

The LLM & Agent Era

Reading time: ~10 min · Prereqs: Lesson 12 (AI & ML Foundations) · *Snapshot 2026 — the highest-volatility lesson in the course. Brands, versions and prices move quarterly; concepts move slowly.*

Why this matters to you

The most public IT story of the early 2020s is the one happening in this lesson. ChatGPT released to the public in November 2022; three and a half years later, **Large Language Models** and the **agents** built on top of them are in your IDE, your browser, your customer-service queue, your search results, and — if you work in IT — almost every meeting. None of it is magic; all of it is a particular kind of machine learning (Lesson 12) applied at unprecedented scale, with a few clever wrappers (retrieval, tool use, loops). This lesson is the snapshot — *as of 2026* — of what changed, what the vocabulary means, and what new products you'll keep hearing about. Expect this page to be the most-edited file in the course a year from now.

1. What an LLM actually is

A **Large Language Model** is a giant neural network — usually a particular architecture called a **transformer** — trained on a huge corpus of text (the open web, books, code, some private data) with one deceptively simple objective: **predict the next word**. Show the model "The cat sat on the...", ask it what comes next; nudge its weights so it gets closer to "mat" (or whatever the training text actually said). Repeat trillions of times across hundreds of billions of parameters. Out of that one objective, applied at extreme scale, emerges the ability to translate, summarize, write code, hold a conversation, answer questions, and — controversially — *reason* about novel problems. The same recipe that produced Lesson 12's image classifiers, scaled to text and to a much bigger model, produced this.

2. The 2026 player landscape

High-volatility section — refresh annually. As of 2026 the **frontier closed-weight models** are roughly: **Anthropic's Claude** family (Opus, Sonnet, Haiku — known for reasoning, safety focus, and long context); **OpenAI's GPT** family (broadest consumer reach via ChatGPT, strong reasoning models); **Google DeepMind's Gemini** family (deeply integrated with Google Search, Workspace, and Android); and **xAI's Grok** in a smaller but visible role. The **open-weight** side is led by **Meta's Llama** family, with **Mistral**, **Alibaba's Qwen**, and **DeepSeek** as serious frontier-class open challengers — the open-weight gap to frontier closed models is much narrower *as of 2026* than it was in 2023. Closed-weight models you typically use via an API call to the vendor's cloud; open-weight models you can download and run on your own hardware (or via a hosting provider like **Together AI**, **Fireworks**, **Groq**, or **OpenRouter**). The right choice depends on cost, privacy, latency, and how much capability you actually need.

3. Prompts, context windows, and tokens

A **prompt** is just text — typically a **system prompt** (instructions, tone, persona — "You are a helpful assistant...") plus a **user prompt** (the actual request). The model sees them concatenated and produces a response. Everything the model knows about your current task lives in the **context window** — its working memory for one request. Modern context windows *as of 2026* range from ~128K tokens at the small end to 1M+ tokens at the large end (roughly 100 000 to 1 000 000 *words*, depending on language). Anything beyond the window simply isn't visible. Pricing also happens here: vendors charge per million tokens of input and per million tokens of output, with input usually cheaper. Real prices *as of 2026* are in the small-dollars-per-million-tokens range and have been falling roughly an order of magnitude every 18 months — verify before quoting.

4. RAG — giving the model your data

An LLM only knows what it saw during training, which has a **knowledge cutoff** date. To answer questions about your private documents — or anything newer than the cutoff — the standard pattern *as of 2026* is **RAG (Retrieval-Augmented Generation)**: when a question comes in, first **retrieve** the most relevant chunks of your documents (using **embeddings** stored in a **vector database** — see Lesson 07), then **augment** the prompt with those chunks ("Here are five relevant excerpts. Now answer the user's question."), then **generate** the answer. Most enterprise "ask our internal docs"

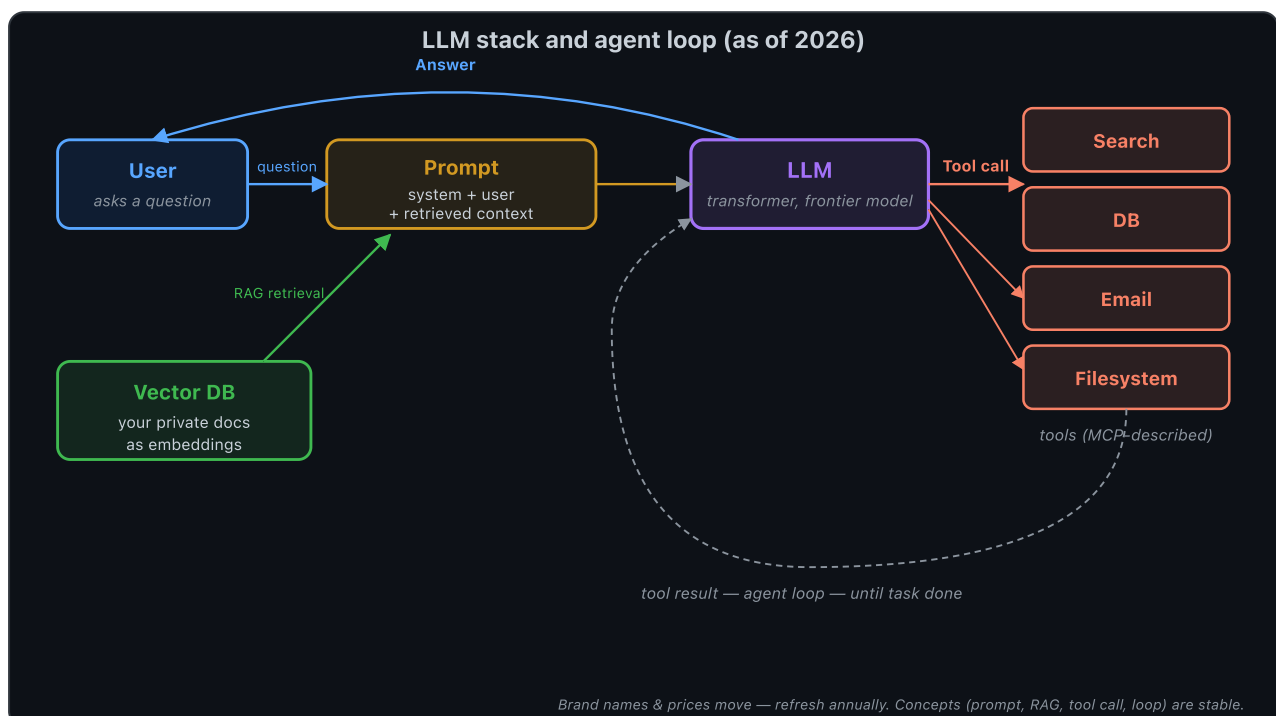
chatbots, most documentation chatbots, and a meaningful slice of customer-support bots in 2026 are RAG under the hood. RAG also keeps the answer grounded — if the retrieved chunks contradict the model's prior, the prompt usually wins.

5. Tools, function calling, and reaching the outside world

A modern LLM can be given a list of **tools** — described as functions with names, parameters, and what each does (`search_web(query)`, `send_email(to, subject, body)`, `query_database(sql)`). The LLM doesn't run the tool itself; it *requests* a call ("call `search_web` with `query='latest GDPR ruling'`"), the host program executes the call, the result is fed back into the conversation, and the model continues. This is how LLMs reach beyond their training data and *take action* in the world. It is also the building block of the next section.

6. The LLM stack and agent loop, one diagram

Here is the same idea drawn as a flow: a **User** asks a question; relevant chunks come out of a **Vector DB** via *RAG retrieval* and are folded into the **Prompt**; the **LLM** either returns a final **Answer** or emits a **Tool call** to one of several labelled tools. Tool results flow back into the model and the loop repeats until the task is done. That outer loop is what makes a chatbot into an **agent**.



7. Agents — LLMs in a loop

High-volatility — refresh brand names annually. An **agent** is, simply, an LLM that calls tools, looks at the result, decides what to do next, and *loops* — until the task is finished or it has decided it can't finish. Not magic; a loop. As of 2026 the agent landscape is busy: **Cursor** and **Windsurf** are IDE-embedded agents that edit your codebase on instruction; **Claude Code** is a terminal-first agent that runs commands and reviews their output; **Devin** brands itself as an autonomous software engineer; **Operator** is OpenAI's browser-driving agent; **Comet** (Perplexity's browser) is in the same family. Enterprise platforms like **Salesforce AgentForce** and **Microsoft Copilot Studio** package the same pattern. Open-source toolkits — **LangChain**, **LlamaIndex**, **AutoGen**, **CrewAI**, **PydanticAI**, **smolagents**, the **OpenAI Agents SDK**, the **Anthropic Claude Agent SDK** — let you build your own. The pattern is consistent across all of them; only the wrappers differ.

8. MCP — the agent integration standard

Every agent needs to plug into the user's tools — Slack, Gmail, GitHub, the filesystem, the database, the internal wiki. Without a standard, every agent vendor had to write a custom integration for every tool. **MCP (Model Context Protocol)** — introduced by **Anthropic** in late 2024 and rapidly adopted — is the open standard for how an agent talks to its tools. As of 2026, **Claude Desktop**, **Claude Code**, **Cursor**, **Windsurf**, **Cline**, **VS Code** (via extension), and a growing list of other clients all speak MCP. The payoff: write your integration once as an **MCP server** and it plugs into any compatible agent. This platform has a dedicated **MCP Servers** course covering it in depth.

What people argue about

- **Will scaling keep working?** For five years, bigger models on more data have kept producing better capabilities. As of 2026, some labs believe the curve will keep going; others believe we'll hit a wall and need new techniques (more data efficiency, neuro-symbolic hybrids, new architectures). Nobody knows yet.
- **Open weights vs closed weights.** Open-weight advocates argue safety, sovereignty, and innovation; closed-weight advocates argue frontier capability

and responsible release. Both camps publish strong work, and the gap between the two has narrowed *as of 2026*.

- **Confabulation (a.k.a. hallucination).** LLMs sometimes confidently produce wrong answers. RAG, tool use, and post-training reduce but do not eliminate this. How much to trust LLM output in high-stakes domains (medical, legal, financial) is an active debate.
- **Prompt injection and security.** When an LLM reads untrusted text (a web page, an email, a PDF), that text can contain instructions that the model may follow. *As of 2026* this is still unsolved — mitigated, not fixed.
- **Agent autonomy.** How much should an agent do without asking? The "ask permission" vs "ask forgiveness" trade-off is the live UX argument of 2026 — especially once an agent can spend money, send email, or push code.

Reading on

- **Anthropic's "Building Effective Agents"** — anthropic.com/research/building-effective-agents. Short, practical, written by the lab that built Claude. The best single starting point for the agent pattern.
- **Simon Willison's blog** — simonwillison.net. The most-read LLM blog in the field *as of 2026*; updated weekly with new model releases, prompt-injection findings, and tool experiments.
- **The Claude & LLMs course on this platform** — claude/claude-course.html. The deep dive: 33 lessons covering API usage, prompt engineering, RAG, fine-tuning, evals, and deployment.
- **The MCP Servers course on this platform** — if the agent-integration story above interested you, the dedicated course walks through writing your own MCP server end-to-end.

LAST-REVIEWED: 2026-05-26