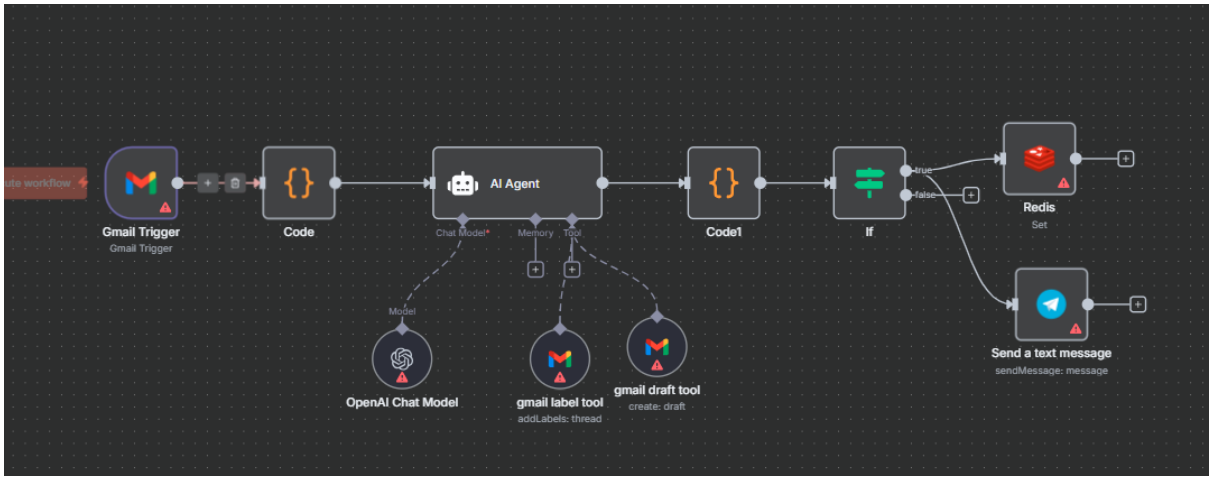
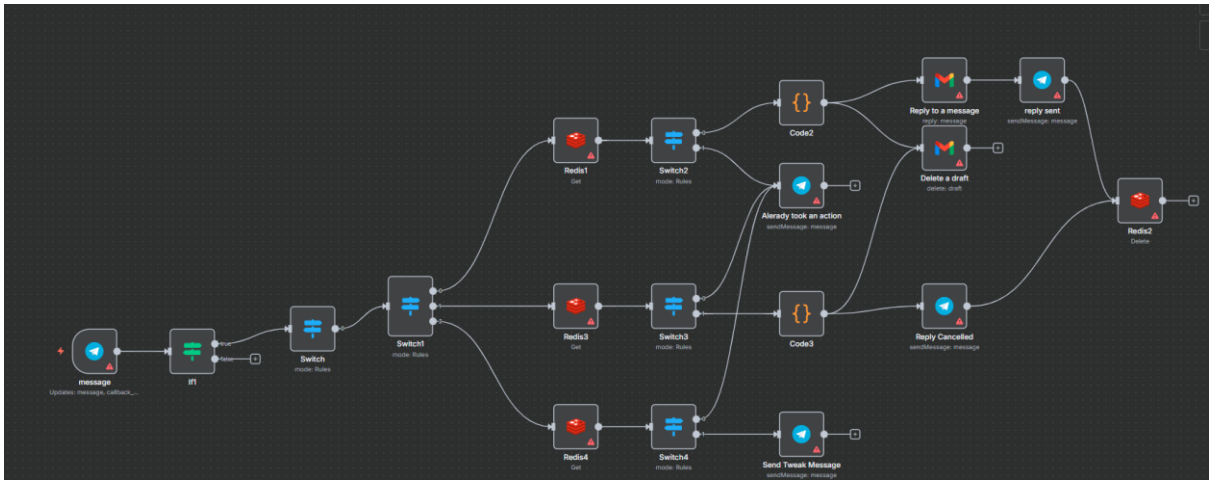
Recap:

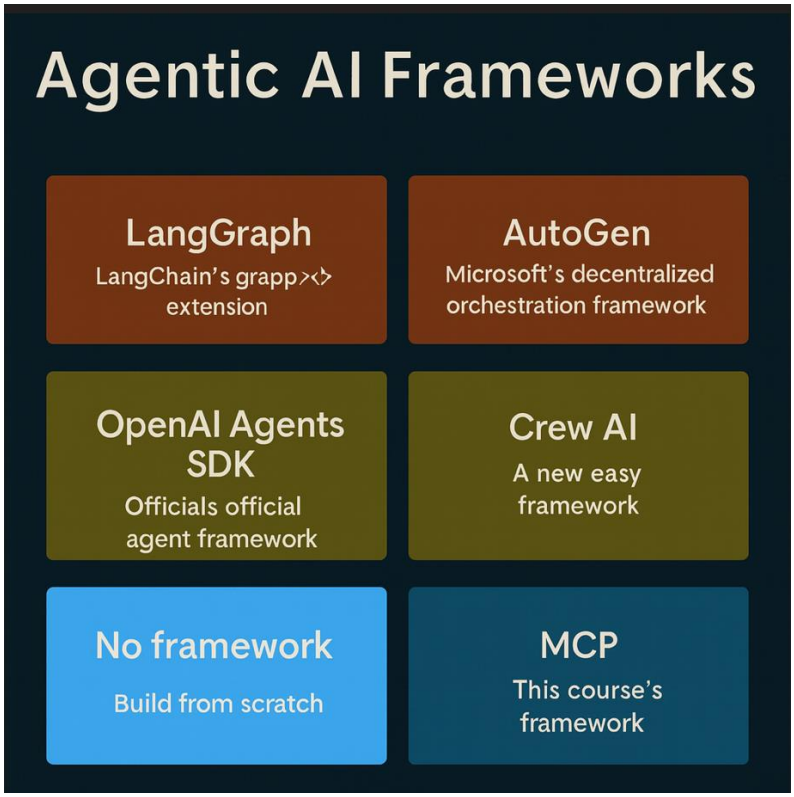
N8N workflow with LLM-tools-resources + integration with gemini + chatgpt openai + telegram + redis + custom code + js





Build Agentic workflow and options available:

Step 0: Agentic AI Frameworks:



Complexity1:

- **No Framework** using API's like we did in lab1 / lab2 – connect directly to LLM's - Custom-built agents from scratch using raw API calls and logic.
- **MCP** – Model Context Protocol by Anthrobic – it's a way models can connect to each other – so we don't need any glue code.

Complexity2:

- **OpenAI Agents SDK :** OpenAI's official toolkit for building and deploying production agents.
- **Crew AI –** low code- config via yaml files : A lightweight agent-coordination library focused on simplicity and teamwork.

Complexity 3: Heavy weight and learning curve, and great power

- LangGraph: A graph-based extension of LangChain for orchestrating agent workflows.
- AutoGen : Microsoft's decentralized multi-agent orchestration framework for collaboration.

Step1: Clone my Repo – for notebook – https://github.com/TEJAPS/agentic-notebooks.git

Step2: download cursor - https://cursor.com/ and open cloned folder

Step3: install UV https://docs.astral.sh/uv/getting-started/installation/

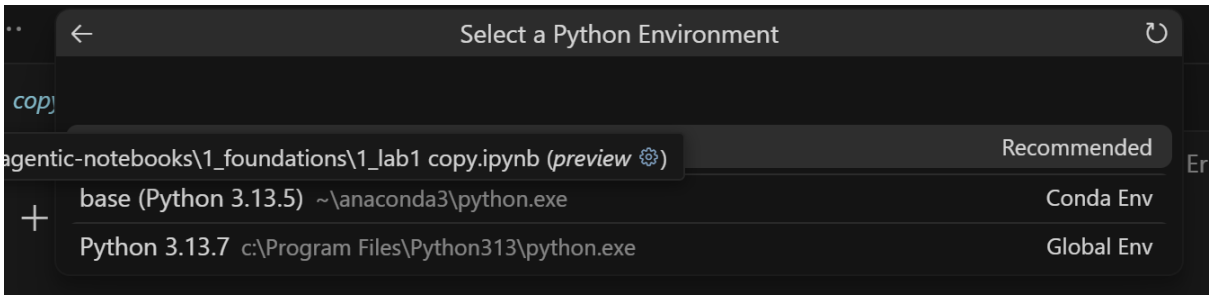
Cmd: powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"

```
PS C:\Users\tejap> powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
Downloading uv 0.8.22 (x86_64-pc-windows-msvc)
Installing to C:\Users\tejap\.local\bin
  uv.exe
  uvx.exe
  uvw.exe
everything's installed!

To add C:\Users\tejap\.local\bin to your PATH, either restart your shell or run:

    set Path=C:\Users\tejap\.local\bin;%Path%   (cmd)
    $env:Path = "C:\Users\tejap\.local\bin;$env:Path"   (powershell)
PS C:\Users\tejap>
```

Kernel status:



Step 4: uv sync (this creates isolated env, with python 3.12)

Run python file: uv run <filename>

Step 5: buy open ai key

https://platform.openai.com/settings/organization/api-keys

Step 6: add key in env variable

OPENAI_API_KEY=sk-proj-*****

Step 7: Lab 1 – make an agentic workflow – only Open AI

Post lab discussion:

Discussion1:

```
response = openai.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
)
```

vs

```
const response = await openai.responses.create({
  model: "gpt-5",
  input: "Write a one-sentence bedtime story about a unicorn."
});
```

Discussion 2: what all models of GPT available for me:
https://platform.openai.com/docs/models

Step 8: Agents and Agentic Patterns and workflow design patterns:

AI Agents: these are programs where LLM outputs control the workflow.

AI solution involves any or all of these:

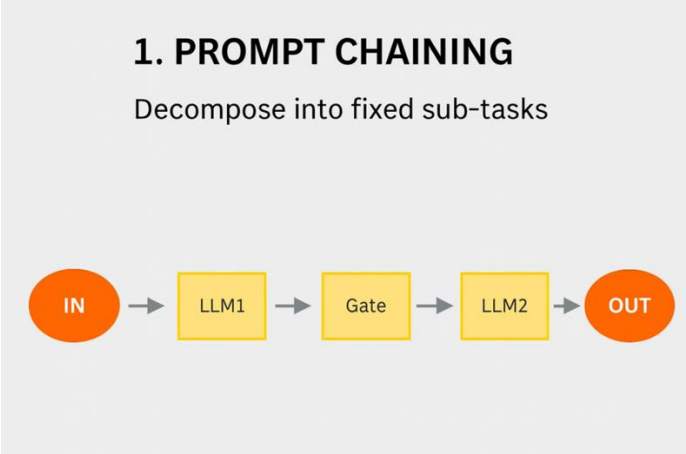
- Multiple LLM calls
- LLMs with ability to use tools
- Environment where LLMs interact
- Planner to coordinate activities
- Autonomy

Anthropic distinguishes 2 types:

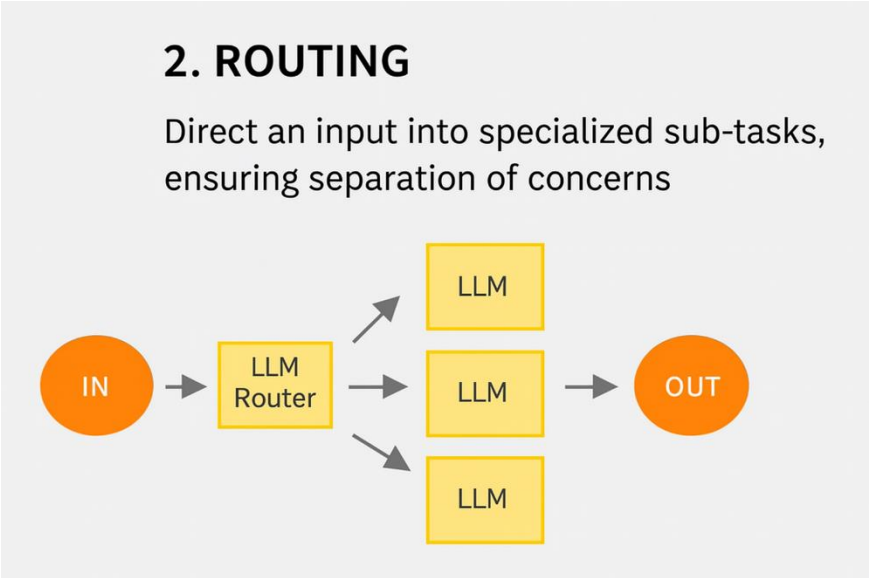
- Workflow are systems where LLMs and tools are orchestrated through predefined code paths.
- Agents are systems where LLMs dynamically direct their own processes and tool usage maintain control over how they accomplish tasks

Workflow design patterns

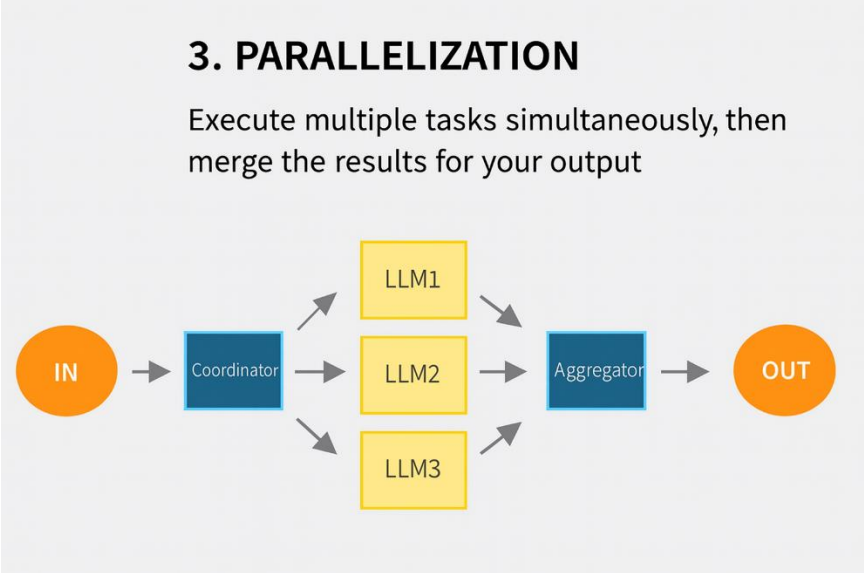
1. Prompt chaining: “Decompose the main goal into smaller, sequential sub-tasks where each LLM’s output becomes the next input.”



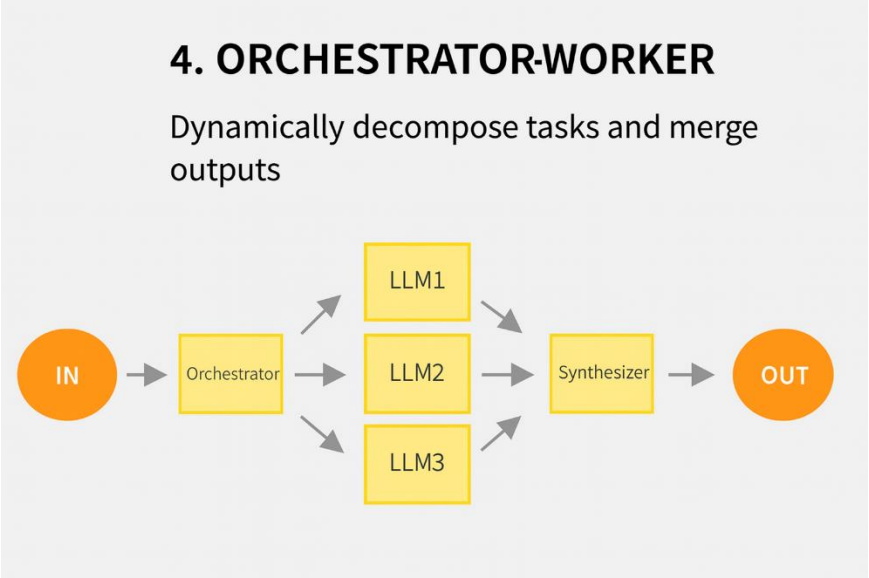
2. **Routing: classify which to call and when - separation of concern -** "Select the right expert LLM or workflow path based on the input type, ensuring each sub-task is handled by the most suitable model."



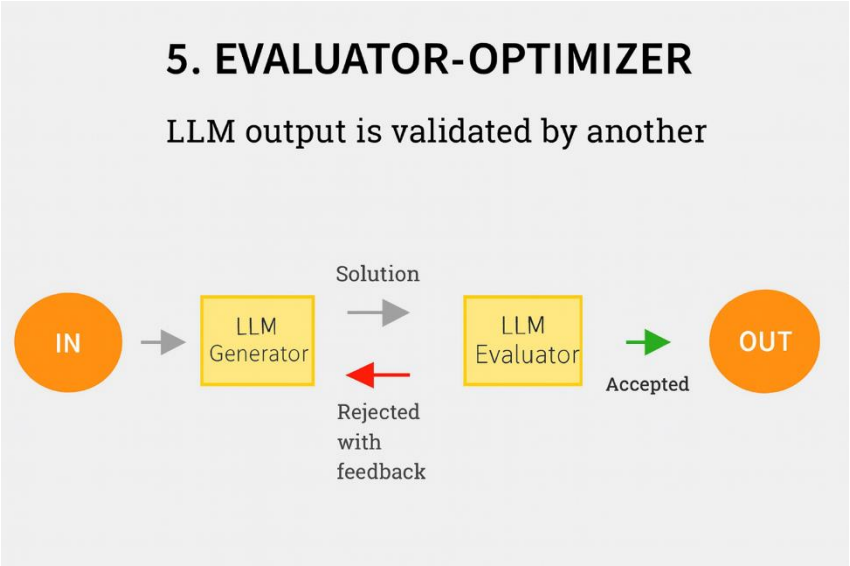
3. **Parallelization:** blue box is our code - "Divide a task into smaller independent parts, run them concurrently, and aggregate their results for the final output."



4. **Orchestrator worker:** “An orchestrator dynamically assigns complex subtasks to worker LLMs and synthesizes their outputs into a unified result.”

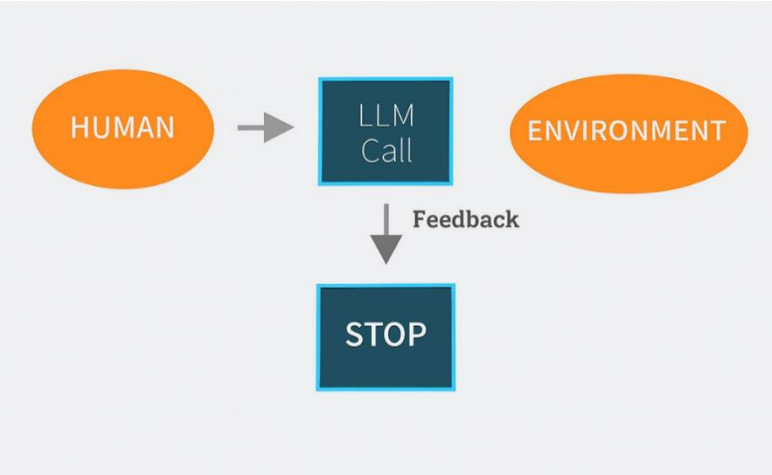


5. **Evaluator optimizer:** “One LLM generates solutions while another evaluates and refines them through feedback until an optimal result is achieved.”



In Contrast,

Agents — "Open-ended systems that continuously interact with their environment through feedback loops, adapting actions dynamically without a fixed path."



Refine once again:

1. **Agents**: Agents are **autonomous decision-makers** that **decide what to do next**, often using tools or calling other agents.
   They maintain *state* (memory, goals, history) and perform *reasoning loops*.
   Example:
   - "ResearchAgent" decides whether to use Google Search, Wikipedia API, or internal database next.
   - "Orchestrator" that assigns subtasks to worker LLMs.
2. **Tools**: Tools are **non-intelligent functions or APIs** the agent can call to act in the world.
   They do not reason — they **just perform a specific operation**.
   Do *one specific* action (e.g., query DB, send email, call API).
   Example:
   - "GoogleSearchTool" → calls Google Search API.
   - "PythonREPLTool" → executes Python code.
   - "DatabaseTool" → runs SQL query.
3. **Resources**: Resources are **external data sources or persistent stores** — not executable logic, but data that can be read or written. They hold *information*, not *logic*. Agents or tools read/write from them.

   Examples:

   - "ChromaDB" or "Pinecone" → Vector store resource.
   - "PostgreSQL database" → persistent structured resource.
   - "MemoryBuffer" → short-term memory for conversation.
4. Non-Agent Nodes (Plain LLM calls or workflow steps): A simple **LLM node** that performs a fixed subtask without autonomy.
   It doesn't decide what happens next — it's just a single-pass computation.

- No decision-making or feedback loop.
- Predefined input/output.
- Used in deterministic pipelines (Prompt Chaining, Parallelization).

Example:
LLM1 → summarize, LLM2 → extract entities, LLM3 → classify sentiment

Type	Role	Has Reasoning?	Stateful?	Example
Agent	Decides next action	✅ Yes	✅ Yes	ResearchAgent, Orchestrator
Tool	Executes fixed function	❌ No	❌ No	GoogleSearchTool, SQLTool
Resource	Provides or stores data	❌ No	✅ Often	Pinecone, VectorStore
LLM Node	Fixed subtask	❌ No	❌ No	Summarizer, Extractor

Risks of Agent Frameworks:

- Unpredictable path
- Unpredictable output
- Unpredictable costs
- Monitor
- Guardrails: ensure your agents behave safely, consistently, and within your intended boundaries

Step 9: Lab 2: Agentic Ai with Multi types of models

Model 1: OpenAI: gpt-4o-mini (gpt-4o,o1,o3-mini) – key we have as part of Lab 1 – key working

Generate keys for below:

Model 2: Anthrophic: Claude-3-7-Sonnet - https://console.anthropic.com/dashboard.

Need 5$ minimum.

Model 3: Google: Gemini-2.0-flash – beta version

Model 4: Deepseek - https://platform.deepseek.com/api_keys. Free tier not available

Model 5: groq: open source LLMs including Llama3.3 - https://console.groq.com/keys

Model 6: Ollama – local service

Install it - https://ollama.com/, once installed and started it should be running on http://localhost:11434/

Recommended max model is llama3.2 or small variants of qwen, gemma, phi or deepseek

Available ollama model's page: https://ollama.com/search

**Restart your cursor for it to detect, ollama app running on our system.

Post lab discussion:

- Which pattern was it?
- Try completing the exercise.

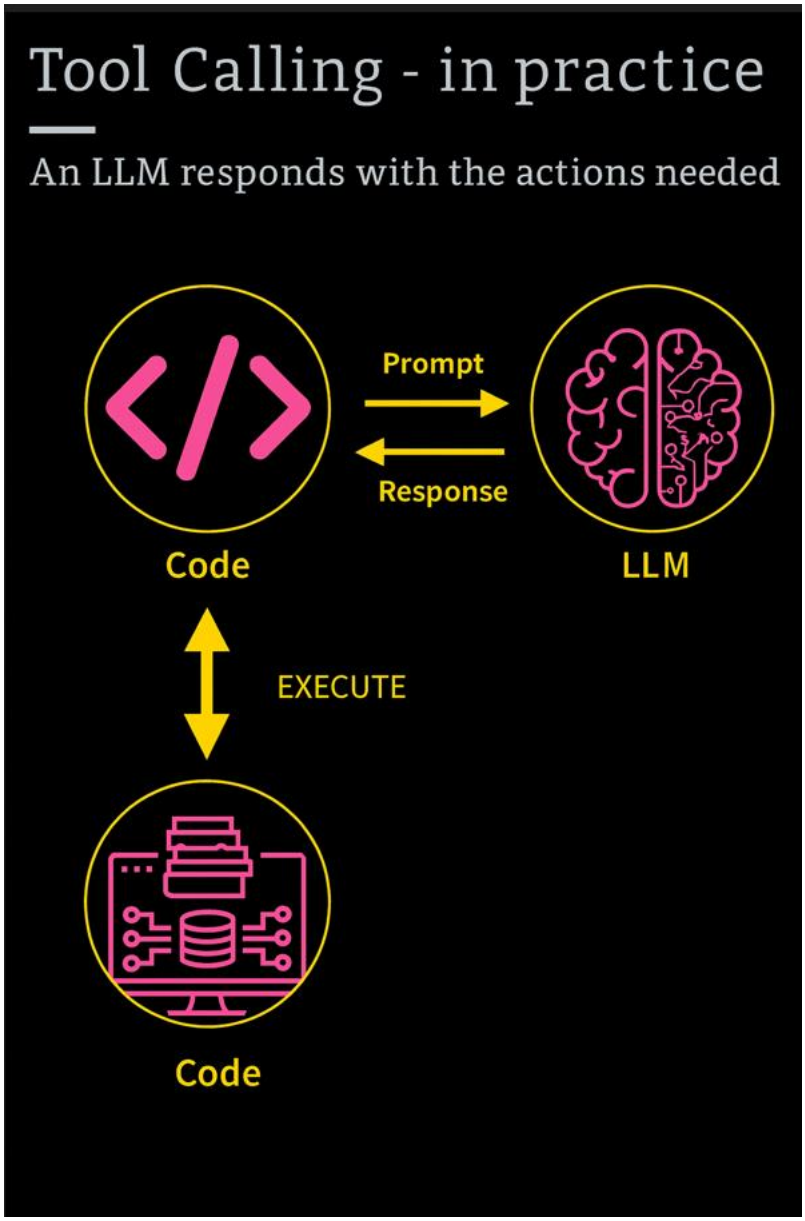
Step 10: Terminology:

Resources: provide LLM with resources to improve its expertise

i.e. shoving data relevant to the question into the prompt.

There are techniques like RAG to get really smart at picking **relevant context.**

Tools: give LLMs autonomy: give LLM power to carry out actions like "query a database" or "message other LLMs".



Step 11: Lab 3 – Feedback, Evaluator and optimizer methodology

11.1: download your LinkedIn as pdf and save in the me folder

https://www.linkedin.com/in/sai-teja-polisetty-92a7aa143/

and write something about yourself into the summary.txt.

11.2: proceed with lab steps

**Gradio : documentation - https://www.gradio.app/guides/sharing-your-app#hosting-on-hf-spaces

Step 12: Lab 4 – First Project using tools (in previous lab we used resources) – Career Conversation

12.1: Visit: https://pushover.net/ ,

- Copy the user key, and keep in .env file with key **PUSHOVER_USER**
- Create application: give some name
- Copy the token and keep in .env file with key **PUSHOVER_TOKEN**

12.2: perform the steps in the lab, talk to tools

Step 13: Deploy on hugging face

Before you start: remember to update the files in the "me" directory - your LinkedIn profile and summary.txt - so that it talks about you! Also change `self.name = "Sai Teja"` in `app.py`..

Delete the readme if any

1. Visit https://huggingface.co and set up an account

2. From the Avatar menu on the top right, choose Access Tokens. Choose "Create New Token". Give it WRITE permissions - it needs to have WRITE permissions! Keep a record of your new key.

3. In the Terminal, run: `uv tool install 'huggingface_hub'` to install the HuggingFace tool, then `hf auth login` to login at the command line with your key. Afterwards, run `hf auth whoami` to check you're logged in

4. Take your new token and add it to your .env file: `HF_TOKEN=hf_xxx` for the future

5. From folder where you have your app.py, enter: `uv run gradio deploy`

6. Follow its instructions: name it "career_conversation", specify app.py, choose cpu-basic as the hardware, say Yes to needing to supply secrets, provide your openai api key, your pushover user and token, and say "no" to github actions.

To delete your Space in the future:

1. Log in to HuggingFace

2. From the Avatar menu, select your profile

3. Click on the Space itself and select the settings wheel on the top right

4. Scroll to the Delete section at the bottom

5. ALSO: delete the README file that Gradio may have created inside this 1_foundations folder (otherwise it won't ask you the questions the next time you do a gradio deploy)