COMS4995W32
Applied Machine Learning

Dr. Spencer W. Luo

Columbia University | Fall 2025

LLM Agents: From Reasoning to Action

Agenda

- Motivation
- Reasoning, Retrieval and Action
- LangChain
- Gemini Agent
- Model Context Protocol

Motivation

From Chatbots to Agents

Early LLMs: static responders - generating text only

- Primarily pattern-matching and next-token prediction
- No persistent state, no awareness of tasks or goals
- Single-turn interactions with no real reasoning chain

Modern LLMs: dynamic problem-solvers - reasoning, planning, acting

- Perform multi-step reasoning
- Decompose goals, choose actions, call tools/APIs
- Maintain context, use memory, interact with environments
- Capable of completing end-to-end tasks, not just answering questions

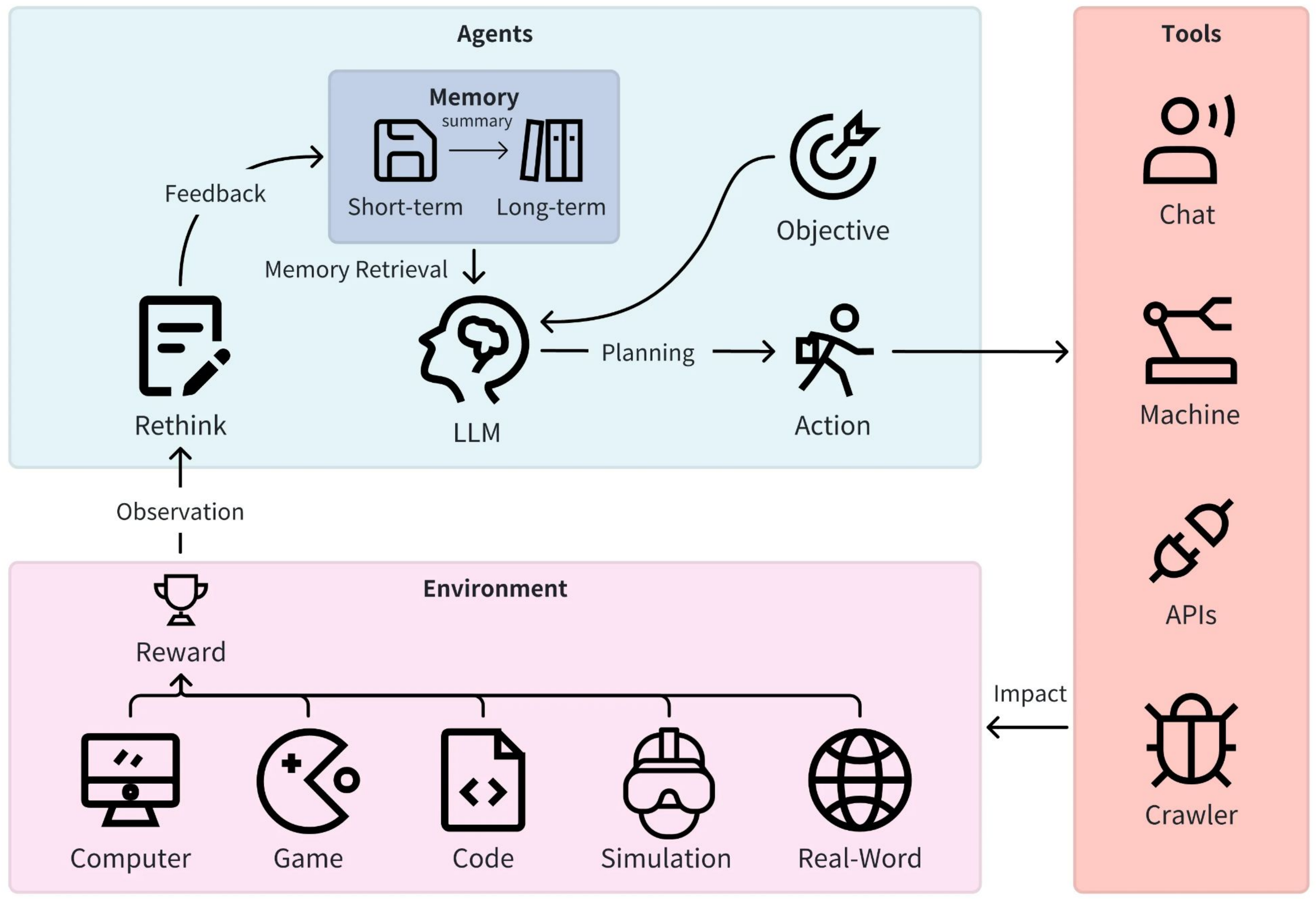
What Is an LLM Agent?

A system that reasons, plans, and takes actions using an LLM as the core 🧠

Integrates:

- Perception → Text / multimodal input understanding, grounding signals
- Reasoning → Goal decomposition, planning, hypothesis generation
- Action → API calls, web browsing, code execution, tool use

An LLM agent = Perception + Reasoning + Memory + Action + Control + ……


Agents
Memory
summary
Short-term
Long-term
Feedback
Memory Retrieval
Objective
Rethink
LLM
Planning
Action
Observation
Tools
Chat
Machine
APIs
Crawler
Impact
Environment
Reward
Computer
Game
Code
Simulation
Real-Word

Why 2025?

Explosion of LLM capabilities

- New frontier models: ChatGPT-5.1, Claude-4.5, Gemini-3.0
- Dramatically improved reasoning, long-context handling, multimodality, and tool-use reliability
- LLMs can now plan, iterate, and execute multi-step tasks

Mature API ecosystems enabling deployable agents

- Standardized tool calling (OpenAI functions, Gemini Tools, Claude MCP)
- Easier integration with search, databases, browsers, and code runtimes

Why 2025?

Enterprise adoption accelerating

- Agents as copilots, customer support bots, research assistants
- Automated workflows in finance, operations, data engineering, software development
- Organizations exploring “LLM workers” to augment human teams

Active research frontier

- How to make agents safe, grounded, verifiable, and reliable
- Challenges: hallucination recovery, planning accuracy, tool misuse, multi-agent coordination
- Growing interest in self-improving, and memory-augmented agents in top conferences

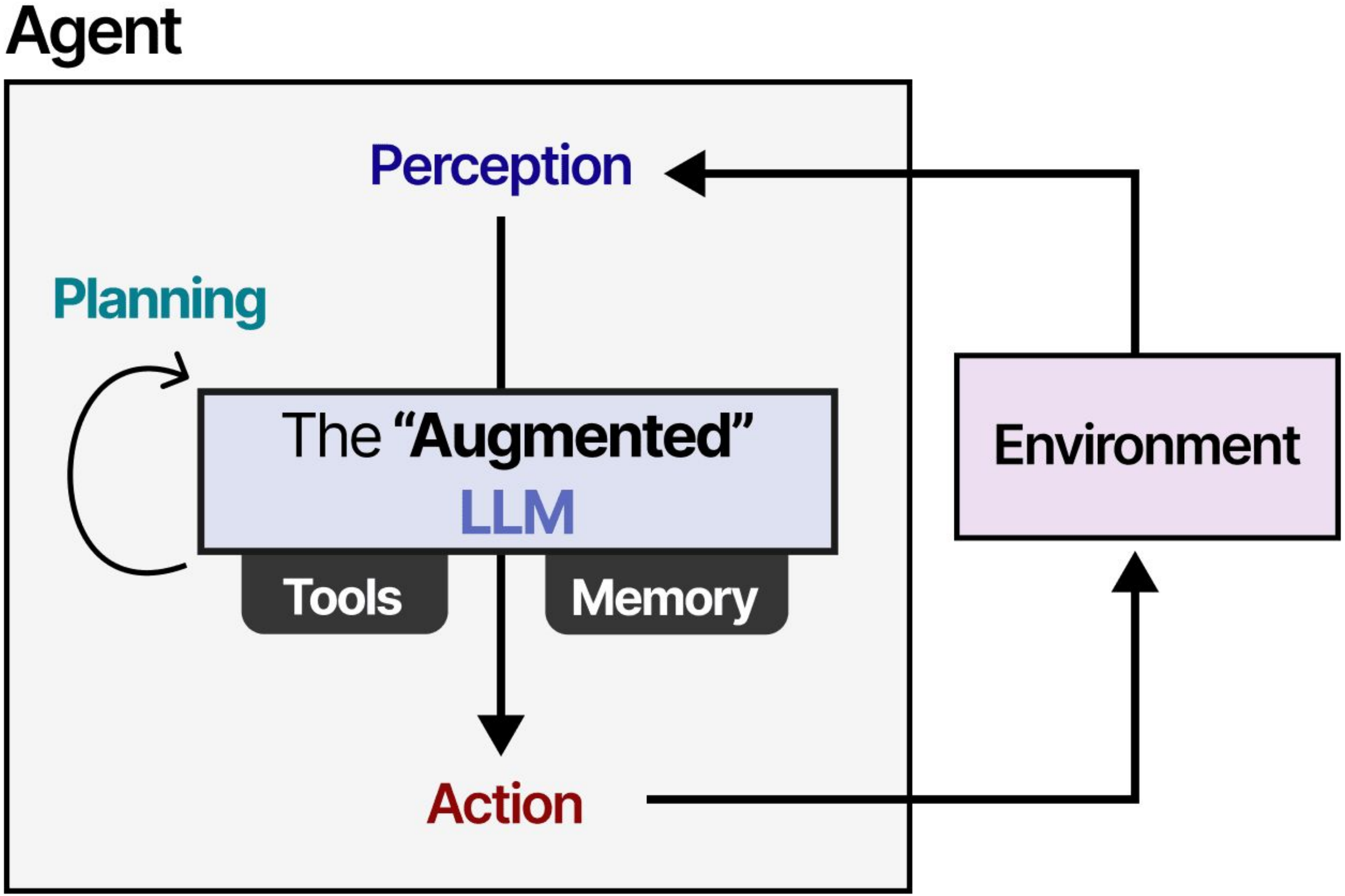
Reasoning, Retrieval and Action

What Makes an LLM an "Agent"?

An agent = an LLM that can reason, retrieve, and act

Core capabilities:

- Reasoning - break down tasks, form plans
- Retrieval - access up-to-date knowledge (internal and external)
- Action - call tools, APIs, code, search, browsers


Agent
Perception
Planning
The "Augmented"
LLM
Tools
Memory
Action
Environment

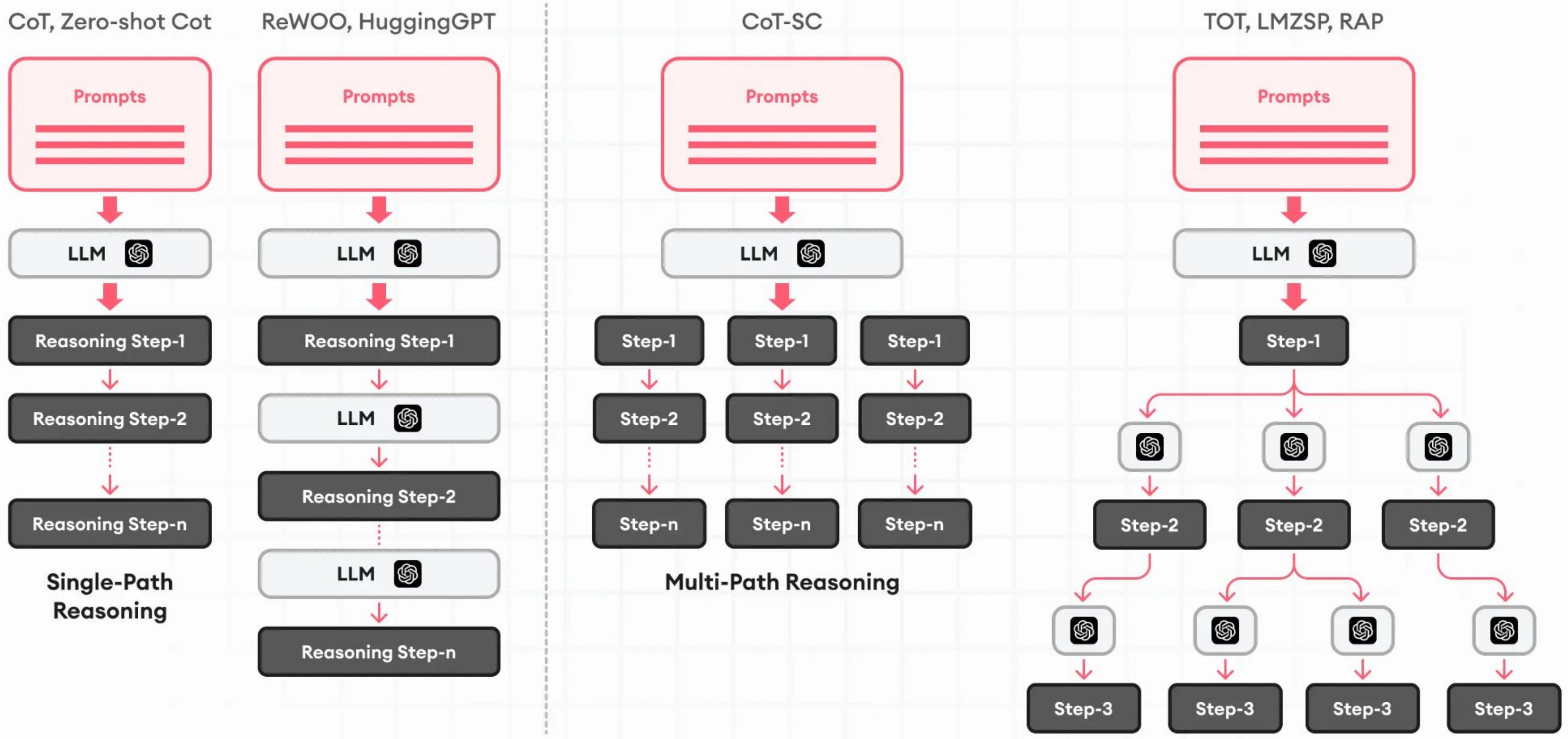
LLM as a Reasoner

Modern LLMs can:

- Produce intermediate thoughts (Chain-of-Thought)
- Explore options (Self-consistency, ToT)
- Self-correct (Reflection)

Key idea:

LLMs can think, and think deeper 🤔


CoT, Zero-shot Cot
ReWOO, HuggingGPT
CoT-SC
TOT, LMZSP, RAP
Prompts
LLM
Reasoning Step-1
Reasoning Step-2
Reasoning Step-n
Step-1
Step-2
Step-3
Step-n
Single-Path Reasoning
Multi-Path Reasoning

LLM as a Retriever

Model knowledge is static + incomplete

Tasks require fresh facts, private data, documents

Agents often need to look things up:

- search engines
- lecture notes
- company docs
- vector stores

RAG Pipeline Overview

Retrieval-Augmented Generation (RAG) == LLM + Retrieval system

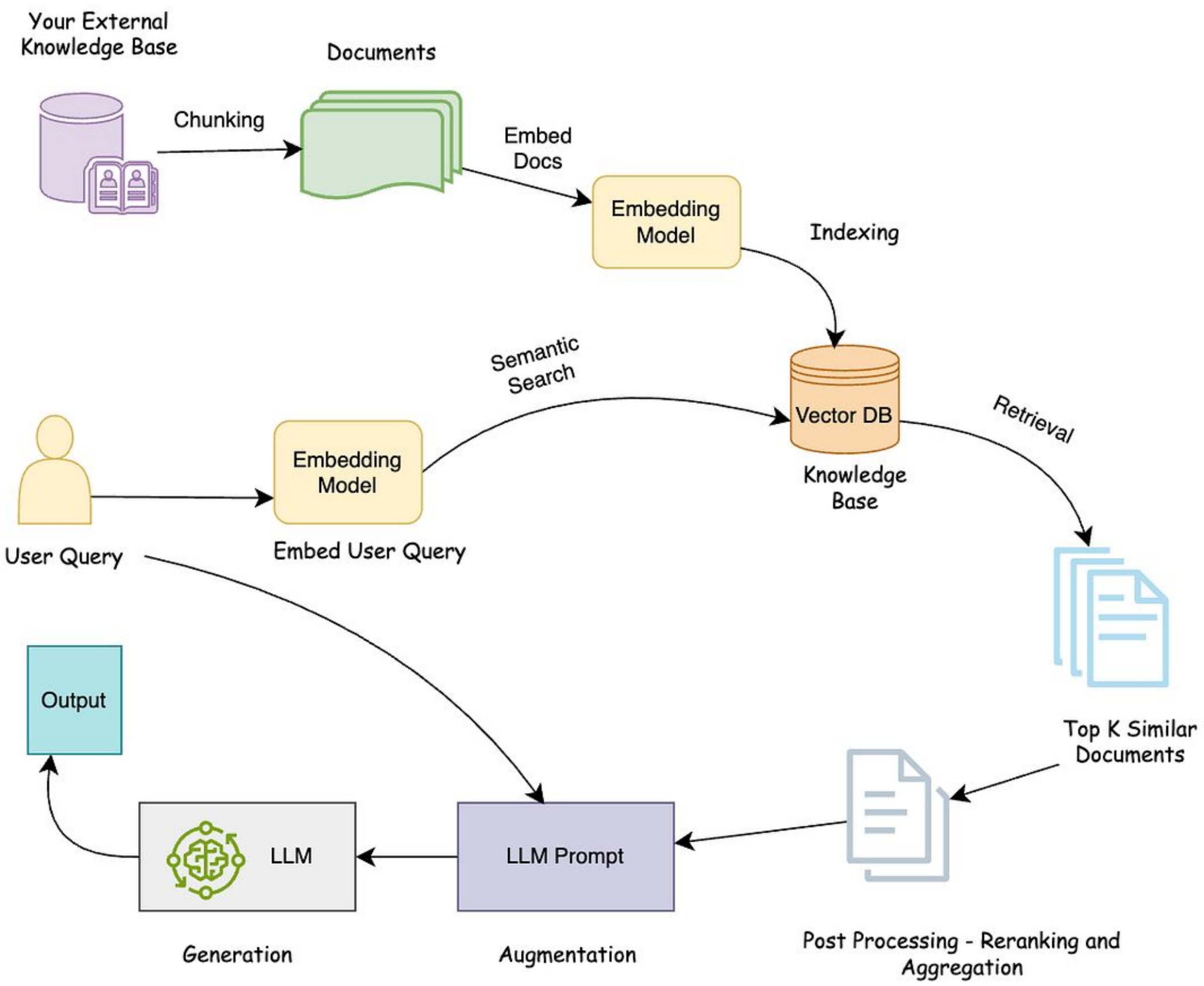
Query → Embed → Retrieve Top-K → LLM

Query - The user provides a question

Embed - Convert the query into a vector representation using an embedding model

Retrieve Top-K - Search a vector store to find the most relevant documents based on semantic similarity

LLM - The retrieved context is injected into the prompt, enabling the model to produce grounded responses


Your External Knowledge Base
Documents
Chunking
Embed Docs
Embedding Model
Indexing
Semantic Search
Vector DB
Knowledge Base
Retrieval
Embedding Model
User Query
Embed User Query
Top K Similar Documents
Output
LLM
LLM Prompt
Generation
Augmentation
Post Processing - Reranking and Aggregation

How Are Embeddings Trained?

Supervised: Train directly on similar vs. dissimilar pairs

Positive pair: (x_a, x□) - semantically similar

Negative pair: (x_a, x□) - semantically different

Pros: maximizes embedding usefulness in retrieval / ranking

Cons: requires curated data

Training Objective: Contrastive Learning

Most modern embedding models = Contrastive Learning

Given a batch of sentences:

- Pull positive pairs close
- Push negative pairs far away

Goal: Learns a geometry where “similar = close, different = far” semantically

Triplet Loss

$$\mathcal{L}_{\text{triplet}} = \max\left(0,\ d(f(x_a), f(x_p)) - d(f(x_a), f(x_n)) + \alpha\right)$$

Goal - Learn an embedding space where:

- Anchor and Positive are close
- Anchor and Negative are far

Input Triplet

- x_a : anchor
- x_p : positive (similar meaning)
- x_n : negative (different meaning)

Triplet Loss

Margin Constraint

```
distance(x_a, x_p) + margin < distance(x_a, x_n)
```

Intuition

- Pull the anchor toward the positive
- Push the anchor away from the negative
- Create a structured embedding space with clear separation

Common Applications

- Semantic similarity
- Retrieval systems

Where Do Positive / Negative Pairs Come From?

Parallel translations

English: “Good morning.”

French: “Bonjour.”

Synthetic paraphrases from LLM

“Explain what a transformer is.”

“Can you describe how transformer models work?”

Instruction → response

Prompt: “Write a summary of this paragraph”

Positive: Model-generated summary

Where Do Positive / Negative Pairs Come From?

Random samples

Query: “How to bake a cake?”

Paired with: “What is quantum entanglement?”

These two have no semantic relation → a very easy negative

Hard negatives from retrievers (most important)

Query: “symptoms of flu”

Hard negative: A document about COVID symptoms

The topic is similar, but it is not the correct answer → very useful for training because the model must learn fine distinctions

LLM as an Actor

LLMs take action through tools:

- API calls: `call_api("weather.today")`
- Function calling: `get_stock_price("AAPL")`
- Code execution: `eval("3 + 7")`
- SQL queries: `SELECT * FROM users WHERE id=42`

The Unified Agent Loop

All LLM agent systems comes to:

🧠 Think → 📚 Retrieve → 🛠️ Act → 👀 Observe → 🔁 Adjust

This loop underlies:

- LangChain Agents
- Gemini Agents
- MCP-based tools

Summary

Component	What it adds	Without it
Reasoning	Planning and multi-step thinking	Answers become shallow and brittle
Retrieval	Correct, grounded knowledge	Model hallucinates or fabricates facts
Action	Ability to execute tasks via tools	Agent cannot impact the real world

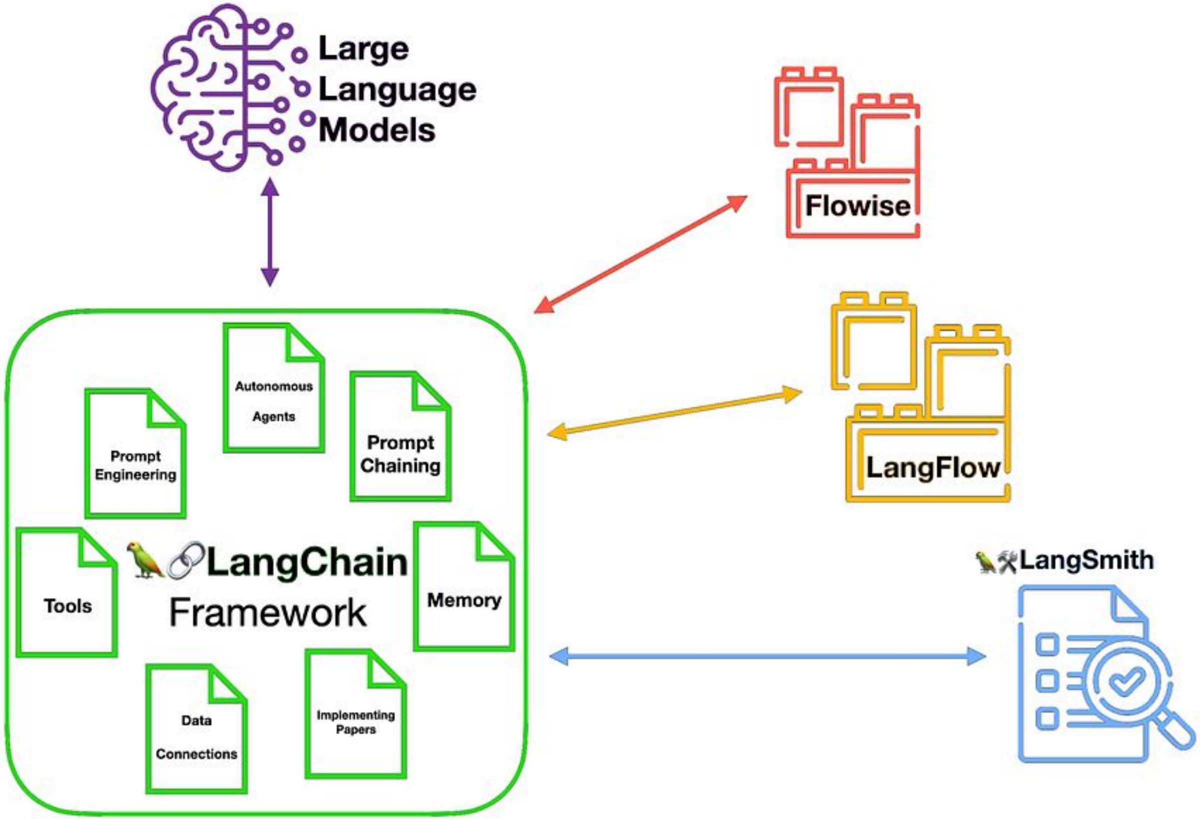
LangChain

LangChain Agents

LangChain is a framework that adds:

- Tool abstraction layer
- Agent controllers
- Memory modules
- Execution tracing
- Multi-agent architectures
- Integration with all LLMs (OpenAI, Claude, Gemini, Llama3)

🦜🔗 LangChain Ecosystem



LangChain Advantages

Mature ecosystem

Huge tool integrations (SQL, browser, python)

Debug-friendly (LangSmith)

Standardized agent loops

Excellent for classroom demos

Widely used in industry (NVIDIA, Tesla etc.)

LangChain Limitations

Too heavy for simple tasks

Not optimized for ultra-large agent workflows

Less multimodal than Gemini

Need separate orchestration for large deployments

Sometimes over-engineered for small projects 🥱

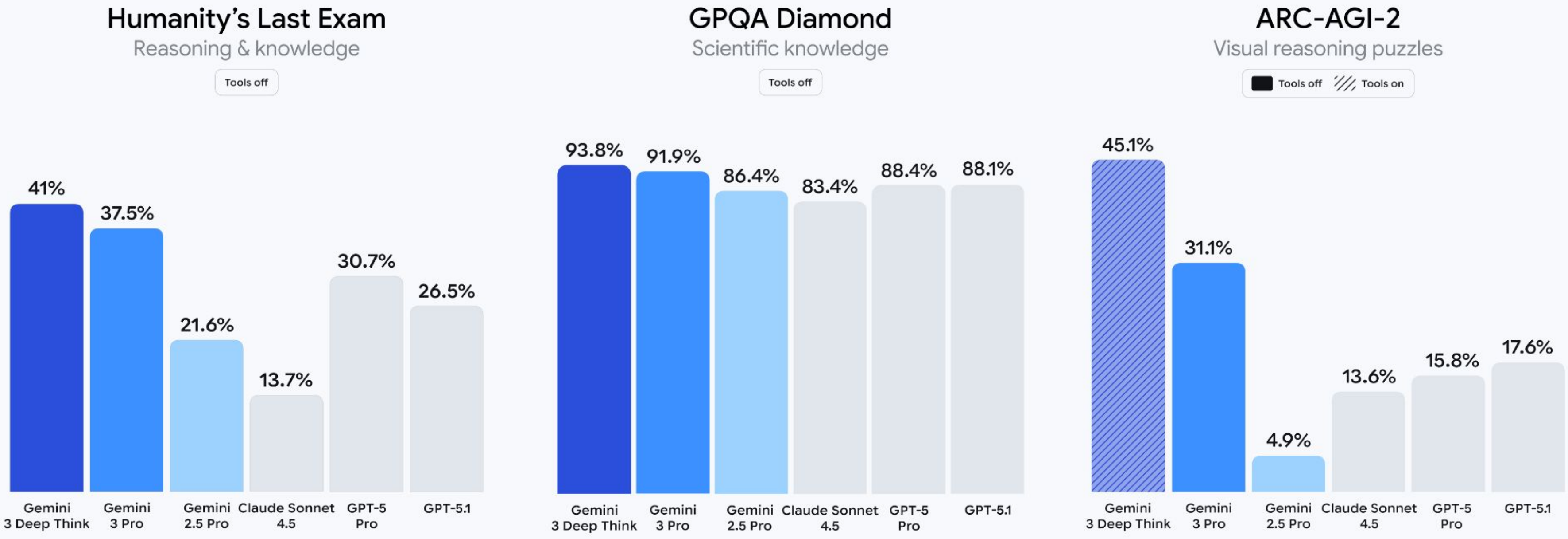
Gemini Agent

Gemini 3

Benchmark	Description		Gemini 3 Pro	Gemini 2.5 Pro	Claude Sonnet 4.5	GPT-5.1
Humanity's Last Exam	Academic reasoning	No tools With search and code execution	**37.5%** **45.8%**	21.6% —	13.7% —	26.5% —
ARC-AGI-2	Visual reasoning puzzles	ARC Prize Verified	**31.1%**	4.9%	13.6%	17.6%
GPQA Diamond	Scientific knowledge	No tools	**91.9%**	86.4%	83.4%	88.1%
AIME 2025	Mathematics	No tools With code execution	**95.0%** **100%**	88.0% —	87.0% **100%**	94.0% —
MathArena Apex	Challenging Math Contest problems		**23.4%**	0.5%	1.6%	1.0%
MMMU-Pro	Multimodal understanding and reasoning		**81.0%**	68.0%	68.0%	76.0%
ScreenSpot-Pro	Screen understanding		**72.7%**	11.4%	36.2%	3.5%
CharXiv Reasoning	Information synthesis from complex charts		**81.4%**	69.6%	68.5%	69.5%
OmniDocBench 1.5	OCR	Overall Edit Distance, lower is better	**0.115**	0.145	0.145	0.147
Video-MMMU	Knowledge acquisition from videos		**87.6%**	83.6%	77.8%	80.4%
LiveCodeBench Pro	Competitive coding problems from Codeforces, ICPC, and IOI	Elo Rating, higher is better	**2,439**	1,775	1,418	2,243
Terminal-Bench 2.0	Agentic terminal coding	Terminus-2 agent	**54.2%**	32.6%	42.8%	47.6%
SWE-Bench Verified	Agentic coding	Single attempt	76.2%	59.6%	**77.2%**	76.3%
τ2-bench	Agentic tool use		**85.4%**	54.9%	84.7%	80.2%
Vending-Bench 2	Long-horizon agentic tasks	Net worth (mean), higher is better	**$5,478.16**	$573.64	$3,838.74	$1,473.43
FACTS Benchmark Suite	Held out internal grounding, parametric, MM, and search retrieval benchmarks		**70.5%**	63.4%	50.4%	50.8%
SimpleQA Verified	Parametric knowledge		**72.1%**	54.5%	29.3%	34.9%
MMMLU	Multilingual Q&A		**91.8%**	89.5%	89.1%	91.0%
Global PIQA	Commonsense reasoning across 100 Languages and Cultures		**93.4%**	91.5%	90.1%	90.9%
MRCR v2 (8-needle)	Long context performance	128k (average) 1M (pointwise)	**77.0%** **26.3%**	58.0% 16.4%	47.1% not supported	61.6% not supported

For details on our evaluation methodology please see deepmind.google/models/evals-methodology/gemini-3-pro

Gemini 3 Deep Think



Methodology: deepmind.google/models/evals-methodology/gemini-3-pro

What Makes Gemini Agent Different

Google's Gemini Agent is designed as:

- Natively multimodal (video, audio, image, code, text)
- Action-centric (tools deeply integrated into model)
- Long-context capable (millions of tokens)
- Grounded with Google Search + tool use
- System-level API for building real agents across modalities

This is NOT "LLM + LangChain"

- Gemini is designed as an agentic model from the start.

Philosophy: "LLM is the Operating System"

Gemini Agents unify:

- Planning
- Tooling
- Multimodal understanding
- Memory
- Computer control
- Web search
- Code execution
- Live video analysis

Gemini Agent: Native Tools

Examples of tools that are seamlessly integrated to Google products:

- Google Search
- YouTube
- Drive / Docs / Sheets / Gmail
- Browser Actions
- Vision tools
- Code execution

Gemini Agent Code Example

```python
from google.generativeai import Agent

def add(a, b): return a + b

agent = Agent(model="gemini-3.0-pro", tools={"add": add})
agent.run("Use the add tool to compute 12 + 30.")
```

Gemini Agent Running Python Code

```python
from google.generativeai import Agent

agent = Agent(model="gemini-3.0-pro", enable_code_execution=True)
agent.run("Plot a sine curve with matplotlib.")
```

RAG + Gemini Agent

```python
from google.generativeai import Agent
import google.generativeai as genai

docs = ["ml_notes.txt", "transformer_lecture.pdf"]
agent = Agent(model="gemini-1.5-pro", documents=docs)

query = "Explain attention mechanism with examples."
response = agent.run(query)
```

Model Context Protocol

The Problem with Today's Tool Ecosystem

Tool calling today is fragmented

- LangChain tools
- OpenAI function calling
- Gemini tools
- Local tool adapters

All use different formats, different schemas, not interoperable

What Is an API?

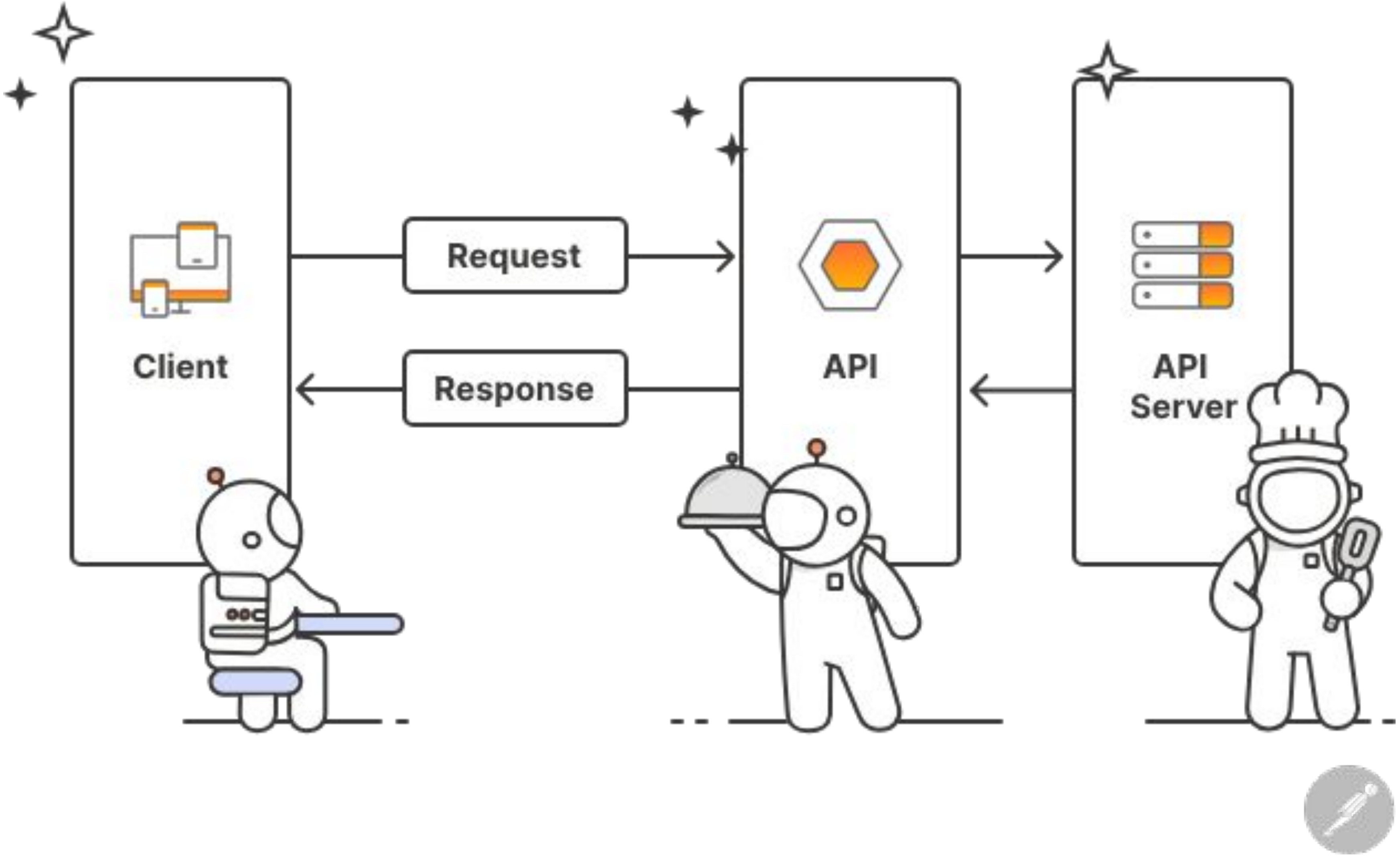
API = Application Programming Interface

A standardized way for two systems to communicate

- Let LLMs access external services (search, weather, finance, maps)
- Allow agents to perform actions (send emails, run code, write files)
- Enable integration with enterprise systems (databases, CRMs, tools)

Simple intuition:

An API is a bridge between any 2 systems


Client
Request
Response
API
API
Server

Introducing MCP (Model Context Protocol)

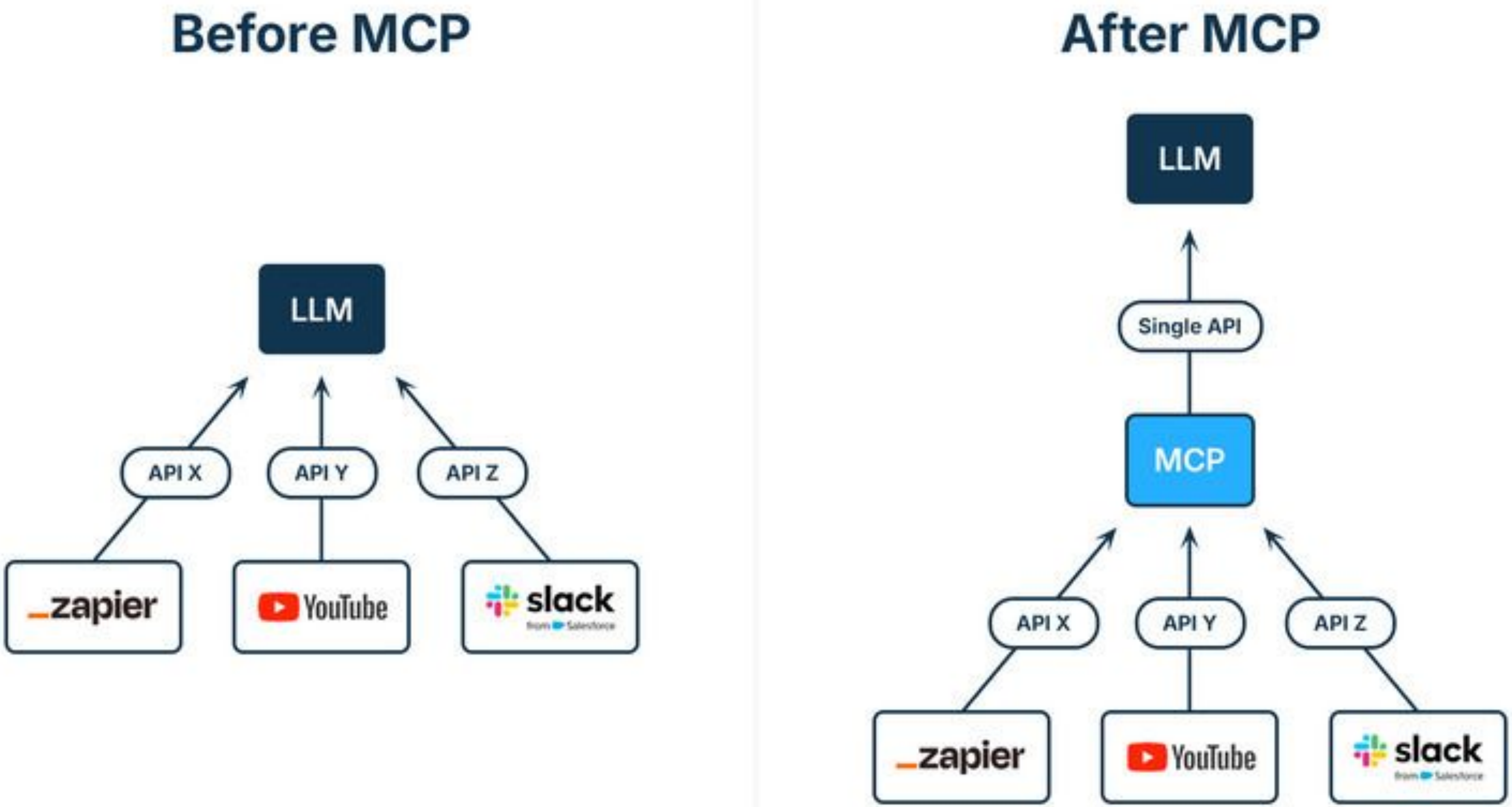
Open protocol created by Anthropic in Nov 2024

Standardizes how LLMs:

- Call tools
- Access data sources
- Manage sessions
- Handle resources

Designed to be model-agnostic, framework-agnostic, cross-platform

MCP = “HTTP/REST for AI Agents”


Before MCP
LLM
API X
API Y
API Z
zapier
YouTube
slack
After MCP
LLM
Single API
MCP
API X
API Y
API Z
zapier
YouTube
slack

Goals of MCP

A universal interface for agent ↔ tool interaction

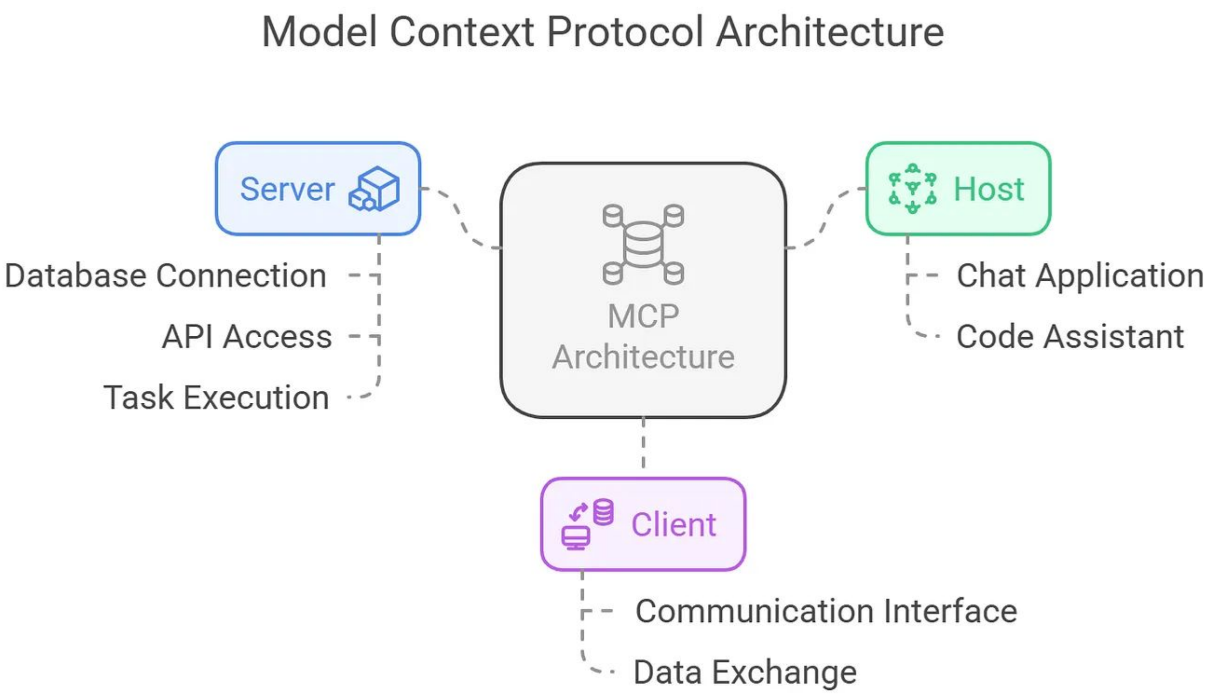
Reduce fragmentation in tool ecosystems

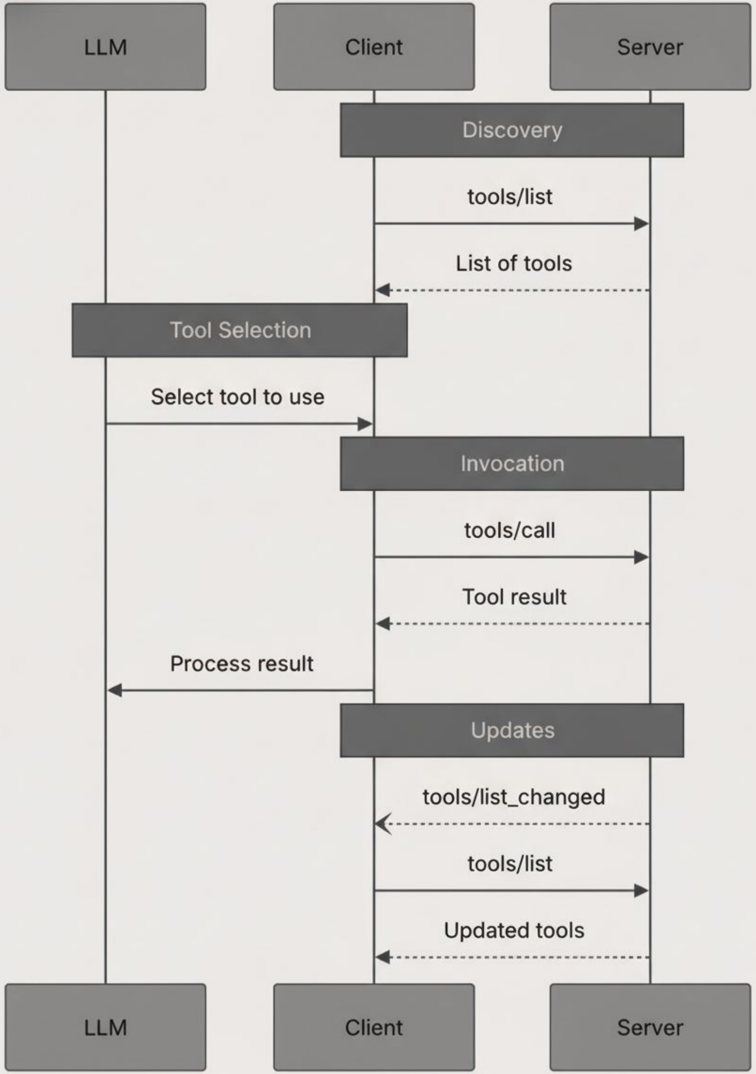
Build composable agent systems

Give developers full control over tools

Improve safety via well-defined protocol boundaries

Architecture Overview




LLM
Client
Server
Discovery
tools/list
List of tools
Tool Selection
Select tool to use
Invocation
tools/call
Tool result
Process result
Updates
tools/list_changed
tools/list
Updated tools
LLM
Client
Server

Cross-Model Interoperability

MCP tools can be used by:

- Claude
- GPT
- Gemini
- Llama
- Local models

Write Tool Once → Use Everywhere

Where MCP Fits

Layer	Examples	MCP's Role
Model	Claude, GPT, Gemini	MCP supplies standardized tools & resources
Agent Framework	ReAct, LangChain, AutoGen	MCP replaces the tool layer with a shared protocol
Tools	Python functions, Databases, APIs	MCP standardizes how tools are defined & exposed
Infrastructure	Docker, local machine, cloud	Tools can run anywhere behind the MCP interface