NSF Access Regional AI Workshop SOCAL Edition

USC Campus

Prakashan Korambath, OARC / CESMII, UCLA

January 22, 2026

CESMII
THE SMART MANUFACTURING INSTITUTE

UCLA Advanced Research Computing

Ambiguity in Natural Language

- I arrived at the **bank** after crossing the ***river/road*.**

Note: ‘**bank’** is ambiguous—could mean a financial institution or riverbank.

- Mary saw the **bicycle** through the store window and she wanted **it.**

Note: ‘**it’** refers to the bicycle, context is crucial.

- She looked at **it** longingly through the store **window** and pressed her nose against **it.**

Note: ‘**it’** refers to the window.

- The singer played **bass**. / The fish was **bass**.

Note: Requires context to know the meaning of **bass**.

Alan Turing: Can Machines Think? The Turing Test, originally called the "imitation game," involves a human judge having a text-based conversation with both a human and a machine, without knowing which is which.

The Turing Test

- A human judge communicates via text with two unseen contestants one human, one machine attempting to determine which is which based solely on their responses.
- Turing test redefined the intelligence as a property characterized by output indistinguishable from that of a human.
- The shift from internal cognitive state to external behavioral performance remains a cornerstone of artificial intelligence research

AI Timeline

- 1950 — Turing Test (Behavioral Intelligence)
  - Turing, A. M. (1950). *Computing Machinery and Intelligence*. **Mind**, 59(236), 433–460.
    - It shifted AI from metaphysical speculation to empirical evaluation.
    - It placed **language** at the center of intelligence, foreshadowing modern NLP and generative models.
- 1956 — Dartmouth Conference (AI is Born)
  - McCarthy, J. et al. (1956). *Dartmouth Summer Research Project on Artificial Intelligence*.
- 1958 — Perceptron
  - Rosenblatt, F. (1958). *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*. Psychological Review.
- 1960s – Symbolic AI / Expert Systems
  - AI research was dominated by **rule-based systems** and **expert systems**, such as ELIZA and MYCIN.
- 1986 — Backpropagation
  - Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning representations by back-propagating errors*. Nature.
- 1980s – Recurrent Neural Networks
  - Elman, J. L. (1990). *Finding Structure in Time*. Cognitive Science.
    - Internal hidden state
    - Parameter sharing across time steps
    - Sequential dependency modeling
- 1997 — LSTM
  - Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory*. Neural Computation.
- 2014 — Seq2Seq Learning
  - Sutskever, I., Vinyals, O., & Le, Q. V. (2014). *Sequence to Sequence Learning with Neural Networks*.
- 2015 — Attention Mechanism
  - Bahdanau, D., Cho, K., & Bengio, Y. (2015). *Neural Machine Translation by Jointly Learning to Align and Translate*.
    - Allowed models to dynamically focus on relevant parts of the input sequence during decoding.
- 2017 — Transformer Architecture
  - Vaswani, A. et al. (2017). *Attention Is All You Need*. arXiv:1706.03762.

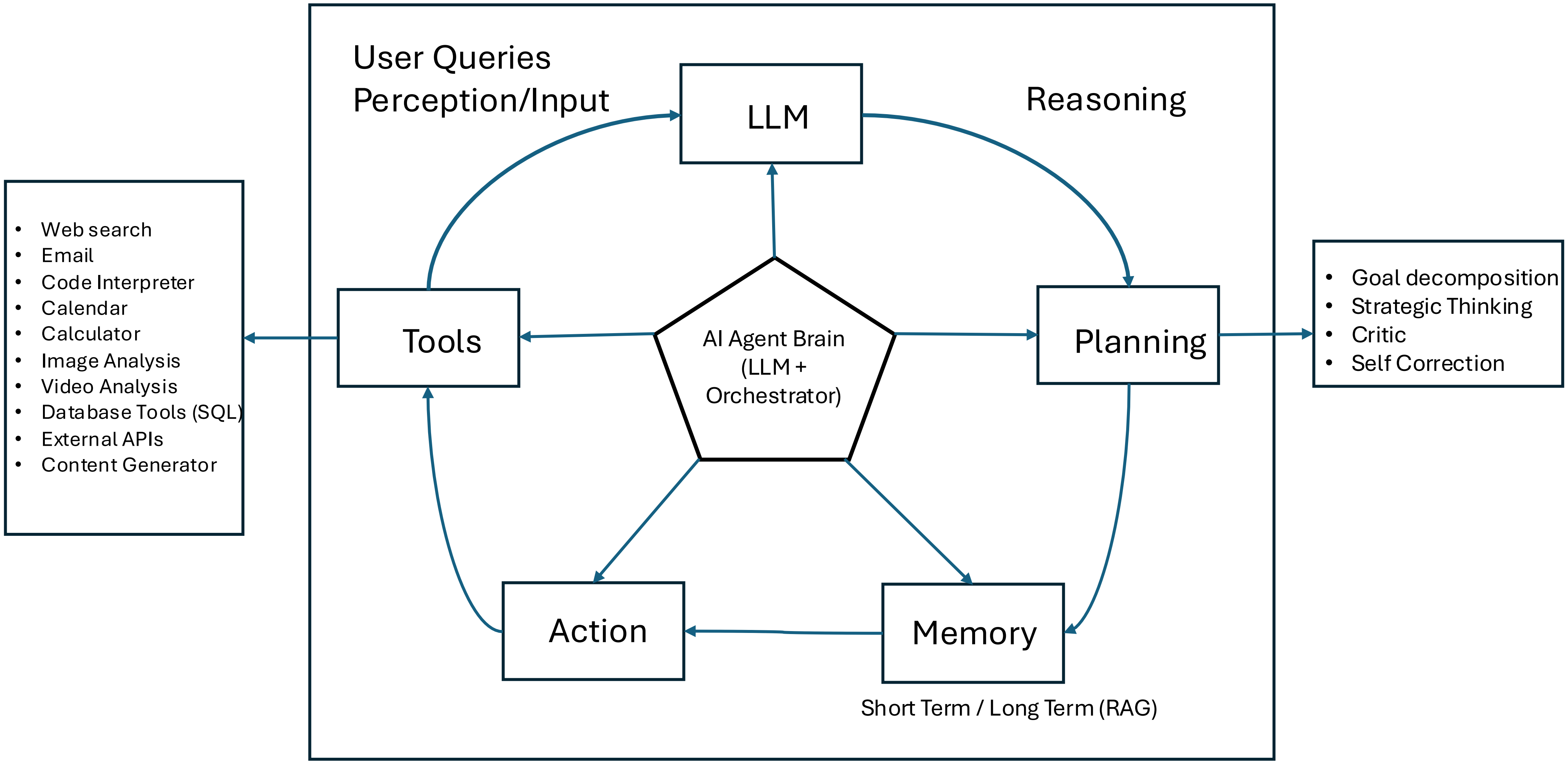
Limitations of LLM

- **LLMs** are built on an architecture called **Transformer** and trained using **autoregression** (predicting next word or token in a sentence).
- **Autoregressive models** analyze a sequence of words (x_1, x_2, …, x_n) to estimate the probability of the next word (x_{n+1})
- This approach can lead to **hallucination** (error propagation): if an early mistake occurs, it becomes part of the "ground truth" for the rest of the output.
- **No Feedback Loop:** Standard LLMs generate responses in a single pass and do not verify their own output. For example, if an LLM writes faulty code, it won't recognize the error unless a human intervenes.
- **No Actionable Power:** LLMs can provide instructions (e.g., how to book a flight) but cannot perform actions like logging into a website and completing the purchase.
- **Knowledge Cutoff:** The model's knowledge is fixed at the end of its training. It cannot access current news, stock prices, or private files unless connected to external tools (e.g., RAG).
- **Context Window Limits:** Even as context windows grow, LLMs still struggle with "Lost in the Middle" syndrome, often missing or ignoring details buried in lengthy documents.
- **No Long-Term Memory:** Each new chat session starts with a blank slate. Standard LLMs do not learn from past mistakes or remember user preferences across sessions.

The Case for Agentic AI: From "Predictor" to "Actor"

Feature	Text-only LLM (The "Scholar")	Multimodal LLM (The "Observer")	Agentic AI (The "Worker")
Primary Perception	**Indirect:** Learns about the world only through text descriptions.	**Direct:** Can "see" pixels and "hear" waveforms directly.	**Environmental:** Gathers data from APIs, sensors, and live web-feeds.
Reasoning Style	**Linguistic:** Uses logic found in sentences and symbols.	**Contextual:** Can reason about spatial layouts, tones, and visual cues.	**Iterative:** Uses a "Think → Act → Observe" loop to test logic.
Action & Control	**None:** It only generates a text response to your prompt.	**None:** It describes what it sees but cannot click or edit it.	**High:** Uses tools (Python, Browsers, APIs) to perform real-world tasks.
Error Handling	**Passive:** If it starts wrong, it continues the error (Autoregressive Trap).	**Passive:** If it misinterprets a pixel, it hallucinates the rest.	**Active:** Runs a "Self-Correction" step; if a tool fails, it tries a new plan.
Planning	**Static:** Writes a "to-do list" but never checks it off.	**Visual:** Can plan based on a visual goal (e.g., "Build a site like this image").	**Dynamic:** Breaks goals into sub-tasks and adjusts them in real-time.
Learning	**Static:** Knowledge is "frozen" on its training cutoff date.	**Static:** Perception is fixed; it cannot learn from new images in real-time.	**Adaptive:** Learns via "Memory" and RAG (retrieving new info from the web).

The Agentic Loop

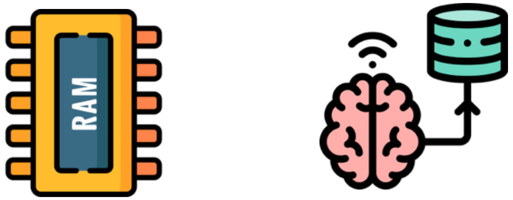
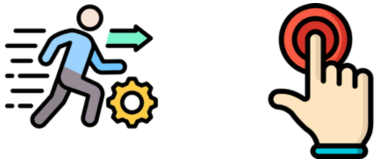


Dynamic entity that can reason, act and use external tools to automate complex tasks. It can make its own decisions and perform tasks with little to no human intervention.

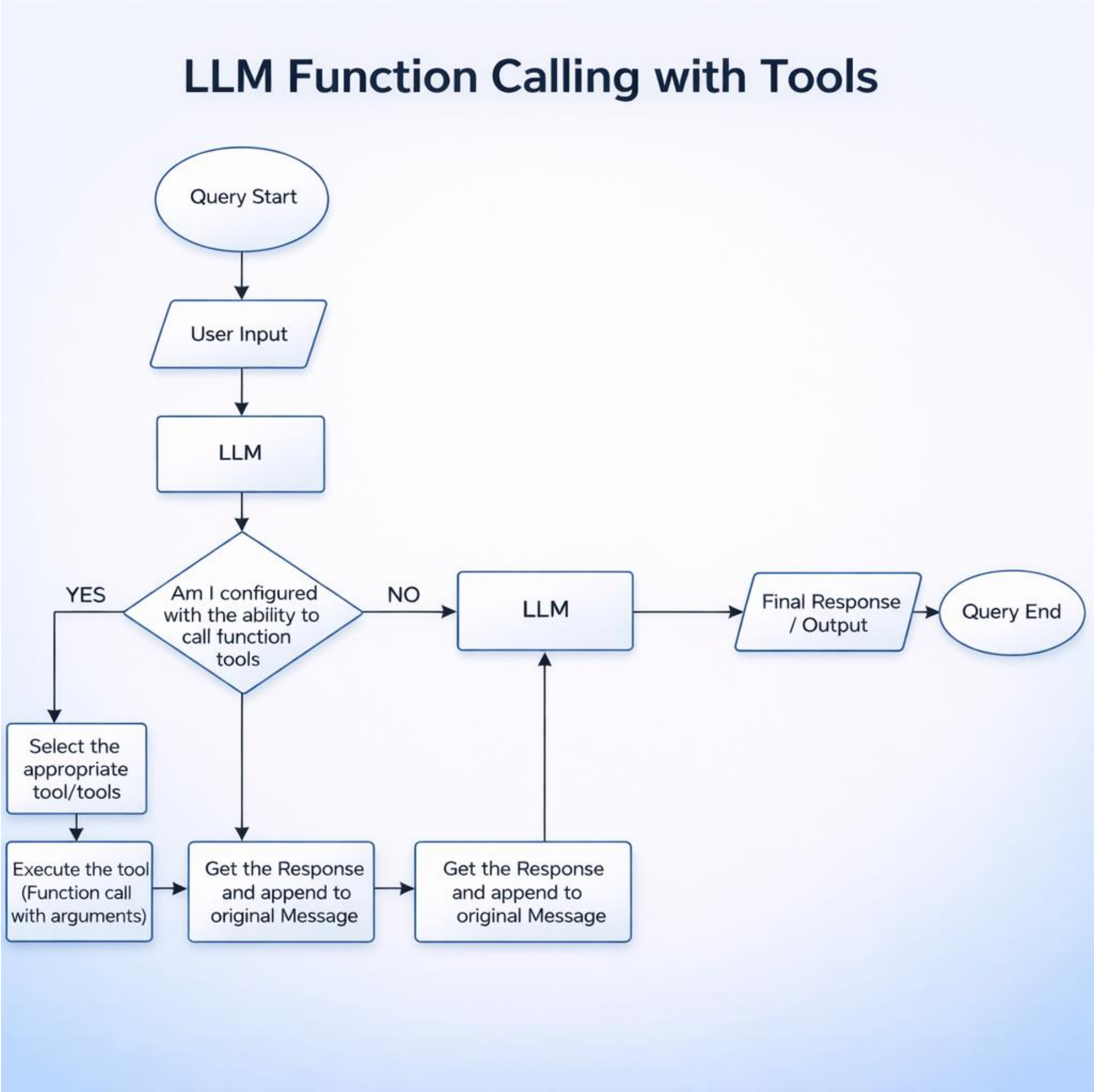
CESMII THE SMART MANUFACTURING INSTITUTE

UCLA Advanced Research Computing

How Agent’s work

Component		Function
1. Perception		Gathers and interprets data from the environment (APIs, databases, sensors, web scraping, Image and Optical character recognition).
2. Reasoning (LLM)		The "brain"—uses a Large Language Model (LLM) to analyze data, break down the goal into subtasks, and formulate a plan (Orchestration).
3. Memory		Retains context (short-term) and knowledge (long-term) for continuity and reference. (Conversation history, Knowledge graphs, vector database)
4. Action		Executes the plan by utilizing external Tools (e.g., calling APIs, running code, accessing external systems).
5. Learning		Evaluates the outcome and adapts/refines its strategy for future tasks.
6. Limitations		Context Window size, Scalability of Tool use, Cost and Latency, Safety in autonomous mode.

Agents in Action



- Examples tool call can be a function that returns current weather data, web search, SQL queries, etc.
- LLM Orchestrator selects the calls based on the function description and expected arguments.
- It is a two-step process.
- First determine what function to call and then call the actual function with arguments.
- Next, append the message from the function call to the original query and call LLM again and return the final response to the user.
- In case LLM must make multiple function call they can be processed sequentially or in parallel

Agentic AI Tool Metadata Definition and Context Injection

1. **Tool Metadata Definition (API Documentation):**

    - **Function Name:** Unique Identifier for the tools (e.g., get_current_weather, search_web, send_email)
    - **Detailed Description (The most Critical Part):** A clear and concise, natural language explanation of what the tool does and when it should be called. LLM heavily relies on this description to perform semantic matching with the user's request.
    - **JSON Schema**: A formal definition of the function's input requirements (parameters). This includes
        - **Parmeter Names**: Required input fields (e.g. location, recipient list, query, filepath)
        - **Data Types**: (e.g. string, integer, array)
        - **Constraints/Descriptions**: Explanation of whether the parameters are required or optional

2. **Context Injection (The System Prompt):**

    - **Instruction**: The system prompt explicitly instructs the LLM that if its internal knowledge is insufficient to fulfill the user's request, it must output a structured, specific format (usually a JSON object or a Python function call) indicating which tool to use and with what parameters. (You must use web search tool to get latest data or call weather tool to get current weather.)
    - **The available tools**: The tool definitions from Step 1 are injected directly into this prompt.

LLM Reasoning and Selection (The Decision)

When the agent receives a user query (e.g.,"What's the weather like in Los Angeles, right now?"), the LLM follows an internal chain of reasoning:

1. **Semantic Match:** The LLM compares the user's intent to the descriptions of all available tools provided in the context. It quickly identifies that the get_current_weather (str location): tools is the most semantically relevant function to call, far surpassing the tools like send_mail or query_database.

2. **Schema Compliance (Parameter Filling:** (Once a tools is selected, the LLM uses the JSON Schema to determine what inputs (arguments) are required. It then extracts the necessary information from the user's original query ("Los Angeles") to correctly populate the parameters.

3. **Tool Call Output:** The LLM does not execute the tool itself. Instead, it outputs a structured response following the required syntax, such as a function call block.

```
{
        "function_name": "get_current_weather",
        "parameters": {
                "location": "Los Angeles"
        }
}
```

CESMII THE SMART MANUFACTURING INSTITUTE

UCLA Advanced Research Computing

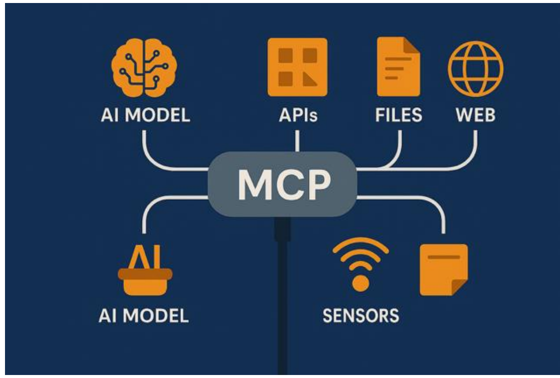
Execution and Observation (The Action)

1. **External Orchestrator:**
   - The Agent's orchestrator component (not the LLM) intercepts this structured output. The orchestrator enables dynamic, real-world actions.
2. **Validation & Execution:**
   - The orchestrator validates the output against the JSON Schema.
   - Calls the actual external function (e.g., get_current_weather API call), and receives the result (e.g., "It is 18°C and cloudy").
3. **Observation Loop:**
   - This result is then injected back into the LLM's context window as an "Observation" or a “Tool Response”. The observation loop ensures the LLM can refine its reasoning based on the actual results.

The LLM then continues its reasoning cycle, using the factual tool output to synthesize the final, accurate answer for the user.

Model Context Protocol (MCP) (The "USB-C" of AI or the Universal Remote)

- **Standardized Communication:** MCP provides a single, open standard that allows any AI model to "talk" to any data source or tool. It acts like a **USB-C port for AI**, replacing messy, custom-coded integrations with a universal plug-and-play interface. Before MCP, every model (OpenAI, Gemini, Claude) needed its own code.
- **Dynamic Discovery:** Instead of hard-coding what an AI can do, an agent can ask an MCP server, *"What tools do you have?"* The server responds with a list of capabilities, and the agent learns how to use them **on the fly**.
- **Reduced Hallucinations:** By providing a direct, structured line to real-world data (like your local files, a database, or a live web search), MCP replaces the model's "guessing" with **factual grounding**.
- **Efficiency & Security:** MCP moves logic out of the model's limited "context window" and onto the server. This keeps the agent fast and cheap while ensuring sensitive data stays behind your secure server gates.
- **Scalability:** You can connect an agent to hundreds of different tools across an enterprise without bloat. The agent only "loads" the specific tool definition it needs for the task at hand.
- **No API Keys Needed** (if your python code does not need it, but the client will need to interpret): You don't need to sign up for Google or OpenAI APIs. It (server part) runs entirely on your local machine.
- **MCP is built on the JSON-RPC 2.0** standard - transport-agnostic (local STDIO, cloud http with SSE (server sent event)), lightweight, etc.



Starting Tools for MCP (for Python)

- FastMCP:
  - This is the Fast API of MCP. You just use @mcp.tool() decorators on your existing functions. (uv init <folder>; cd <folder>; uv add fastmcp)
- https://modelcontextprotocol.io/docs/getting-started/intro
- https://modelcontextprotocol.io/docs/develop/build-server
  - It walks you through building a weather-fetching server.
- Claude Desktop, VS Code:
  - You need hosts like these tools to see MCP.
- MCP Inspector:
  - Run npx @modelcontextprotocol/inspector uv --directory path/to/server run your_server.py. (View usually at http://localhost:3000)
- UV (Python package manager, instead of Anaconda or pip)