Maximilian Schwarzmüller

Developer, AWS Solutions Architect, Online Instructor

academind.com

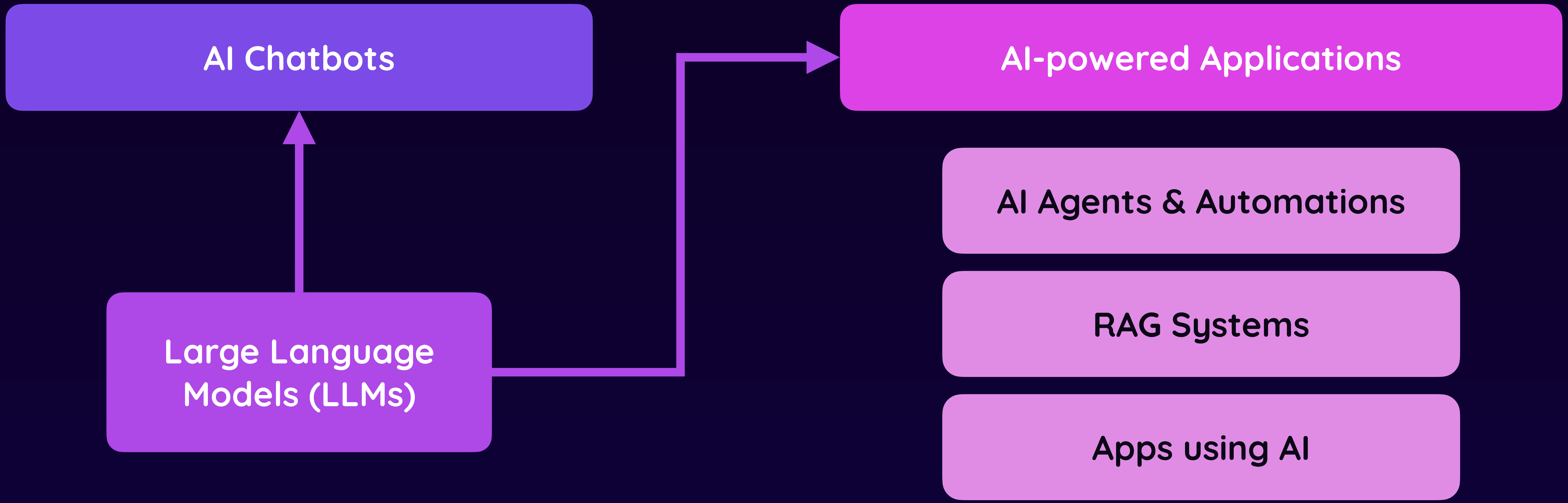
maximilian-schwarzmueller.com

@maxedapps

What exactly is “Generative AI”?

Generative AI?

Generative AI refers to a category of **artificial intelligence** models that are capable of **creating new content**—such as text, images, audio, video, or code—that is **similar to what a human might produce.**



AI Chatbots

AI-powered Applications

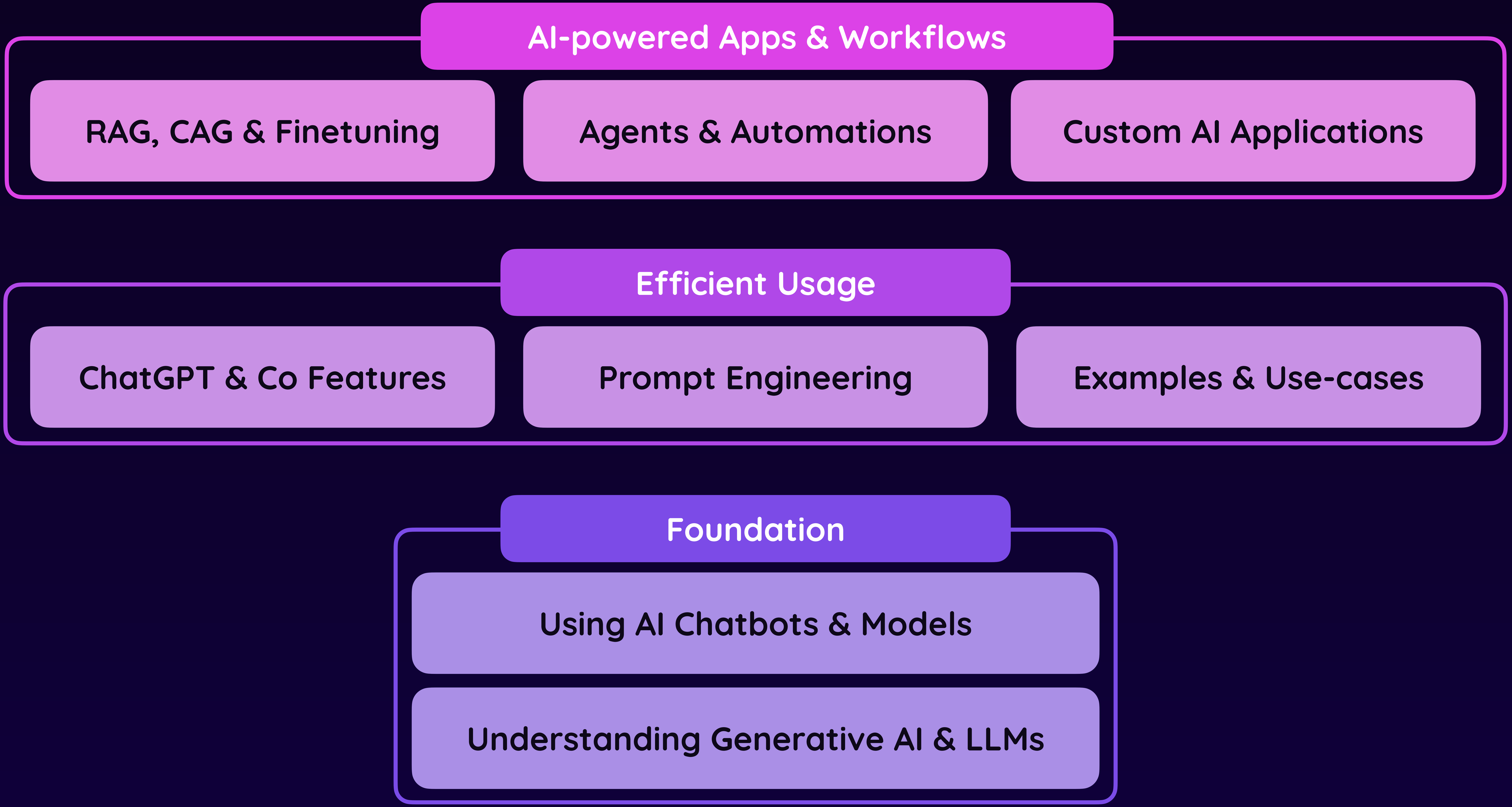
AI Agents & Automations

RAG Systems

Apps using AI

Large Language Models (LLMs)

About This Course



AI-powered Apps & Workflows

- RAG, CAG & Finetuning
- Agents & Automations
- Custom AI Applications

Efficient Usage

- ChatGPT & Co Features
- Prompt Engineering
- Examples & Use-cases

Foundation

- Using AI Chatbots & Models
- Understanding Generative AI & LLMs

The Generative AI Revolution

Nov 2022

Models

- ChatGPT GPT-3
- ChatGPT GPT-4
- Google Gemini
- Anthropic Claude
- ChatGPT GPT-4o
- DeepSeek R1
- . . .

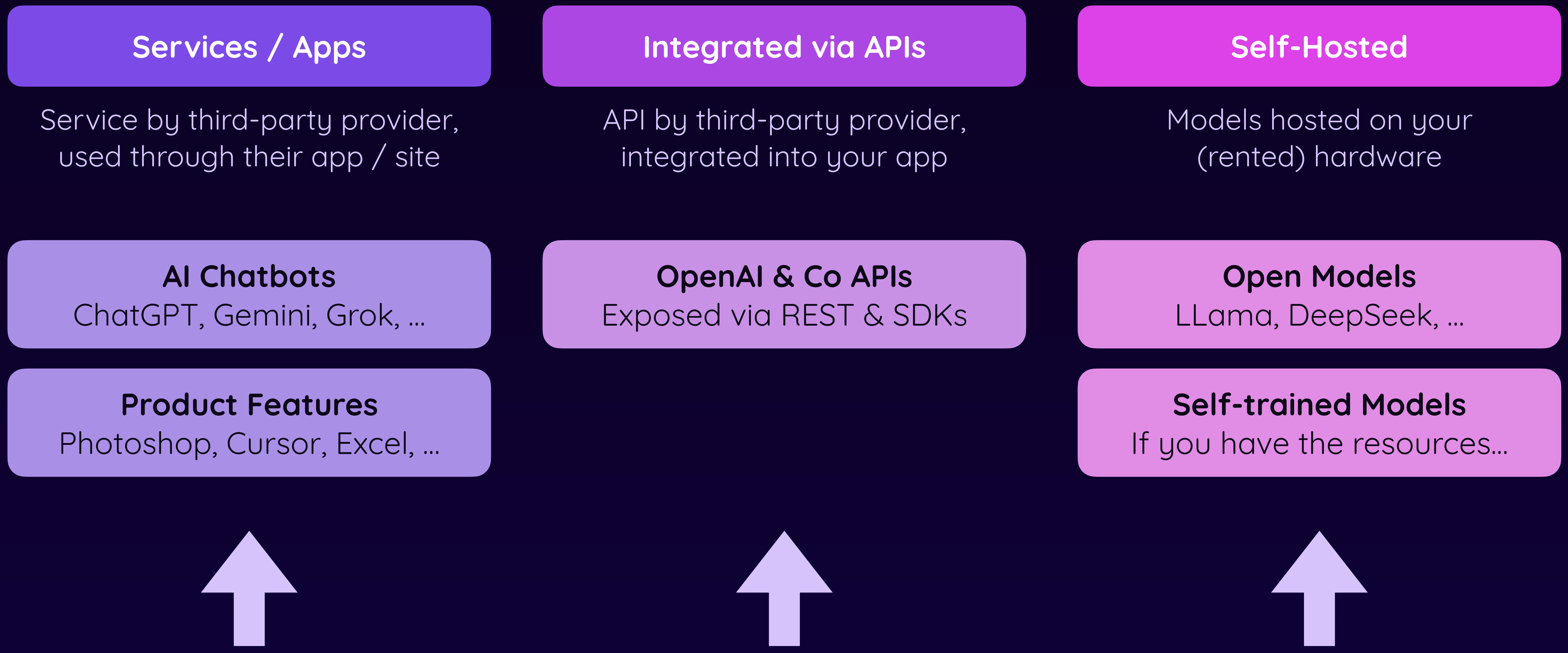
Capabilities

- Images
- Code Execution
- Audio
- Web Search
- Deep Research
- Agents
- . . .

Highly Dynamic: New AI models & services built on top of those models are launched all the time

Highly Innovative: New application areas and “tools” are unlocked frequently

There Are Multiple Ways Of Using Gen-AI



Services / Apps

Service by third-party provider, used through their app / site

AI Chatbots
ChatGPT, Gemini, Grok, …

Product Features
Photoshop, Cursor, Excel, …

Main focus of this course

Integrated via APIs

API by third-party provider, integrated into your app

OpenAI & Co APIs
Exposed via REST & SDKs

Also covered in-depth, including how to build custom apps & agents

Self-Hosted

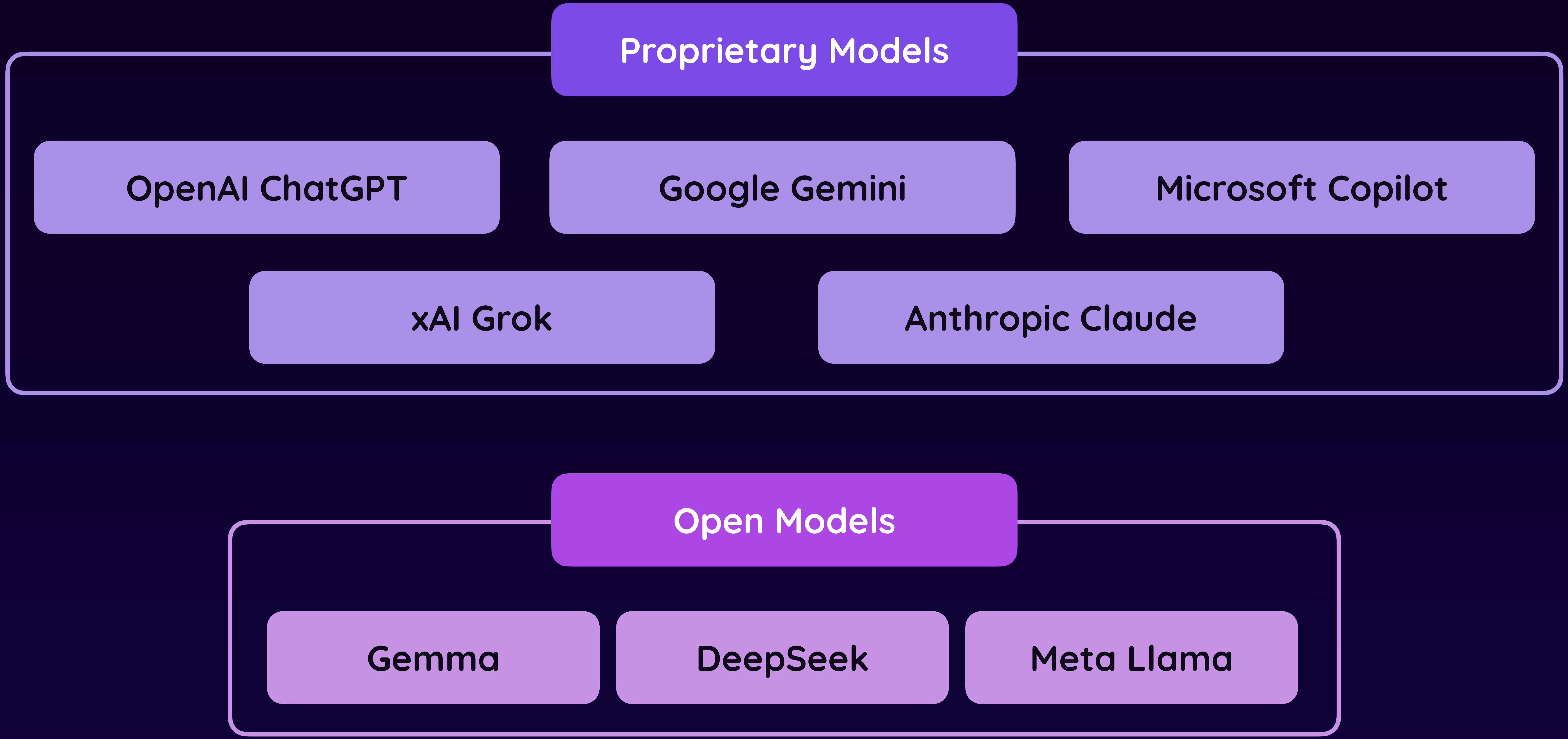
Models hosted on your (rented) hardware

Open Models
LLama, DeepSeek, …

Self-trained Models
If you have the resources…

Self-hosting intro included, training your own models is NOT covered

Key Gen-AI Service Providers—Overview



Proprietary Models

- OpenAI ChatGPT
- Google Gemini
- Microsoft Copilot
- xAI Grok
- Anthropic Claude

Open Models

- Gemma
- DeepSeek
- Meta Llama

Important

You will not necessarily be able to reproduce the responses you see in the videos!

Even when using the exact same prompts.
All these AI models have a certain degree of “randomness”
& the underlying models will evolve and change.

❗

Important

Free Usage Is Possible

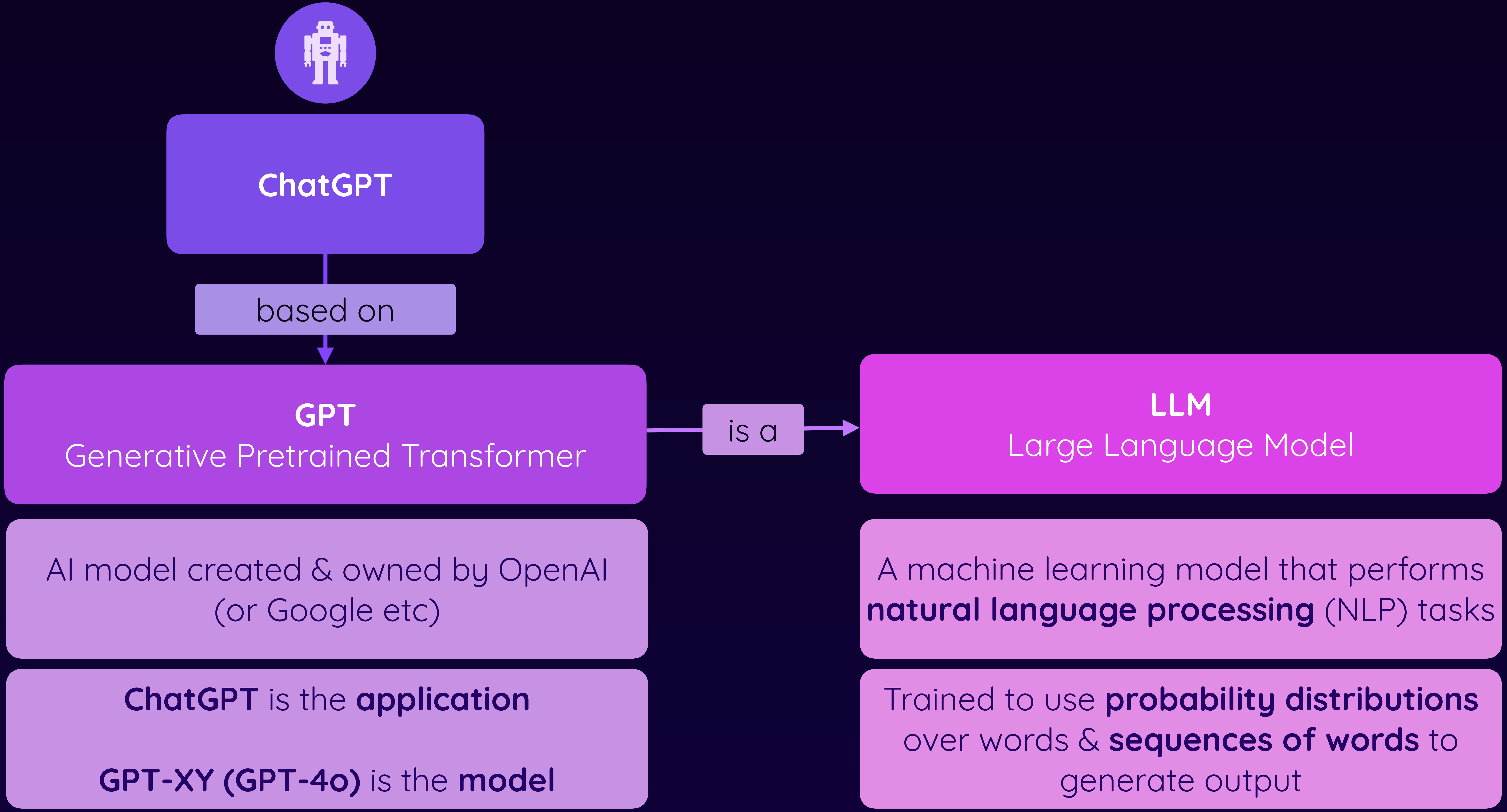
But often not enough

Generative AI — Behind The Scenes

How LLMs Works Under The Hood

- How ChatGPT & LLMs Were Trained
- How They Work
- Features & Limitations

ChatGPT & Co Under The Hood



ChatGPT

based on

GPT
Generative Pretrained Transformer

is a

LLM
Large Language Model

AI model created & owned by OpenAI (or Google etc)

ChatGPT is the **application**

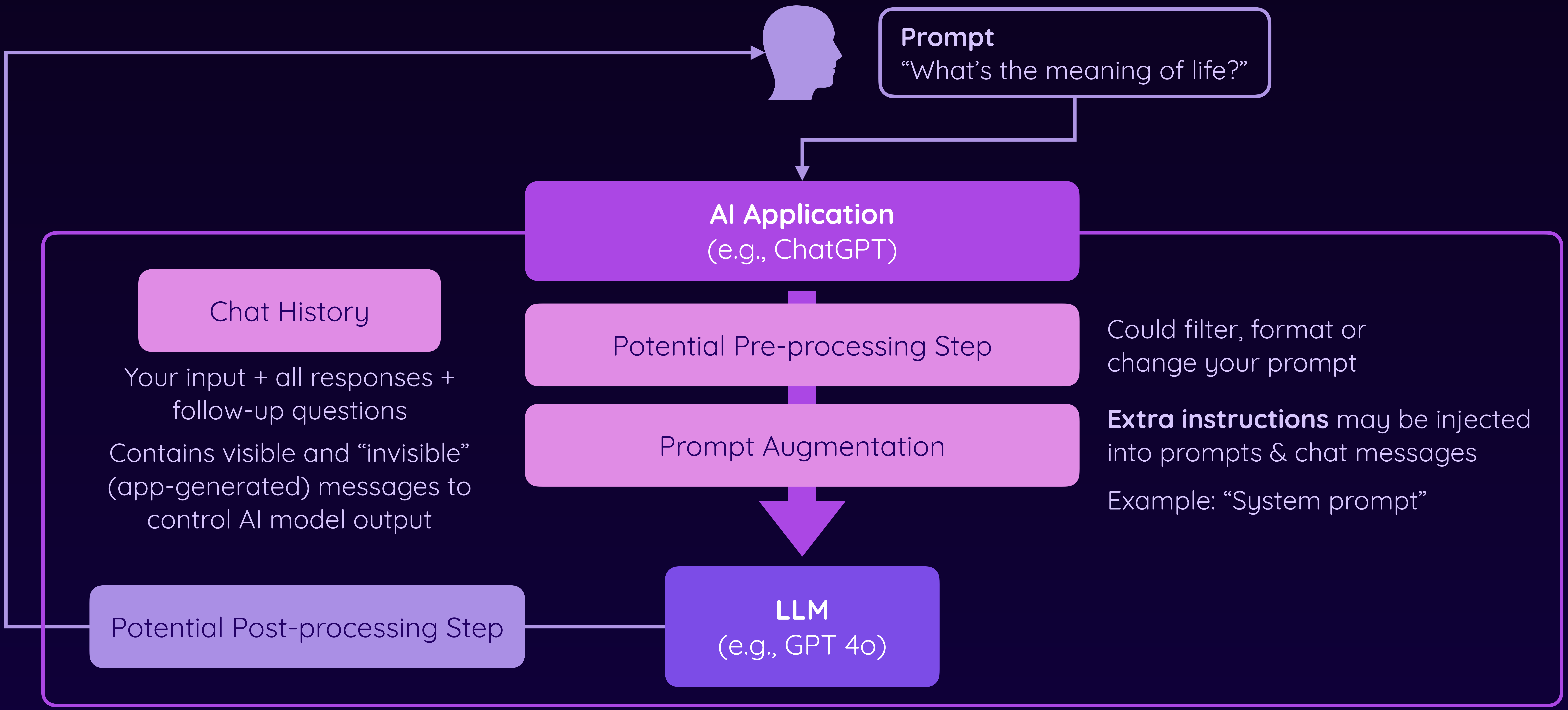
GPT-XY (GPT-4o) is the **model**

A machine learning model that performs **natural language processing** (NLP) tasks

Trained to use **probability distributions** over words & **sequences of words** to generate output

Model vs Application

when we interact with an LLM usually we are dealing
with 2 different thingss thata are working together:
1) The application (which preprocess the request or postprocess the answer)
2) the model



Prompt
"What's the meaning of life?"

AI Application
(e.g., ChatGPT)

Chat History

Your input + all responses + follow-up questions

Contains visible and "invisible" (app-generated) messages to control AI model output

Potential Pre-processing Step

Prompt Augmentation

Could filter, format or change your prompt

Extra instructions may be injected into prompts & chat messages

Example: "System prompt"

LLM
(e.g., GPT 4o)

Potential Post-processing Step

Augmented Prompts & Messages

AI applications (like ChatGPT) may **edit your prompt** or **insert extra (invisible) messages** into the chat history

Why?
To control the output of the underlying AI model

Examples

System Prompt

A special message that's injected into the chat history

Aims to define the general "behavior" of the AI model

Example: "You are a friendly and helpful assistant. If the user asks for news or recent developments, reply with 'I don't know that, sorry'"

Tools

Covered in-depth later!

Some AI applications expose "tools" to the AI models

The AI models may request the use of a provided tool to better fulfil a user request

Example: "You can search the web, if needed. Reply with WEB SEARCH: <search term> if you want to use this tool"

Retrieved Data (RAG)

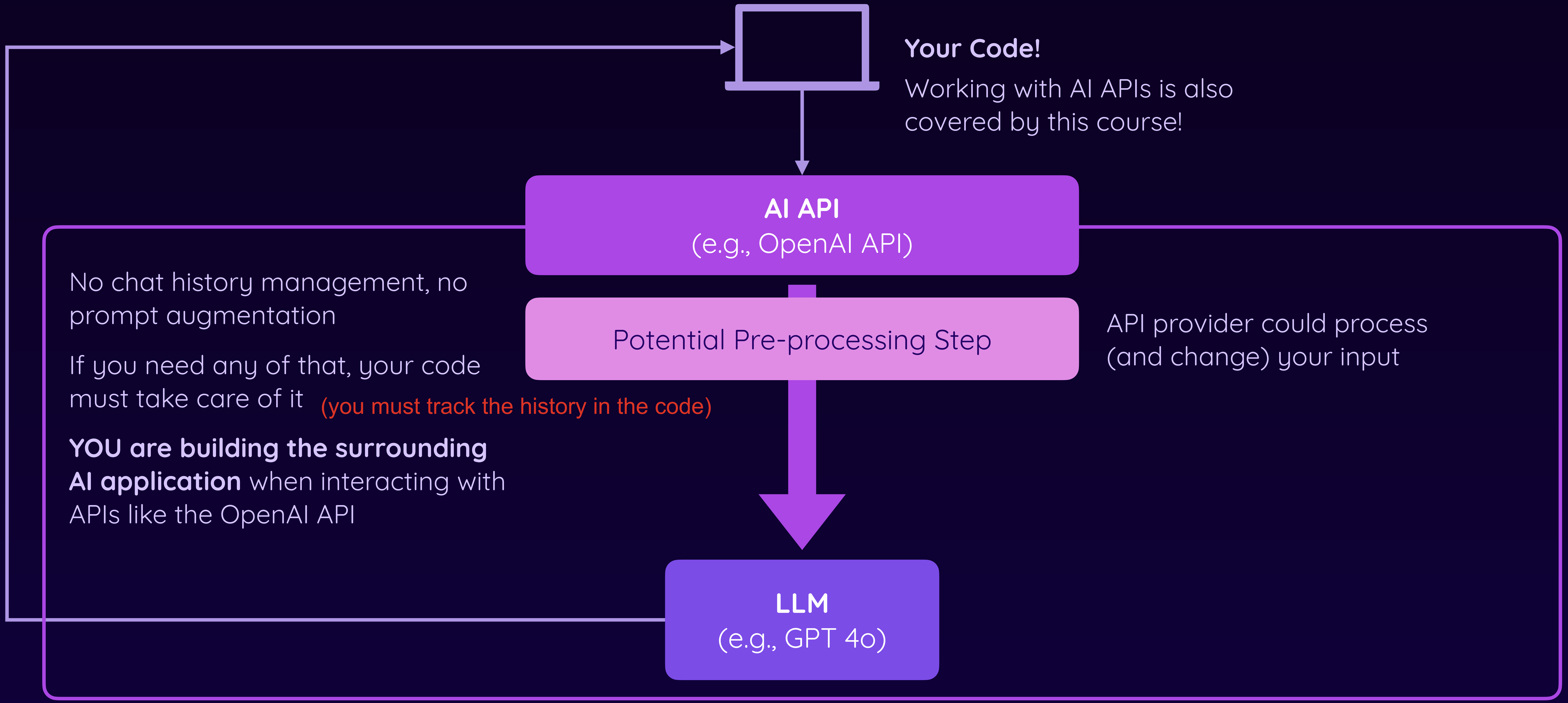
Covered in-depth later!

For certain requests, personal or restricted data may be required

AI apps can inject retrieved data into a prompt to make it available

Example: Fetch and inject PDF document contents

Interacting with AI via APIs



Your Code!

Working with AI APIs is also covered by this course!

AI API
(e.g., OpenAI API)

Potential Pre-processing Step

API provider could process (and change) your input

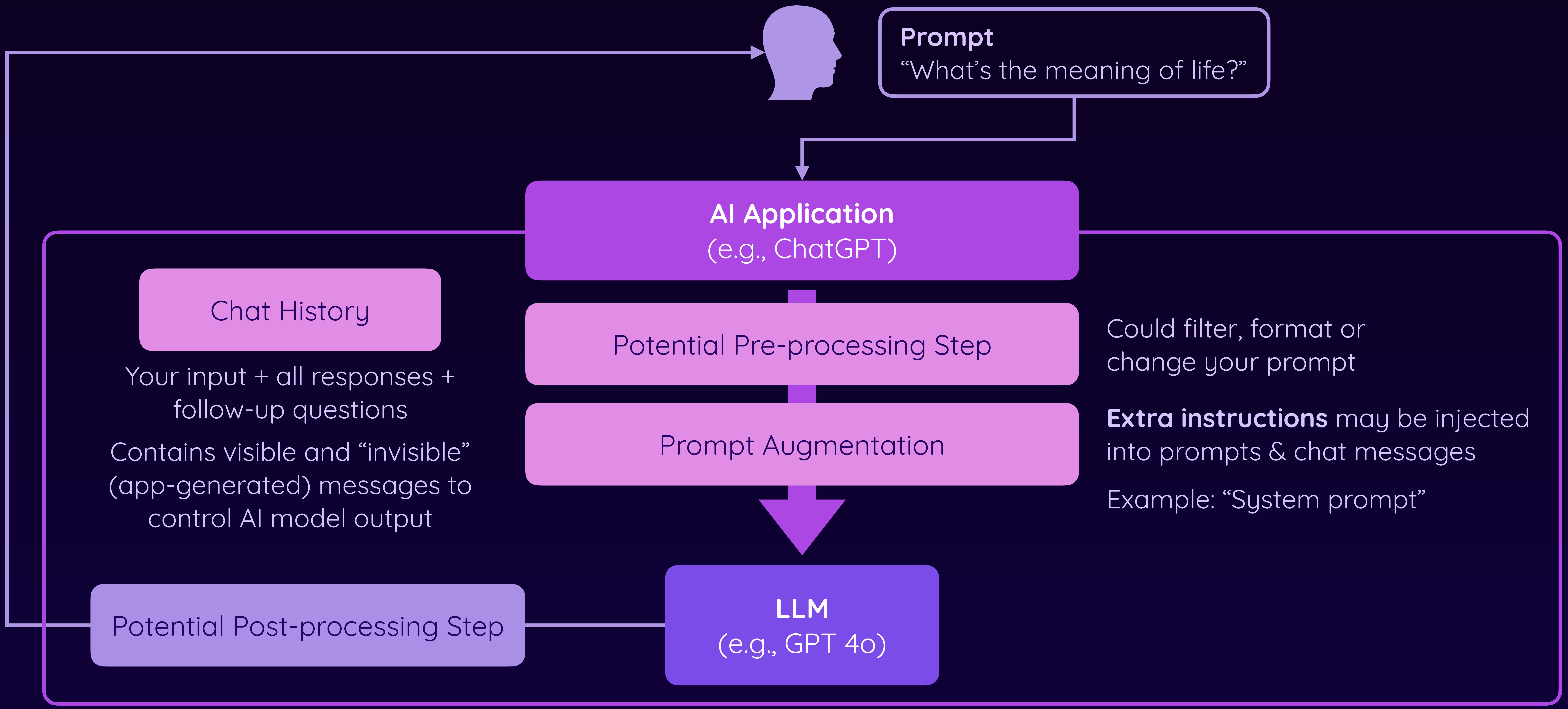
No chat history management, no prompt augmentation

If you need any of that, your code must take care of it (you must track the history in the code)

YOU are building the surrounding AI application when interacting with APIs like the OpenAI API

LLM
(e.g., GPT 4o)

Model vs Application



Prompt
"What's the meaning of life?"

AI Application
(e.g., ChatGPT)

Chat History

Your input + all responses + follow-up questions

Contains visible and "invisible" (app-generated) messages to control AI model output

Potential Pre-processing Step

Could filter, format or change your prompt

Prompt Augmentation

Extra instructions may be injected into prompts & chat messages

Example: "System prompt"

LLM
(e.g., GPT 4o)

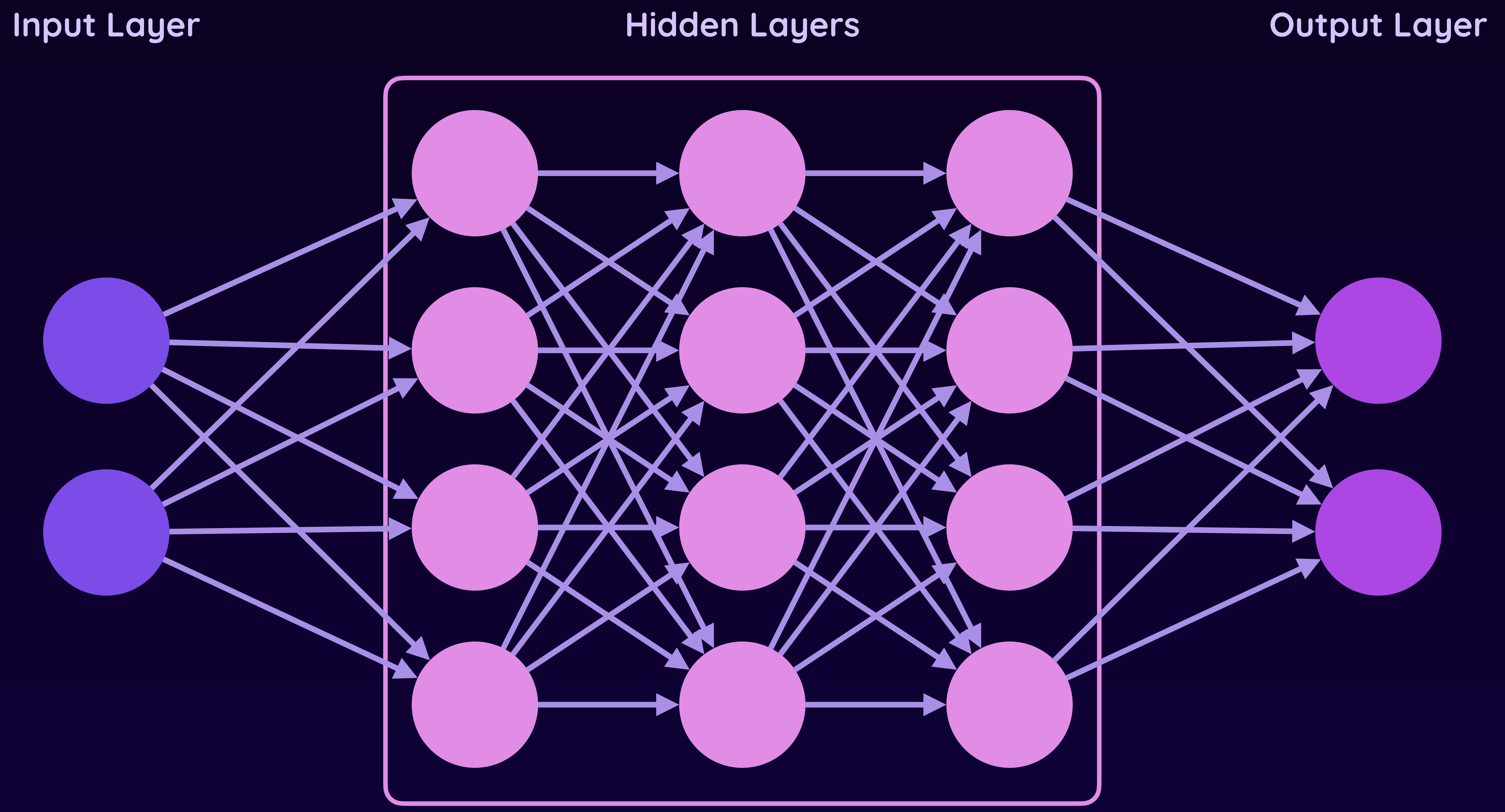
Potential Post-processing Step

The Most Important Part: The Model

LLM
(e.g., GPT 4o)

The **Large Language Model** that generates the response text

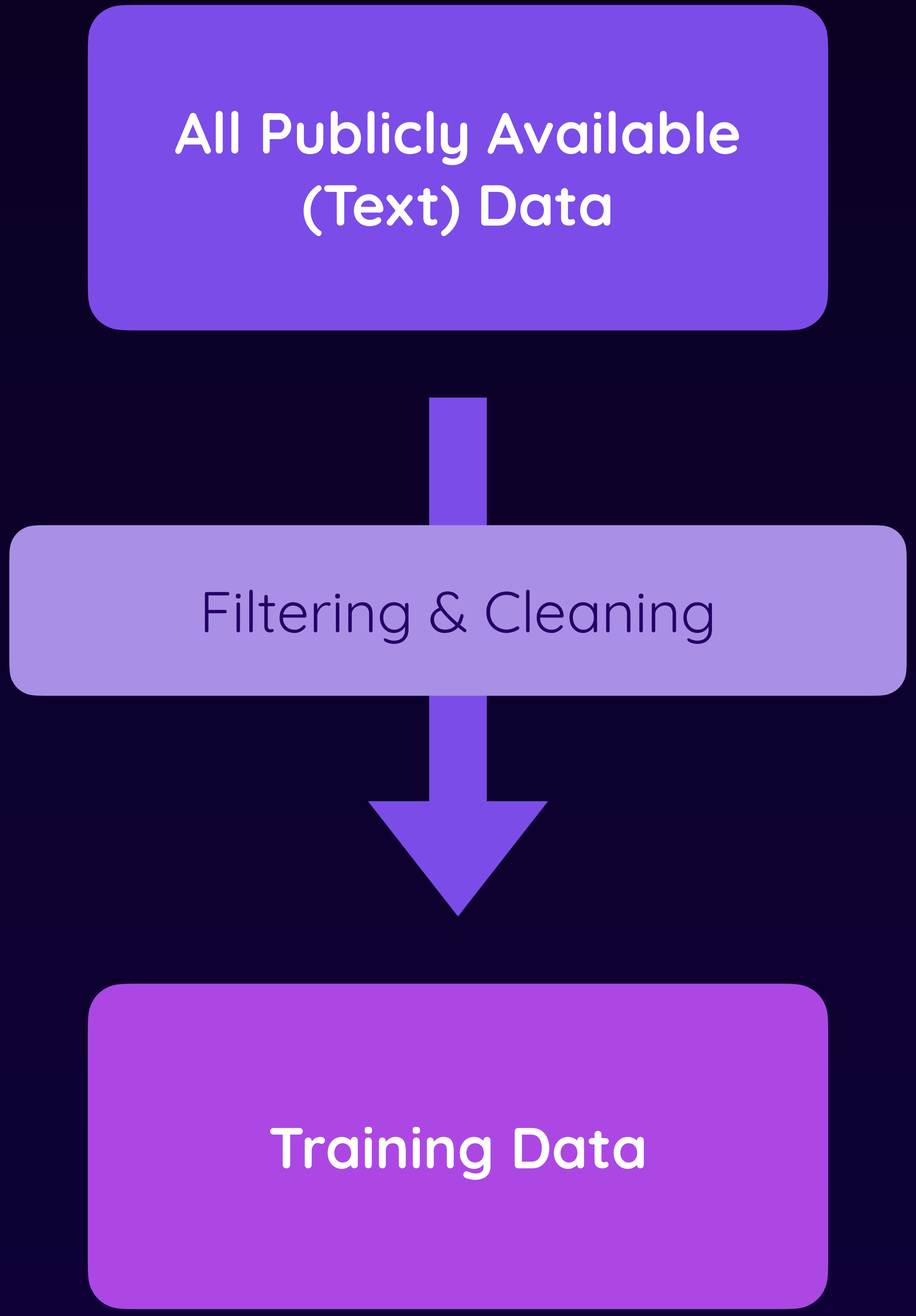
LLMs Are (Huge!) Neural Networks



Each connection has a weight assigned to it

These weights are the so-called **parameters**—they are derived during the model training phase

Training LLMs Requires Text—Lots Of Text



All Publicly Available (Text) Data

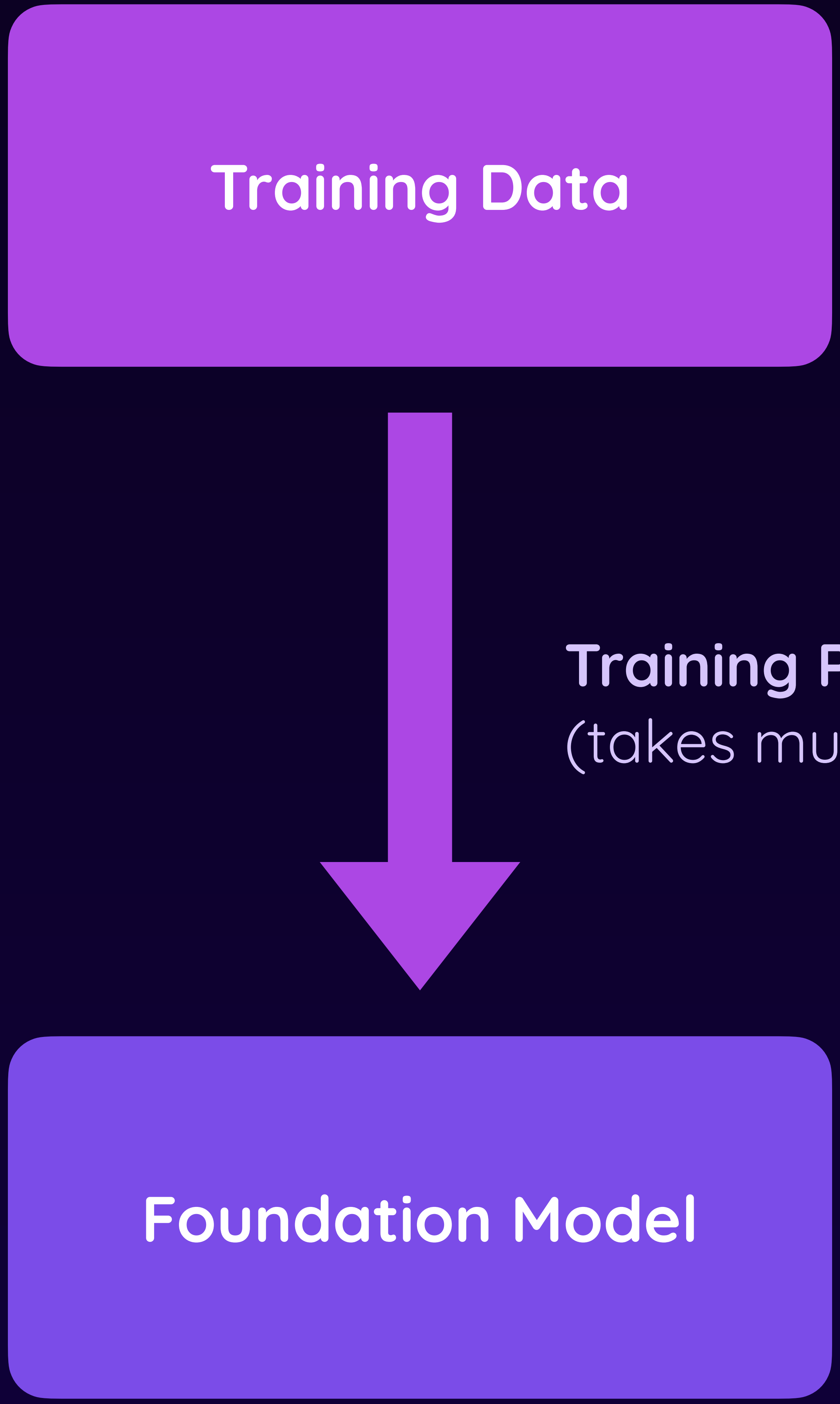
Filtering & Cleaning

Training Data

News articles, blog articles, forum discussions, …

Example: Filter out adult content, illegal content, …

Training a Foundation Model



Training Data

Training Process
(takes multiple months for large models!)

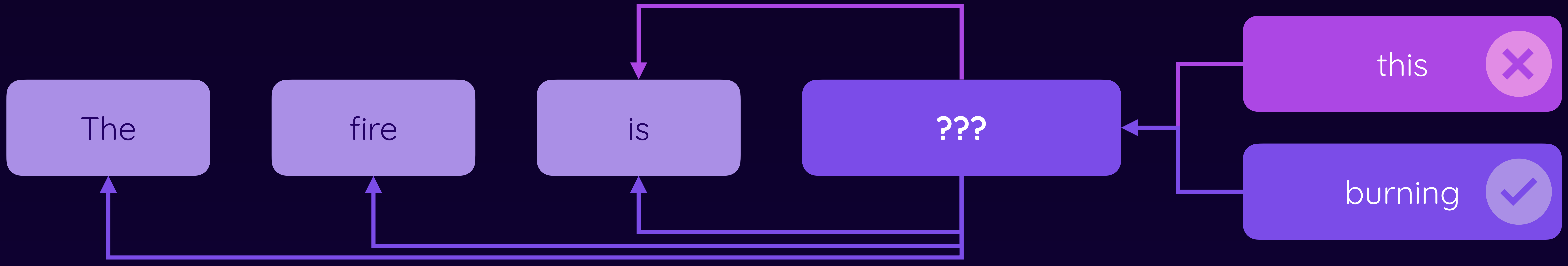
Foundation Model

A LLM that “understands” the relation between words & sentences

This model is able to generate text based on a received input sequence of words

Understanding Relations Between Words

Large Language Models “understand” the relation between words
This allows them to predict future words & complete sequences



LLMs **don’t** just understand relations between subsequent words

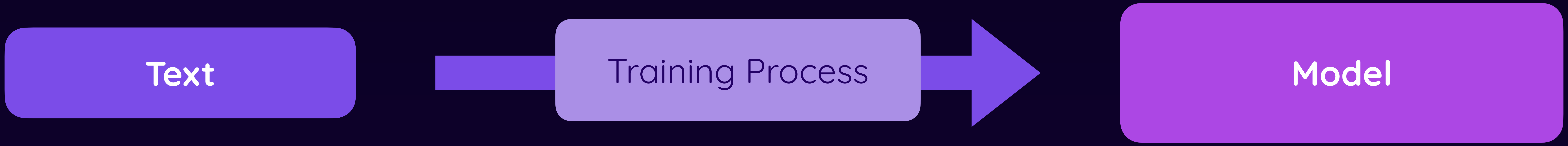
Instead, they understand complex multi-word sequences

A naive model might suggest “this” since it makes sense after “is”

An LLM may suggest “burning” since it understands the entire sentence

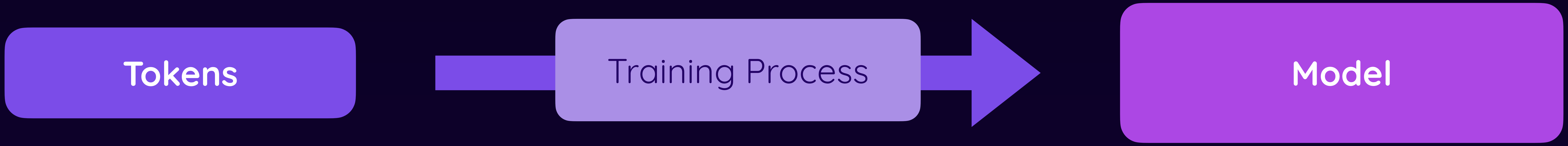
LLMs Are Trained With (Lots Of!) Text

LLMs are trained with large amounts of text



From Text To Tokens

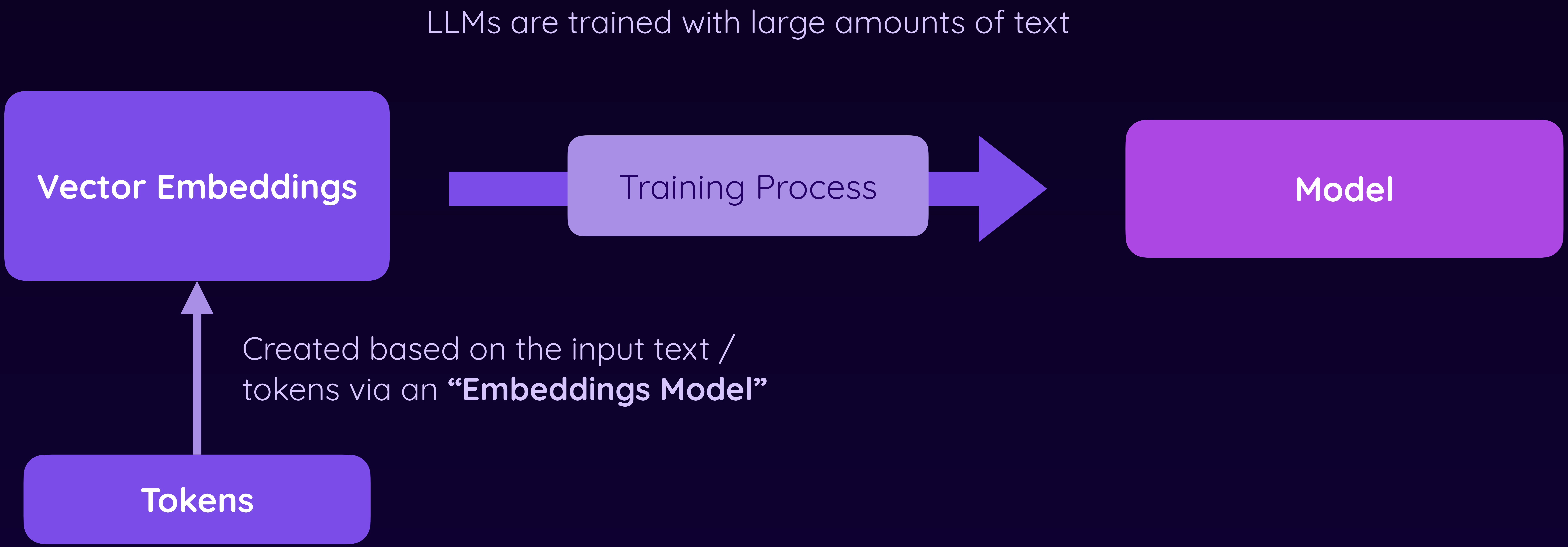
LLMs are trained with large amounts of text


Tokens

Training Process

Model

LLMs Operate on Vector Embeddings



LLMs are trained with large amounts of text

Vector Embeddings

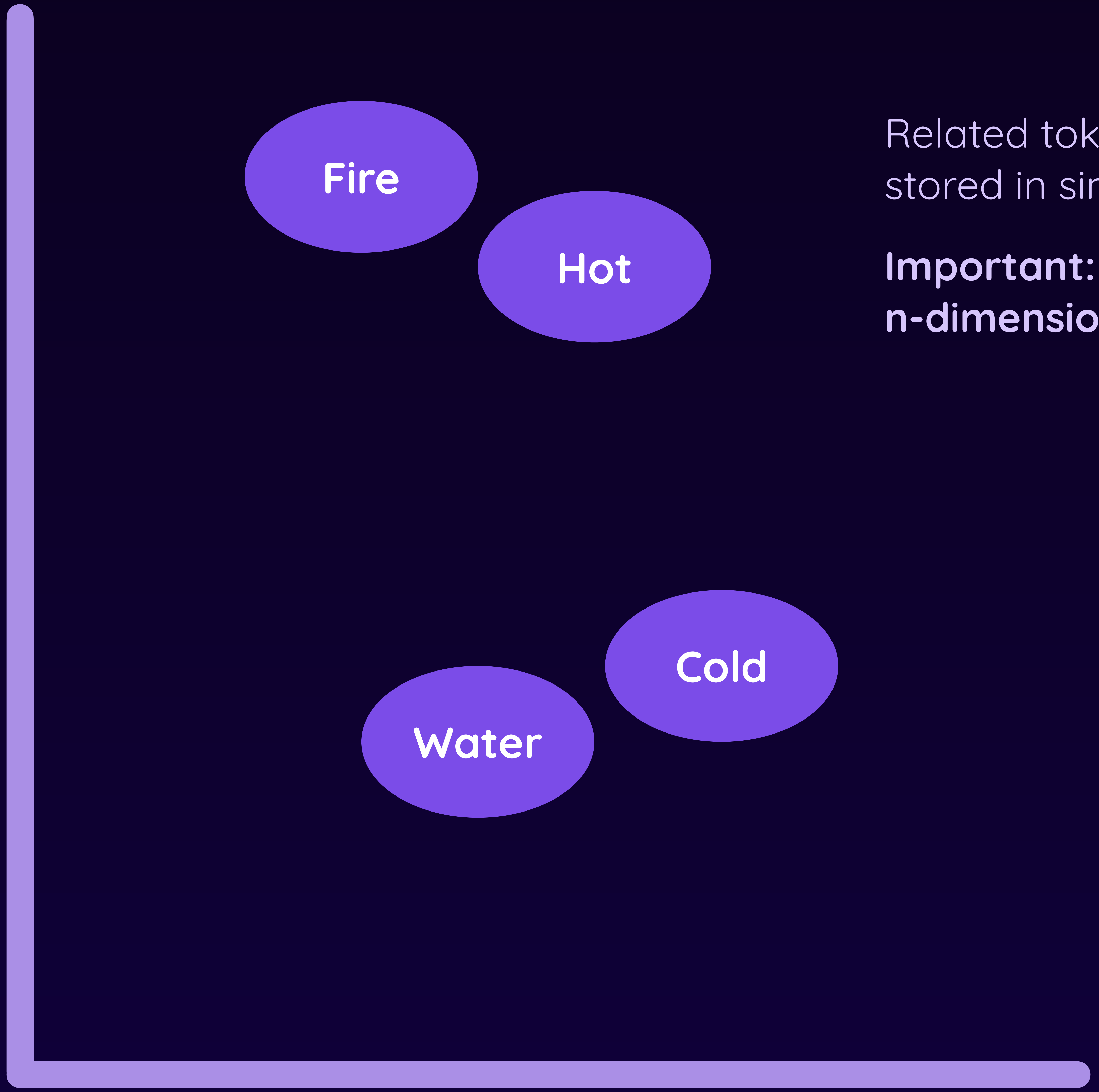
Training Process

Model

Created based on the input text / tokens via an **“Embeddings Model”**

Tokens

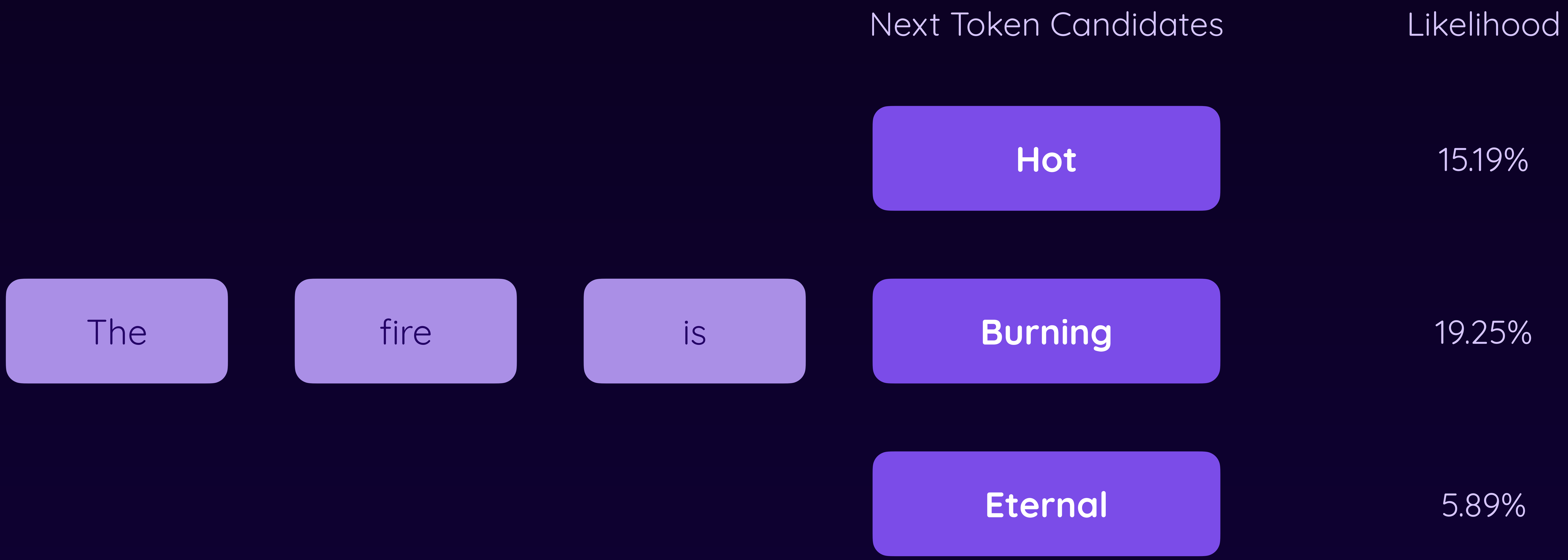
Vector Embeddings Represent Relations



Related tokens / words are stored in similar places

Important: In reality, it's a n-dimensional space!

LLMs Generate Tokens



The fire is

Next Token Candidates	Likelihood
Hot	15.19%
Burning	19.25%
Eternal	5.89%

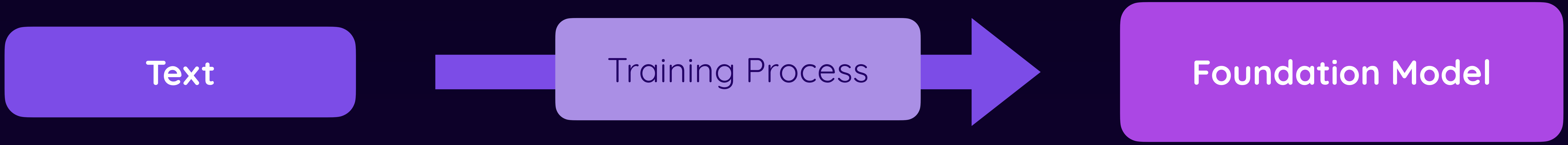
The application then chooses one of those candidates -
based on the probability and possibly also other settings

Foundation Models

Training phase: aim to learn models to predict likely tokens.

Finetuning phase: aim to transform the model for the purpose (for instance an AI assistant).

After finishing training, you have a **foundation model**



Foundation models are models trained only to predict next tokens so at the beginning they can just complete the sentences not necessarely answer to questions like a chatbot.

Foundation Models are LLMs that can complete word / token sequences

They don't necessarily make for great AI assistants or chatbots, though!

User What's the meaning of life?

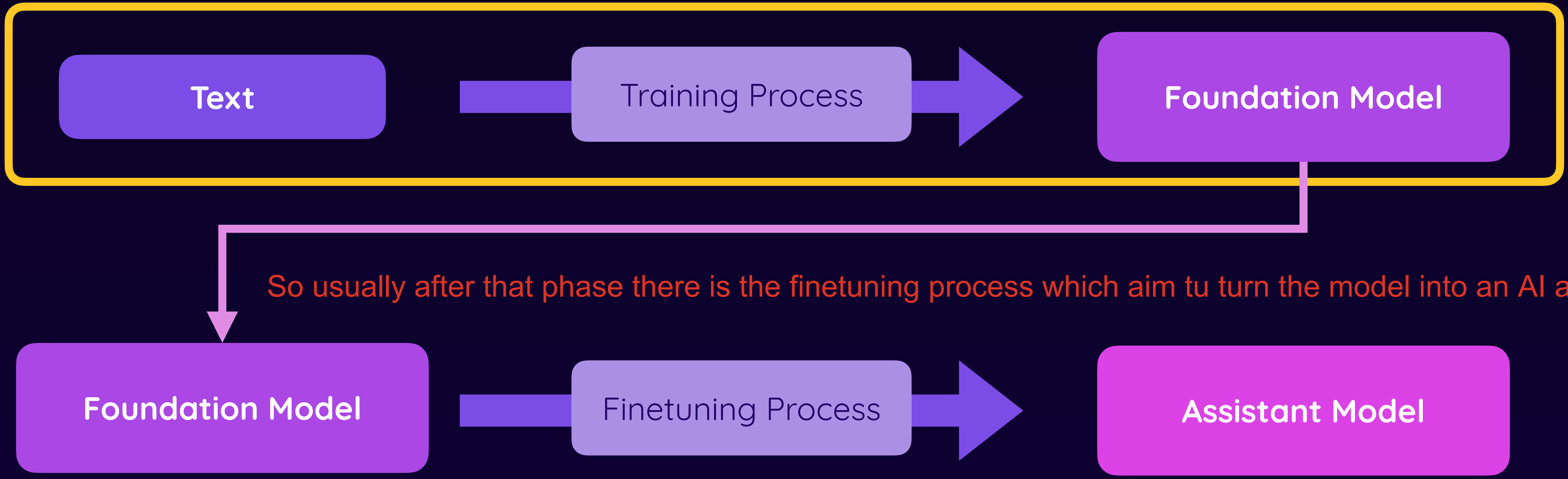
Foundation Model That's the question. And also: Why do we even exist? Will we ever find out?

From a helpful **AI assistant**, you **would expect an answer, not just a completion.**

From Foundation Models To Assistants

This phase is called **"pre-training"**

The result of the pre-training phase is a **foundation model**



Text

Training Process

Foundation Model

So usually after that phase there is the finetuning process which aim tu turn the model into an AI assistant.

Foundation Model

Finetuning Process

Assistant Model

During finetuning, the foundation model is used as a base to train a **finetuned assistant model**

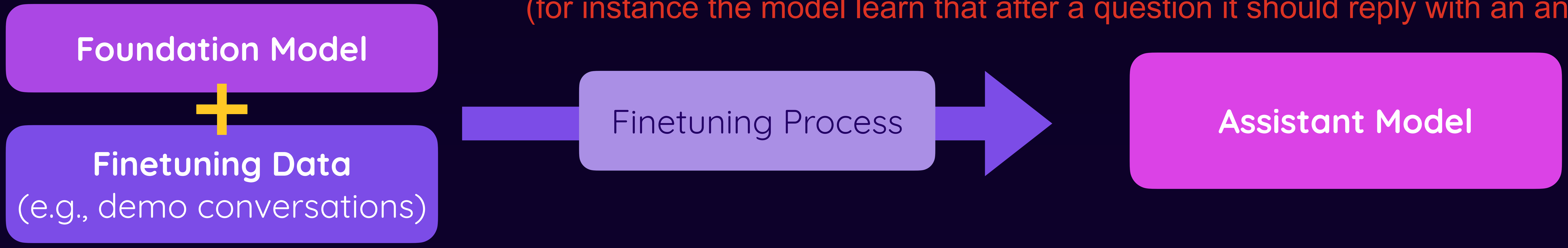
Finetuning is performed by adding (human-created) training data that simulate human<=>AI assistant interactions

Assistant Models

It's also worth noting that as part of this fine-tuning process, the model, of course, is adjusted such that it maybe doesn't answer certain kinds of questions or that it always stays friendly so that even if you throw insults at the model, it does not insult the user, but instead stay friendly, because, of course, you can also put some angry conversations into the fine-tuning training data and then make sure that in that training data you're providing, the assistant always stays calm, replies in a kind way, and therefore, of course, the model learns, so to say, to also behave in that manner, though it's always dangerous to say "learn" and "behave", because still we're talking about token predictions. But it simply gets more likely that nice tokens are predicted as the next tokens, uh, based on that extra fine-tuning training data.

After finishing finetuning, you end up with an assistant model

So extra (human created) data are added during finetuning
(for instance the model learn that after a question it should reply with an answer).



Assistant models are still LLMs that predict future tokens

But they are finetuned to generate tokens in-line with the finetuning data

Supervised Fine-Tuning (SFT)
1) Human-created examples (like Q&A pairs or CoT reasoning steps) are added.
2) The model is fine-tuned to behave more like an assistant
(answering questions, following instructions, and sometimes reasoning step-by-step).
3) This is where CoT is introduced:
Either explicitly (via examples that include reasoning steps),
Or implicitly (by showing that step-by-step answers are preferred).

User What's the meaning of life?

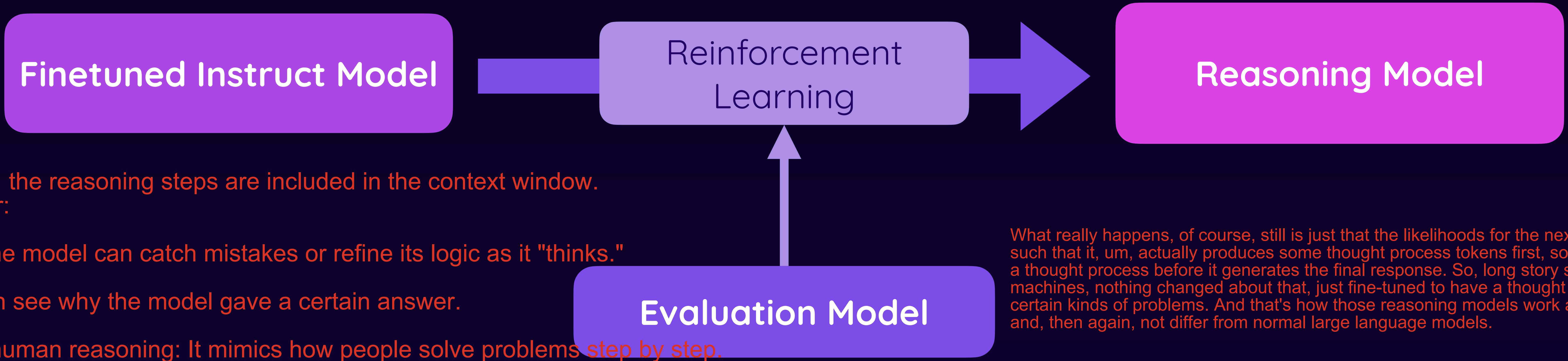
Foundation Model Great question! Whilst there is no definitive answer …

When you interact with ChatGPT etc., you're interacting with assistant models
(also called "instruct models")

Onwards To Reasoning Models

Models that are finetuned to print the CoTs before provide the solution, useful for math or more complex problems, because it increase the likelihood of better responses.

Some modern LLMs can "think" before they answer



in many implementations, the reasoning steps are included in the context window.
This "thinking" matters for:

1) Improved accuracy: The model can catch mistakes or refine its logic as it "thinks."

2) Transparency: You can see why the model gave a certain answer.

3) Better alignment with human reasoning: It mimics how people solve problems step by step.

What really happens, of course, still is just that the likelihoods for the next token are shifted such that it, um, actually produces some thought process tokens first, so some tokens that describe a thought process before it generates the final response. So, long story short, still token prediction machines, nothing changed about that, just fine-tuned to have a thought process first, at least for certain kinds of problems. And that's how those reasoning models work and how they differ and, then again, not differ from normal large language models.

During the training phase, problems are given to the instruct model

The responses / results are than evaluated & graded by a separate evaluation model

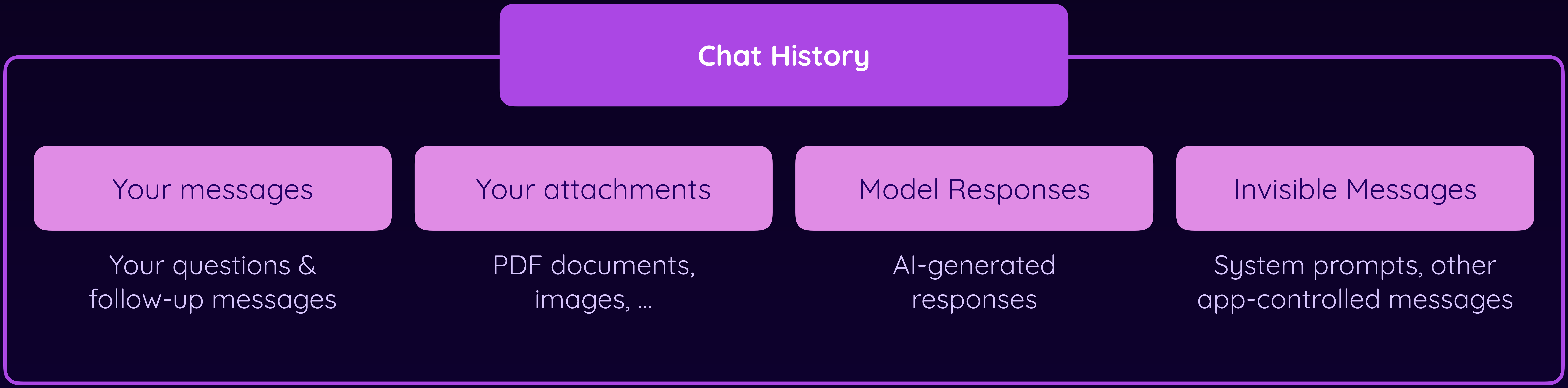
That "feedback" is then used to adjust the LLM parameters until satisfying results are produced consistently

Reasoning models are still predictive models at their core—they generate the next token based on probabilities. However, they are fine-tuned to simulate a reasoning process before giving a final answer.

Going through an initial "thinking process" was "learned" during this training phase

2 ways to obtain reasoning models:
1) Structured Training: These models are trained on datasets where each example includes a thought process followed by an answer.
2) Fine-Tuning Method: (more tipical) Instead of relying solely on human-written reasoning examples, models are often prompted with questions and their outputs are evaluated by another model or by code/humans. This process is called Reinforcement Learning: The model is updated based on whether its reasoning and final answer are good, gradually learning to produce better intermediate reasoning steps.
Outcome: The model learns to generate a chain of thought before the final answer, especially for tasks that benefit from reasoning (like math or logic problems).
Still Token Predictors: Despite the added reasoning behavior, they remain fundamentally token prediction machines—just trained to predict reasoning steps first.

LLMs Have Context Windows



Chat History

Your messages

Your questions & follow-up messages

Your attachments

PDF documents, images, …

Model Responses

AI-generated responses

Invisible Messages

System prompts, other app-controlled messages

Context Window

This entire history must be processed by the LLM for every (!) new response it generates

It's the "available context" the model takes into account for its response generation

Context window: Each LLM has a **maximum amount of tokens** it can consider

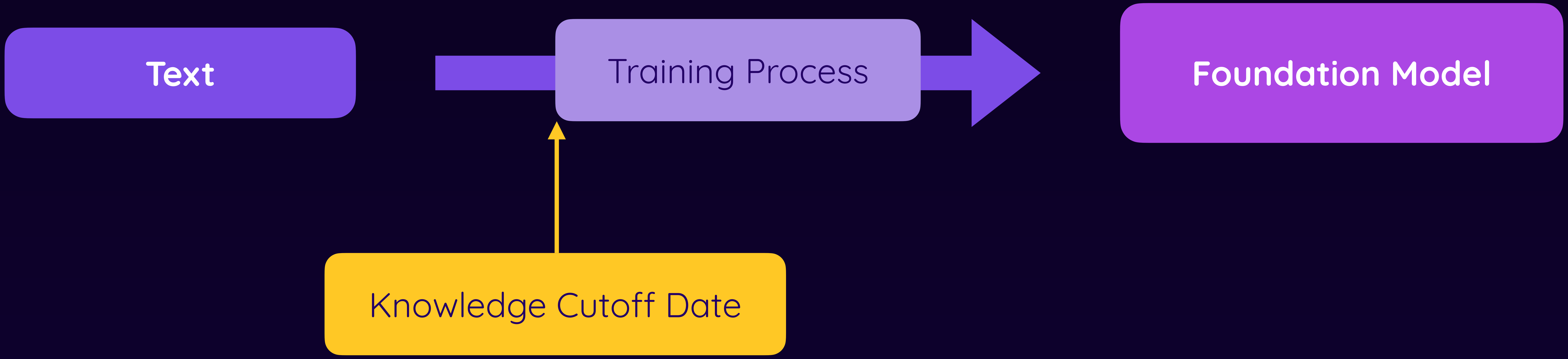
It take into account not only the prompt size but also the chat history (your profile included).

Knowledge Cutoff

Some potencial LLms' limitations:
1) Hallucinations (ever possible)

2) Knowledge cutoff (several solutions)

3) Context window (for some models and depending by the request).



The model can't "know" any facts after the start of its training process

Solution 1: Prevent Hallucinations

Finetune data to "detect" questions it's likely not able to answer

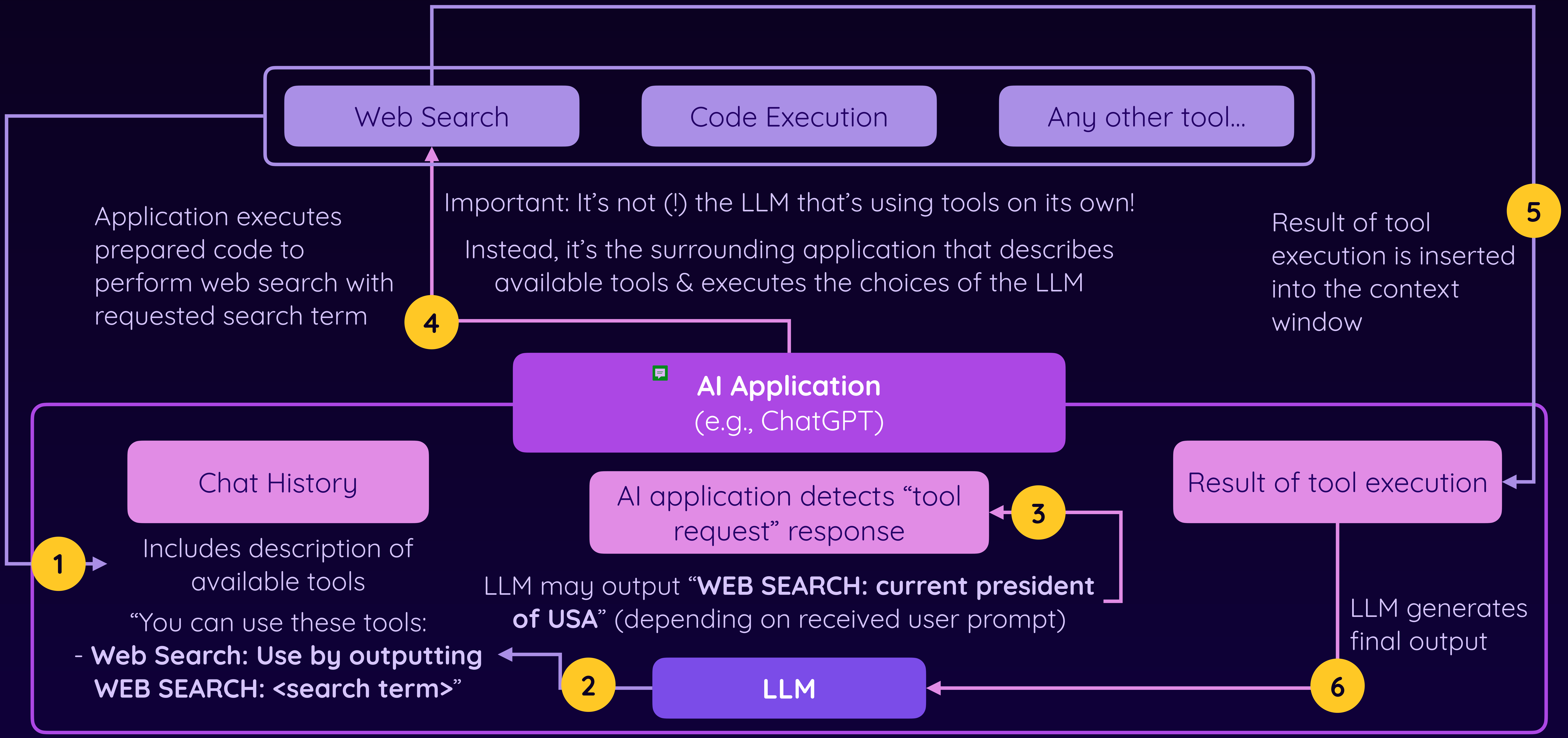
Generate a generic response ("Sorry, I don't know that")

Solution 2: Expand Knowledge

Give AI application access to web search & finetune to detect relevant questions

Application can perform web search & include results in context

How AI Assistants Use Tools



Web Search

Code Execution

Any other tool...

Important: It's not (!) the LLM that's using tools on its own!

Instead, it's the surrounding application that describes available tools & executes the choices of the LLM

AI Application
(e.g., ChatGPT)

Chat History

1

Includes description of available tools

"You can use these tools:
- **Web Search: Use by outputting WEB SEARCH: <search term>**"

2

LLM

LLM may output "**WEB SEARCH: current president of USA**" (depending on received user prompt)

3

AI application detects "tool request" response

4

Application executes prepared code to perform web search with requested search term

5

Result of tool execution is inserted into the context window

Result of tool execution

6

LLM generates final output

It’s All Highly Dynamic!

ChatGPT

Still an AI chatbot but …

… more models

… more capabilities

… older models are gone

AI chatbots & their capabilities keep evolving

And different AI model providers offer different features

Running LLMs Locally

ChatGPT & Co vs Self-hosted Solutions

- ChatGPT & Co Disadvantages
- Exploring Open Models
- Understanding Quantization
- Running LLMs Locally

Typically, the weights / parameters are “open”

Not the code / algorithm

ChatGPT & Co Disadvantages

Cost

Most models are not accessible for free

Having multiple subscriptions can add up

API access is charged based on usage

Availability

No access without a (stable) internet connection

Services may go offline (e.g., due to demand)

Usage limits may apply

Privacy

You're sharing your data with the model providers

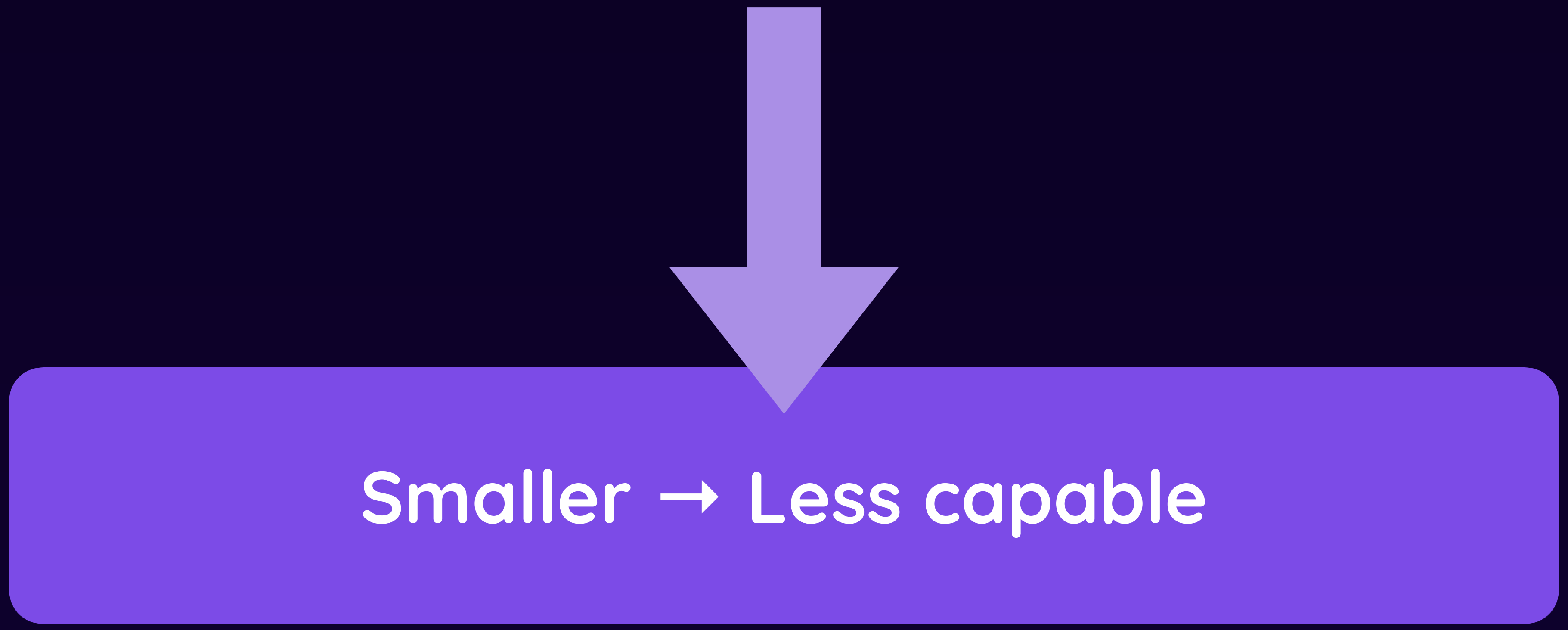
Usage restrictions may apply

But: If you want / need to use the **most powerful, versatile and knowledgeable models**, there's (almost) no way around ChatGPT & Co

Almost: There is DeepSeek - an open-source model provider. But running / **self-hosting large LLMs is very difficult & costly**

Onwards To Open-Source Models

Most open-source models are **smaller** (i.e., less parameters) than the most capable proprietary models



Smaller → Less capable

Depends on your use-case!

For many tasks, a small, locally running model may be equally good or even better than ChatGPT & Co

Summarization Tasks

Data Extraction Tasks

Local Knowledge Tasks

Open-Source Models—An Overview

Whilst most OpenAI models are proprietary (i.e., you can't self-host them),
there are **plenty of popular & powerful open-source models** available

Meta's Llama Models

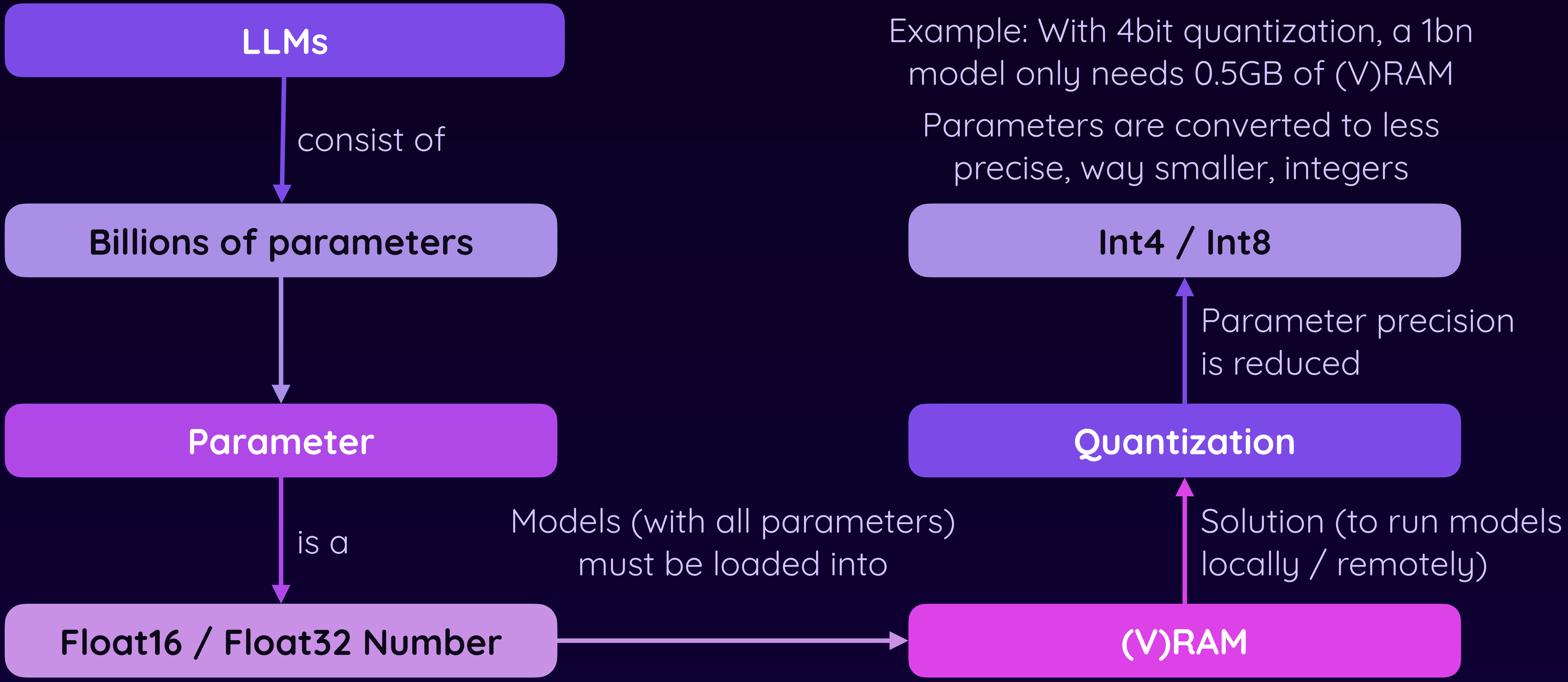
Google's Gemma Models

DeepSeek's Models

And many others!

Huggingface is a great place to explore all those open-source models in detail!

Understanding Quantization



LLMs

consist of

Billions of parameters

Parameter

is a

Float16 / Float32 Number

Models (with all parameters) must be loaded into

(V)RAM

Solution (to run models locally / remotely)

Quantization

Parameter precision is reduced

Int4 / Int8

Example: With 4bit quantization, a 1bn model only needs 0.5GB of (V)RAM

Parameters are converted to less precise, way smaller, integers

0.1234130859375

16 bits (2 bytes) per parameter in a Float16 model

Example: A (very small!) 1bn parameter model needs >2GB of (V)RAM

Running LLMs Locally

There are different solutions that make running open-source LLMs locally a breeze

LM Studio

Ollama

llama.cpp

You can also use open-source models programmatically via Ollama & Co APIs OR via Huggingface transformers library

Self-hosting LLMs

You can also run (host) open-source models on owned / rented servers

On-premise / VPS	Managed Service
You own / rent & configure the server	You use a managed service like Amazon Bedrock
You manage the software and install Ollama / LM Studio etc.	You select models & features → No setup required
You configure the network	No technical expertise required
You only pay for the server	You pay for the usage

❗

Important

Self-hosting LLMs on a remote server is **not cheap** and **not necessarily trivial**

You will need at least somewhat capable hardware and go through various setup steps to run and expose LLMs from a remote server

Prompt Engineering

Ensuring Good Results

- What & Why?
- Understanding Key Techniques
- Best Practices & Recommendations

What Is Prompt Engineering?

What Is A Prompt?

What Is Prompt Engineering?

The Skill Of Crafting Great Prompts

Why Prompt Engineering?

Good Prompts

=

Good Results

Main Prompt Engineering Goals

Control Output **Content**

A LinkedIn Post

+

Control Output **Format**

Plain Text, Markdown, JSON, …

Main Prompt Engineering Goals

Control Output **Content**

You want the LLM to generate content that you can use with as little modifications as possible

Of course, you also want a response that's correct, contains no hallucinations and meets any other requirements you might have

+

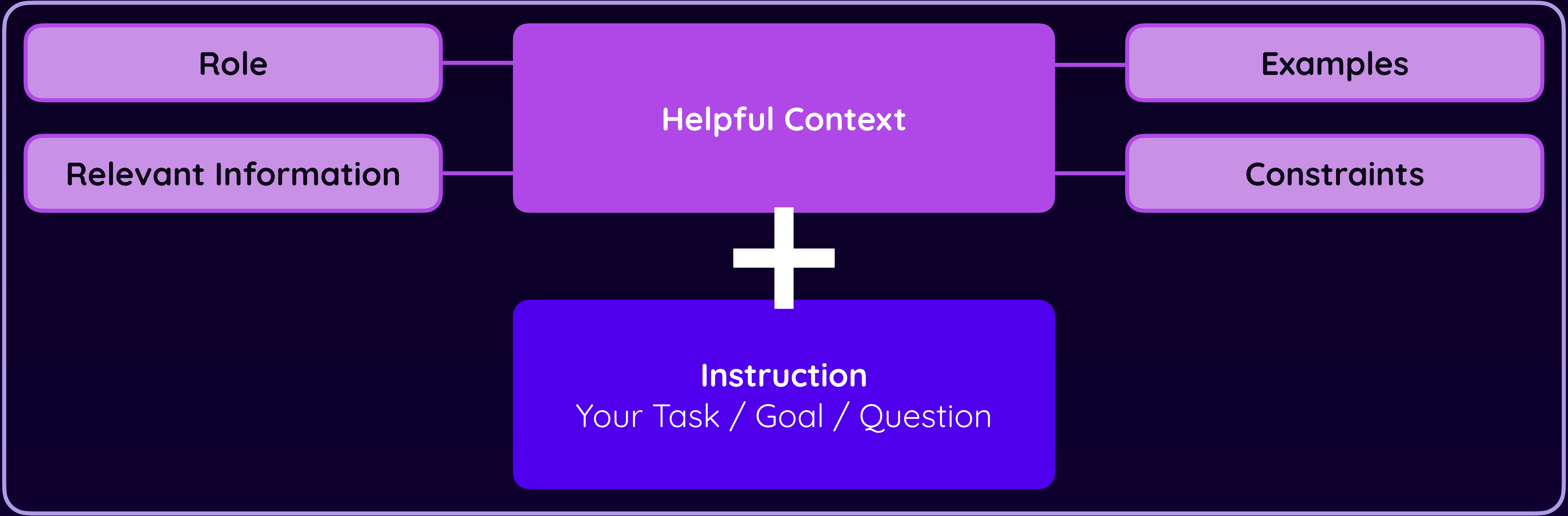
Control Output **Format**

For some (but not all) use-cases, you also might care about the format

You might want a response formatted as JSON or markdown

Or maybe you need text that's structured as a list of bullet points

What Defines A Good Prompt?



A good prompt includes a detailed task description and helpful context

Irrelevant information must be avoided

Complex (multi-task) prompts should be avoided

⚠

Keep In Mind

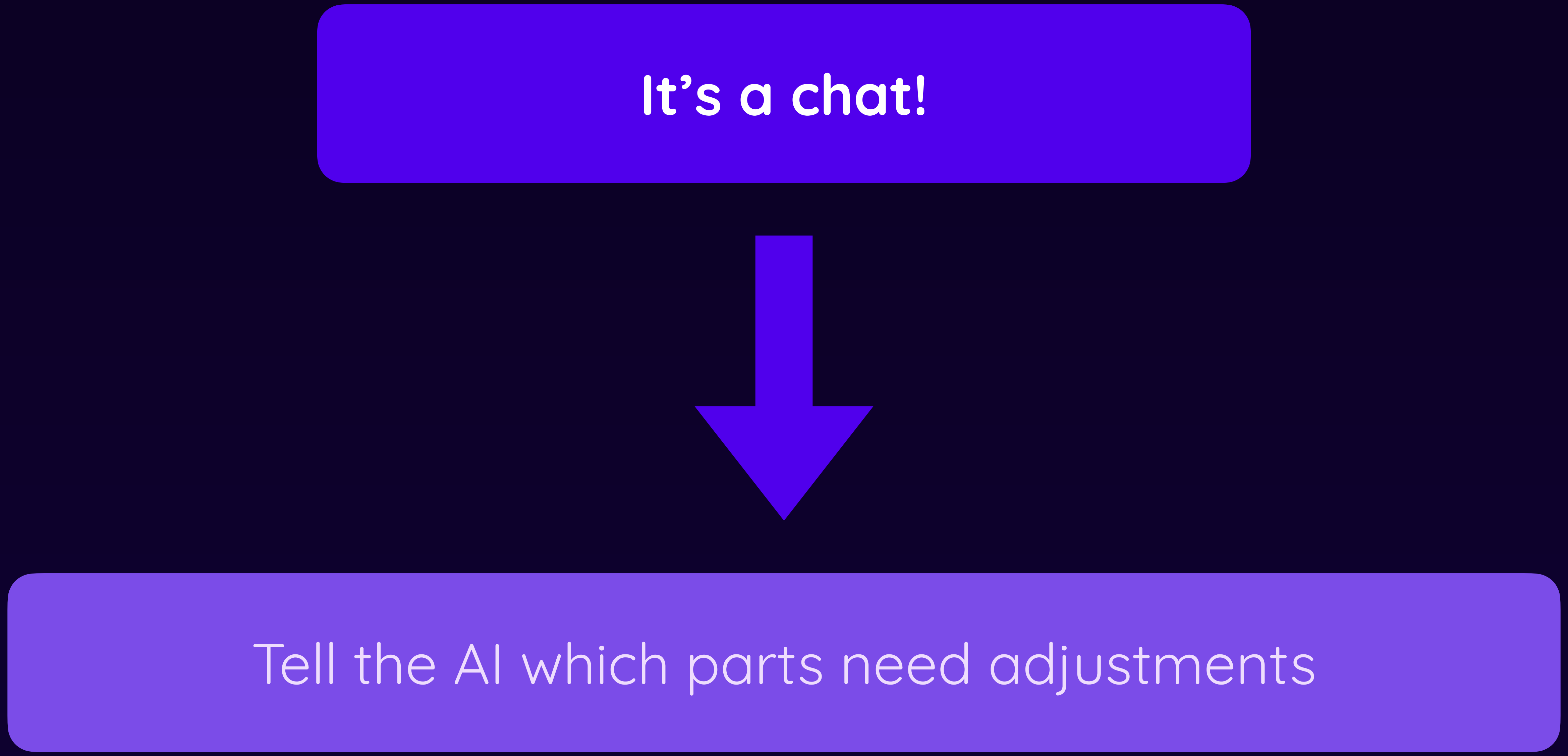
ChatGPT Output Is A Starting Point

Fine-tune & adjust as needed

Adding Meaningful Context

- Prefer **short**, focused **sentences**
- Add important **keywords** & avoid unnecessary information and ambiguity
- Define the **target audience**
- Control **tone, style & length** of the output
- Control the **output format** (text, JSON, unformatted markdown, …)

Refine Results Over Time



It’s a chat!

Tell the AI which parts need adjustments

“Remove the Emojis and all hashtags.”

Hands-On!

1 Create a short product announcement text for a new AI-powered website generator

2 Create a Python code snippet that searches & deletes all .png & .jpg files in a given folder

3 Write an email to a colleague that you need feedback on your submitted prototype until end of the week

Key Prompt Engineering Techniques

- Zero- and Few-Shot
- Chain-of-Thought
- Using Delimiters
- Output Templates
- Persona Prompting
- Output Formatting
- Contextual Prompting
- Negative Prompts
- Self-reflective Prompting

Zero- & Few-Shot Prompting

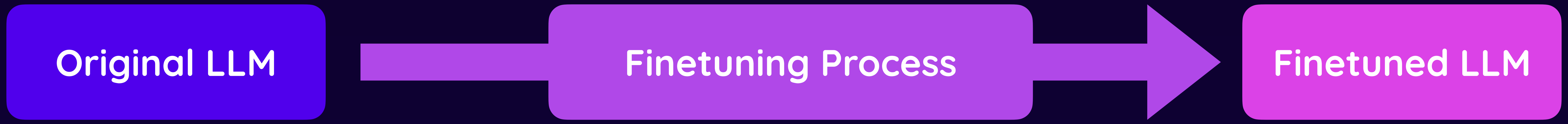
Providing examples can help fine-tune the result tone, style & content

0	N
Zero-Shot	**Few-Shot**
Provide no examples	Provide multiple examples
“Write a post that explains the core idea behind ChatGPT.”	“Write a post that explains the core idea behind ChatGPT. Use a similar tone & structure as I do in my regular tweets. But don’t use the content. Here are **two example posts**: …”

Finetuning Models

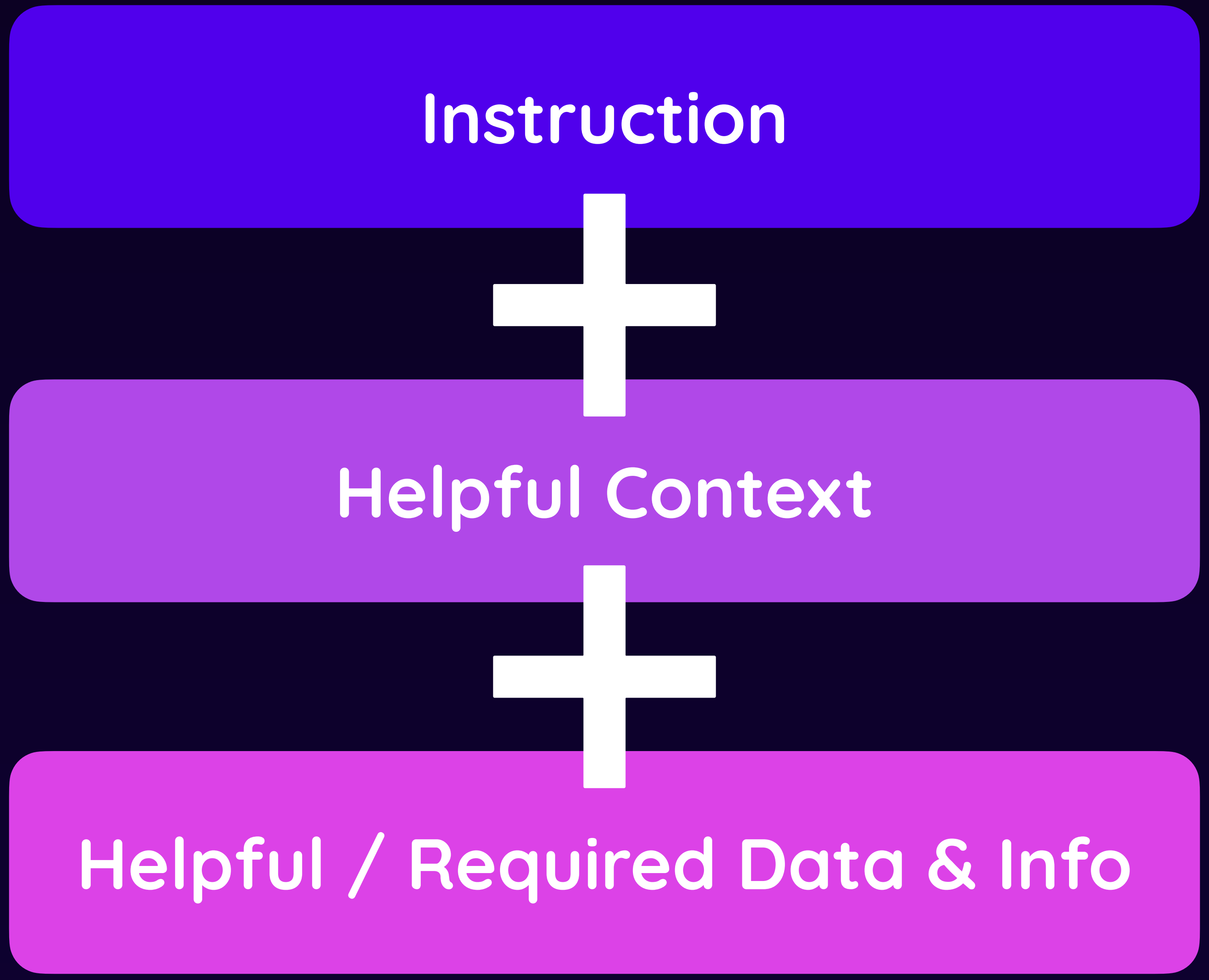
If you need to provide many examples, you could consider finetuning

Finetuning **changes the model’s internal weights and values**, leading to a different model behavior



Important: Finetuning can be expensive & the model may perform worse at other tasks thereafter!

Include Relevant Information



Instruction

+

Helpful Context

+

Helpful / Required Data & Info

ACADEMIND

Generating Images & Videos with AI

From Text To Images & Video

- Available Options
- Using AI For Generating Images
- Using AI For Generating Videos

AI Image Generation — Available Options

There are many options!

ChatGPT & OpenAI

- ChatGPT Image Generation
- OpenAI API Image Generation

Other Providers & Models

- Gemini, Copilot, Grok, ...
- Midjourney
- Flux, Stable Diffusion

Integrated Tools

- Adobe Photoshop

And many, many others!

ChatGPT For Programming

Writing Code with ChatGPT

- Generating Code With & Without Programming Experience
- Debugging & Optimizing Code
- Explaining & Refactoring Code

ChatGPT Is Great For Everyone!

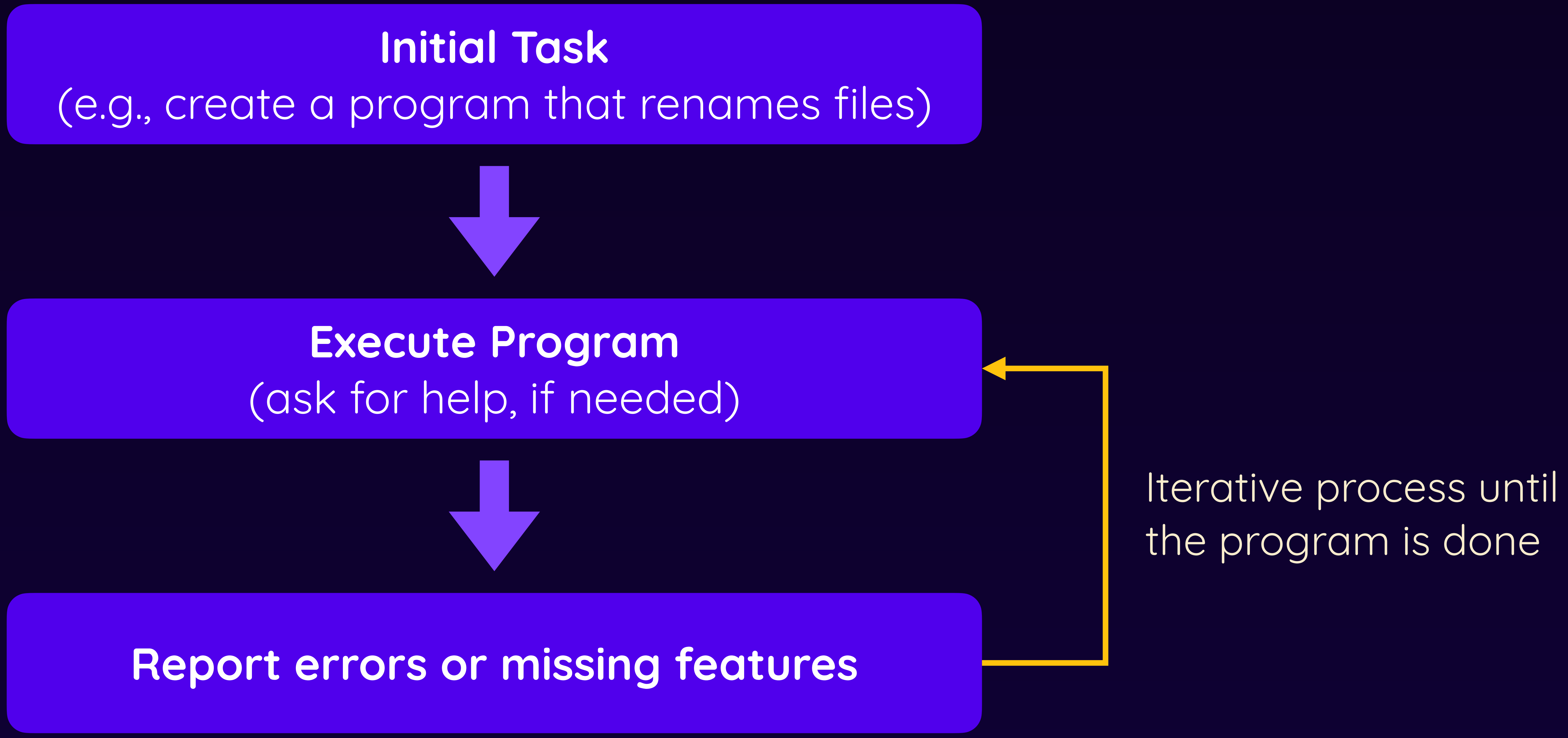
Non-Developers

Build basic programs (utility scripts) & websites with zero development experience

Developers

Boost your productivity, outsource the boring parts, generate dummy data & code faster

ChatGPT For Non-Developers



Initial Task
(e.g., create a program that renames files)

Execute Program
(ask for help, if needed)

Report errors or missing features

Iterative process until the program is done

Exercise Time!

Let ChatGPT build a **basic website**!

Starting Page

- Your name
- Image
- List of hobbies

CV Page

- List with career history

Website should have a modern, clean, dark-mode styling.

Be Careful

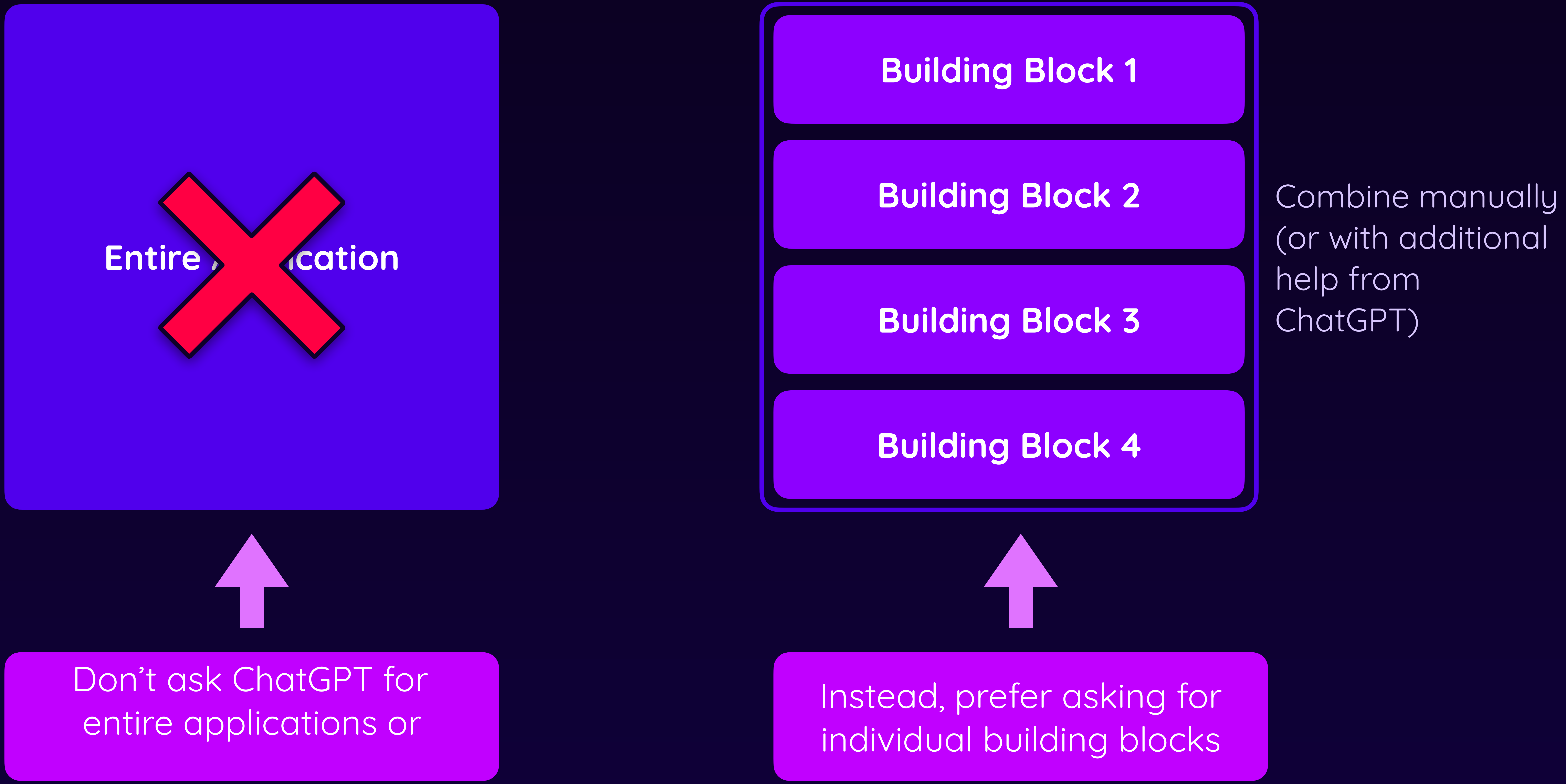
Code you don’t know could cause harm!

Depending on your request & prompt it could delete files, erase data, crash your system etc.

ChatGPT 🤝 Developers

As a developer, you can massively **boost your productivity** by using ChatGPT!

ChatGPT For Developers



Combine manually (or with additional help from ChatGPT)

Don't ask ChatGPT for entire applications or

Instead, prefer asking for individual building blocks

ChatGPT For Developers

Building Blocks > Entire Apps

Use ChatGPT to speed up the development of the individual application building blocks

Iterative Development

Add more and more features by splitting your requests across multiple prompts

Refine Code Manually

Instead of deriving fancy prompts, consider performing fine-tuning tasks manually

Use ChatGPT For Debugging

Report errors & bugs (+ relevant code snippets) to ChatGPT to speed up debugging

Explain Code

Instruct ChatGPT to quickly explain & summarize unfamiliar code

Use ChatGPT for Refactoring

Let ChatGPT refactor code or use ChatGPT to get improvement ideas

Don't Limit Yourself To Just ChatGPT!

ChatGPT

Great for generating entire building blocks

+

GitHub Copilot

Great for generating smaller code snippets "on the fly" & fine-tuning your code

Useful Prompting Techniques

If Code Gets Cut Off

“Output ‘Continuing’ and continue”

Skip Explanations

“Provide just the code without any extra explanations or text.”

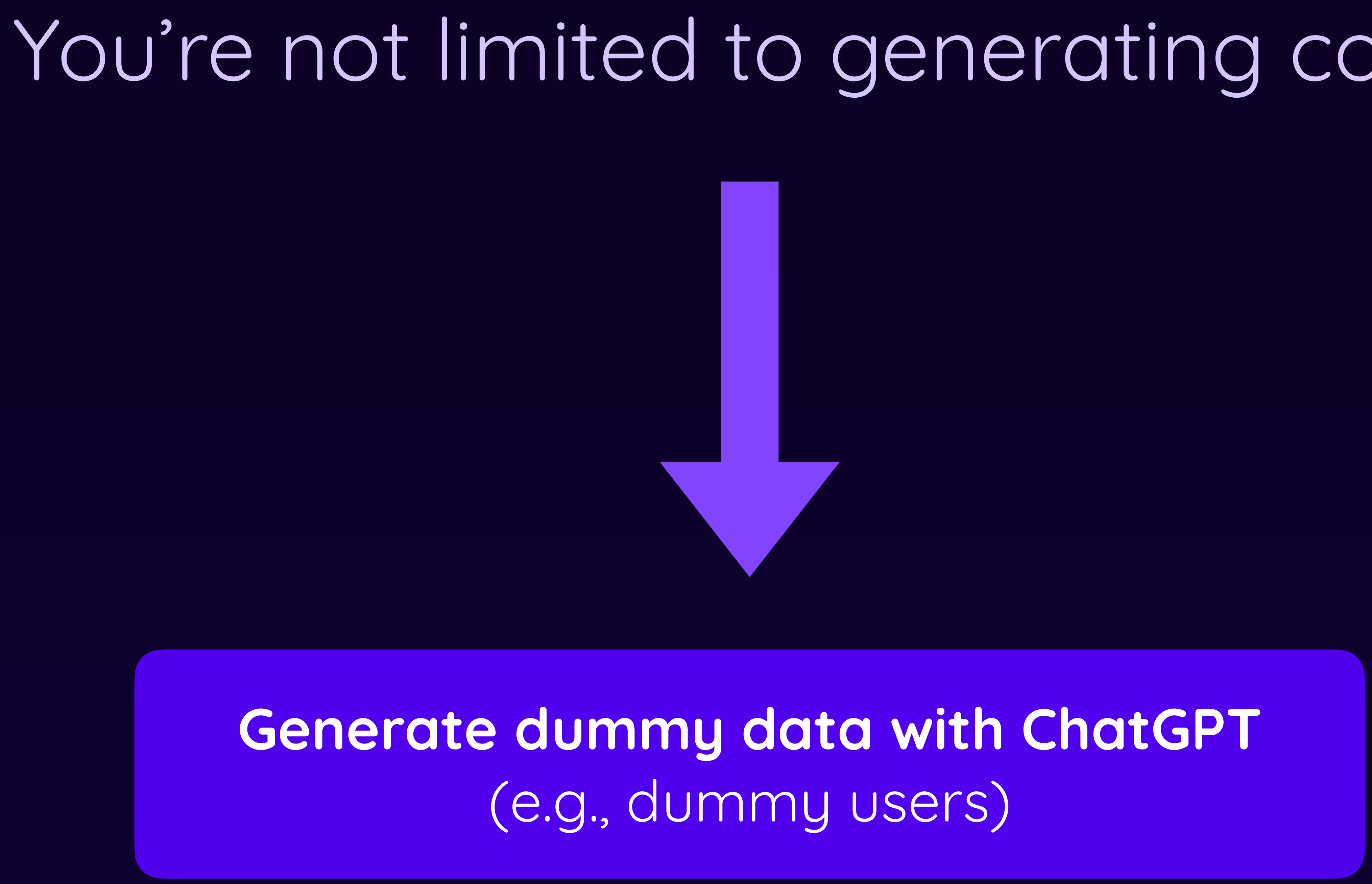
Add Context To Errors

“The user authentication code seems to break the program with the following error message: [message]”

Let ChatGPT Improve Itself

“How could the code be improved?”

Generate Dummy Data with ChatGPT



You’re not limited to generating code

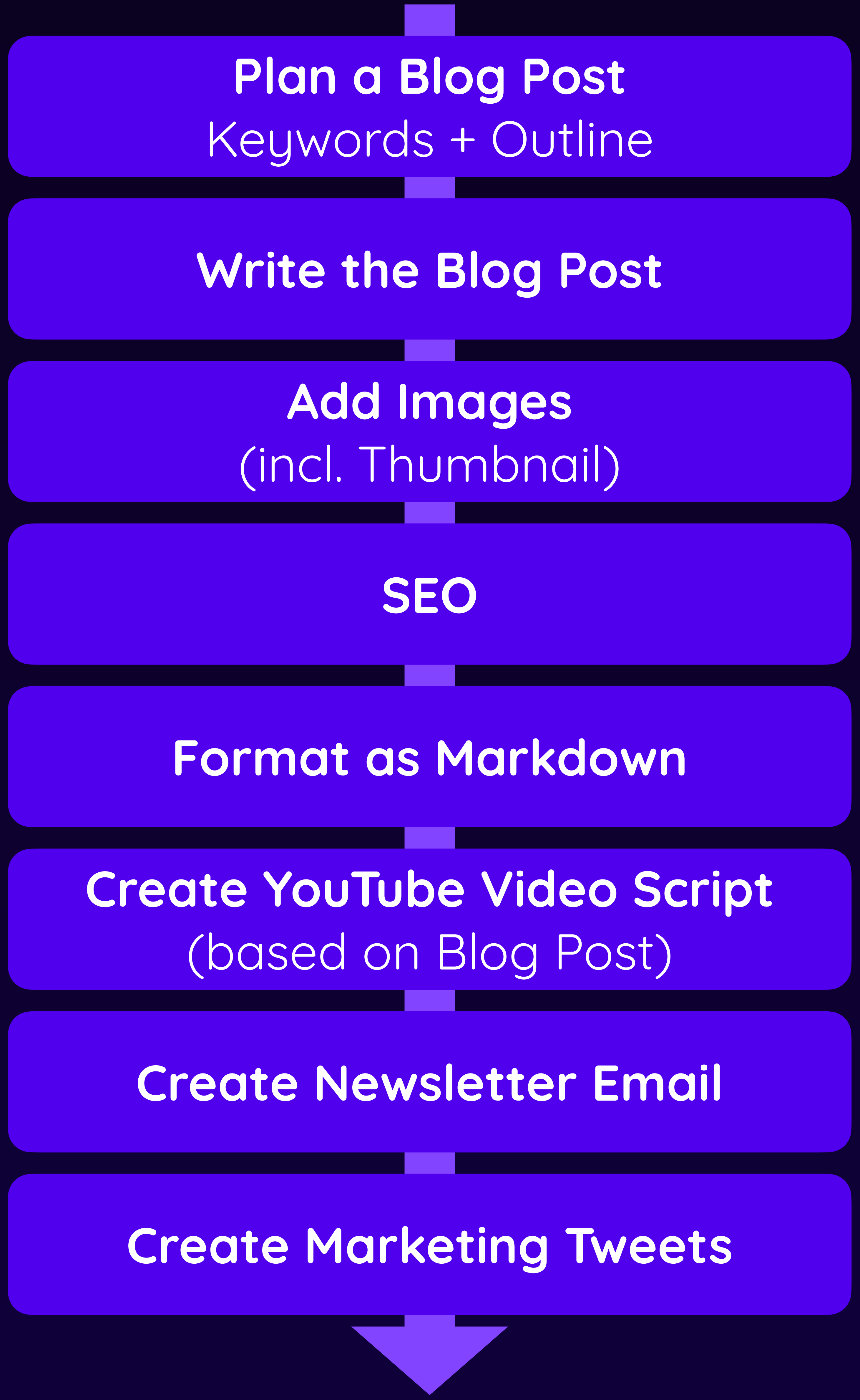
Generate dummy data with ChatGPT
(e.g., dummy users)

Hands-On: ChatGPT Content Creation

Practicing How To Generate Content With ChatGPT

- Create & Advertise a Realistic Blog Post
- SEO
- Add Images (Midjourney)
- Creating a Video Script

Creating Content with ChatGPT



Plan a Blog Post
Keywords + Outline

Write the Blog Post

Add Images
(incl. Thumbnail)

SEO

Format as Markdown

Create YouTube Video Script
(based on Blog Post)

Create Newsletter Email

Create Marketing Tweets

Topic

Understanding ChatGPT & How It Works Behind The Scenes

Hands-On: Programming & ChatGPT

Practicing How To Write Code With ChatGPT

- Non-Developer Example: Building a “Monster Slayer” Game
- Developer Example: Building a Meetups REST API

Building a "Monster Slayer" Game

Using ChatGPT with **no / minimal** programming experience

Building a **text-based / command-line based** game: "Monster Slayer"

Description

It's a **turn-based** game where the user (= player) fights a monster (= computer).
During every turn, the player can perform a **regular** or **strong attack** or **heal**.
The **strong attack** should only be available **every three turns**. **Healing** should only be available **every five turns**.
After each turn, the **monster attacks**.
Damage & heal values are calculated **randomly**.
The first participant to go below **0 health loses**.
Both participants **start with 100 health**.
Once the game is over, the **winner** should be **displayed on the screen** and the player should be **asked if a new game** should be started.

Enhancing The “Monster Slayer” Game

Add Username

Allow the user to choose a username when the program starts

Add Difficulty Levels

Adjust damage & heal values based on chosen level

Manage High Score

Save the number of required turns in text file

Display the current high score after every game

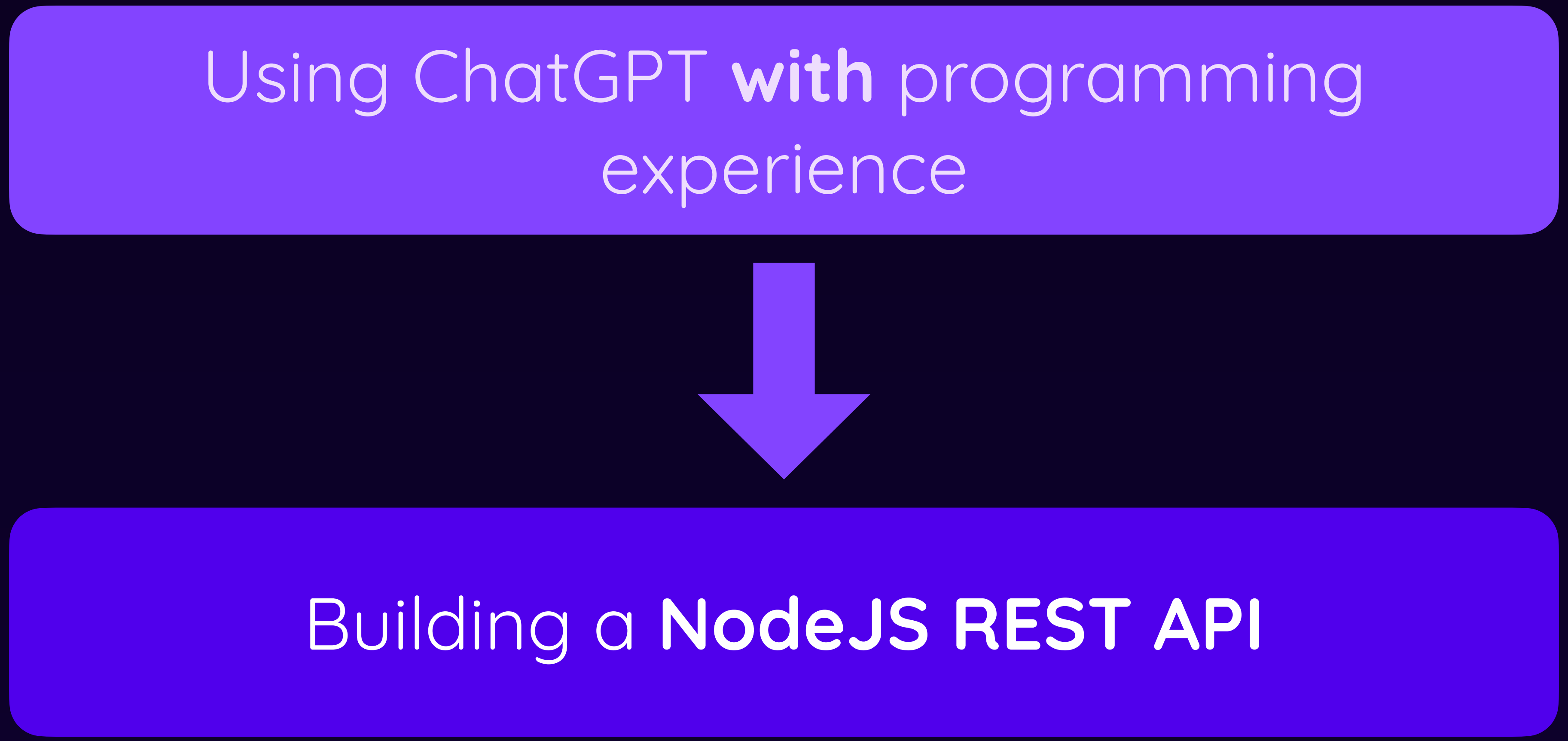
ACADE MIND

Results May Deviate

The Code You Get Will Likely Differ

Because of built-in randomness, different prompts & GPT model evolution, you will very likely not get the same results!

Building a “Meetups” REST API



Using ChatGPT **with** programming experience

Building a **NodeJS REST API**

Description

It’s a REST API with the following **endpoints**:
POST /meetups → Create a new meetup
GET /meetups → Fetch meetups
PATCH /meetups/<id> → Update existing meetup
DELETE /meetups/<id> → Delete existing meetup

Every meetup has an **id, title, summary & address**.
Meetup data should be **stored** in a **meetups.json** file, incoming data must be **validated**.
Data should be exchanged in **JSON format.**

Using AI Programmatically via APIs

Building AI-powered Applications

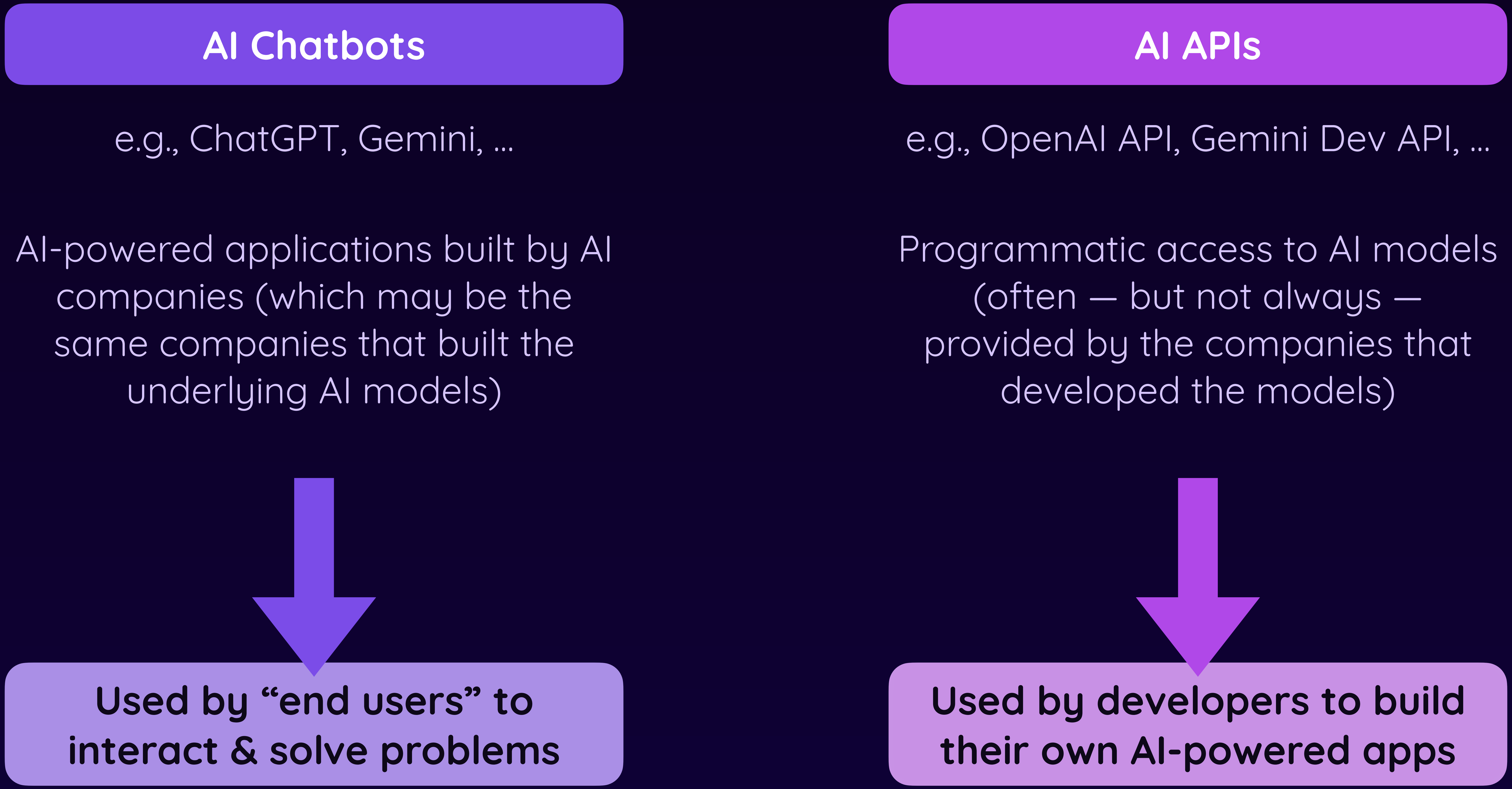
- ChatGPT vs OpenAI API
- Using the OpenAI API

API
Application Programming Interface

A set of rules and protocols that allow software to communicate with other software

For example, the OpenAI API allows your software (e.g., some mobile app, or some internal tool) to use OpenAI's AI models through code

ChatGPT vs GPT APIs



AI Chatbots

e.g., ChatGPT, Gemini, …

AI-powered applications built by AI companies (which may be the same companies that built the underlying AI models)

Used by “end users” to interact & solve problems

AI APIs

e.g., OpenAI API, Gemini Dev API, …

Programmatic access to AI models (often — but not always — provided by the companies that developed the models)

Used by developers to build their own AI-powered apps

Prerequisites

Interested in building AI-powered apps

Programming Experience

Python (or some other language)

RAG, CAG & Finetuning

Expanding The Knowledge of AI Models

- Understanding RAG & CAG
- Building RAG & CAG Workflows
- Finetuning

Problem

Key data may be unknown to the AI model

E.g., personal data, company-internal data, or data generated after the knowledge cutoff date

Making Sense of RAG & CAG

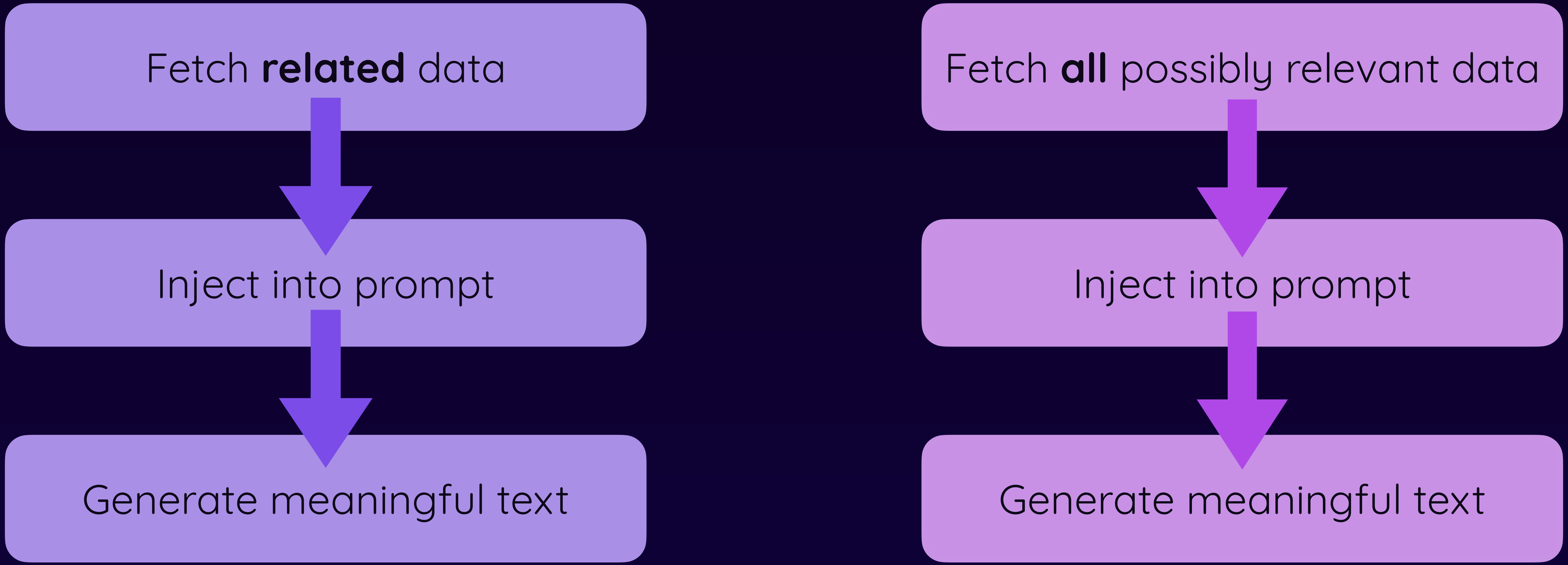
RAG

Retrieval-Augmented Generation

CAG

Cache-Augmented Generation

Both are about enhancing the AI prompt with extra information that allows the AI model to generate a better response



Fetch **related** data

Inject into prompt

Generate meaningful text

Fetch **all** possibly relevant data

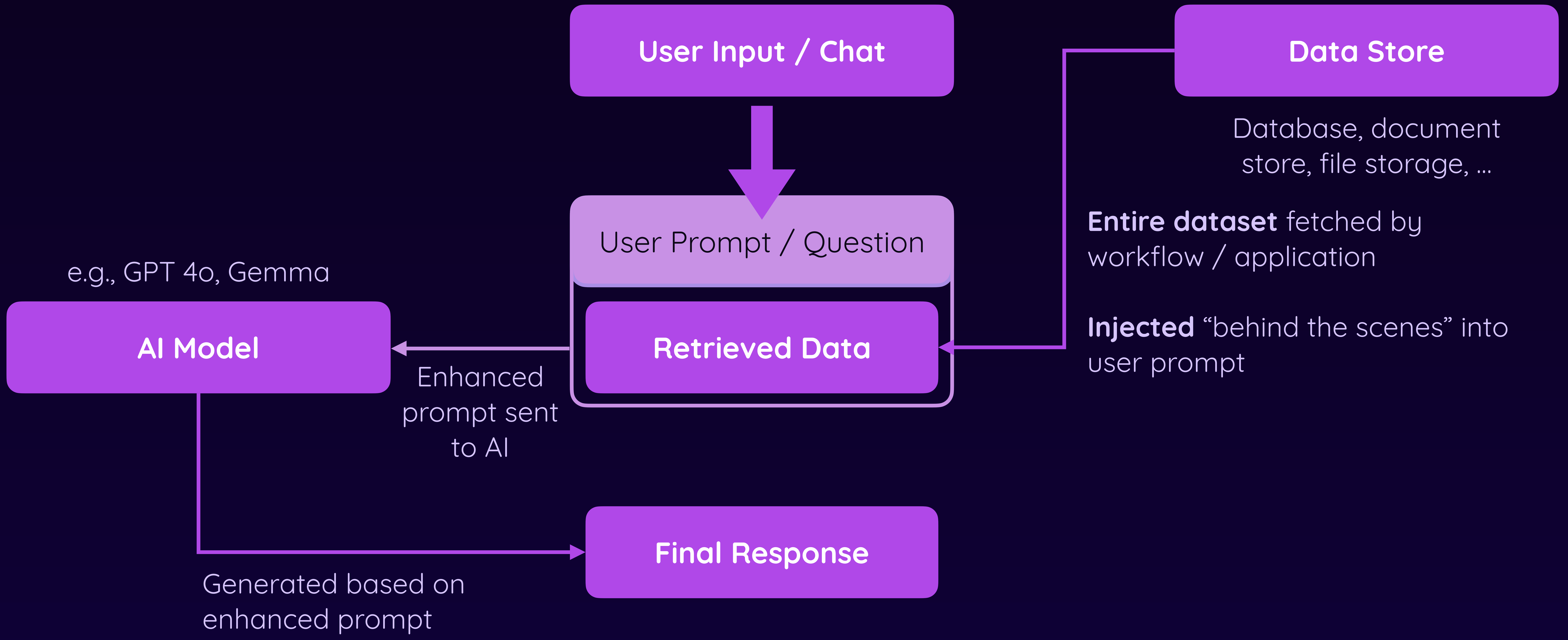
Inject into prompt

Generate meaningful text

Efficient, secure but requires more complex setup & may miss loosely related data

Less efficient, requires large context window but is less complex & can help with loosely related data

Building CAG Workflows



User Input / Chat

User Prompt / Question

Retrieved Data

Data Store

Database, document store, file storage, …

Entire dataset fetched by workflow / application

Injected “behind the scenes” into user prompt

e.g., GPT 4o, Gemma

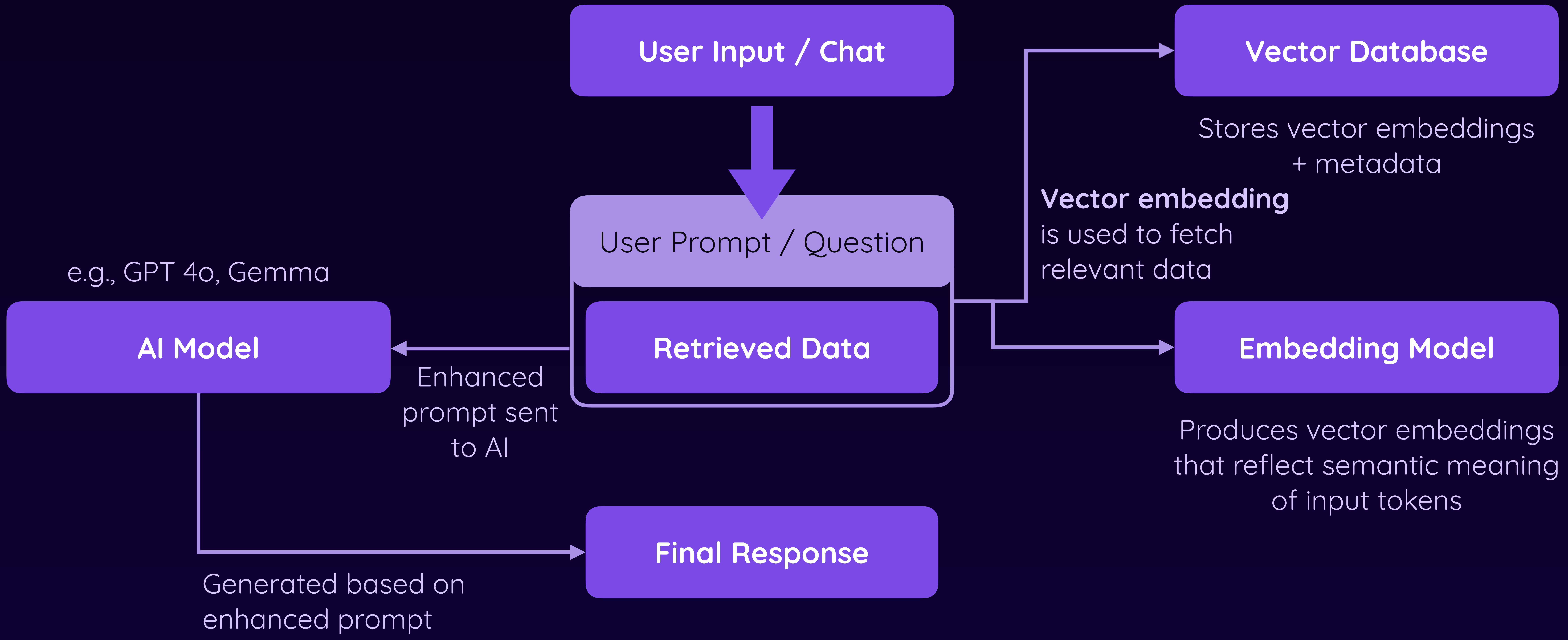
AI Model

Enhanced prompt sent to AI

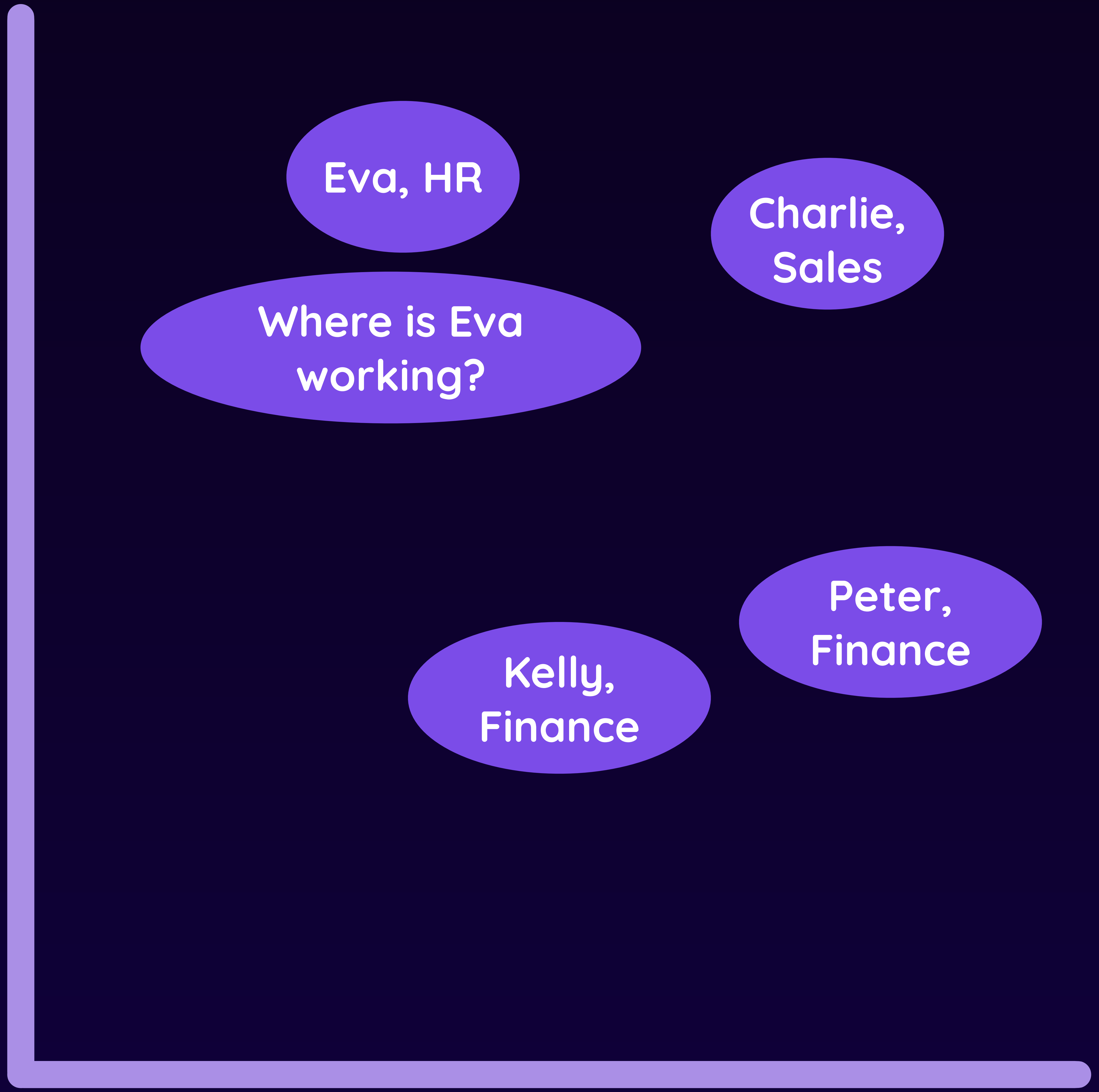
Final Response

Generated based on enhanced prompt

Building RAG Workflows


User Input / Chat
User Prompt / Question
Retrieved Data
Vector Database
Stores vector embeddings + metadata
Vector embedding is used to fetch relevant data
Embedding Model
Produces vector embeddings that reflect semantic meaning of input tokens
e.g., GPT 4o, Gemma
AI Model
Enhanced prompt sent to AI
Final Response
Generated based on enhanced prompt

Vector Embeddings Represent Relations



Related tokens / words are stored in similar places

Important: In reality, it’s a n-dimensional space!

RAG / CAG & Few-Shot Prompting

RAG / CAG

Data / examples are inserted into prompt

Without the enhanced prompt, the model won’t know about the data

Few-Shot Prompting

Examples are inserted into prompt

Without the enhanced prompt, the model won’t know about the examples

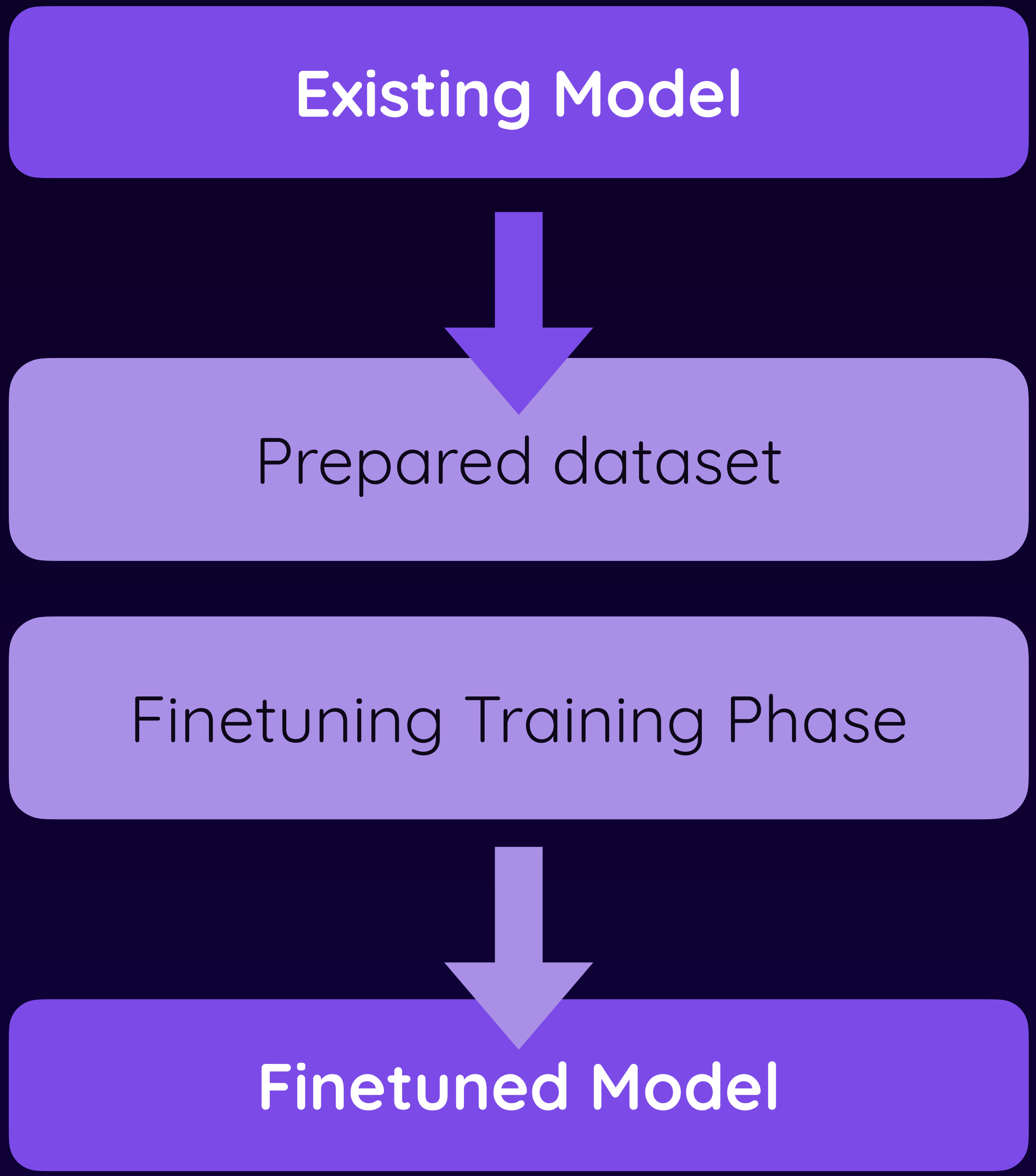
RAG / CAG is a form of few-shot prompting

Typically an automated process to enrich prompts with data

Typically a manual process to enrich prompts with examples

Finetuning Models

Unlike few-shot prompting or RAG / CAG, Finetuning is about **permanently adjusting** the weights (and therefore **“knowledge” and behavior**) of the underlying model



Existing Model

Doesn’t know anything about your data

Prepared dataset

E.g., selected list of posts or company-internal documents

Finetuning Training Phase

Can take hours or even days
Potentially very high cost!

Finetuned Model

Knows the finetuning data and related behaviors (e.g., writing style)

Finetuning vs RAG / CAG

RAG / CAG / Few-shot	Finetuning
Data is inserted into prompt	Existing model is re-trained on custom data
Without the enhanced prompt, the model won't know about the data	Model "learns" about data & specific behaviors / text style
Low / mediocre complexity	Mediocre / high complexity
Cost may increase for long prompts / lots of data	Mediocre / high training cost BUT potentially lower prompting costs
Prefer in most use-cases, especially when context window size & data fetching is not an issue	Prefer if you need a specialized model or when data fetching is not an option

Automation & Agents

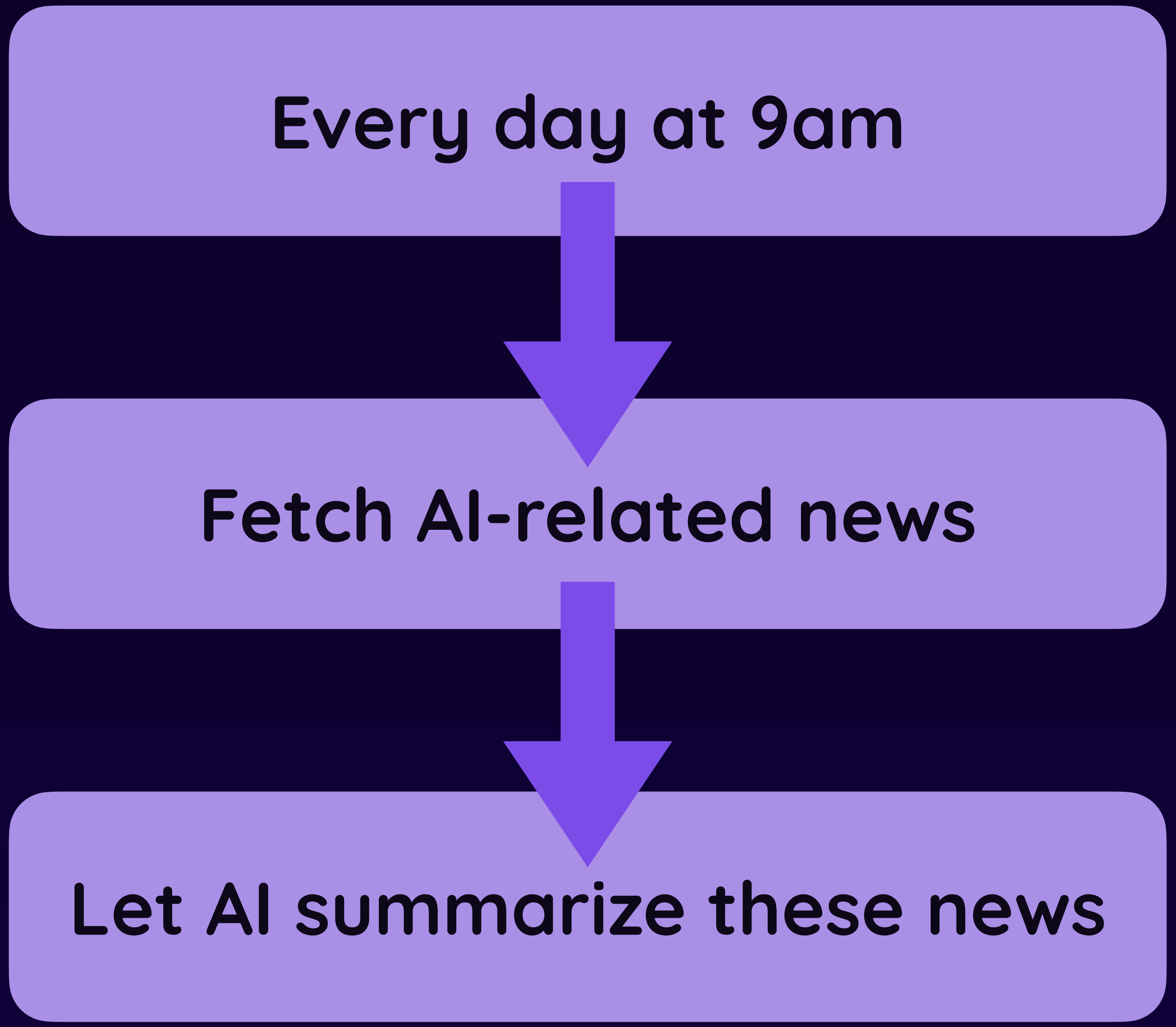
Building Automated AI-powered Workflows

- AI Automation vs AI Agents
- Building Agents Without Writing Code
- Building Agents With Code

Automation vs Agents

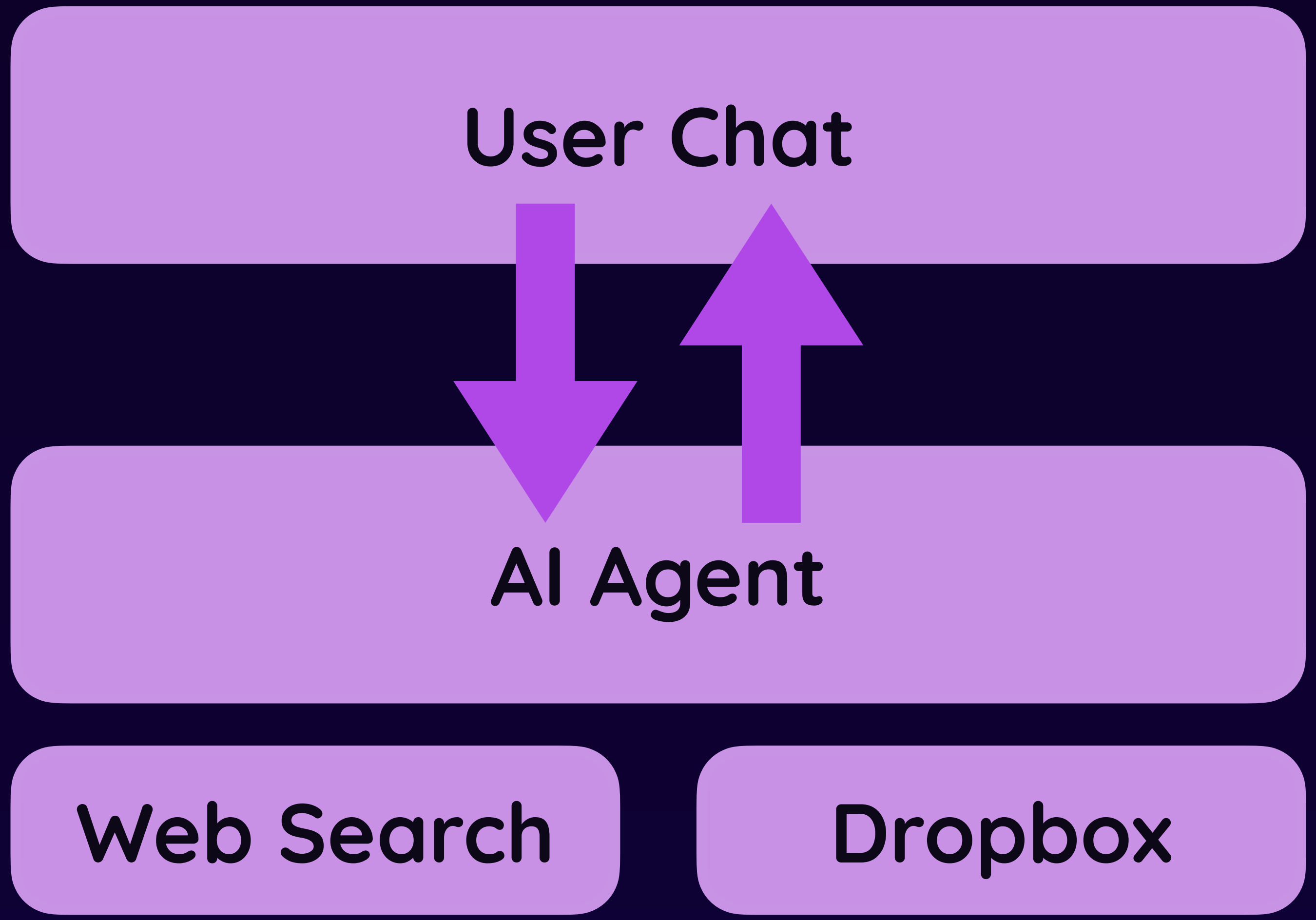
AI Automation

A pre-defined workflow that uses AI in one or more steps



AI Agents

An AI model that can solve a variety of tasks by using assigned tools



Building AI Workflows (Automation & Agents)

You got many options & platforms

No Code

n8n

Gumloop

Flowise

Trilex AI

And many, many others...

With Code

Custom code

LangChain & LangGraph

Pydantic AI

CrewAI

Agno

Vercel AI SDK

And many, many others...

Enhancing The “Meetups” REST API

Add Authentication

Add POST /signup & POST / login routes

Implement JWT-based authentication

Protect all routes except for GET /meetups

Handle Errors

Throw errors & use generic error handling middleware

Use appropriate error status codes (e.g., 401 if not authenticated)

Developer AI Tools

Beyond ChatGPT

- GitHub Copilot & GitHub Copilot Chat
- Cursor IDE

You’re Sharing Data!

Use these AI tools with care

Carefully evaluate the vendors’ privacy statements & consider if you’re allowed (and want to) share code & prompts!

AI Tools Covered

GitHub Copilot	GitHub Copilot Chat	Cursor IDE
Smart code completions	AI chat interface integrated in IDE	IDE based on VS Code with integrated AI
Enhance the default IDE auto-completions	Uses project code as context	Prompt-focused, AI-driven programming
Can be triggered in different ways & requires no prompt writing	Use for explanations, fixes, test generation & more	Generate, explain & enhance code, fix errors, search docs & more
Paid Plans **General Availability**	**Copilot Extension** **Limited Availability**	**Free & Paid Plans** **General Availability**

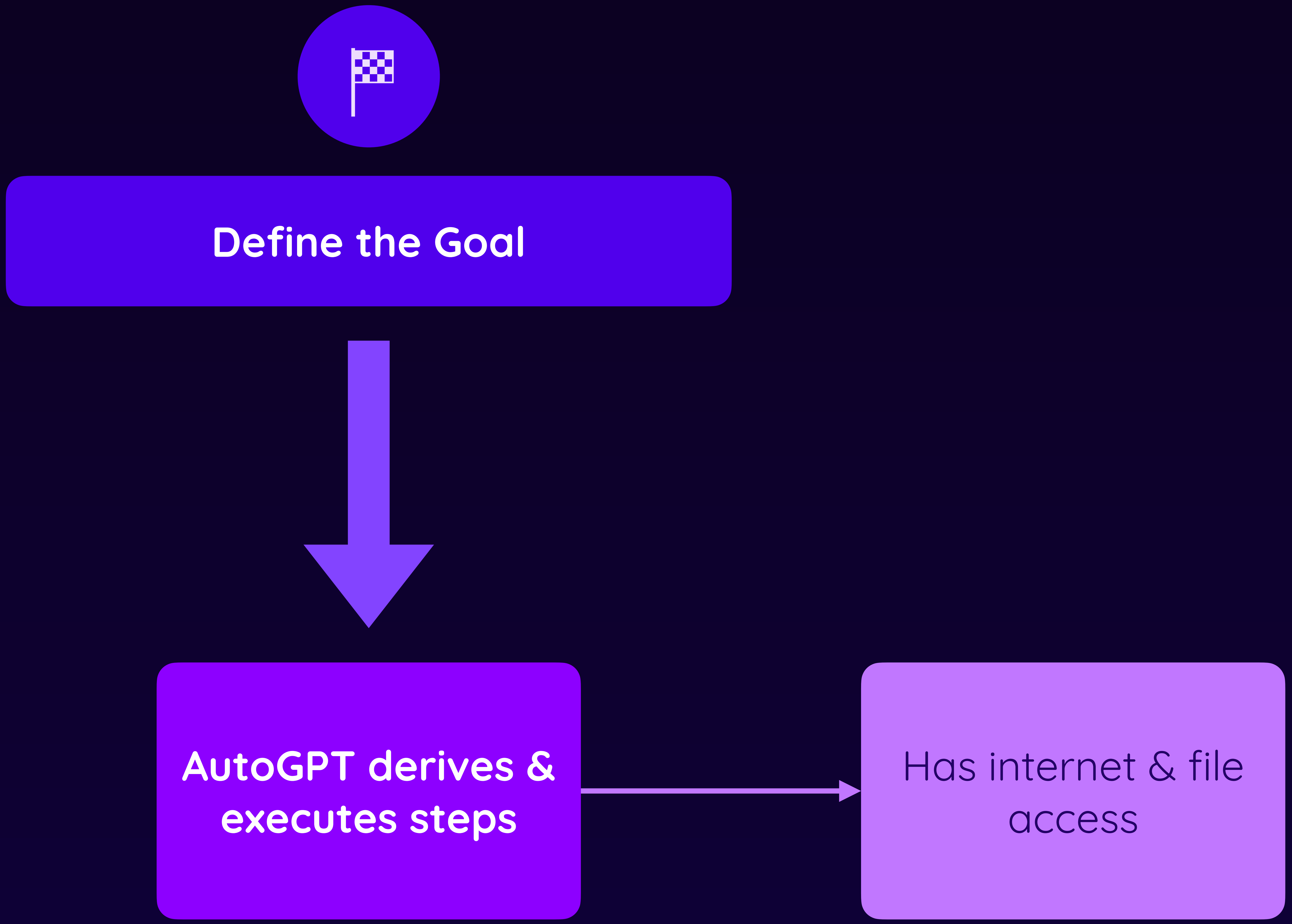
ACADE
MIND

Building Automated AI Workflows

Just Define The Goal, Not The Steps

- Get Started with AutoGPT
- Explore LangChain For Custom AI Workflows

What Is AutoGPT?

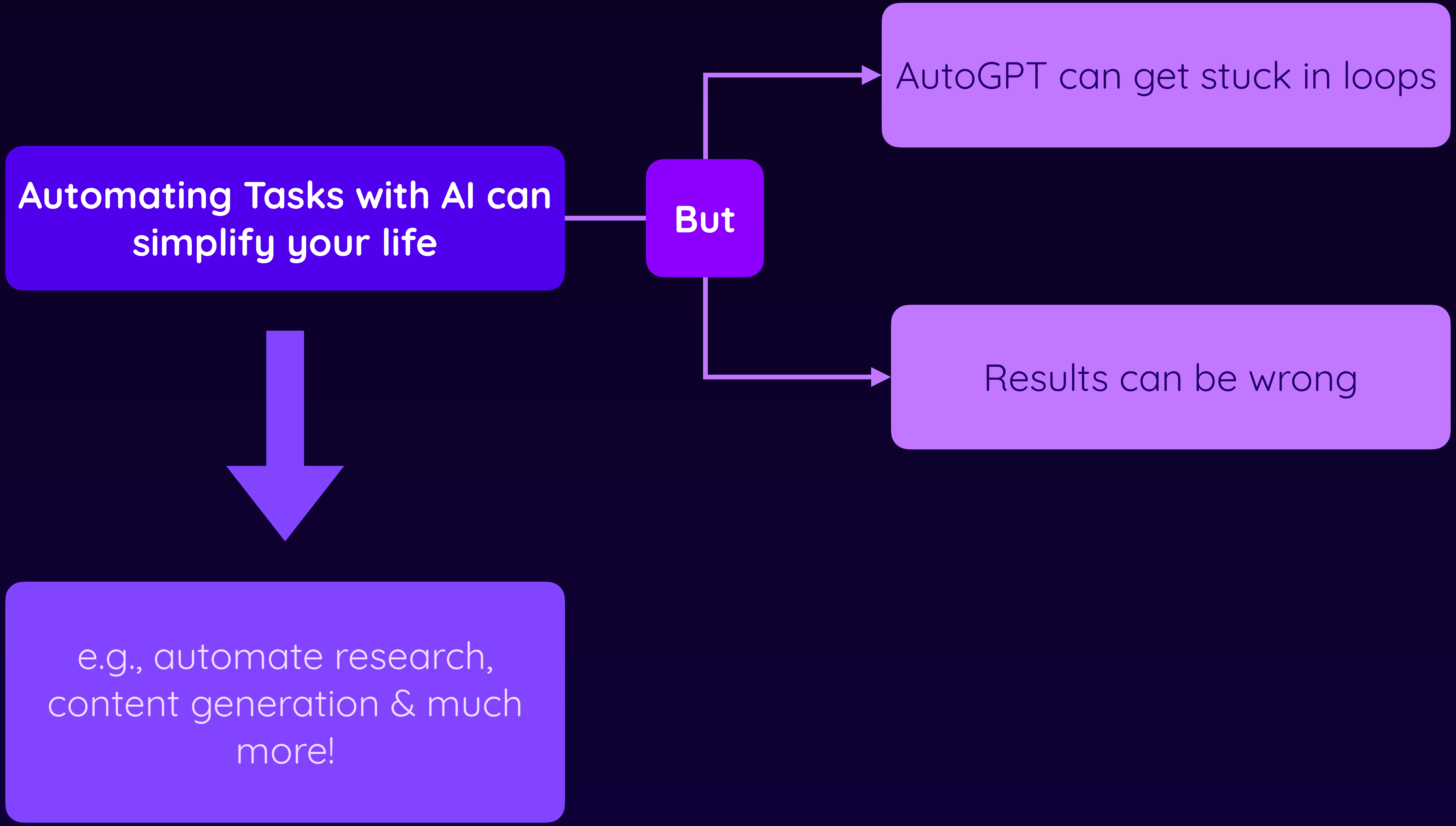


Define the Goal

AutoGPT derives & executes steps

Has internet & file access

AutoGPT Can Be Awesome



Automating Tasks with AI can simplify your life

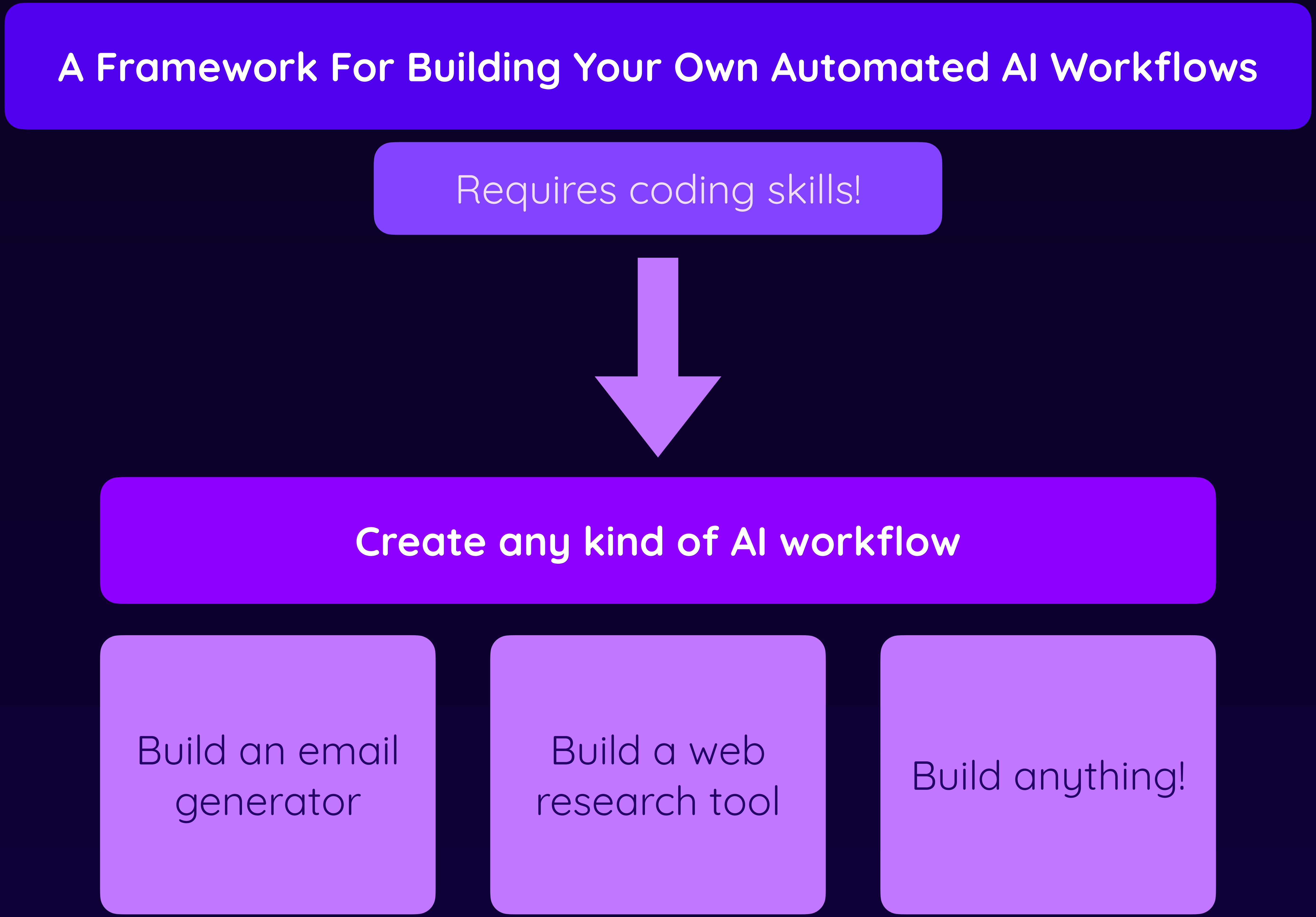
e.g., automate research, content generation & much more!

But

AutoGPT can get stuck in loops

Results can be wrong

What Is LangChain?



A Framework For Building Your Own Automated AI Workflows

Requires coding skills!

Create any kind of AI workflow

Build an email generator

Build a web research tool

Build anything!