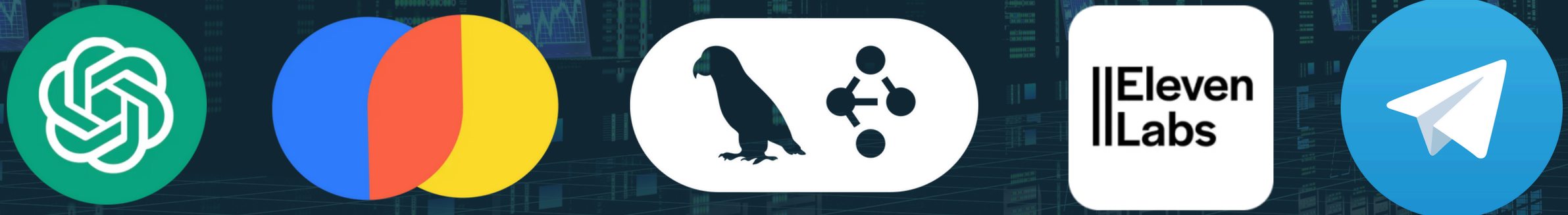
The Neural Maze

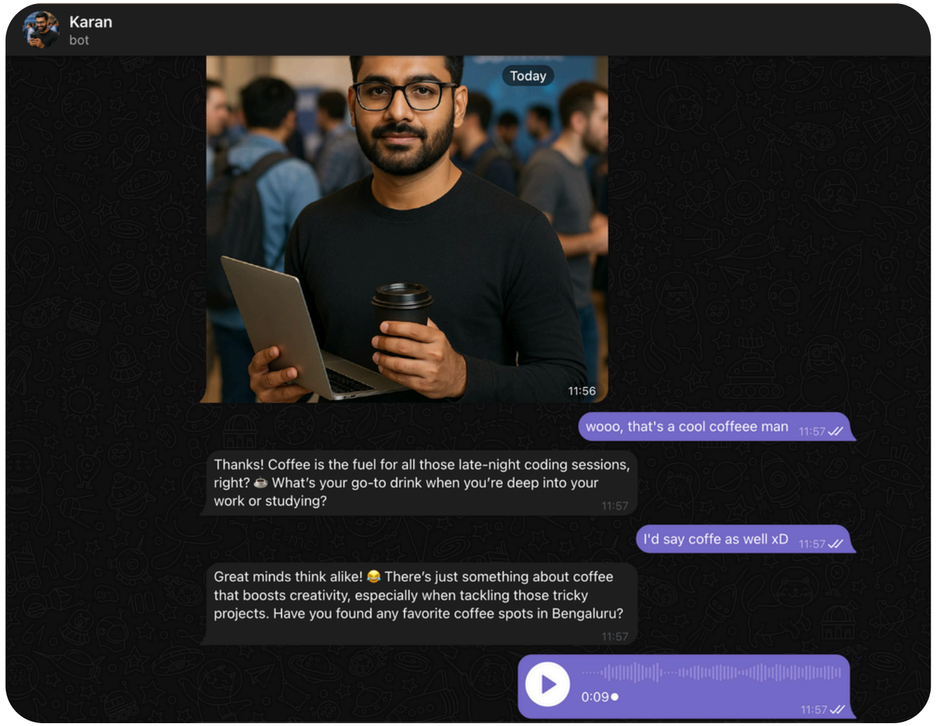
MULTIMODAL AGENTS

LANGGRAPH WORKSHOP



Project Description

- In this workshop, we're building a **multimodal Telegram agent** from the ground up.
- We'll explore how to use **LangGraph** to create **agentic workflows**.
- We'll rely on **OpenAI models** to handle **text**, **audio**, and **image** messages.
- For **long-term memory** we're using **Chroma**.
- And to give our agent a distinct voice, we'll bring in **ElevenLabs**.

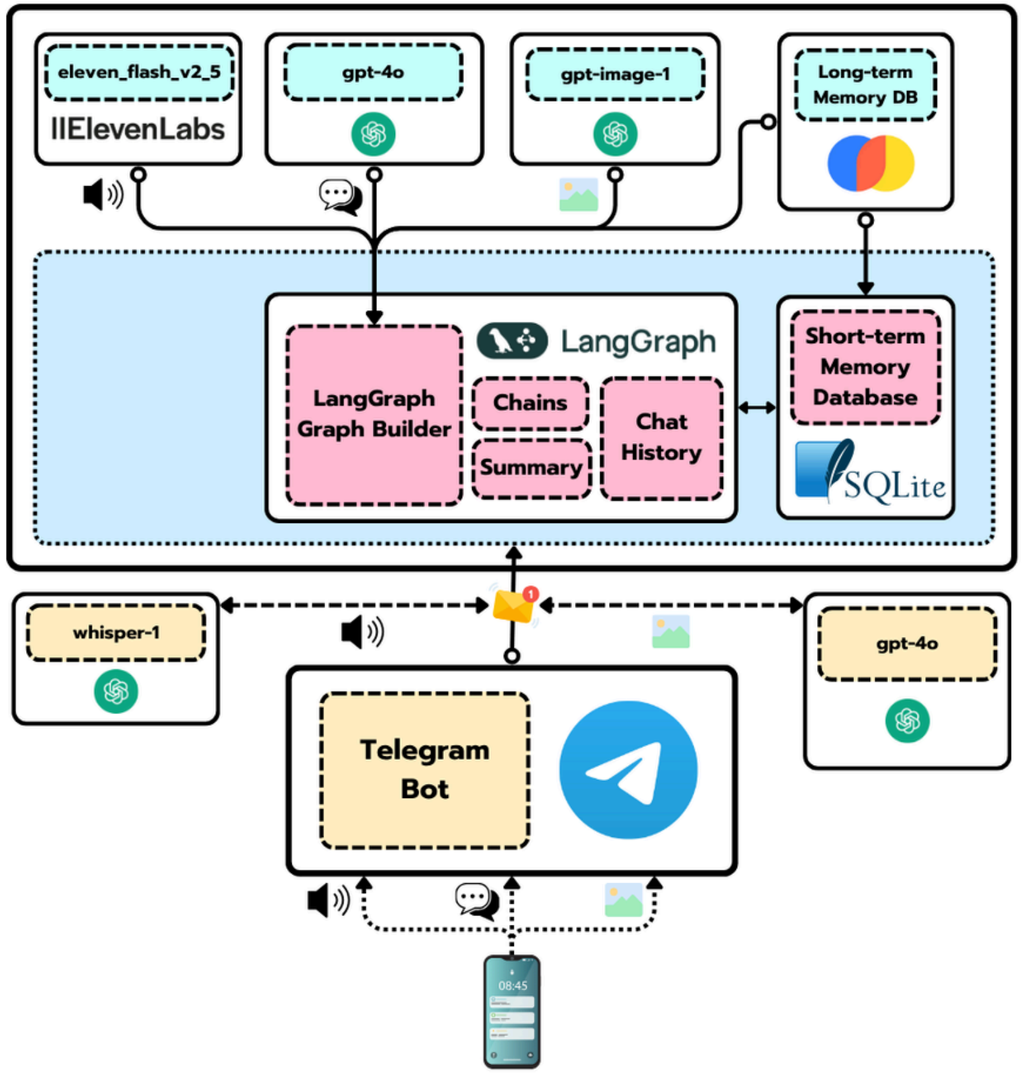


IIElevenLabs

Project Description

The system is built of **7 components:**

- **Agent Workflow**
- **Agent Memory**
- **STT (Speech-To-Text) Module**
- **TTS (Text-To-Speech) Module**
- **Image Understanding**
- **Image Generation**
- **Telegram Bot**

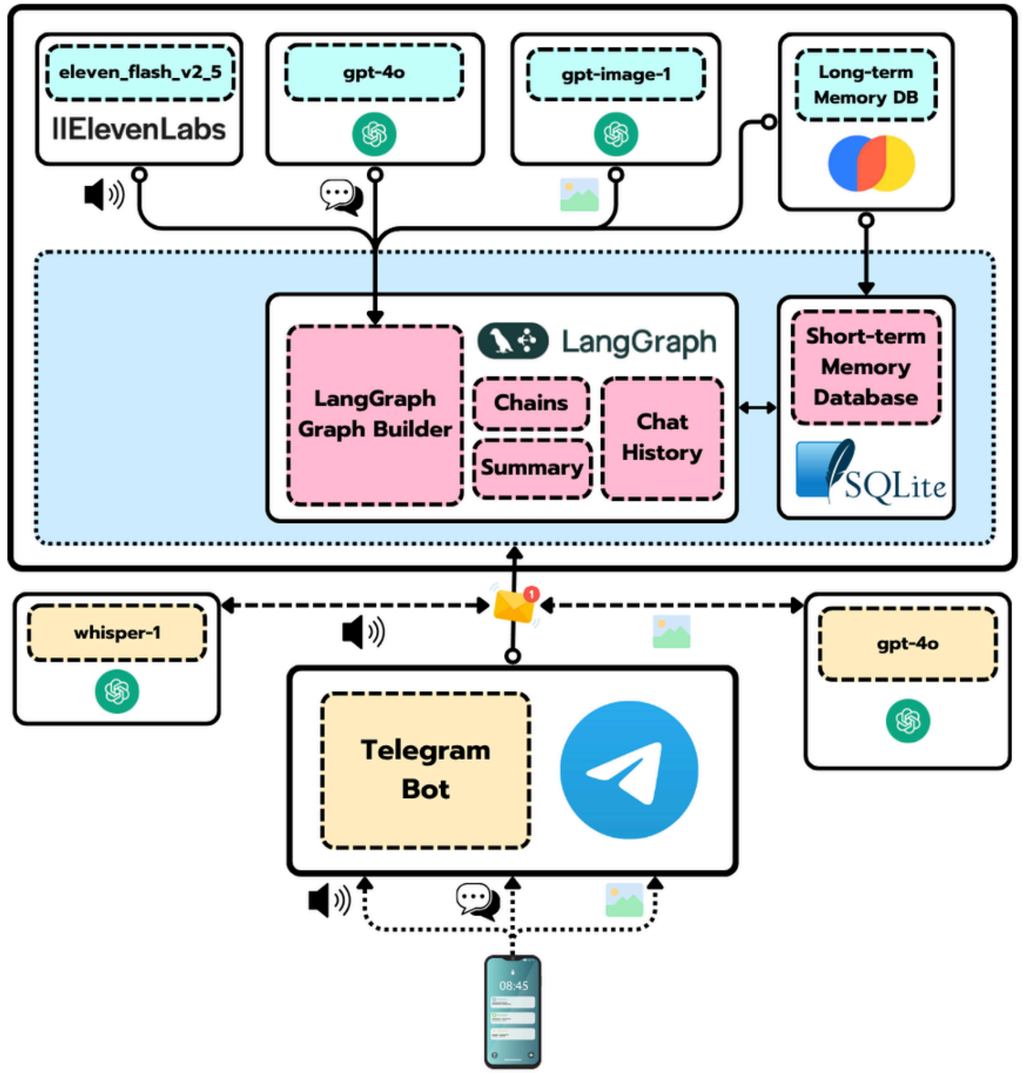


Project Description

This project is structured in two parts:

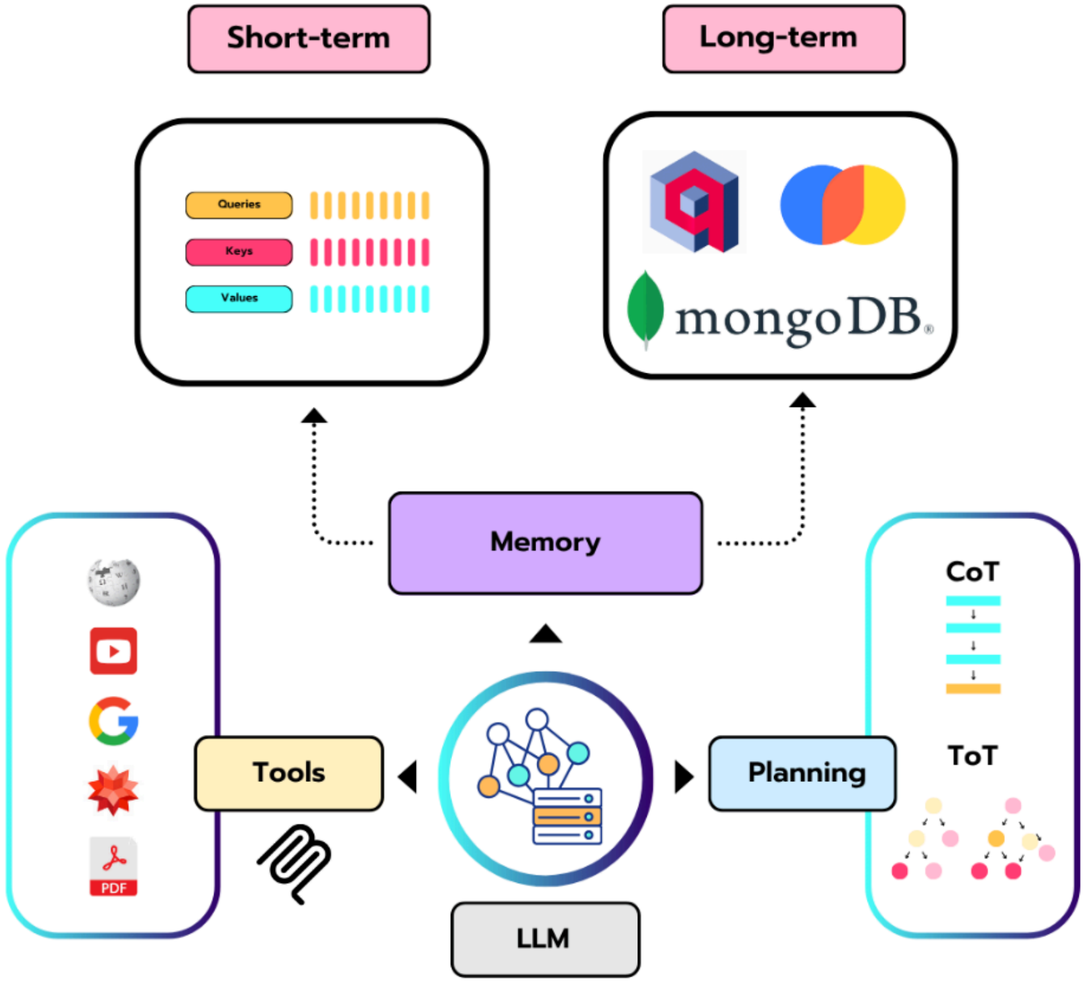
- **Part 1 (Free)**
  - 📅 - October 8th
  - Covered in this Workshop
  - Code and video free
  - Based on Collab notebooks

- **Part 2 (Premium Subscribers)**
  - 📅 - October 15th
  - **NOT** covered in this Workshop
  - Full application
  - CI / CD → Deployment to AWS



Before we begin … what is an Agent? 🤔

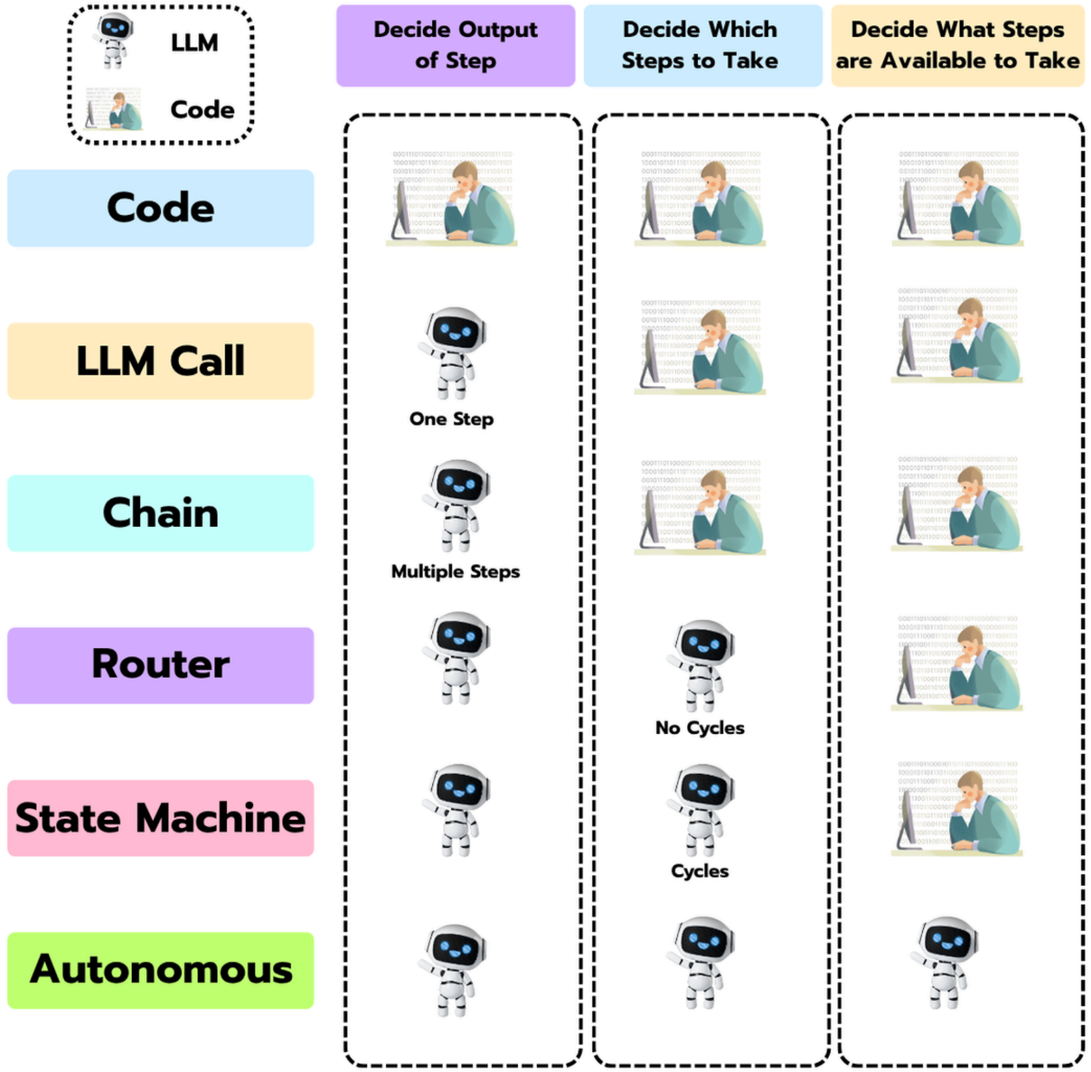
- **LLM -** The Agent's "brain"
- **Planning -** prompt techniques, workflows, etc.
- **Memory -** short-term and long-term
- **Tools -** External resources or actions an agent can use



LangGraph Crash Course

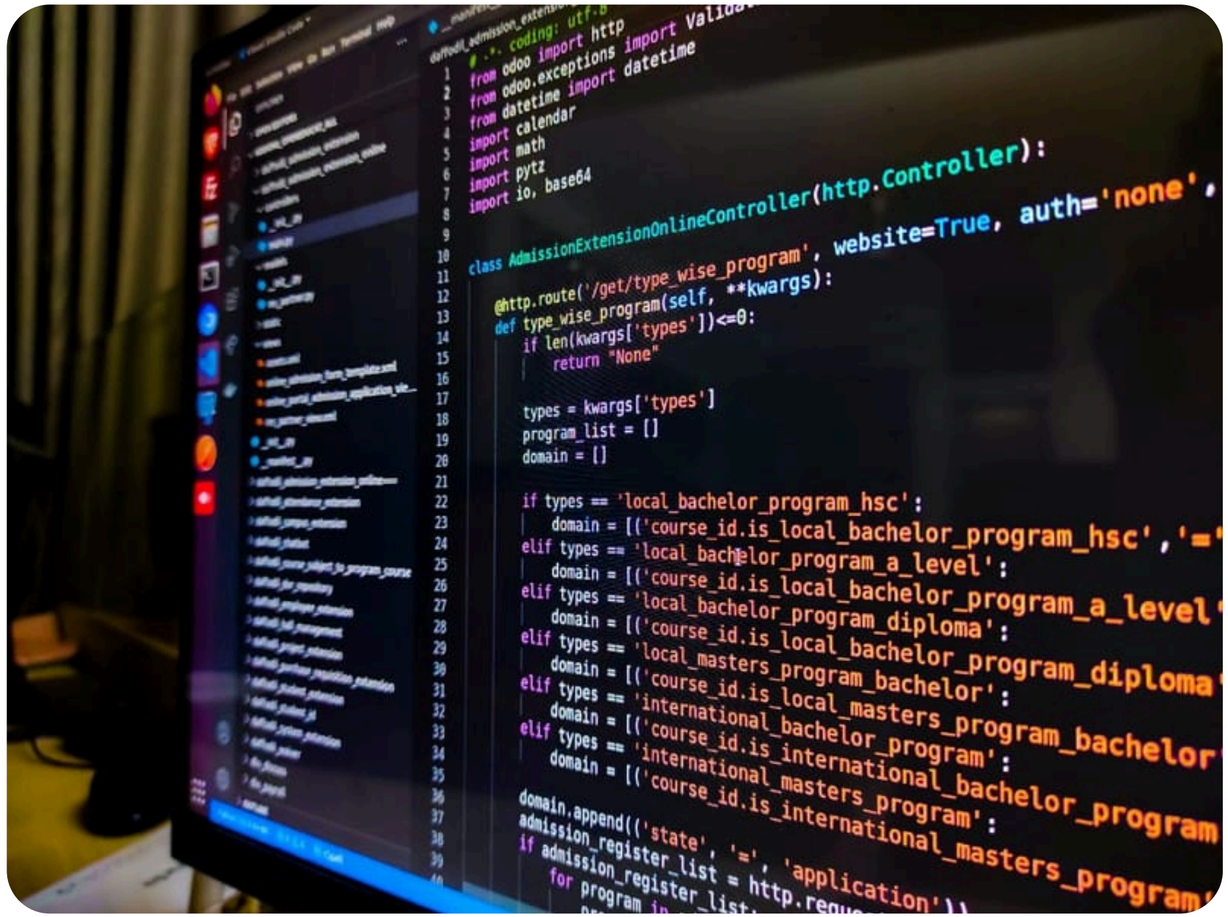
There are **6 levels of autonomy** in LLM applications:

- Code
- LLM Call
- Chain
- Router
- State Machine
- Autonomous



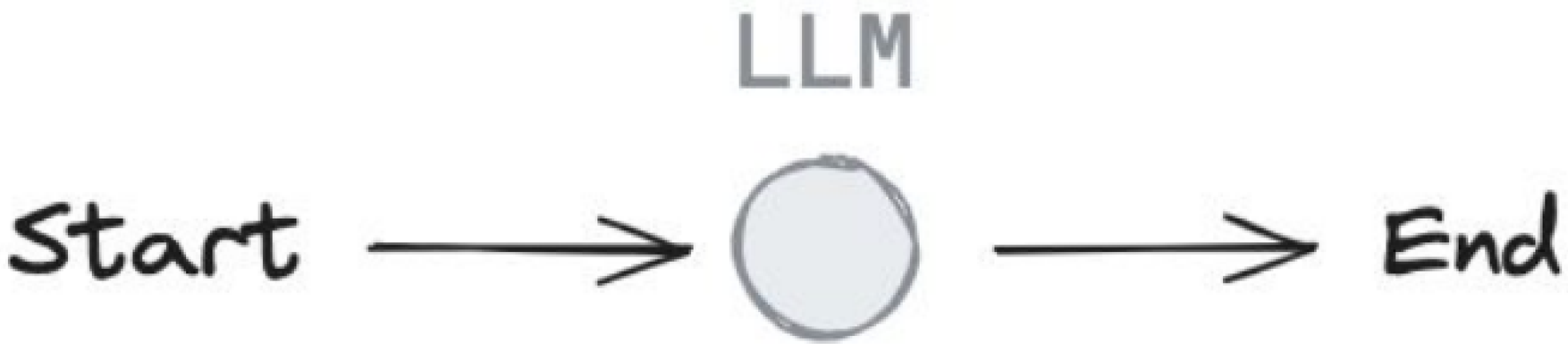
LangGraph Crash Course

The **first level** (it's not even an LLM application) is just "traditional code"



LangGraph Crash Course

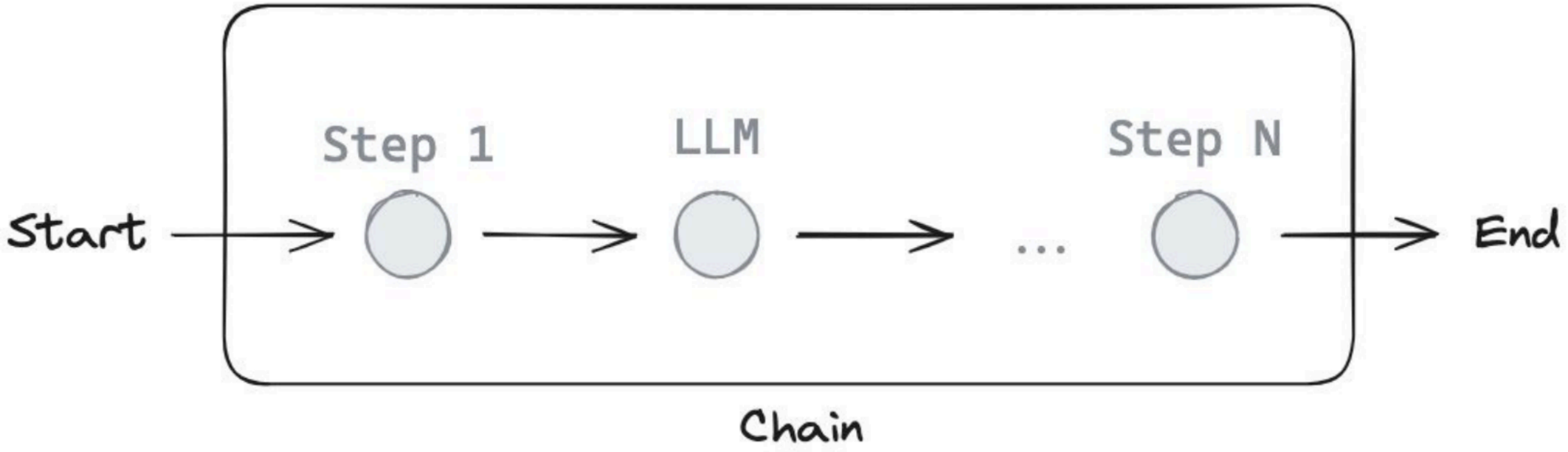
The **second level** is reduced to single LLM Calls (not bad, but fairly limited)



LangGraph Crash Course

The **third level** introduces **Chains** (that was the original goal of **LangChain**).

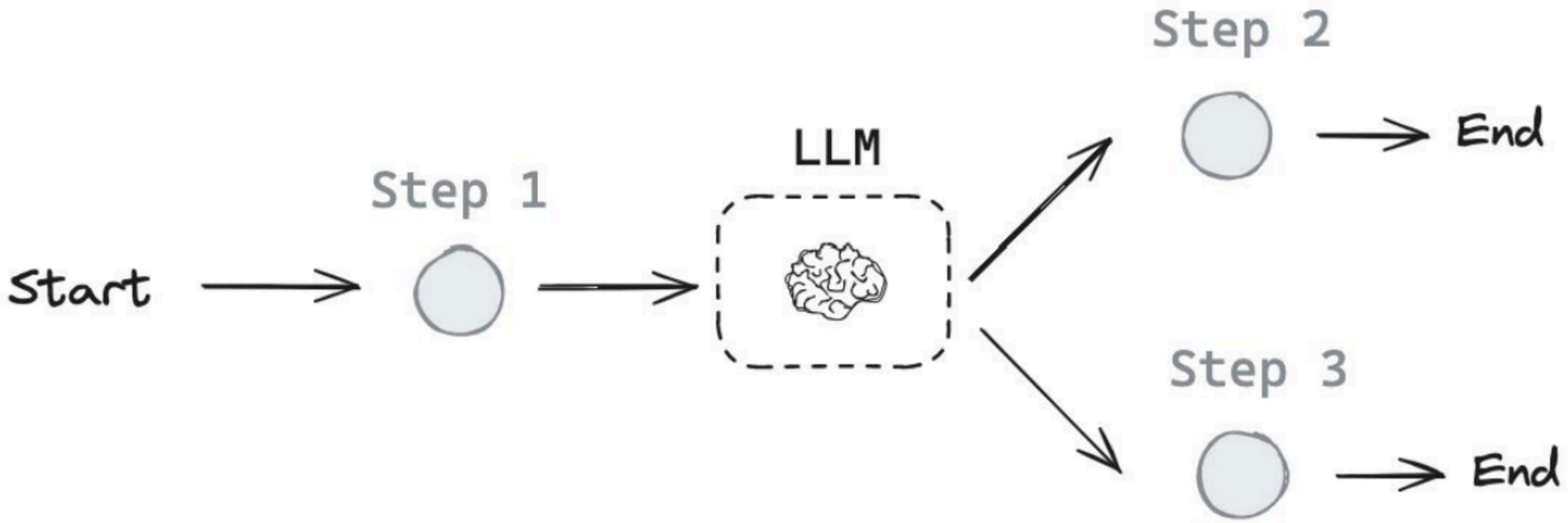
Chains are a **set of steps** defining a **control flow**.



LangGraph Crash Course

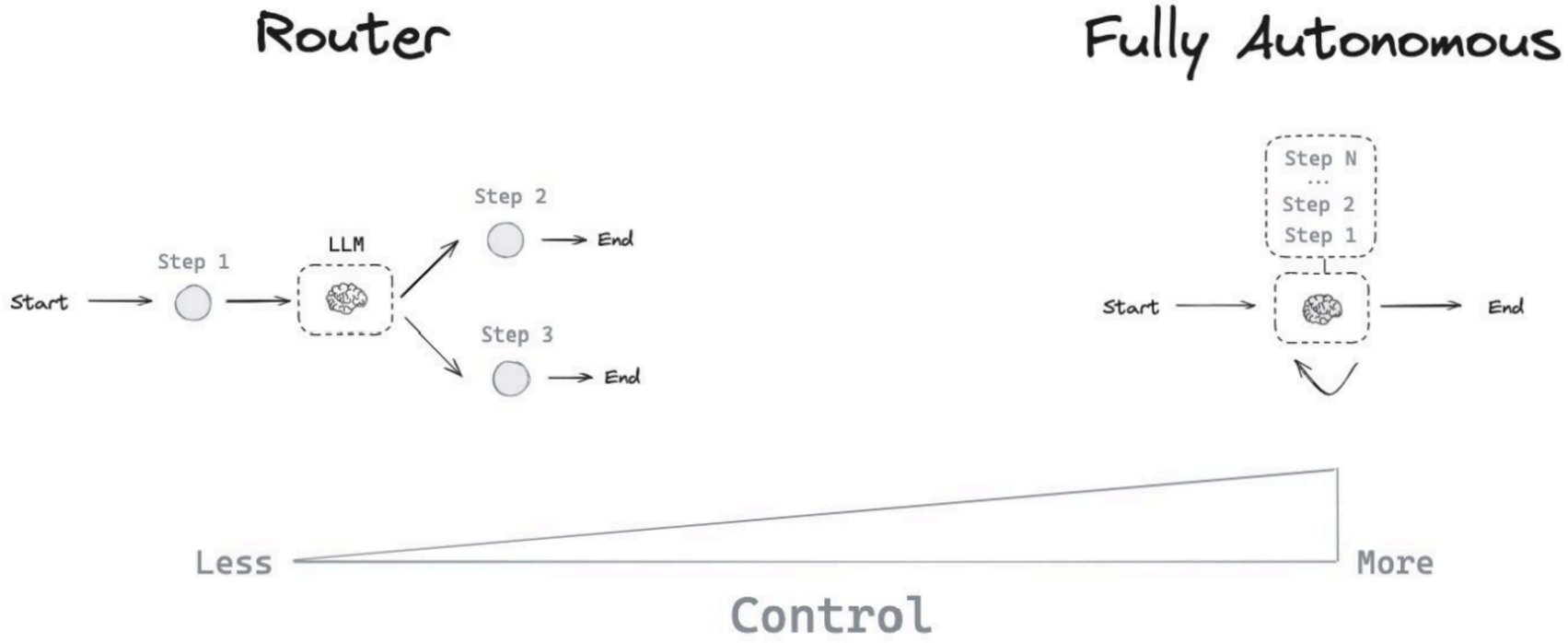
The **fourth level** introduces **the Router Pattern**

This pattern enables LLMs to pick their **own control flow**



LangGraph Crash Course

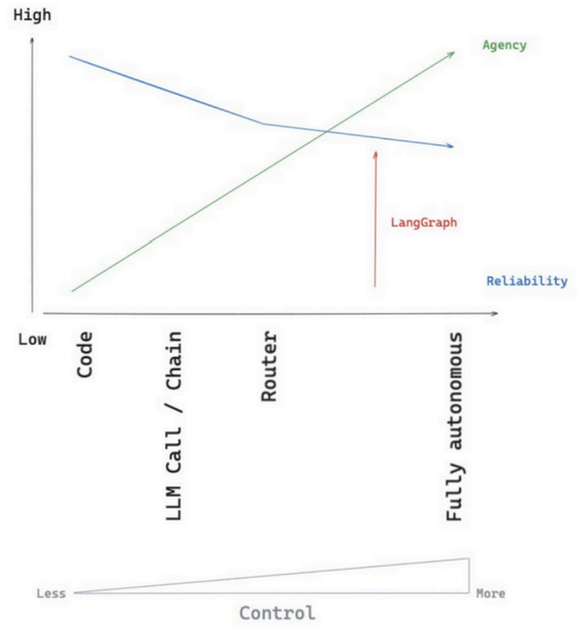
Higher levels (5th and 6th) point in the direction of semi-autonomous / fully autonomous agents



LangGraph Crash Course

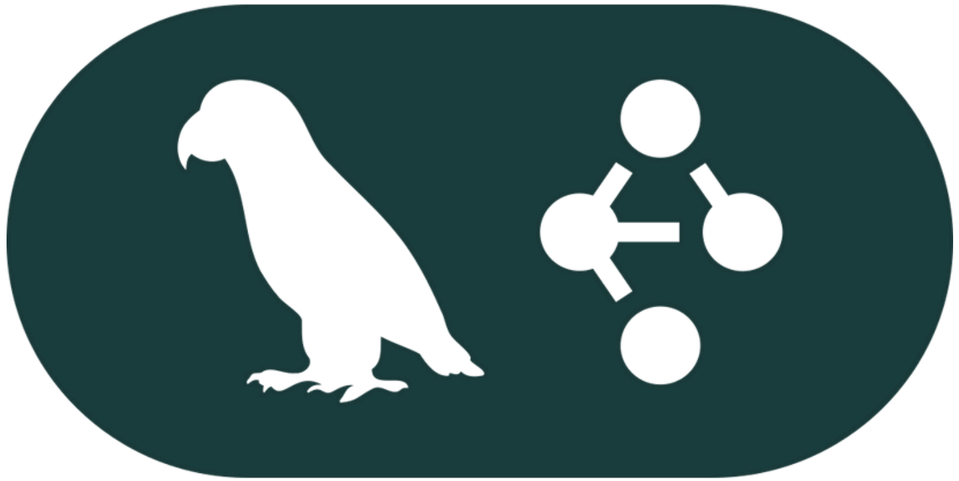
For our **Telegram Agent**, we need to find a tradeoff between **Flexibility** and **Reliability**.

LangGraph will become handy for that as it helps us **bend the reliability curve**



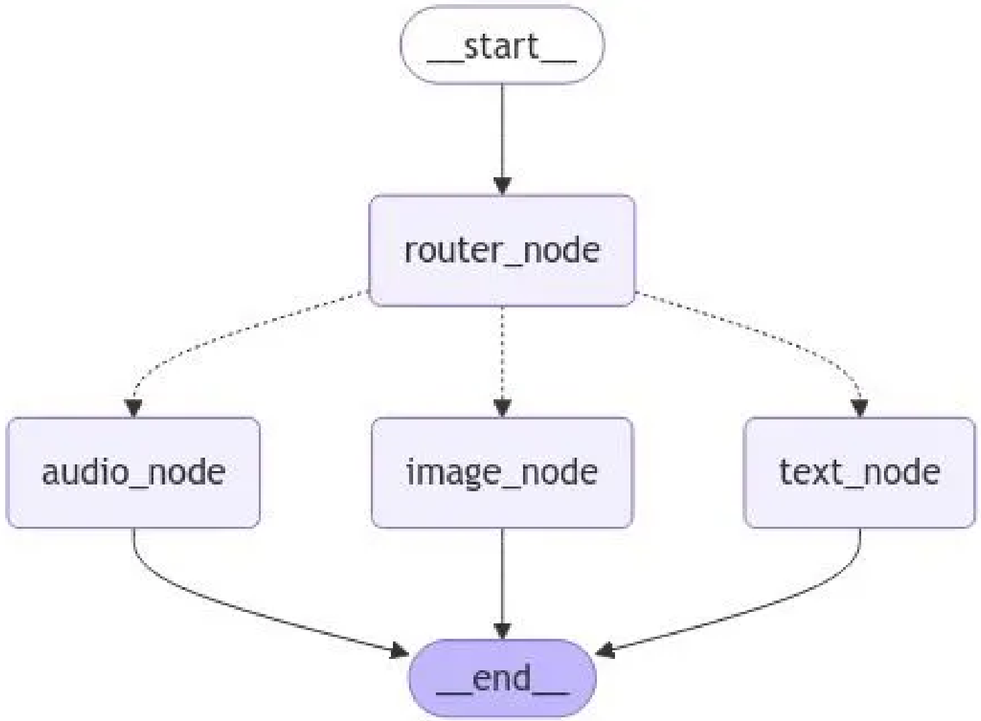
LangGraph Crash Course

- **LangGraph** is a framework designed for building **complex, stateful LLM agent** and multi-agent applications
- LangGraph uses a graph-based architecture that gives developers **fine-grained control** over agent behavior
- It's ideal for creating **robust, production-ready AI systems** that require **precision** and **adaptability**



LangGraph Crash Course

- **LangGraph** models **agent workflows** as **graphs**, using three main components:
  - **State** - A **shared data structure** that tracks the current status of your app (workflow)
  - **Nodes** - Python functions that define the **agent behaviour**. They take in the current state, perform actions, and return the updated state.
  - **Edges** - Python functions that decide which Node runs next based on the State, allowing for **conditional or fixed transitions**

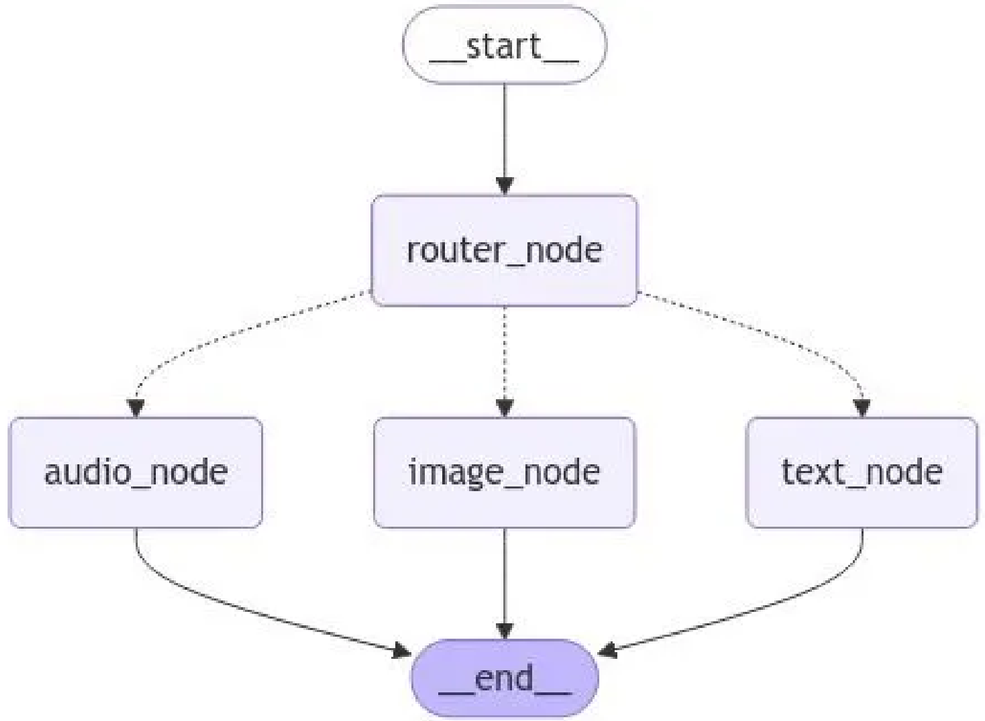


LangGraph Crash Course - Coding Session

- **Time for a coding session!**
- Open up the Colab notebook titled **1_langgraph_crash_course.ipynb**
- See you there! 👋

LangGraph Crash Course

- To recap, we've explored how **LangGraph** can be used to build **agentic workflows**, ranging from **simple systems** to more **complex architectures**
- We covered how to use the three fundamental components of the framework: **nodes**, **edges**, and **state**
- We also implemented a **Router**, which will serve as the **core architecture** for this workshop
- Now, it’s time to explore a very important component of any Agent architecture: **the memory**

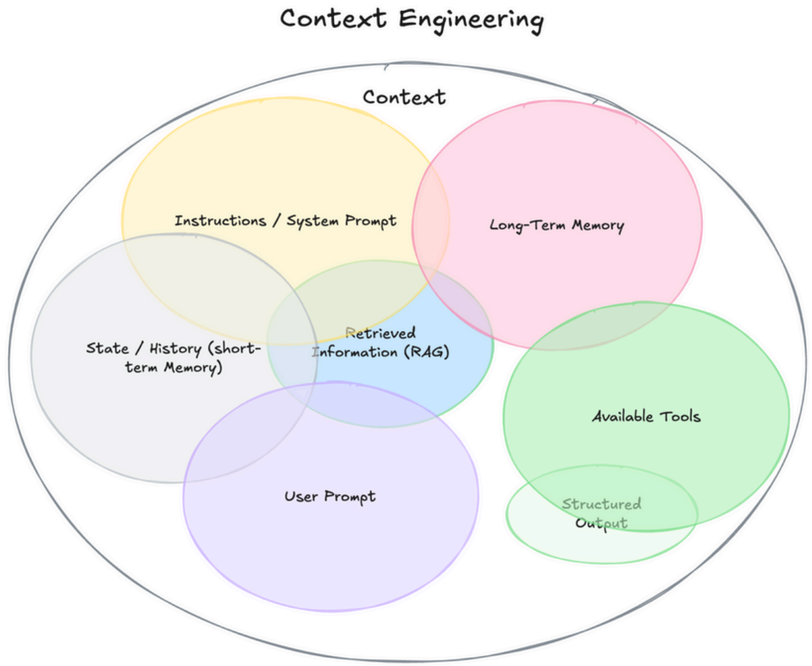


Agent Memory Systems

- In **agentic systems**, two types of memory are typically defined: **short-term memory** and **long-term memory**

- **Short-term memory** (also known as **working memory**) refers to all the context available to the LLM at the time of generating a response

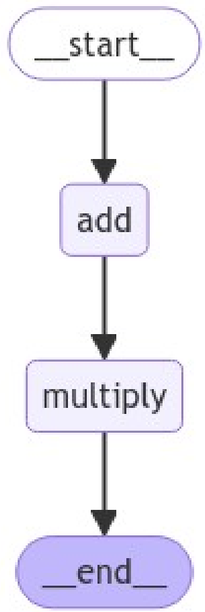
- **Long-term memory** refers to information stored in external databases, which the LLM can access when needed (e.g. Chroma)

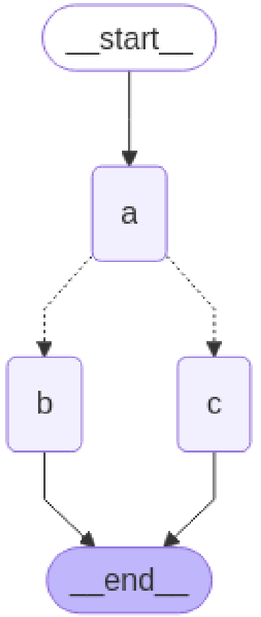


Chroma

Agent Memory Systems

- The **short-term** memory in LangGraph will involve three steps:
  - We'll store the message history inside the Graph State (remember it contains a property for the **messages**)
  - We'll summarise the conversation to reduce the token consumption
  - We'll use an external DB to store the **Graph State**





Agent Memory Systems - Coding Session

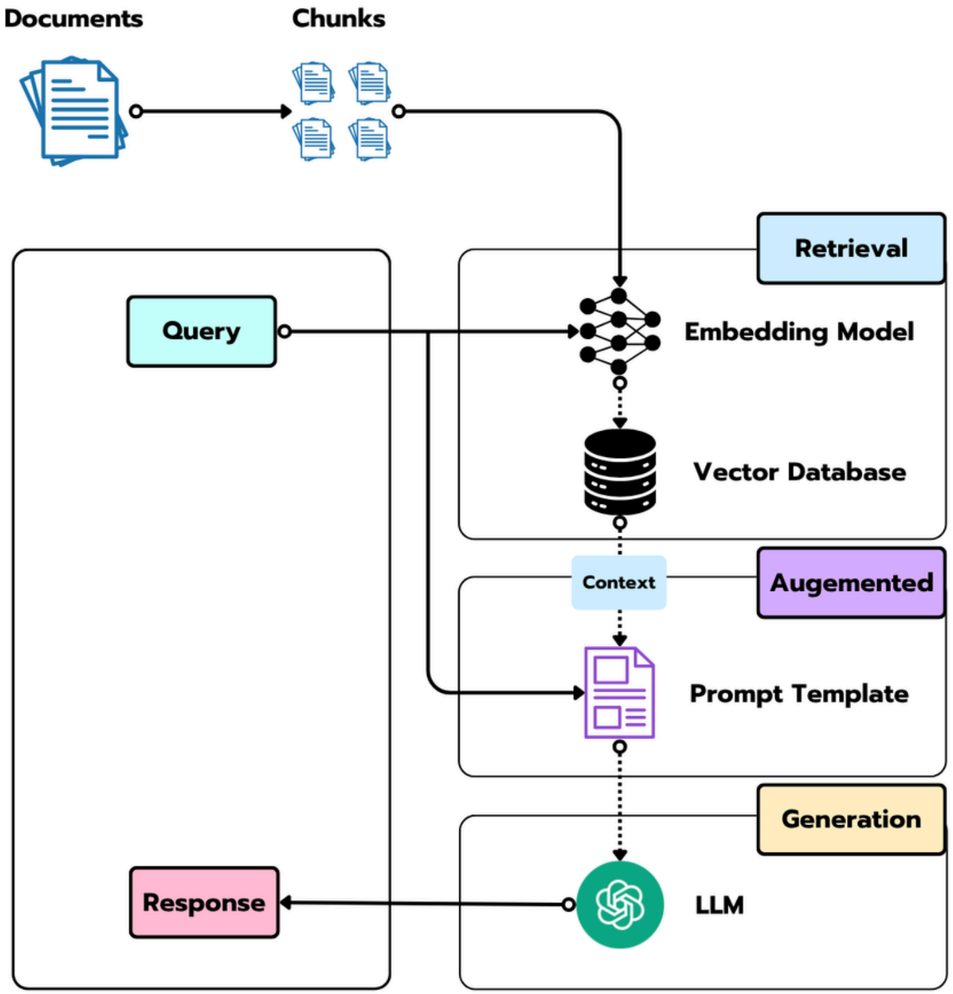
- **Time for a coding session!**
- Open up the Colab notebook titled **2_1_short_term_memory.ipynb**
- See you there! 👋

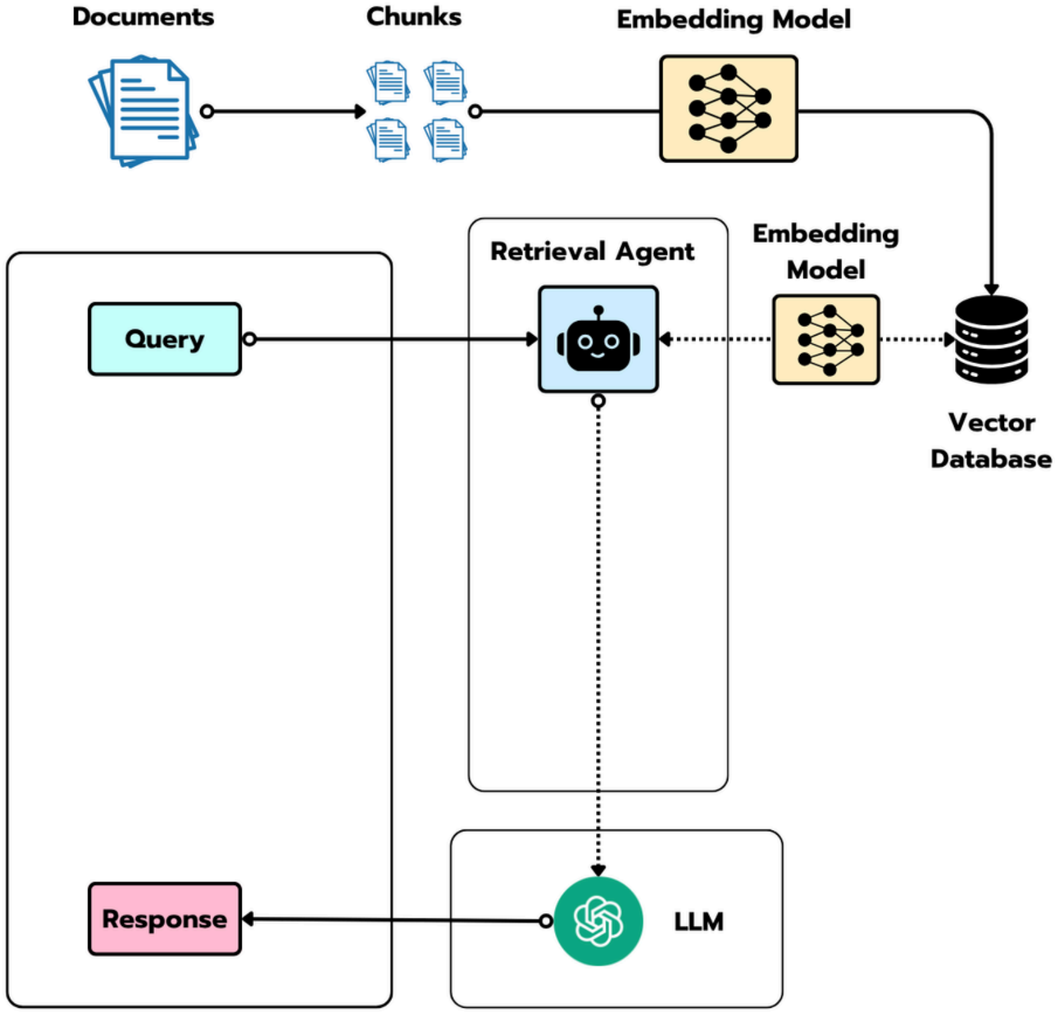
Agent Memory Systems

- The **long-term** memory in LangGraph will be implemented using **ChromaDB**
- First of all, we'll download some documents and store them in the **Vector Database**
- Then, we'll retrieve these documents from the **Vector Database** when they are relevant to the user question

Chroma

Agent Memory Systems





Agent Memory Systems - Coding Session

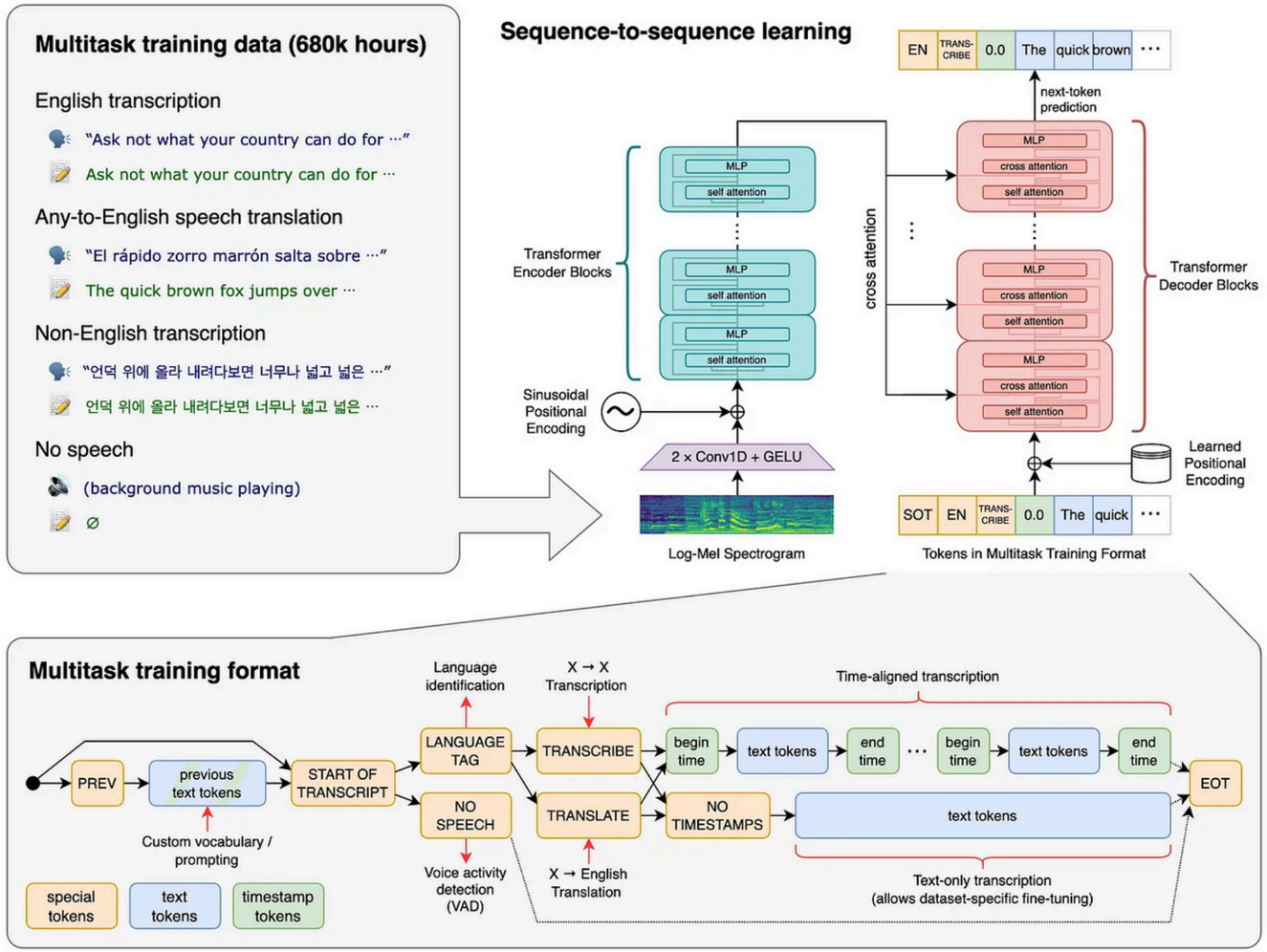
- **Time for a coding session!**
- Open up the Colab notebook titled **2_2_long_term_memory.ipynb**
- See you there! 👋

Speech-To-Text Systems

- An **STT (Speech-To-Text)** system turns incoming voice notes into text

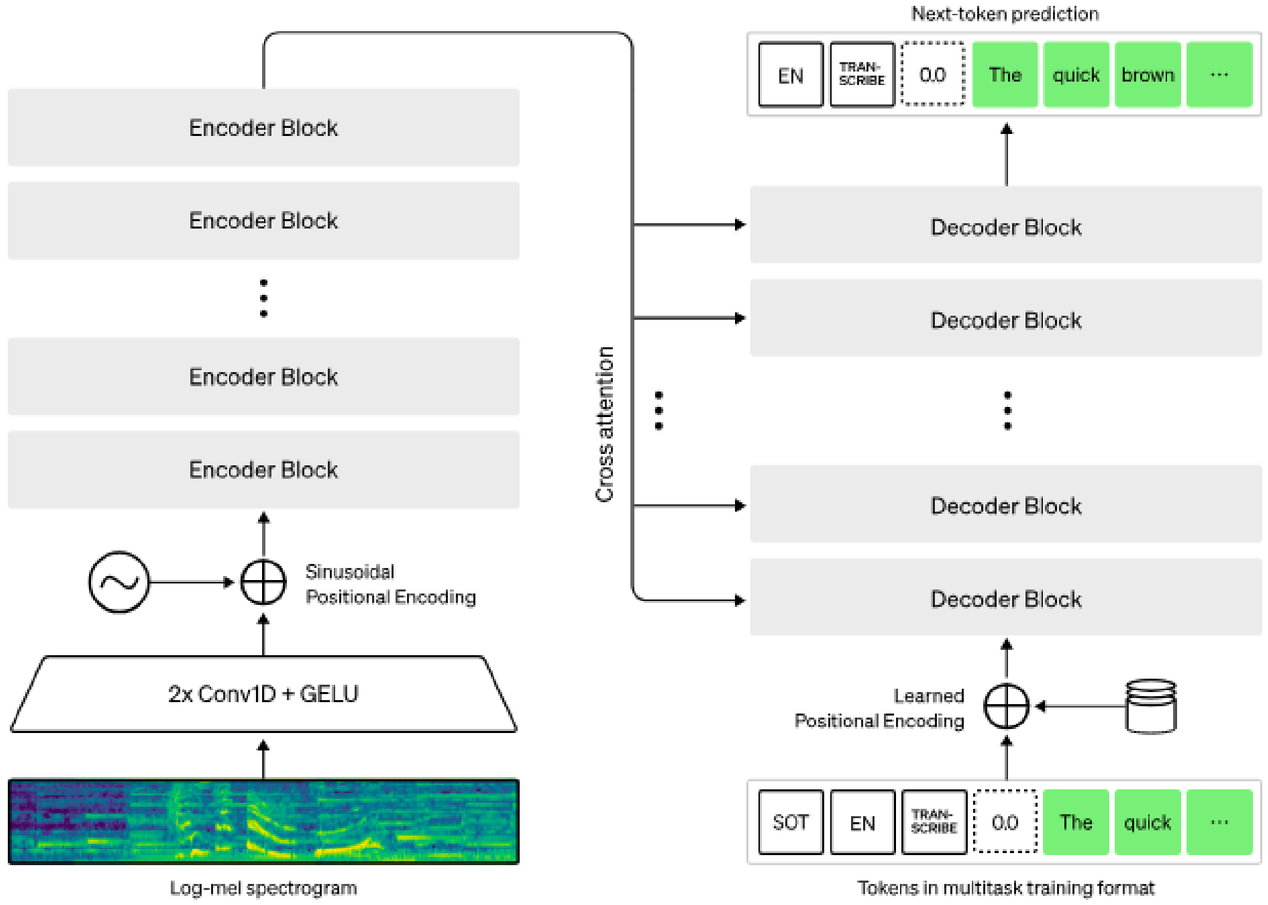
- We need this system if we want to transform the **user voice notes** (from Telegram) into text our LangGraph application can process

- We will be using **Whisper** as the STT model. This is an advanced model from OpenAI that can handle multiple languages, different accents and even background noises



Speech-To-Text Systems (Whisper)

- Open-source **speech recognition** system by **OpenAI**.
- Trained on **680,000 hours** of supervised, multilingual data.
- Near **human-level performance** in English speech recognition
- Handles **accents**, **background noise**, and **technical language** robustly
- Uses an **encoder-decoder** transformer architecture

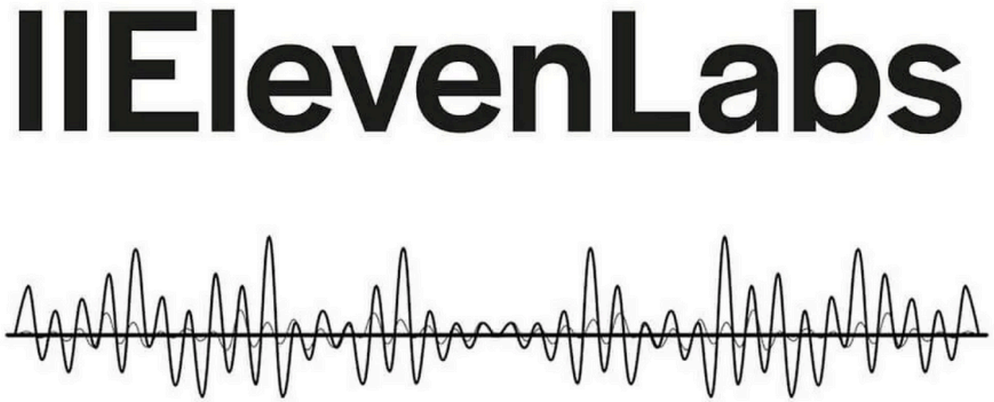


Speech-To-Text Systems - Coding Session

- **Time for a coding session!**
- Open up the Colab notebook titled **3_speech_to_text_systems.ipynb**
- See you there! 👋

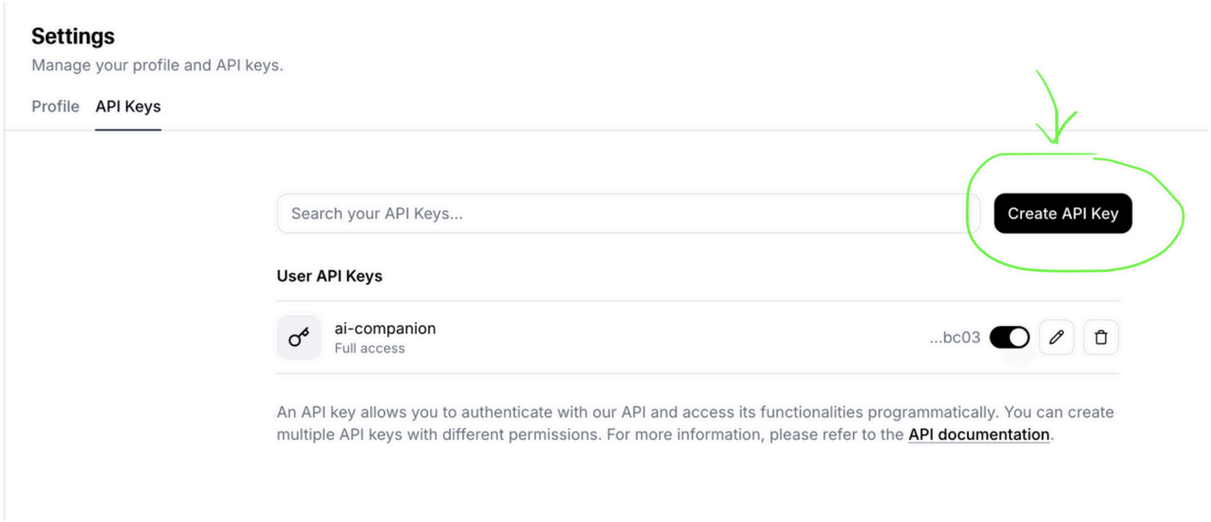
Text-To-Speech Systems

- A **TTS (Text-To-Speech)** system turns text into voice.
- To achieve this, we'll use **Elevenlabs voices**, as they provide a huge catalog and amazing quality.
- The **TTS system** will be run inside one of the **LangGraph nodes**. The **Router** will decide between a **conversational**, **image** or **voice** response.



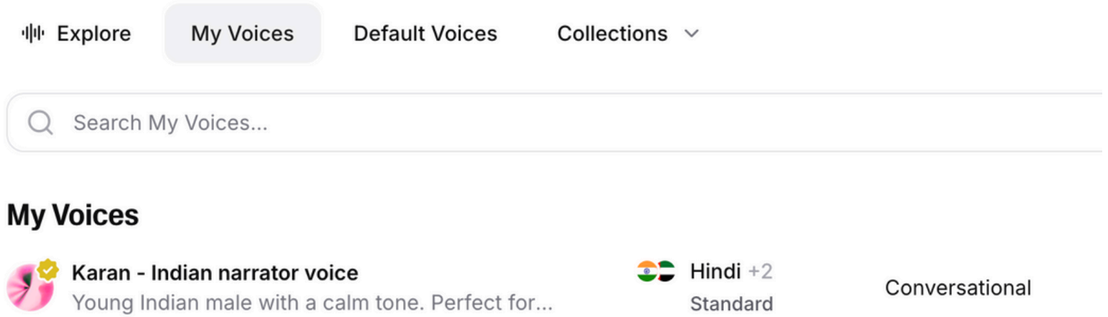
Text-To-Speech Systems

- Start by signing up for an ElevenLabs account (if you haven't already)
- For this workshop, you can use the **Free Tier**
- To build the Telegram Agent, you'll need an **ELEVENLABS_API_KEY**. You can generate one by following the steps shown in the image on the right.



Text-To-Speech Systems

- The voice I've chosen for the Telegram Bot is this one.
- Add this voice to your voices to have it available when using the ElevenLabs client.



Text-To-Speech Systems - Coding Session

- **Time for a coding session!**
- Open up the Colab notebook titled **4_text_to_speech_systems.ipynb**
- See you there! 👋

Image Understanding - VLM

- **Vision Language Models (VLMs)** process both **images** and **text**, generating text-based insights from visual input.
- They help with tasks like **object recognition**, **image captioning**, and **answering questions** about images. Some even understand spatial relationships, identifying objects or their positions.
- For our **Telegram Agent**, **VLMs** are key to making sense of incoming images.

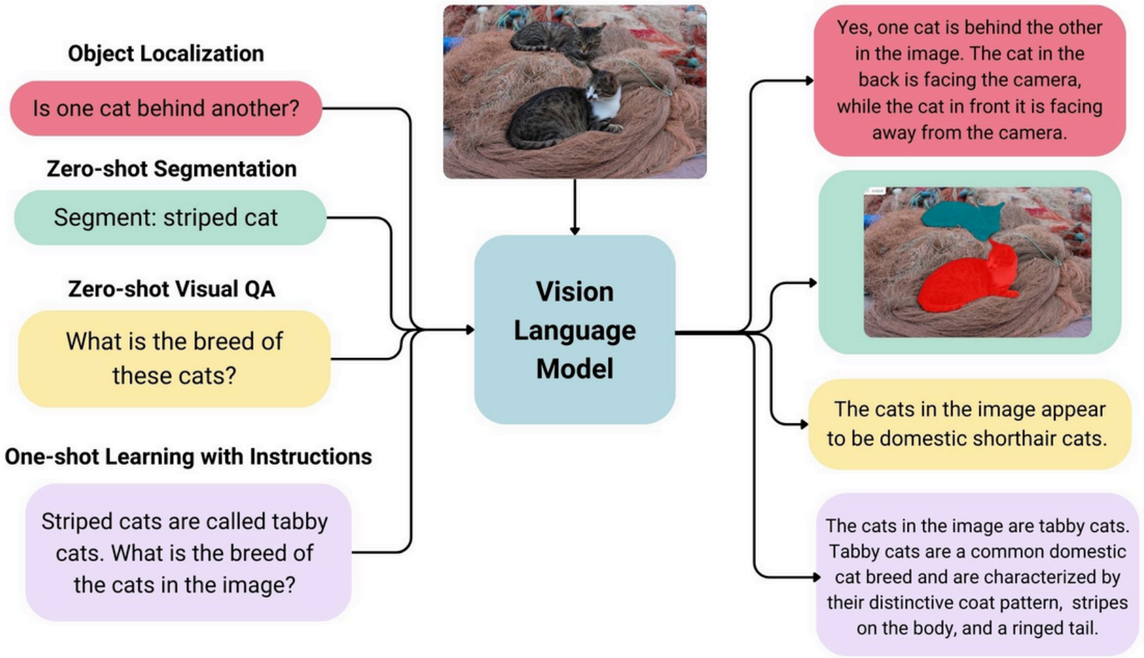


Image Understanding - Coding Session

- **Time for a coding session!**
- Open up the Colab notebook titled **5_vision_language_models.ipynb**
- See you there! 👋

Image Generation (AR Image Models)

- Generate images **token by token**, in sequence —just like how LLMs generate text
- Operate on **1D sequences**, so **2D images** are flattened (e.g., row-by-row or via multi-scale methods)
- Use **Transformer decoders** with causal attention, predicting the next token based only on previous ones
- Require **image tokenization** to convert images into a format usable by LLM models

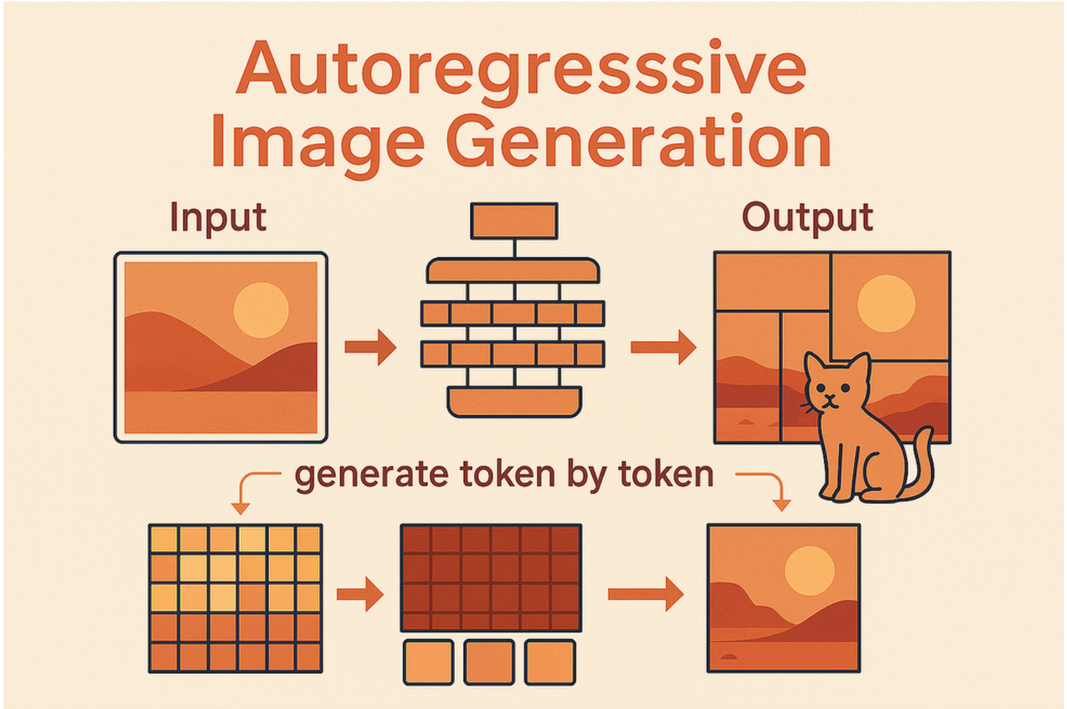


Image Understanding - Coding Session

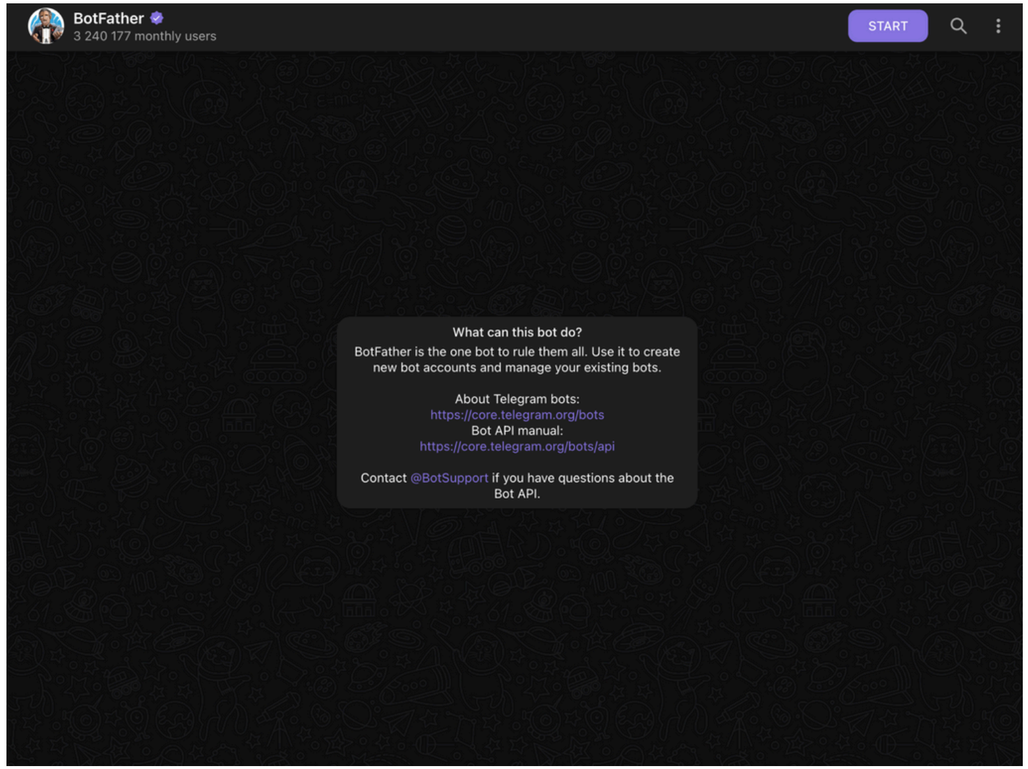
- **Time for a coding session!**
- Open up the Colab notebook titled **6_image_generation.ipynb**
- See you there! 👋

Telegram Integration

- **Enough theory** ... 😴
- Time to put everything into **practice**
- Time to build the **Multimodal Telegram Agent**! 💪

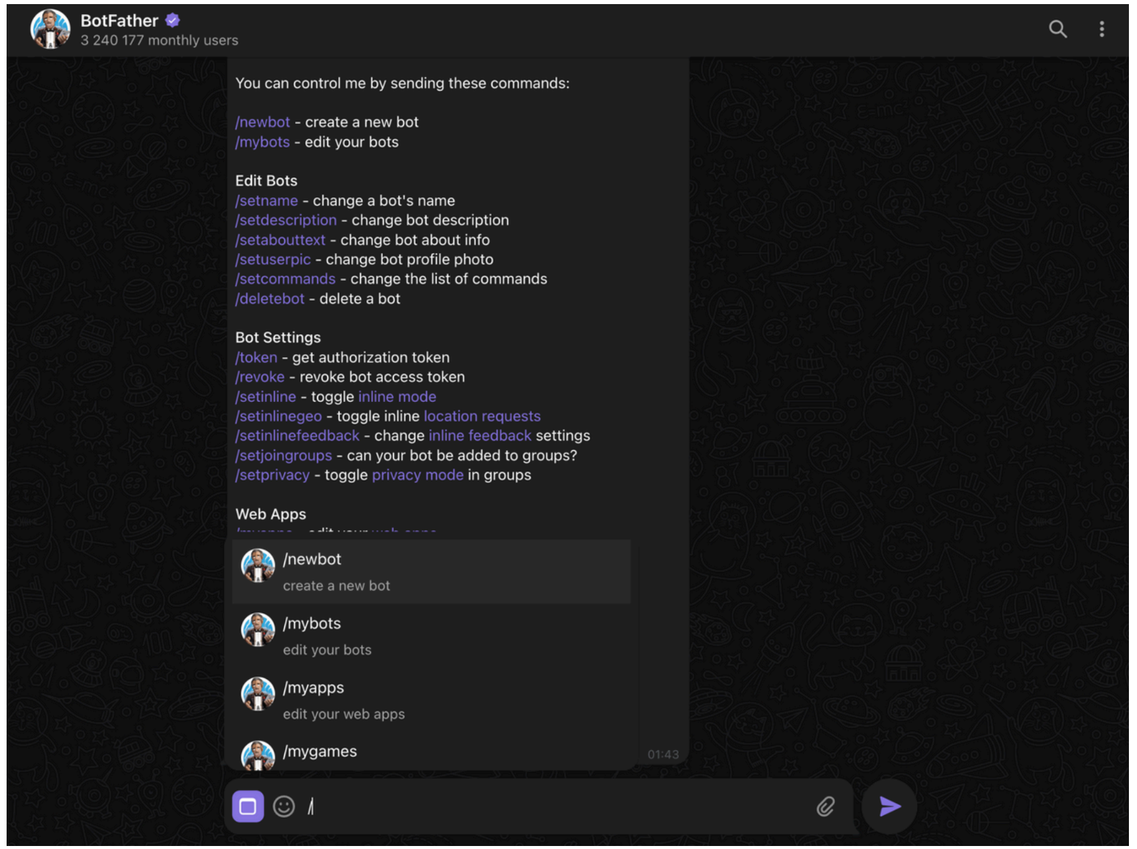
Telegram Integration

- The first thing we need to do is to open Telegram and start a chat with another bot: **BotFather**.
- When you see a screen like the one on the right, go ahead and click the **START** button.
- Everything we're about to cover works on both your phone and Telegram Web



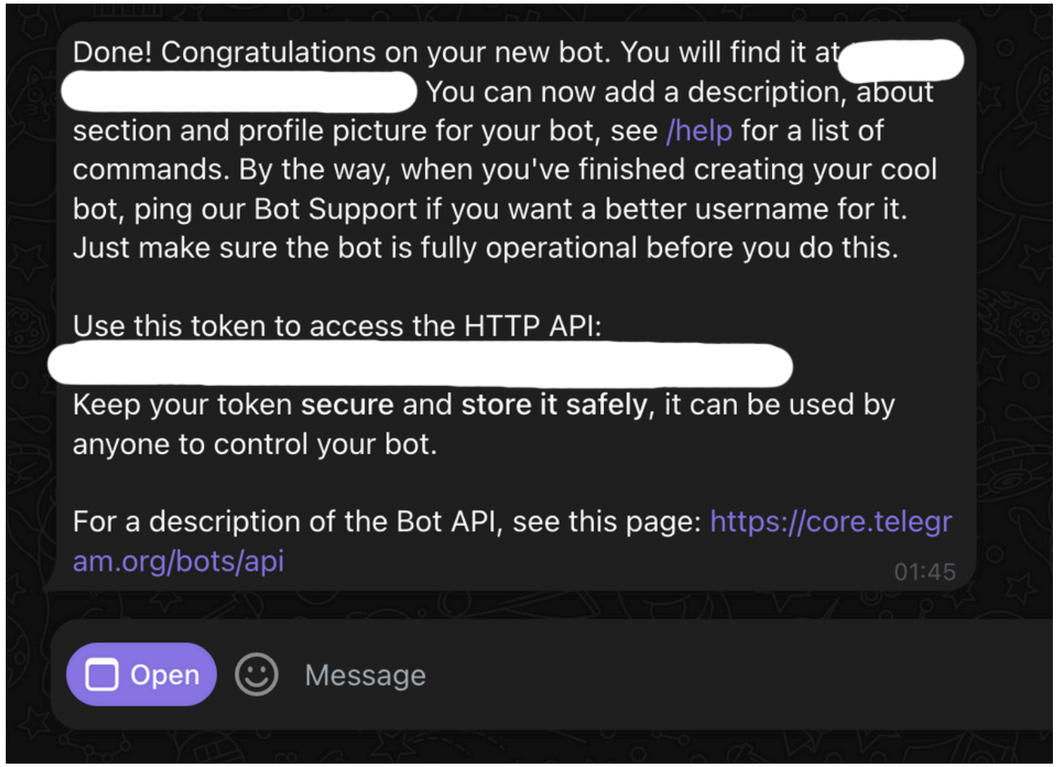
Telegram Integration

- Run the **/newbot** command



Telegram Integration

- Run the **/newbot** command
- Then, follow the instructions to create your own bot.
- Once your bot is created, you'll see a message like the one on the right.
- **Important**: copy the token and save it somewhere safe — **we'll need it later!**

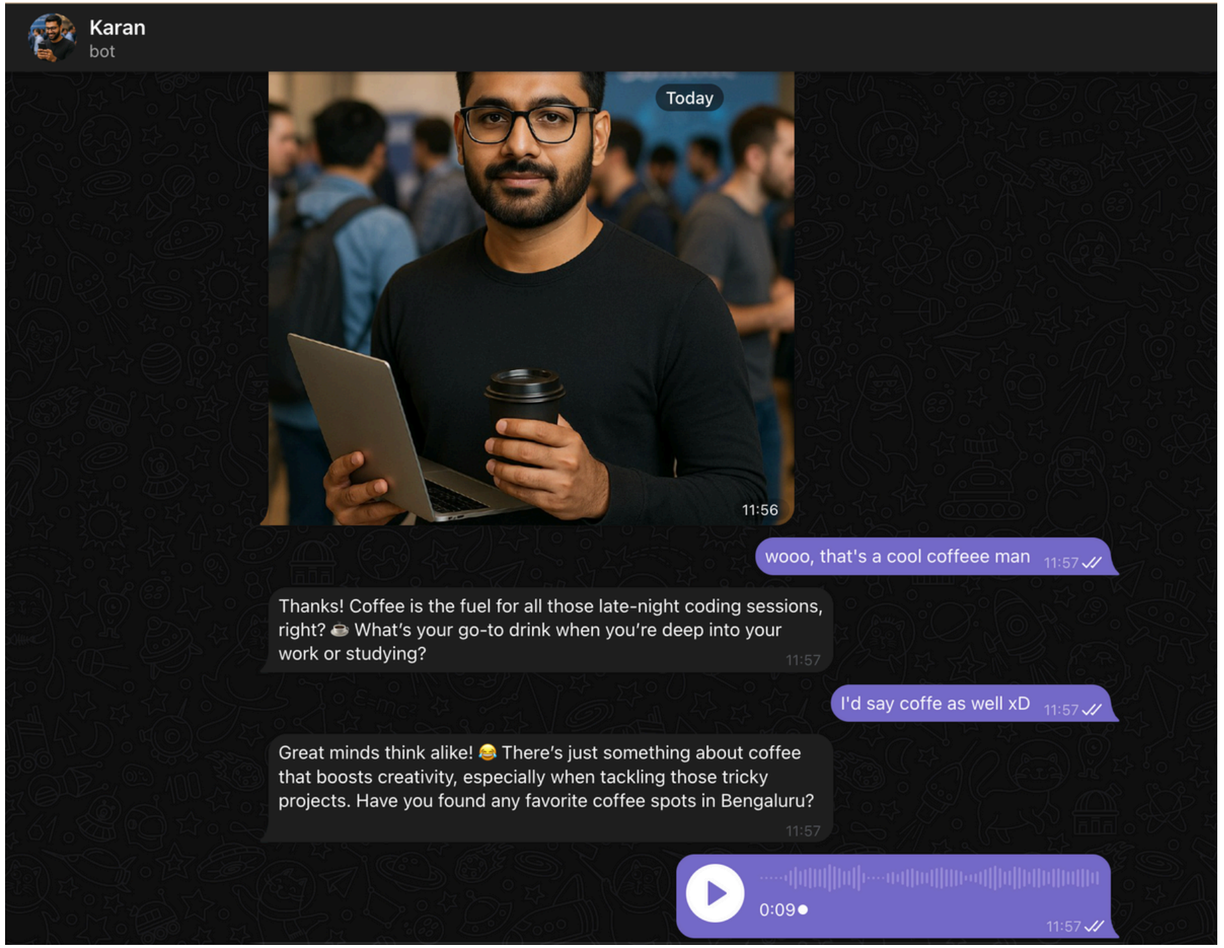


Telegram Integration

- Now that the Telegram Bot is created, we need to implement the logic behind!
- It's time to lay the foundation for our **Telegram agent**.
- Just head over to **7_telegram_agent.ipynb** to implement the final project.

Telegram Integration

- My friend, take a moment to be proud ...
- You've built a fully functional Multimodal Agent — **from scratch!**
- **CONGRATULATIONS!** 👏
- And this is just the beginning. Imagine the possibilities for Karan! Surf advisor? Travel planner? The use cases are endless ...
- **Let's keep building** 😎



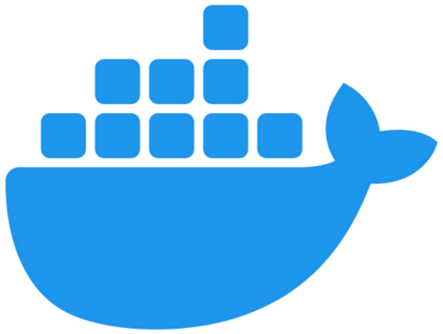
Next Steps ...

- As you can imagine, even though we now have a functional notebook implementation, this is only the beginning...

- **Part 2** will take things to the next level — transforming these notebooks into a **fully-fledged production application**.

- As a **premium subscriber**, you will get exclusive access to the code and setup guide to **deploy this Telegram Agent to AWS**, complete with:
  - CI / CD pipelines
  - LLMOps integration
  - AWS Lambda Function deployment

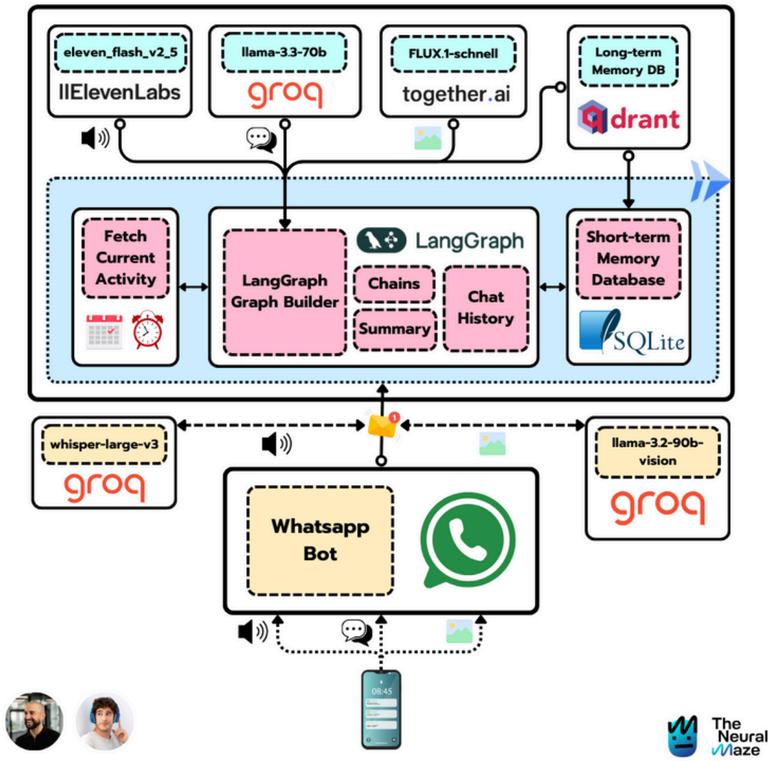
- You'll be able to take this framework, **build any Telegram Bot** you want, and **share it with the world** 🌍



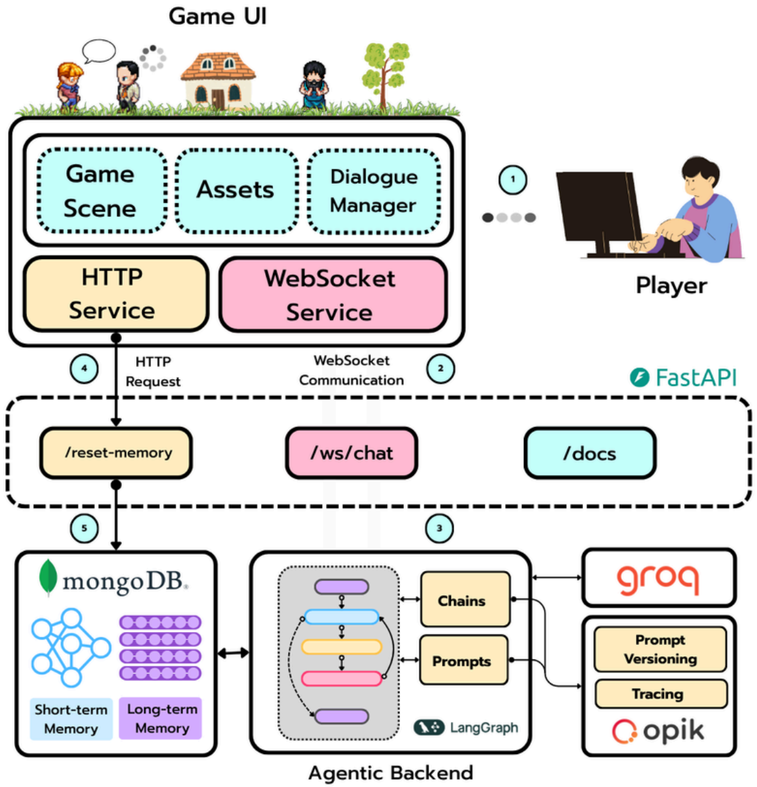
Other projects you might like ...

I embrace the idea that the **best** way to learn is **by doing**.

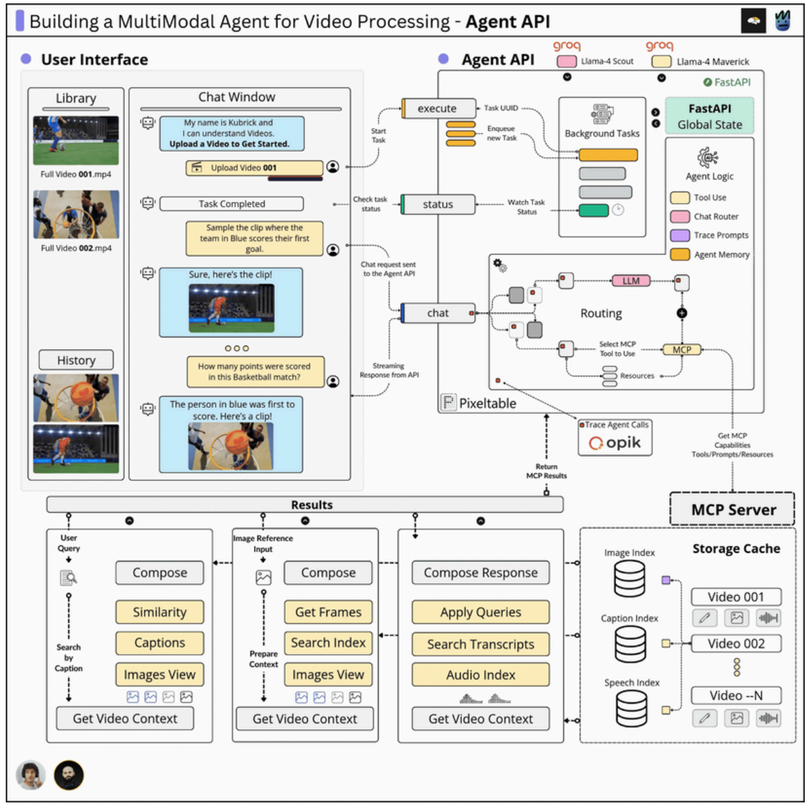
Ava, the WhatsApp Agent



PhiloAgents



Kubrick



The Neural Maze