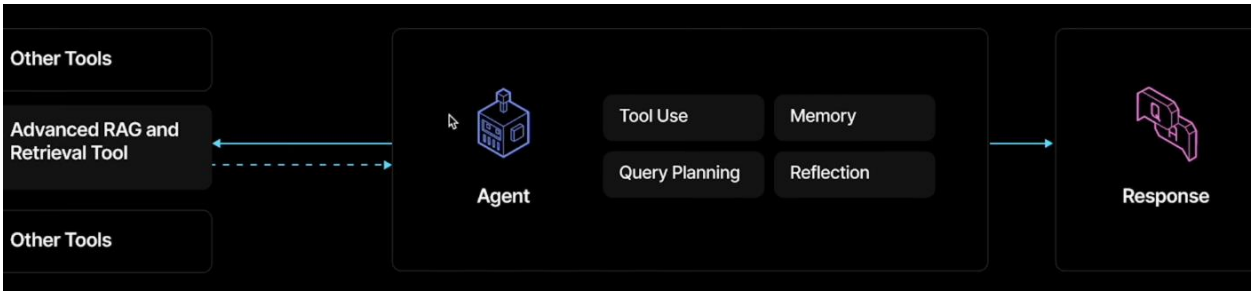
A knowledge assistant's purpose is to take input from the human and gives back some response where the input is of arbitrary complexity and the response generated is also of arbitrary complexity like generating simple answers for simple questions and generating structured/research oriented response for specific domain/research questions and also perform actions in the world instead of just generating a response.

A Basic RAG applied system which many try to productionize is mostly input-in and output-out which does not drive knowledge automation (decision making power is limited) into the response generated thus limiting the investment the user has made giving rise to hallucinations. So to deploy a production ready agent which gives more end user value on the basic rag we create knowledge assistants for our specifc purposes.

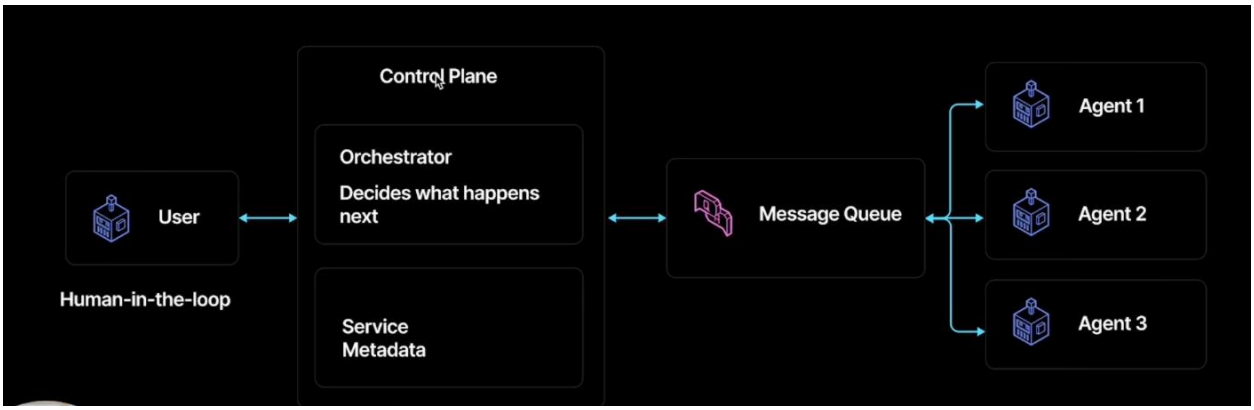
There are 4 steps into building this type of assistant:High quality data and retrieval interface

, Leveraging some Agnetic reasoning layer over complex inputs to process these inputs in a manner that allows you to solve more complex tasks



, Agentic decision making where instead of just generating a simple chatbot response the agent must decisively generate more sophisticated response in regard to the complex/simple input the user gives and output generation

and finally stepping towards a scalable full-stack application going from code present in a venv into a complete web application.



Good data quality is a necessary component of any production LLM application, In RAG instance the ETL layer where it takes unstructured data, parses it, chunks it and then putiing it into vector database is a simple process but doing it well is difficult and if we process the data in a wrong way then our llm will hallucinate.

Majority of the documents are complex as they not only contain clean text only but have tables, charts, images and such additional things which can be challenging to include them into the vector database and one of the ways to advance in this area is through parsing one of the first pieces of ETL which helps the llm to understand the components present in the complex documents and now after parsing the next step is to index this data properly to be used by llms in the right manner.

The steps followed for end-to-end parsing indexing retrieval is that we first parse the documents containing the elements of text, tables, diagrams by extracting a summary representation and index that representations and then during synthesis (retrieval) you retrieve the index representations which are called nodes and we de-reference them into source element to be given out onto the output response generated thus this technique is powerful in case of applying it to RAG by linking various source elements with the document.

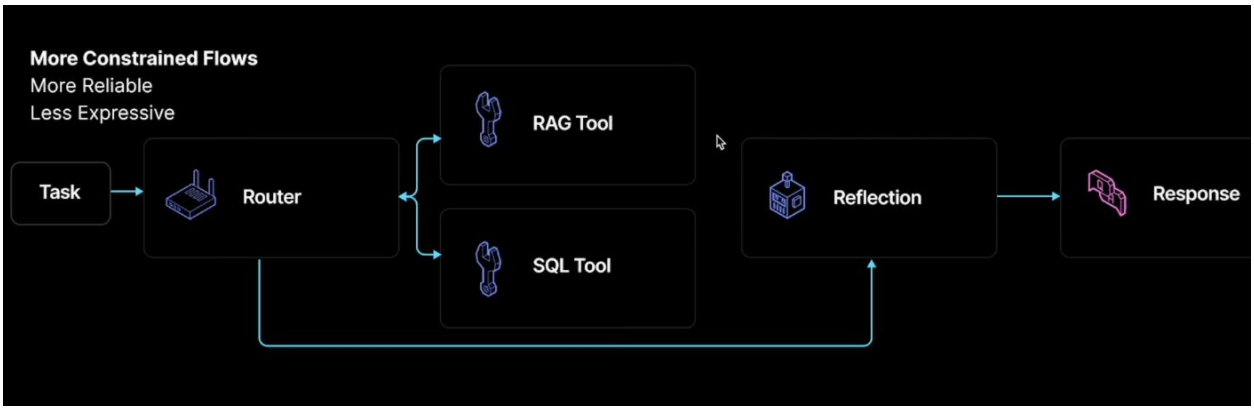
Naive RAG works well for pointed questions but fails on complex tasks even if your data pipeline is good or you are using best interfaces for parsing , indexing, retrieval the issue is that LLM is still used for generation but not reasoning like in Multi-part questions/Research tasks where the LLM basically retrieves the top-k chunks that are to be retrieved and feed it into the prompt will not solve this issue.

So to solve this issue we utilize the Agentic reasoning layer to process the query before it goes to different interfaces for instances of vector database thus our goal here is to treat every single data interface (vector database, sql database, graph database, web search or more) as a tool and run agentic loops on the input text with different components in an agent to process the input

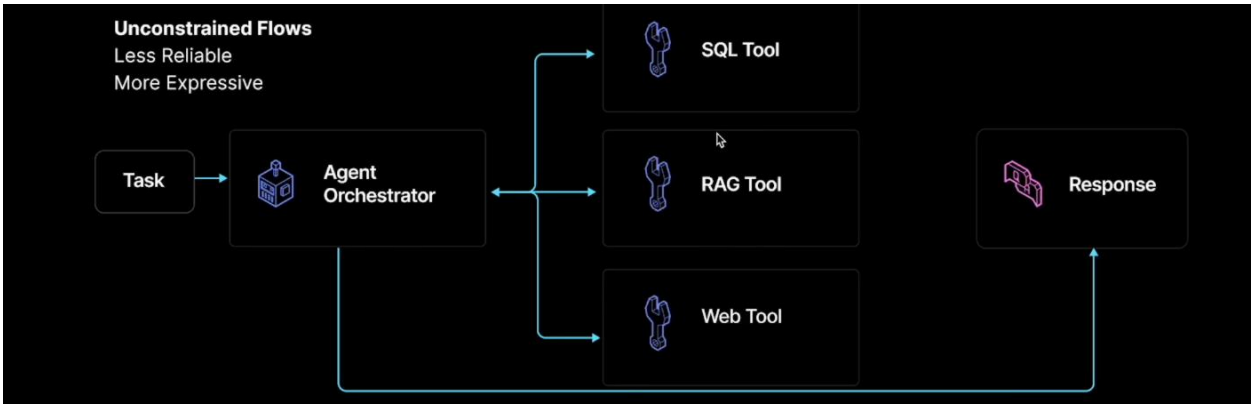
and tackle these complex tasks thus resulting in stuff like chain of thoughts, dag based planning, query decomposition.

Instead of directly feeding the enduser input and embedding it and putting it into a vector database this agent reasoning layer enables to build personalized QA systems capable of handling more complex tasks.

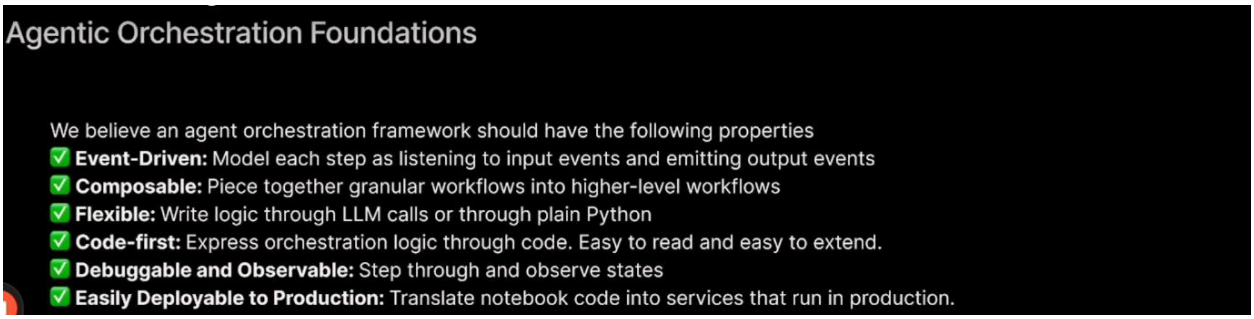
Let us get a pictorial view of agent's flow built by different users which are constrained flows and unconstrained flows: Constrained flows is where we use llm to make one call/decision and the rest of the flow after that call is kind of pre-determined/static thus making the flow constrained,



Unconstrained flows is where once the user gives the input to the agent orchestrator it will decide which tools to use and how many loops over the tools it has to use until it reaches the required output it wants to generate, so unlike constrained we do not restrict the no. of tools to be used or no. Of times a tool is used thus leading to a dynamic pathway and handing more control to the llm



We need to get the best of both types of flows to be present in our framework of designing an agent that has more flexibility but also follows some of the guidelines for the task:



We can also agentic reasoning to help with the output generation since the purpose of agents is not only restricted to chatbot responses/ search capabilities it is to create knowledge artifacts and also take actions. Report generation is a very broad area where you basically generate the complete unit of output with various types of data in them and the most practical use cases for this are: Generating a full research report or presentation (with text, images or tables), Fill out an example form ot questionnaire, Fill out an Excel sheet.

Lets say you are generating a survey report over a bank of documents and we can generate text, tables or images responses by researching them and the architecture usually followed are: Researcher retrieves relevant chunks and documents and puts them into a data cache and the writer who uses the data cache to generate a structured output of interleaving text and image blocks., Thus producing a generation of Multimodal report.

One can use an architecture that shows how to use llama index workflows with llama cloud and llama parse to appropriately index your knowlege base, parse the RFP and create an agentic system that can finally generate a response to the RFP.

Action taking and output generation potentially lead to much greater ROI in terms of time savings and capability improvement.

One needs the right architecture and components to serve complex, agentic workflows to end users as a production application and some of the requirements for this are: Encapsulation of the workflows behind an API, creating a standardized communcating agent api which can communicate with other api/tools, being able to scale up the number of users and number of agents within your architecture and can communicate with each other and you add/remove some and also include human in the loop aspects so that agents can pause execution when needed to wait on human inputs and providing the right tooling for developers to observe their agent systems.

Llama deploy is one such architecture where it directly connects with llama index workflows and it deploys any sort of agent tech workflow we define as microservices and allows you to model every agent workflow as a service api and allows model communication between the workflows through a message cube which gives us back a core easy to use API server that you plug in your workflow and deploy it using llama-deploy and one line of code and get this api server to serve

client requests and so its a great service for hosting agents on some service and it also includes human in the loop for monitoring purposes.