Developing Agentic Applications Workshop: Multi-Agent Applications

Henry Ruiz, PhD.
Research Scientist
Texas A&M University
GDE AI & GCP
@devharuiz
https://haruiz.github.io/

Juan Guillermo, M.Sc.
Tech Lead @ WordBox
GDE AI & GCP & Firebase
@jggomezt
https://devhack.co

Agenda

- Recap from previous section
- Emerging Capabilities of LLMs (Reasoning, Planning, etc.)
- Controlling and Aligning LLMs (State-of-the-Art Post-Pre Training Techniques)
- From Chat Applications to Agents
- Anatomy of an agentic application
- Frameworks for Agent Development
- Multi Agent Applications Patterns

GCP Credits

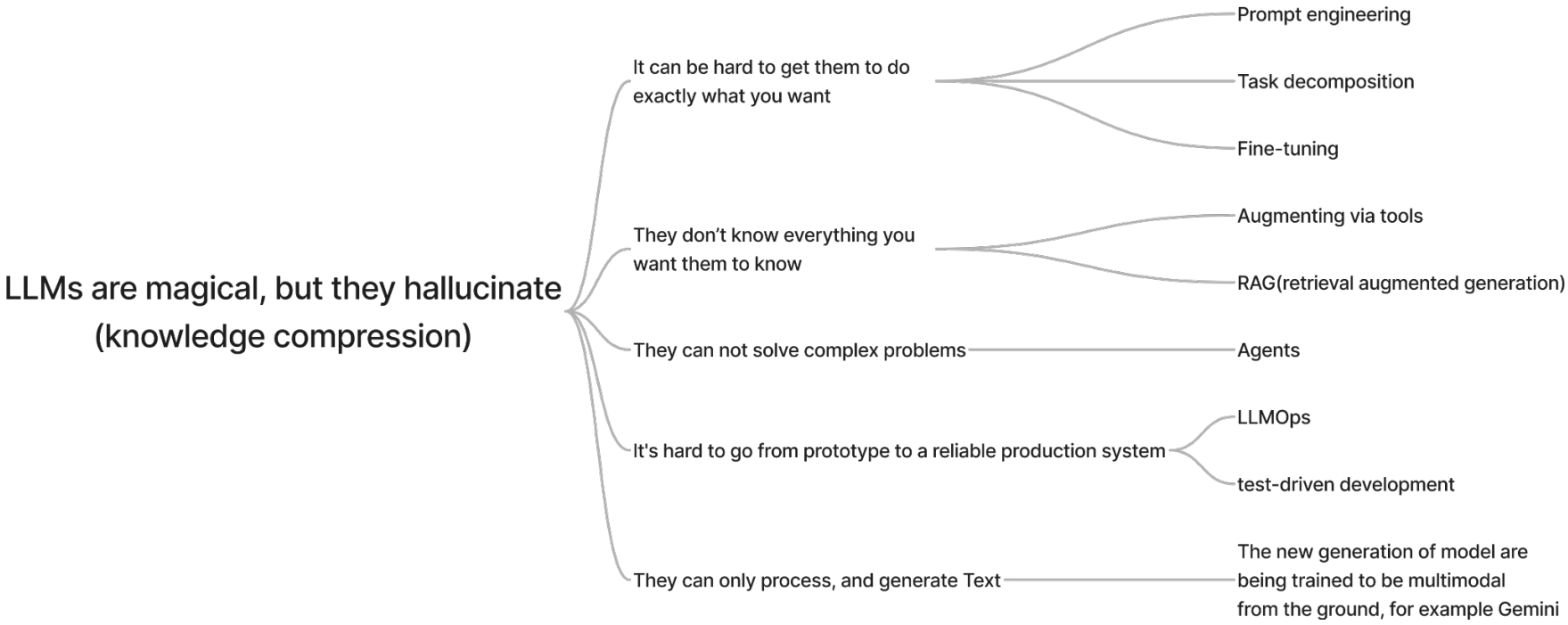
Experts

https://trygcp.dev/e/3p-intro-to-agents-01

From our Previous section....

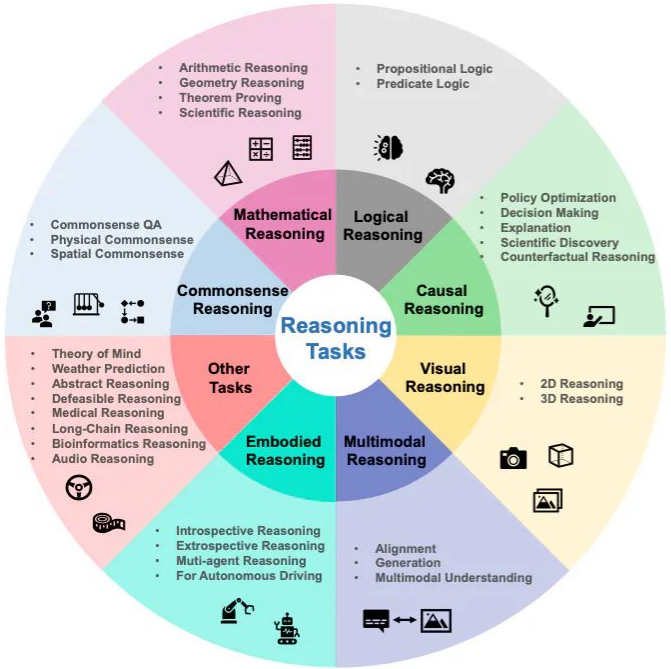
“agent” is not a discrete, 0 or 1 definition: instead, “agency” evolves on a continuous spectrum, as you give more or less power to the LLM on your workflow.

https://huggingface.co/docs/smolagents/conceptual_guides/intro_agents

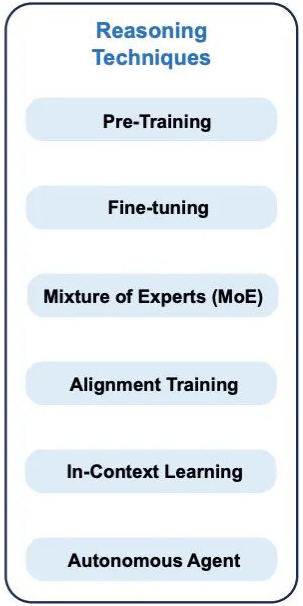

LLMs are magical, but they hallucinate (knowledge compression)
It can be hard to get them to do exactly what you want
Prompt engineering
Task decomposition
Fine-tuning
They don't know everything you want them to know
Augmenting via tools
RAG(retrieval augmented generation)
They can not solve complex problems
Agents
It's hard to go from prototype to a reliable production system
LLMOps
test-driven development
They can only process, and generate Text
The new generation of model are being trained to be multimodal from the ground, for example Gemini

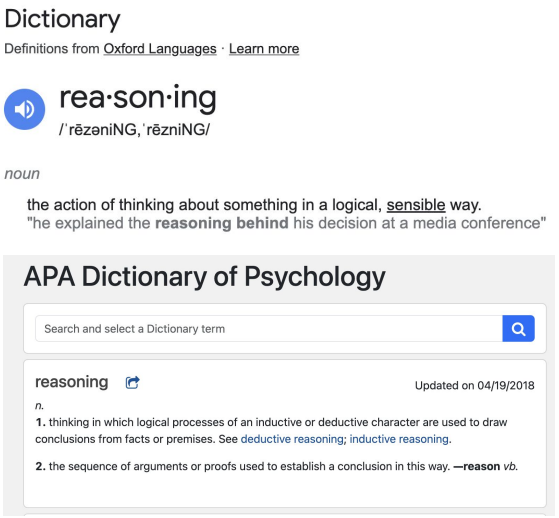
Recent advances in Large Language Models (LLMs) have demonstrated impressive reasoning capabilities.

LLMs can convert unstructured text into structured outputs, enabling the automation of a wide range of tasks across diverse domains



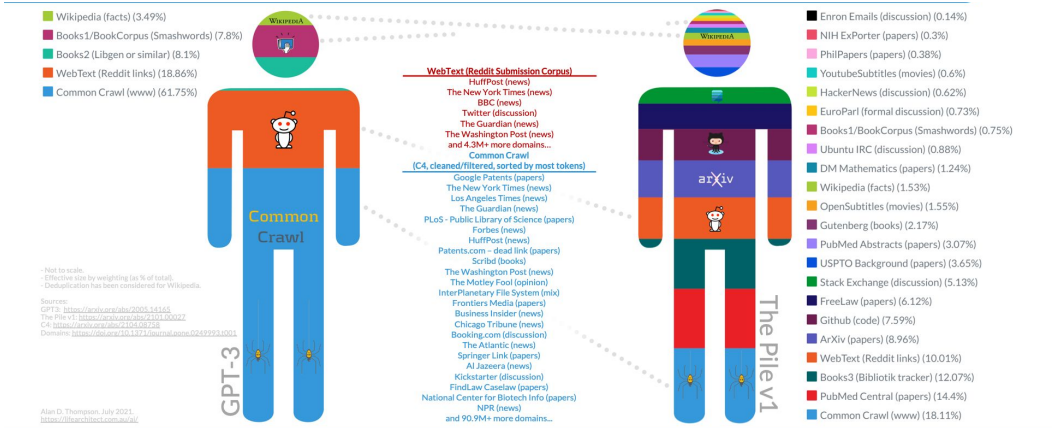
Support

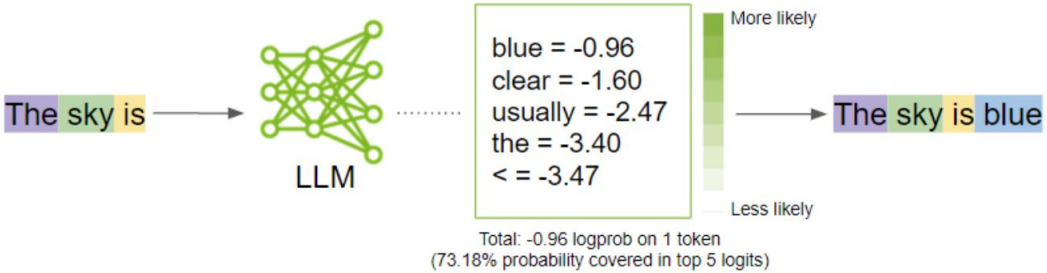


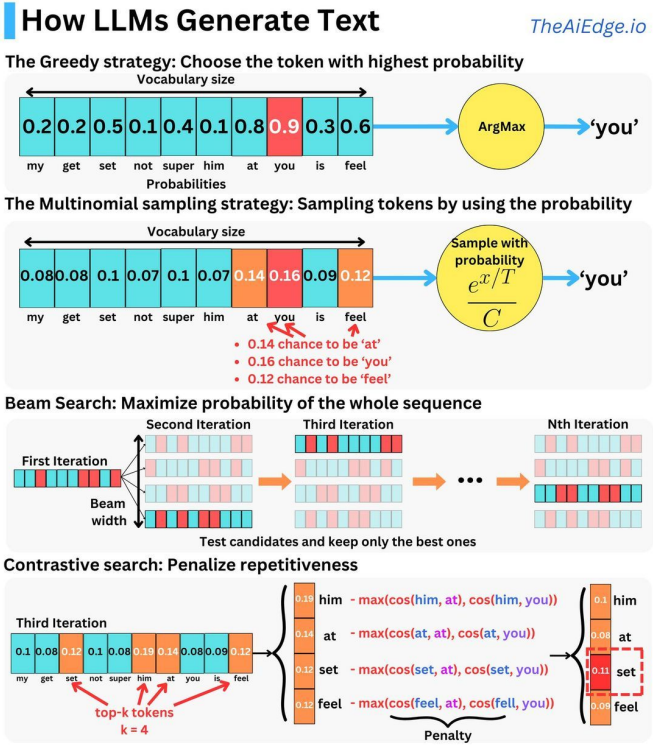


Zhang, Y., Mao, S., Ge, T., Wang, X., De Wynter, A., Xia, Y., Wu, W., Song, T., Lan, M., & Wei, F. (n.d.). A Survey of Strategic Reasoning with Large Language Models LLM AS A MASTERMIND: A SURVEY OF STRATEGIC REASONING WITH LARGE LANGUAGE MODELS. Retrieved July 26, 2024, from https://arxiv.org/pdf/2404.01230

How Do Large Language Models (LLMs) Work?

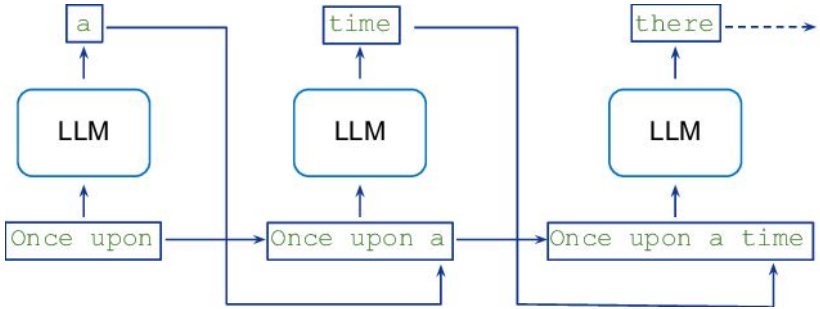






Credits to: https://newsletter.theaiedge.io/about

The autoregressive nature of LLMs suggests that they are capable of reasoning



Autoregressive (AR) language models:

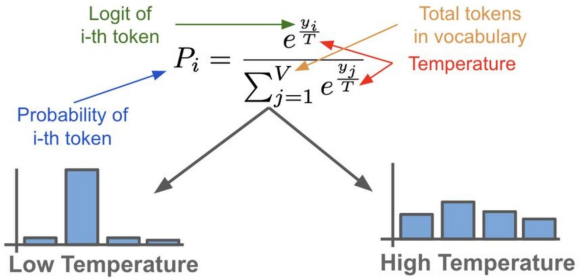
$$p(x_1, \ldots, x_L) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\ldots = \prod_i p(x_i|x_{1:i-1})$$

No approx: chain rule of probability

Stanford

=> You only need a model that can predict the next token given past context!

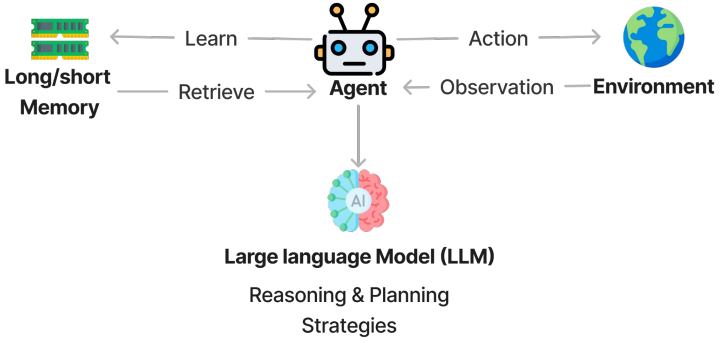
We can adjust the temperature to modulate the uniformity of the token distribution produced by the softmax transformation

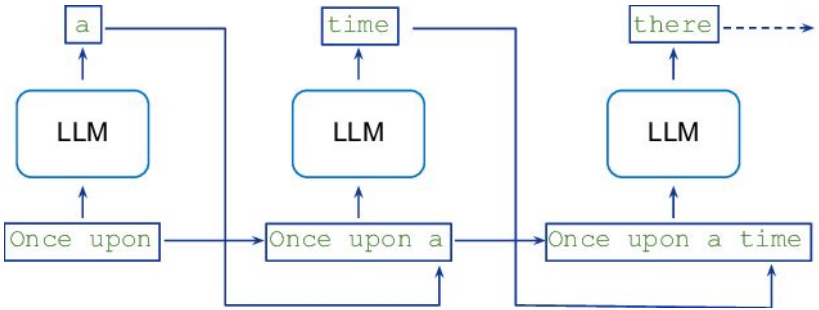


$$\arg\max \mathbb{P}(\text{final answer}|\text{problem})$$

$$= \sum_{\text{reasoning path}} \mathbb{P}(\text{reasoning path, final answer}|\text{problem})$$

Key idea of reasoning: Derive the final answer through intermediate steps & Post training alignment





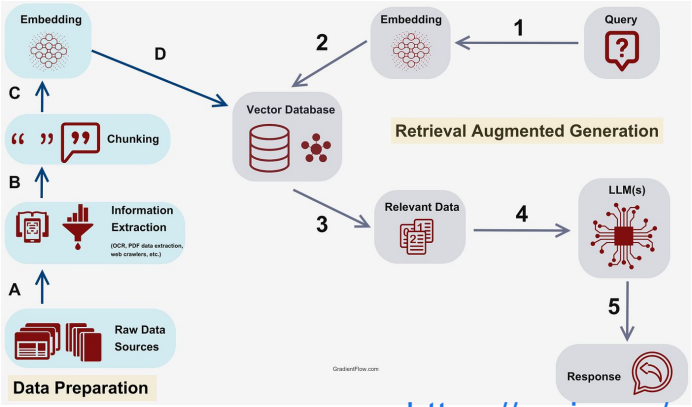
$$\arg\max \mathbb{P}(\text{final answer}|\text{problem})$$
$$= \sum_{\text{reasoning path}} \mathbb{P}(\text{reasoning path, final answer}|\text{problem})$$

Wei, Jason, et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." *ArXiv:2201.11903 [Cs]*, 10 Oct. 2022, arxiv.org/abs/2201.11903.

Yao, Shunyu, et al. "ReAct: Synergizing Reasoning and Acting in Language Models." *ArXiv.org*, 9 Mar. 2023, arxiv.org/abs/2210.03629.

Ling, Wang, et al. "Program Induction by Rationale Generation : Learning to Solve and Explain Algebraic Word Problems." ArXiv:1705.04146 [Cs], 23 Oct. 2017, arxiv.org/abs/1705.04146.

Today, we can only certainly say that LLMs are universal approximators for retrieval tasks.This capability is supported by their contextual understanding and the autoregressive des



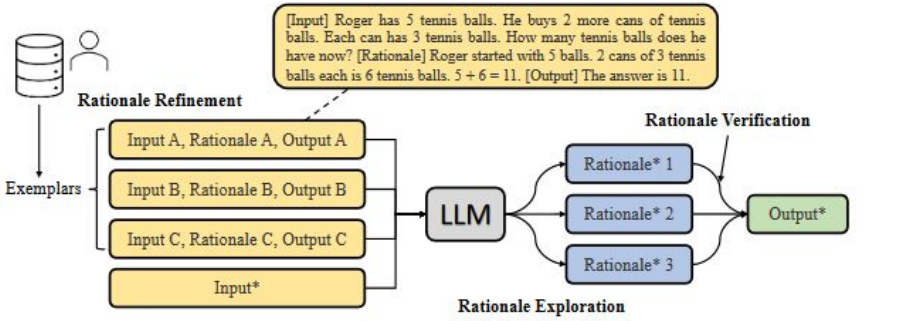


Figure 2: An illustration of *Chain-of-Thought Prompting* and *Rationale Engineering*, where asterisk (*) denotes the target problem to be solved.

https://arxiv.org/pdf/2310.07554

https://arxiv.org/pdf/2403.04121


GPT-4o
hallucinations
o1 hallucinations

These capabilities has revolutionized the way we interact with and utilize Large Language Models (LLMs) today.

Q: What is 1 + 2? →  → A: 3

Q:Which crop, cotton or soybeans, is better suited for the hot and dry climate of West Texas? →  → Requires knowledge

Retrieval augmented generation for knowledge (RAG)

Q: A farmer has 5 hectares of land and wants to apply 150 kg of fertilizer per hectare. How much total fertilizer does the farmer need? →  → Requires Computation

Code augmentation

Q: A farmer in Texas has two fields: Field A receives 30% more rainfall than Field B. If Field B receives 500 mm of rainfall per year, how much rainfall does Field A receive? →  → Requires reasoning

Tool use
(Nested calls)

What if both knowledge and reasoning are needed?

However, it is still a controversial claim

Towards Reasoning in Large Language Models: A Survey

Jie Huang **Kevin Chen-Chuan Chang**
Department of Computer Science, University of Illinois at Urbana-Champaign
{jeffhj, kcchang}@illinois.edu

However, despite the strong performance of LLMs on certain reasoning tasks, it remains unclear whether LLMs are actually reasoning and to what extent they are capable of reasoning. For example, Kojima et al. (2022) claim that "LLMs are decent zero-shot reasoners (p. 1)", while Valmeekam et al. (2022) conclude that "LLMs are still far from achieving acceptable performance on common planning/reasoning tasks which pose no issues for humans to do (p. 2)." This limitation is also stated by Wei et al. (2022b):

https://arxiv.org/pdf/2403.04121

https://arxiv.org/abs/2212.10403

Can Large Language Models Reason and Plan?

Subbarao Kambhampati
School of Computing & Augmented Intelligence Arizona State University email: rao@asu.edu

(A version appears in the Annals of The New York Academy of Sciences:
https://nyaspubs.onlinelibrary.wiley.com/doi/10.1111/nyas.15125)

To summarize, nothing that I have read, verified, or done gives me any compelling reason to believe that LLMs do reasoning/planning, as normally understood. What they do instead, armed with web-scale training, is a form of universal approximate retrieval, which, as I have argued, can sometimes be mistaken for reasoning capabilities. LLMs do excel in idea generation for any task–including those involving reasoning, and as I pointed out, this can be effectively leveraged to support reasoning/planning in LLM-Modulo frameworks[6]. In other words, LLMs already have enough amazing approximate retrieval abilities that can be gainfully leveraged, that we don't need to ascribe questionable reasoning/planning capabilities to them.††

Study with infants suggests language not necessary for reasoning ability

by Bob Yirka , Medical Xpress

A team of researchers from Spain, Hungary and Poland has found via a study with infants that language may not be a necessity for the ability to reason. In their paper published in the journal *Science*, the group describes their study and what they found, and also offer some opinions on how their findings might be used to better understand the ability to reason. Justin Halberda with Johns Hopkins University offers a Perspective piece on the work done by the team in the same journal issue.

The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity

Parshin Shojaee*† Iman Mirzadeh* Keivan Alizadeh
Maxwell Horton Samy Bengio Mehrdad Farajtabar

Apple

Abstract

Recent generations of frontier language models have introduced Large Reasoning Models (LRMs) that generate detailed thinking processes before providing answers. While these models demonstrate improved performance on reasoning benchmarks, their fundamental capabilities, scaling properties, and limitations remain insufficiently understood. Current evaluations primarily focus on established mathematical and coding benchmarks, emphasizing final answer accuracy. However, this evaluation paradigm often suffers from data contamination and does not provide insights

The Illusion of the Illusion of Thinking

A Comment on Shojaee et al. (2025)

C. Opus* A. Lawsen†

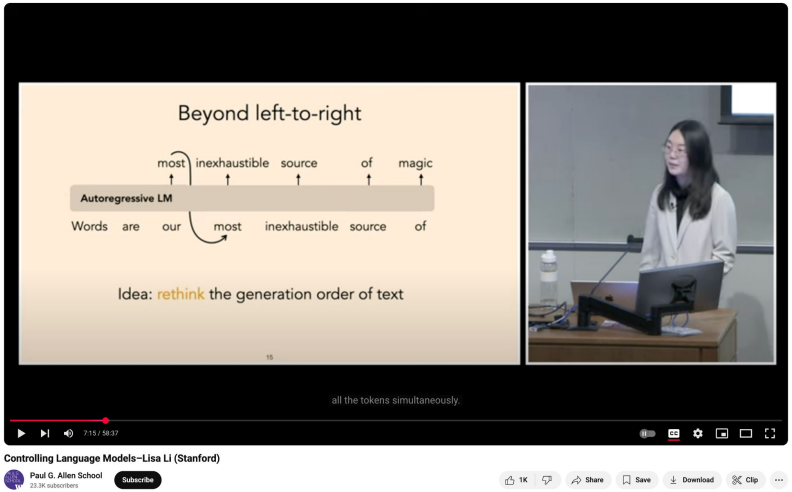
June 10, 2025

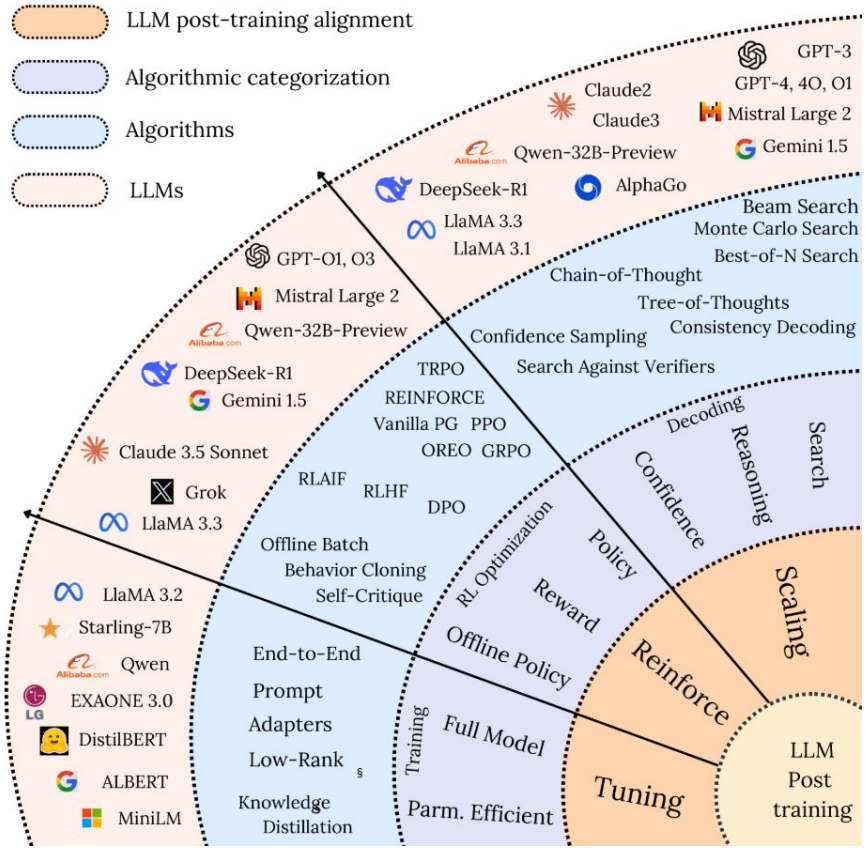
Abstract

Shojaee et al. (2025) report that Large Reasoning Models (LRMs) exhibit "accuracy collapse" on planning puzzles beyond certain complexity thresholds. We demonstrate that their findings primarily reflect experimental design limitations rather than fundamental reasoning failures. Our analysis reveals three critical issues: (1) Tower of Hanoi experiments systematically exceed model output token limits at reported failure points, with models explicitly acknowledging these constraints in their outputs; (2) The authors' automated evaluation framework fails to distinguish between reasoning failures and practical constraints, leading to misclassification of model capabilities; (3) Most concerningly, their River Crossing benchmarks include mathematically impossible instances for $N \geq 6$ due to insufficient boat capacity, yet models are scored as failures for not solving these unsolvable problems. When we control for these experimental artifacts, by requesting generating functions instead of exhaustive move lists, preliminary experiments across multiple models indicate high accuracy on Tower of Hanoi instances previously reported as complete failures. These findings highlight the importance of careful experimental

LLM Post-Training

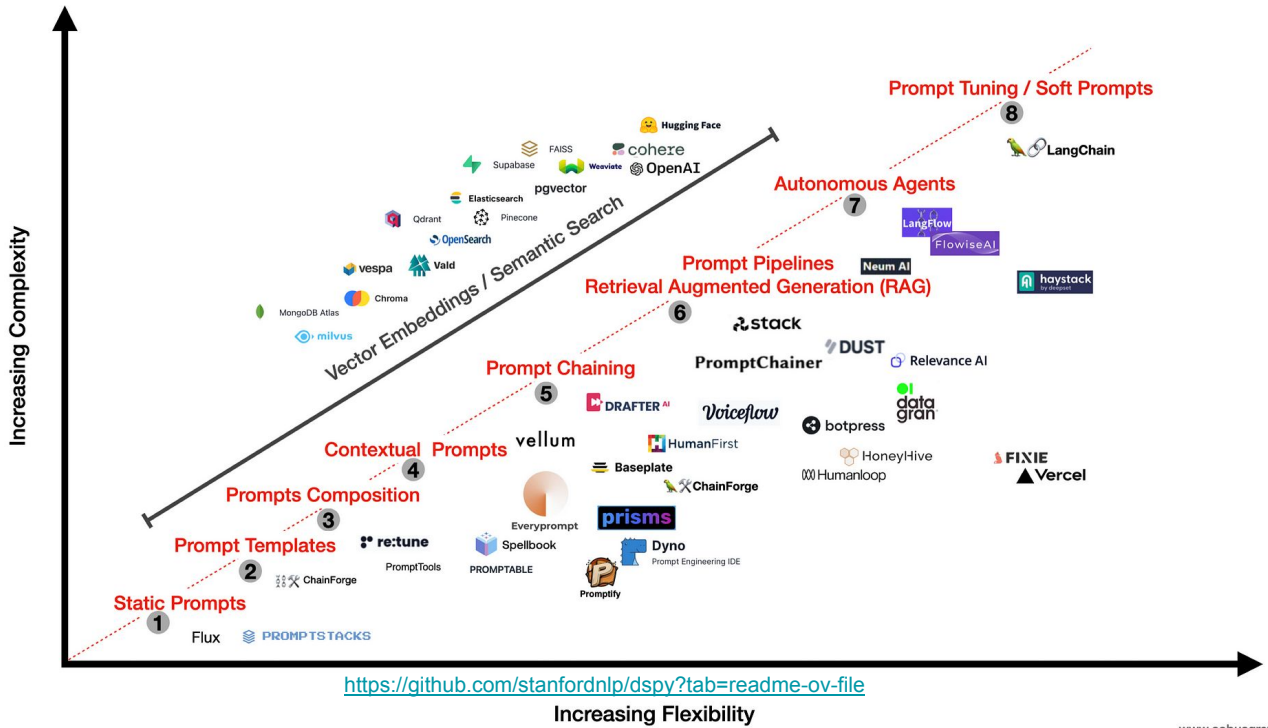
https://www.youtube.com/watch?v=tEQ9N5JjGW0&t=3204s





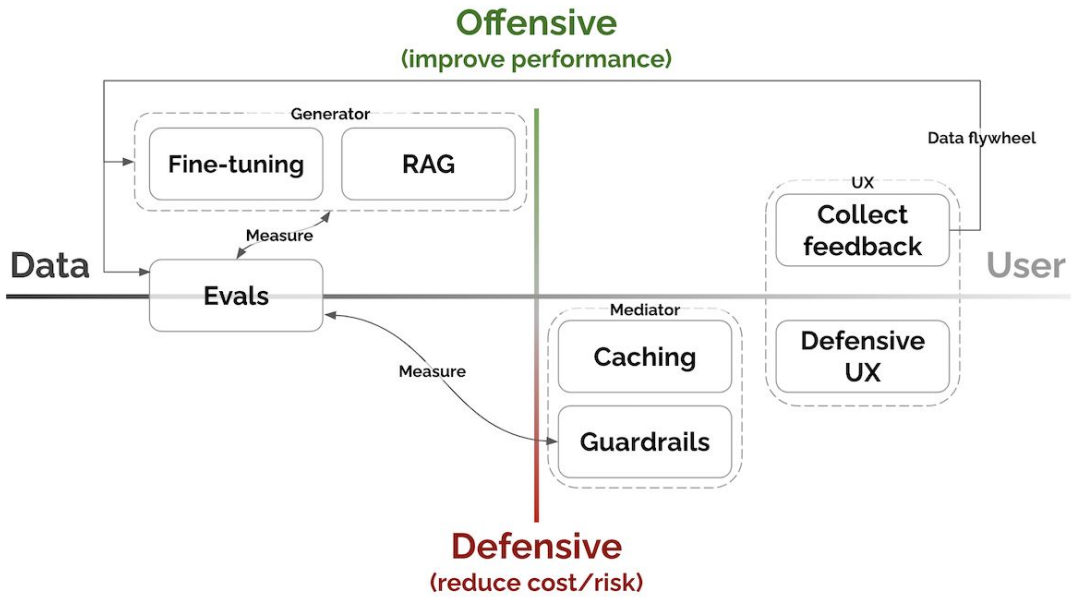
Kumar, K., Ashraf, T., Thawakar, O., Anwer, R. M., Cholakkal, H., Shah, M., Yang, M., Torr, P. H. S., Khan, F. S., & Khan, S. (2025, February 28). *LLM Post-Training: A Deep Dive into Reasoning Large Language Models*. arXiv.org. https://arxiv.org/abs/2502.21321v2

The challenges have opened the door to a new wave of emerging architectures and design patterns for developing LLM applications.



https://github.com/stanfordnlp/dspy?tab=readme-ov-file

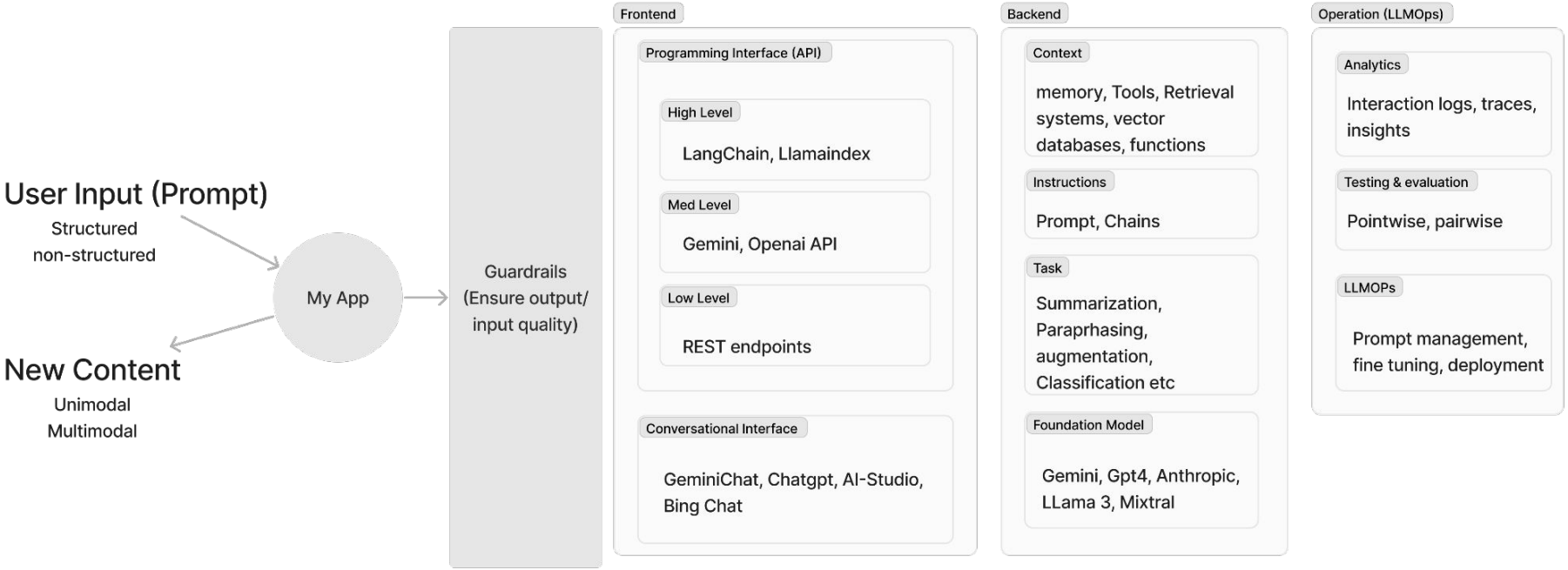
The GOAL: Improving LLMs performance



- Evals: To measure performance
- RAG: To add recent, external knowledge
- Fine-tuning: To get better at specific tasks
- Caching: To reduce latency & cost
- Guardrails: To ensure output quality
- Defensive UX: To anticipate & manage errors gracefully
- Collect user feedback: To build our data flywheel

https://eugeneyan.com/writing/llm-patterns/#evals-to-measure-performance

Anatomy of an LLM app



The key:
Memory = Context Window

Retrieve Anything To Augment Large Language Models

https://arxiv.org/pdf/2310.07554

Augmented language models

…In the following slides, we'll explore how taking full advantage of the expanded context window in LLMs unlocks capabilities that go far beyond simple memorization—enabling more sophisticated reasoning, task automation, and contextual understanding

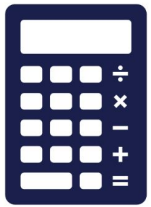
Retrieval

Augment with a bigger corpus

Chains

Augment with more LLM calls

Tools



Augment with outside sources

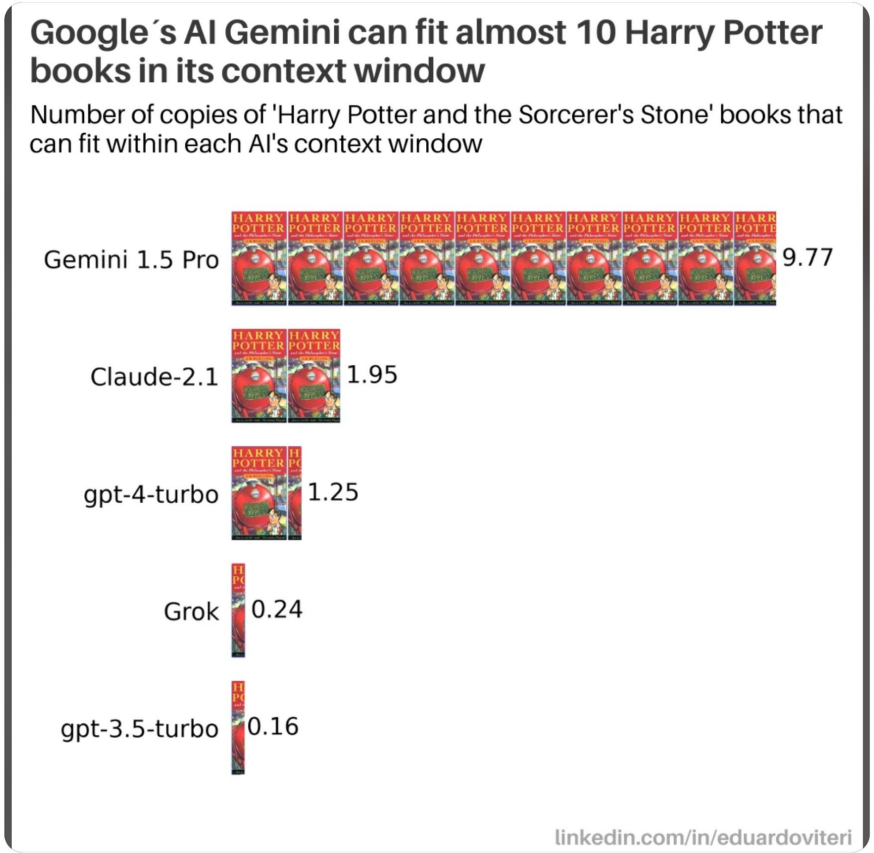
Context provides LLMs with unique, up-to-date information.

…But it only fits a limited amount of information

r/OpenAI · 6 mo. ago
veleros

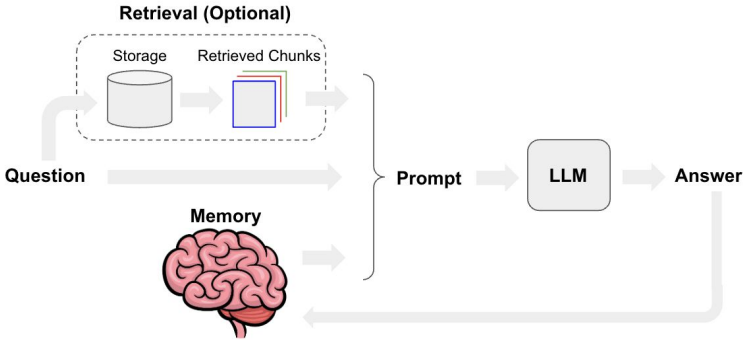
Gemini's context window is much larger than anyone else's

Image



Chat application

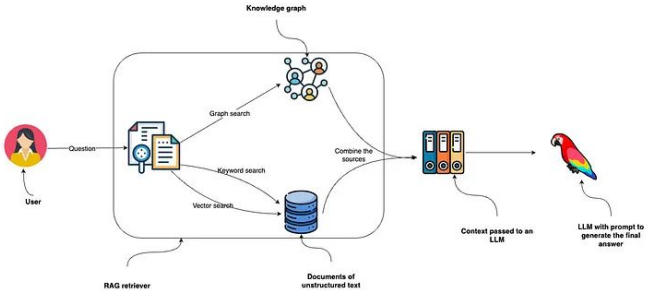
- The chat history is injected into the prompt.
- This provides the LLM (Large Language Model) with access to previous messages, referred to as "memory."
- The LLM uses these messages as context to generate its output.
- Alternative we can connect a chat app with a retrieval system




```python
def get_chat_history():
    """
    This function returns the chat history.
    :param history:
    :return:
    """
    return [
        HumanMessage("What is the capital of France?"),
        AIMessage("The capital of France is Paris."),
    ]

template = (
    "Given the following conversation:"
    "\nChat History:\n"
    "{history}"
    "\n, and a follow-up question:\n"
    "{question}\n"
    "respond to the follow-up question in english:"
)
prompt = ChatPromptTemplate.from_template(template)
llm = ChatGoogleGenerativeAI(model="gemini-pro")
chat_history = get_chat_history()

chat_context = RunnablePassthrough.assign(history=lambda _: chat_history)
chain = chat_context | prompt | llm | {"answer": StrOutputParser()}

response = chain.invoke({"question": "Can you respond again?"})
print(response)
```

RAG application (Vector index/ knowledge graph)

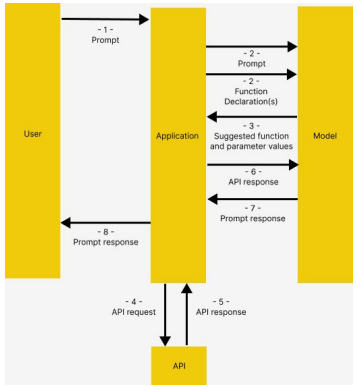
- Documents are transformed into embeddings.
- These embeddings are stored in an embedding database to create a vector index or a knowledge graph.
- When querying:
  - Relevant documents are retrieved from the vector index based on the prompt.
  - The LLM uses these documents as context to generate its output.




```python
def get_rag_retriever():
    """
    This function returns the context for the RAG model.
    :return:
    """
    loader = TextLoader("state_of_the_union.txt")
    documents = loader.load()
    text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
    docs = text_splitter.split_documents(documents)
    embeddings_model = GoogleGenerativeAIEmbeddings(
        model="models/embedding-001", task_type="retrieval_document"
    )
    db = FAISS.from_documents(docs, embeddings_model)
    print(db.index.ntotal)
    return db.as_retriever()


template = """Answer the question based only on the following context:
{context}
Question: {question}
"""
prompt = ChatPromptTemplate.from_template(template)
llm = GoogleGenerativeAI(model="gemini-pro")
retriever = get_rag_retriever()

chat_context = RunnablePassthrough.assign(context=itemgetter("question") | retriever)
chain = chat_context | prompt | llm | {"answer": StrOutputParser()}
response = chain.invoke({"question": "what the state of the union talk about?"})
# response = chain.invoke({ "question": "talk me about apples?"})
print(response)
```

Function Calling

- A list of functions defined:
  - The signature of each function, including its description and arguments, is injected into the prompt.
  - This context provides access to a list of functions for delegating data processing tasks.
- The model does not directly call the functions.
  - Instead, it provides a structured data output.
  - This output includes the name of a selected function and the arguments that the model proposes for the function to be called with.

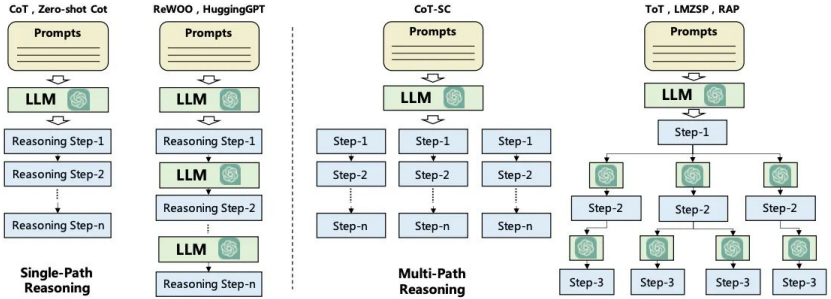



```python
"""
Basic example of using function calling with Gemini
:return:
"""


    # Create a model
def multiply(a: float, b: float):
  """returns a * b."""
  return a * b

def divide(a: float, b: float):
  """returns a / b."""
  return a / b

model = genai.GenerativeModel(
  model_name="gemini-pro",
  tools=[multiply,divide]
)
chat = model.start_chat(enable_automatic_function_calling=True)
response = chat.send_message(
    "I have 57 cats, each owns 44 mittens, how many mittens is that in total?"
)
console.print(response.text)
for content in chat.history:
    part = content.parts[0]
    print(content.role, "->", type(part).to_dict(part))
    print("-" * 80)
```

Agents

LLMs of today (e.g., Gemini, GPT4, LLAMA 3, etc.):

- When given access to tools (e.g., external memory,a calculator, web search, code execution, etc.),
- And context, Become agents that can **reason** and **act** on our behalf to solve tasks.



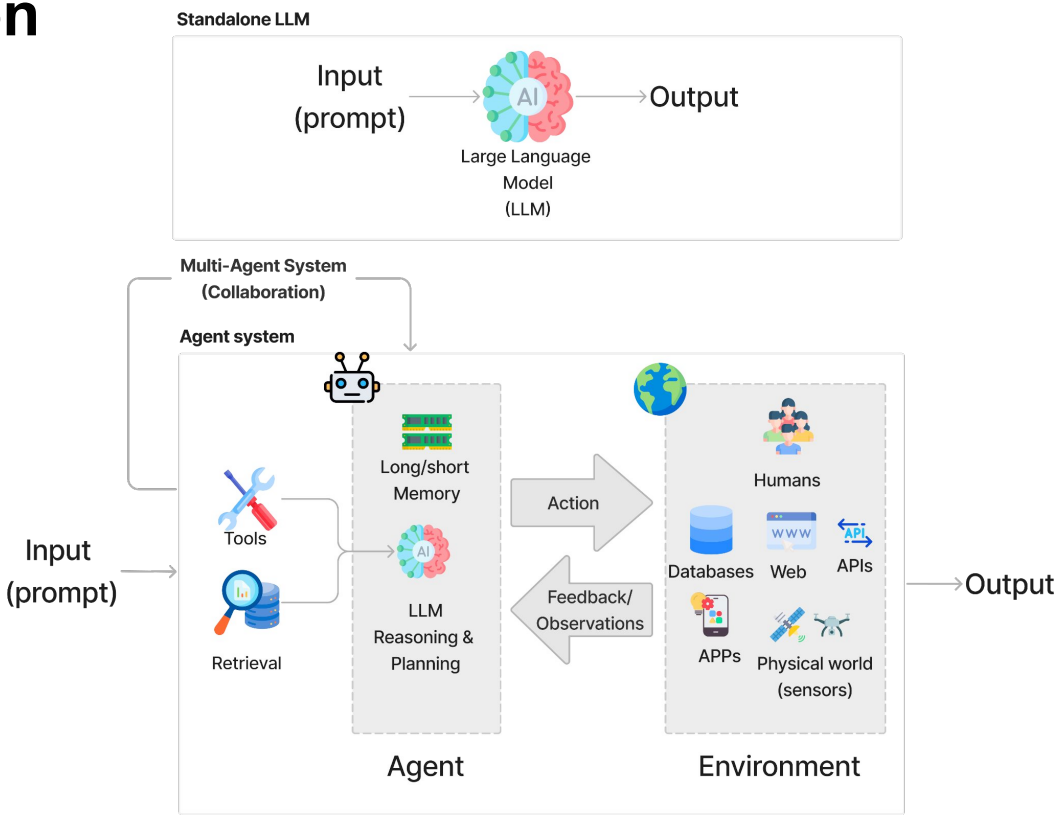
https://www.promptingguide.ai/research/llm-agents

```python
from dotenv import load_dotenv
from langchain import hub
from langchain.agents import (
    AgentExecutor,
    create_openai_tools_agent,
)
from langchain_openai import ChatOpenAI
from langchain_google_genai import ChatGoogleGenerativeAI
from tools import ImageCaptionTool, ObjectDetectionTool

load_dotenv()

tools = [ImageCaptionTool(), ObjectDetectionTool()]
# Get the instructions prompts for the agent
prompt = hub.pull("hwchase17/openai-tools-agent")
prompt.pretty_print()
# llm = ChatGoogleGenerativeAI(model="gemini-pro")
# llm = ChatOllama(model="llama2")
llm = ChatOpenAI(
    temperature=0,
    model_name="gpt-3.5-turbo"
)
# Construct the OpenAI Tools agent
agent = create_openai_tools_agent(llm, tools, prompt)
# Create an agent executor by passing in the agent and tools
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
response = agent_executor.invoke({
    "input": "describe what is in the image  images/test.jpg, and return the objects found",
})

print(response)
```

Agents (An unified solution for reasoning and Act)

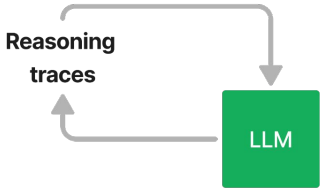
A verbal agent is a service that interacts with a **Large Language Model (LLM)** to perform goal-oriented tasks by leveraging available tools and contextual information.It relies on the LLM's natural language understanding capabilities to support reasoning, planning, and decision-making.

Agents <u>can autonomously execute complex tasks and solve multifaceted problems.</u>



CoT
(Chain-of-Thought)

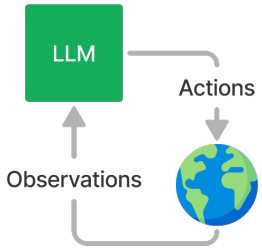
Reasoning (update internal state)



Lack of external knowledge and tools

RAG/Code/Tool use

Acting (obtain external feedback and knowledge)



Lack of reasoning (Retrieval, Search engine, Calculator, Python)

ReAct: A zero-shot emerging paradigm that introduces the concept of agents capable of both reasoning and taking actions

Large language Model (LLM)

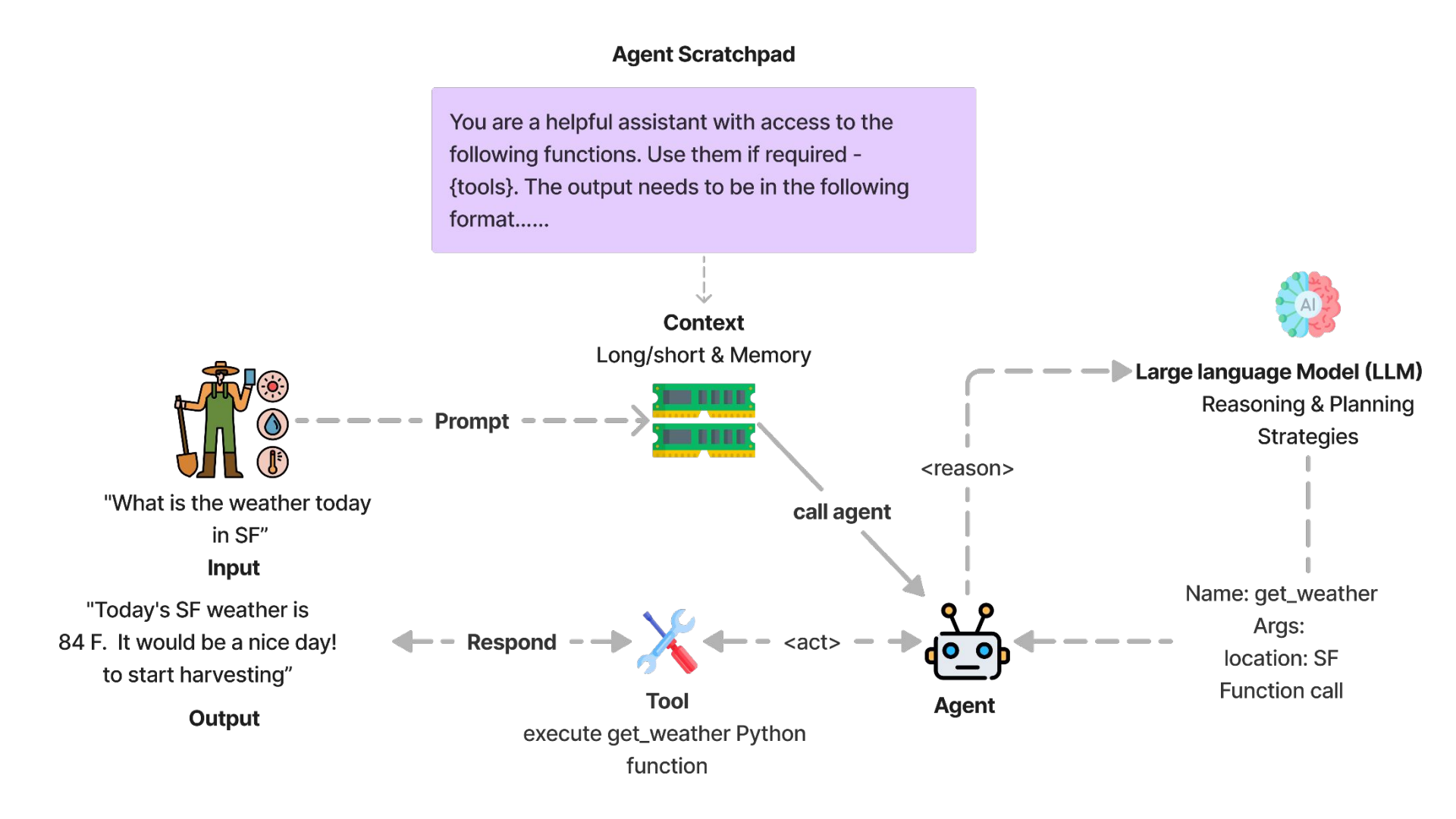
Reasoning

Agent

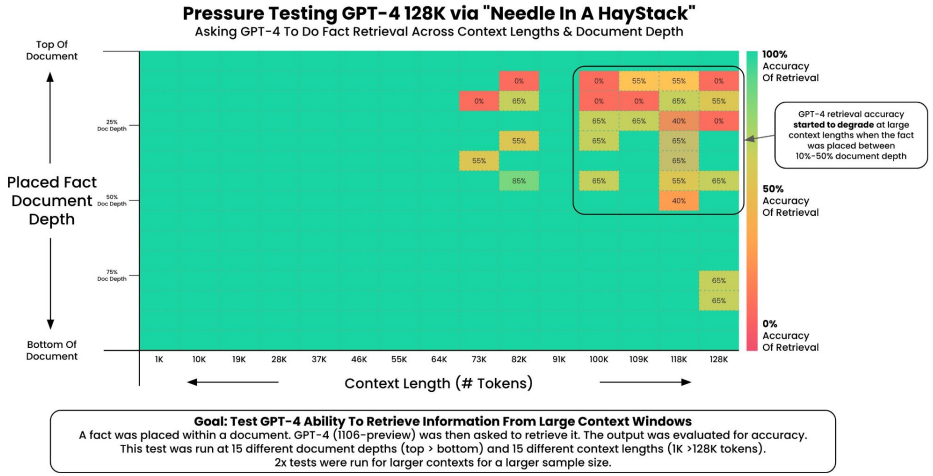
Action

Observation

Environment


Agent Scratchpad
You are a helpful assistant with access to the following functions. Use them if required - {tools}. The output needs to be in the following format.....
Context
Long/short & Memory
Large language Model (LLM)
Reasoning & Planning
Strategies
Prompt
"What is the weather today in SF"
Input
call agent
<reason>
"Today's SF weather is 84 F. It would be a nice day! to start harvesting"
Output
Respond
<act>
Tool
execute get_weather Python function
Agent
Name: get_weather
Args:
location: SF
Function call

Why do we need multi-agent architectures?



"Needle in a Haystack' test

. . . Interestingly, if the "needle" is positioned towards the beginning of the context, the model tends to overlook or "forget" it — whereas if it's placed towards the end or as the very first sentence, the model's performance remains solid....

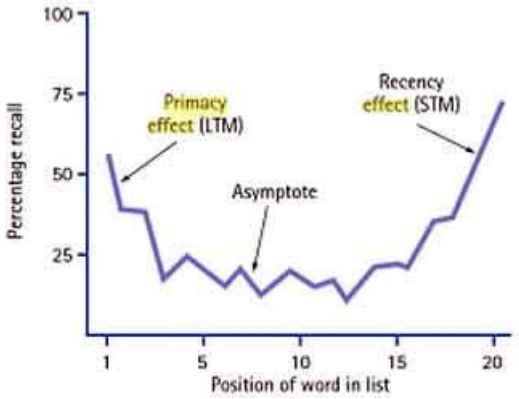
https://arxiv.org/pdf/2307.03172

https://arxiv.org/pdf/1906.04341

That phenomenon also occurred in humans.

The observation that LLMs tend to focus more on the beginning and end of a sequence has interesting parallels with biological systems, particularly in human cognition and memory. This effect can be likened to the **serial position effect** in psychology, which encompasses two phenomena:

1. **Primacy effect** – People tend to remember the first items in a sequence more easily.
2. **Recency effect** – People tend to recall the most recent (or last) items more effectively.

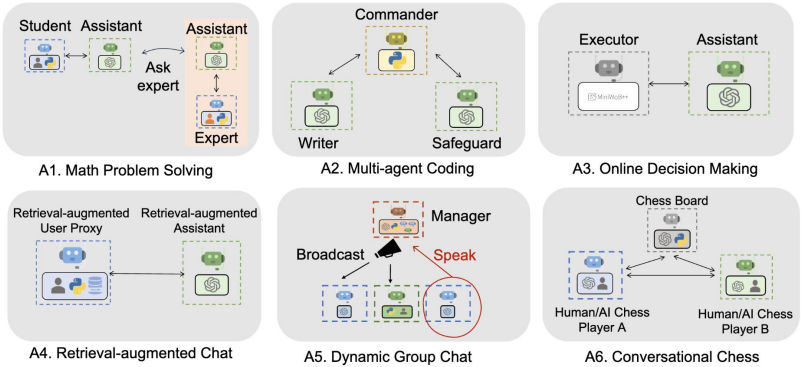


Murdock discovered that recall probability depends on a word's position in a list, with better recall for words at the beginning (primacy effect) and end (recency effect), while middle words are often forgotten.Murdock's experiment involved a total of 103 participants, all of whom were students from an introductory psychology course

Murdock, B. B., Jr. (1962). The serial position effect of free recall. *Journal of Experimental Psychology, 64*(5), 482–488. https://doi.org/10.1037/h0045106

Moving beyond single-agent systems to tackle complex tasks while mitigating and reducing hallucinations.

- A team of agents with diverse capabilities—language models and specialized tools—collaborating to solve complex tasks
- **Why?**
- Leverage a team of specialized agents (LLMs + tools)
- Group responsibilities to improve task success
- Focused agents outperform those managing many tools
- Use separate prompts for better performance
- Evaluate and improve agents independently without breaking the system



Agents collaboration patterns

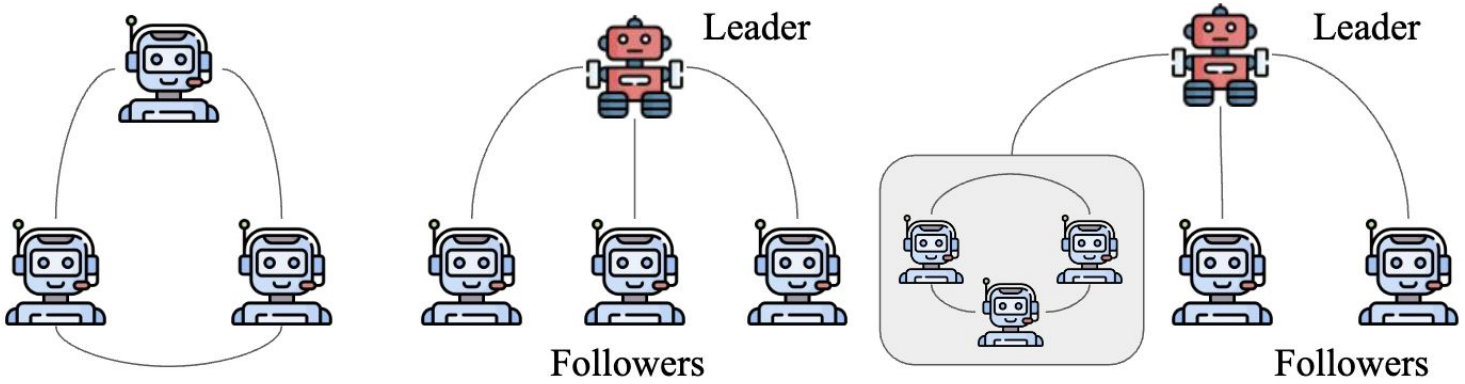


Figure 1. Structures of multi-agent systems.

Han, Shanshan, et al. “LLM Multi-Agent Systems: Challenges and Open Problems.” *ArXiv.org*, 5 Feb. 2024, arxiv.org/abs/2402.03578.

Building Effective Agents

"We've worked with dozens of teams building LLM agents across industries. Consistently, the most successful implementations use simple, composable patterns ***rather than complex frameworks.****"*

"Agent" can be defined in several ways. *Some customers define agents as 1)* ***fully autonomous systems that*** *operate independently over extended periods,* ***using various tools to accomplish complex tasks****. Others use the term to describe more 2)* ***prescriptive implementations that follow predefined workflows****. At Anthropic, we categorize all these variations as agentic systems, but draw an important architectural distinction between workflows and agents:*

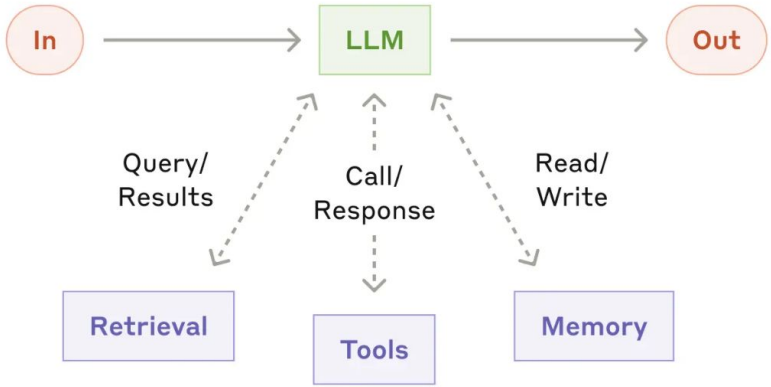
- ***Workflows*** *are systems where LLMs and tools are orchestrated through predefined code paths.*
- ***Agents****, on the other hand, are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks.*

– Anthropic

https://www.anthropic.com/engineering/building-effective-agents

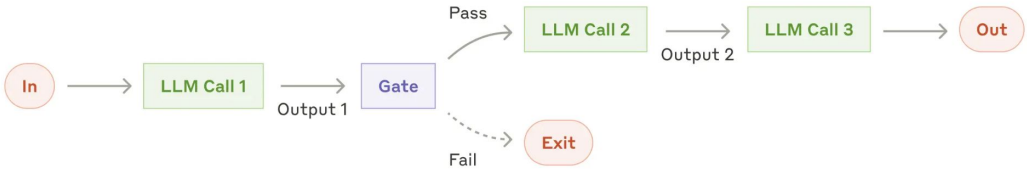
Building block: The augmented LLM

The basic building block of agentic systems is an LLM enhanced with augmentations such as retrieval, tools, and memory.

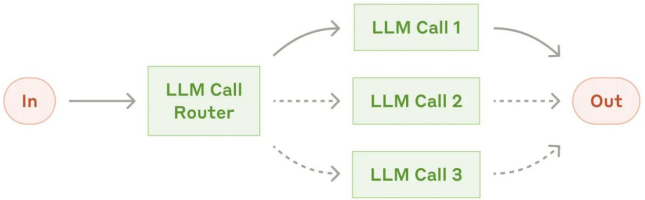


Workflows

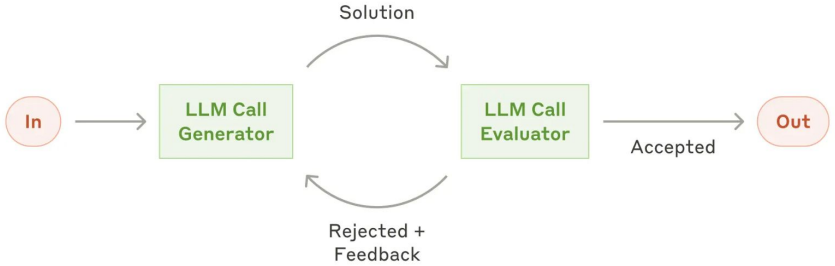
Workflow: Prompt chaining



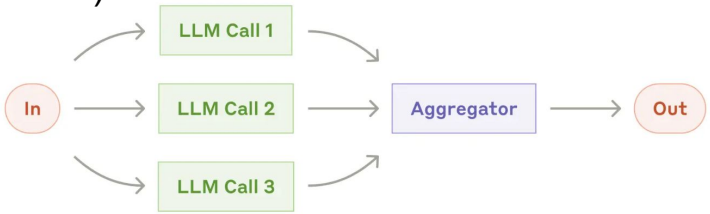
Workflow: Routing



Workflow: Evaluator-optimizer (reflection)

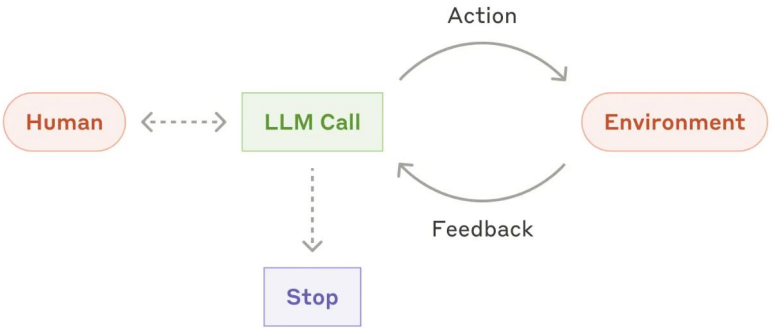


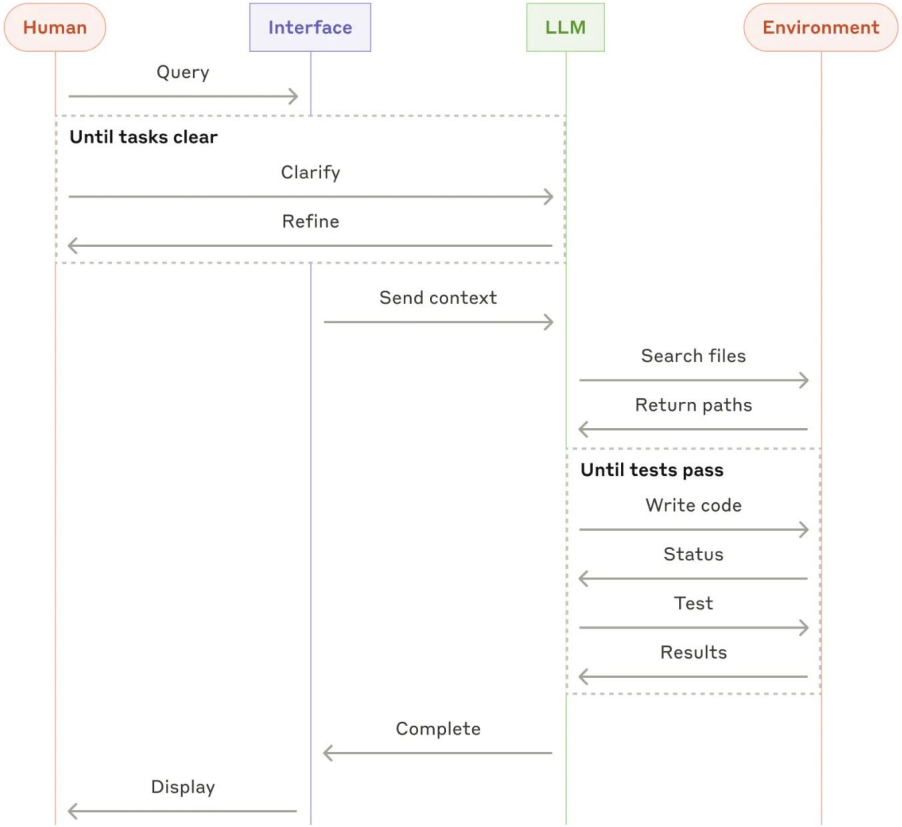
Workflow: Parallelization (Breaking a task into independent subtasks run in parallel & vote)



Workflow: Orchestrator-workers (Supervisor)

Agents


Action
Human
LLM Call
Environment
Feedback
Stop


Human
Interface
LLM
Environment
Query
Until tasks clear
Clarify
Refine
Send context
Search files
Return paths
Until tests pass
Write code
Status
Test
Results
Complete
Display

Building a Multi-Agent System - *from scratch*

- Goal definition
- Baseline
- Tools
- Eval testbed
- Agent

This comes last, not first!

https://newsletter.victordibia.com/p/4-ux-design-principles-for-multi?open=false#%C2%A7building-a-multi-agent-system-from-scratch

Frameworks provide different Levels of Abstraction

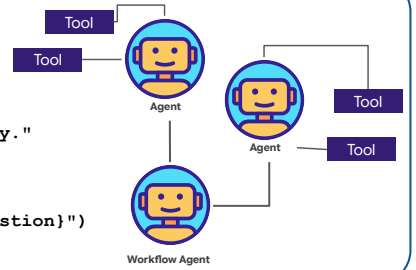
Agent framework
(Agent Development Kit)

Low-level orchestration framework
(LangGraph)

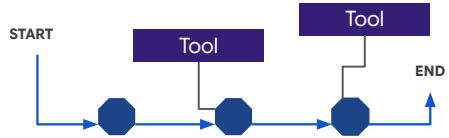
Low-level LLM framework
(Langchain)

DIY

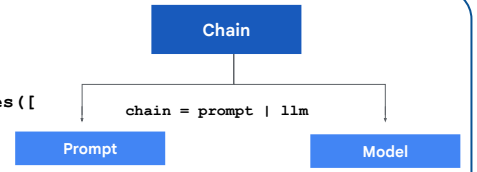
```
capital_finder_agent = LlmAgent(
        name="capital_finder_agent",
        model="gemini-1.5-flash-latest",
        instruction="You are a helpful assistant
that provides the capital city for a given country."
    )

    question = "What is the capital of France?"
    print(f"Calling ADK Agent with question: {question}")
```




```
workflow = StateGraph(GraphState)
workflow.add_node("llm_call", call_llm)
workflow.add_edge(START, "llm_call")
workflow.add_edge("llm_call", END)
```




```
    prompt = ChatPromptTemplate.from_messages([
        ("user", "{question}")
    ])
    chain = prompt | llm
    question = "What is the capital of France?"
    capital = chain.invoke({"question": question})
```



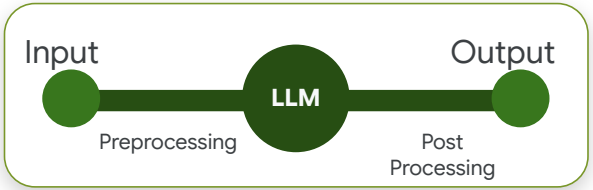

```
    client = OpenAI()

    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "user",
             "content": "What is the capital of France?"}
        ]
    )
```

Vendor Specific

Restful

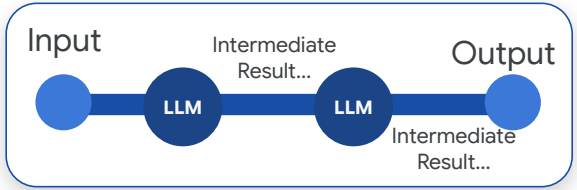
Architectural Patterns

Single Model Architecture



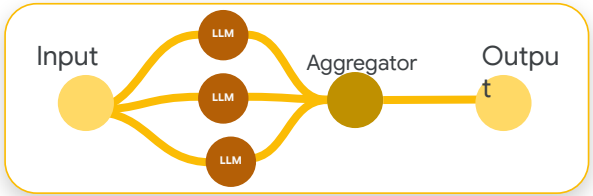
One LLM for all tasks

Sequential Models



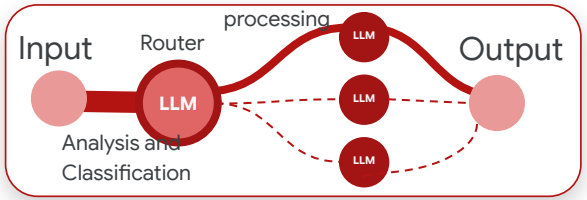
Chains multiple specialized LLMs in a pipeline.

Parallel Models



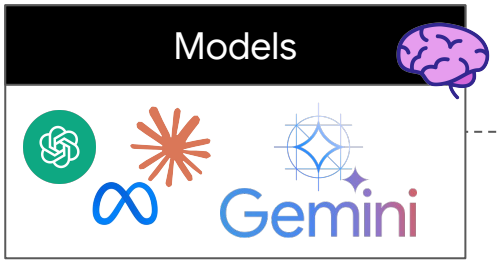
Multiple LLMs concurrently and aggregate outputs

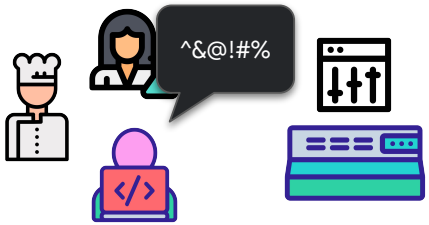
Hierarchical Routing

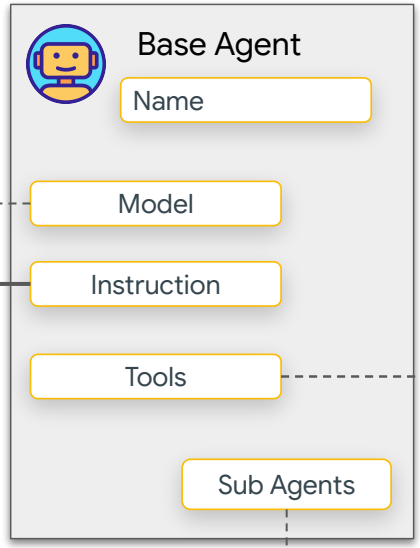


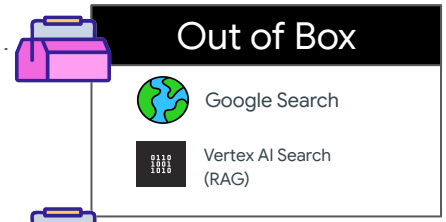
Router model to direct inputs to specialized LLMs based on content or intent

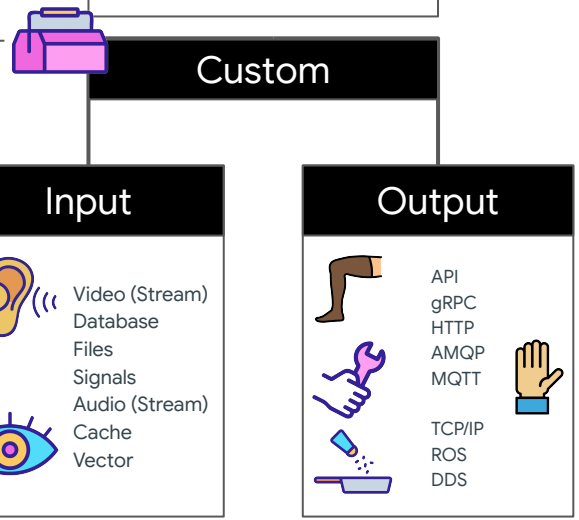
ADK Agent - Base

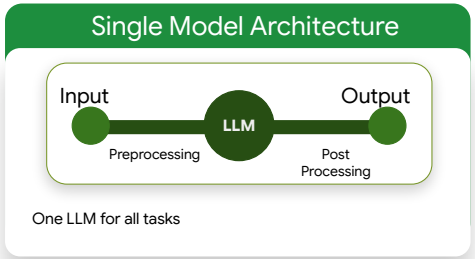


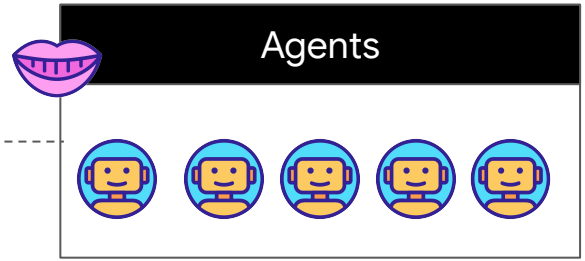






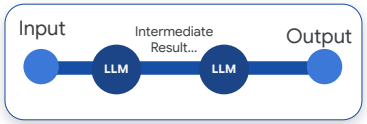






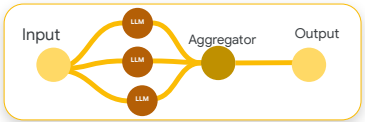
ADK Agent - Workflow

Sequential Models



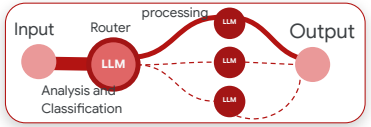
Chains multiple specialized LLMs in a pipeline.

Parallel Models

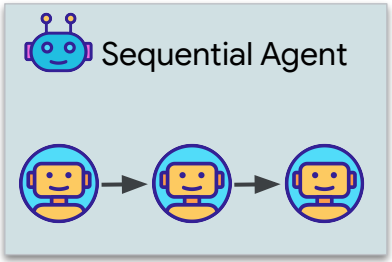


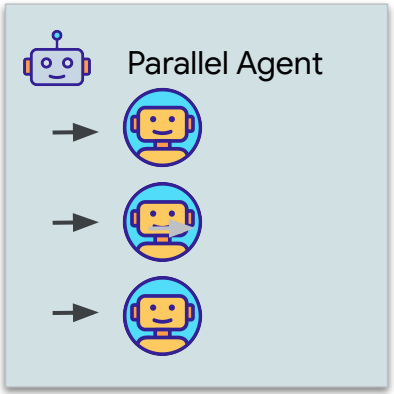
Multiple LLMs concurrently and aggregate outputs

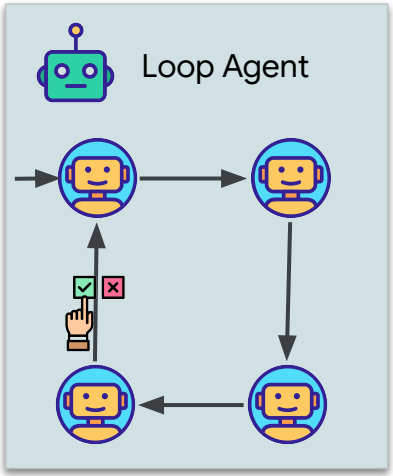
Hierarchical Routing

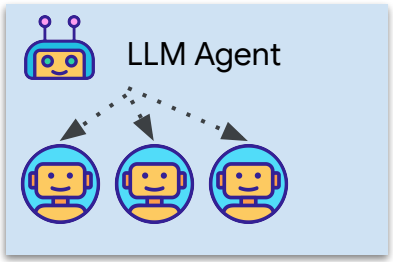


Router model to direct inputs to specialized LLMs based on content or intent





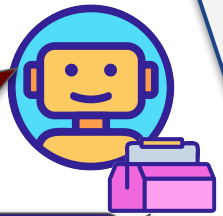




ADK - Memories

^&@!#%

Gemini



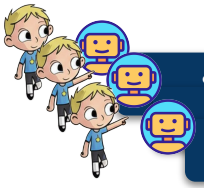
756798

&^)**%%

HGFKUY

384723

)($$%&*)



Session

Short Term

Events

^&@!#%

user messages

756798

agent response

&^)**%%

user messages

yiueue

call tool

yiueue

tool response

yiueue

state change

384723

agent response

)($$%&*)

user messages

State

key

value

abc

yiueue

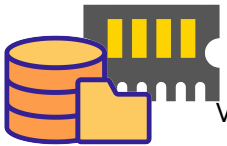
And artifacts...

Long/Short Term (Your Call) - Memory

Session

Session

Session

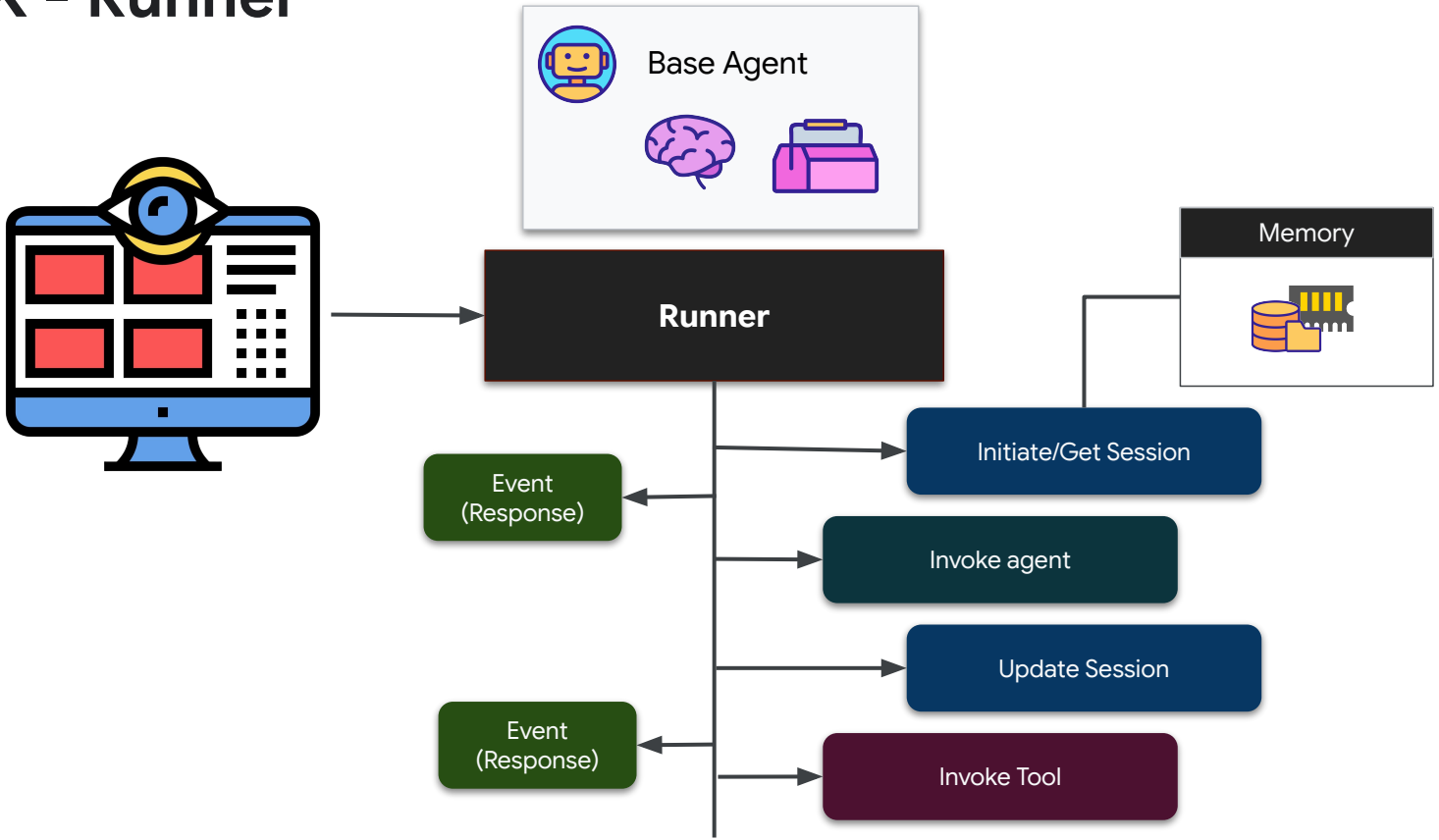


In memory

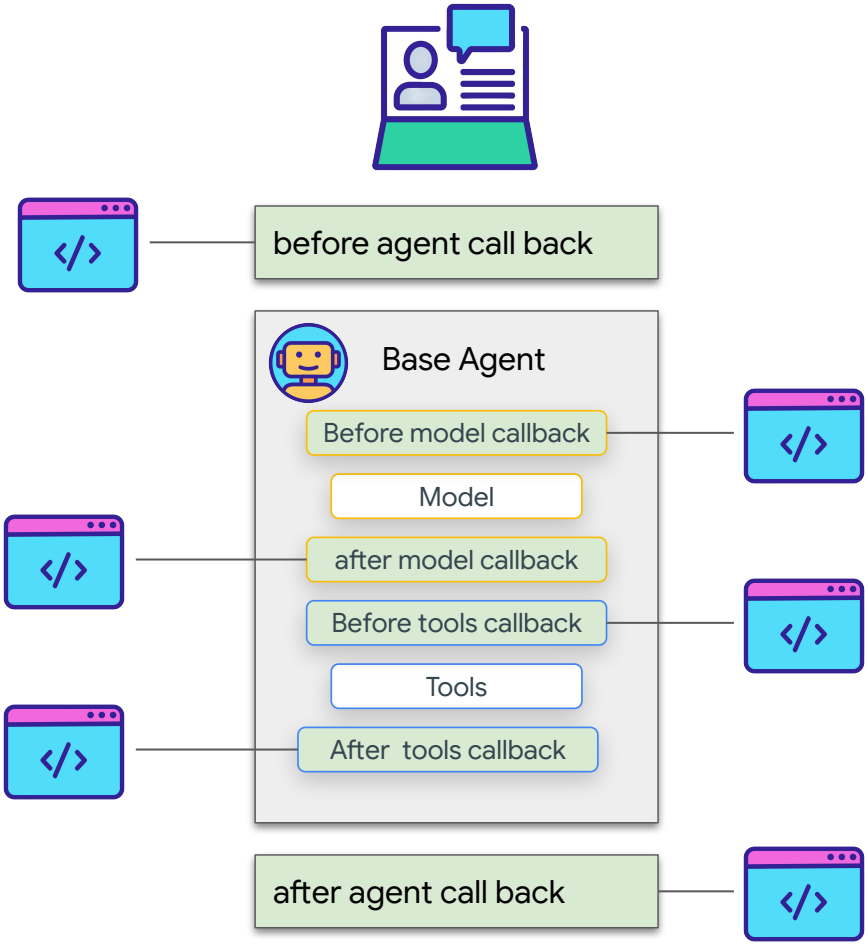
Vertex AI

Google Cloud

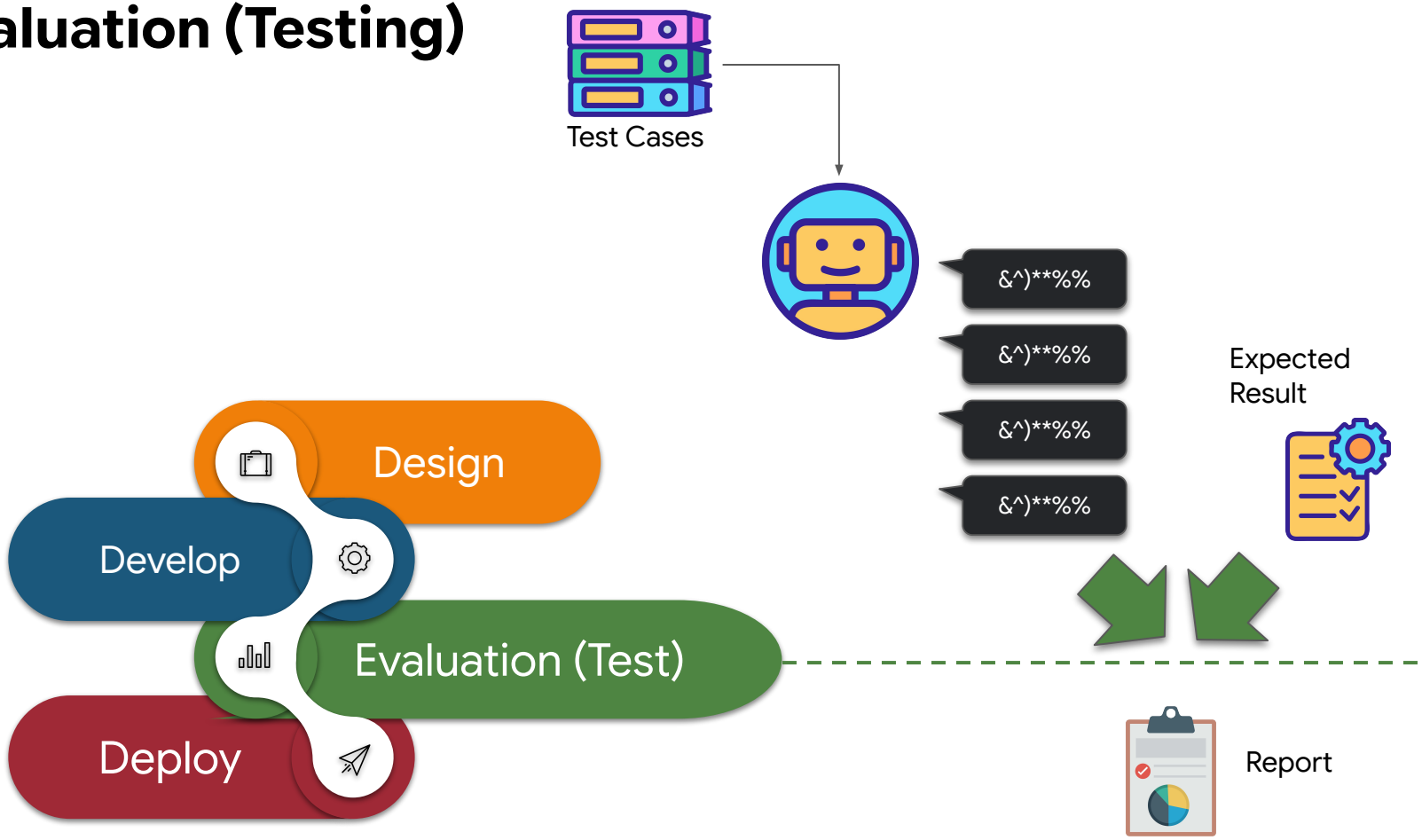
ADK - Runner



Call backs

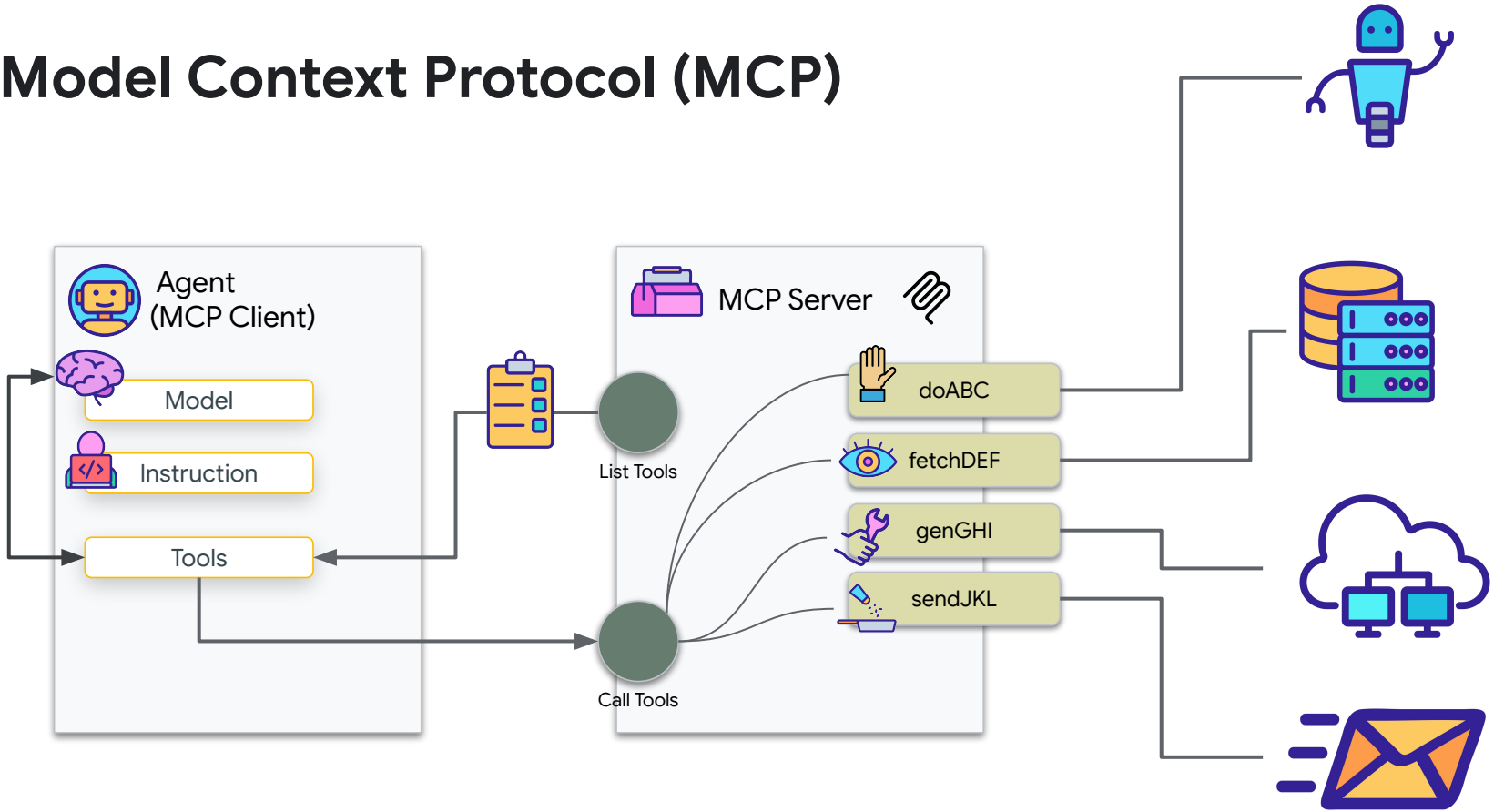

before agent call back
Base Agent
Before model callback
Model
after model callback
Before tools callback
Tools
After tools callback
after agent call back

Evaluation (Testing)



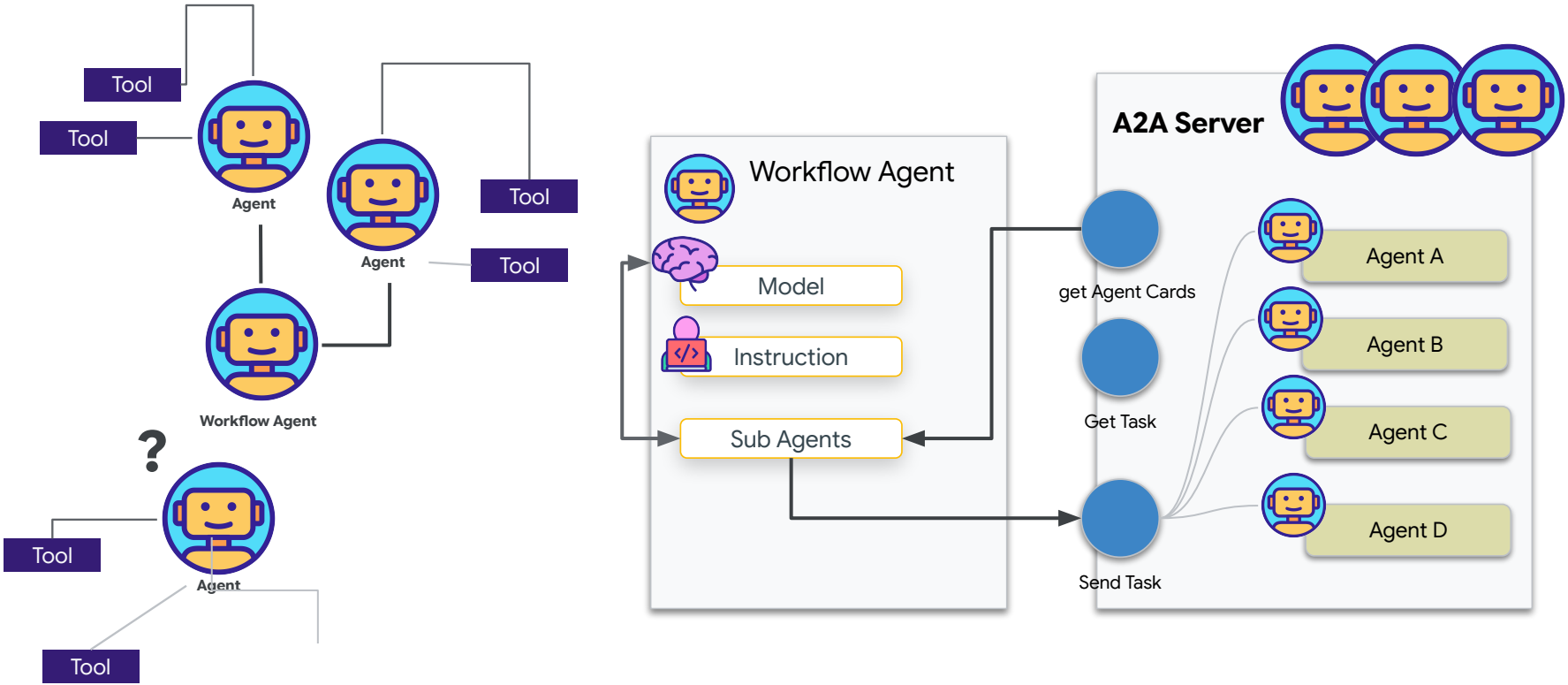
Google Cloud

Model Context Protocol (MCP)



Google Cloud

Agent2Agent (A2A) Protocol



Google Cloud

A2A - Agent Card

- A digital profile or manifest for an agent. It's a standardized metadata file (JSON)
- To orchestrating systems
- For discovery

Name: Agent ABC
Provider: Google (URL)
Preferred input/output: text

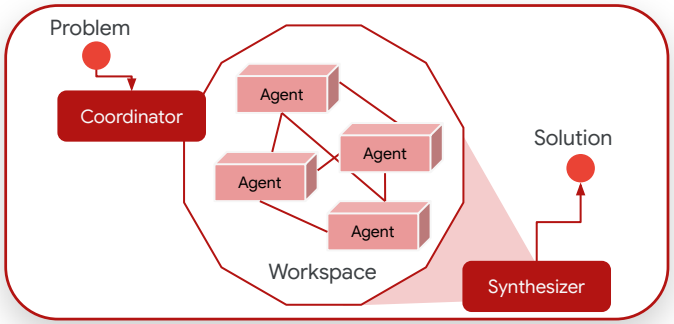
Capabilities:
I can do streaming
I can do Push Notifications

Skills:
- Skill A
  - Description
  - Input
  - Output
- Skill B
  - Description
  - Input
  - Output

Authentication:
Scheme: Bearer
Key: xxx

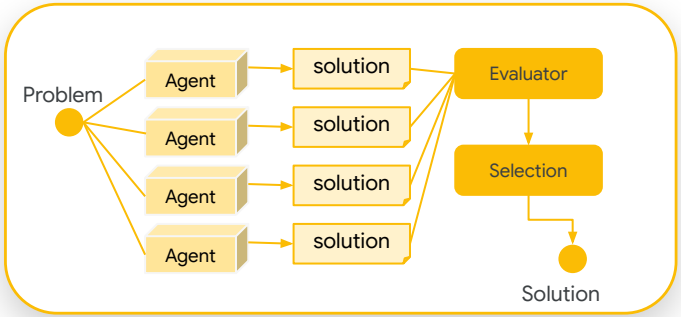
Multi-Agent Interaction

Cooperative Multi-Agent System



Multiple LLM-based agents collaborating to solve problems

Competitive Multi-Agent System

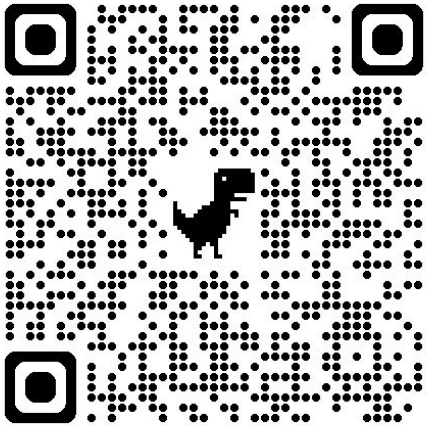


Independent LLM agents to propose competing solutions, with evaluator model selecting best or integrating elements

What is Next?

Advance Workshop: Building AI Agents

https://docs.google.com/document/d/1SVc_1-JVlo-6hsSUfF29jyiqT6vntRnzVz622TdjSyw/edit?tab=t.0



https://rdi.berkeley.edu/events/agentic-ai-summit

Google Cloud

Q&A