AI Agents

Building LLMs That Take Action and Use Tools

NLP Course – Lecture 2

Advanced Topics in Natural Language Processing

What If LLMs Could DO Things?

LLMs Today

- Excellent at generating text
- Answer questions from training data
- No ability to take actions
- Cannot interact with the world

LLM Agents

- Use tools (search, code, APIs)
- Execute multi-step plans
- Observe results and adapt
- Accomplish real-world tasks

This lecture: Building AI systems that take action

Agents transform LLMs from passive responders to active problem solvers.

Beyond Question-Answering

LLMs Are Great At...

- Generating text
- Summarizing documents
- Translating languages
- Answering questions

But what if we want them to **DO** things?

Real-World Tasks Require Action

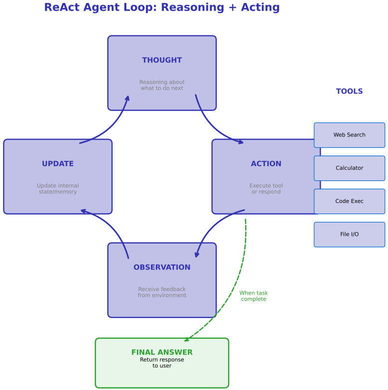
- Book a flight (API calls)
- Write and run code (execution)
- Search the web (retrieval)
- Manage files (system access)
- Send emails (communication)

The Gap: LLMs generate text, but can't act.

Solution: Give LLMs the ability to use *tools* and reason about *when* to use them.

This is the leap from "AI assistant" to "AI agent"

The Agent Loop: Perceive, Plan, Act, Observe



Core cycle: User task → LLM decides action → Tool executes → Result feeds back → Repeat until done

---

Agents are LLMs in a loop – the magic is in the orchestration, not a new architecture

The ReAct Pattern: Reasoning + Acting

ReAct Example

`User: What's 15% of Apple's current market cap?`

Thought: I need to find Apple's current market cap first.
Action: search_web("Apple market cap 2025")
Observation: Apple market cap: $3.2 trillion

Thought: Now I can calculate 15% of 3.2 trillion.
Action: calculate("3200000000000 * 0.15")
Observation: 480000000000

Thought: I have the answer.
Final Answer: 15% of Apple's market cap is $480 billion.

Key Innovation

Interleave:

- Reasoning (Thought)
- Acting (Tool use)
- Observing (Feedback)

Why It Works

LLMs are good at reasoning about *what to do next* given context.

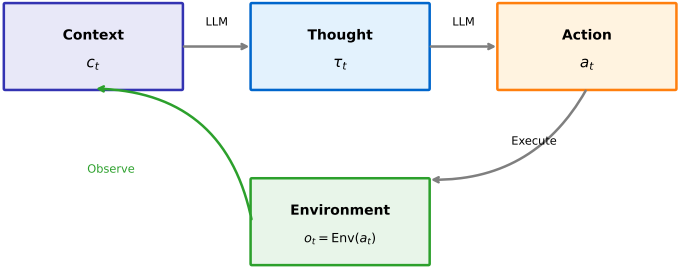
Formal Loop

$\tau_t = \text{LLM}(c_t, h_{<t})$ (thought)
$a_t = \text{LLM}(\tau_t, h_{<t})$ (action)
$o_t = \text{Env}(a_t)$ (observation)

Yao et al. (2023): "ReAct: Synergizing Reasoning and Acting in Language Models"

Formal Agent Loop: The Mathematics

The Agent Loop: Formal Definition



Agent Loop Equations

1. Generate Thought: $\tau_t = \text{LLM}(c_t, h_{<t})$
2. Select Action: $a_t = \text{LLM}(\tau_t, h_{<t})$
3. Execute & Observe: $o_t = \text{Env}(a_t)$

History: $h_t = (c_0, \tau_0, a_0, o_0, \ldots, o_{t-1})$

Terminate: a_t = FINISH

This formalism underlies all modern agent frameworks – the LLM is both brain and narrator

Tool Use: The Key Innovation

Function Calling Format

LLMs learn to output structured tool calls:

```
{
  ‘‘tool’’: ‘‘search_web’’,
  ‘‘parameters’’: {
    ‘‘query’’: ‘‘AAPL stock price’’
  }
}
```

Available Tool Types

- Web search
- Calculator / code interpreter
- File system access
- API calls (weather, stocks, etc.)
- Database queries

How It Works

1. Define tools with JSON schema
2. Include tool definitions in prompt
3. LLM outputs tool call (structured)
4. System executes tool
5. Return result to LLM
6. Repeat until done

OpenAI Function Calling

Built into GPT-4, Claude, etc.:
Models trained to output valid JSON for tool calls.

Connection to RAG

RAG is just a "retrieval tool" that agents can use!

Tool use transforms LLMs from text generators to action-capable systems

Agent Frameworks: Evolution

Timeline

- **2022:** ReAct (Google) – Reasoning + Acting
- **2023:** Toolformer (Meta) – Self-supervised tool learning
- **2023:** AutoGPT / BabyAGI – Autonomous task completion
- **2024:** LangChain Agents – Production frameworks
- **2024:** Microsoft AutoGen – Multi-agent systems
- **2025:** Agentic AI – Enterprise deployment

Current Landscape

Frameworks:

- LangChain / LangGraph
- LlamaIndex
- CrewAI
- AutoGen

Trends:

- Multi-agent collaboration
- Specialized agents for tasks
- Human-in-the-loop workflows
- Enterprise security/compliance

We're at the "early internet" stage of agents – rapid evolution, no clear winner

LangChain and LangGraph: Key Concepts

LangChain Core Concepts

LCEL (LangChain Expression Language):

- `prompt | llm | parser` – Pipe syntax
- Composable, streamable, async-ready
- Built-in retry/fallback logic

Key Abstractions:

- `ChatModel` – LLM interface
- `Tool` – Function with schema
- `Retriever` – Document search
- `Memory` – Conversation history

LangGraph for Complex Agents

Graph-Based Workflows:

- `StateGraph` – Define typed state
- `add_node()` – Add processing steps
- `add_edge()` – Connect nodes
- `add_conditional_edges()` – Branching

Key Features:

- Cycles for iterative agents
- Checkpointing for recovery
- Human-in-the-loop breakpoints
- Multi-agent coordination

When to Use: LangChain for simple RAG/chains — LangGraph for stateful agents with cycles

LangChain ecosystem dominates (2024), but alternatives exist: LlamaIndex, CrewAI, AutoGen

Current Agent Limitations

Reliability Issues

- Agents get stuck in loops
- Wrong tool selection
- Hallucinated tool parameters
- Failure to know when to stop

Cost Accumulation

- Each step = API call
- Complex tasks = many calls
- Costs can spiral quickly

Security Concerns

- Tool access = system access
- Prompt injection attacks
- Unintended actions

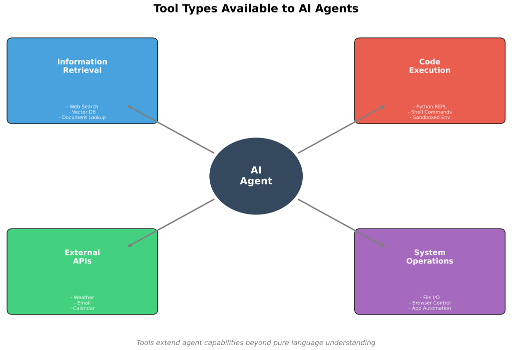
What Works Today

- Well-defined, bounded tasks
- Human oversight/approval
- Retrieval-heavy workflows
- Single-domain expertise

"Agents are promising but not production-ready for autonomous operation." – 2024 consensus

Current agents work best with human oversight and well-defined tasks.

Tool Types: A Comprehensive View



Information Retrieval: Web search, RAG, SQL, SPARQL
Code Execution: Python REPL, shell, Jupyter, sandboxed
External APIs: Weather, email, CRM, custom business
System Operations: File I/O, browser automation, GUI

Key Principle: Tools should be *atomic*, *well-documented*, and *safely sandboxed*.

The power of agents comes from combining multiple tool types in a single workflow.

Function Calling and Structured Outputs

JSON Schema Definition

```
{
  "name": "search_web",
  "description": "Search the web",
  "parameters": {
    "type": "object",
    "properties": {
      "query": {"type": "string"}
    },
    "required": ["query"]
  }
}
```

LLM Output

```
{
  "tool": "search_web",
  "arguments": {"query": "AAPL"}
}
```

Why Structured Outputs Matter

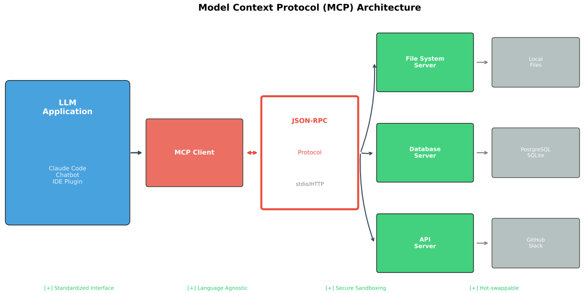
- Guaranteed valid JSON
- Type checking at generation
- Reliable tool invocation
- No regex parsing needed

Constrained Decoding

- Grammar-constrained generation
- Only valid tokens sampled
- 100% schema compliance
- Supported by GPT-4, Claude, etc.

Structured outputs eliminate parsing errors – critical for production agent systems.

Model Context Protocol (MCP)

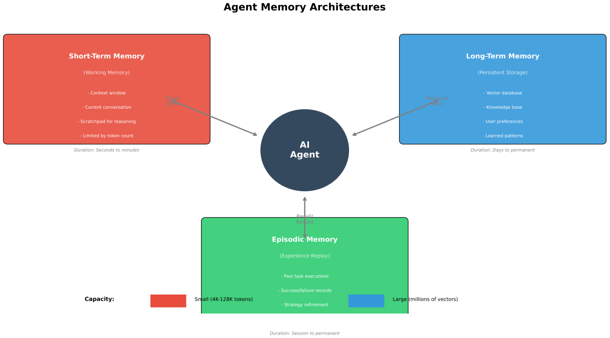


The Problem: Every tool has different API, no standard for discovery
MCP Solution (Anthropic, 2024): Standard protocol, self-describing tools, dynamic discovery

Key Components: `Resources` (data), `Tools` (actions), `Prompts` (templates)
Adoption: Claude Desktop native, open source spec

MCP aims to be the "USB for AI" – standardizing how models connect to tools.

Agent Memory Architectures



Short-Term: Context window, task state, recent outputs

Long-Term: Vector store, summarized history, user preferences

Episodic: Past experiences, learning from success/failure

Challenge: Deciding what to remember vs. forget – information overload degrades performance.

Memory is what transforms a stateless LLM into a persistent assistant.

Multi-Agent Systems

Why Multiple Agents?

- Specialization (expert agents)
- Division of labor
- Cross-checking results
- Complex workflow orchestration

Communication Patterns

- *Sequential*: Agent A $\rightarrow$ Agent B $\rightarrow$ Agent C
- *Hierarchical*: Manager delegates to workers
- *Debate*: Agents argue, human decides
- *Voting*: Consensus among agents

Example: Code Review System

1. `Coder Agent`: Writes code
2. `Reviewer Agent`: Finds issues
3. `Security Agent`: Checks vulnerabilities
4. `Manager Agent`: Coordinates, decides

Frameworks

- AutoGen (Microsoft)
- CrewAI
- LangGraph (multi-actor)
- CAMEL (role-playing)

Multi-agent systems can outperform single agents but add coordination complexity.

Agent Evaluation: Metrics and Benchmarks

Task Completion Metrics

- Success rate (task completed?)
- Steps to completion
- Cost per task (API calls)
- Time to completion

Quality Metrics

- Correctness of results
- Tool selection accuracy
- Reasoning trace quality
- Recovery from errors

Popular Benchmarks

- **WebArena**: Web navigation tasks
- **MINT**: Multi-turn interaction
- **AgentBench**: General agent tasks
- **SWE-bench**: Software engineering

Challenges

- Non-deterministic outputs
- Environment variability
- Expensive to run at scale
- Real vs. simulated environments

Evaluation is hard: same agent can succeed or fail on identical tasks due to stochasticity.

Debugging Agent Failures

Common Failure Modes

- Infinite loops: Agent repeats same action
- Tool confusion: Wrong tool for task
- Hallucinated params: Invalid arguments
- Premature stop: Quits before done
- Context overflow: Loses track of goal

Debugging Techniques

- Trace logging (every step)
- Breakpoints at tool calls
- Replay from checkpoints
- Manual intervention hooks

Mitigation Strategies

Loop Prevention:

- Max iterations limit
- Action history deduplication
- Escalation to human

Reliability:

- Retry with backoff
- Fallback tools
- Confidence thresholds
- Structured validation

Tools: LangSmith, Weights & Biases, Phoenix

Production agents need observability – you cannot improve what you cannot see.

Key Takeaways: AI Agents

1. **Agents extend LLMs** from text generators to action takers
2. **ReAct pattern**: Think $\rightarrow$ Act $\rightarrow$ Observe $\rightarrow$ Repeat
3. **Tool use** enables interaction with external systems
4. **MCP** standardizes how agents connect to tools
5. **Memory** is critical for maintaining context across actions
6. **Evaluation** of agents is challenging but essential

Key Insight: Agents are still unreliable for complex tasks – human oversight remains essential.

Agent capabilities are rapidly improving but require careful deployment.

Further Reading: AI Agents

Foundational Papers:

- Yao et al. (2023) - "ReAct: Synergizing Reasoning and Acting"
- Schick et al. (2023) - "Toolformer"
- Significant-Gravitas - AutoGPT

Frameworks & Tools:

- LangChain, LangGraph, CrewAI
- Claude Code, Cursor, Devin
- Model Context Protocol (MCP)

Repository: github.com/Digital-AI-Finance/Natural-Language-Processing