

Building with AI

Meet the team



Vitor Ventura

Lead Security
Researcher at Cisco
Talos



**Martin
Wendiggensen**

PhD candidate at
Johns Hopkins

Agenda

1



Setup

Env setup
Quick LLM/AI basic
concepts intro

2



RAG from ground up

Understanding the
components of a RAG
system from within

3



Agentic World

Everything got simpler
with the agentic
approach

4



Closing thoughts

Ideas and Resources to
take for your use-cases

Setup

- Fetch the repo
- `./run_all.sh build`
- `./run_all.sh start`
- `./run_all.sh pull_model gemma3:4b`
- `./run_all.sh start_model gemma3:4b`
- `./pivot2025-shell.sh`
- `./python3 chat_test.py`

Basic concepts



Large Language Model

In simple words it's a list of tokens with different weights

Basically massive precomputed neural network of interconnected tokens.

Current models are **NOT** open sourced, we only know the weights we don't know the relation between the words

Large Language Model usage components

Prompt

Inference engine

Interface

Agents/tools (model tools and agents)

Model

Large Language Model usage components

Prompt – A template of communication with the LLM. Contains certain special tokens to mark positions with intent.

Inference engine – The engine that will run the prompt on the model, Ollama, LLamaCPP,

Interface – We need to send the data somehow. REST api, web page, app, etc

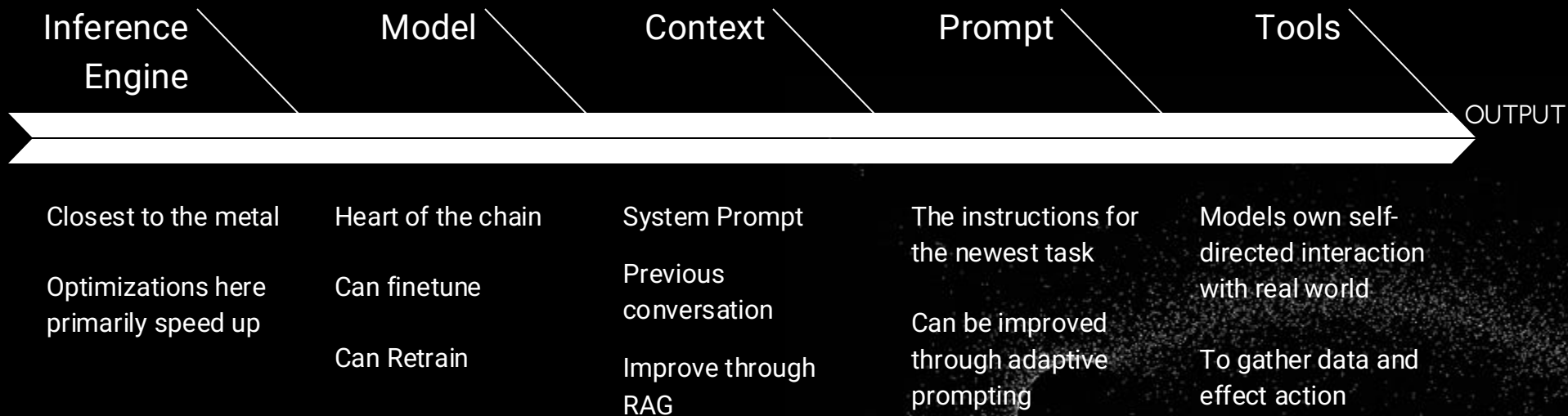
Large Language Model usage components

Where can we get models? Huggingface (for open source), model provider APIs

Interesting models:

- LLama the model we will be using
- MS Phi models are interesting because they are SLMS (Small Language models)
- Gemma Google open weight models family

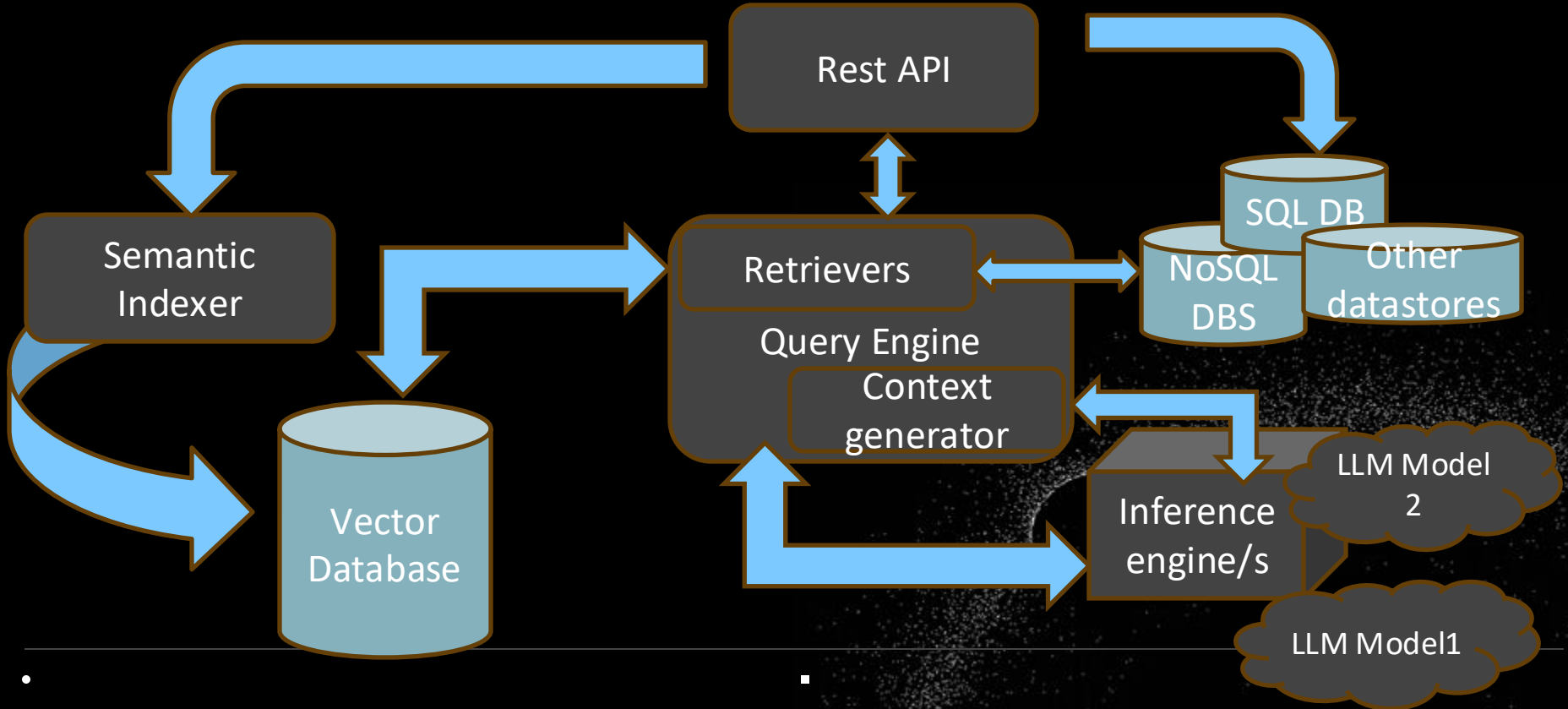
Layers



RAG from ground up



Possible RAG architecture



RAG Step 1: Indexing

1. First we start by indexing data
 - We need to index it in such a way that we keep the semantic similarity.
 - Then we store it on the vector database.

But what is semantic similarity and how do we achieve it?!

- It can't be just a word, otherwise you would lose the semantic meaning.
(And for that we already have fuzzy logic and BM25 full text search algorithms)

RAG Step 1: Indexing

We need pieces of information to which we will call chunks.

Chunks can be created by splitter

- A splitter will split the information based on textual characteristics of the information

Examples:

Text splitter, Character splitter, Recursive Character Splitter, Sentence splitter, Semantic splitting

RAG Step 1: Indexing

We need pieces of information to which we will call chunks.

Chunks can be created by splitter

- A splitter will split the information based on textual characteristics of the information

Examples:

Text splitter, Character splitter, Recursive Character Splitter, Sentence splitter, Semantic splitting

And we embed those chunks:

Embeddings are a vector of floats between -1 and 1

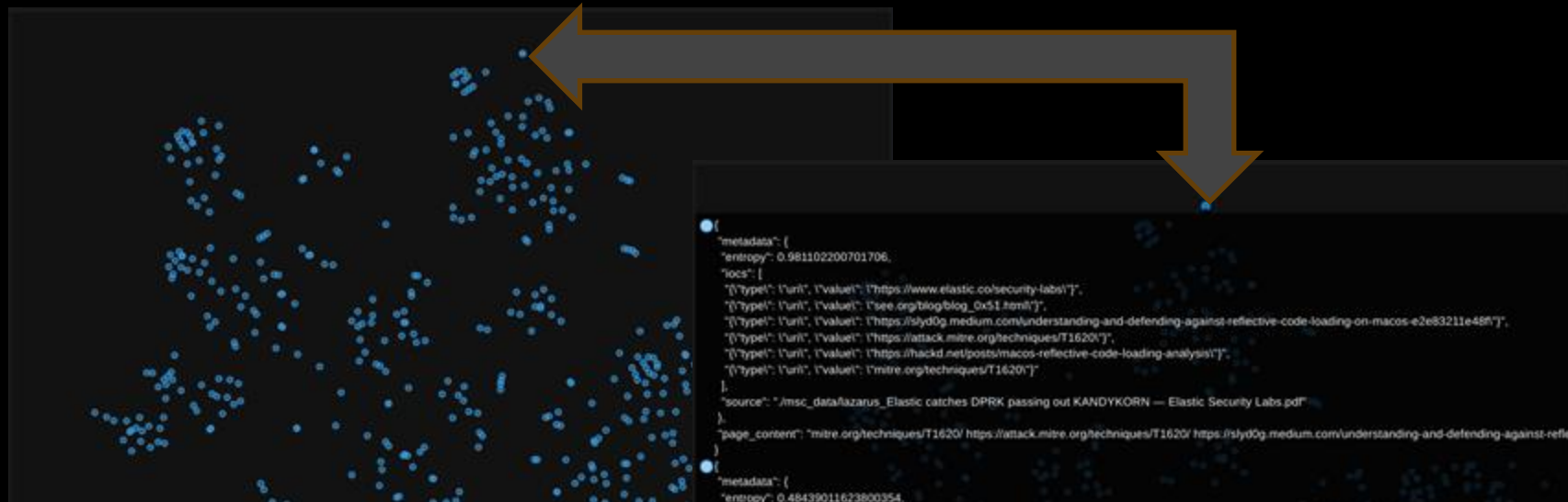
A hand-drawn diagram illustrating a vector embedding. On the left, a vertical column of five boxes represents a vector. To its left, the following labels are written in blue: 'Royalty', 'Masculinity', 'Femininity', 'Age', and '...'. To the right of this column are four more vertical columns, each representing a different role: 'King', 'Queen', 'Woman', and 'Princess'. Each of these role columns contains five numerical values corresponding to the characteristics on the left. The values are written in green. The 'King' column has values 0.99, 0.99, 0.05, 0.7, and a vertical ellipsis. The 'Queen' column has values 0.99, 0.05, 0.93, 0.6, and a vertical ellipsis. The 'Woman' column has values 0.02, 0.01, 0.999, 0.5, and a vertical ellipsis. The 'Princess' column has values 0.98, 0.02, 0.94, 0.1, and a vertical ellipsis. To the right of the 'Princess' column is an ellipsis '...'. The entire diagram is drawn with simple lines and colored text.

	King	Queen	Woman	Princess	...
Royalty	0.99	0.99	0.02	0.98	
Masculinity	0.99	0.05	0.01	0.02	
Femininity	0.05	0.93	0.999	0.94	
Age	0.7	0.6	0.5	0.1	
...	⋮	⋮	⋮	⋮	⋮

How to semantically index data

```
"page_content": "google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze trac. Learn more. 7/25/24, 3:22 PM APT41  
Has Arisen From the DUST | Google Cloud Blog https://cloud.google.com/blog/topics/threat-intelligence/apt41-arisen-from-dust 9/27 List installed security sware  
Get system info List user accounts Get system boot time Enumerate hidden and visible process windows File Manipulation Operations Open le Write le CRC32 le  
content Read le Close le Keylogger Activate Delete log Active Directory Operations Enumerate domain controller information Add user Delete user Get server  
configuration Get server shares Get detailed server and workstation domain information Enumerate servers Get list of services Get list of network shares Add  
network share Disconnect network share Get list of users Set user password File Uploader Upload le resident on disk RDP Enumerate remote desktop sessions DNS  
Operations Peerm DNS lookups DNS Cache Operations Contact sales Get started for freeCloud Blog cloud.google.com uses cookies from Google to deliver and enhance  
the quality of its services and to analyze trac. Learn more. 7/25/24, 3:22 PM APT41 Has Arisen From the DUST | Google Cloud Blog https://cloud.google.com/blog/topics/threat-intelligence/apt41-arisen-from-dust 10/27 Retrieves DNS cache table operations Registry Operations Get registry value Dump registry path and  
children to disk Set registry value Delete registry value Figure 3: Full execution flow of DUSTTRAP SQLULDR2 SQLULDR2 is a command-line utility wrien in C/C++  
that can be used to expo the contents of a remote Oracle database to a local text-based le. There are multiple command-line parameters available to specify the  
details of the data expo including but not limited to: query, user, rows, and text. APT41 expoed data from Oracle Databases to CSV formats with the following  
command: C:\\ProgramData\\luldr\\luldr\\sqluldr.exe user=USER@%<DATABASE> charset=utf8 safew=yes head=yes text=csv r batch=yes query=SQL QUERY> file=OUTPUT>  
,  
"vector": [,  
-0.007903519,  
-0.012354377,  
-0.00039804904,  
0.04552234,  
-0.01694772,  
-0.01849544,  
-0.0085329795,  
-0.011437501,  
0.057074207,  
0.051309537,  
0.01555647,  
0.018520728,  
0.025231227,  
-0.025213003,  
-0.03821052,  
0.025274215,  
-0.00050200166,  
-0.013977106,  
-0.04433055,  
0.008452741,  
-0.00061100564,]  
-0.0009042986,  
-0.027423006,  
-0.019047594,  
-0.03450402,  
0.012202677,  
-0.04971138,
```


How to semantically index data



Away from everything
(useless)
content similar

How to semantically index data

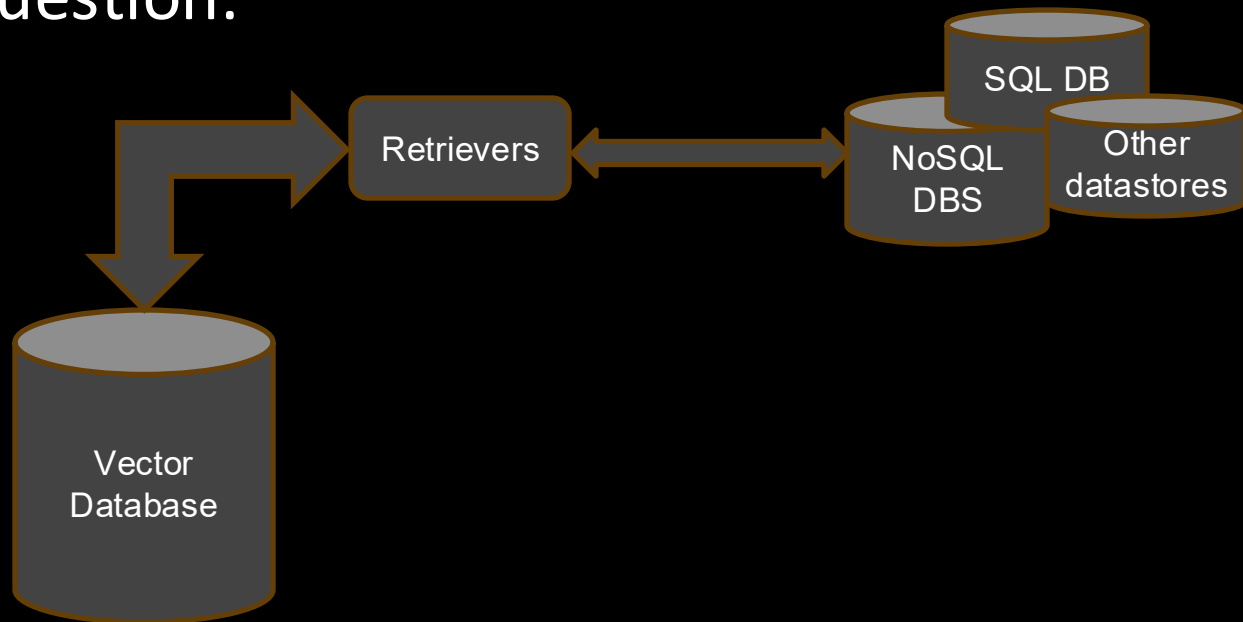
Not perfect but
we can see some
clustering



Retrievers

We want to retrieve the information relevant for our question.

Vector search
Hybrid search
Fulltext search
Others



Context generators

What is context?

Information that is given to the LLM to generate the answer.

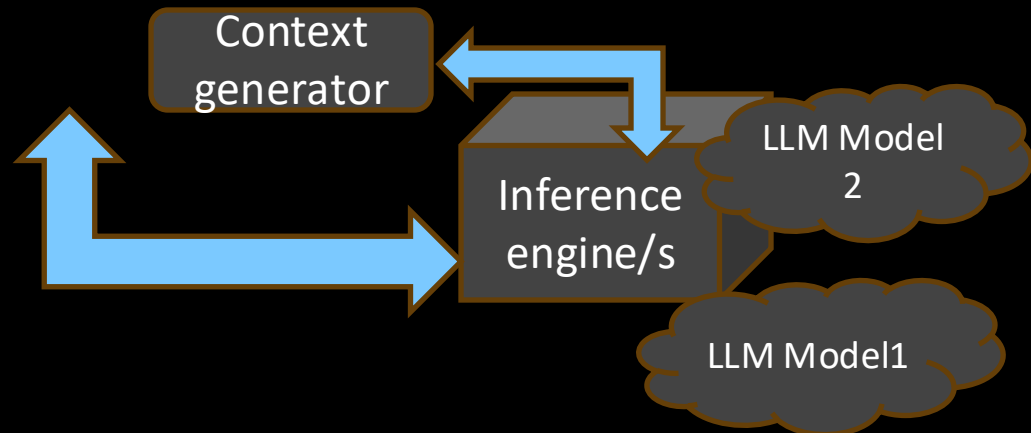
This information is collected from the available sources

It increases the LLM "knowledge" beyond the contents of the training dataset.

Context generators

What is context generator?

- Multiple sources of context
- Quality assurance
- Context merging and reranking
- Previous interactions
- AI agents



Context generators - advanced

- Merge all context and summarize
- Remove noise from the context
- Recursively search for more information
- Extract entities to increase focus
- Translate context for normalization
- Along with the question determine the best model to generate answers
- Enrich context with external metadata

Prompts

The way to convey the LLM the context and make the question

Different models have different prompt formats.

Roles

- system
- assistant
- user
- ipython

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>
```

```
Cutting Knowledge Date: December 2023
```

```
Today Date: 23 July 2024
```

```
You are a helpful assistant<|eot_id|><|start_header_id|>user<|end_header_id|>
```

```
What is the capital of France?<|eot_id|>
```

```
<|start_header_id|>assistant<|end_header_id|>
```

Prompts - advanced

Run call tools

Run python code

```
# for Search
<|python_tag|>
brave_search.call(query="...")
<|eom_id|>

# for Wolfram
<|python_tag|>
wolfram_alpha.call(query="...")
<|eom_id|>
```

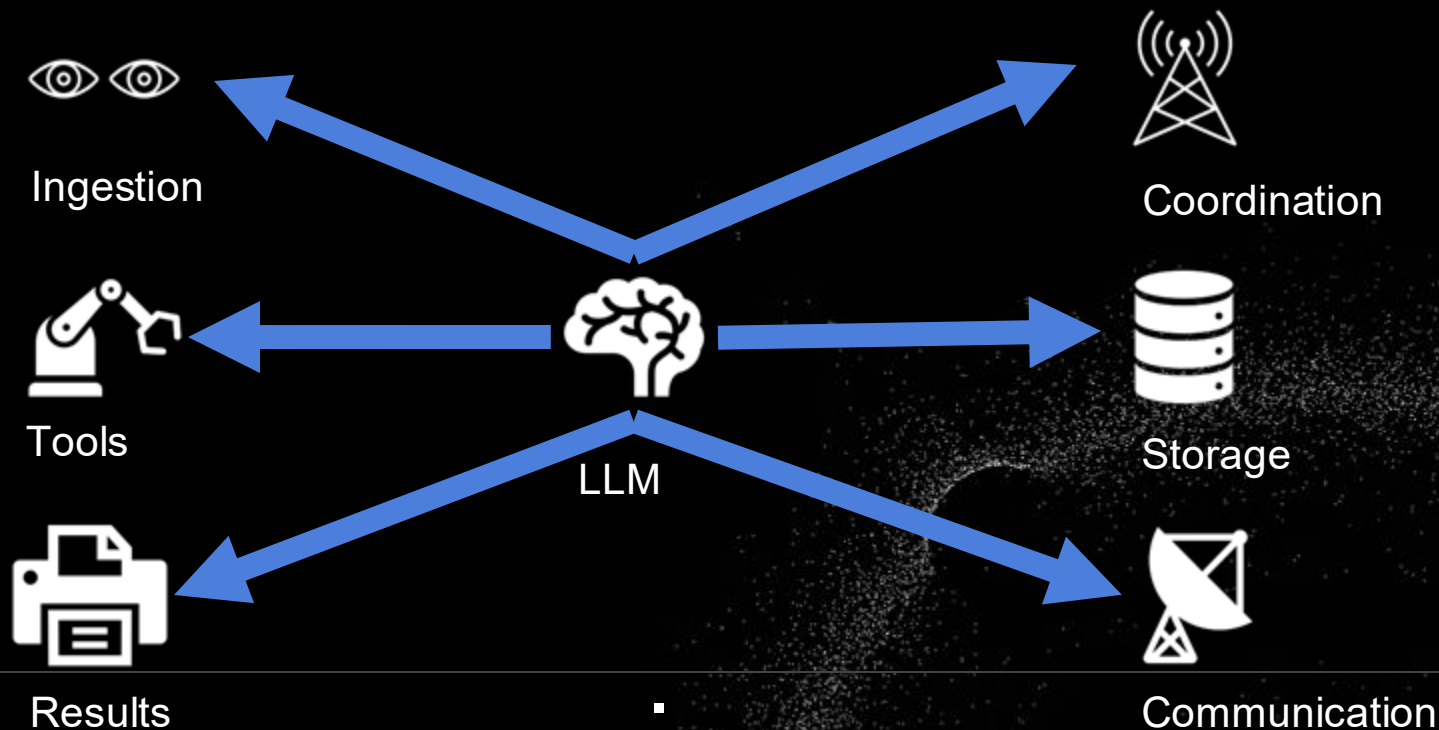

Prompts - examples

```
"reply": {
  "answer": "```\n{\n  \"text\": \"June 11, 2024\", \n  \"type\": \"DATE\", \n}, \n{\n  \"text\": \"AhnLab\", \n  \"type\": \"ORGANIZATION\", \n}, \n{\n  \"text\": \"UAT-5394\", \n  \"type\": \"GROUP\", \n}, \n{\n  \"text\": \"June 12, 2024\", \n  \"type\": \"DATE\", \n}, \n{\n  \"text\": \"95.164.86.148\", \n  \"type\": \"IP_ADDRESS\", \n}, \n{\n  \"text\": \"MoonPeak\", \n  \"type\": \"ORGANIZATION\", \n}, \n{\n  \"text\": \"RDP\", \n  \"type\": \"SOFTWARE\", \n}, \n{\n  \"text\": \"Port 9999\", \n  \"type\": \"SOFTWARE\", \n}, \n{\n  \"text\": \"July 4, 2024\", \n  \"type\": \"DATE\", \n}, \n{\n  \"text\": \"27.255.81.118\", \n  \"type\": \"IP_ADDRESS\", \n}, \n{\n  \"text\": \"July 5, 2024\", \n  \"type\": \"DATE\", \n}, \n{\n  \"text\": \"Port 9966\", \n  \"type\": \"SOFTWARE\", \n}, \n{\n  \"text\": \"167.88.173.173\", \n  \"type\": \"IP_ADDRESS\", \n}, \n{\n  \"text\": \"Port 9936\", \n  \"type\": \"SOFTWARE\", \n}\n}\n```\",
  "nodes": [],
  "iocs": []
},
"request": {
  "query": "Since June 11, 2024, we saw a distinct shift in the actor's tactics with respect to setting up supporting infrastructure. After AhnLab's disclosure, UAT-5394 moved from hosting their malicious payloads on legitimate cloud storage providers to systems and servers they now owned and controlled. It is likely that this move was made to preserve their infections from potential shutdown of cloud locations by the service providers. \n95[.]164[.]86[.]148 is one of the earliest servers set up and actively used by UAT-5394 since at least June 12, 2024, to host malicious artifacts (described in AhnLab's disclosure) and served as a MoonPeak C2 server on Port 9999 until at least July 4, 2024. This C2 server was accessed between this time frame, over RDP by 27[.]1255[.]81[.]118, another UAT-5394 IOC resolving multiple malicious domains registered by the threat actors. \nOn July 5, 2024, the threat actors now used 95[.]164[.]86[.]148 to RDP into a second malicious server, 167[.]88[.]173[.]173 which was already serving as a MoonPeak C2 on Port 9966. This RDP access to 167[.]88[.]173[.]173 resulted in a second deployment of MoonPeak's C2 on Port 9936. ",
```

Agentic world



What is Orchestration?

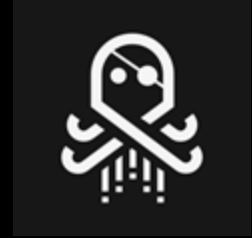


A Note about Orchestration Options

- There are many options



- Rigging



- Langchain



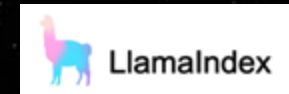
- Fence



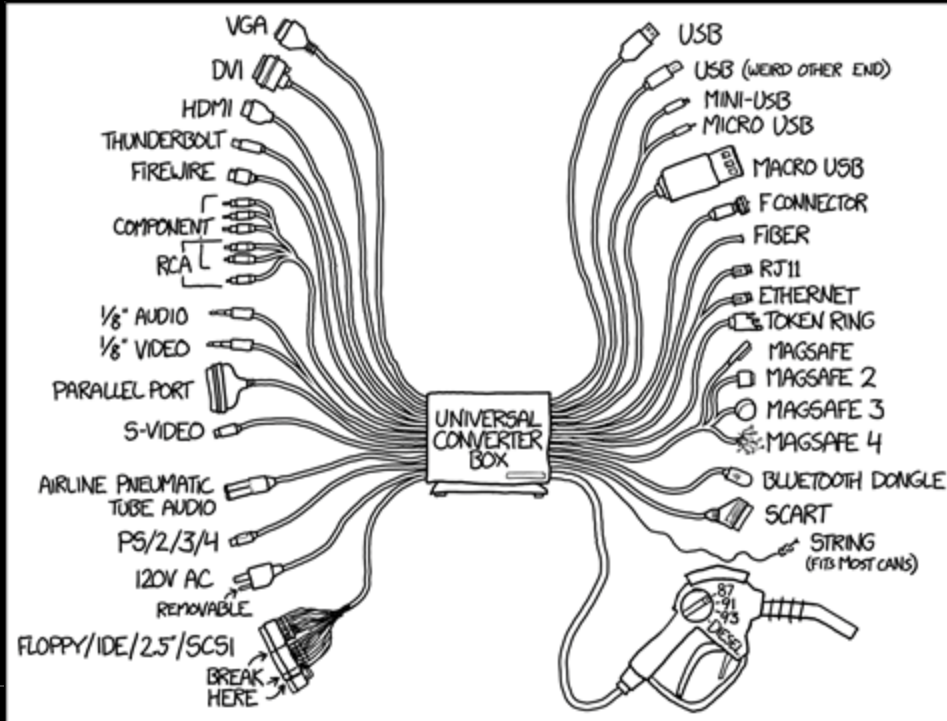
- AutoGen



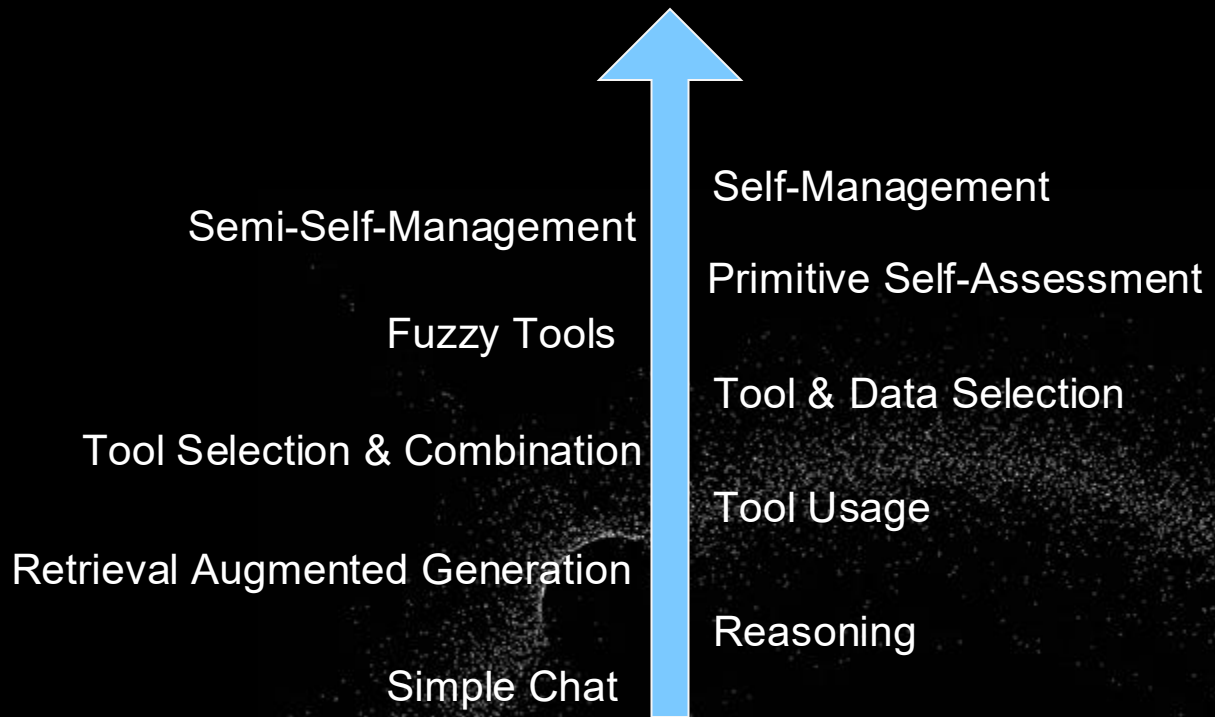
- Llama Index



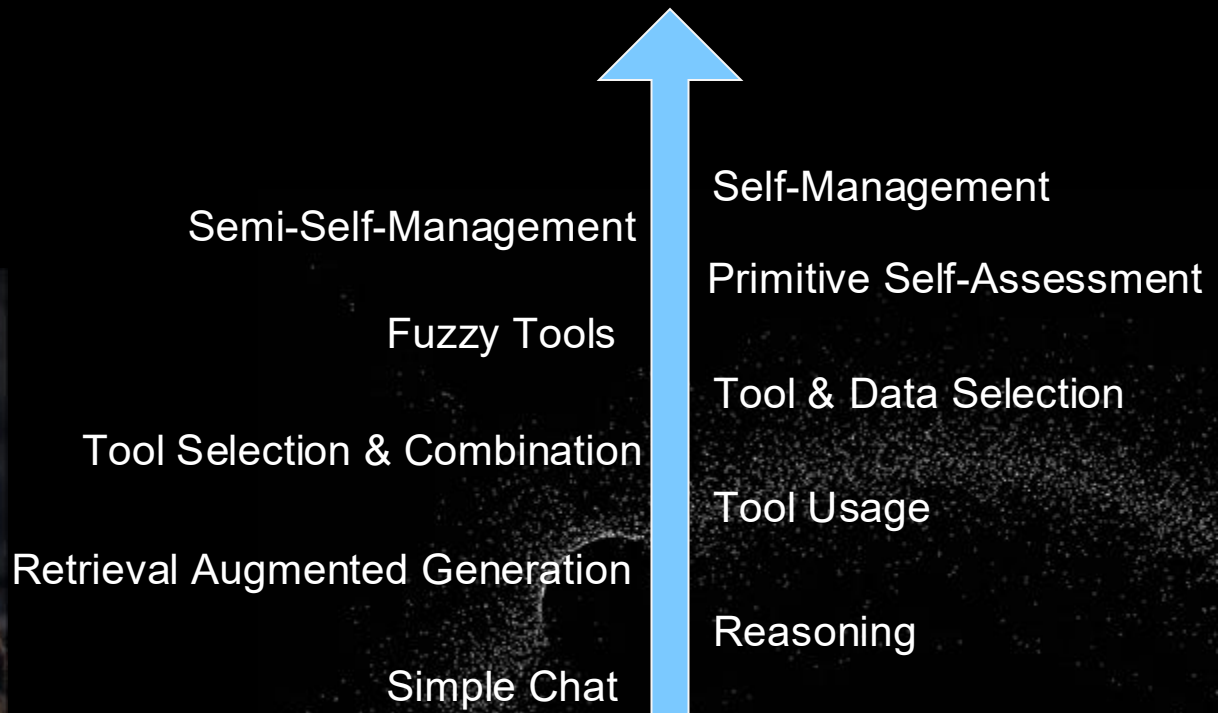
This is real now – but you have to build the cables



Levels Of Autonomy



Levels Of Autonomy



Levels Of Autonomy



Semi-Self-Management
Fuzzy Tools
Tool Selection & Combination
Retrieval Augmented Generation
Simple Chat

Self-Management
Primitive Self-Assessment
Tool & Data Selection
Tool Usage
Reasoning



Levels Of Autonomy



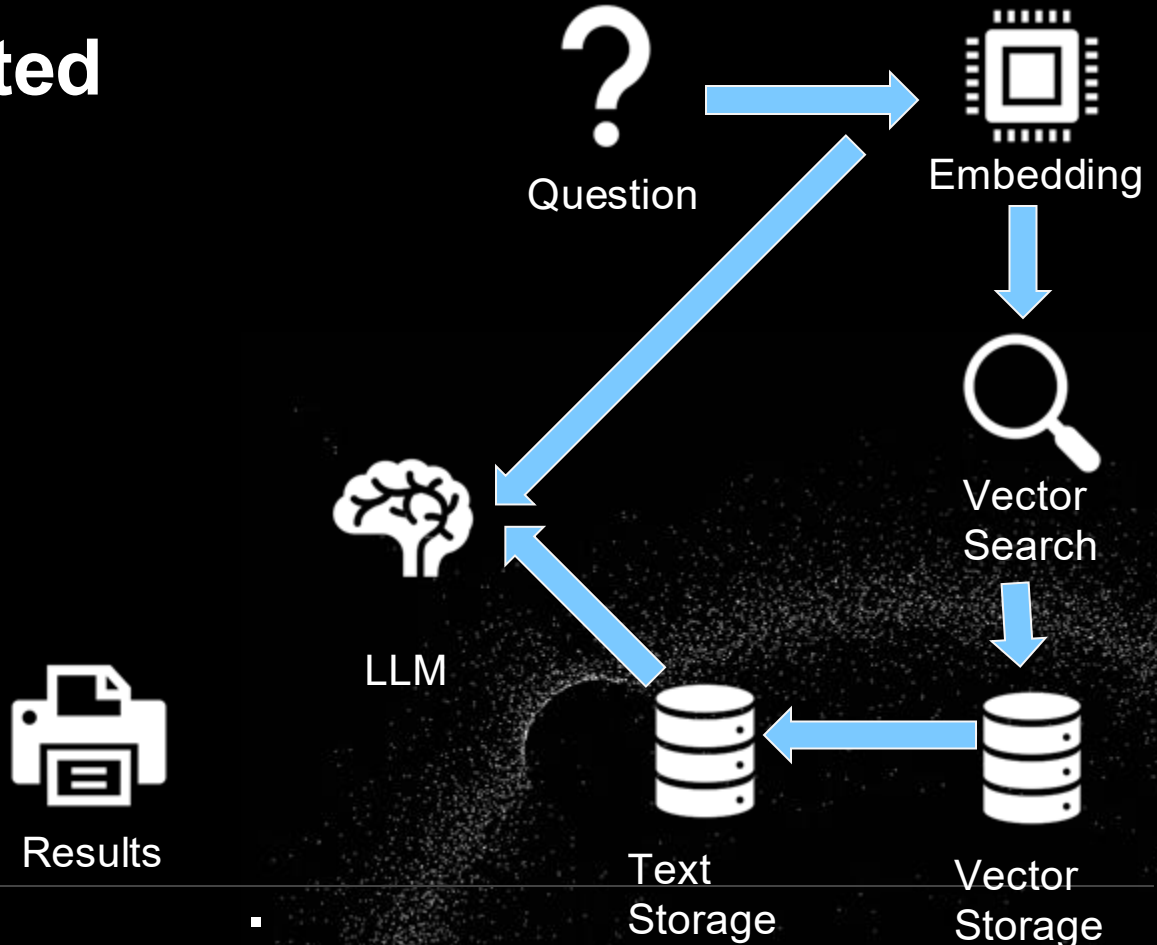
Semi-Self-Management
Fuzzy Tools
Tool Selection & Combination
Retrieval Augmented Generation
Simple Chat

Self-Management
Primitive Self-Assessment
Tool & Data Selection
Tool Usage
Reasoning



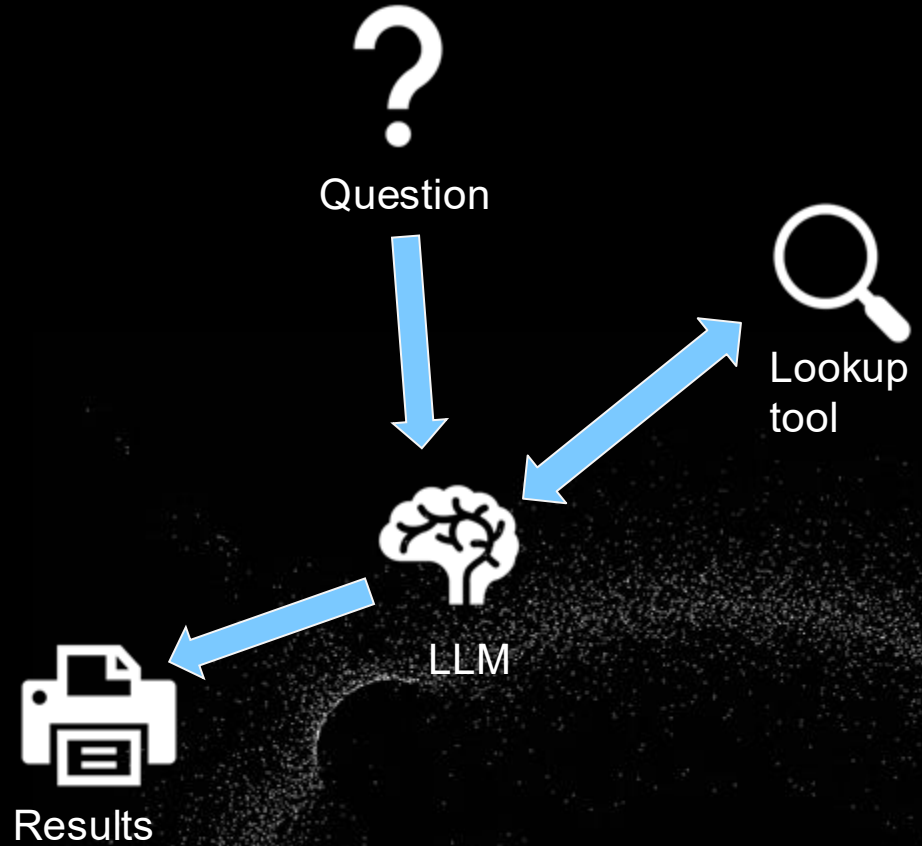
Retrieval Augmented Generation

- In this case the system always does the same things
- Works great for context-specific questions



Tool Usage: CEO view

- Now the tool is optional
- The LLM goes from just processing to reasoning
- The tool invocation and result processing is up to it



Tool Usage

- If you want the LLM to be able to pick a tool you need to make it optional
- This is when you REALLY need orchestration
- Now it can pick to trigger a tool or just directly answer

```
from typing import Annotated
import requests
import rigging as rg

def get_weather(city: Annotated[str, "The city name to get weather for"]) -> str:
    "A tool to get the weather for a location"
    try:
        city = city.replace(" ", "+")
        return requests.get(f"http://wttr.in/{city}?format=2").text
    except:
        return "Failed to call the API"
```

```
chat = (
    await
    rg.get_generator("gpt-4o-mini")
    .chat("How is the weather in london?")
    .using(get_weather)
    .run()
)
```

```
print(chat)
print(chat.last)
```

```
chat = (
    await
    rg.get_generator("gpt-4o-mini")
    .chat("who played Harry Potter?")
    .using(get_weather)
    .run()
)
print(chat)
print(chat.last)
```

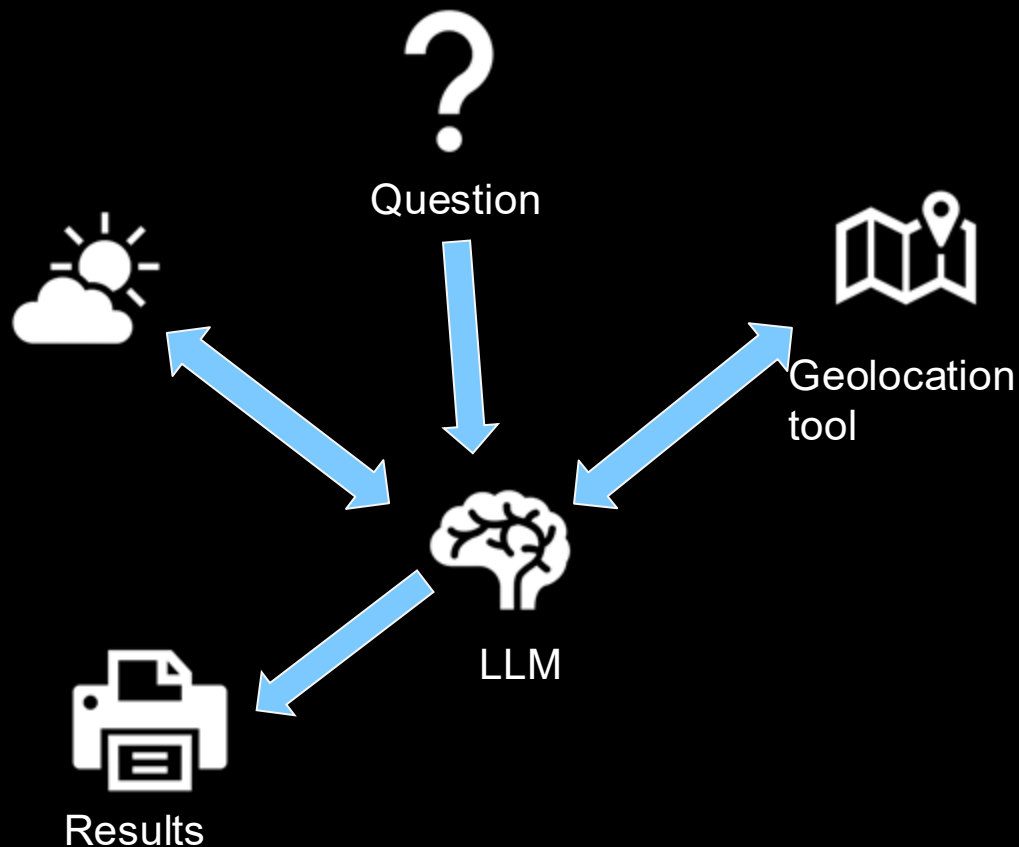
```
uuid=UUID('a5d5c278-3944-49d3-a24f-97f857296cc5') messages=[Message(role='user', parts=[], tool_calls=None, tool_call_id=None, content='How is the weather in london?'), Message(role='assistant', parts=[], tool_calls=[ToolCall(id='call_1Mzuoba7jn0YAAxZbZsGkd3Z', type='function', function=FunctionCall(name='get_weather', arguments={'city':'London'}))], tool_call_id=None, content=''), Message(role='tool', parts=[], tool_calls=None, tool_call_id='call_1Mzuoba7jn0YAAxZbZsGkd3Z', content='☀️ 47°F 🌬️ 6mph\n')] generated=[Message(role='assistant', parts=[], tool_calls=None, tool_call_id=None, content='The current weather in London is sunny, with a temperature of 47°F and a light breeze at 6 mph.')] metadata={} stop_reason='stop' failed=False
[assistant]: The current weather in London is sunny, with a temperature of 47°F and a light breeze at 6 mph.
uuid=UUID('89079ec6-7528-4d92-9968-8dc7f9981069') messages=[Message(role='user', parts=[], tool_calls=None, tool_call_id=None, content='who played Harry Potter?')] generated=[Message(role='assistant', parts=[], tool_calls=None, tool_call_id=None, content='Daniel Radcliffe played Harry Potter in the film series adapted from J.K. Rowling's novels.')] metadata={} stop_reason='stop' failed=False
[assistant]: Daniel Radcliffe played Harry Potter in the film series adapted from J.K. Rowling's novels.
```

Let's eat our veggies

What I am thinking about	How Rigging does it
Reliability	Parsing & Templating
Response Validation	Callbacks & Mapping
Scaling	Iterating and Batching
Tools	Tools
Storing & Exporting	Serialization

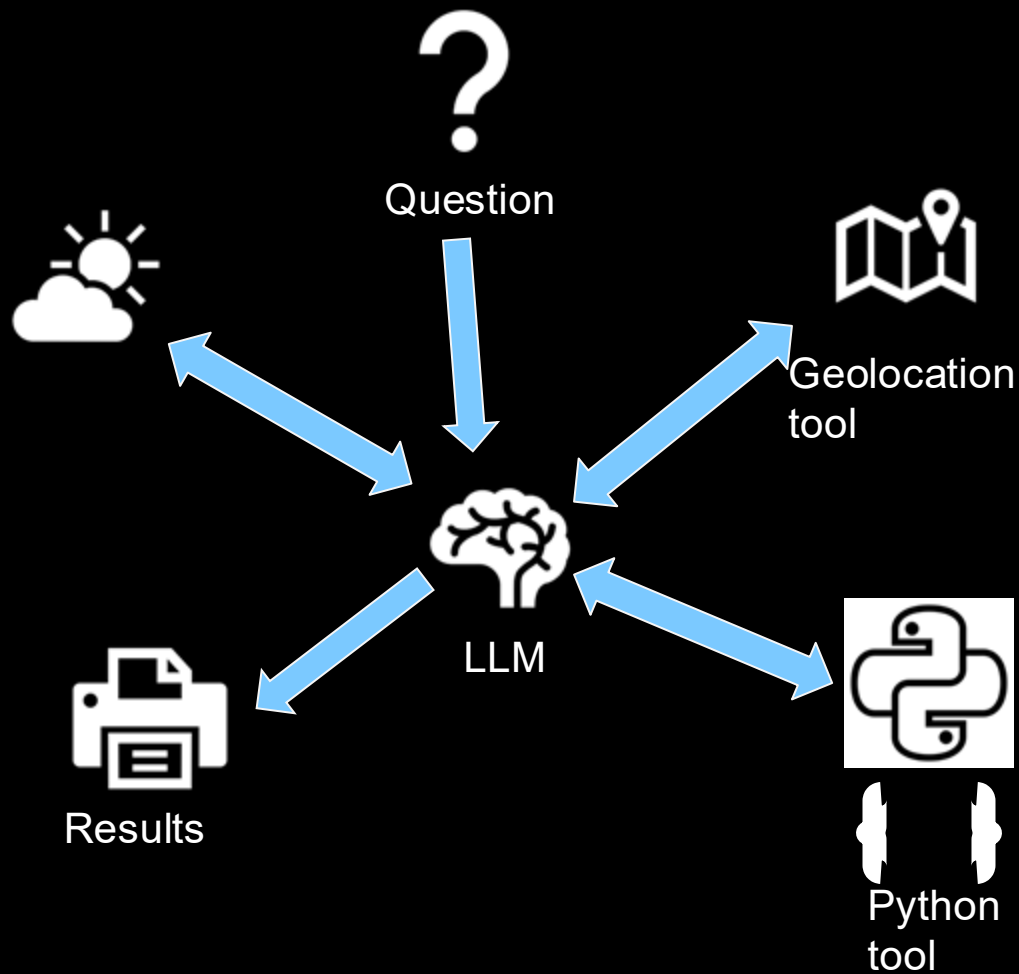
Tool Combinations: CEO view

- Now tools can be combined and activated together
- But again, the LLM is at the centre, calling the shots



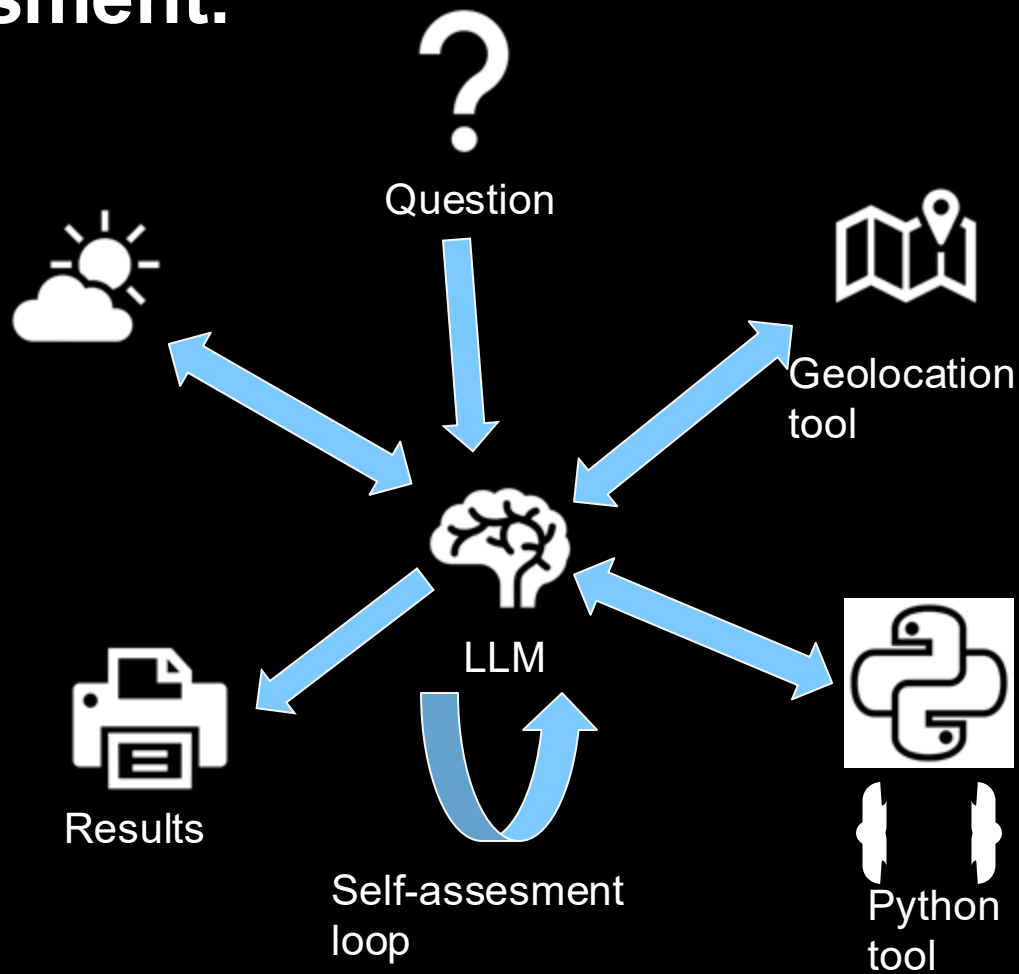
Fuzzy Tools: CEO View

- Fuzzy tools are more malleable
- Include a larger element of self-definition
- Good example is a basic python tool
- But again, the LLM is at the centre, calling the shots



Hardcoded Self-Assessment: CEO View

- Just a simple check for a certain type of XML tag
- Or a specific value
- Otherwise continuing in a loop



Let's get semi-autonomous

Information
Ingestion

Remote
Analysis

Data Enricher

Pivot
Discovery

Self
Management

Self-focus

Analysis
Strategies

Concurrent
Splitoffs

Knowledge
Management

Graph
Memory

Pivot
Classifier

Completeness
Explorer

Closing thoughts

Thank you