Agentic AI

An Introduction

LLMs

- LLMs: Given a series of words (tokens) a language model, models the probability of the next word. A model for “Language Completion”
- What else?
  - The weather in Austin yesterday was …
  - John has 3 apples. He gives away 1 and then buys 4 more. He now has …
  - Please buy 300 shares of QQQ …
- Knowledge?
- Reasoning?
- Acting?

LLMs → Reasoning

- LARGE (more params, more data) models began to reason!
  - Logical Reasoning
  - Chain-of Thought Problem Solving
  - Common Sense Inference
  - Mathematical Reasoning
- Why does this happen?
- It is imitated/simulated reasoning, not exactly the same as human reasoning
  - don’t have internal models of logic or facts
  - reasoning can be inconsistent or brittle
  - don’t “understand” in a human sense; they pattern-match effectively

Actions

- Write an Email Vs Send an Email

Reasoning is central

- Tool use and delegation
  - Agents often use tools (e.g., calculators, APIs) or even other agents.
  - It must reason about when and how to use those tools effectively.
- Goal-directed planning
  - “If I want X, then I must do Y and Z first.” Requires multi-step reasoning, not just reacting.
- Decision-making
  - Choosing between options requires evaluating outcomes, costs, risks — all forms of reasoning.
- Self-reflection and correction
  - An agent needs to reason about its own performance and revise plans if they’re not working.

Agents: Wrappers around LLMs

- Core differentiators
  - Tools : take *actions* in the world (query a DB, send email, call APIs)
  - Looping / Autonomy : keeps reasoning + acting until the goal is complete
- Additional capabilities
  - Knowledge (RAG, KB etc)
  - Memory (beyond context window, short/long term)
  - Goals (multi-step planning)
  - Control & Guardrails (safety, limits, permissions)
  - Planning / Reasoning processes (ReAct, Plan-and-Execute)
  - Environment / Context awareness (time, state, user preferences)
  - Collaboration abilities (multi-agent ecosystems, delegation)
  - Personality / Role (shaping communication style and behavior)

Communications

- How Systems Communicate
  - Functions → pass parameters, return values
  - Programs / APIs → structured calls (e.g., JSON, gRPC, REST)
  - Client–Server → request/response over sockets, HTTP
  - Web Browsers → HTML, JavaScript, HTTP requests, DOM events
  - Apps → SDKs, OS-level APIs, message passing

- Agents communicate in natural language, not just parameters.

A ReAct Agent

- Reason → Act → Observe → Repeat
  - Goal-driven loop that alternates brief reasoning with tool use.
  - Language is the glue: the agent explains (succinctly) what it needs, calls a tool, reads the result, then adapts.
  - Why it works: incremental feedback reduces hallucination and helps with planning under uncertainty.
  - Guardrails: cap steps/seconds, validate tool outputs, and stop on success criteria.

An Example

Goal: “Email the team a lunch spot within 10 minutes’ walk, open now, avg price <$15.”

Step	Reasoning (summary)	Action (tool)	Observation
1	Need candidate restaurants nearby and open now.	WebSearch("cheap lunch open now near CS building")	3 candidates with hours & links.
2	Filter by walking time ≤10 minutes.	Maps.distance(candidate addrs, origin)	Two options within 8–10 minutes.
3	Check average prices (must be <$15).	Browser.fetch(menu pages)	One option averages $12–$14.
4	Pick best trade-off: distance + price + ratings.	— (decision step)	Chosen: “Taco Garden” (8 min, $13 avg).
5	Draft and send the email.	Email.send(to=team, body=details)	Email queued/sent; task complete.

The ReAct Prompt

Answer the following questions as best you can. You have access to the following tools:

{tools}

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {input}
Thought:{agent_scratchpad}

Alternative Architectures

Approach	What it is (mental model)	When to use	Strengths	Gotchas
Tool-calling agent (single-turn)	One LLM call that may **choose & call tools** once, then answer. ("Function calling.")	Simple tasks with predictable I/O (translate file, fetch weather, call API, format output).	Fast, cheap, low loop risk; easy to test.	Limited adaptivity; no multi-step planning; brittle if one tool call isn't enough.
ReAct	Loop of **Think → Act → Observe → Repeat**. Uses a scratch pad built from action/observation pairs.	Exploratory tasks with uncertainty that benefit from incremental feedback.	Great intuition; reduces hallucination by grounding each step; flexible.	Can loop; needs **max steps/timeouts**; more tokens & latency.
Plan-and-Execute	Model first creates a **plan** (sub-steps), then executes steps (optionally with ReAct inside each).	Long-horizon tasks with clear decomposition (research, multi-API workflows).	Fewer loops; clearer progress; easier to audit ("here's the plan").	Brittle if initial plan is wrong; needs **re-planning** hooks; a bit more plumbing.
AutoGPT-style	Highly autonomous loop that **sets sub-goals, critiques, retries**, often with long-term memory.	Open-ended goals where human oversight is okay and cost/latency are secondary.	Powerful for discovery; can run unattended; showcases "agentic" flavor.	Unpredictable, costly; strong loop risk; heavy guardrails/memory curation needed.
LangGraph DAGs	**State-machine/graph**: nodes = tools/LLM steps, edges = routing/ conditions; ReAct is one possible subgraph.	Production flows needing reliability, observability, retries, branching, and tests.	Deterministic control; easy to add guardrails, fallbacks, concurrency; great traces.	More engineering up front; you design the rails (but that's the point!).

“Pick a lunch spot”

- Tool-calling agent: “Find a cheap lunch near me.” → calls Search API once → returns a guess.
  If wrong, you must ask again; no self-correction.
- ReAct: Search → see options → measure walking time → fetch menu → choose → email.
  Iterates with evidence each step; needs a recursion_limit.
- Plan-and-Execute: Plan: (1) find candidates, (2) filter by distance, (3) check price, (4) email. Then execute steps; re-plan if a step fails.
- AutoGPT-style: Sets sub-goals, saves notes to memory, tries different searches, maybe overthinks; needs strict budget/step caps.
- LangGraph DAGs: Nodes: search → filter_distance → check_price → decision → send_email; add edges for errors/timeouts; swap a node without changing the rest.

Agents with Memory

- Adding memory to agents can help
  - Personalization
  - Context Continuity
  - Efficiency
  - Long-term reasoning
- Short term vs. Long term

Model Context Protocol (MCP)

- Communication protocol for LLM agents to discover and use tools at runtime
- Open protocol for models/agents to access tools, data, and prompts in a standard way.
- Decouples agent reasoning from tool implementation
- MCP server is a process/web service exposing tools/resources via MCP endpoints