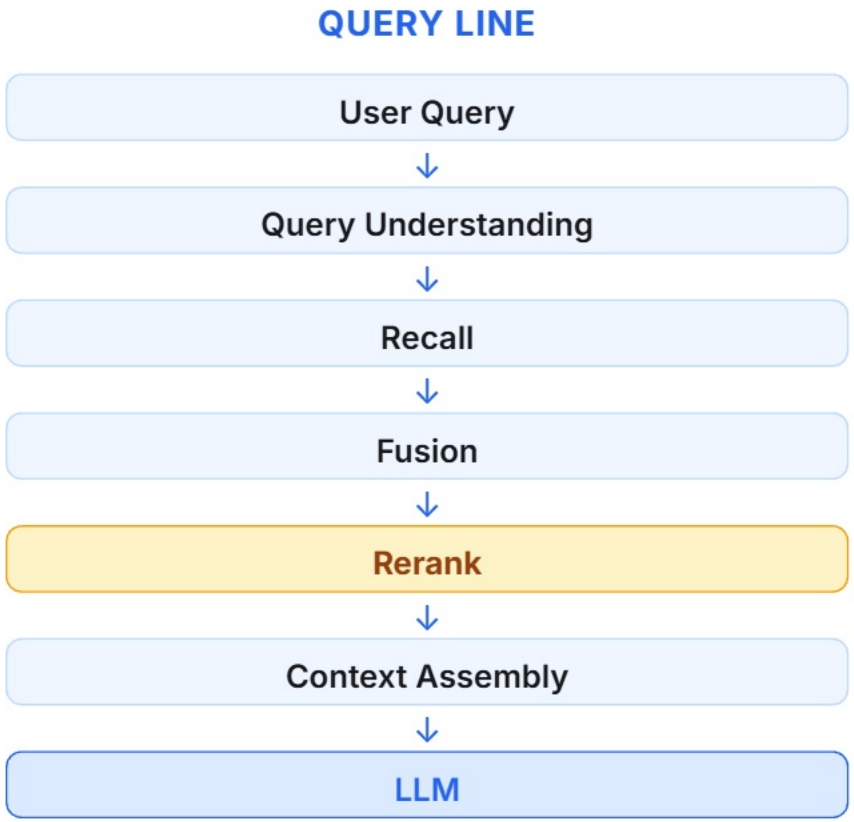


- Dense & sparse scores live on different scales — cosine ∈ [0,1] vs BM25 unbounded
- Fuse by **rank**, not weighted sum

$$\mathrm{RRF}(d) = \Sigma_i \, 1 \,/\, (k + \mathrm{rank}_i(d)) \qquad k \approx 60$$

CONTEXT MANAGEMENT

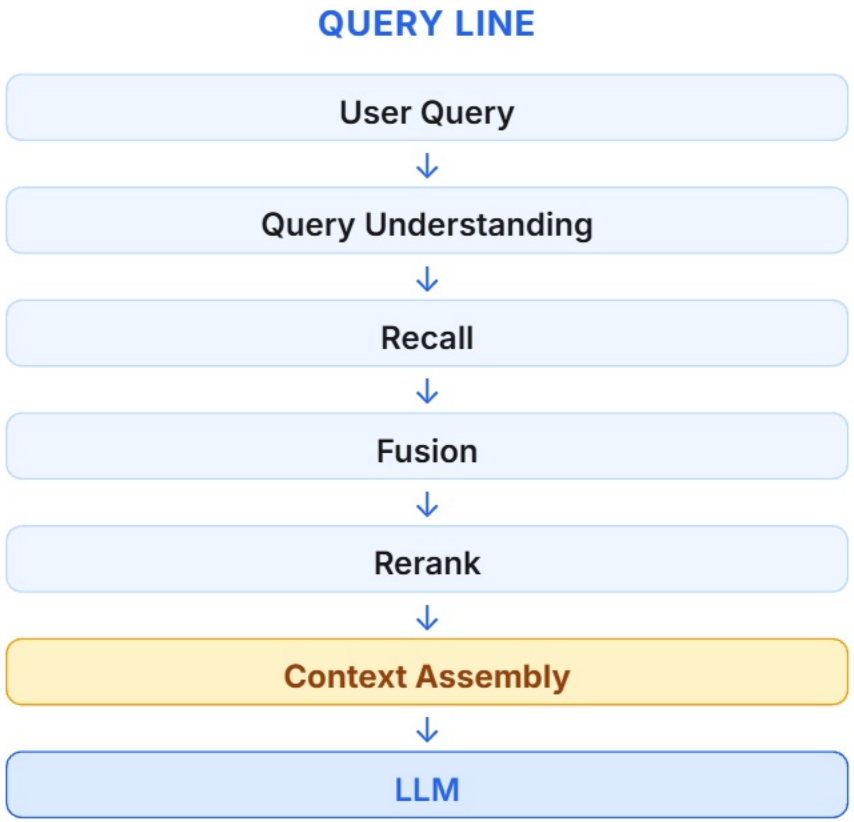
Cross-Encoder Rerank



- **Cross-Encoder** jointly scores query + doc pairs — slow but precise
- Run on a small rerank pool (e.g. top 30–50 after fusion), not the full corpus
- Models: bge-reranker-v2-m3 · ms-marco-MiniLM-L-6-v2

CONTEXT MANAGEMENT

Context Assembly



- **Dedup** — exact / near / semantic duplicate removal
- Cross-Encoder rerank → **MMR** — diversity after pointwise scoring
- **Trim** — drop low-rank chunks or truncate within budget
- **De-dump** — strip noisy raw payloads before injection
- Assemble final chunks + user query for the LLM

CONTEXT MANAGEMENT

Context Compression

- Multi-turn dialogue fills the window — must free space for new content
- Goal: compress history while preserving what the agent still needs

JUST-IN-TIME RETRIEVAL

What Is JIT Retrieval?

- Selectively load into context **during** generation — not before the model call
- Pointer-for-payload: start thin, expand on demand — complements Traditional RAG
- Append-only results preserve prompt cache; prepend/inject-at-front breaks the prefix

JUST-IN-TIME RETRIEVAL

Resident vs JIT

	High hit rate	Low hit rate
Small	Preload — core tools, system constraints, CLAUDE.md	Keep resident — retrieval latency > token cost
Large	Compress & stay — or split high-freq core	JIT — long-tail tools, skills, document corpus

JUST-IN-TIME RETRIEVAL

Agentic RAG

- Parallel to **Traditional RAG** — model decides when and what to retrieve
- Iterative: judge need → query → read results → retry if insufficient
- Dynamic knowledge during the agent loop — not a fixed pre-retrieval pipeline

JUST-IN-TIME RETRIEVAL

Search-as-a-Tool

- Primary implementation of **Agentic RAG**
- Already have a web-scale index (PageRank / search engine)? Expose search as a tool — don't duplicate a RAG stack
- Model chooses query mid-task, evaluates results, can retry

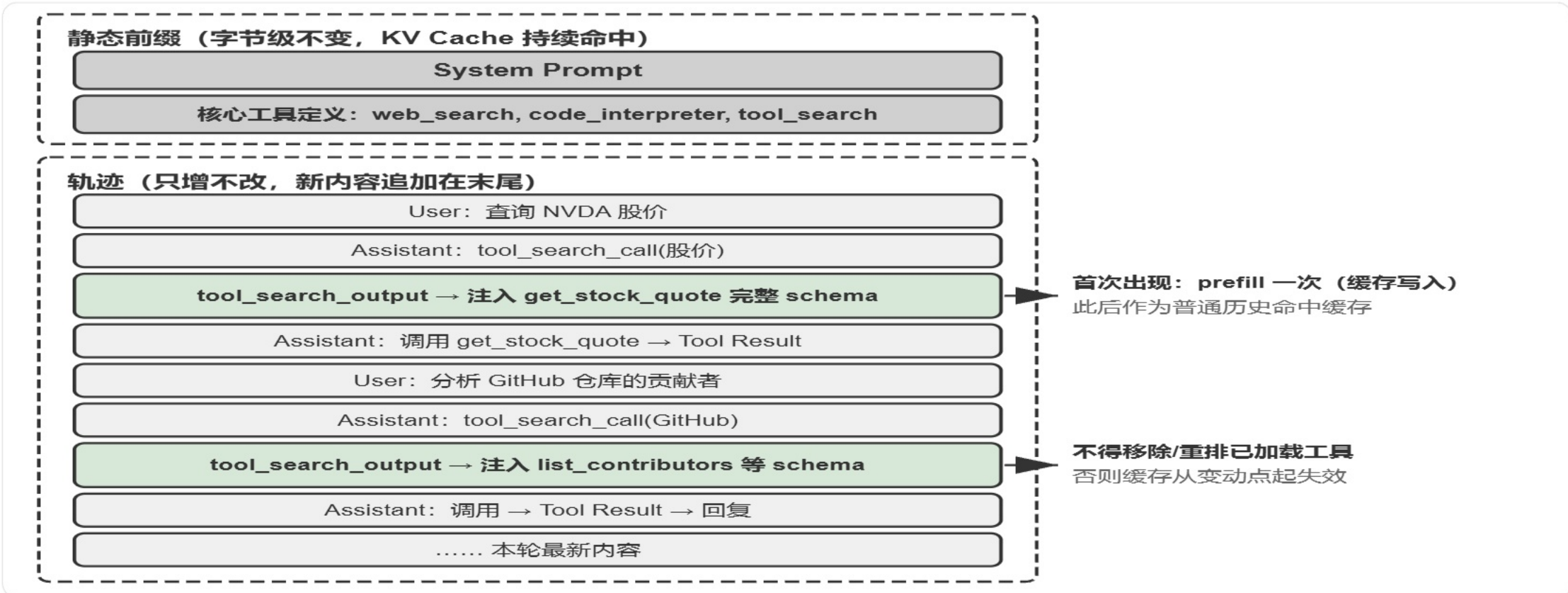
JUST-IN-TIME RETRIEVAL

Tool Retrieval

- Tool Search Tool — declare capability gap, system injects schema
- MCP-Zero: no tools in system prompt → semantic routing · one-shot pre-filter misses multi-step needs

JUST-IN-TIME RETRIEVAL

Skill Progressive Disclosure



- Thin catalog first (name + description) — full body only when invoked

JUST-IN-TIME RETRIEVAL

JIT Retrieval Compared

Dimension	Tool Retrieval	Skill Progressive Disclosure	RAG
Retrieval object	Capability / action space	Procedural knowledge (how-to)	Declarative knowledge (facts)
After loading changes	What model **can do**	**How** model does it	What model **knows**
Initiator	Model (harness may pre-filter)	Model	Classic: harness · Agentic: model
Levels	2 (name/desc → schema)	3+ recursive	1 (chunk in/out)
Addressing	Embedding → tool_reference	Filesystem path + read	embedding/BM25 black box
Prompt cache	Append after stable tools	tool_use/tool_result append-only	Front injection = disaster

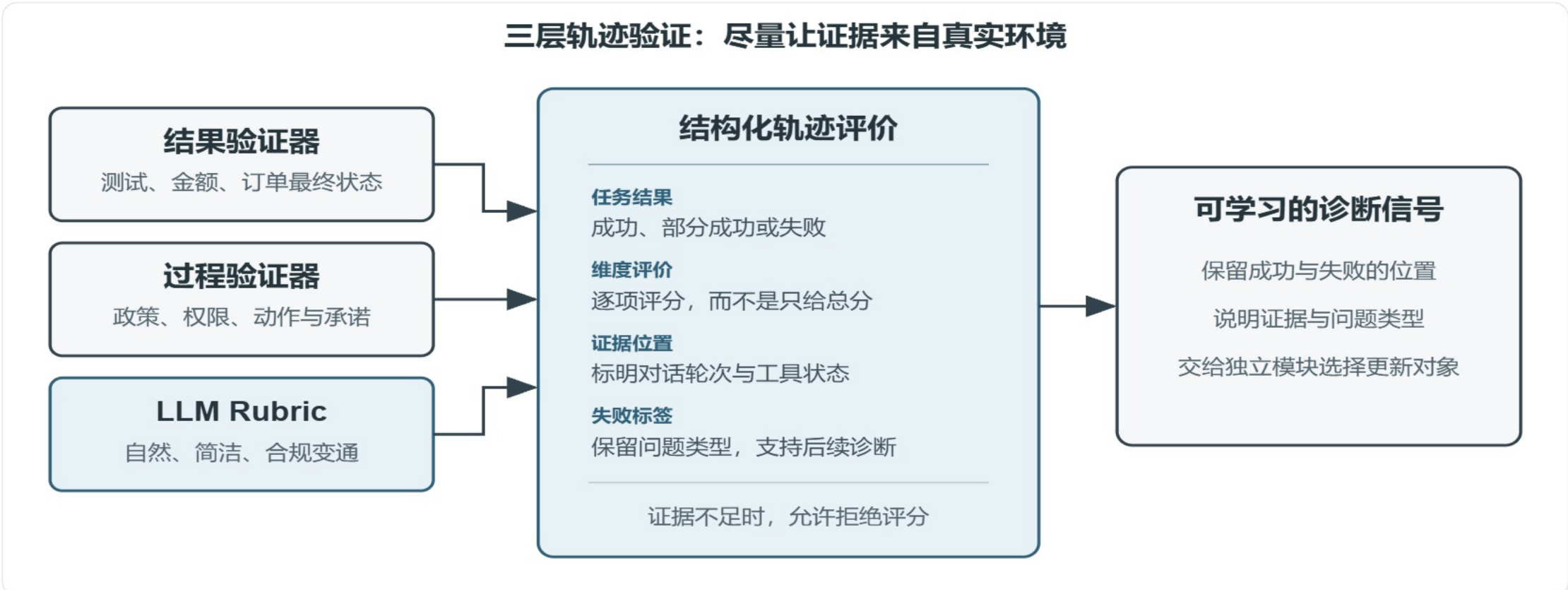
EVALUATION & EVOLUTION

Why Evaluate Agents?

- Building ≠ building **correctly** — measure before you optimize
- Model · tools · knowledge · memory · prompts · harness constraints — all design choices

EVALUATION & EVOLUTION

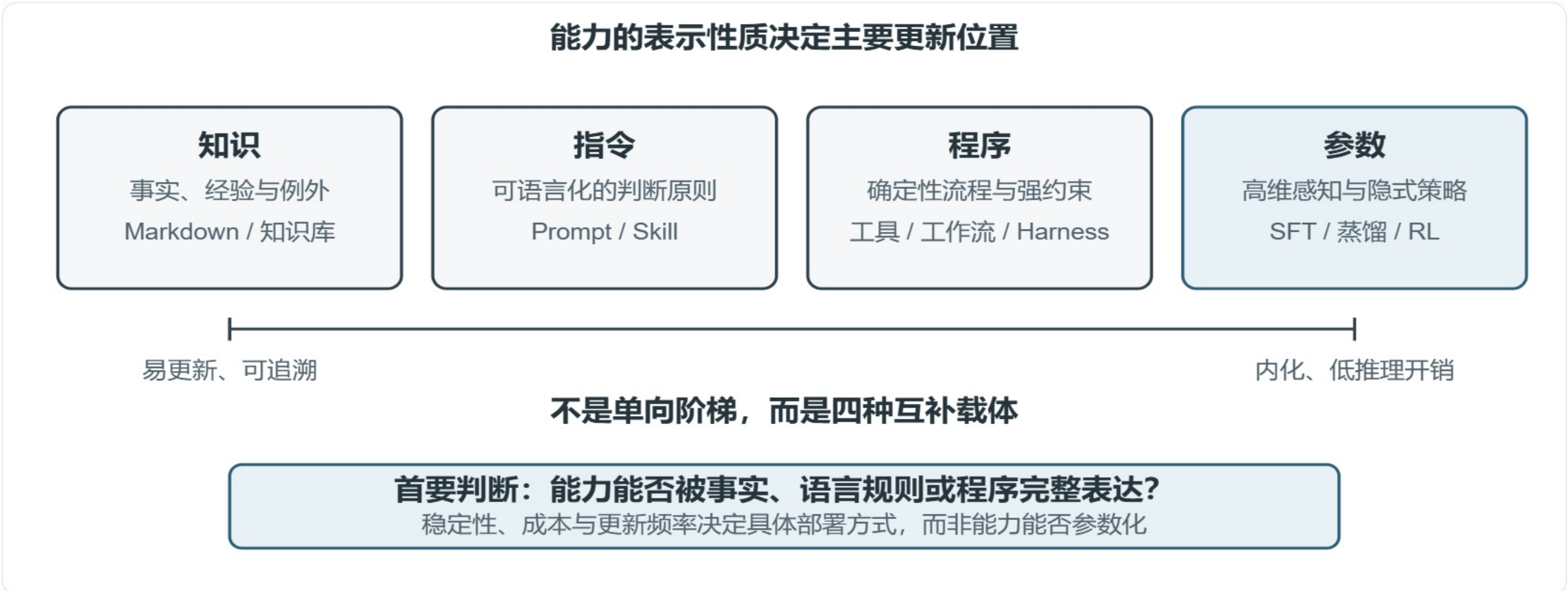
Telemetry & Benchmarks



- Trajectories are the ground truth — log, benchmark, iterate

EVALUATION & EVOLUTION

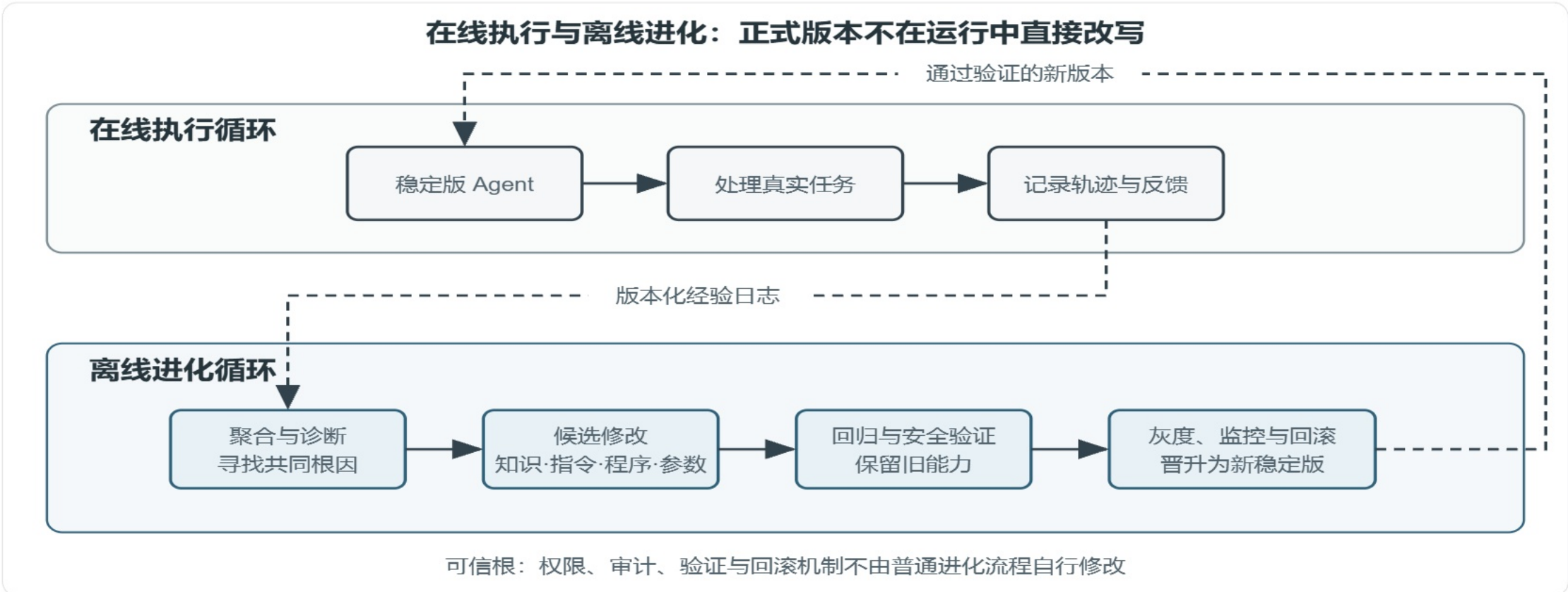
From Eval to Evolution



- Evaluation closes the loop — failures become training & product signals

EVALUATION & EVOLUTION

Continuous Improvement



- Ship → measure → learn — the agent improvement flywheel

POST-TRAINING & RL

Four Stages of Model Development

Stage	Data	Objective	What is learned	Typical cost
Pre-train	Massive raw internet text	Predict next token	Language patterns, world knowledge, basic reasoning	Extremely high (millions–tens of millions USD)
Mid-train	Target domain / capability corpora	Continue next-token prediction	Domain knowledge, language, foundational capabilities	Medium to high (depends on token scale & full-param training)
SFT	Thousands–tens of thousands of input–output demos	Next-token (loss on response only)	Instruction following, format, style, process protocols	Low (hours–days)
RL	Tasks, environment + reward signals	Maximize expected reward	Transferable decision policies, novel solutions	High (often tens–hundreds × SFT cost)

POST-TRAINING & RL

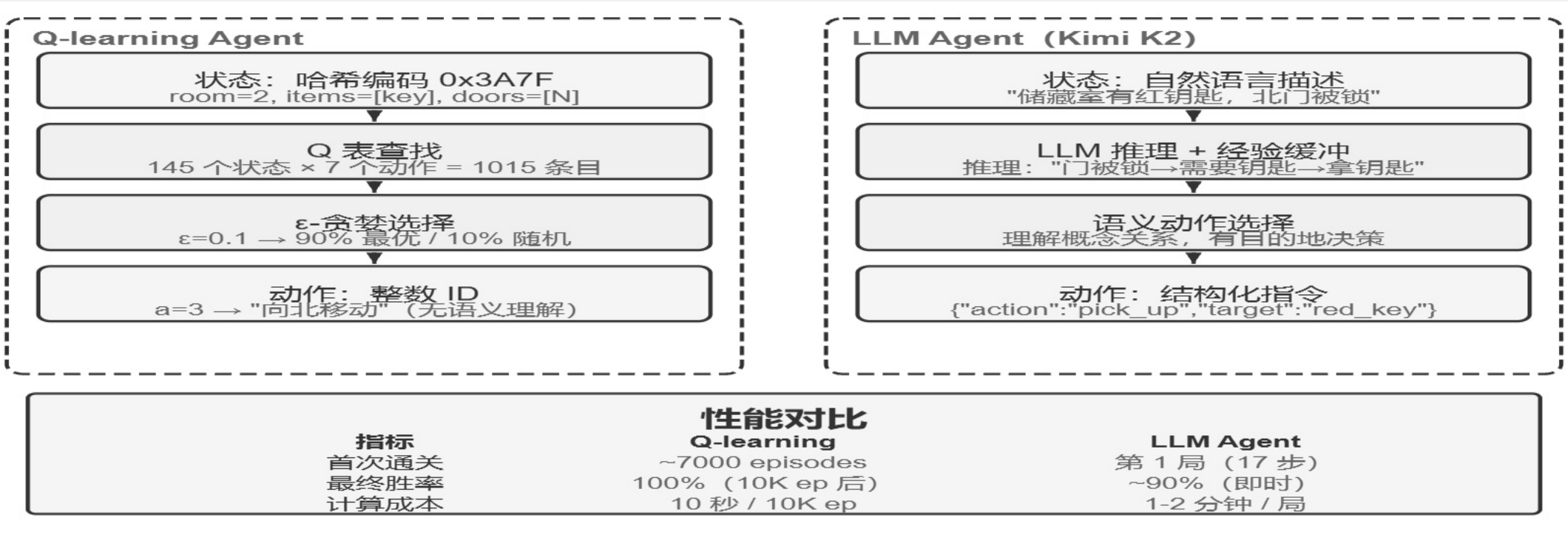
Classic RL Agent vs LLM Agent

经典 RL Agent		LLM Agent
封闭有限：{东,西,南,北,拿,攻击}	动作空间	开放无限：自然语言 + 工具调用
零先验，从随机策略开始	先验知识	海量预训练知识 + 语言游戏规则
哈希编码：state_id=0x3A7F	状态表示	语义理解："储藏室里有红钥匙"
标量奖励：R ∈ {-1, 0, +5, +50}	学习信号	奖励 + 语言反馈 + 自我反思
无内部思考（纯反应式）	思考能力	思考作为特殊动作（CoT）
单环境，规则变化需重训	泛化能力	跨任务零样本/少样本迁移
~10000 episodes / 10 秒	样本效率	~1 episode / 2 分钟

- Action space · prior knowledge · state representation · reward · internal thinking

POST-TRAINING & RL

RL: Two Agent Paradigms



- **Q-learning agent** — discrete state hash, Q-table, action IDs
- **LLM agent** — natural language state/action; policy = the model

MULTI-AGENT

Don't Blindly Multi-Agent

- Ask first: is one agent enough, or do you truly need several?
- **Information increment** is the only criterion that matters
- Sets up every engineering choice below

REFERENCE

Why Do Multi-Agent LLM Systems Fail?

Cemri et al. · NeurIPS 2025 · MAST taxonomy

arxiv.org/abs/2503.13657

MULTI-AGENT

When Is Multi-Agent Worth It?

Collaboration mode	New info?	Effect
Same model self-review (re-read own output)	No	Usually ineffective or harmful
Different agents debate same text	No	≈ single agent at equal compute
Reviewer uses **test execution** to review code	Yes (execution feedback)	Significant gain
Reviewer uses **rendered screenshots** for UI/PPT	Yes (visual feedback)	Significant gain
Reviewer uses **external tools** to verify facts	Yes (tool feedback)	Significant gain

MULTI-AGENT

Designing Multi-Agent Systems

- **How is context passed** between agents? (full trajectory vs summary vs shared files)
- **How do control and information flow?** (messages, hierarchy, bus, A2A — structure matters)

CLOSING

Further Reading

Claude Code Source Study

Claude Code architecture & harness deep dive

github.com/luyao618/Claude-Code-Source-Study

AI Agent Book

Open-source agent engineering textbook (course reference)

github.com/bojieli/ai-agent-book/tree/main

Traditional RAG vs Agentic RAG (NVIDIA)

Agentic RAG: dynamic retrieval during agent loops

developer.nvidia.com/blog/traditional-rag-vs-agentic-rag-why-ai-agents-need-dynamic-knowledge-to-get-smarter/

Thank You

Q & A