* Agents *

* Introduction

- We humans use books, internet to learn complex topics, messy pattern recognition.
- Just like humans, we can train AI models to use external tools to access real time information.
- This combination of reason, logic & access to information that are all connected to Generative AI envokes concept of Agents.
- It's a program that extends beyond the standalone capabilities of Gen AI models.

* What is an Agent?

- Generative AI Agents can be defined as application that attempts to achieve a goal by observing the world & acting, upon it using the tools it has at its disposal.
- Agents are autonomus & can impact independently of human intervention.
- Agents can reason about what it should do next to achieve the goal even in absence of explicit instructions by human.
- The combination of components which drive agent's behaviour, action & decision making, can be described as Cognitive Architecture.

The 3 components which forms Cognitive Architecture are

1. Orchestration
2. Model
3. Tools

* The Model.

- In context of Agents, the model refers to the Language model (LM)
- This LM will be utillized for centralized decision making.
- This LM coulde be a single or multiple LM of any size (small / Large)
- This LM's shoulde be capable of ~~making~~ following instructions based reasoning & logic frameworks like ReAct, Chain-of-Thought.

- LM's can be general purpose, Multimodal or fine-tuned based on need of specific Agent.

- The important thing to note that the model is typically not trained with the specific configuration settings (i.e. tool choices, orchestration / reasoning setup)

- However, it is possible to refine the model for agent's tasks by providing it with examples that showcases Agents capabilities.

*The tools

- Tools bridge the gap between traditional foundational models by allowing them ~~access~~ to interact with outside world.

- Tools can take various forms & have varying depths of complexity, but typically align with comman web API methods like GET, POST, PATCH & DELETE.

- Tools allows agents to access real world information, this empowers them to support more specialized systems like. Retrieval Augmented Generation (RAG)

* The Orchestration layer.

- The orchestrating layer describes a cyclical process that governs how the agent takes in information, performs some internal reasoning & uses that reasoning to inform its next action or decision.

- In geneal, this loop will continue until the agent has reached its goals.

- The complexity of orchestration layer vary deeply depending on the agent & task it's performing.

* Cognitive Architectures:
How agents operate

- Agents can use cognitive Architecture to reach their end goals by iteratively processing information, making informed decisions, & refining next actions based on previous outputs.

- At the core of ~~Orchestratic~~ Cognitivite Architecture lies orchestration layer, responsible for maintaining memory, state, reasoning & planning.

- It uses rapidly evolving field of prompt engineering & associated frameworks to guide reasoning & planning, enabling the agent to interact more efficetively with its enviarment.

- Following are some of the most famous frameworks & reasoning techniques available at time of publication (Sept 24)

1. ReAct
2. Chain-of-Thought (CoT)
3. Tree-of-Thought (ToT)

* ReAct

- It is a prompt engineering framework that provides a thought process statergy for LLM to Reason & take action on user query, with or without in-context examples.

- ReAct prompting has shown to outperform several SOTA baselines & improve human interoperability & trustworthiness of LLMs.

ReAct steps → i) Question
ii) Thought
iii) Action
iv) Action input
v) observation
vi) Final Answer.

* Chain-of-Thought (CoT)

- This is a prompt engineering frame work that enables reasoning capabilities through 'intermediate steps.

- There are various sub-techniques of CoT including self-consistency active-prompt & multimodal CoT that each have strengths & weakness depending on specific application.

* Tree-of-Thoughts (TOT)

- this is a prompt engineering framework that is well suited for exploration or statergic lookahead tasks.

- It generalizes over chain of thought prompting & allows the model to explore various thought chains that serve as immediate steps for general problem solving.

* Tools : our key to outside world.

- A Language model is only as good as what it learned from its training data. But regardless of how large the training data is, they still lack the fundamental ability to interact with outside world.

- Functions, Extensions, Data steres & plugins are all ways to provide this critical ability to the model.

- Tools are what create a link between foundational models & outside world.

- As of date of the publication (sept 2024), there are three primary tools types that google models are able to interact with:
  i) Extensions
  ii) Functions
  iii) Data stores.

* Extensions

- The easiest way to understand extensions is to think of them as bridging the gap between an API & an Agent in a standardized way;

- Extensions allows agents to seamlessly execute APIs regardless of their underlying implementation.

- Extension bridges gap between an agent & an API by:

  i) Teaching the agent how to use API endpoint using Examples.

ii) Teaching the agent what arguments or parameters are needed to successfully call the API endpoint.

- The agent uses the model & examples at the run time to decide which Extension, if any, would be suitable for solving user's query.

- This highlights key strength of extensions, their built-in-example types, that allow the agent to dynamically select most appropriate Extension for the task.

* Functions

- A model can take a set of unknown functions & decide when to use each function & what arguments the function needs based on its specification.

- Functions differs from Extensions in few ways, most notably:

  i) A model outputs a Function & its arguments, but doesn't make a live API call.

  ii) Functions are executed on the client-side, while Extensions are executed on agent-side.

Note!- Difference here is neither the function nor the agent interact directly with the Google Flights API

- With Functions, the logic & execution of calling the actual API endpoints is offloaded away from agent & back to the client side application.
- This offers developer a more granular control over the flow of data in application.

* Reasons for choosing function over extension

- API calls need to be made at another layer of the application stack, outside of the direct agent architecture flow.
- Security or Authentication restrictions that prevent the agent from calling an API directly.

- Timing or order-of-operations constraints that prevent the agent from making API calls in real-time.

- Additional data transformation logic needs to be applied to the API Response that agent cannot perform.

- The developer want's to iterate on agent development without deploying additional infrastructure for the API endpoints.

* Use cases

- A model can be used to invoke functions in order to handle complex, client-side execution flows for the end user, where the agent developer might not want language model to manage the API execution.

- you want LM to suggest a function that you can use in your code, but you don't want to include credentials in your code. Because function calling doesn't run the function.

- You are running asychronus operations that can take more than few seconds. These scenarios work well with function calling.

- You want to run functions on a device that's different from the system producing the function calls & their arguments.

* Data stores

- Language models have a knowledge cut-off due to they being trained on a limited data.

- Language models are not able to access more dynamic & up-to date information.

- Data stores address this's problem by giving them access to such information; ensuring model's response remains grounded to factuality & relevance

- Data stores allows developers to provide additional data in its orignal format to an agent, eliminating the need for time consuming data transformations, model retraining or fine-tuning

- The Data store converts the incoming document into a set of vector database embeddings that agent can use to extract the information it needs to supplement its next action or response to the user.

* Implementation & Application.

- In context of Gen AI Agents, Data sources are typically implemented as vector database that the developer wants the agent to have access at runtime.

- The key point to understand is that they store data in the form of vector embeddings, a type of high-dimensional vector or mathematical representation of data provided.

- one of the prolific examples of implementation has been Retrieval Augmented Generation (RAG) based applications.

- RAG seek to extend the breadth & depth of ~~kr~~ model's knowledge beyond the foundational training data by giving model. access to data in various formats like:

  i) Website content

  ii) Structurd Data formats like PDFs, word docx, CSV etc.

  iii) Unstructured Data in formats like HTML, PDF, TXT etc.

- The end result is an application that allows the agent to match user's query to a known data stere through ~~veeter~~ search, retrieve the original content, & provide

it to the orchestration layer & model for further processing. The next action might be to provide final answer to the user, or perform an additional Vector search to further refine results.

* Enhancing model performance with targeted learning.

- A crucial aspect of using models effectively is their ability to choose the right tools when generating output.
- To achieve optimal model performance & help model gain access to specific type of knowledge, several approaches exist.

i) In context learning
ii) Retrieval-based in-context learning
iii) Fine-tuning based learning.

* In-context learning

- This method provides a generilized model with a prompt, tools & few-shot examples at inference time which allows it to learn 'on the fly' how & when to use those tools for specific task.

- The ReAct framework is an example of this approach in natural language.

* Retrieval-based in-context learning

- This technique dynamically populates the model prompt with most relevant information, tools & associated examples by retrieving them from external memory.

* Fine tuning based learning

- This methods involves training a model using larger dataset of specific examples prior to inference.
- This helps the model understand when & how to apply certain tools prior to receiving any user queries.