



Agentic AI with TrueFoundry



Anant Agarwal

(Principal Applied Scientist)





This is a [hands-on course on Agentic AI](#), where we start with pre-agentic GenAI, then dive into Agentic AI and MCP development & deployment with [TrueFoundry](#).

What You'll Learn

Core Agentic AI ->

Day 1: Primers on LLMs,
LangChain, RAG

Day 3: Agents
(cont.) and MCP

TrueFoundry Platform ->

Day 5:
Deployment
and
Evaluations

Get Certified ->

**Certification
Test**

Assessments 14%

14%

14%

14%

14%

30%

Technologies Used (feel free to experiment)



LangChain

LLM Orchestrator Framework

 **truefoundry**

Agentic AI Deployments

 **CURSOR**
Development IDE


tavily
Agentic Search

 **Chroma**
Vector Database



LangGraph

LangChain's Agentic AI Framework

 **gradio**

Prototyping UI for LLM Apps



Package Manager

 **groq**

API for Accessing Models



Day 1: Primers on LLMs, LangChain and RAG



Module 0: Setup



Getting Started

Step 1: Clone the Repository

- Clone the repository by performing `git clone https://github.com/truefoundry/agentic-ai-with-truefoundry.git`
- Execute command `cd agentic-ai-with-truefoundry` to move into the course directory

Step 2: Setup Virtual Environment

- Install `uv` package manager: <https://docs.astral.sh/uv/getting-started/installation/>
- Create and activate virtual environment with Python 3.12 version: <https://docs.astral.sh/uv/pip/environments/>
- Execute command `uv sync` command to install all the packages

Step 3: Initialize Environment File (.env) at root

- LLM API keys: OpenAI, Google, Anthropic, DeepSeek (*paid, optional*); Groq (*free, required*); TrueFoundry (*given*)
- Other API keys: Tavily, Linkup, HuggingFace

Step 4: Install Cursor

- Install Cursor IDE and open the project
- Select the correct environment

And you're set!

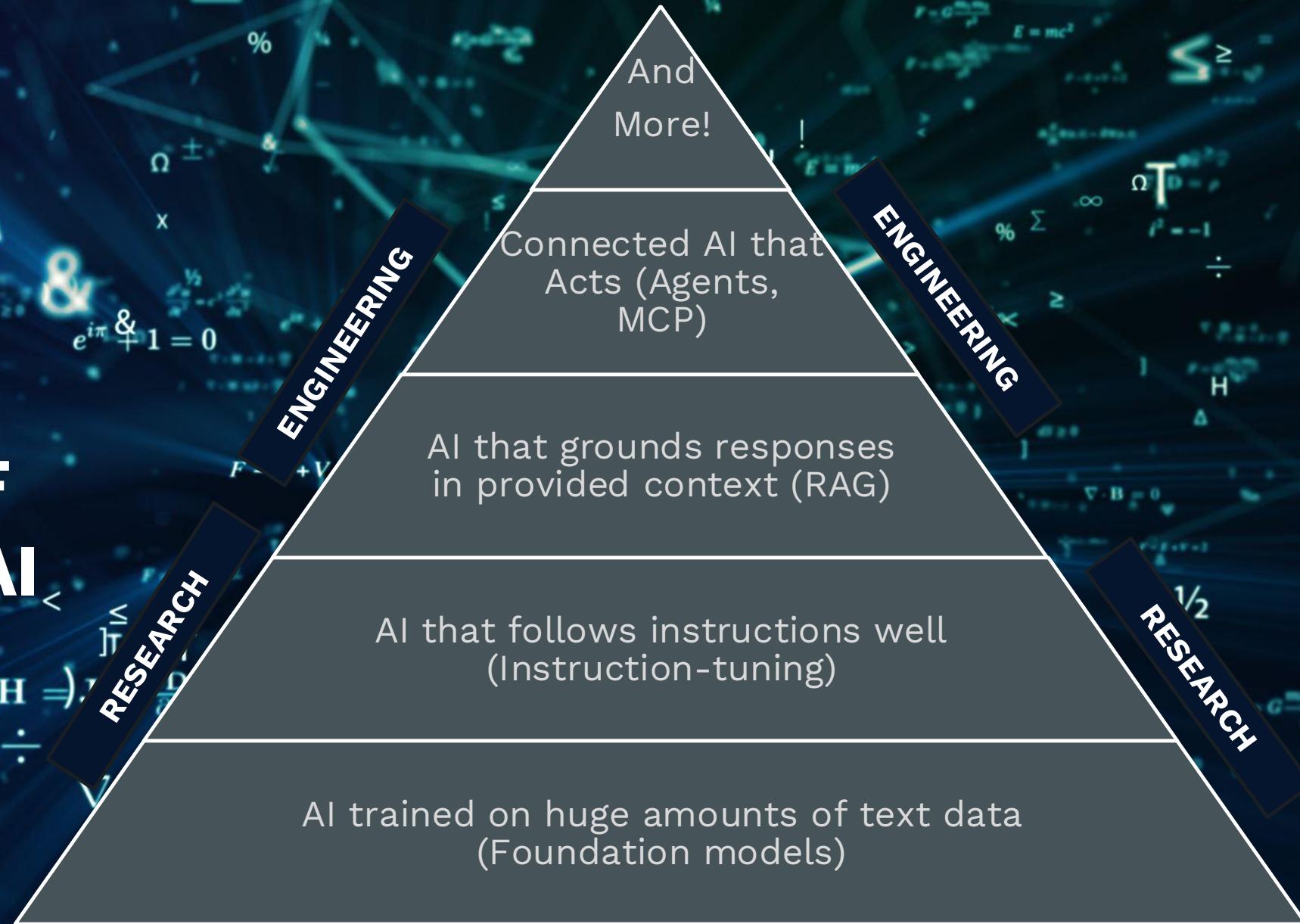


Hands-on Tutorial



Module 1: Basics of LLMs

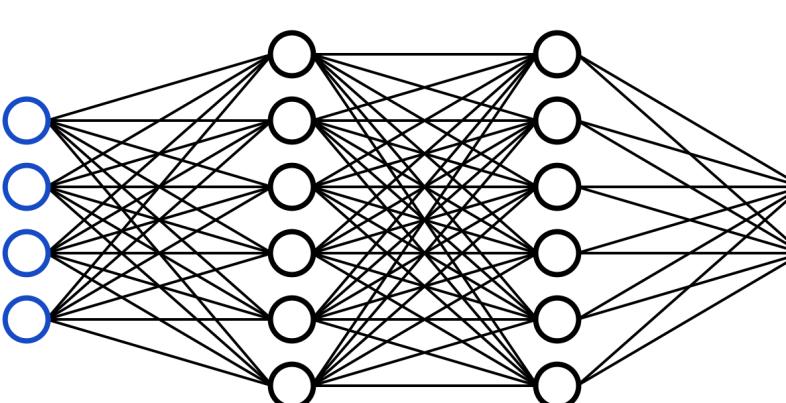
THE VALUE OF GENERATIVE AI



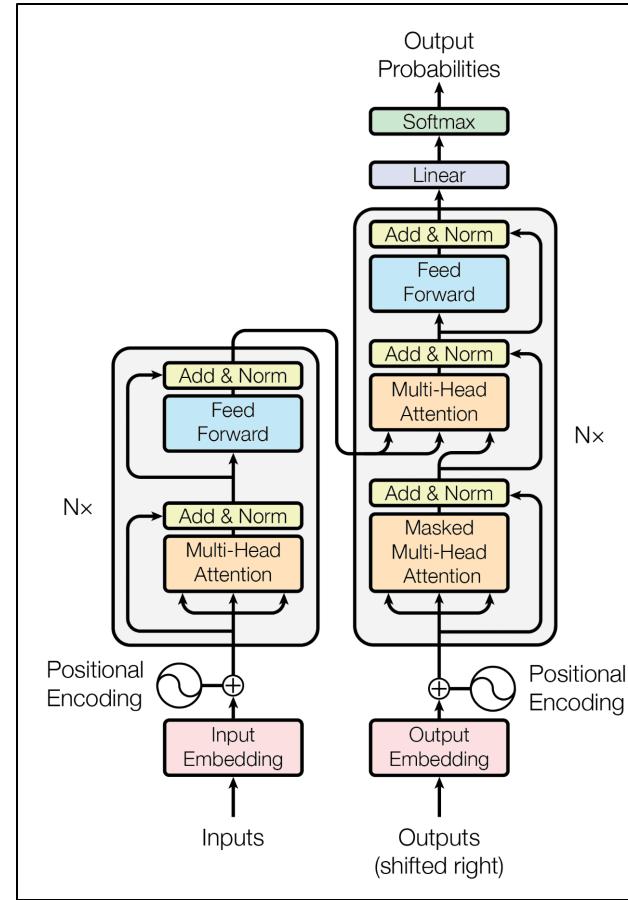
Fundamentals
became stronger



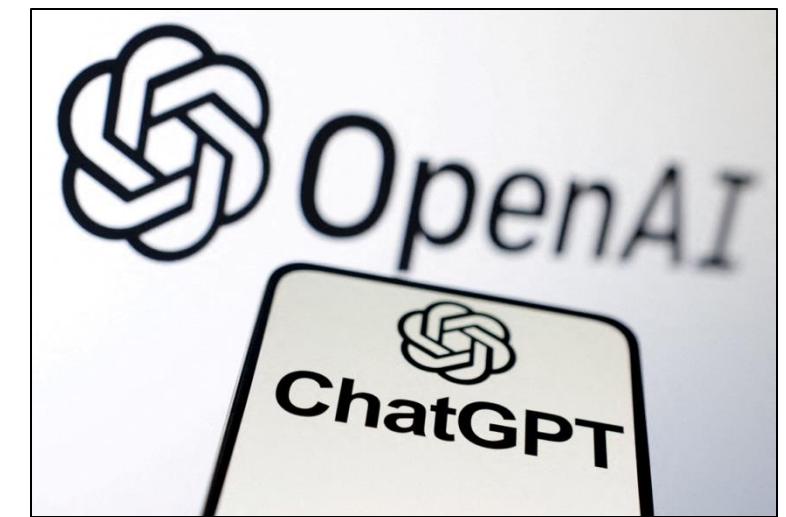
First came Neural Networks...



*Then Google released
Transformers!*



ChatGPT shook the world in Nov-22



Since then, the landscape of Generative AI is rapidly evolving!

Formulating a “Language Model”

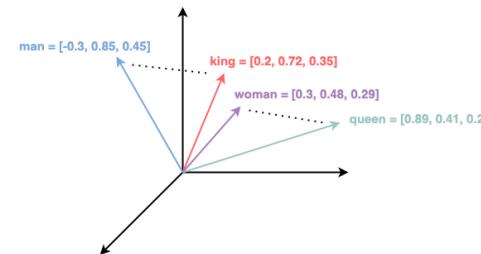
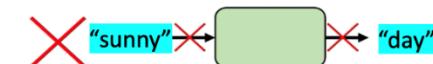
Language Modelling Problem

It was a bit hot outside, and I wanted to have
cold coffee.

given these words

predict the next words

- A Language Model is tasked with the objective of predicting the next few words, given a sequence of words
- However, language can not be directly represented as numbers.
- This is where “embedding” transforms into vectors of fixed sizes



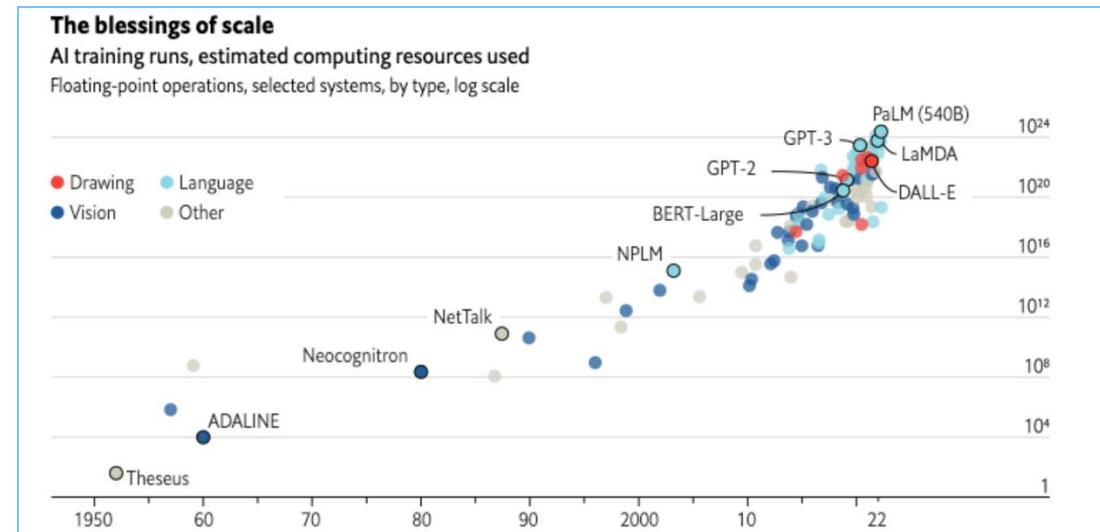
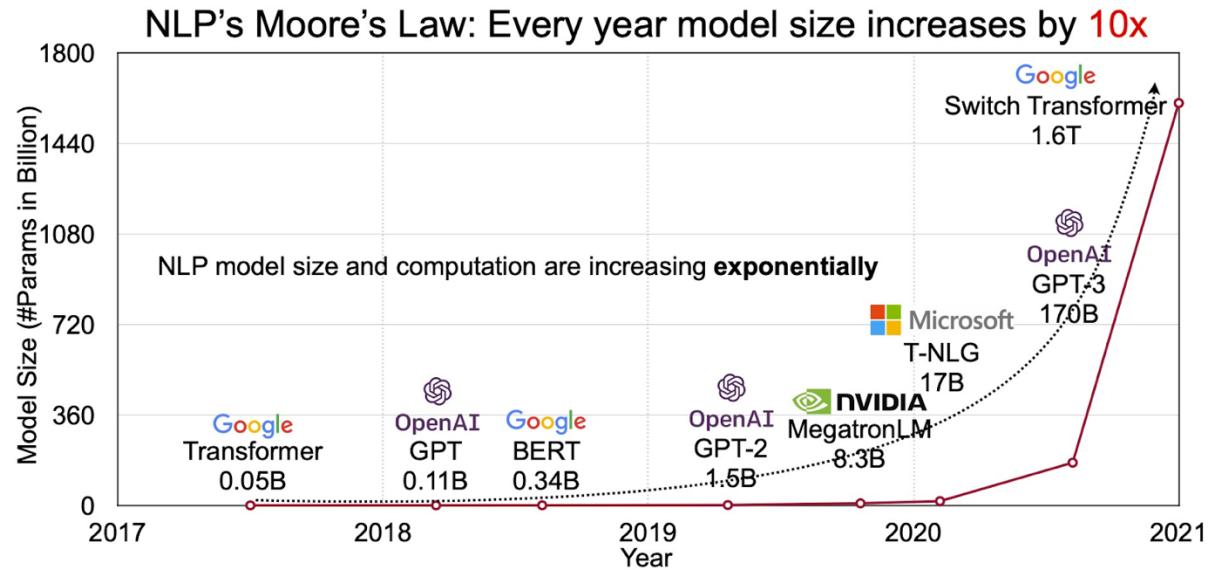
- Language models not only learn syntax and semantics, but the long-term dependencies as well

Long-term dependency
She went into the **cafe** to get some coffee. When she
walked out of **the** _____.
Semantics **Syntax**

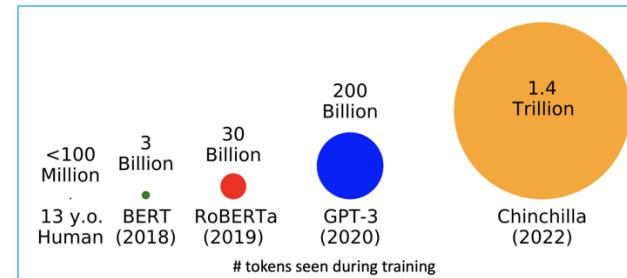


What Makes a Language Model “Large”

Number of Model Parameters



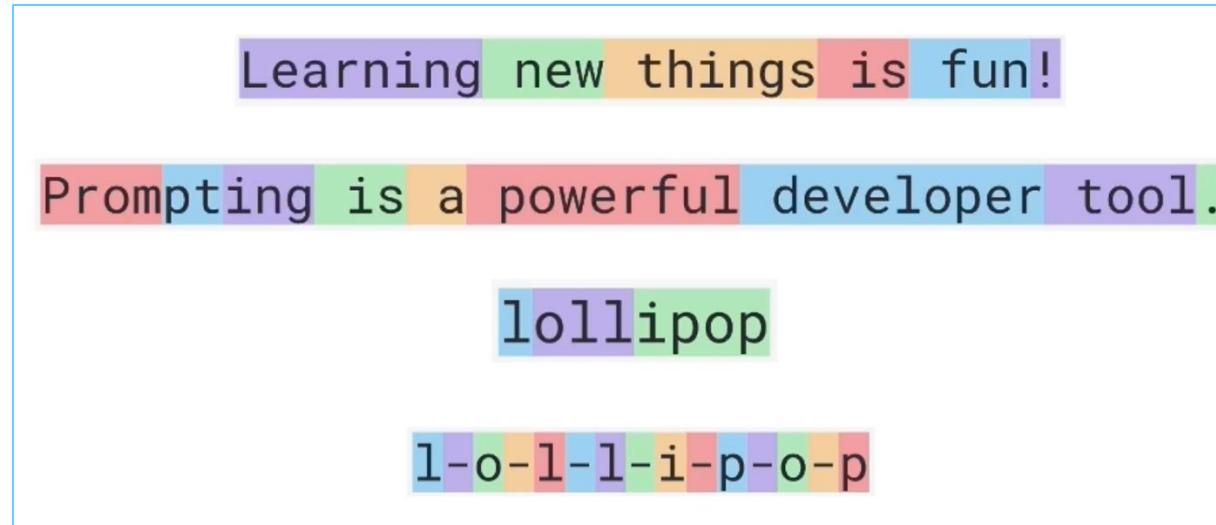
Trained on More and More Data!





Token in the Context of LLMs

- **Token** is the minimal amount of text used to generate text by the LLM models.
- This can be a character, piece of word, word or even sequence of words for certain languages.
- For English language input, 1 token is around 4 characters, or $\frac{3}{4}$ of a word.
- The **Byte Pair Encoding (BPE) algorithm** is one of the most widely used tokenization methods in LLMs
- Nice summary of other tokenizers is [here](#)





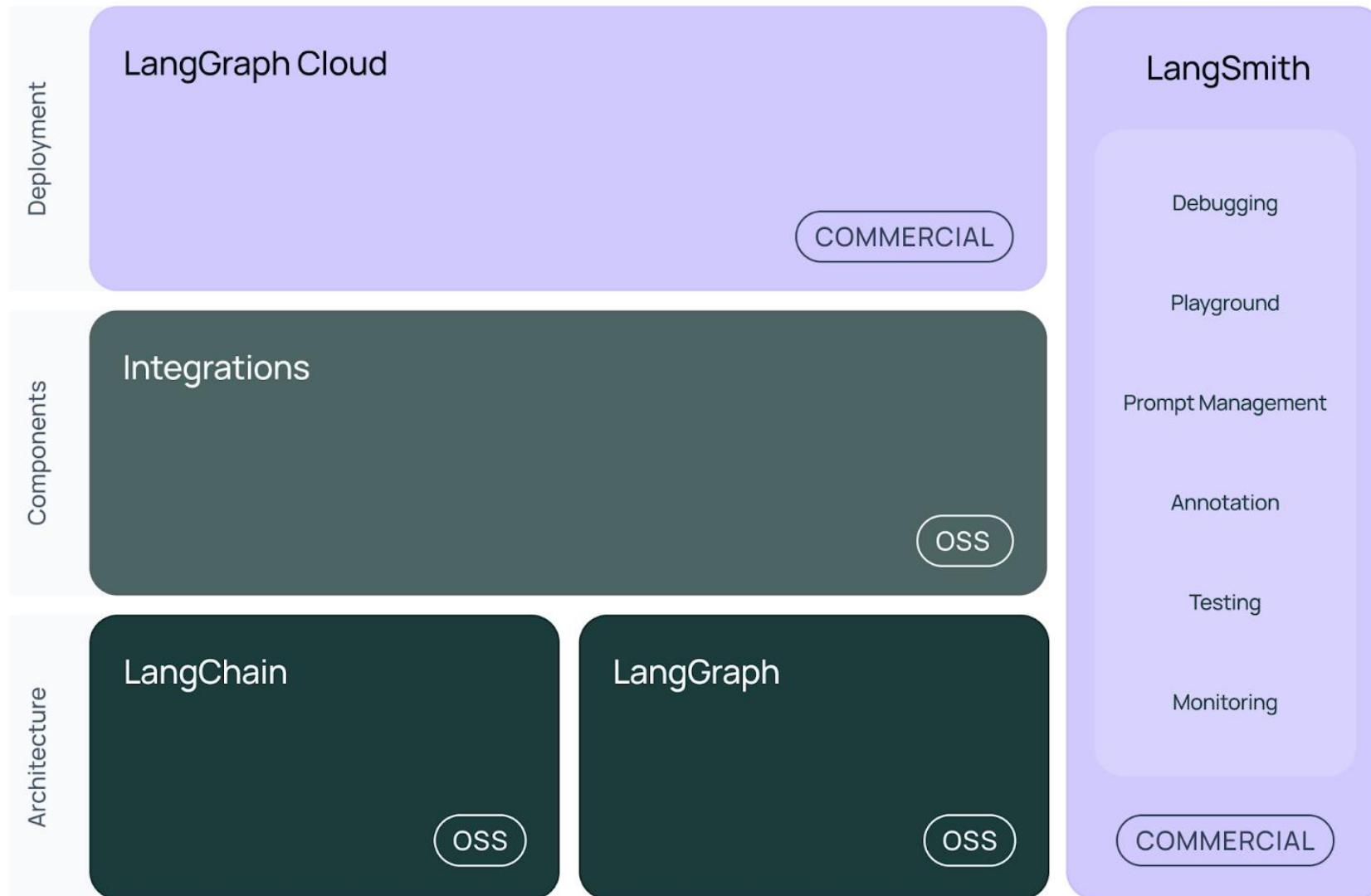
Hands-on Tutorial



Module 2: LangChain Primer



LangChain Ecosystem





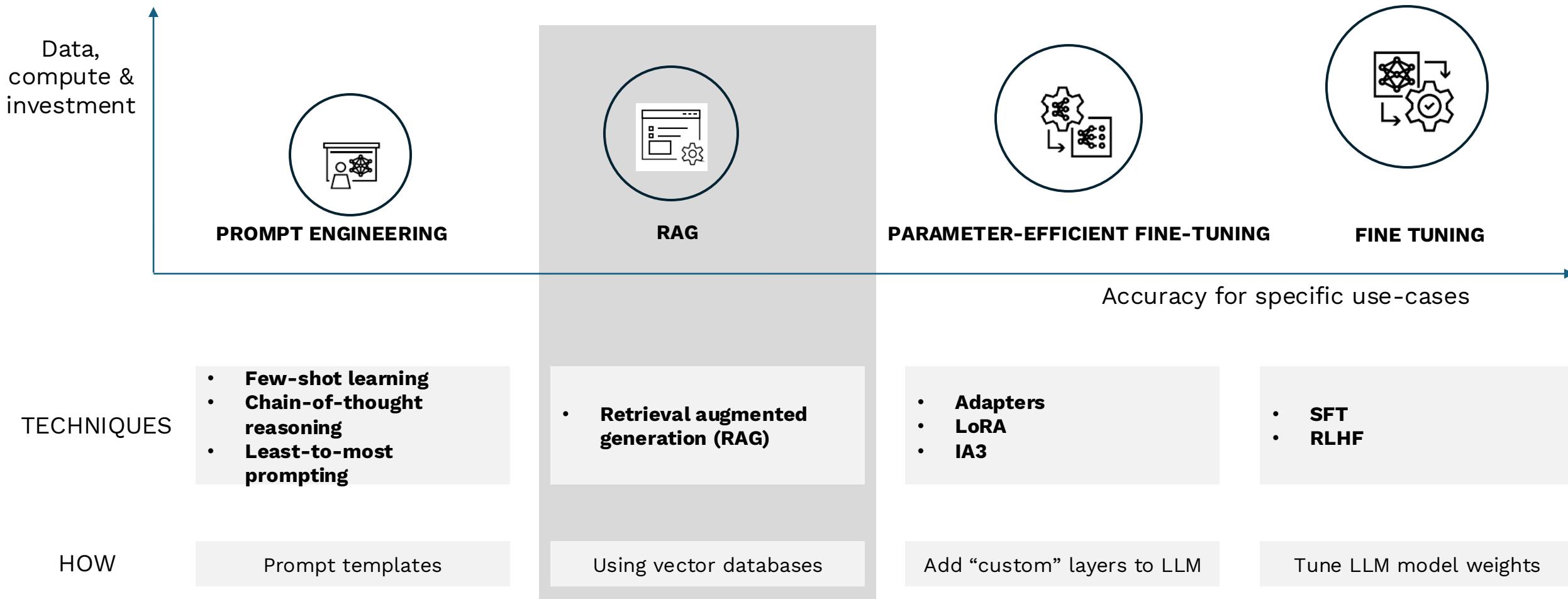
Hands-on Tutorial



Module 3: RAG Deep-Dive



Customizing* LLMs for Tasks - Overview



Adapted from Nvidia Research with a few changes

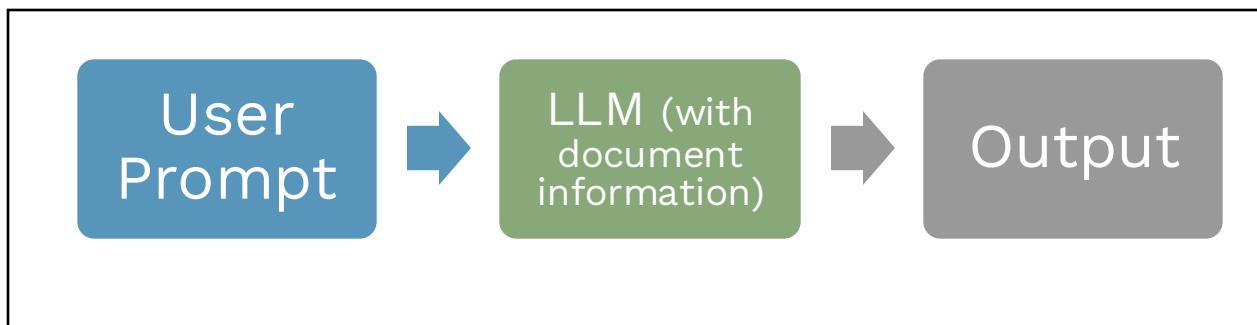
*Note that this is called “finetuning” loosely, and even Prompt Engineering and RAG can be considered as finetuning for your application - a common understanding of the word “finetuning”

What We Know

This is how you could typically chat with an LLM:



But how do you arm LLMs with the domain-specific / org-specific information you want it to refer to, rather than its own general training data knowledge? **This is the common use case that we'll focus on today.**



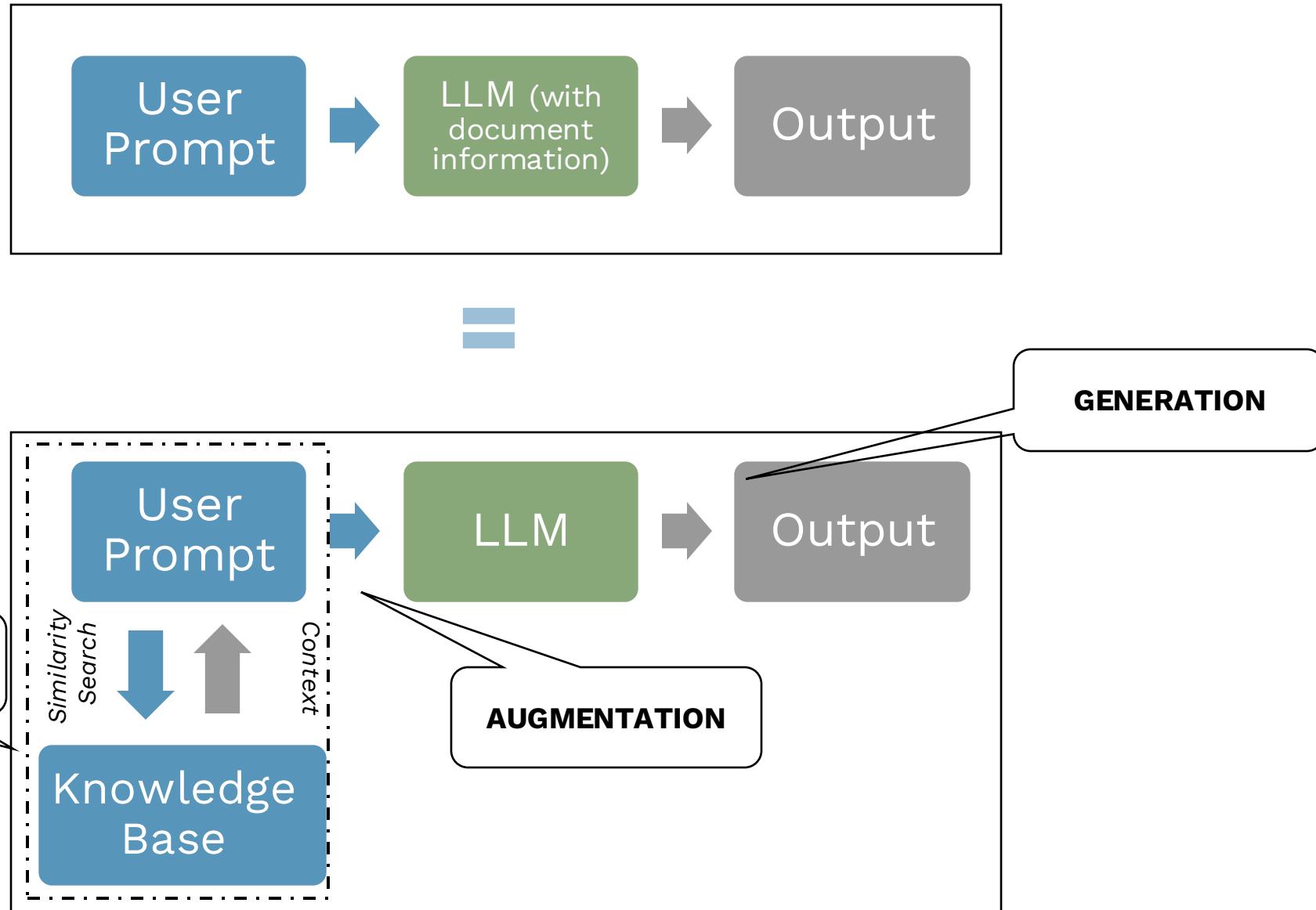
How do you think this would work?



Introduction to RAG

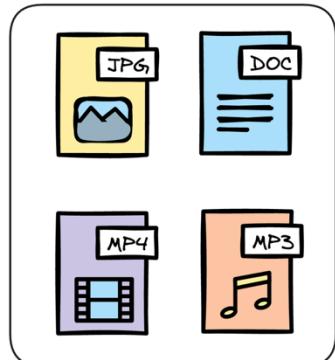
Retrieval Augmented Generation (aka RAG) is how it'll work.

Let's expand our flow.



Backbone of RAG: Vector Databases

Creating a
Vector DB



unstructured
data

Chunk and embed using an
embedding model

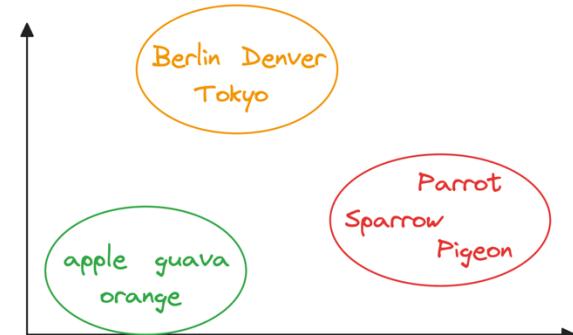
Embeddings

0.2	-1.7	•••	2.3
0.4	0.5	•••	-1.7
4.1	-1.9	•••	-1.5
-1.1	0.7	•••	5.3
-3.5	2.3	•••	0.5
-1.7	0.4	•••	0.2
2.3	0.2	•••	0.7
-1.9	4.1	•••	-2.4
0.5	-1.5	•••	2.3

A **vector database** stores unstructured data,
in the form of vector embeddings

"Venice"

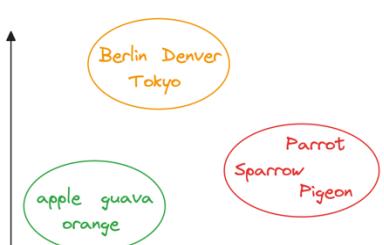
find nearest
neighbors



There are different methods by which similarity search is done

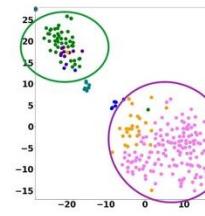
Why
embedding
works?

Embeddings of similar meaning words
are near to each other →

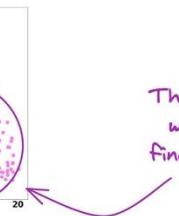


Contextualized embedding models are used,
instead of static embedding models

The word 'bank'
when used in
water sense



The word 'bank'
when used in
financial sense



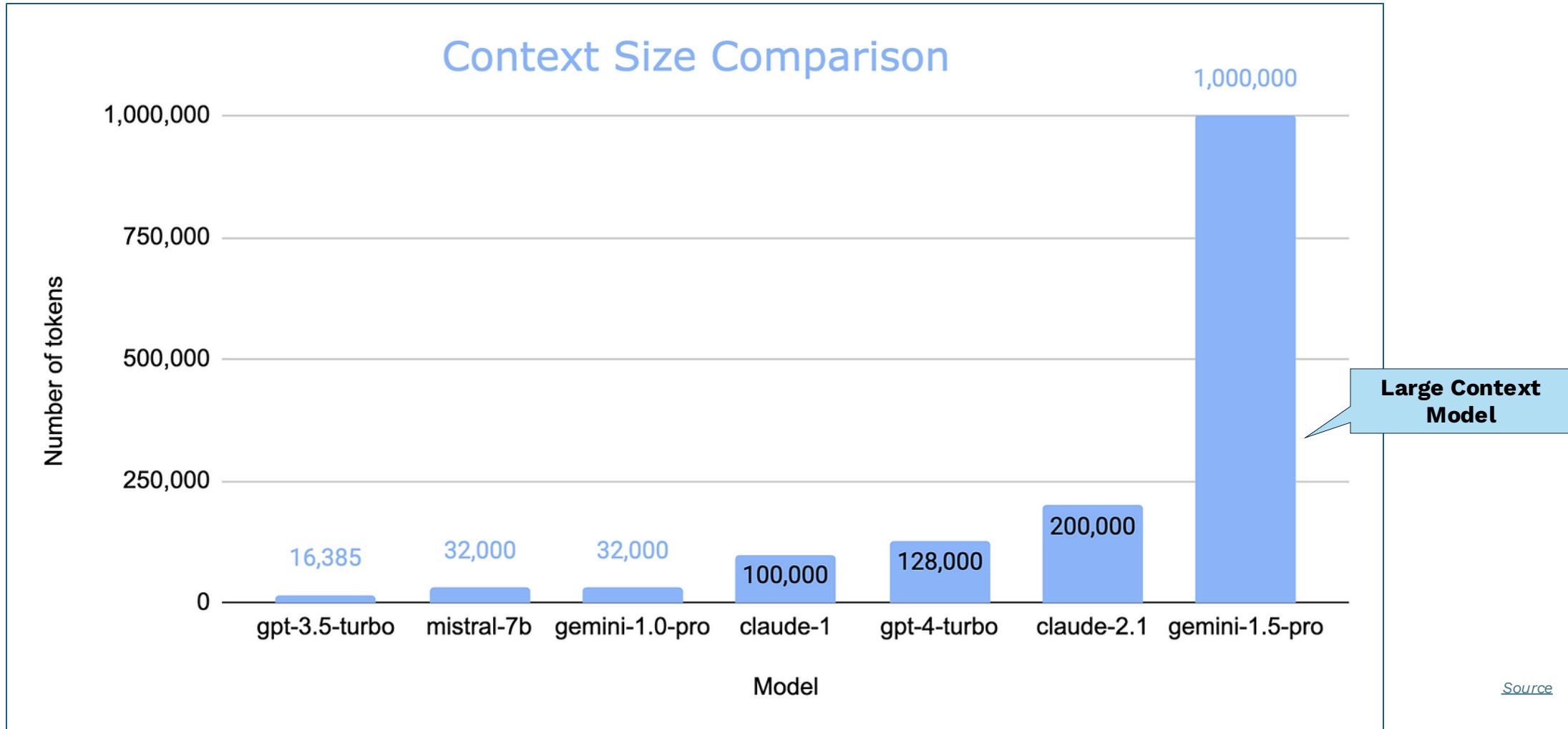
Important
Note



LCM vs. RAG



Increased Context Length – RAG still relevant?





Increased Context Length – RAG still relevant?

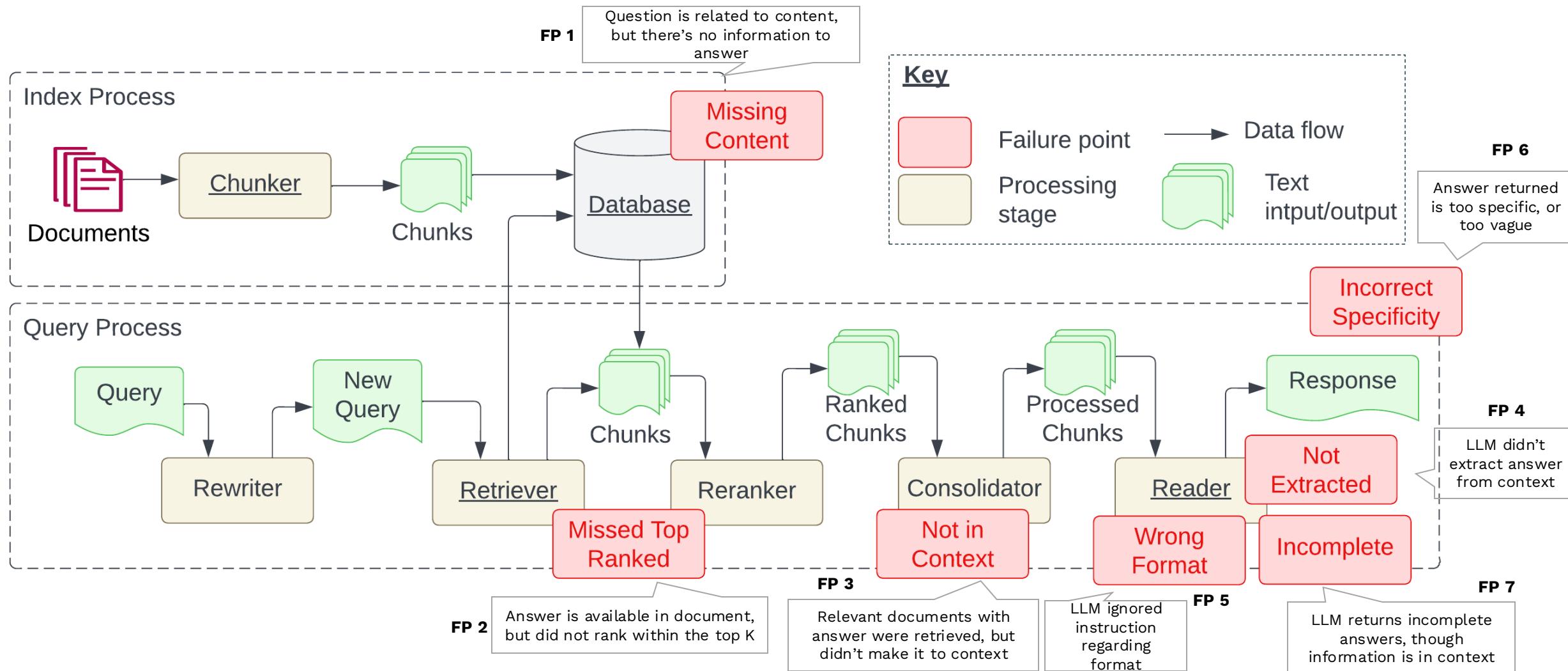
- **Cost:** Large Context Models (LRAG winsCMs) like Gemini 1.5 are expensive per token, making RAG models more cost-effective for most use cases! RAG wins!
- **Latency:** LCMs take longer to process longer contexts. RAG models can achieve lower latency with smaller contexts. RAG wins!
- **Context Learning:** LCMs excel at complex reasoning and learning from long contexts, demonstrated by reasoning about Apollo 11 transcripts and learning new skills. LCMs win!
- **Long Context Recall:** LCMs may struggle with recalling information in the middle of long contexts. RAG models can curate relevant content before feeding it to the LLM. RAG wins!
- **Flexibility & Customization:** RAG models offer more control over data sources and retrieval strategies, making them more customizable. LCMs are a faster solution but less flexible. RAG wins!
- **Security & Data Privacy:** RAG models provide more control over data privacy, especially for self-hosted deployments. LCMs rely on an external provider's security measures. RAG wins!

Conclusion: RAG is here to stay!



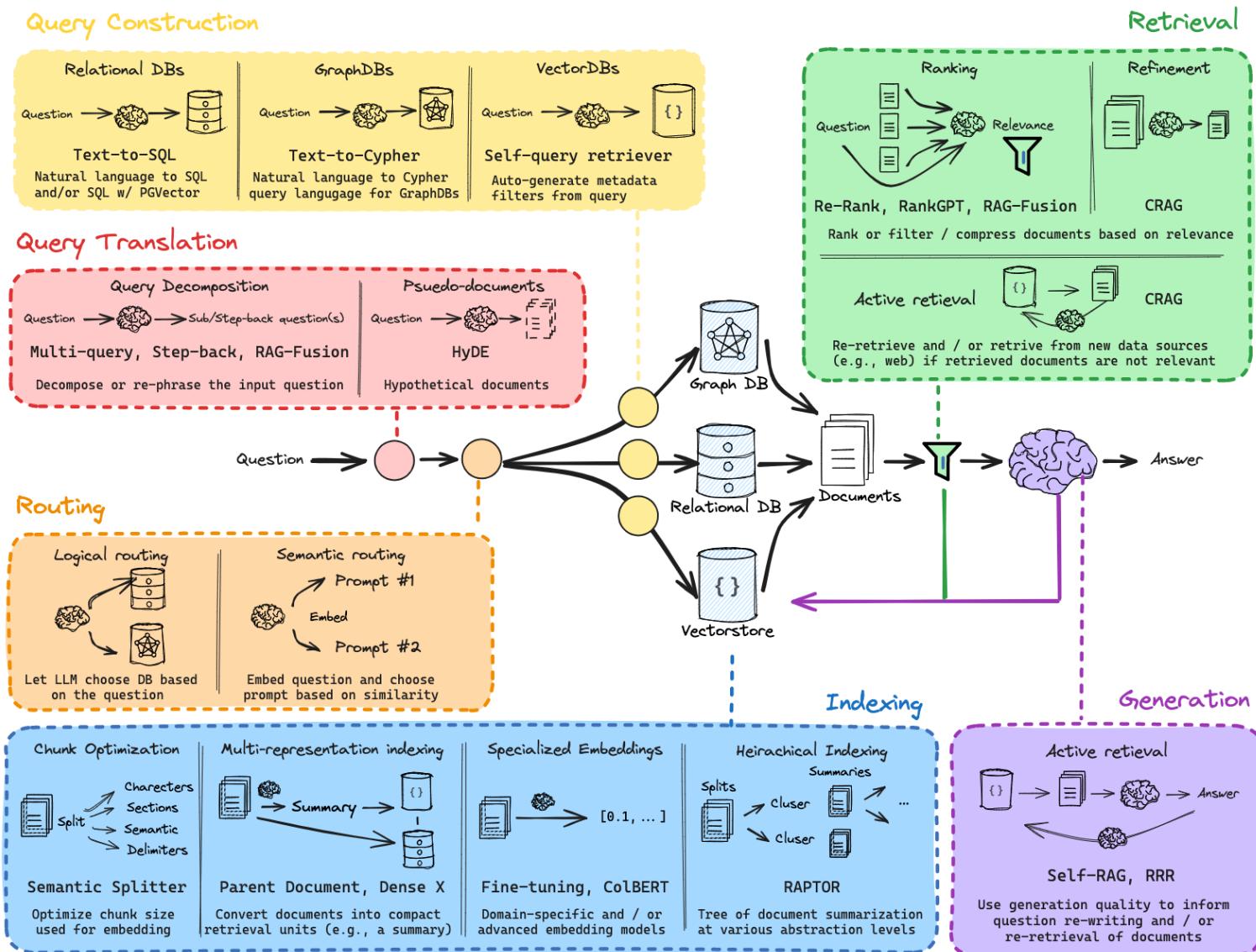
Improving RAG Systems

Key Failure Points of a RAG System





Methods to Improve RAG



Assuming prompts and choice of LLM are optimized for, we're optimizing for RAG!



Multimodal RAG

Multimodal RAG



Building Retrieval-Augmented Generation (RAG) pipelines for multimodal data presents unique challenges. Here are the main approaches for tackling them, focusing on image and text inputs:



Unified Vector Space:
Embed all modalities (like images and text) into a single, shared vector space.
(multimodal embedding model needed)

1



Primary Modality
Grounding: Convert or "ground" all modalities into one primary modality (e.g., transforming images into textual descriptions).
(text embedding model needed)

2



Separate Modality
Stores: Maintain distinct storage and retrieval systems for each different modality.
(each modality's embedding model needed)

3

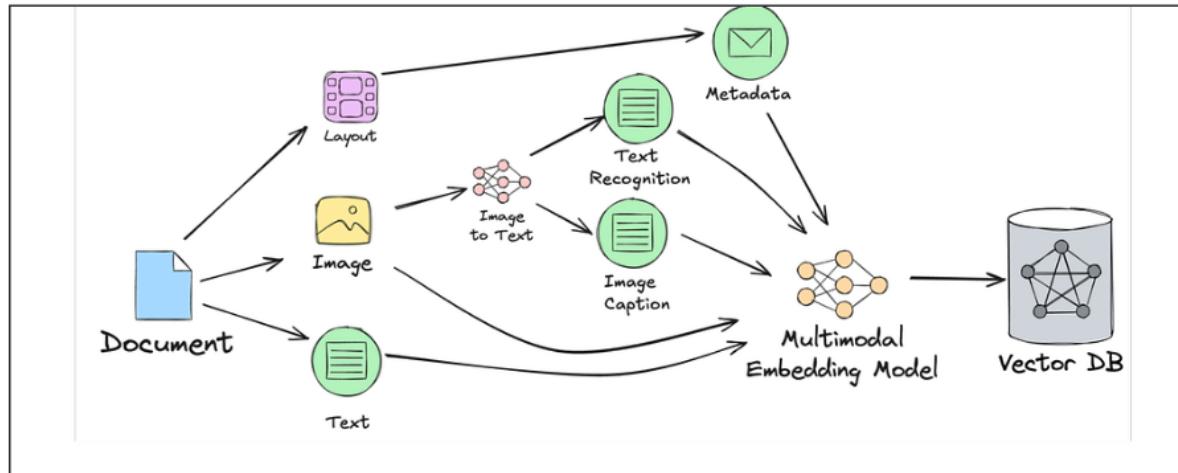


Image source here

Pros/Cons

1

Embedding into single vector space is good, enables cross-modal retrieval, but suffers in latency, cost and scalability

2

Embedding everything into a single modality like text is fast, with wider LLM compatibility, however, rich information of other modalities (like image) is lost and quality of modality translation (from image to text) depends heavily on model used

3

Embedding into multiple vector stores, like a hybrid approach is best of both worlds, but very complex



Hands-on Tutorial



Assessment (14%)

Day 2: Building Agents with LangGraph



Module 4: Building Agents with LangGraph



Why LLMs Alone Fall Short

- LLMs can think, but not act
- LLMs are limited by what's available in their training data
- There is a single inference basis the user query, and no continuous context management (i.e., chat history)



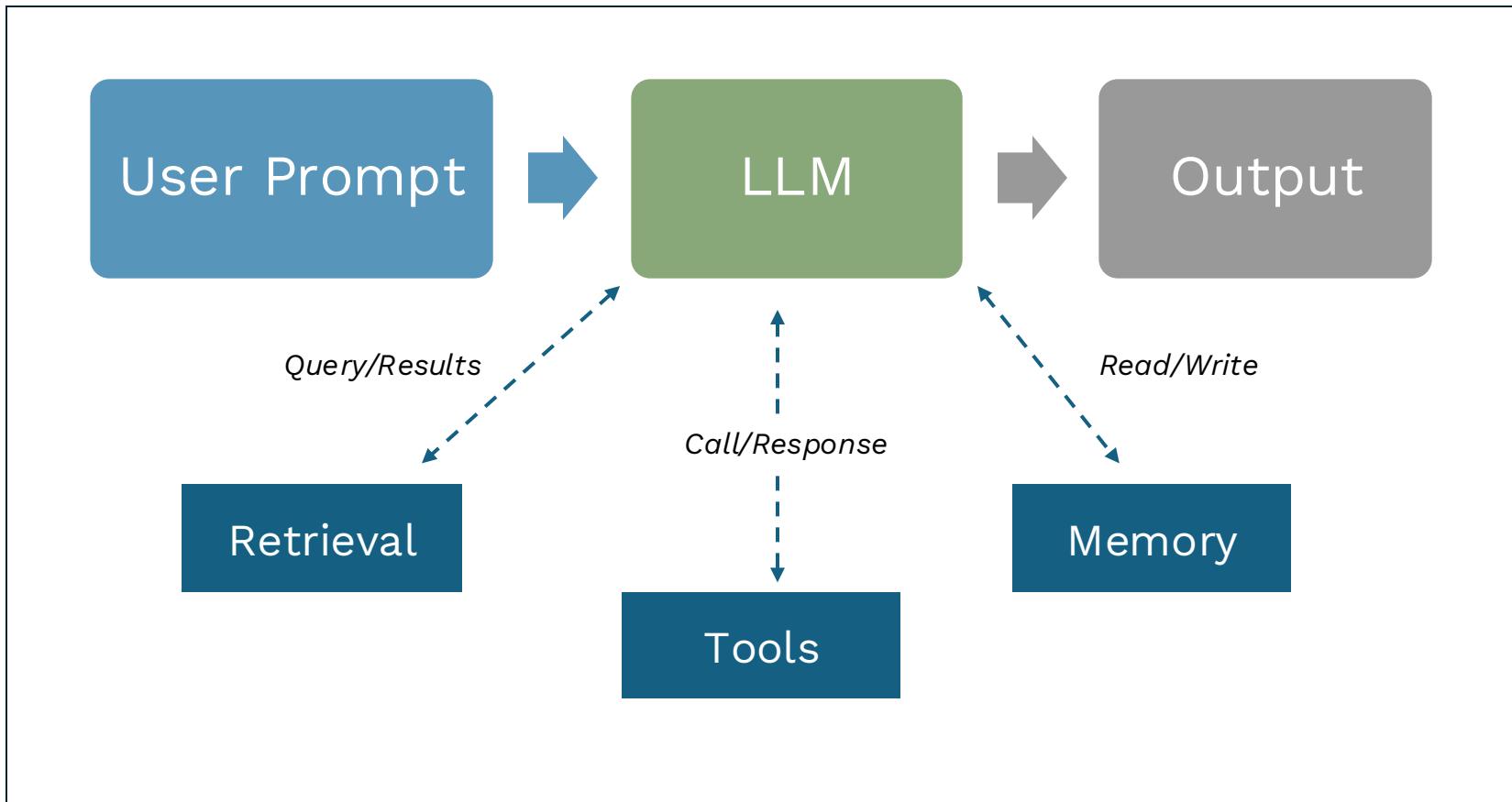
What We Need Instead

- LLMs can think, but not act
 - Need to be able to perform actions like making an API call
- LLMs are limited by what's available in their training data
 - Need to extend knowledge through connection with external systems such as search tools
- There is a single inference basis the user query, and no continuous context management (i.e., chat history)
 - Need to have a managed session history, which allows for multi-turn conversations



The Augmented LLM

The basic idea is to augment the LLM with retrieval, tools and memory, where they can generate their own search queries, select appropriate tools and determine what information to retain.





Agents vs. Workflows

- **Workflows** are systems where LLMs and tools are orchestrated through predefined code paths
- **Agents** are dynamic systems where LLMs direct their own processes and tool usage, maintaining control over how they accomplish tasks (i.e., LLM controls execution logic)
- Both are characterized as “**Agentic AI systems**”, however the architectural difference exists!

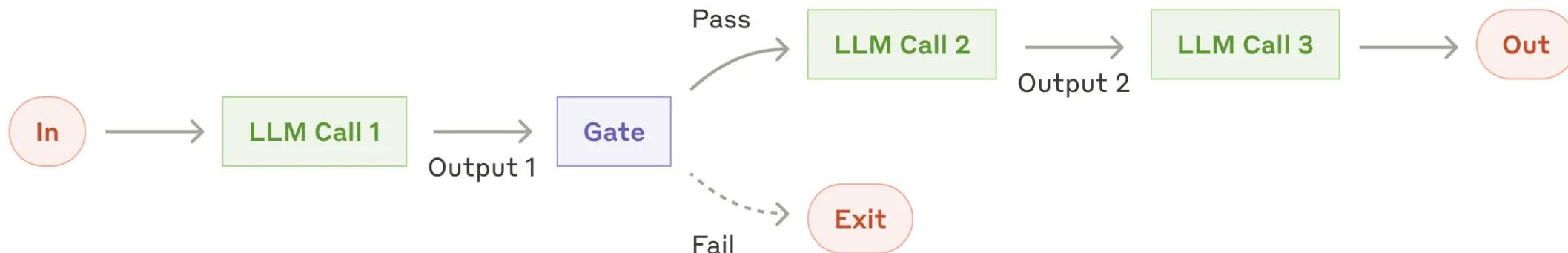
Capability	Workflow	Agent
Augmented LLMs	Yes	Yes
Control Flow	Fixed, pre-defined	Dynamic
Costs	Predictable	More unpredictable



Agentic AI: Workflows

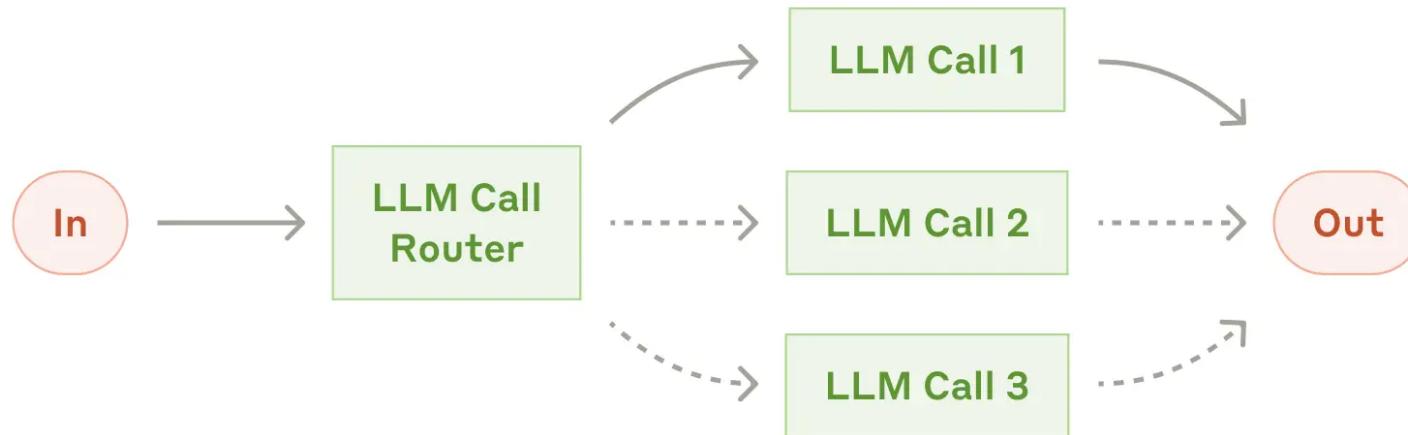
Workflow: Prompt Chaining

- **Goal:** Decompose into fixed sub-tasks; any programmatic checks can be added to ensure process is on track
- **When to Use:** Where task can be easily and cleanly decomposed into fixed subtasks
- **Use Cases:**
 - Generate marketing copy, then translate to different languages
 - Write outline of document, check if it meets specific criteria, then write document



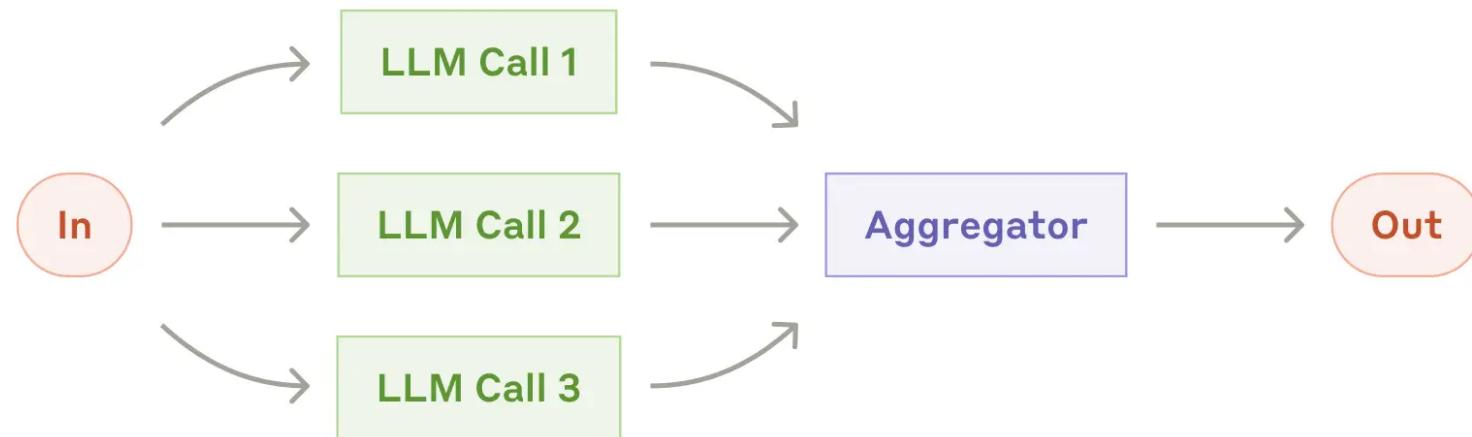
Workflow: Routing

- **Goal:** Direct an input into a specialized sub-task, ensuring separation of concerns
- **When to Use:** Where distinct categories of tasks are better handled separately, and category classification is easy
- **Use Cases:**
 - Directing customer service queries (general questions, refund requests, technical support)
 - Routing questions based on complexity to smaller/more capable models, to optimize cost and speed



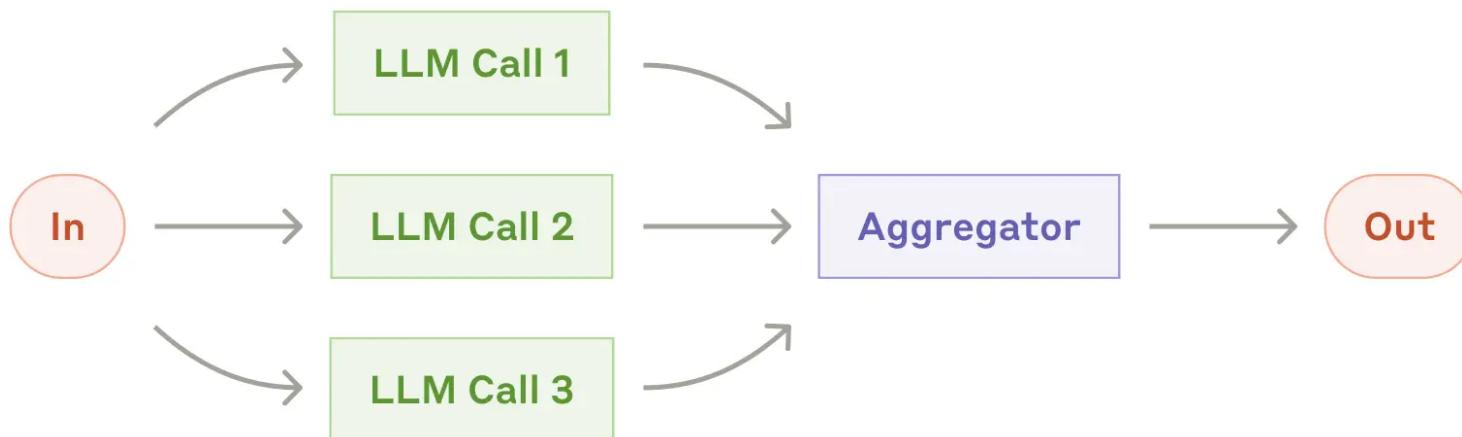
Workflow: Parallelization (Sectioning)

- **Goal:** Breaking down into different tasks and running multiple subtasks concurrently, post which aggregation is done.
- **When to Use:** Where complex tasks can be broken down and parallelized for speed
- **Use Cases:**
 - Guardrails, where one model instance processes user query and another screens for inappropriate requests
 - Automating evals, where each LLM call evaluates a different aspect of performance



Workflow: Parallelization (Voting)

- **Goal:** Running same task multiple times concurrently to get diverse outputs, post which voting is done
- **When to Use:** Where higher confidence in the output is needed
- **Use Cases:**
 - Reviewing a piece of code for vulnerabilities
 - Evaluating whether a given piece of content is inappropriate, with different thresholds to balance false positives and negatives.



Workflow: Orchestrator-Worker

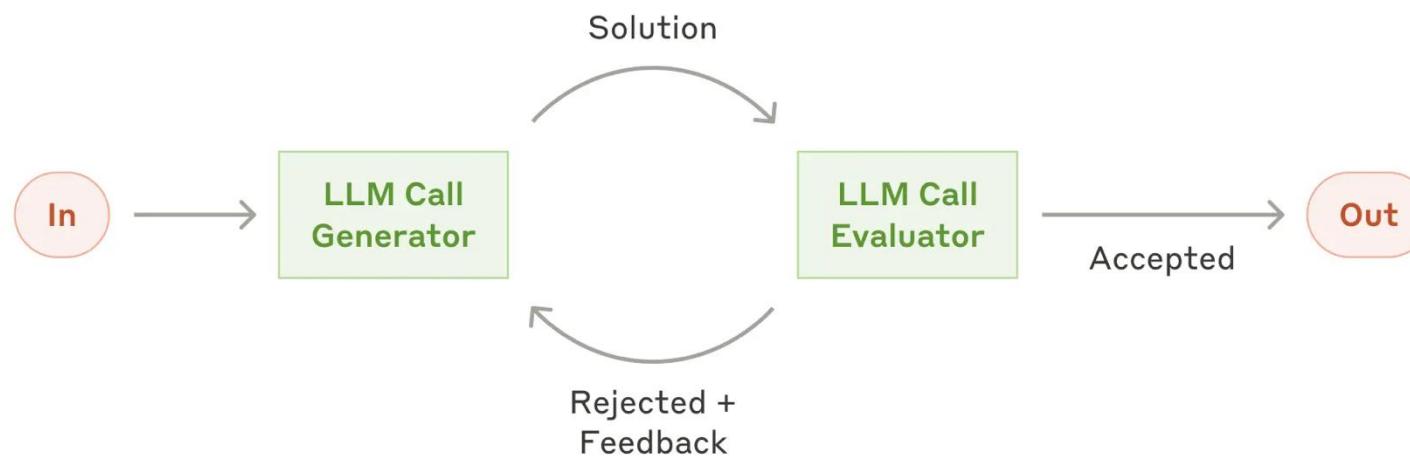
- **Goal:** Complex tasks are broken down dynamically and combined
- **When to Use:** Where subtasks can't be predicted, but need to be determined (also allows for more flexibility)
- **Use Cases:**
 - Search tasks that involve gathering and analyzing information from many sources





Workflow: Evaluator-Optimizer

- **Goal:** LLM output is validated by another
- **When to Use:** Where 1) LLM responses can be clearly evaluated (via criteria), 2) human feedback significantly improves LLM responses iteratively, and 3) whether LLM can provide such feedbacks
- **Use Cases:**
 - Literary translation, where translator LLM responses can be improved with evaluator LLM's critiques
 - Complex search tasks where evaluator decides whether further searches are needed

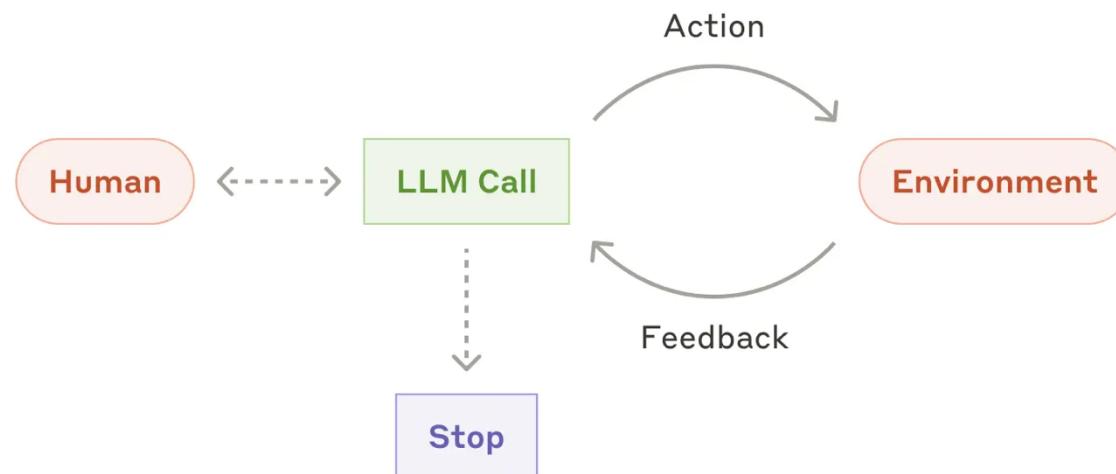




Agentic AI: Agents

In contrast, Agents -

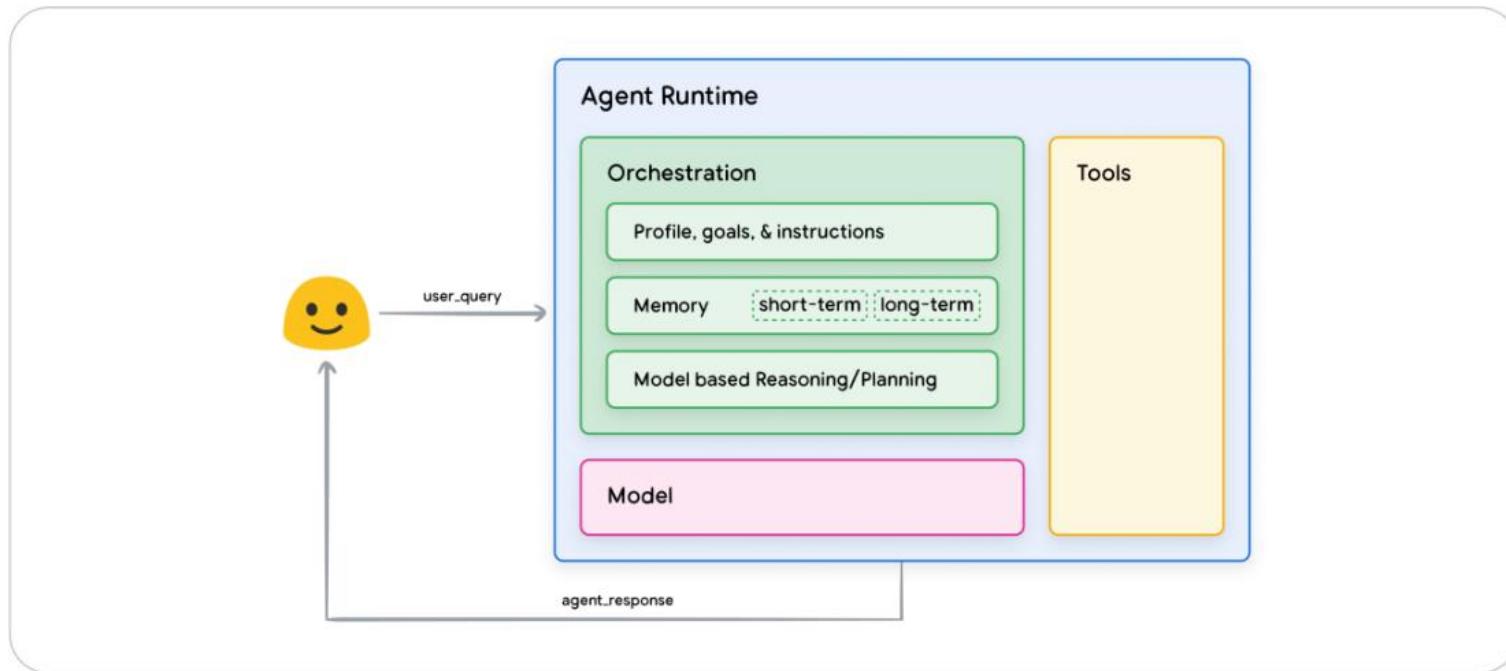
- **Goal:** With more advanced LLM capabilities (understanding complex inputs, engaging in reasoning and planning, using tools reliably, and recovering from errors), plan and operate independently while returning to human for feedback
- **When to Use:** For open-ended problems, where it's difficult to predict required number of steps or find a fixed path
- **Many Use Cases** (complex)
 - Customer support (talk, pull data, act, resolve)
 - Coding agents (design, build, test, measure)





What constitutes an AI Agent

Agents are AI systems that leverage LLMs to reason through a solution, leveraging external tools, planning (like reflection and task decomposition), and having memory (short-term or long-term).





Deeper into Tools



Tools and Function Calling

- Function calling in Large Language Models (LLMs) enables these models to **interact with external tools and APIs**, thereby extending their capabilities beyond text generation to include real-time data retrieval



Tools and Function Calling

- The function-calling feature is implemented using a combination of **fine-tuning, prompt engineering, and structured output generation** techniques.
 - **Finetuning:** Teach the LLM to call functions, by showing it examples of user query – correct function
 - **Prompt Engineering:** Include the function's name, a description of what it does, and a list of its required parameters and their data types
 - **Structured Output Generation:** LLM should generate a structured output, typically a JSON, containing the function's name and specific arguments needed
 - Note: Prompt engineering tools is as important as overall prompt engineering, as agents often fail at their tasks when they don't call tools the right way!



Tools and Function Calling

- Primary use cases that leverage function calling:
 - Agents
 - For example, if you say, "What's the weather like in Boston?", the LLM can decide that it needs a weather API to get the answer. It then generates a structured function call (e.g., `get_weather(location="Boston")`), which your application executes. The results from that API are then sent back to the LLM, which uses them to formulate a natural language response like, "The weather in Boston is currently sunny and 75°F."
 - Complex multi-turn chatbots
 - By using function calls, the chatbot can "write" to a memory or database and "read" from it in subsequent turns.
 - Batch inference feature extraction
 - For example, you could have millions of customer reviews and use an LLM with a function-calling capability to extract key information like `product_name`, `sentiment` (positive/negative), and `key_complaint`.
 - Structured output generation
 - In general!



Tools and Function Calling

- Not all LLMs are made the same, and LLMs differ in their capabilities to call the right tools

[Berkeley Function-Calling Leaderboard](#)



Tools and Function Calling

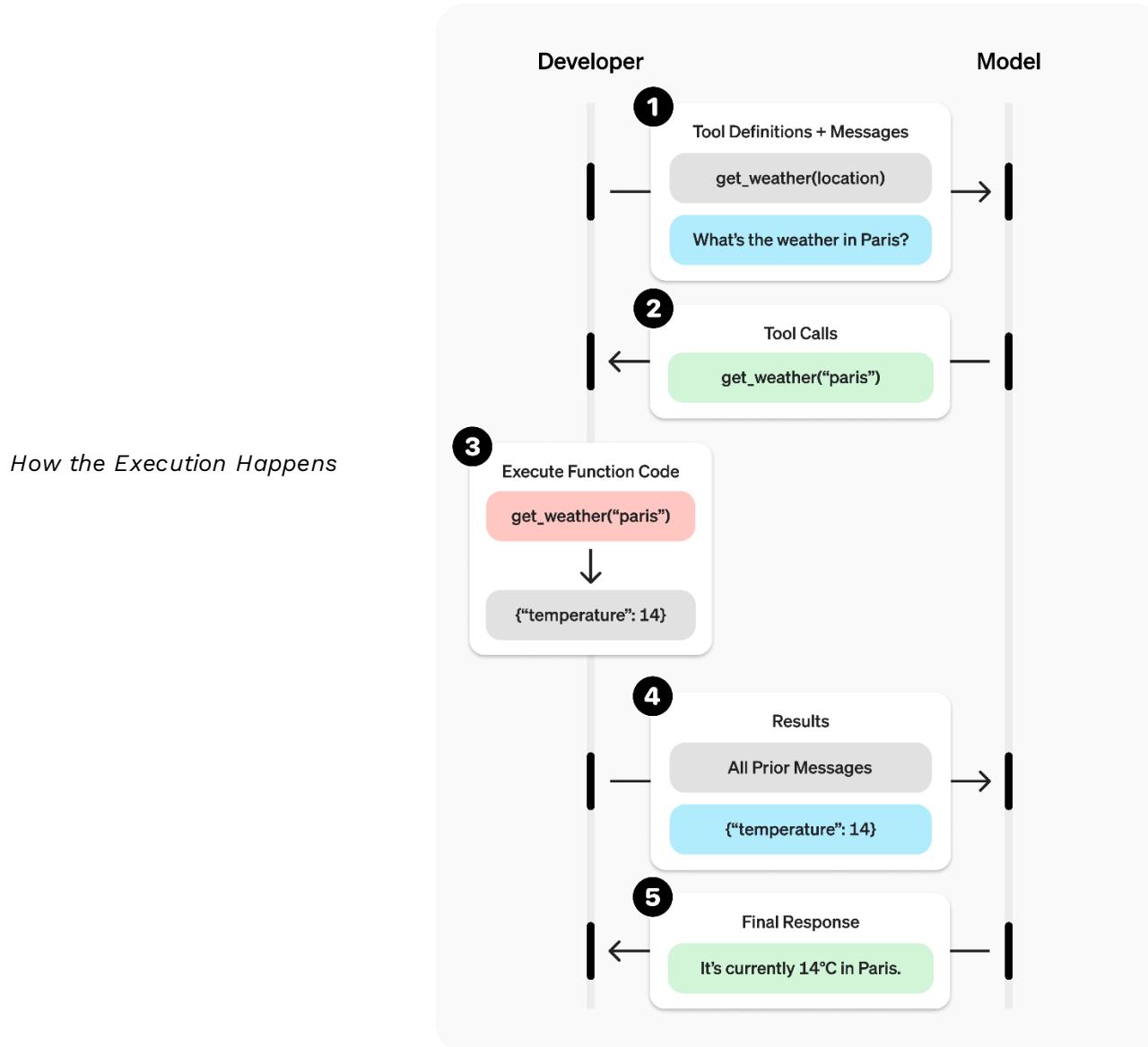


Image Source: [OpenAI](#)

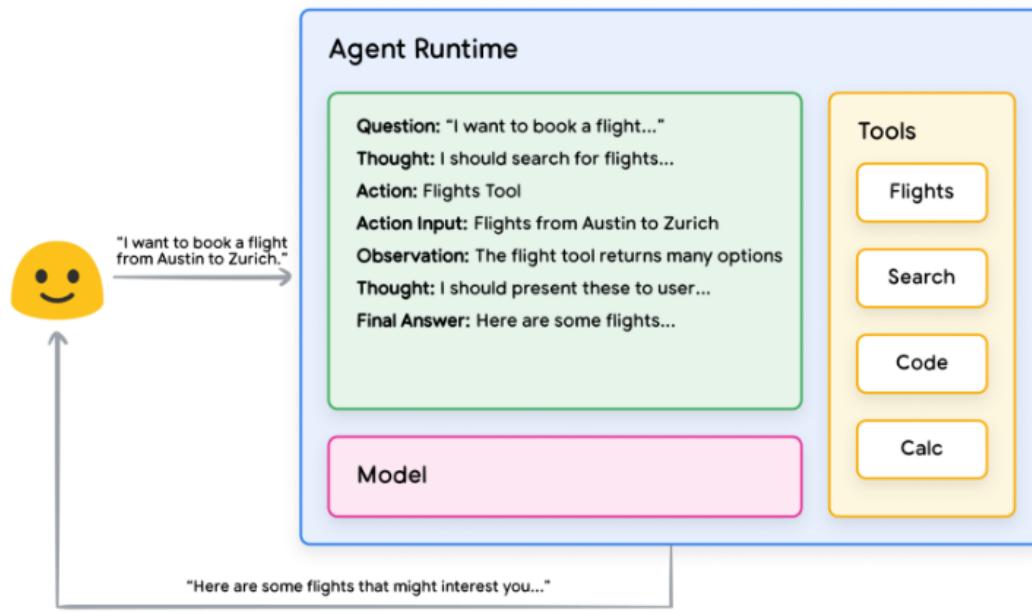
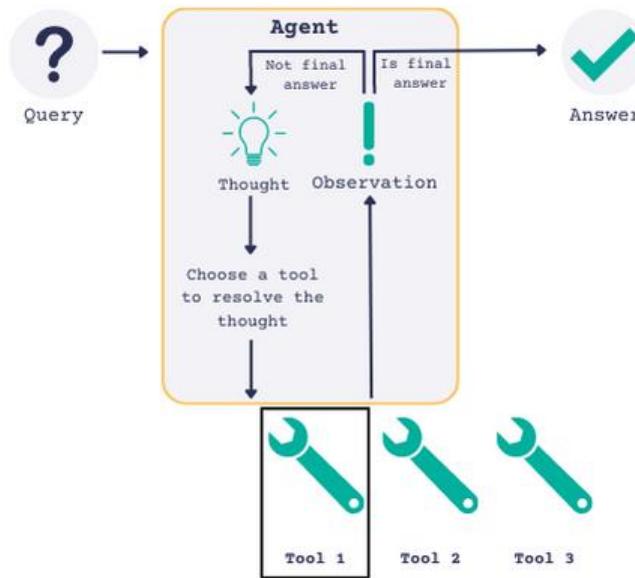


Agentic Design Patterns

The Simplest Form of Agent: ReACT Agent

The ReAct (Reasoning and Action) agent model

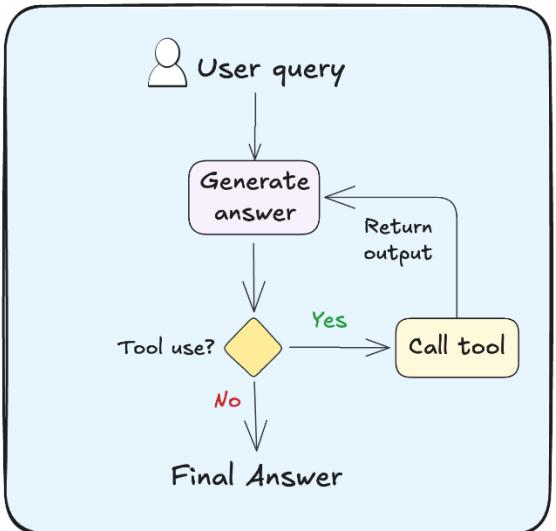
- Integrates the reasoning capabilities of large language models (LLMs) with the ability to take actionable steps
- Creates a more sophisticated system that can understand and process information
- Can evaluate situations, take appropriate actions, communicate responses, and track ongoing situations.



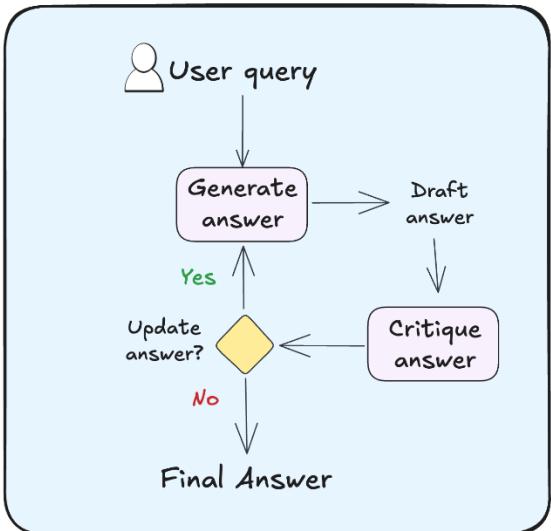


Some Single-Agent Design Patterns

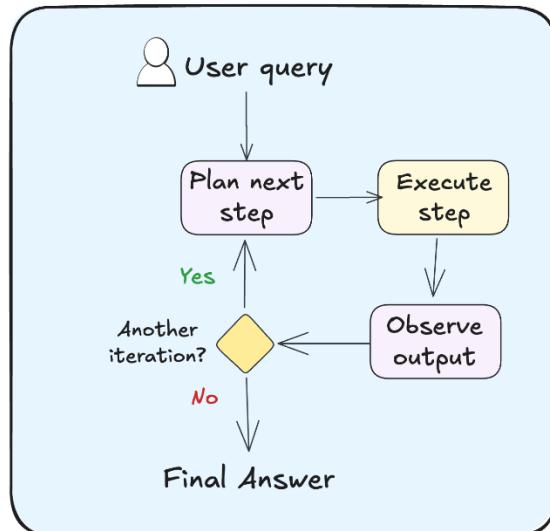
Tool Use



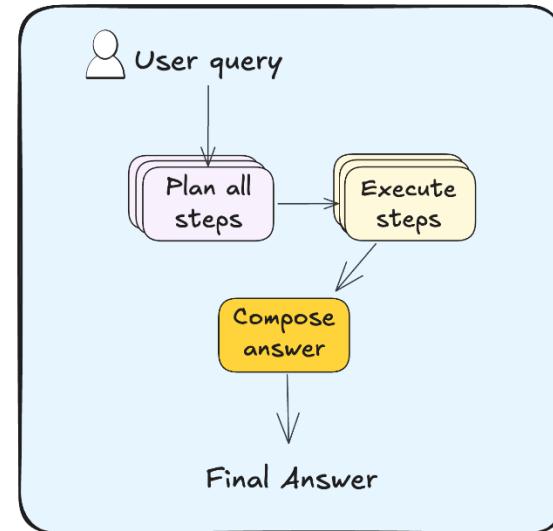
Basic Reflection



ReAct



Plan-then-Execute



Legend

- LLM (Light Blue Box)
- Tool (Yellow Box)
- Orchestration logic (Yellow Diamond)

[Image Source](#)

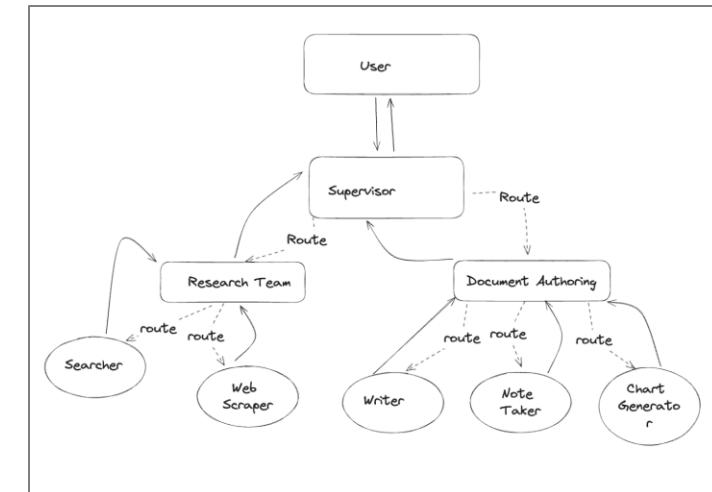
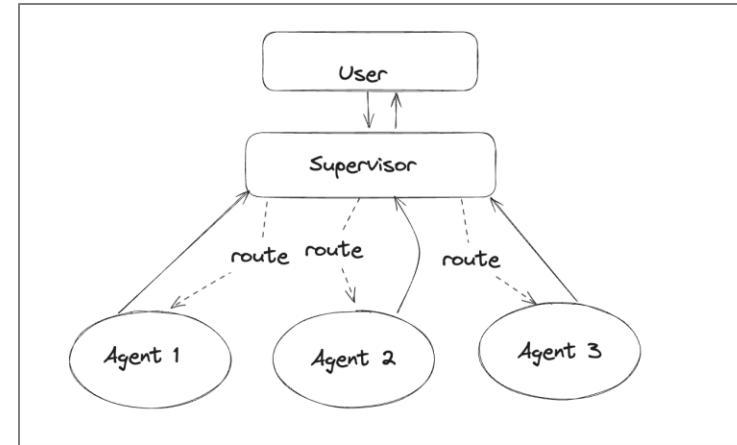
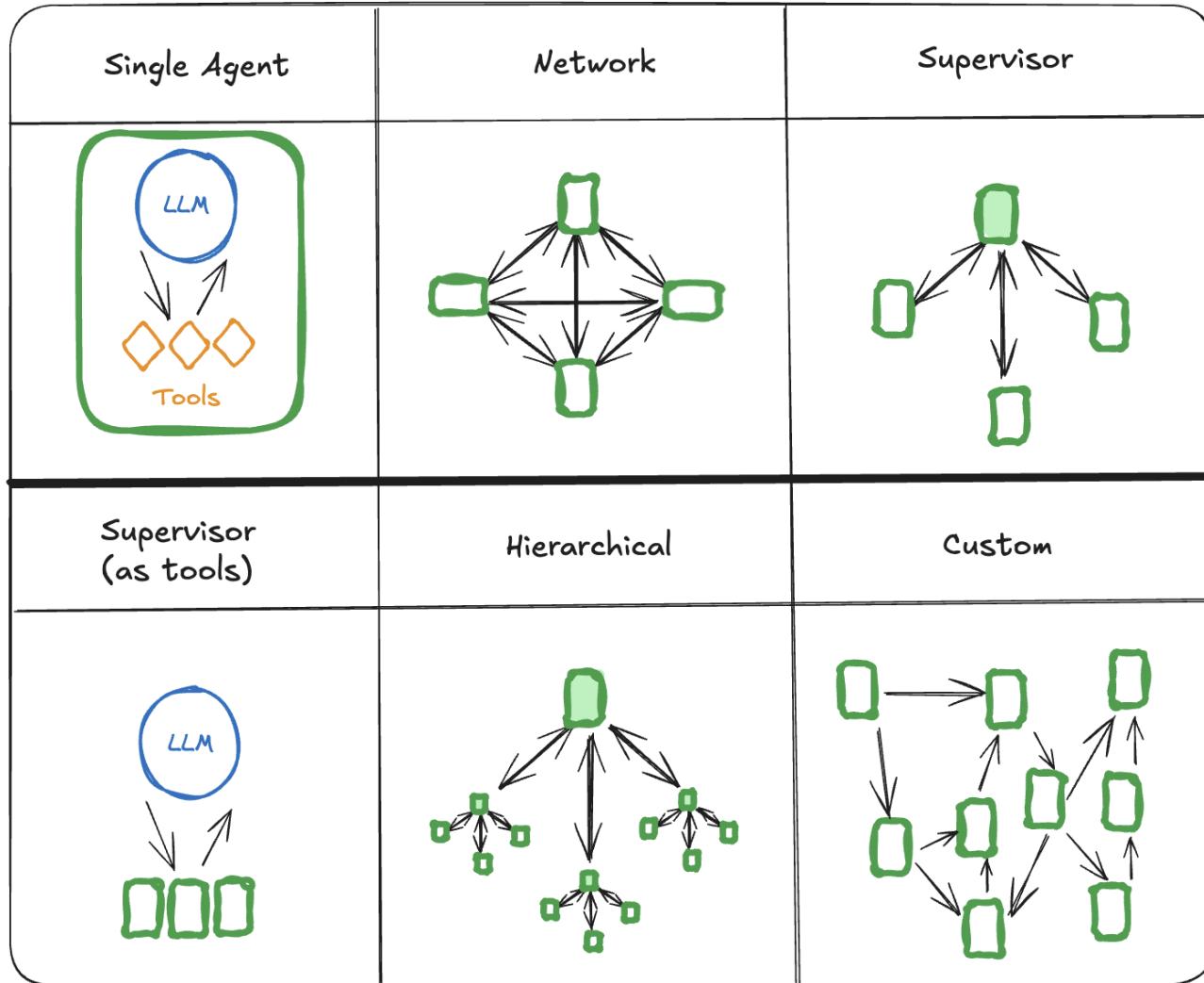


When Single-Agent Systems Fall Short

- **Overloaded Agents:** A single agent can struggle to make good decisions when it has access to too many tools, leading to poor performance.
- **Complex Context:** A single agent can have difficulty managing a large or complex conversational context, causing it to lose track of the user's request.
- **Lack of Specialization:** A single agent isn't equipped to handle a wide range of tasks that require different skills, like planning, research, and mathematical expertise.



Multi-Agent Design Patterns





Guidelines and Considerations



General Guidelines for Building Agents



Stage	Define	Design Standard Operating Procedure	Build MVP for Core Prompts	Connect & Orchestrate	Test & Iterate	Deploy, Scale & Refine
Description	Identify a realistic agent use case and gather 5-10 concrete scenarios the agent should handle	Create a step-by-step procedure, breaking down the task(s) into clear steps a human would follow	Design agent architecture; Build prompt(s) that handle the core reasoning	Connect to live data sources and build orchestration logic	Evaluate agent against examples; Identify failures & iterate to improve reliability & performance	Deploy agent & monitor how users interact; continuously refine based on real usage patterns.
Duration	~1-2 week(s)	~1-2 week(s)	~1-2 week(s)	~2-3 week(s)	~1-2 month(s)	Ongoing
Output	Clear agent task scope & concrete use cases	Clear Standard Operating Procedure of step-by-step workflow, incorporating identified scenarios	Agent architecture and core prompt(s) that works with static data	End-to-end agent with live data	Agent with systematically tested performance	Improved agent with expanded capabilities
Stakeholders	Product Owner, Subject Matter Expert	Product Manager, Subject Matter Expert	Product Manager, AI & Prompt Engineer	Engineers	Product Manager, QA Engineer	PM, Engineers, Support
Example: Building an Email Agent	Example tasks: <ul style="list-style-type: none"> Categorize email by priority Schedules meetings Answer product questions 	Step-by-step procedure: <ul style="list-style-type: none"> With incoming emails, label with email category & response priority Checks calendar availability & schedule meeting Draft response & queue for human review 	Email agent architecture design with core prompts, incl. email routing & response generation; Test with manually provided context and sample data	Example integrations: <ul style="list-style-type: none"> Gmail & Outlook for email access Google & Outlook Calendar CRM database for sender context 	Evaluate email responses over success criteria: response quality, accuracy, and professional tone	Monitor traffic, use cases, and feedback. Iterate to improve performance and add additional features



Considerations When Building Agents

Risks

- Unpredictable path
- Unpredictable output
- Unpredictable costs

The Need to Monitor

- Guardrails: they ensure agents behave safely, consistently and within intended boundaries
- Cost, Latency, Tracing etc.
- *Enter AI Gateway as a one-stop solution!*

Recommendation

- Build the right system, not always the most sophisticated one
- Start with simple prompts, optimize with evaluation, then go to multi-step agentic systems



Comparison of Popular Agentic AI Frameworks

Feature/Framework	LangGraph	CrewAI	AutoGen
Core Idea	Graph-based workflows, powerful state management for complex cycles.	Role-based multi-agent systems for intuitive collaboration.	Flexible conversation patterns and agent topologies.
Best For	Complex, cyclical processes; human-in-the-loop; production-ready applications.	Team-based workflows; automated customer support, research, or development.	Complex problem-solving; code generation & debugging; diverse agent interactions.
Key Strength	Superior state control & flow; highly customizable; production-grade.	Intuitive collaborative intelligence; clear role delegation.	Flexible conversational dynamics; strong code execution & debugging.
Learning Curve	Steeper (graphs & state)	Medium	Medium
Scalability	High (built for production)	Medium-High	Medium-High
Tooling	Strong via LangChain	Flexible custom tools	Strong (code execution, APIs)

- **Choose LangGraph if:** You need precise control over complex, looping workflows and robust state management for production.
- **Choose CrewAI if:** You want to model collaborative teams of AI agents with distinct roles and intuitive interaction.
- **Choose AutoGen if:** You need highly flexible agent conversations, especially for code-centric problem-solving and iterative tasks.



Hands-on Tutorial



Assessment (14%)

Day 3: Agents (cont.) and Model Context Protocol



Module 5: Agentic RAG



Hands-on Tutorial



Module 6: Model Context Protocol



Pre-MCP Context Management

This was a set of one-off solutions that required extensive engineering, and wasn't completely proactive from the model's side.

Context management

Truncation and Sliding Windows

Summarization

Template-based prompts

External context enhancement

Static Prompting (pre-tools)

RAG

Prompt chaining and agents

Function calling mechanisms

Model either knew something or didn't

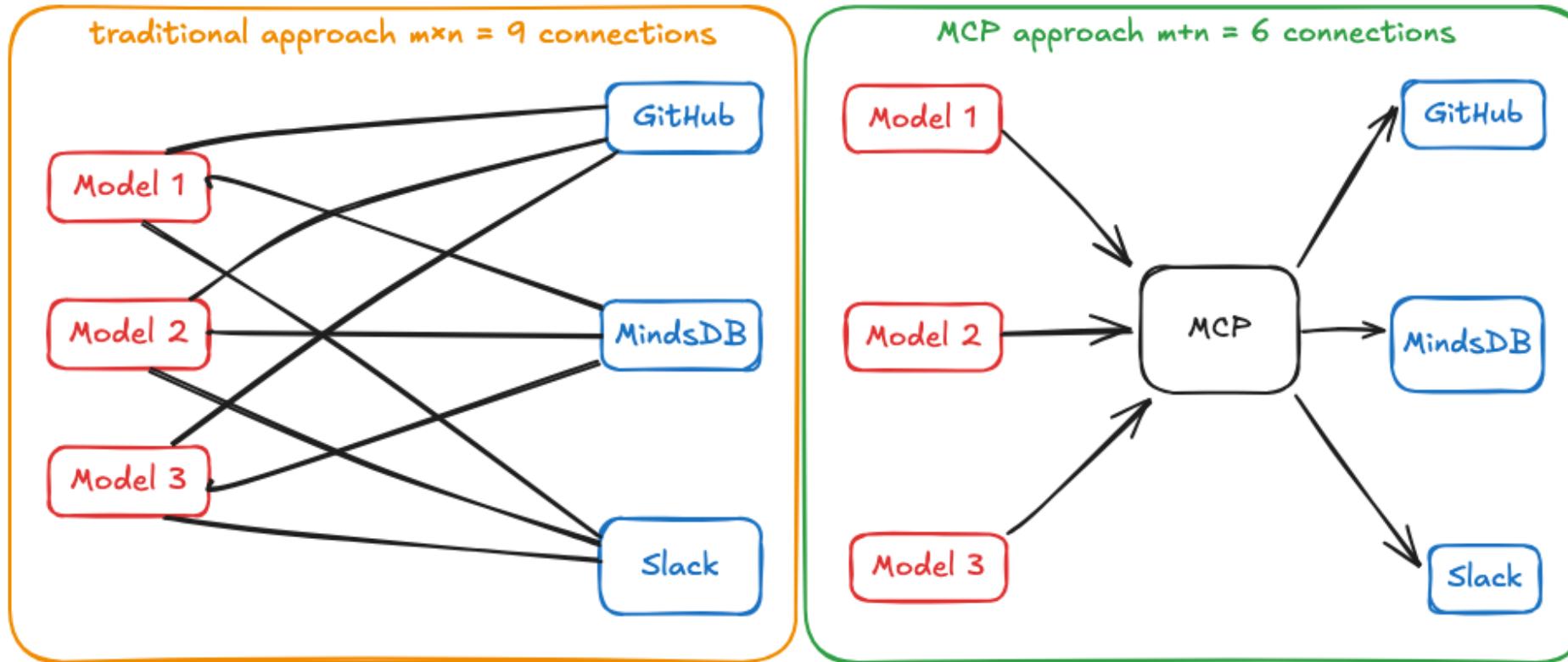
Model doesn't initiate lookups/developer puts the logic in

Integrating new tools, meant custom prompt logic and parsing

See MxN integration problem



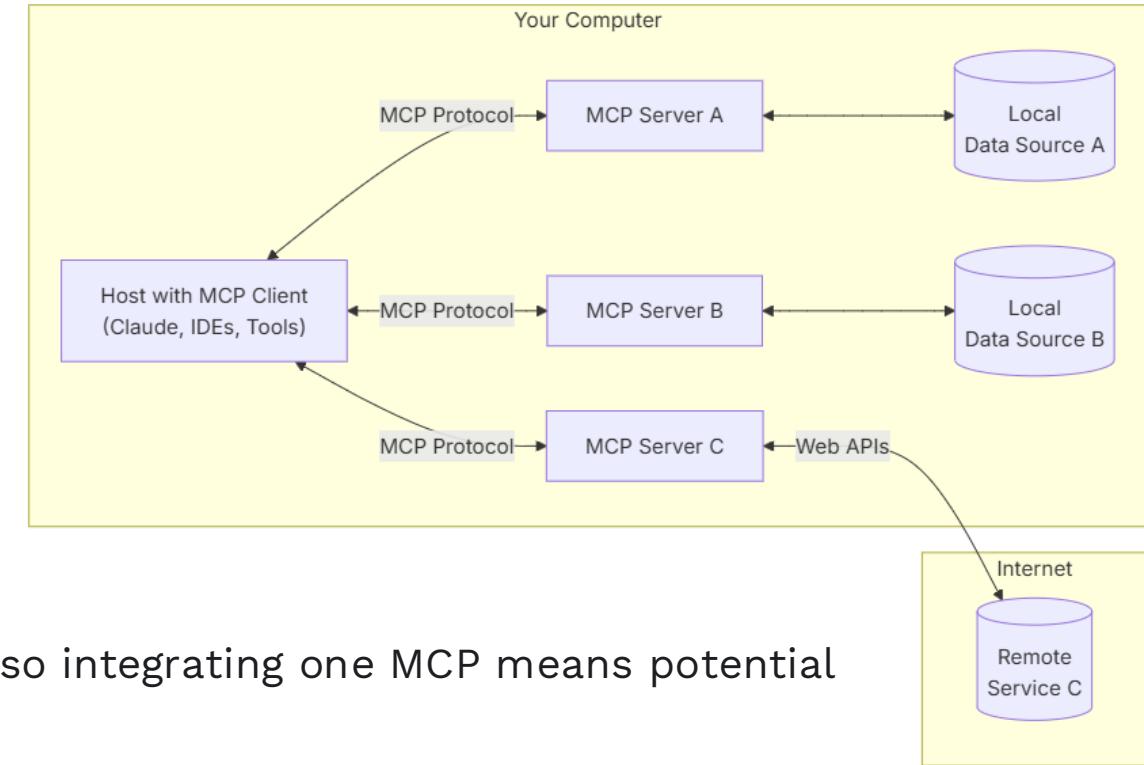
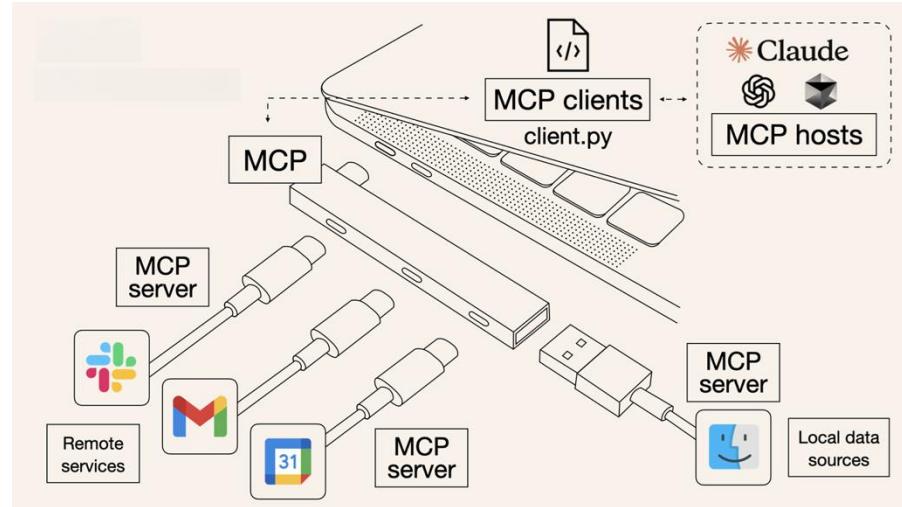
MxN Integration Problem



[Source](#)

Model Context Protocol (MCP)

MCP is an open protocol that standardizes how applications provide context to LLMs.



Single protocol: MCP acts as a standardized "connector," so integrating one MCP means potential access to multiple tools and services, not just one

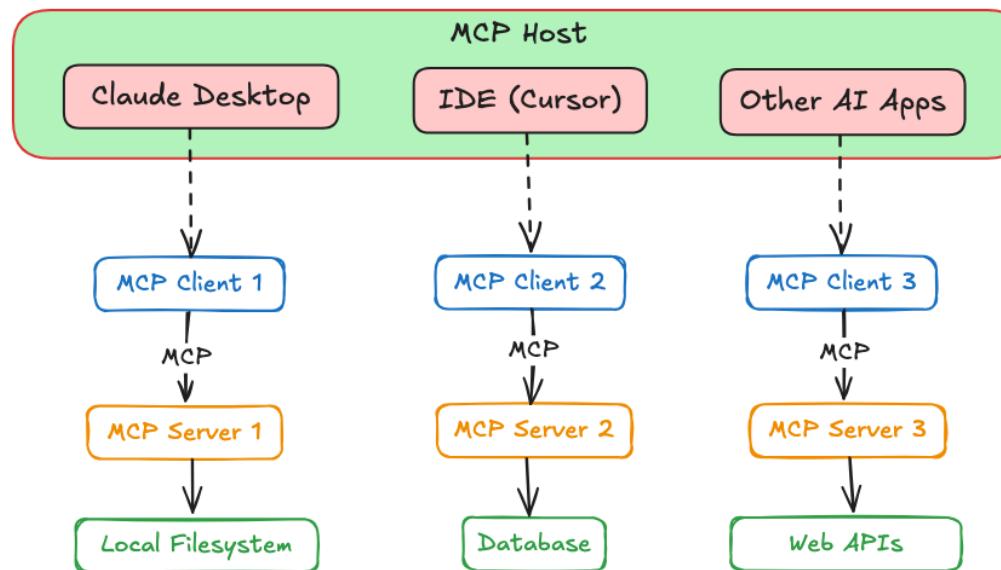
Dynamic discovery: MCP allows AI models to dynamically discover and interact with available tools without hard-coded knowledge of each integration

Two-way communication: MCP supports persistent, real-time two-way communication. The AI model can both retrieve information and trigger actions dynamically

Architecture

MCP (Model Context Protocol) uses a client-server architecture to connect AI models with external resources, built on three key components: the **Host**, **Clients**, and **Servers**.

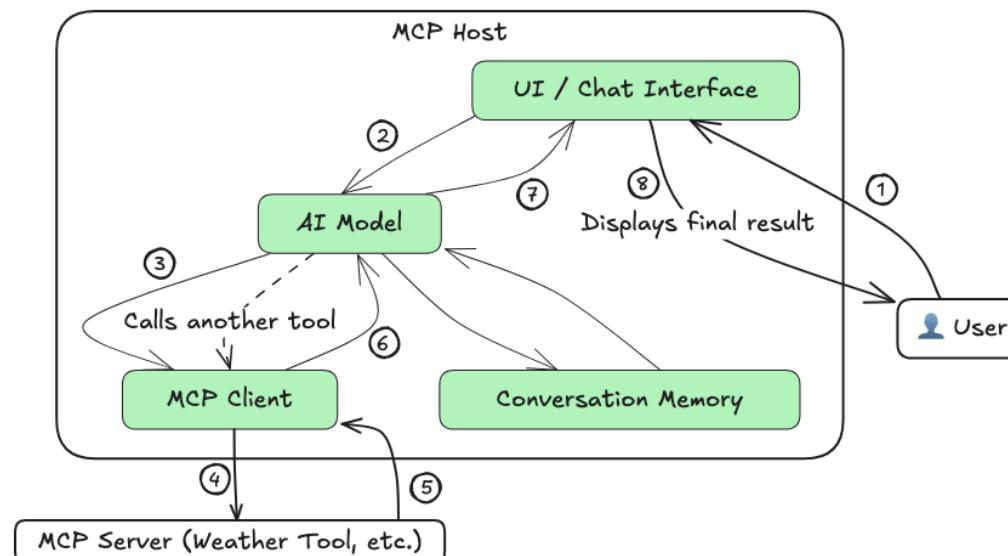
- The **Host** is the AI application or agent the user interacts with (e.g., a desktop app, IDE plugin, or custom LLM app). It can simultaneously connect to multiple MCP Servers.
- An **MCP Client** acts as an intermediary, managing each server connection. Each client handles communication with a single MCP Server, providing security isolation. The Host spawns one client per server, creating a dedicated, one-to-one link.
- The **MCP Server** is an external program implementing the MCP standard. It offers specific capabilities, including tools, data access, and domain-related prompts. Servers can interface with databases, cloud services, and other data sources, with existing examples for Google Drive, Slack, GitHub, various databases, and web browsers.





How it Works

- 1. Capability Discovery:** The MCP client fetches the list of available tools/resources from the server and shares it with the AI model.
- 2. Augmented Prompting:** The user's query is sent to the model along with descriptions of the server's tools/resources, enabling the model to understand what external actions are possible.
- 3. Tool/Resource Selection:** The model decides if an external tool is needed and responds with a structured request (per MCP spec) indicating the desired tool (e.g., "Weather API").
- 4. Server Execution:** The client invokes the requested tool on the MCP server, which performs the action (e.g., API call or database query) and returns the result.
- 5. Response Generation:** The model receives the result, integrates it with its own knowledge, and generates a final, enriched response for the user.





MCP vs. RAG and Vector Search

- **RAG** appends retrieved documents directly into prompts as raw text — useful but unstructured.
- **MCP** offers **structured context management**: retrieved data is slotted into labeled *Resource* blocks (e.g., “Retrieved Documents”), keeping different types of information organized.
- You can **combine RAG with MCP**: fetch via vector search, then pass results as structured context — easier to manage, update, and debug.



MCP vs. Monolithic Prompts and Finetuning

- Traditional methods relied on:
 - **Monolithic prompts** – bulky, hard to edit, expensive (token-wise).
 - **Fine-tuning** – slow, costly, hard to adapt on the fly.
- **MCP** enables **modular prompting**:
 - Define reusable context layers (e.g., “Policy Guidelines”, “User History”).
 - Update individual layers without rewriting or retraining.
- Saves time and tokens while improving flexibility and maintainability.



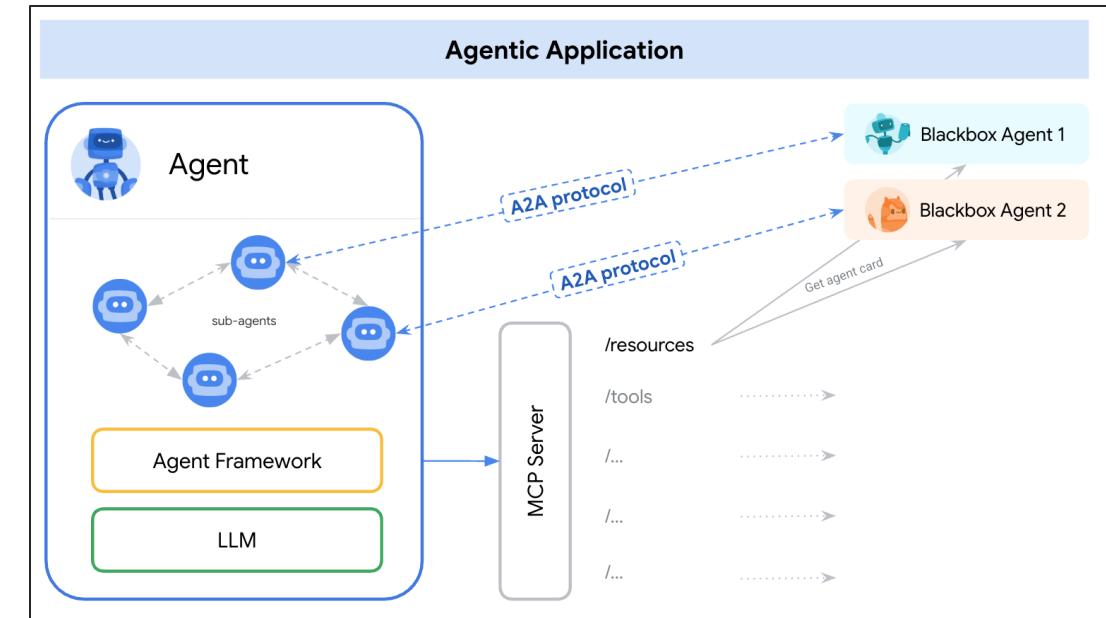
MCP vs. Function Calling and Plugins

- Function calling/tools in LLMs (like OpenAI plugins) are **powerful but fragmented** — each provider has its own setup.
- **MCP standardizes tool use** across models with unified *Tool*, *Prompt*, and *Resource* primitives.
- **Interoperable & extensible:**
 - Write once, work across providers (e.g., OpenAI, Claude, Llama).
 - Covers non-functional context needs (like guidance, templates) beyond API calls.

MCP vs. A2A

A2A and MCP are both complementary protocols

Feature	Agent2Agent Protocol (A2A)	Model Context Protocol (MCP)
Interaction	Between agents (from different frameworks also!)	Between agent and tools
Use Case	Delegating tasks to other agents	Enhancing context with external data
Dependency	Relies on other agents	Relies on tool integrations
Architecture	Multi-agent system	Tool-augmented single-agent system



MCP and A2A Integration

- ✓ By representing **A2A agents as MCP resources**—encapsulated within their **AgentCards**—AI applications facilitate **dynamic agent discovery**.
- ✓ This allows agents on an MCP server to **connect and collaborate** via the A2A protocol.

Image source [here](#)



Basics of FastMCP and CrewAI



Important Links for FastMCP

<https://gofastmcp.com/tutorials/mcp>

<https://gofastmcp.com/tutorials/create-mcp-server>

<https://gofastmcp.com/getting-started/quickstart>



What CrewAI Says

How CrewAI Compares

CrewAI's Advantage: CrewAI combines autonomous agent intelligence with precise workflow control through its unique Crews and Flows architecture. The framework excels at both high-level orchestration and low-level customization, enabling complex, production-grade systems with granular control.

- **LangGraph:** While LangGraph provides a foundation for building agent workflows, its approach requires significant boilerplate code and complex state management patterns. The framework's tight coupling with LangChain can limit flexibility when implementing custom agent behaviors or integrating with external systems.

P.S. CrewAI demonstrates significant performance advantages over LangGraph, executing 5.76x faster in certain cases like this QA task example ([see comparison](#)) while achieving higher evaluation scores with faster completion times in certain coding tasks, like in this example ([detailed analysis](#)).

- **Autogen:** While Autogen excels at creating conversational agents capable of working together, it lacks an inherent concept of process. In Autogen, orchestrating agents' interactions requires additional programming, which can become complex and cumbersome as the scale of tasks grows.
- **ChatDev:** ChatDev introduced the idea of processes into the realm of AI agents, but its implementation is quite rigid. Customizations in ChatDev are limited and not geared towards production environments, which can hinder scalability and flexibility in real-world applications.

<https://github.com/crewAIInc/crewAI?tab=readme-ov-file#how-crewai-compares>



Hands-on Tutorial



Assessment (14%)

Day 4: Deep-Dive into AI Gateway



Module 7: Deep-Dive into AI Gateway



The Problem



The Productionizing Nightmare

- Prototyping with GenAI is easy.
- Moving to production with a decentralized approach is messy.
- This applies to both open-source and proprietary models.





Real-Life Scenarios

Scenario 1

1



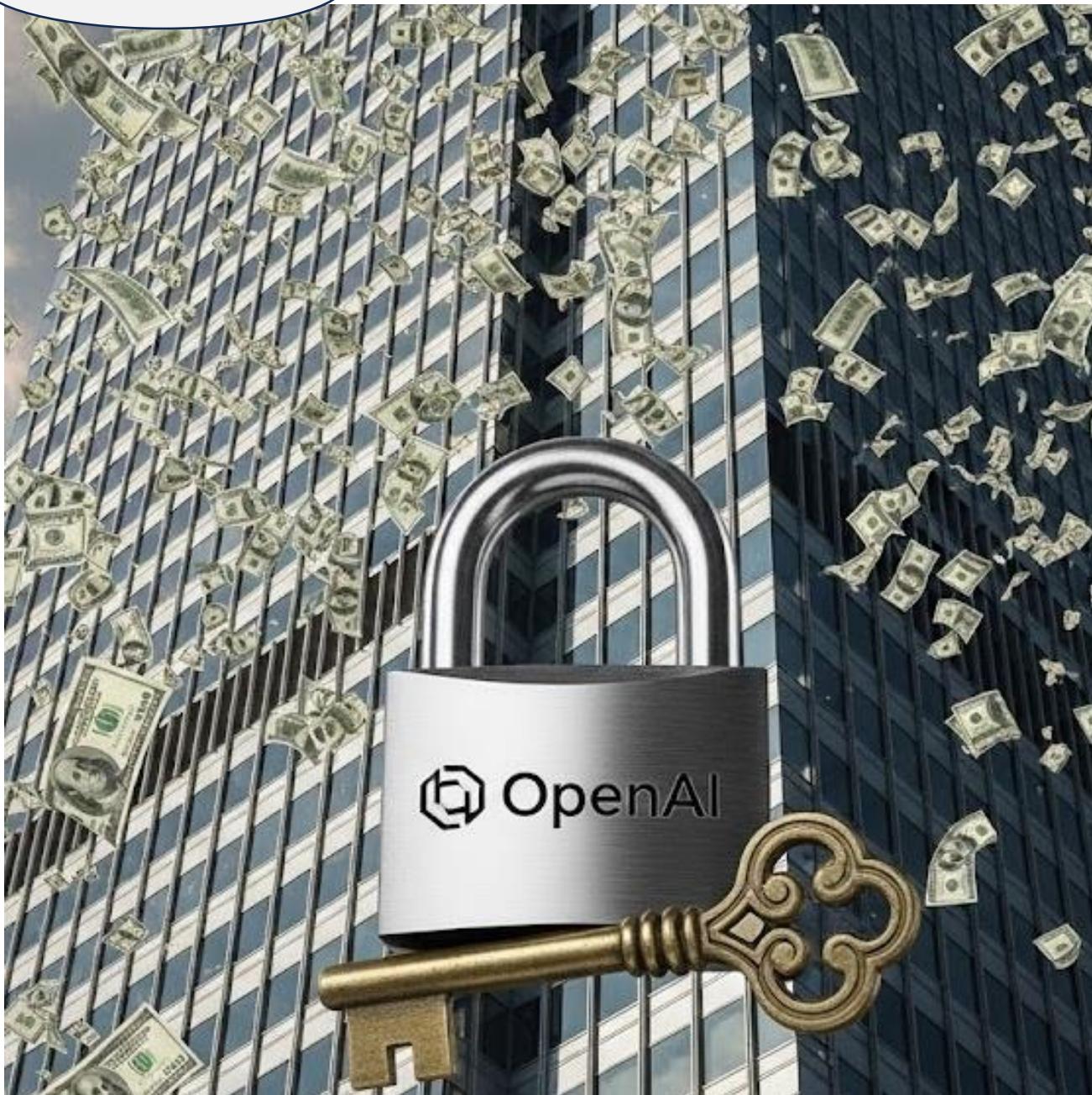
The Compliance & Audit Nightmare

- **The Problem:** Your teams are handling sensitive data.
- **The Risk:** Without a centralized, verifiable record of every prompt and response, you lack an auditable chain of custody.
- **The Impact:** You cannot prove data was handled securely in a regulatory audit.

What if you had to tell your CEO

that a mission-critical AI project was on hold because your company could face a multimillion-dollar fine for non-compliance?

Scenario 2



Unpredictable Budget & Vendor Lock-in

- **The Problem:** Each business unit manages its own budget and vendor contracts.
- **The Risk:** No central way to monitor token usage or set granular spending caps.
- **The Impact:** You're locked into a single vendor's pricing, making it impossible to switch when better options are available.

Imagine trying to explain to CFO

that your company's AI spend is growing by 30% month-over-month, and you have no way to prove which teams are responsible or how to rein it in.

Scenario 3



The Operational Continuity Crisis

- **The Problem:** Your GenAI application relies on a single-point-of-failure—a single LLM provider.
- **The Risk:** A third-party outage can bring your entire business to a halt.
- **The Impact:** No automatic fallbacks or load balancing to ensure uptime, leading to lost revenue and reputational damage.

How do you explain to your board

that you have no disaster recovery plan for
your most critical AI systems?



Pain Points



The Problems Run Deeper

- These critical business problems are caused by day-to-day technical complexities.
- Your teams are facing a fragmented and chaotic AI stack for development, deployment and governance.

1. Integration Maze

2. Inconsistency
and Versioning

3. Scalability and
Reliability

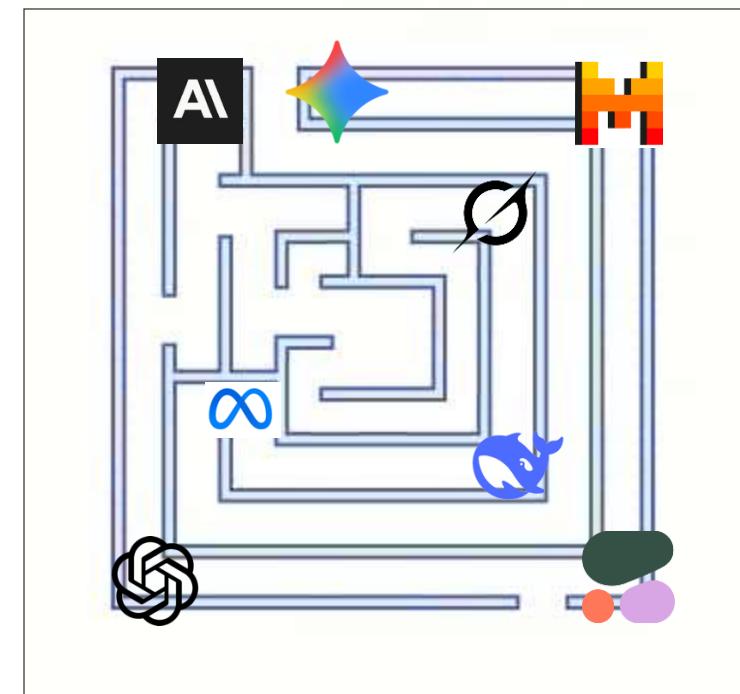
4. Lack of
Governance

5. Open-Source
Integration

6. Observability
Black Hole

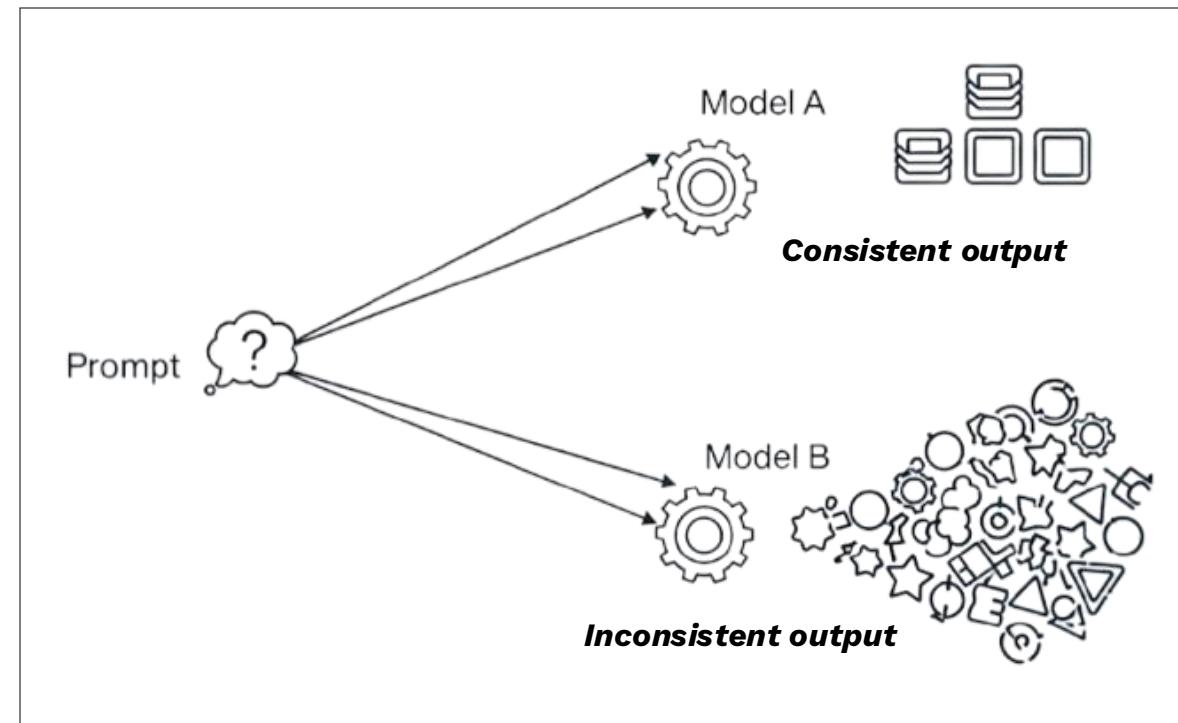
1. Integration Maze

- **Problem:** Developers write custom code for each new model.
- **Impact:** Your system becomes fragile and difficult to maintain.
- **Risk:** Switching to a new model requires a massive, time-consuming refactor.



2. Inconsistency & Versioning

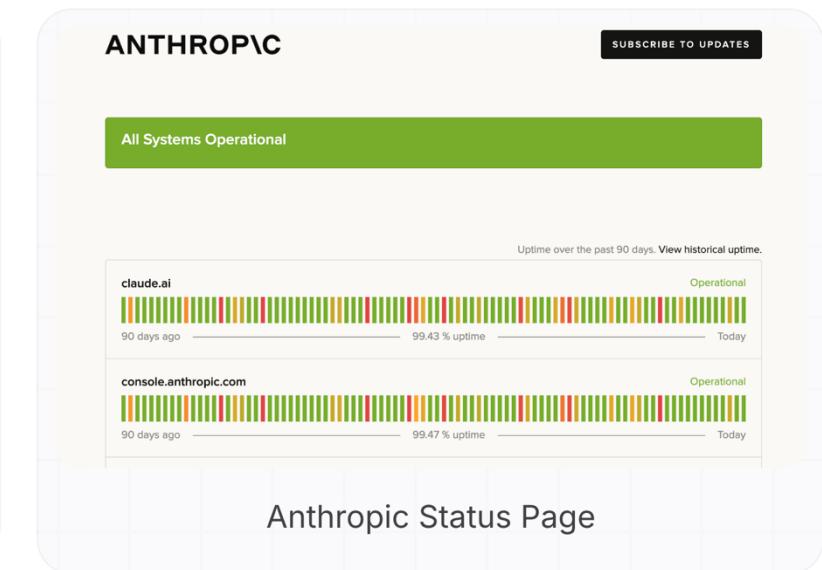
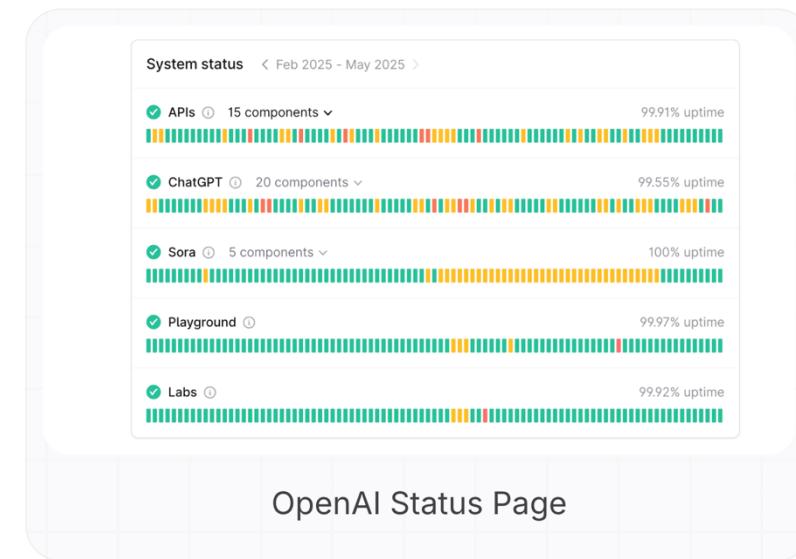
- **Problem:** Prompts are hard-coded in different places.
- **Impact:** You can't A/B test or ensure everyone is using the latest, most effective prompt.
- **Risk:** The same prompt can produce unpredictable results depending on the model.





3. Scalability & Reliability

- **Problem:** Your application is at the mercy of a single model provider.
- **Impact:** You get hit with rate limits during traffic spikes, or deal with service outages.
- **Risk:** Your entire application can go down with it.





4. Lack of Governance

- **Problem:** As you scale GenAI in your organization, you lose control over security, access, and costs.
- **Impact:**
 - **Security:** You cannot apply a consistent security policy or centrally manage API keys.
 - **Access:** You cannot enforce a standard like Role-Based Access Control (RBAC).
 - **Costs:** You cannot set cost-based or token-based quotas for budget control.
- **Risk:** Every team operates in a silo, and the risk of data leakage, unauthorized access, or accidental overspending skyrockets.

5. Open-Source Integration

- **Problem:** Integrating self-hosted, open-source models into an existing GenAI stack is a huge pain.
- **Impact:** You have to handle all the infrastructure yourself (GPU scheduling, autoscaling, deployment) and lose the consistency of a unified API.
- **Risk:** It becomes nearly impossible to manage both commercial and open-source models side-by-side.

6. Observability Black Hole

- **Problem:** When a request fails or a model hallucinates, you're flying blind.
- **Impact:** You have no centralized way to debug.
- **Risk:** You can't see the history, latency, or costs, making it impossible to troubleshoot and improve.





The Solution: Enter AI Gateway



What is an AI Gateway

AI Gateway is a **middle layer** for all applications to talk to **external/internal AI services**. It eases the complexity of using GenAI with everything in one place - from security to costs.



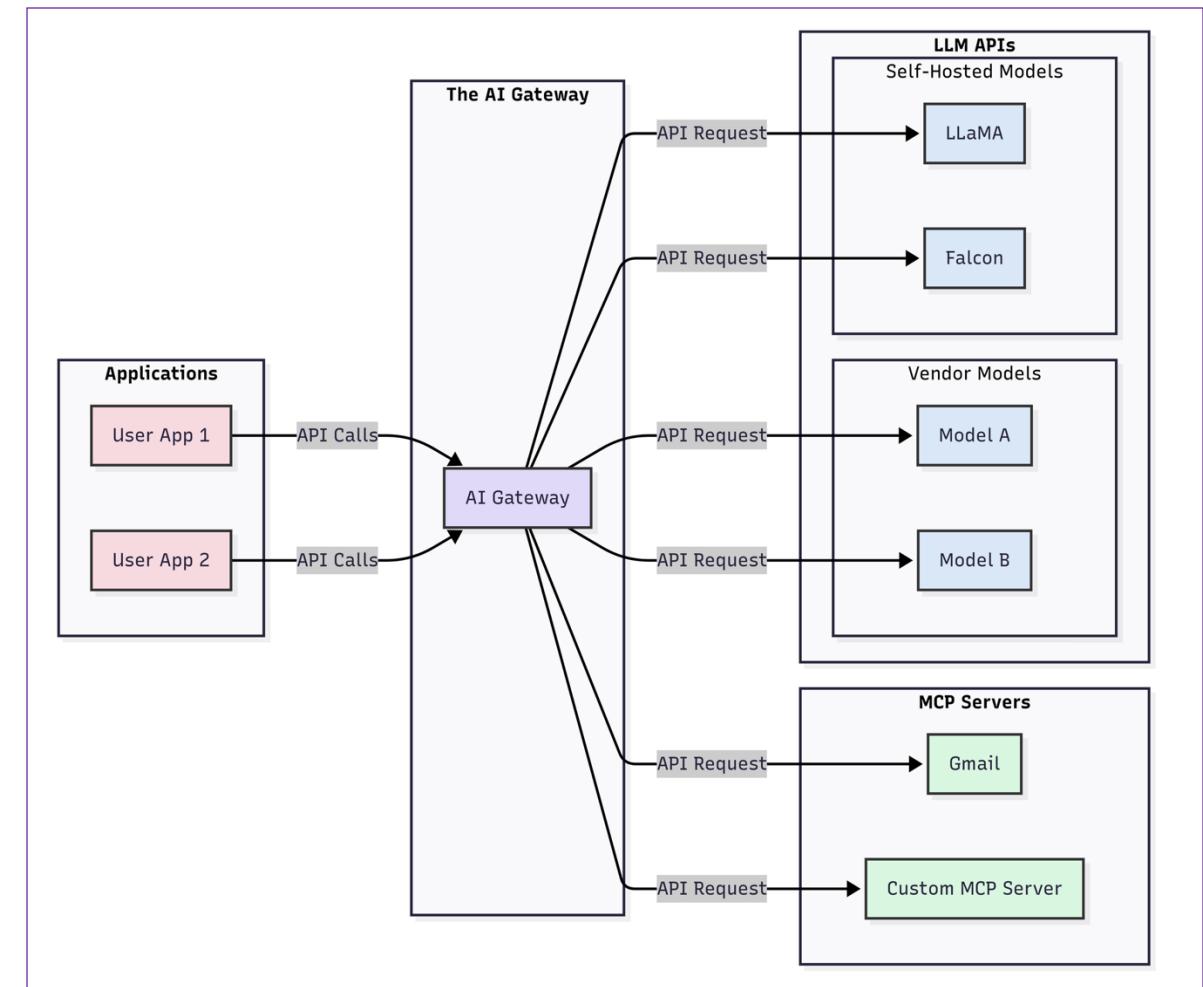
Middle layer

It sits between user applications and any number of AI services



External/internal AI services

These can be anything from self-hosted to proprietary LLMs, or MCP servers





Diving Deeper into AI Gateway

AI Gateway is a specialized **middleware** component and **egress proxy** that serves as a **centralized control plane** providing a **unified observable interface** for the entire GenAI stack.

Middleware

- Abstraction layer between user apps and LLM APIs
- Operates at application level
- Middle position in stack



Egress Proxy

- Reverse proxy, manages outbound traffic (apps to LLMs)
- Data plane, at network level
- This defines the direction of traffic



Centralized Control Plane

- Consolidates all application-level traffic into single point
- Control plane, at network level
- This is controller directing all traffic

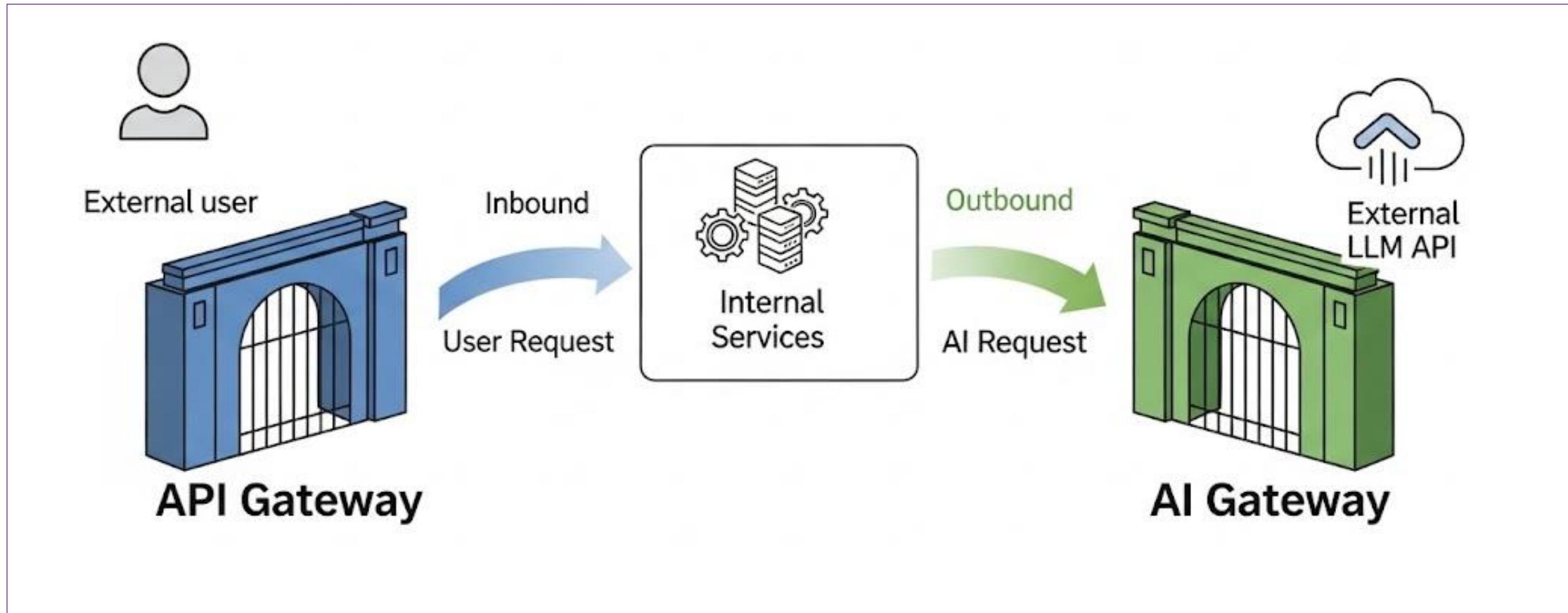


Unified Observable Interface

- Provides centralized policy enforcement and full observability
- Single user-friendly API at application level
- This is the final benefit of the AI Gateway



API Gateway vs. AI Gateway



API Gateway is primarily an [ingress proxy](#). AI Gateway is typically an [egress proxy](#).



Key Features



Centralized Access and Enterprise-Grade Security



Unified Access

Connect to multiple LLM providers (e.g., OpenAI, Anthropic, Cohere, in-house models) via a **single API** — no need to manage individual SDKs, endpoints, or API keys.



Authentication & Authorization

Enforce secure access with centralized key management. Developers get **scoped API keys** & root credentials stay secure with tools like AWS SSM, GCP Secret Manager, or Azure Vault.



Role-Based Access Control (RBAC)

Control who can access what — down to specific models or operations — **ensuring compliance** with enterprise security standards.



Smart Routing, Fallbacks, and Load Management

Routing & Fallbacks

Route requests based on latency, cost, or model reliability. Automatically retry or failover to alternate providers when needed.

Rate Limiting & Load Balancing

Enforce quotas per user or team, and balance load across providers to ensure high availability and fairness.

Caching

Reduce latency and token spend by caching repeated prompts and responses — improving performance without sacrificing accuracy.



Full Visibility into LLM Performance and Usage



Performance Monitoring

Track latency, error rates, and token throughput per model endpoint — enabling **SLA adherence** and early issue detection.



Usage Analytics

View detailed logs and dashboards showing who accessed which model, when, and how — helping teams **understand adoption** and behavior.



Cost Management

Attribute token-level costs to users, teams, or features. **Prevent budget overruns** and gain transparency into model spend.



Built to Plug into Your GenAI Stack

API Integrations

Easily connect to tools like evaluation pipelines, prompt guardrails, vector databases, or monitoring platforms.

Custom Model Support

Bring your own fine-tuned or proprietary models into the Gateway — run them alongside commercial providers with a consistent interface.

Composable Architecture

Designed to extend with your workflows, tools, and compliance stack — from dev sandboxes to production deployments.



Why AI Gateway Matters

- One gateway to rule all LLMs
- Enforced security, roles, and cost controls
- Intelligent routing, caching, and fallbacks
- Full observability into usage and performance
- Plug-and-play with your AI/ML ecosystem

AI Gateway provides a structured and scalable foundation for enterprise GenAI!



Revisiting Scenarios with AI Gateway



How AI Gateway Resolves the Real Risks



The Risk:

- ✗ No centralized logs = No audit trail
- ✗ Exposure to regulatory non-compliance

Solved by AI Gateway:

- Unified logging of prompts & responses
- Role-based access control



The Risk:

- ✗ No cost visibility, no controls = runaway spend
- ✗ Locked into a single LLM vendor

Solved by AI Gateway:

- Token-level cost tracking by team/user
- Usage analytics + spending caps
- Multi-vendor routing + abstraction



The Risk:

- ✗ Single provider failure = business outage
- ✗ No fallback or DR (disaster recovery) strategy for GenAI

Solved by AI Gateway:

- Multi-provider routing + auto-fallbacks
- Load balancing across endpoints
- Built-in caching for resilience

Key Takeaway: AI Gateway isn't just infrastructure — it's how you make GenAI safe, scalable, and enterprise-ready.



Overview of Features



TrueFoundry AI Gateway is a **centralized control plane** for Agentic AI, that manages interactions between LLMs & applications with **unified access, security, observability, control**.

Capabilities of AI Gateway

Tracing

- Audit each step of the complex workflow
- For RAG/Agent Building

Fallback and Load Balancing

- Fallover for high availability
- Load balance for cost/quality

Caching & Optimization

- Semantic caching for repeated prompts

Security & Access Controls

- Centralized API key management
- RBAC and Audit Trail
- Rate Limiting and Throttling

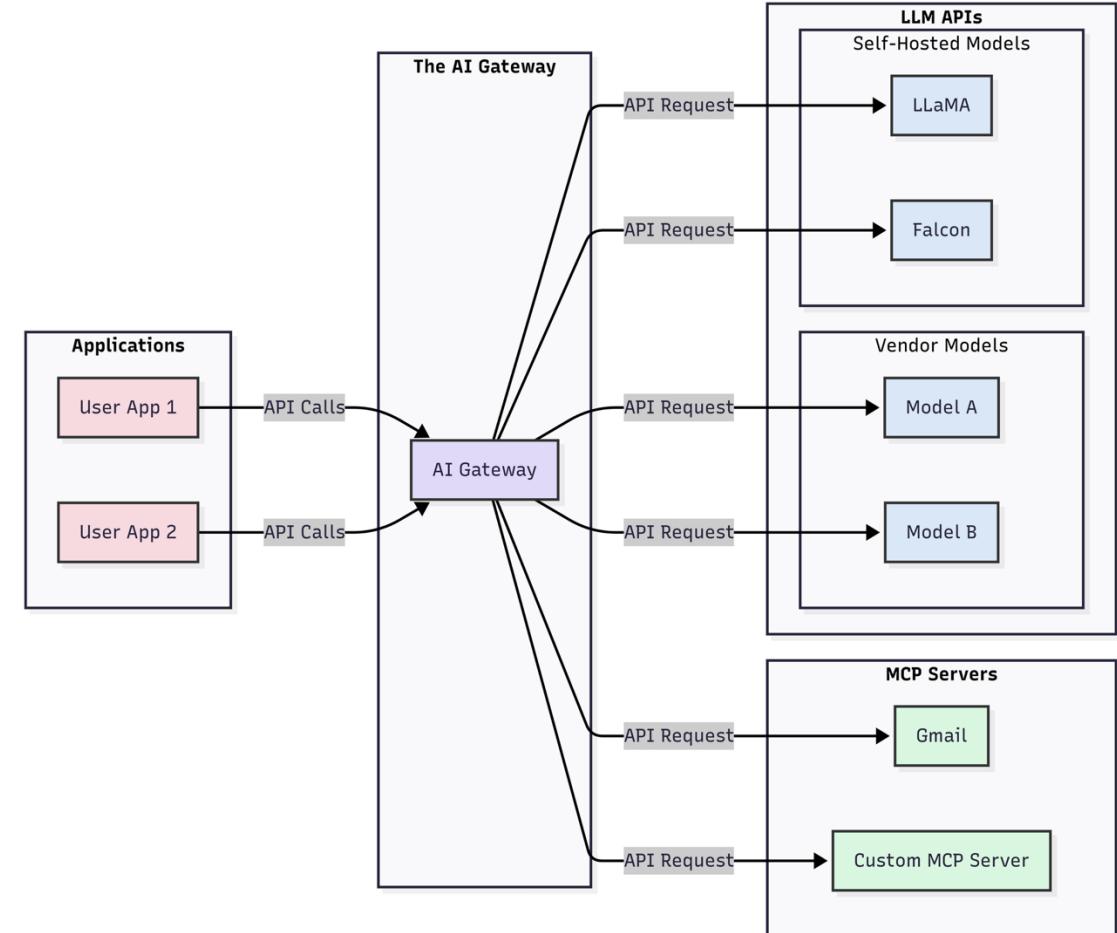
Guardrails

- PII Risks
- Bias/Fairness policies

Cost and Usage Monitoring

- Monitor cost from one pane
- Complete prompt <> Response logging

RAG, Agents, Chatbots <Gateway> LLM Models, Tools, Agents





Hands-on Tutorial



Assessment (14%)

Day 5: Deployment and Evaluations



Module 8: Deployment with TrueFoundry



Hands-on Tutorial



Module 9: Evaluating LLM Applications



Why Evaluate at All?

Evaluation helps address and identify the bottlenecks related to:

- Hallucinations (*absence of relevant context in final prompt is the most common reason for this, but there could be others*)
- Reduced reasoning capabilities (*if all the relevant context is not provided to LLM, it can lead to this*)
- Unexpected latencies (*what would happen if the retrieval might not be efficient leading to high latencies*)
- Explainability of outputs (*i.e., was performance bad due to the LLM itself or due to other aspects of the application – think agents, RAG, agentic RAG etc.*)

It also helps build trust among users especially for use in high-stakes industries like Finance, Healthcare etc.

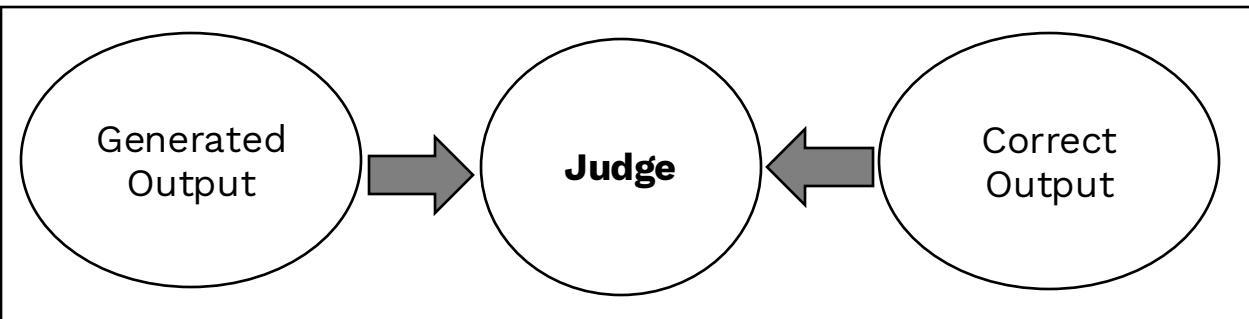
And in general, it is a best practice for building any kind of an “AI” application.

Evaluating a RAG Application

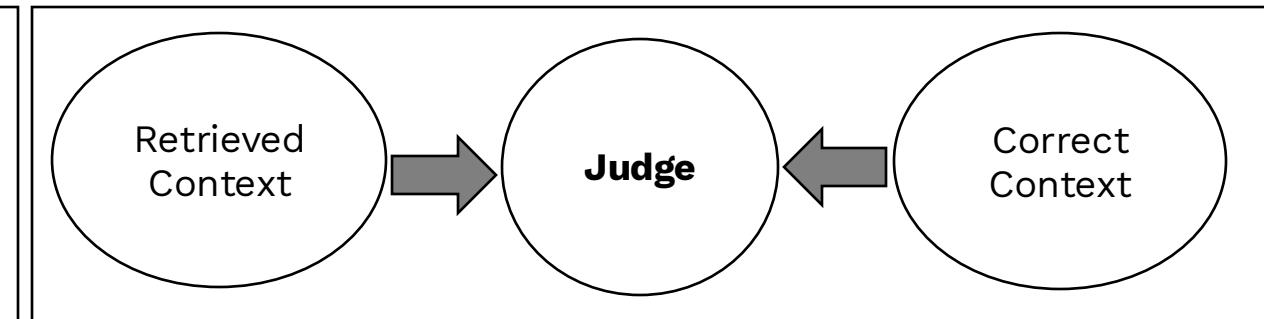
Typically we think of evaluating only the outputs. And if the result is poor, we might attribute it to the LLM itself.

However, remember such an LLM application is based off RAG, and instead of only focusing on “G” = Generation, **we need to also evaluate the “R” = Retrieval which is the other main pillar of such an application.**

Output vs. Output Evaluation



Context vs. Context Evaluation



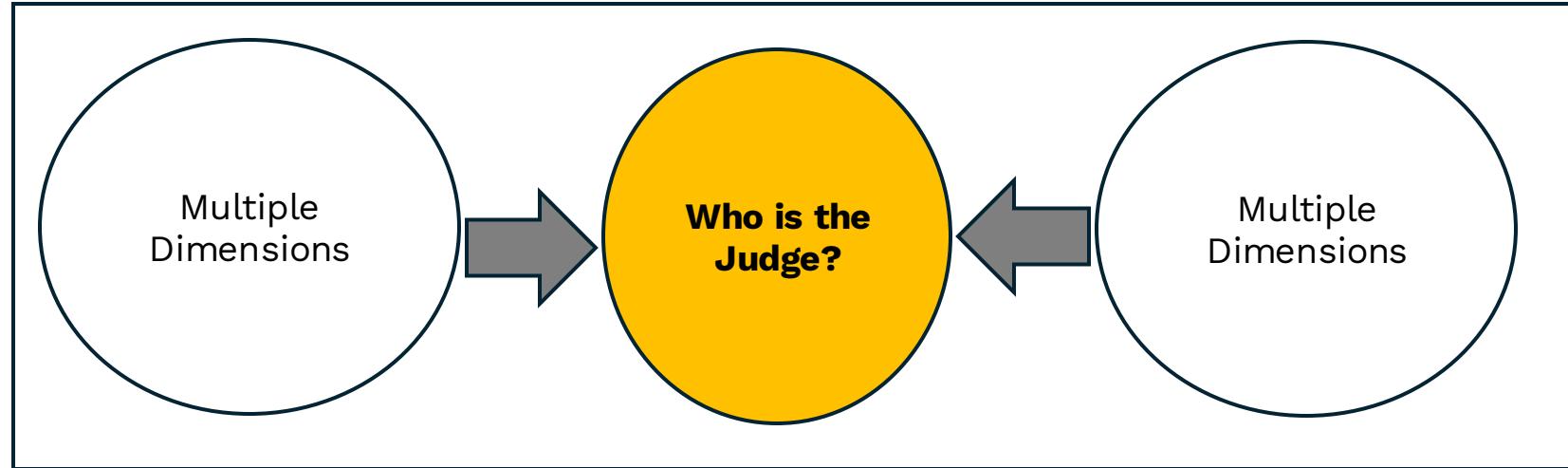
But even then, with the correct context, sometimes the LLM doesn't ground the output in the context leading to hallucinations!

SOLUTION: There are a lot of evaluation metrics that help address these varied issues.

LLM-as-a-Judge Paradigm

As we discussed, there are more dimensions than just *output vs. output evaluation*.

But HOW do we do this?



Judge can be

- 1) Human (**pros**: super good; **cons**: time-expensive, not scalable)
- 2) a Statistical NLP based method (**pros**: deterministic, scalable; **cons**: limited in semantic capabilities)
- 3) **an LLM itself! = LLM as a Judge!** (**pros**: good semantic capabilities, scalable; **cons**: scale comes with cost, black box/hallucination could potentially occur)

Overview of Evaluation Metrics

Now let's see an overview of evaluation metrics, and the dimensions each of these is designed to evaluate.

Dimensions						
S.No.	Metric	User Question	Ground Truth Context	Retrieved Context	Actual Answer	Generated Answer
Retrieval Metrics	1 Context Precision@k	✗	✓	✓	✗(✓)	✗
	2 Context Recall	✗	✓	✓	✗(✓)	✗
Generation Metrics	3 Faithfulness	✗	✗	✓	✗	✓
	4 Answer Relevancy	✓	✗	✗	✗	✓
5	Prompt Alignment	✓	✗	✗	✗	✓
6	Toxicity	✗	✗	✗	✗	✓

We'll deep-dive into S.No. 1-6 using the popular frameworks **RAGAs** and **DeepEval** in Python.



Evaluation Metric 1: Context Recall

Key Takeaway

Out of the relevant available context, how much relevant context is retrieved?
(can be done using actual answer also, instead of ground truth context)

Definition

$$\text{Context Recall} = \frac{\text{Number of claims in the reference supported by the context}}{\text{Total number of claims in the reference}}$$

Code Snippet (RAGAs)

```
● ● ●  
1 sample = SingleTurnSample(  
2     user_input="What are the interest rates of loans for the new vehicles?",  
3     response="Interest rates for new vehicle's loans are expected to be around 6.7% on average",  
4     reference="Interest rates for new vehicle's loans are expected to be around 6.7% on average",  
5     retrieved_contexts=[ "Average interest rates for new-vehicle loans are expected to be 6.7%, down 16 basis points from a year ago.",  
6                           "Average monthly finance payments this month are on pace to be $734, up $11 from January 2024"]  
7 )  
8  
9 await context_recall.single_turn_ascore(sample)
```

Output

1.0

Relative Cost of Evaluation

Small to medium depending on system prompt length

Evaluation Metric 2: Context Precision@k

Key Takeaway

Out of the all context that's retrieved, how much context is relevant and ranked higher than irrelevant ones?

(can be done using actual answer also, instead of ground truth context)

User Question	Ground Truth Context	Retrieved Context	Actual Answer	Generated Answer
✗	✓	✓	✗ (✓)	✗

Definition

$$\text{Context Precision}@k = \frac{\sum_{k=1}^K (\text{Precision}@k \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

Code Snippet (RAGAs)

```
● ● ●
1 sample = SingleTurnSample(
2     user_input="What are the interest rates of loans for the new vehicles?",
3     response="Interest rates for new vehicle's loans are expected to be around 6.7% on average",
4     retrieved_contexts=[ "Average monthly finance payments this month are on pace to be $734, up $11 from January 2024",
5                         "Average interest rates for new-vehicle loans are expected to be 6.7%, down 16 basis points from a year ago." ]
6 )
7
8 await context_precision.single_turn_ascore(sample)
```

Output

0.5

Relative Cost of Evaluation

Small to medium depending on system prompt length

Evaluation Metric 3: Faithfulness

Key Takeaway

How much of the generated answer is attributed to the retrieved context, versus, the model's own knowledge base?

User Question	Ground Truth Context	Retrieved Context	Actual Answer	Generated Answer
✗	✗	✓	✗	✓

Definition

$$\text{Faithfulness Score} = \frac{\text{Number of claims in the response supported by the retrieved context}}{\text{Total number of claims in the response}}$$

Code Snippet (RAGAs)

```
● ● ●
1 sample = SingleTurnSample(
2     user_input="What are the interest rates of loans for the new vehicles?",
3     response="Interest rates for new vehicle's loans are expected to be around 6.7% on average",
4     retrieved_contexts=[ "Average monthly finance payments this month are on pace to be $734, up $11 from January 2024",
5                          "Average interest rates for new-vehicle loans are expected to be 6.7%, down 16 basis points from a year ago."]
6 )
7
8 await faithfulness.single_turn_ascore(sample)
```

Output

1.0

Relative Cost of Evaluation

Small to medium depending on system prompt length

Evaluation Metric 4: Answer Relevancy

Key Takeaway

How relevant is the response generated by the LLM, compared to what the user had originally asked?

User Question	Ground Truth Context	Retrieved Context	Actual Answer	Generated Answer
✓	✗	✗	✗	✓

Definition

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^N \text{cosine similarity}(E_{g_i}, E_a)$$



$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^N \frac{E_{g_i} \cdot E_a}{\|E_{g_i}\| \|E_a\|}$$

by default, 3 questions (i.e., N = 3) are generated by the LLM but this parameter can be changed

Code Snippet (RAGAs)

```
● ● ●
1 sample = SingleTurnSample(
2     user_input="What are the interest rates of loans for the new vehicles?",
3     response="Interest rates are the rates charged by the financier for the loans borrowed by the customer.",
4     retrieved_contexts=[ "Average monthly finance payments this month are on pace to be $734, up $11 from January 2024",
5                         "Average interest rates for new-vehicle loans are expected to be 6.7%, down 16 basis points from a year ago."]
6 )
7
8 await relevancy.single_turn_ascore(sample)
```

Output

0.56

Relative Cost of Evaluation

Medium to large as multiple LLM calls are involved

Evaluation Metric 5: Prompt Alignment

Key Takeaway

How well did the generated answer follow the instructions mentioned in the prompt?

User Question	Ground Truth Context	Retrieved Context	Actual Answer	Generated Answer
✓	✗	✗	✗	✓

Definition

$$\text{Prompt Alignment} = \frac{\text{Number of instructions followed}}{\text{Total number of instructions}}$$

Code Snippet (DeepEval)

```
● ● ●
1 test_case = LLMTestCase(
2     input="What are the interest rates of loans for the new vehicles?",
3     actual_output="Interest rates for new vehicle's loans are expected to be around 6.7% on average.",
4     context=[("Average monthly finance payments this month are on pace to be $734, up $11 from January 2024",
5               "Average interest rates for new-vehicle loans are expected to be 6.7%, down 16 basis points from a year ago."),
6   )
7 metric = PromptAlignmentMetric(prompt_instructions=["You are an AI assistant that helps with answering user's question according to the context given."],
8                                 model=groq_llm)
9
10 output = evaluate([test_case], metrics=[metric], show_indicator=False, print_results=False)
```

Output

1.0

Relative Cost of Evaluation

Medium to large as multiple LLM calls are involved

- By default it's 2 (1. Checks the no. of instructions which were followed in the generated answer
 2. Generates a reason for the score calculated given the verdict for the opinions for the instructions which were not followed)



Evaluation Metric 6: Toxicity

Key Takeaway

Measures how toxic the generated response is, in terms of number of opinions.

User Question	Ground Truth Context	Retrieved Context	Actual Answer	Generated Answer
✗	✗	✗	✗	✓

Definition

$$\text{Toxicity} = \frac{\text{Number of toxic opinions}}{\text{Total number of opinions}}$$

Code Snippet (DeepEval)

```
● ● ●  
1 test_case = LLMTestCase(  
2     input="What are the interest rates of loans for the new vehicles?",  
3     actual_output="The interest rate is 6.7%, which is down 16 pathetic basis points",  
4     context=[ "Average monthly finance payments this month are on pace to be $734, up $11 from January 2024",  
5             "Average interest rates for new-vehicle loans are expected to be 6.7%, down 16 basis points from a year ago."]  
6 )  
7 metric = ToxicityMetric(threshold=0.1, model=groq_llm)  
8  
9 output = evaluate([test_case], metrics=[metric], show_indicator=False, print_results=False)
```

Output

1.0

Relative Cost of Evaluation

Medium to large as multiple LLM calls are involved

By default it's 3 (1. Call to generate all the opinions in the generated answer

- 2. Checks which opinions are toxic*
- 3. Generates a reason for the score calculated given the toxic opinions)*



Assessment (14%)

Certification Test (30%)