LLM Agents and Tool Calling

Concepts

Tirtharaj Dash

Department of CS & IS
BITS Pilani, Goa Campus, India
Homepage

February 11, 2026

Outline

We will build up mathematical understanding of:

1. Base LLM: Next token prediction probability
2. LLM as Agent: Sequential decision making
3. LLM calling Tools: Extended action space

Key Question: What happens to the probability distribution at each level?

Base LLM: Next Token Prediction

Standard Language Model

At each time step t, predict next token:

$$P(x_t \mid x_{<t}) = \text{softmax}(f_\theta(x_{<t}))$$

where:

- x_t = next token
- $x_{<t} = (x_1, x_2, \ldots, x_{t-1})$ = previous tokens (context)
- f_θ = neural network with parameters θ
- Output: probability distribution over vocabulary V

Base LLM: Sequence Generation

Full sequence probability

$$P(x_{1:T}) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$$

Key property:

Distribution is **only** over tokens in vocabulary V
Typically $|V| \approx 32K$ to $100K$ tokens

Example: $P($"Paris" | "Capital of France is"$) = 0.87$

LLM as Agent: Definition

Agent: Makes sequential decisions to achieve goals

$$\pi(a_t \mid s_t, G) = P(a_t \mid s_t, G; \theta)$$

where:

- a_t = action at time t (not just text!)
- s_t = state (environment + history)
- G = goal
- π = policy (the LLM)

Key Change:

LLM now produces **actions** that affect an **environment**

LLM as Agent: Example

Scenario: User asks *"What is the weather in Paris?"*

Components of $\pi(a_t \mid s_t, G) = P(a_t \mid s_t, G; \theta)$:

- s_t (State): Current conversation
  - Previous messages: "Hello", "I'm planning a trip"
  - Current query: "What is the weather in Paris?"
- G (Goal): "Answer user questions accurately"
  - Encoded in system prompt
  - Guides what actions are appropriate
- θ (Parameters): The 175B weights of GPT-3.5
  - Fixed neural network weights
  - Learned during training

LLM as Agent: Example

Action space $\mathcal{A}$:

- $a_1 =$ `call_weather_api("Paris")`
- $a_2 =$ `generate_text("I don't know")`
- $a_3 =$ `search_web("Paris weather")`

LLM computes logits using θ:

$$\mathbf{z} = f_\theta(\text{concat}[s_t, G]) = \begin{bmatrix} 8.3 \\ 2.1 \\ 5.7 \end{bmatrix}$$

Convert to probabilities (softmax):

$$\pi(a_t \mid s_t, G) = \text{softmax}(\mathbf{z}) = \begin{bmatrix} 0.82 \\ 0.03 \\ 0.15 \end{bmatrix}$$

LLM as Agent: State Updates

Environment dynamics

Action a_t changes the state:

$$s_{t+1} = \tau(s_t, a_t)$$

where τ is the environment transition function.

Full trajectory probability:

$$P(\tau) = P(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t, G) \cdot P(s_{t+1} \mid s_t, a_t)$$

Distinction from Base LLM:

- Base LLM: $x_t \in V$ (vocabulary tokens)
- Agent: $a_t \in \mathcal{A}$ (action space, can differ from tokens)

Agent Example: Action Space

Example Action Space $\mathcal{A}$:

- `generate_text`(text)
- `search_web`(query)
- `calculate`(expression)
- `stop`()

Agent selects: $a_t \sim \pi(\cdot \mid s_t, G)$

Example: $P(\texttt{search\_web} \mid s_t) = 0.75$
: $P(\texttt{generate\_text} \mid s_t) = 0.25$

LLM Calling Tools: Action Decomposition

Tool-augmented LLM: Action space includes tool calls

$$\mathcal{A} = \mathcal{A}_{\text{text}} \cup \mathcal{A}_{\text{tool}}$$

where:

- $\mathcal{A}_{\text{text}} = V^*$ (text generation)
- $\mathcal{A}_{\text{tool}} = \{(f_i, \text{args}_i)\}$ (tool calls)

At each step, LLM chooses action type:

$$P(a_t \mid s_t) = P(\text{type}_t \mid s_t) \cdot P(a_t \mid \text{type}_t, s_t)$$

where $\text{type}_t \in \{\text{text}, \text{tool}\}$

Tool Calling: Two Cases

Case 1: Generate text

$$P(\mathsf{type}_t = \mathsf{text} \mid s_t) = \sigma_{\mathsf{text}}(s_t)$$
$$P(a_t \mid \mathsf{type}_t = \mathsf{text}, s_t) = \mathsf{softmax}(f_\theta(s_t))$$

Case 2: Call tool

$$P(\mathsf{type}_t = \mathsf{tool} \mid s_t) = \sigma_{\mathsf{tool}}(s_t)$$
$$P(a_t \mid \mathsf{type}_t = \mathsf{tool}, s_t) = P(\mathsf{tool}_i, \mathsf{args} \mid s_t)$$

Note: Model learns when to use tools vs generate text

Tool Execution and Observation

Tool execution produces observation:

$$o_t = \text{tool}_i(\text{args})$$

Observation updates state:

$$s_{t+1} = s_t \oplus o_t$$

where $\oplus$ means incorporating observation into context

Example:

$\text{tool} = \texttt{calculator}$
$\text{args} = \text{"}2 + 2\text{"}$
$o_t = 4$
$s_{t+1} = s_t + \text{"Result: 4"}$

Tool calling: Example

Scenario: User asks *"What is 37 × 89?"*

Step 1: Decide action type

$$P(\text{type}_t \mid s_t) = \begin{cases} P(\text{text} \mid s_t; \theta) = 0.15 \\ P(\text{tool} \mid s_t; \theta) = 0.85 \end{cases}$$

LLM recognizes this needs calculation

Step 2: Choose specific action given type
If text: $P(a_t \mid \text{type} = \text{text}, s_t)$

$$\rightarrow \text{Generate tokens: "The answer is..."}$$

If tool: $P(a_t \mid \text{type} = \text{tool}, s_t)$

$$\rightarrow \text{Choose tool: } \texttt{calculator}(\text{“}37 \times 89\text{”})$$

Tool calling: Example

Action chosen: $a_t = \texttt{calculator}(\text{"}37 \times 89\text{"})$

Step 3: Execute tool (deterministic!)

$$o_t = \texttt{calculator}(\text{"}37 \times 89\text{"}) = 3293$$

Step 4: Update state with observation

$$\begin{aligned} s_t &= [\text{“What is } 37 \times 89\text{?”}] \\ s_{t+1} &= s_t \oplus o_t \\ &= [\text{“What is } 37 \times 89\text{?”}, \text{“Result: 3293”}] \end{aligned}$$

Step 5: Generate final response using updated state

$$P(x_t \mid s_{t+1}; \theta) \to \text{“The answer is 3293”}$$

Key: Tool provides **verified** information that LLM incorporates!

Full Probability with Tools

Complete trajectory probability:

$$
\begin{aligned}
P(\tau) = P(s_0) \prod_{t=0}^{T-1} \Big[& P(\mathsf{type}_t \mid s_t) \\
& \cdot P(a_t \mid \mathsf{type}_t, s_t) \\
& \cdot P(o_t \mid a_t) \\
& \cdot P(s_{t+1} \mid s_t, o_t) \Big]
\end{aligned}
$$

Full Probability with Tools

What does each component mean:

- $P(\text{type}_t \mid s_t)$: Choose text vs tool
- $P(a_t \mid \text{type}_t, s_t)$: Specific action given type
- $P(o_t \mid a_t)$: Tool execution (often deterministic)
- $P(s_{t+1} \mid s_t, o_t)$: State update

Full Probability Decomposition: Example

Full trajectory for "What is 37×89?"

$$P(\tau) = P(s_0) \cdot \underbrace{P(\text{tool} \mid s_0; \theta)}_{0.85} \cdot \underbrace{P(\texttt{calc} \mid \text{tool}, s_0; \theta)}_{0.95}$$
$$\cdot \underbrace{P(o_0 = 3293 \mid \texttt{calc})}_{1.0 \text{ (deterministic)}} \cdot P(s_1 \mid s_0, o_0)$$

Compare without tool:

- LLM guessing: $P(\text{"3293"} \mid s_0; \theta) \approx 0.001$ (hallucination risk!)
- With tool: $P(o_0 = 3293 \mid \texttt{calc}) = 1.0$ (guaranteed correct!)

This is why tools matter: Convert uncertain LLM generation into **deterministic verified computation**

So, what did we learn?

1. Base LLM	$x_t \sim P(\cdot \mid x_{<t})$ Domain: Vocabulary V
2. LLM Agent	$a_t \sim \pi(\cdot \mid s_t, G)$ Domain: Action space $\mathcal{A}$ NEW: State updates $s_t \to s_{t+1}$
3. LLM + Tools	$a_t \sim P(\cdot \mid s_t)$ $a_t \in \mathcal{A}_{\text{text}} \cup \mathcal{A}_{\text{tool}}$ NEW: Tool execution $o_t = \text{tool}(\text{args})$ NEW: State update $s_{t+1} = s_t \oplus o_t$

How Probability Changes

Evolution of the distribution:

1. **Base LLM**

$$P(\text{next\_token} \mid \text{context}) \text{ over } \approx 50K \text{ tokens}$$

2. **Agent**

$$P(\text{action} \mid \text{state}, \text{goal}) \text{ over action space}$$

Could be: $\{\texttt{generate\_text}, \texttt{search}, \texttt{calculate}, \texttt{stop}\}$

3. **Tool-augmented**

$$\begin{aligned} P(\text{action} \mid \text{state}) = & \, p_1 \cdot P(\text{token} \in V) \\ & + p_2 \cdot P(\text{tool\_call}) \end{aligned}$$

Chain of Thought (CoT)

Problem: Roger has 5 tennis balls. He buys 2 cans with 3 balls each. How many balls does he have now?

Without CoT:
Q: [Problem]
A: 11

With CoT (add: "Let's think step by step"):
Q: [Problem] Let's think step by step.
A: Roger started with 5 balls. 2 cans with 3 balls each is $2 \times 3 = 6$ balls. $5 + 6 = 11$. The answer is 11.

Key idea: Prompt LLM to show intermediate reasoning steps

Chain of Thought (CoT)

Standard LLM:

$$P(y \mid x; \theta)$$

Generate answer y directly from question x

Chain of Thought:

$$P(y, r \mid x; \theta) = P(r \mid x; \theta) \cdot P(y \mid x, r; \theta)$$

where $r = (r_1, r_2, \ldots, r_k)$ = reasoning steps

Generation process:

1. Sample reasoning: $r \sim P(\cdot \mid x; \theta)$
2. Sample answer: $y \sim P(\cdot \mid x, r; \theta)$

Intuition: Intermediate steps provide context $\rightarrow$ better response

Blocks World Planning

Task: Plan to stack blocks correctly

Blocks World Planning

Base LLM approach:

$$P(\text{“move(a,1,b)”} \mid \text{context}) = \prod_i P(c_i \mid c_{<i})$$

where c_i are characters

Problem: No guarantee this is a valid move!
(Block a might have something on top)

Blocks World Planning

Agent with Prolog Tool:

Action space:

$$\mathcal{A} = \{\texttt{generate\_text}\} \cup \{\texttt{prolog\_plan}(s_0, G)\}$$

Decision:

$$P(a_t \mid s_t) = \begin{cases} 0.8 & \text{if } a_t = \texttt{prolog\_plan}(\ldots) \\ 0.2 & \text{if } a_t = \texttt{generate\_text} \end{cases}$$

Blocks World Planning

Tool execution:

$$o_t = \texttt{prolog\_plan}(s_0, G) = [\mathsf{move}(c, a, 2), \mathsf{move}(a, 1, b), \ldots]$$

Updated state:

$$s_{t+1} = s_t \oplus \{\text{"plan"} : o_t\}$$

Key property:

o_t is **deterministic** and **guaranteed correct**
(from Prolog formal planner)

Blocks World Planning

Summary:

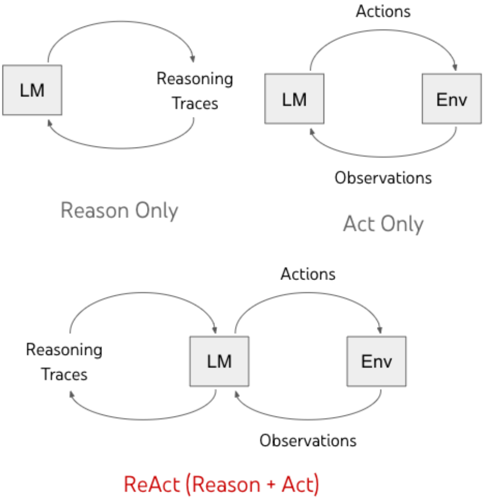
Aspect	**Base LLM**	**Agent**	**Agent+Tools**
Output	$x_t \in V$	$a_t \in \mathcal{A}$	$a_t \in \mathcal{A}_{\text{text}} \cup \mathcal{A}_{\text{tool}}$
Probability	$P(x_t \mid x_{<t})$	$P(a_t \mid s_t, G)$	$P(a_t, \text{type} \mid s_t)$
State	None	s_t updated by actions	s_t updated by o_t (tool observations)
Guarantees	None	None	Tool-dependent

Blocks World Planning

1. Base LLM: Pure text generation over vocabulary
2. Agent: Probability over *actions* that affect state
3. Tool-augmented: Extended action space with tool calls
4. Critical insight: Tool observations can be **deterministic** and **verified**, unlike LLM generations

One reason why hybrid systems (LLM + symbolic tools) work!
See: LMLF (AAAI 2024), GenMol (arXiv, 2025)

ReAct: Reasoning + Acting



ICLR 2023

ReAct: Reasoning + Acting

Core Idea: Interleave Thought (reasoning: CoT) and Action (tool use)

Traditional: Think → Act → Done
ReAct: Think → Act → Observe → Think → Act → ...

Trajectory format:

$$\begin{aligned}\tau = [&\text{Thought}_1, \text{Action}_1, \text{Observation}_1,\\ &\text{Thought}_2, \text{Action}_2, \text{Observation}_2,\\ &\ldots,\\ &\text{Thought}_n, \text{Action}_n]\end{aligned}$$

Key benefit: Agent can **adapt** based on what it observes

ReAct: Reasoning + Acting

Example: Multi-hop question

Question: "What is the capital of the country where the Eiffel Tower is located?"

Thought 1: I need to find where the Eiffel Tower is
Action 1: `search("Eiffel Tower location")`
Observation 1: "The Eiffel Tower is in Paris, France"

Thought 2: Now I know it's in France. I need France's capital
Action 2: `search("capital of France")`
Observation 2: "Paris is the capital of France"

Thought 3: I have the answer
Action 3: `finish("Paris")`

ReAct: Reasoning + Acting

Example: Blocks world

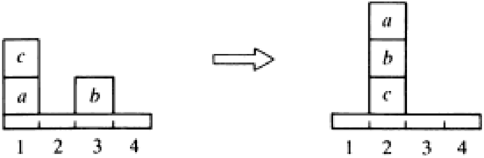


Figure 17.1 A planning problem in the blocks world: find a sequence of actions that achieve the goals: *a* on *b*, and *b* on *c*. These actions transform the initial state (on the left) into a final state (on the right).

Problem: Stack blocks a on b on c
Initial: c on a, a on 1, b on 3

ReAct: Reasoning + Acting

Thought 1: I need to plan, but should verify initial state first
Action 1: `verify_state("c on a, a on 1, b on 3")`
Observation 1: "State is valid"

Thought 2: Now use Prolog planner for guaranteed correct plan
Action 2: `prolog_plan(initial, goals)`
Observation 2: "Plan: [move(c,a,2), move(a,1,b), move(b,3,c)]"

Thought 3: Plan looks correct. Check if valid moves?
Action 3: `verify_plan(plan)`
Observation 3: "All moves are valid"

Thought 4: Verified! Return plan
Action 4: `finish(plan)`

ReAct: Reasoning + Acting

Key point:

Agent (LLM) = *when* and *how* to use tool
Tool (Prolog) = *what* the correct answer is

ReAct: Reasoning + Acting (summary)

Policy generates alternating thoughts and actions:

At step t (thought step):

$$\text{Thought}_t \sim P_\theta(\cdot \mid s_t, \text{context})$$

Then (action step):

$$\text{Action}_t \sim \pi(\cdot \mid s_t, \text{Thought}_t)$$

Then (observation):

$$\text{Obs}_t = \text{execute}(\text{Action}_t)$$

State update:

$$s_{t+1} = s_t \oplus [\text{Thought}_t, \text{Action}_t, \text{Obs}_t]$$

This allows humans to see: Explicit reasoning traces.

Demo

$\boxed{\texttt{llm\_tools\_prolog\_blocksworld.py}}$