Two Clouds Over Agents: Real-time Interaction with the Environment, and Learning from Experience

Invited Talk

Bojie Li

Co-Founder & Chief Scientist, Pine AI

September 2025

Press Space for next page →

The physics knowledge is almost complete. Two small "clouds" remain over the horizon.

- April 1900, Lord Kelvin

Outline

🌩️ First Cloud: Real-time Interaction

Core Challenges

- High latency in voice interaction (tens of seconds)
- GUI operations are 3-5 times slower than humans
- Serial bottleneck of the traditional ReAct loop

Technical Breakthroughs

- **SEAL Architecture (Streaming, Event-driven Agent Loop)**
  - **Perception Layer**: Streaming processing of voice signals
  - **Thinking Layer**: Interactive ReAct with asynchronous Observation, Thinking, and Action
  - **Execution Layer**: VLA / TTS with a feedback loop

🌩️ Second Cloud: Learning from Experience

Core Challenges

- Starting each task from scratch
- Inability to accumulate domain knowledge
- Lack of improvement in task proficiency

Three Paradigms for Agents to Learn from Experience

1. **Post-training**: RL parameter updates
2. **In-context Learning**: Attention soft-updates
3. **Externalized Learning**:
   - Knowledge Base: Persistent experience storage
   - Tool Generation: Agent self-evolution

Two Clouds Over Agents: Real-time Interaction, Learning from Experience

The first problem pointed out by OpenAI research scientist Shunyu Yao: Lack of real-person interaction during the agent's task process

Shunyu Yao
姚顺雨

The Second Half

tldr: We're at AI's halftime.

Inertia is natural, but here is the problem. AI has beat world champions at chess and Go, surpassed most humans on SAT and bar exams, and reached gold medal level on IOI and IMO. But the world hasn't changed much, at least judged by economics and GDP.

I call this the **utility problem**, and deem it the most important problem for AI.

Perhaps we will solve the utility problem pretty soon, perhaps not. Either way, the root cause of this problem might be deceptively simple: **our evaluation setups are different from real-world setups in many basic ways**. To name two examples:

- **Evaluation "should" run automatically**, so typically an agent receives a task input, do things autonomously, then receive a task reward. But in reality, an agent has to engage with a human throughout the task — you don't just text customer service a super long message, wait for 10 minutes, then expect a detailed response to settle everything. By questioning this setup, new benchmarks are invented to either engage real humans (e.g. Chatbot Arena) or user simulation (e.g. tau-bench) in the loop.

Two Clouds Over Agents: Real-time Interaction, Learning from Experience

The second problem pointed out by OpenAI research scientist Shunyu Yao: Lack of a mechanism to learn from experience

- **Evaluation "should" run i.i.d.** If you have a test set with 500 tasks, you run each task independently, average the task metrics, and get an overall metric. But in reality, you solve tasks sequentially rather than in parallel. A Google SWE solves google3 issues increasingly better as she gets more familiar with the repo, but a SWE agent solves many issues in the same repo without gaining such familiarity. We obviously need long-term memory methods (and there are), but academia does not have the proper benchmarks to justify the need, or even the proper courage to question i.i.d. assumption that has been the foundation of machine learning.

Part One: Agent Real-time Interaction with the Environment

Interaction with:

- **Humans**: Conversation and collaboration through real-time voice.
- **The Internet**: Operating computers, browsing websites, using software.
- **The Physical World**: Controlling robots, interacting with the real environment.

Real-time Interaction Challenges for Voice Agents

Conflict: Serial Processing vs. Real-time Demand

- **Must wait**: Must finish listening before thinking, and finish thinking before speaking
- **Blocking wait**: Each step becomes a bottleneck
  - User finishes speaking (VAD) → Speech Recognition (ASR) → Complete sentence
  - Complete sentence → LLM with thought process → Complete output after thinking
  - Complete output → Split into sentences → Text-to-Speech (TTS) → Voice response
- **Cumulative latency**: Total latency far exceeds human tolerance

Dilemma of Fast and Slow Thinking

- **Deep thinking**: High-quality reasoning, but CoT takes 10+ seconds
  - User: *"Help me book a flight"*
  - Agent: * (thinking for 10 seconds)...* User has already lost patience
- **Quick response**: Latency <1 second, but prone to errors
  - Agreeing to an inappropriate plan without thinking
- **Key issue**: Unable to anticipate and think while listening

Technical Bottlenecks: Waiting at Every Step

Perception Stage

- Voice: High latency from waiting for the whole sentence to finish before transcribing; fragmented audio fed to the ASR model leads to low recognition accuracy
- Vision: High prefill latency for 2K token screenshots

Thinking Stage (5-20 seconds)

- Must receive the complete input before starting to think
- Cannot anticipate user intent
- Test-time scaling makes latency even worse

Execution Stage

- Starts acting (speaking, operating mouse/keyboard) only after thinking is complete
- Each step in GUI operation requires a new screenshot and thought process

Our Architecture: SEAL (Streaming, Event-driven Agent Loop)

Core Idea: Abstract all interactions as asynchronous event streams to achieve low-latency, interruptible real-time interaction

1. Perception Layer

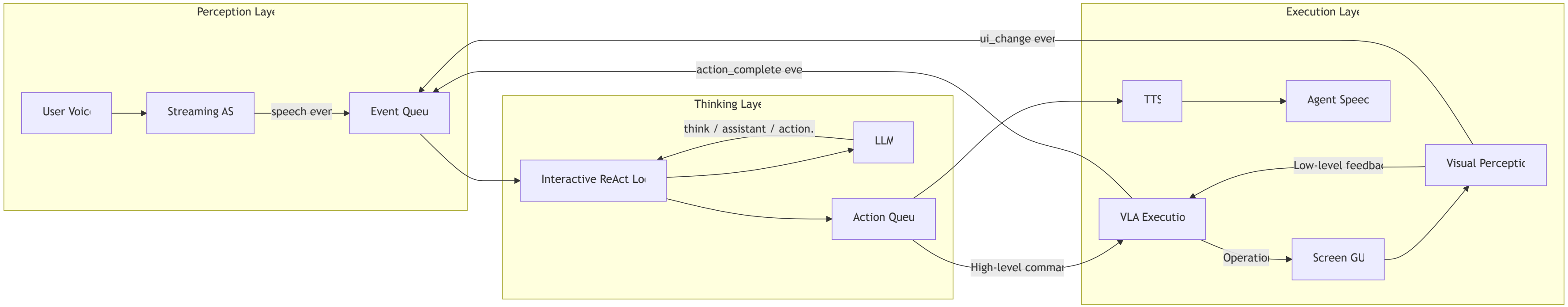
Convert continuous real-world signals (voice, GUI video) into a stream of discrete events.

2. Thinking Layer

Process events asynchronously, enabling thinking while listening and speaking while thinking, generating an interleaved sequence of thoughts and actions.

3. Execution Layer

Convert discrete action commands back into continuous real-world signals (TTS speech, mouse movements).



Layer 1: Perception Layer

Converting continuous real-world signals into discrete events

Input

Continuous signals: voice stream, GUI video stream

Output

Discrete events: `speech_start` , `interrupt` , `laugh` , `speech_fragment` , `ui_change` , etc.

Streaming Speech Perception Model: Replacing VAD + ASR

Problems with the traditional VAD + ASR cascaded architecture

Limitations of VAD + ASR

- **Latency accumulation**: VAD detection latency + ASR processing latency
- **Information loss**: VAD only outputs a binary signal, losing acoustic details
- **Error propagation**: VAD models are small and often misjudge, leading to ASR missing or mis-triggering recognition
- **Lack of context**: ASR recognizes speech in segments, destroying semantic continuity and leading to low speech recognition accuracy

Problems in practical scenarios

- **Interruptions are not intelligent enough**: Any sound, even the other person's simple "uh-huh", interrupts the AI's speech, making the interaction unnatural
- **Low accuracy in speech recognition due to lack of context**: Error rates are particularly high for contexts requiring information like email addresses, brand names, personal names, and domain-specific terms
- **Loss of emotional information**: Non-verbal information in speech like sighs and laughter
- **Loss of background information**: Inability to perceive whether the environment is noisy or if there is background music

Streaming Speech Perception Model: Replacing VAD + ASR

Streaming Speech Perception Model based on Open-Source Autoregressive LLM

- Unlike traditional ASR models like Whisper, it uses an autoregressive architecture to reduce speech recognition latency
  - Processes incoming speech tokens in a streaming manner
  - Outputs text and acoustic events in a streaming manner
- Post-trained on an open-source LLM
  - Retains conversational context, supports in-context learning, significantly improving recognition accuracy for user’s personal information, domain-specific terms, etc.
  - Possesses world knowledge and common sense, significantly improving recognition rates for brand names, amounts, etc.

Rich output information, including not only text but also acoustic events

Text Tokens: Real-time transcribed text fragments

Special Tokens (Acoustic Events):

- `<speak_start>` User starts speaking
- `<speak_end>` User finishes speaking
- `<interrupt>` Interruption intent detected
- `<emotion:happy>` Emotion marker
- `<laugh>` `<sigh>` Paralinguistic information
- `<music>` Environmental sound information

The Myth of End-to-End Speech Models

Modality Conflict: Why "End-to-End" Can Be Worse?

Apparent Advantages vs. Actual Problems

- **In theory**: End-to-end speech models should be better than ASR + LLM + TTS
- **In practice**: LLMs processing audio tokens directly often perform poorly
- **Root cause**: Parameter divergence due to modality conflict

Manifestations of Modality Conflict

- **Parameter divergence**: Some experts in MoE specialize in handling speech
- **IQ drop**: Reasoning ability in speech tasks significantly decreases
- **Training difficulty**: Multimodal post-training can easily damage original capabilities

Layer 2: Thinking Layer

Achieving interruptible, asynchronous think-while-listening and speak-while-thinking based on an event-driven loop

Input

Stream of discrete events (from Event Queue)

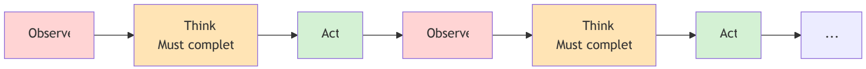
Output

Interleaved thoughts and action commands (to Action Queue)

Core Innovation: Interactive ReAct

Core Idea: Breaking the rigid loop to achieve flexible interleaving of observation, thinking, and action

Traditional ReAct: Rigid OTA Loop

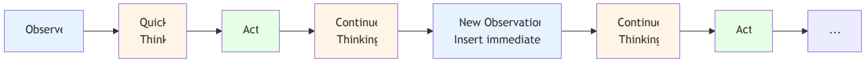


Limitations:

- **Fixed loop**: Must complete the entire OTA cycle
- **Thinking easily interrupted**: New input causes thought loss
- **High response latency**: Falls silent during deep thought

- O = Observation
- T = Thinking
- A = Action

Interactive ReAct: Flexibly Interleaved OTA



Advantages:

- **Think while listening**: New observations can be inserted at any time
- **Speak while thinking**: Respond immediately after quick thinking
- **Unified context**: Maintains continuity of thought

SEAL Thinking Layer: Interruptible Interactive ReAct Loop

Key Insight: LLM thinking speed far exceeds voice I/O, fully utilize the "gap time"

🧠 **LLM Processing Speed**

- Input processing: **500+ tokens/second**
- Thinking/Output: **100+ tokens/second**

🗣️ **Voice I/O Speed**

- Voice input/output: **Only 5 tokens/second**
- Speed difference: **20-100 times**

In the "gap time" outside of observation (voice input) and action (voice output), we have ample time for deep thinking! The rigid observe-think-act loop just doesn't fully utilize this gap time!

⚡ Fast Thinking → 🐢 Slow Thinking → 🐌 Continuous Thinking

1. **Quick Response (0.5s)**: 50 tokens of quick thinking → Immediate preliminary response (within 5s)
2. **In-depth Analysis (after 5s)**: 500 tokens of slow thinking → Generate a more complete answer
3. **Continuous Thinking (as needed)**: If 500 tokens are still not enough, continue thinking for 5 more seconds → Continue generating the answer until thinking and speaking are complete. If multiple rounds of thinking are needed, the result is to continuously output a summary of the current round of thought, like a person "thinking out loud".

Interactive ReAct: Think while Listening

Gracefully handling interruptions in conversation

Traditional ReAct: As soon as the user interrupts, all previous thoughts are discarded, and it has to start over.

Interactive ReAct: Retains the interrupted thought process, appends the new user input, and lets the model continue thinking from where it left off.

```
<user>I want to change my plan from the current $109 to your new... </user>
<think>The user wants to change their plan, currently on the $109 plan.
Let me check the information on the new plan...
Need to know: 1) User's current plan details 2) New plan price...<interrupted/></think>
<user>(interrupts) By the way, is the new plan the one that's $79 a month?</user>
<think>(Continues previous thought) The user confirmed the new plan price is $79.
This will save $30 per month, going from $109 to $79.
I still need to confirm: 1) Differences in plan content 2) Any contract restrictions...</think>
<assistant>Yes, the $79 plan. Let me confirm, your current $109 plan includes...</assistant>
```

Advantage: The thought process is coherent and can quickly adjust strategy based on the latest information.

Interactive ReAct: Speak while Thinking

Using "filler words" to buy time for deep thought and reduce first-word latency

Scenario: A user asks a complex question, and the agent needs time to think.

Traditional ReAct:

```
<user>Do you confirm ordering this plan?</user>
* (a long silence of up to 10 seconds...) *
<assistant>After consideration, I confirm the order.</assistant>
```

Interactive ReAct:

```
<user>Do you confirm ordering this plan?</user>
<think> (quick thought, <0.5s) </think>
<assistant>Let me just confirm, this is the $79 a month plan, right?</assistant> (preliminary response)
<think> (in-depth thought) </think>
<assistant>Yes, this plan is a great deal. I confirm the order!</assistant> (final answer)
```

Advantage: Greatly improves the fluency of the interaction, avoiding awkward long waits.

Interactive ReAct: Detailed Sequence Example

A complete conversational sequence of deep thinking

```
<user>Do you confirm ordering this plan?</user>
<think>This is a serious question, I need to use a filler phrase first, then think deeply.</think>
<assistant>Let me just confirm, this is the $79 a month plan, right?</assistant>
<think>Let me think deeply. The user's original plan was $109, now it's down to $79.
The user's Wi-Fi speed is reduced from 800 Mbps to 400 Mbps,
but the user said they are okay with the speed reduction.<interrupted></think>
<user>Yes, $79 a month.</user>
<think>The customer service representative confirmed this information, but I need more time to think,
so I need another filler phrase.</think>
<assistant>Let me think...</assistant>
<think>Continuing to think, the plan offered by the customer service includes... it looks pretty good overall,
but to be safe, I need to confirm the plan information again.</think>
<assistant>This plan looks good, but I'd like to confirm again, besides the Wi-Fi speed,
do all other services remain unchanged?</assistant>
<user>Yes, all other services remain unchanged.</user>
<think>Looks good, I should proceed with the booking.</think>
<assistant>Great, I'll book it now!</assistant>
```

Interactive ReAct: Interruption Handling Example

Interruption and resumption in real-time conversation

```
<user>This is Alaska Airlines, how can I help you?</user>
<think>Finally past the IVR and connected to a real person. I need to briefly describe the user's needs.</think>
<assistant>I'd like to book a flight from San Francisco to Seattle, for the price of<interrupted></assistant>
<user>What date are you planning to depart?</user>
<think>The customer service interrupted me. I need to state the user's departure date.
I found in the context that it's September 1st.</think>
<assistant>Departing on September 1st, one way.</assistant>
<user>Let me check that for you, please hold.</user>
<think>I need to wait for the operation to complete.</think>
<assistant><wait></assistant>
<user>[SYSTEM HINT] There has been no voice from the other party for 10 seconds.</user>
<think>The customer service is checking for flights, I can ask for an update.</think>
<assistant>Have you found anything?</assistant>
<user>I've found 10 flights so far. What time would you like to depart?</user>
<think>This is a simple question. The user wants the cheapest option, and if the prices are similar, a morning departure.</think>
<assistant>I'm looking for the cheapest price<interrupted></assistant>
<user>Economy or business class?</user>
<think>The customer service interrupted me. I need to describe my price preference.</think>
<assistant>Economy class, as cheap as possible. If prices are similar, I'd prefer a morning departure.</assistant>
```

Note: Only the most recent turns of thought are kept to save context. For older turns, only <user> and <assistant> are retained.

Training an Adaptive Thinking Model for Interactive ReAct

Training Method

Label Design:

- Multi-turn alternating <think> and <assistant> tags
- Support for interruption and continuation of thought

Data Source:

- Use Claude 4 Sonnet to generate training data
- Avoid using OpenAI/Gemini models (thought processes are summarized)

Data Quality Control:

- Diverse thinking scenarios
- Balance of simple and complex questions
- Include interruption and resumption cases

Optimization Goals

Core Capabilities:

- Quickly determine if deep thinking is needed
- Natural filler words and time management
- Avoid hallucinations on simple questions

Performance Metrics:

- First-word latency < 500ms
- No degradation in thought quality
- Significant improvement in user experience

Interactive ReAct: Data Engineering Essentials

Thought Length Control

1. **First-turn thought**: Strictly control within 50 tokens (~0.5 seconds)
2. **Simple question optimization**: Minimize thought time, complete in a single turn
3. **Thought length as part of the RL reward function**

Scenario-based Handling

- **Simple greetings**: Respond directly without deep thought
- **Complex decisions**: Generate appropriate filler words + deep thought
- **Multi-turn dialogue**: Maintain context coherence

Interruption Handling Capability

1. **Thought interruption recovery**:
    - Utilize existing thought content
    - Quickly generate an appropriate response
    - Continue the unfinished reasoning
2. **Speech interruption handling**:
    - Record the point of interruption
    - Understand the other party's interjection
    - Naturally connect to the subsequent conversation

Layer 3: Execution Layer

Converting discrete action commands into continuous real-world signals

Input

Discrete action commands: `speak(...)`, `click(...)`

Output

Continuous signals (voice waveform, mouse trajectory)

The "Last Mile" Problem in Computer Use Agents

Moravec's Paradox in Agents

The model "knows" what to do

- **Powerful world knowledge**: Multimodal models can accurately understand screen content and clearly describe the operational goal.
- **Example**: "Click that blue 'Submit' button."

The model "can't do" it

- **Clumsy actions**: Difficult to accurately output click coordinates or perform precise actions.
- **Action Space conflict**: Pre-training data is mostly text, lacking data to map high-level instructions to precise physical coordinates.

Why the Inaccuracy?

Difficulty in outputting coordinates

- **Unfriendly to LLMs**: Requiring an LLM to directly output `(x, y)` coordinates is like asking a human to state the precise coordinates of a point on the screen out of thin air—very difficult and unnatural.

Limitations of "Bounding Box" Assistance

- **Method**: Number all clickable elements on the page and have the LLM output the number.
- **Problem**: For complex interfaces, there can be hundreds of boxes (e.g., Gmail interface, where every email and category is a clickable area), making the screen cluttered and unusable. Some software cannot even have bounding boxes drawn (e.g., document editors, spreadsheet editors, drawing software, video editing software).

Solution: Train an End-to-End VLA Model

Core Idea: Borrow VLA models from the robotics field and post-train the model through RL to directly output actions.

Option 1: Main model directly outputs mouse click coordinates

- **Task**: Given a screenshot and the full agent context, the LLM directly outputs click coordinates, such as `click(x,y)`.
- **Training**: Use the LLM's coding ability to generate a massive number of test pages for RL.
- **Pros**: A single model completes the click action end-to-end, with lower operational latency.
- **Cons**: No mouse movement trajectory, similar to web crawlers, easily identified as a bot; model is sensitive to screen resolution; requires RL training of a larger main model, which is costly.

Option 2: Train a separate VLA model to imitate human mouse movement patterns

- **Imitate human operation**: Adopt a "move-finetune-click" closed-loop feedback model.
- **Architecture**: The action output by the main model describes the desired click location or operation in natural language. The VLA model generates the specific mouse movement or click action and quickly adjusts the next action based on the relative position of the mouse pointer and the element.
- **Requirement**: A low-latency (~100ms) VLA model that can quickly adjust the mouse pointer position based on natural language instructions like "click the product in the third row, second column" or "click the search button" until the operation is complete.
- **Pros**: Can pass most CAPTCHAs based on mouse trajectory detection; not sensitive to screen resolution; the VLA model only needs to perform grounding tasks, can be smaller, and training cost is lower.
- **Cons**: The main model needs to output intermediate natural language expressions, which are then handled by the smaller VLA model, leading to higher latency; intermediate expressions can be ambiguous, potentially reducing operational accuracy.

How Text-to-Speech Passes the Turing Test

Core Insight: Human "imperfection" stems from the limits of thinking speed

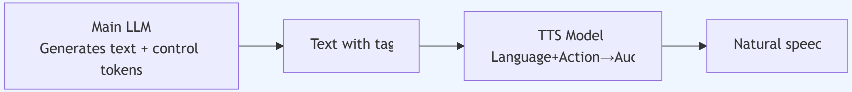
🧠 Natural Patterns of Human Speech

- **Thinking speed < Speaking speed**
- When unsure what to say next → naturally produces:
  - Pauses (thinking time)
  - Filler words ("um," "uh," "like")
  - Repetitions ("this, this...")

🤖 The "Too Perfect" Problem of AI

- **Thinking speed >> Speaking speed**
- Always generates clean, complete sentences → leading to:
  - Overly fluent (no pauses)
  - Zero filler words
  - Immediately reveals it’s a machine

Solution: Main LLM generates cognitive pauses, then TTS generates speech



- **Main LLM**: Decides where to pause
  - Sentence transitions: insert `[THINKING]`
  - Searching for words: insert `[SEARCHING]`
  - When uncertain: insert `[UNCERTAIN]`
- **TTS Model**: Multimodal generator
  - Input: Text + control actions
  - Output: Corresponding audio

Main LLM Generates Control Tags → TTS Renders

Architectural Division of Labor: LLM Decides, TTS Executes

Control Tags Generated by Main LLM

```
# LLM inserts cognitive pauses based on context
[THINKING] "Um..." "Let me think"
[SEARCHING] "That..." "How should I put it"

# LLM decides emotion and speech rate
[EMO:happy] "Great!"
[SPEED:0.8x] "Let me explain slowly"

# LLM chooses speaking style
[STYLE:casual] "Hey, what's up"
[STYLE:formal] "Hello, how may I help you"

# LLM outputs special sounds
[BREATH]
[SIGH]
[CLEAR_THROAT]
[LAUGH:small]
```

Actual Generation Process

1. Main LLM Output (simulating cognitive load)

```
[STYLE:casual] [THINKING]
"I think, you know," [SEARCHING] "this"
"plan is pretty good" [CLEAR_THROAT] "What do you think?"
```

2. TTS Model Processing

- **Input**: Language (text) + Action (control tags)
- **Processing**: Multimodal generation (Language + Action → Audio)
- **Output**: Natural speech

SEAL Architecture Summary: Agents with Real-time Environmental Interaction

A unified event-driven loop that decouples perception, thinking, and execution to achieve true real-time and parallel processing.

Perception Layer

Input: Continuous signals (voice, GUI)
Output: Discrete event stream
Solves: Latency, unnatural interruptions, and acoustic information loss in traditional voice perception.

Thinking Layer

Input: Discrete event stream
Output: Interleaved thought/action commands
Solves: The serial bottleneck of traditional ReAct, enabling interruptible, asynchronous think-while-listening and speak-while-thinking.

Execution Layer

Input: Discrete action commands
Output: Continuous signals + feedback events
Solves: The "last mile" problem of clumsy agent actions and lack of feedback, forming a closed action loop.

Future Outlook: End-to-End Model

Three Levels of AI Agent Interaction with the World

From Voice to Vision to the Physical World

🗣️ Real-time Voice Calls

- Input Modality: Voice
- Output Modality: Voice
- Data Density: 15-50 tokens/second
- Latency Requirement: <500ms
- Main Challenge: Balancing fast and slow thinking

💻 Graphical User Interface Operations

- Input Modality: Vision
- Output Modality: Mouse clicks, key presses, etc. action sequences
- Data Density: ~2K tokens/screenshot
- Latency Requirement: <1 second
- Main Challenge: Precise action execution

🤖 Physical World Interaction

- Input Modalities: Vision + Voice + Touch
- Output Modalities: Voice + Joint action sequences
- Data Density: ~20K tokens/second
- Latency Requirement: <100ms
- Main Challenge: Real-time perception and control

Technical challenges increase at each level, but solutions can be reused and transferred across levels.

Part Two: Agents Learning from Experience

Why Agents Need to Learn from Experience: From "Smart" to "Proficient"

SOTA Model ≈ Top Graduate

- **Knowledgeable**: Possesses vast general knowledge.
- **Lacks Experience**: May not perform as well as experienced professionals on specialized tasks (e.g., accounting and tax filing).

Pine AI’s Business Challenges

- **Verifying Information**: Pine AI helps users make phone calls. The first time calling a customer service, it doesn’t know they need to verify the last four digits of a credit card. The second time calling the same company’s customer service, can it ask the user for the credit card information beforehand?
- **Task Procedures**: The first time processing a service cancellation, the customer service on the phone said a form needs to be filled out. The second time processing such a cancellation, can it directly fill out the form instead of making a call?
- **Business Rules**: What are the conditions for receiving a certain discount? (e.g., veterans, loyal customers of over 2 years)
- **Price Estimation**: Is $60 a month for 3 Gbps home broadband in a certain area high or low? Is there room for negotiation?

Core Problem: Many business processes are dynamic and not publicly documented. Simply improving the general capabilities of the base model cannot solve these "experience-based" problems.

Learning from Experience: A Fundamental Problem in Machine Learning

The Essence of Machine Learning is to Learn from Experience

Paradigm 1: Post-training via SFT/RL

Method: Parameter Update (Post-training)

• Update weights via gradient descent

• Requires a large amount of labeled data

• Model is fixed after training

• Learning process is slow and expensive

Paradigm 2: In-context Learning

Method: In-context Learning

• Implicit learning through the attention mechanism

• Utilizes long context as temporary memory

• Learning effect is limited to the current session, not permanent

Paradigm 3: Externalized Learning

Method: Externalizing Knowledge and Processes

• **Knowledge Base**: Efficient, reliable, and hallucination-free external knowledge

• **Tool Generation**: Codifying processes to achieve self-evolution

• Transcending the limitations of parameterized knowledge

💡 **Core Insight: From "optimizing model parameters" to "knowledge representation beyond parameters"**

Method 1: Post-training - RL Makes Agents "Proficient"

Core Insight: RL struggles to improve IQ but can solidify experience

Limitations of RL ❌

- **Cannot improve IQ**: Reasoning ability is capped by the base model
- **Cannot create knowledge**: Can only optimize the application of existing knowledge
- **Depends on base model capabilities**: Cannot compensate for fundamental flaws

The True Value of RL ✅

- **Instruction following**: Internalize hundreds of complex rules (industry know-how)
- **Success rate improvement**: Turns processes into "muscle memory," improving success rates and converting `pass@k` to `pass@1`
- **Latency reduction**: Internalizing rules from prompts into the model can shorten the chain of thought, reducing response latency in real-time interactions

Practical Path Based on SFT/RL

Basic Rule Following

- Constitutional AI
- Following complex rules through RLHF

Tool-use Training

- Model as Agent
- The key is to build a simulation environment and reward function
- For example, many agent operations depend on interaction with the external world (live phone calls, web operations, robotics). Some of these experiments are costly, so high-quality simulation environments are needed, or training in low-cost real interaction scenarios and then transferring the capabilities to high-cost scenarios.

Feasibility Verification of Low-Cost RL Solutions

arxiv:2504.11536

Core Challenges

- **Data scarcity**: High-quality tool-use trajectories are difficult to obtain
- **Evaluation difficulty**: The correctness of tool use is hard to judge automatically
- **Training cost**: Traditional RL requires significant computational resources

Solutions

1. **Automatic Trajectory Generation**:
    - Use LLMs to generate tool-use sequences
    - Execute and verify for correctness
2. **Simplified Reward Design**:
    - Binary reward (success/failure)
    - Reduce the need for manual labeling

Experimental Results

Training Efficiency:

- Effective with just a few thousand trajectories
- Suitable for academic research and small-scale applications

Performance Improvement:

- Tool-use accuracy: +15-20%
- Task completion rate: +25-30%
- Significantly enhanced error recovery ability

Implications:

- RL training for tool use does not necessarily require massive resources
- Well-designed small-scale data can also bring significant improvements

Kimi K2 - Model as Agent

Large-scale Agent Data Synthesis and Joint RL Training

Based on the Kimi K2 Technical Report

Agent Data Synthesis Pipeline

- **Scale generation**: Systematically construct tools, agents, tasks, and trajectories
- **Environmental diversity**: Combine real and simulated environments
- **Verification mechanism**: Ensure the correctness and high fidelity of trajectories

Technical Innovations:

1. **MuonClip Optimizer**: 15.5T tokens of zero-loss peak training
2. **32B active parameters**: 1T total parameter MoE architecture
3. **Token efficiency**: Maximize the learning signal from each token

RLVR (RL with Verifiable Rewards)

- Verifiable task rewards
- Self-Critique Rubric
- Extend from static alignment to open domains

Model as Agent Concept: The first open-source model to truly implement "Model as Agent"

Model as Product Concept: The simulation environment required during RL training is already an agent product. Therefore, once the model training is complete, the agent can be directly released as a product.

Kimi K2 Critic Rubrics: An Evaluation Framework for General RL

A Systematic Approach from Rules to Self-Critique

Core Rubrics

1. **Clarity and Relevance**
   - Respond to user intent concisely and sufficiently
   - Eliminate unnecessary details
   - Use efficient formatting
2. **Conversational Fluency and Engagement**
   - Maintain a natural and smooth conversation
   - Provide relevant observations and insights
   - Adapt tone to different contexts
3. **Objective and Grounded Interaction**
   - Avoid meta-comments, focus on the task itself, and maintain a neutral, professional tone

Prescriptive Rules and Limitations

Prohibitive Rules (Prescriptive):

- Do not start with praise
- Avoid explicit self-justification
- Reduce unnecessary qualifiers

Known Limitations:

- May be overly confident, lacking caution in ambiguous scenarios, and preferring a single, clear answer

Innovations:

- Systematize human preferences into evaluable rules
- Internalize these rules through RL rather than relying on prompts
- Achieved a closed loop of evaluation and training

Method 2: In-context Learning - Using Conversation History as Built-in Memory

The Transformer's Attention Mechanism: An Efficient "Information Query"

The core of the Transformer is the **Self-Attention** mechanism, which can be understood as a key-value database based on similarity matching.

Query (Q)	**Key (K)**	**Value (V)**
The "query need" of the current token	The "content index" of historical tokens	The "actual content" of historical tokens

Calculation Process: Use the current **Q** to compute similarity with all historical **K**s. The resulting weights are used to perform a weighted sum of the corresponding **V**s.

Note: Unlike a vector database where **K** and **V** are combined (the content to be searched), and the query vector **Q** has the same distribution as the content (both are text embeddings), in a Transformer, the distributions of **Q, K, and V** are all different.

Two Bottlenecks of Long Context: Memory Wall vs. Compute Wall

Two Fundamental Directions for Long Context Optimization

🧠 Memory Wall

Problem: Excessive KV Cache usage

- Traditional approach: `batch × seq_len × d_model`
- 128K context requires **56GB** of KV Cache
- Limits batch size and maximum sequence length

Optimization Solutions:

- **MLA (DeepSeek-V2)**: Low-rank compression
  - 16x compression ratio
  - 128K requires only 3.5GB
- **GQA**: Shared KV heads
- **MQA**: Single-head KV

⚡ Compute Wall

Problem: $O(n^2)$ attention complexity

- Traditional approach: Interaction between all token pairs
- 100K sequence = **10 billion computations**
- Extremely slow for both training and inference

Optimization Solutions:

- **NSA (DeepSeek-V3)**: Sparse attention
  - Computes only important token pairs
  - Hardware optimization for acceleration
- **Linear Attention**: $O(n)$ complexity
- **Flash Attention**: IO optimization

Memory Wall: DeepSeek MLA (Multi-head Latent Attention)

A Fundamental Method to Reduce KV Cache Memory Overhead

Core Innovation: DeepSeek-V2 compresses the KV Cache to 1/16 of its original size through MLA, significantly reducing memory usage (arxiv:2405.04434).

Technical Principle

- **Low-rank Projection**: Project KV vectors into a low-dimensional latent space
  - Original: `d_head × n_heads`
  - Compressed: `c_kv` (typically 512)
- **Shared Representation**: Multiple attention heads share the same low-dimensional KV representation
- **Decoupled RoPE**: Separate positional encoding from content representation
  - Content is passed through the low-rank space
  - Positional information is processed independently

Practical Effects

- **Memory Savings**:
  - KV Cache reduced by **93.75%** (16x compression)
  - 128K context requires only **3.5GB** of KV Cache
- **Performance Preservation**:
  - Almost no performance loss on benchmark tests
  - Long-text understanding ability is fully maintained
- **Inference Acceleration**:
  - Reduced memory bandwidth requirements
  - Increased batch processing throughput

MLA is one of the core technologies enabling DeepSeek-V2 to efficiently handle long contexts

Compute Wall: Sparse Attention

Core Insight: Turning the KV Cache into a Vector Database

Traditional Full Attention

```
# Compute all QK similarities
scores = Q @ K.T  # O(n²)
attention = softmax(scores)
output = attention @ V
```

Problem: Must compute the full n×n matrix

Sparse Attention

```
# Find only the Top-K relevant K's
top_k_indices = vector_search(Q, K, k)  # O(n log n)
sparse_scores = Q @ K[top_k_indices].T
sparse_attention = softmax(sparse_scores)
output = sparse_attention @ V[top_k_indices]
```

Advantage: Computes only `k<<n` relevant items

Disadvantage: Traditional hash-based acceleration methods for vector databases are not applicable because Q and K have different distributions in space, and high-similarity matches are rare. Therefore, finding the top K still requires scanning a large proportion of K. If heuristics (e.g., attention sinks for the first few tokens and recent tokens) are used for filtering, some high-matching K's may be missed.

Compute & Memory Wall: Linear Attention

Core Idea: Change the computation order to achieve linear complexity

How it Works

- Standard attention is calculated as `softmax(Q * K^T) * V`.
- Linear attention approximates softmax using a kernel function (\phi) and changes the computation order to `( \phi(Q) * ( \phi(K)^T * V) )`, thus avoiding the construction of a huge n x n matrix.

Advantages

Extremely low computational cost, fast inference speed.

Disadvantages

- It is a "lossy compression," performing poorly on tasks requiring "needle-in-a-haystack" precise retrieval.
- The needle-in-a-haystack capability is the foundation of In-Context Learning, which in turn is the basis for instruction-following and long-chain reasoning abilities, which are the foundation of tool-use capabilities. If the needle-in-a-haystack ability is weak, instructions and data in a long context cannot be fully utilized.
- For shorter contexts, with the same KV Cache size, Softmax Attention performs better than Linear Attention.

Case Study: Google's Infini-Attention

Efficient Transformer for Infinite Context

Core Innovation: Infini-Attention introduces **compressive memory** into the traditional attention mechanism, enabling infinite context processing with bounded memory (arxiv:2404.07143).

Architectural Design

- **Dual Attention Mechanism**:
  - **Local Attention**: Handles masked attention for the current segment
  - **Long-term Memory**: Compressive storage based on linear attention
- **Unified Computation**: Reuses the same set of Q, K, V for both types of attention calculations
- **Memory Update**: Old KV states are stored in the compressive memory instead of being discarded

Experimental Results

- **Memory Efficiency**: Achieves a **114x** memory compression ratio compared to standard attention
- **Scalability**: A 1B model naturally extends to a **1M** sequence length
- **SOTA Performance**: An 8B model achieves new state-of-the-art results on a 500K-length book summarization task

DeepSeek NSA: Native Sparse Attention

Hardware-aligned Hierarchical Sparsity Strategy

Core Innovation: NSA (Native Sparse Attention) combines algorithmic innovation with hardware optimization to achieve efficient long-context modeling (arxiv:2502.11089).

Technical Architecture

- **Dynamic Hierarchical Sparsity**:
  - **Coarse-grained**: Token compression to maintain global awareness
  - **Fine-grained**: Token selection to ensure local precision
- **Hardware Optimization**:
  - Algorithm design balances arithmetic intensity
  - Implementation optimized for modern GPU architectures

Performance

- **Quality Preservation**: Matches or exceeds full-attention models on various benchmarks
- **Significant Acceleration**:
  - Decoding acceleration at 64K sequence length
  - Substantial improvements in both forward and backward passes
- **Practicality**: Can be trained directly without adapting pre-trained models

NSA is one of the key technologies enabling DeepSeek-V3 to efficiently handle long contexts

MiniMax-01: Lightning Attention

A Practical Solution Combining Linear Attention + Softmax Attention

Core Innovation: MiniMax-01 achieves efficient processing of ultra-long contexts through Lightning Attention and a MoE architecture (arxiv:2501.08313).

Lightning Attention

- **Linear Attention Mechanism**: Replaces traditional softmax attention for linear complexity
- **Hybrid Architecture Design**:
  - Insert 1 Softmax Attention block after every 7 Lightning Attention blocks
  - A total of 80 layers to balance long- and short-range dependencies
  - The Softmax Attention layers ensure the "needle-in-a-haystack" information retrieval capability, thereby guaranteeing model abilities like instruction following and long-chain reasoning
- **Technical Details**:
  - Based on Transnormer blocks (Qin et al., 2022a)
  - Four-way parallel processing for Q, K, V, G
  - Combination of SiLU and Sigmoid activation functions

Achievements

- **Context Length**:
  - Supports **1M** tokens during training
  - Extrapolates to **4M** tokens during inference
- **Model Open-Sourced**

Theoretical Insight: Learning without Training

Based on the paper "Learning without training: The implicit dynamics of in-context learning" (arxiv:2507.16003)

Core Finding: Transformer blocks can implicitly update MLP weights through context

Implicit Weight Update Mechanism

Key Insight:

- Stacking Self-Attention and MLP layers produces a special effect
- Contextual information is converted into a low-rank update to the MLP weights
- Each attention head contributes a rank-1 update

```
# Simplified mathematical expression
MLP_effective = MLP_original + Σ(rank_1_updates)
# where rank_1_updates come from the attention output
```

Significance: Explains how LLMs can learn new patterns from prompts without training

The Nature of In-Context Learning

Feature	Traditional Training	In-Context Learning
Weight Update	Explicit (gradient)	Implicit (attention)
Persistence	Permanent	Current sequence only
Computation	Backpropagation	Forward pass
Adaptation Speed	Slow (many epochs)	Fast (single inference)

- LLMs are effectively performing "soft training" during inference
- Explains why few-shot learning is so effective
- Provides a theoretical basis for designing better prompts

ICL vs. Fine-tuning: Counterintuitive Findings on Implicit Pattern Learning

Based on the paper "Deeper Insights Without Updates: The Power of In-Context Learning Over Fine-Tuning" (arxiv:2410.04691)

Core Finding: For tasks containing implicit patterns, ICL can capture and utilize these patterns better than Fine-tuning

🔍 Experimental Design

Implicit Pattern Task Types:

- **Mathematical Computation**: Expression evaluation, Boolean functions
- **Textual Reasoning**: Relational reasoning (judging city connections)
- **Code Understanding**: Output prediction

Implicit Pattern Example:

```
# Expression contains a zeroing term
(6-1) + (6-6) * (-10+1+2+13) = 5
#         ↑ This part is 0, can be a shortcut

# Connection shortcut in relational reasoning
A-B-Z direct connection vs. A-B-C-D-...-Y-Z
```

📊 Key Experimental Results

Method	Accuracy Improvement	Sample Efficiency
ICL	Significant	Just a few examples
Fine-tuning	Limited	Needs thousands of times more samples

Performance Comparison:

- ICL can quickly identify deep patterns
- Fine-tuning struggles to learn implicit patterns even with more data
- ICL performs more robustly on OOD (out-of-distribution) data

Model Scale Impact:

- Verified on models from 0.5B to 7B parameters
- The advantage of ICL is consistent across all model sizes

Circuit Shift Theory: Why is ICL Better at Pattern Recognition?

Explaining the difference between ICL and Fine-tuning from an interpretability perspective

Circuit Shift Mechanism

What is a Circuit?

- A subgraph in the model responsible for a specific behavior
- Composed of specific attention heads and MLP layers
- Represents the model’s "thinking path" for solving a problem

Key Finding: ICL leads to a larger-scale circuit shift, meaning ICL more thoroughly changes the model’s problem-solving method

Activation Patching Analysis:

- By manipulating the activation values of specific components
- Quantify the contribution of different components to the task
- Found that ICL activates different circuit patterns

Pattern Capturing Ability:

- **ICL**: "Can quickly capture deep patterns and significantly improve accuracy"
- **Fine-tuning**: "Even with thousands of times more training samples, the improvement is very limited"

Mechanism Explanation:

- ICL is not simple pattern matching but activates different computational circuits in the model
- Although fine-tuning updates parameters, the circuit shift is smaller for implicit pattern tasks
- This explains why ICL can achieve better results without parameter updates

Method 3: Externalized Learning

Transcending the Fundamental Limitations of Parameterized Knowledge

Separating knowledge and processes from model parameters to achieve reliable, efficient, and evolvable persistent learning.

3.1 Knowledge Base

Solving the unreliability, inefficiency, and hallucination problems of parameterized knowledge

- Separate volatile, precise knowledge from model parameters
- Achieve efficient, reliable, and hallucination-free knowledge retrieval
- Support rapid updates and expansion

3.2 Tool Generation

Solving the inefficiency and unreliability of parameterized learning

- Solidify repetitive processes into reliable, efficient code
- Agents achieve self-evolution by creating tools
- From minimal pre-definition to maximum self-evolution

3.1 External Knowledge Base - Rapidly Accumulating Experience

Equip the agent with an external knowledge base to record and summarize task experiences for retrieval in similar scenarios.

Knowledge Representation

Summary

- **Content**: Store summary, conclusive information.
- **Example**: "To cancel service B from company A, you need to verify the user's identity by providing the order number, registered email, and the last four digits of the credit card number."

Raw Details

- **Content**: Store raw details, such as the complete record of customer service conversations.
- Use descriptive filenames and store in the file system.

Tree-based Summarization (Example: RAPTOR)

- Treat raw details as leaf nodes of a tree, perform clustering to generate summaries, then cluster the summaries to generate nodes at a higher level of the tree.
- For retrieval, you can use a vector database for similarity matching or index it like a file system.

Agentic Autonomous Summarization

- The agent autonomously decides to use tools like `write_file` , `edit_file` , etc., to write summaries to the file system.

Knowledge Retrieval

Automatic Matching and Context Insertion

- Automatically match corresponding entries from the knowledge base based on user ID or business type and insert them into the context.

Agentic Semantic Search

- The agent autonomously decides which keywords to use for searching. The retrieval can use vector databases or methods like BM25 to match against the knowledge base.

Agentic File System Reading

- The agent autonomously decides to use tools like `read_file` , `find` , `grep` , etc., to read content from the file system.

Contextual Retrieval: Improving RAG Retrieval Accuracy

Based on Anthropic's Contextual Retrieval

Core Problem: Traditional RAG loses context when chunking documents, leading to decreased retrieval accuracy.

Contextual Embeddings & Contextual BM25

- Before embedding and creating a BM25 index, add specific explanatory context to each chunk.

Example Transformation

```
# Original chunk (lacks context)
original = """The company's revenue grew by 3%
           over the previous quarter."""
# Contextualized chunk
contextual = """This chunk is from an SEC filing
             on ACME corp's Q2 2023; previous
             quarter revenue was $314M.
             The company's revenue grew by 3%..."""
```

Implementation

- Use Claude to automatically generate context for each chunk (50-100 tokens).

Results

Reduction in retrieval failure rate (top-20 chunks):

- **Contextual Embeddings only**: ↓35%
- **Contextual Embeddings + BM25**: ↓49%
- **+ Reranking**: ↓67%

Key Advantages:

1. **Preserves document context**: Each chunk knows where it came from.
2. **Improves exact matching**: BM25 handles proper nouns, error codes, etc.
3. **Enhances semantic understanding**: Embeddings capture more accurate meaning.

Best Practice Combination: Contextual Embeddings + Contextual BM25 + Reranking + Top-20 chunks

Learning Method Comparison: External Knowledge Base vs. Built-in Attention

External Knowledge Base (RAG)

- **Analogy**: Guiding an LLM that doesn't support thinking to think by using a "Think step by step" prompt.
- **Pros**: No retraining required, plug-and-play. Can use additional compute power for multi-faceted summarization, and the summarization process can flexibly incorporate industry know-how.
- **Cons**: Effectiveness depends on the precision and recall of retrieval. Often, semantically related content across multiple fragments is difficult to extract. Additionally, RAG-extracted fragments lack context, affecting understanding.

Built-in Attention (Long Context)

- **Analogy**: A model that natively supports long context and built-in thinking capabilities.
- **Pros**: End-to-end optimization, higher potential for effectiveness.
- **Cons**: Currently costly. The summarization process is generally determined by the model itself, making it difficult to incorporate industry know-how or leverage additional compute power.

Fine-tuning vs. RAG: An Empirical Comparison of Knowledge Injection Methods

Based on the paper "Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs" (EMNLP 2024)

💡 **Paper's Core Insight**: RAG is not only more effective but also avoids the knowledge forgetting and hallucination problems that fine-tuning can cause.

The Paper's Comparative Experiment

- Compared Unsupervised Fine-tuning with RAG
- Evaluated the handling of Existing Knowledge and New Knowledge

Key Findings

1. **RAG performs better overall**
    - Outperforms fine-tuning in handling both existing and new knowledge
    - More reliable retrieval of factual information
2. **Limitations of Fine-tuning**
    - LLMs have difficulty learning new factual information through unsupervised fine-tuning
    - Requires multiple variations of the same fact to learn effectively
3. **Improvement Strategies**
    - Expose the model to multiple expressions of the same fact during training
    - Use a combination of methods rather than relying on a single one

Scenarios Where RAG Excels

- When accurate injection of factual knowledge is needed
- Handling new domains or new knowledge
- Avoiding contamination of model parameters

Challenges of Fine-tuning

- Unsupervised fine-tuning alone is not effective for injecting new facts
- Requires data augmentation: multiple expressions of the same knowledge
- May affect the model's original capabilities

Recommended Strategy

- Prioritize RAG for knowledge injection
- If fine-tuning is necessary, ensure data diversity
- Consider hybrid solutions to leverage the strengths of both

Large-Scale Knowledge Summarization with LLMs

Turning compute power into a scalable knowledge base

Problems with Traditional Knowledge Base Construction

- **Fragmented knowledge**: A vast amount of knowledge and industry experience is not summarized
- **Inefficient querying**: Information is scattered across the internet
- **High cost**: The cost of summarizing industry knowledge is extremely high
- **Difficult to build knowledge bases**: Building internal company knowledge bases is challenging

Building Large-Scale Knowledge Bases with LLMs

- **General Knowledge ➜ Foundational Model**
  - Model training itself is a knowledge summarization process
  - Transformers are not good at memorizing a large number of factual details, which would lead to an explosion in parameter count
- **Factual Information, Industry Experience ➜ LLM-automated Summarization**
  - Specific facts like personal information, industry data
  - Industry experience like verification information, procedures, rules, price estimations
  - Organize massive amounts of raw data to form a structured knowledge base

💡 **Core Insight**: LLMs can convert compute power into a scalable knowledge base

3.2 Tool Generation - Enabling Agent Self-Evolution

Core Idea: From Minimal Pre-definition to Maximum Self-Evolution (Alita: (arxiv:2505.20286))

Minimal Pre-definition Principle

- **Minimalist Architecture**: Equipped with only a single core component (Web proxy)
- **Avoid Over-design**: Do not pre-define complex tools and workflows
- **Prioritize Generality**: Reduce domain-specific hard-coding

Results: GAIA benchmark

- Pass@1: **75.15%**
- Pass@3: **87.27%**
- Outperforms many complexly designed Agent systems

Maximum Self-Evolution Mechanism

Core Capabilities:

1. **Autonomous Tool Creation**: Generate new tools based on task requirements
2. **Capability Refinement**: Iteratively improve the performance of existing tools
3. **Experience Reuse**: Solidify successful patterns into reusable components

Model Context Protocols (MCPs):

```
# Agent automatically generates task-related protocols
mcp = agent.generate_protocol(task)
# Execute and optimize
result = agent.execute_with_mcp(mcp)
# Store successful patterns
agent.store_successful_pattern(mcp)
```

The Challenge of Scaling Tools: MCP-Zero Active Tool Discovery

Based on the paper MCP-Zero: Active Tool Discovery for Autonomous LLM Agents

The Problem of Exploding Tool Numbers

Scale Challenge: MCP ecosystem has 308 servers, 2,797 tools

Dilemma of Traditional Methods:

- **Full Injection**: GitHub MCP with 26 tools requires 4,600 tokens
  - The complete toolset would require **248k tokens** → Context explosion
- **Static Retrieval**: Selection based on the initial query, cannot anticipate task evolution
  - "Debugging a file" requires file system + code analysis + command execution

Fundamental Problem: The agent becomes a passive selector, not an active discoverer

MCP-Zero: From Passive to Active

Core Concept: Let the agent actively identify capability gaps and request tools on demand

Three Mechanisms:

1. **Active Tool Request**: Agent generates a structured request

   ```
   server: github  # Platform domain
   tool: search_repos  # Operation type
   ```

2. **Hierarchical Semantic Routing**: First filter by server, then match the tool
3. **Iterative Capability Expansion**: Dynamically discover and build toolchains during execution

Measured Effects:

- APIBank test: **98% token savings**
- Accurately selects from 3000 tools while maintaining high accuracy
- Stable performance as the tool ecosystem grows

Agent Auto-generates Tool Case 1: Intelligent RPA for Computer Use

The Dilemma of Computer Use

- **Slow Speed**: Every step requires interaction with the LLM, resulting in high latency.
- **High Cost**: Each operation is an expensive LLM call.

Inspiration from Traditional RPA

- **Fast Speed**: Runs pre-written scripts, much faster than humans.
- **Problem**: Cannot handle dynamic interfaces, lacks understanding and judgment.

New Idea: Let the LLM automatically summarize the operation process into an "intelligent RPA" tool.

Intelligent RPA: Technical Implementation Details

Core Challenges of Generating RPA Code from Operation Sequences

Three Major Technical Hurdles

1. **Coordinate Drift Problem**
    - Page element positions change dynamically
    - Clicking fixed coordinates often fails
2. **Dynamic Content Judgment**
    - Some steps require understanding the content
    - Example: Selecting the cheapest flight ticket
3. **Uncertain Response Time**
    - Web page loading times are not fixed
    - Cannot handle exceptions

Solutions

1. **Intelligent Element Localization**
    - Use element ID/XPath instead of coordinates
    - Playwright automatically adapts to interface changes
2. **LLM Analysis and Optimization**
    - Distinguish between fixed processes and dynamic judgments
    - Extract automatable operation sequences
3. **Event-driven Execution**
    - Listen for browser event completion states
    - Intelligently wait for elements to be clickable
    - Automatically switch to LLM takeover on timeout

The Effect of Intelligent RPA

Use Case 1: Checking the Weather (9 steps)

- **Traditional Method**: 47 seconds, 9 LLM calls
- **After Acceleration**: 10 seconds, 2 LLM calls + 1 RPA

Use Case 2: Booking a Flight on the Official Website (240 steps)

- **Traditional Method**: 19 minutes, 240 LLM calls
- **After Acceleration**: 4 minutes, 25 LLM calls + 4 RPAs

Core Advantage: Zero manual development cost, fully automatic generation, and can automatically generate new tools to adapt to website changes.

Agent Auto-generates Tool Case 2: Agent Log Parsing and Visualization

Challenge: Agent Execution Flow Logs

- **Requirement**: Visualize the agent's execution flow (trajectory) for observation and debugging.
- **Format Diversity**: The agent's tools and sub-agents are diverse, each with different parameter formats and tool return result formats, which are constantly changing.
- **Large Log Volume**: The entire agent execution process often has hundreds of steps, requiring compressed display for better intuition. But it must also be possible to expand the full details of each LLM call and each tool call for debugging.
- **High Maintenance Cost**: Manually written parsing code needs constant updates and it's difficult to predict all possible log formats.

Agent's Self-Evolving Solution

Core Mechanism: Frontend automatically reports parsing failures to the Agent.

1. **Intelligent Recognition**: Agent analyzes the pattern of the failed log.
2. **Automatic Parsing Code Generation**: Generates new frontend parsing code based on the sample.
3. **Automatic Testing of Parsing Code**: Uses a virtual browser execution environment to check if the new parsing code can correctly parse the new log type (any errors), and uses a Vision LLM to check if the visualization effect meets expectations.
4. **Hot-Update Deployment**: The parsing code is automatically updated.

Agent Auto-generates Tool Case 3: Automatic Diagnosis of Production System Issues

Challenges in Production System Debugging

- **Difficulty in Problem Localization**: Finding the root cause from a massive amount of agent execution flow logs.
- **High Reproduction Cost**: Production issues are difficult to reproduce in a test environment.
- **Fixed Issues Tend to Reappear**: High cost of building regression test cases.

Agent's Automated Diagnosis Flow

Working Mechanism: Automatically triages issues from production environment agent execution flow logs and generates issue reports and test cases.

1. **Agent Execution Flow Log Analysis**:
    - Combines system architecture documents and PRDs (Product Requirement Documents) to automatically analyze if the agent's execution flow meets expectations.
2. **Test Case Generation**:
    - Automatically generates Regression Test Cases.
    - Ensures that the issue does not reappear after being fixed.
3. **Work Item Creation**:
    - Automatically creates a Scrum Work Item.
    - Includes a problem description, scope of impact, and a suggested fix.

Summary: From Post-training to In-context Learning to Externalized Learning

Paradigm 1: Post-training

Method: RL Parameter Update

- **Pros**: Solidifies experience into parameters, strong generality
- **Cons**: Slow to update, high cost, unreliable for facts

Paradigm 2: In-context Learning

Method: Attention "Soft Update" during inference

- **Pros**: Fast adaptation, no training required
- **Cons**: Temporary, non-persistent, limited by context window

Paradigm 3: Externalized Learning

3.1 Knowledge Base

Pros: Solves hallucination problems, knowledge updates faster than post-training, can leverage extra compute for in-depth summarization

3.2 Tool Generation

Pros: Codifies processes through tool generation, achieving efficient and reliable automation

Externalized Learning: Beyond the Limitations of Attention

Attention opened the door to in-context learning, but it is not the end of learning.

Evolution and Limitations of Learning Paradigms

1. **Parameterized Learning (Training)**
   - **Mechanism**: Gradient Descent
   - **Pros**: Internalized knowledge, strong generality
   - **Limitations**: High cost, slow updates, facts easily confused
2. **In-context Learning (Inference)**
   - **Mechanism**: Attention "soft update"
   - **Pros**: Fast adaptation, no training required
   - **Limitations**: Session-level, non-persistent, dependent on context window
3. **Externalized Learning**
   - **Mechanism**: Structured read/write + code generation
   - **Pros**: Persistent, reliable, efficient, hallucination-free
   - **Positioning**: Combines the advantages of parameterized and in-context learning

Why Externalized Learning is an Inevitable Trend

Fundamentally solves the three major problems of Transformers:

- **Hallucination and Unreliability**: External knowledge bases provide a verifiable, precise source of truth.
- **Inefficient Learning and Retrieval**: Targeted retrieval is far superior to "finding a needle in a haystack" among billions of parameters.
- **Clumsiness in Repetitive Tasks**: Tool generation solidifies vague "experience" into precise, efficient code.

Paradigm Shift in Agent Design:

- Past: Static, all-knowing models
- Present: Dynamic models dependent on context
- Future: Learning entities that interact efficiently with the external world

🚀 **Core Idea: Let the model do what it does best (reasoning), and entrust knowledge and processes to more reliable external systems.**

Scaling Law: From Pre-training to RL

> *"We want AI agents that can discover like we can, not which contain what we have discovered. Building in our discoveries only makes it harder to see how the discovering process can be done." — Rich Sutton, The Bitter Lesson*

Phase 1: Pre-training based on predicting the next token

- **Core Law**: The model's performance predictably improves with increases in computational resources, model parameters, and training data volume.
- **Implementation**: Unsupervised pre-training on massive amounts of internet text to learn general world knowledge.
- **Essence**: **Internalizing** knowledge into the model's parameters, forming a vast, static knowledge base.
- **Achievement**: Achieved general language understanding and reasoning capabilities.

Phase 2: Reinforcement Learning: Post-training through interaction with the world

- Agents allow the model to move beyond passive learning to actively exploring the world.
- **Interaction with**:
  - **Humans**: Conversation and collaboration through real-time voice.
  - **The Internet**: Operating computers, browsing websites, using software.
  - **The Physical World**: Controlling robots, interacting with the real environment.
- **Learning Method**: Through reinforcement learning, discovering and learning from the successes and failures of interactions, which is the second curve of the Scaling Law.
- **Goal**: To transform from a container of knowledge to an engine of discovery.

The Bottleneck of Scaling Law: The Precise Memory Limitation of Transformers

- The Transformer architecture is not good at precisely and reliably memorizing and updating these highly dynamic, specific details.
- Forcing memorization easily leads to **hallucinations** and **information confusion**, and is not cost-effective (requiring traversal of a large portion of all knowledge during training and inference), becoming a huge obstacle for the Scaling Law.

The Future Path: Externalized Learning Continues the Scaling Law

> *"The two methods that seem to scale arbitrarily ... are **search** and **learning**." — Rich Sutton, The Bitter Lesson*

Breaking the Bottleneck: Externalized Learning

- The LLM itself provides powerful tools to solve the memory bottleneck—**summarization** and **code generation**

Precise, Structured Knowledge Bases

- SOTA LLMs can summarize unstructured experiences (like conversations, documents) into structured knowledge
- Before LLMs, this high-quality summarization required expensive domain experts, so expert systems based on human-written rules could not scale; but today, LLMs are like an infinite number of domain experts
- In the NLP field, Summarization is the only traditional NLP topic that remains active in the LLM era

Code as a Universal Knowledge Representation

- LLMs can precisely describe any experience or knowledge as code
- Code generation is not just a tool for programmers, but a universal, structured data representation that is precise, verifiable, and composable

Co-evolution of LLMs with External Knowledge and Tools

- **Model does what it does best**: Leverage the powerful general reasoning and understanding capabilities of LLMs
- **External systems do their job**: Manage precise, dynamic knowledge and processes with external knowledge bases and codebases
- **Co-evolution**: The agent learns by interacting with external systems (people, the internet, the physical world) and consolidates the learned outcomes (compressed summaries of knowledge, tools) into the external systems, forming a continuously strengthening closed loop

Continuing the Scaling Law

- With more compute power, the agent can extract more detailed and precise knowledge from its interactions with the world
- **Search**: Corresponds to external knowledge bases and tool libraries
- **Learning**: Corresponds to the LLM's ability to summarize interaction experiences into knowledge and code
- "Externalized learning" breaks the limitation of the number of model parameters, extending the boundary of the Scaling Law to the external ecosystem of knowledge and tools

Pine AI

We are looking for full-stack engineers who can build SOTA autonomous AI agents.

Our philosophy:

Everyone's contribution to the company's valuation should be in the tens of millions of dollars.

Requirements to Join Pine AI

🤖 1. Proficient in AI-assisted programming

- 80%+ of code written through human-machine collaboration
- Coding interview: Complete feature development in 2 hours with AI assistance
- All internal systems built based on AI

💻 2. Love solving problems hands-on

- "Talk is cheap, show me the code"
- Become a combination of an architect and a product manager
- Directly command the AI to reduce information loss

🏗️ 3. Solid software engineering skills

- Complete documentation and testing
- Make the code understandable and maintainable by AI
- High-quality engineering practices

🧠 4. Understand LLM principles

- Know the basic principles and capability boundaries
- The right way to harness LLMs
- Provide appropriate context and tools

🚀 5. Confidence to solve world-class problems

- Strive for SOTA level
- Grow with a startup
- Constantly surpass existing levels

🎯 Our Mission

To truly solve users' troubles and get things done by building agents that can interact with the world in real-time and learn from experience.

Pine AI - Building Agents That Get Things Done

```
mail -s "Join Pine AI" -A /path/to/your_resume.pdf boj@19pine.ai
```

Two Clouds Over Agents: Real-time Interaction and Learning from Experience

> *"The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin."*
> *— Rich Sutton, The Bitter Lesson*

Powered by Slidev