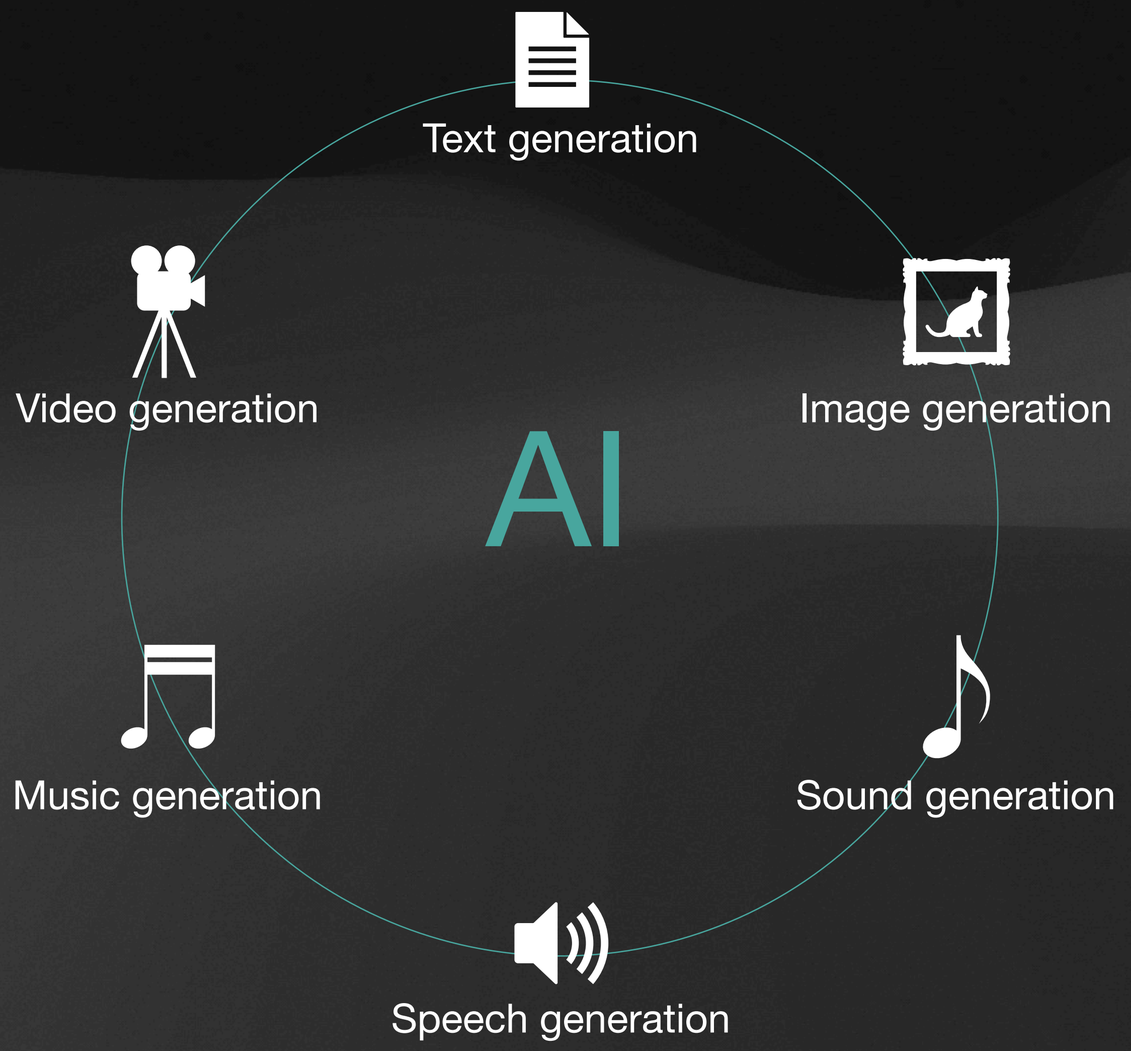
Navigating the New Frontier

Lessons from Building and Deploying Agentic AI Apps

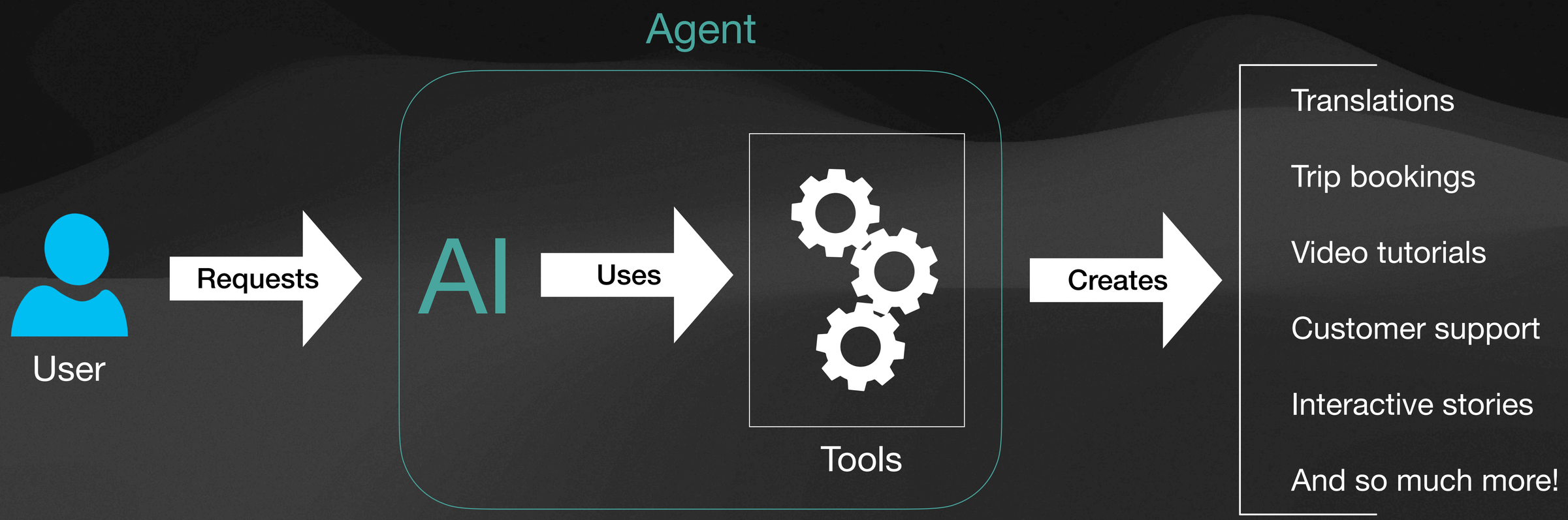
April 3rd, 2025

Juan Peredo

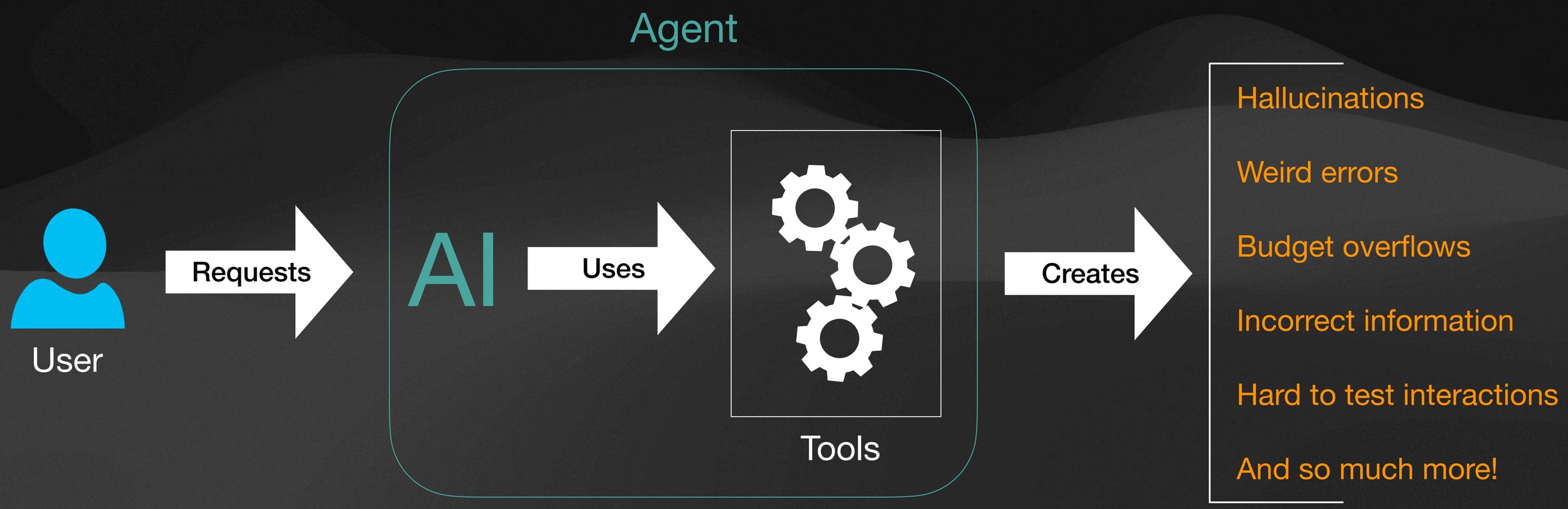
Welcome to the renaissance of computing



Interacting with the world via agents



Well… At least trying to interact with the world …



AI introduces a lot of new variables into a product creation journey

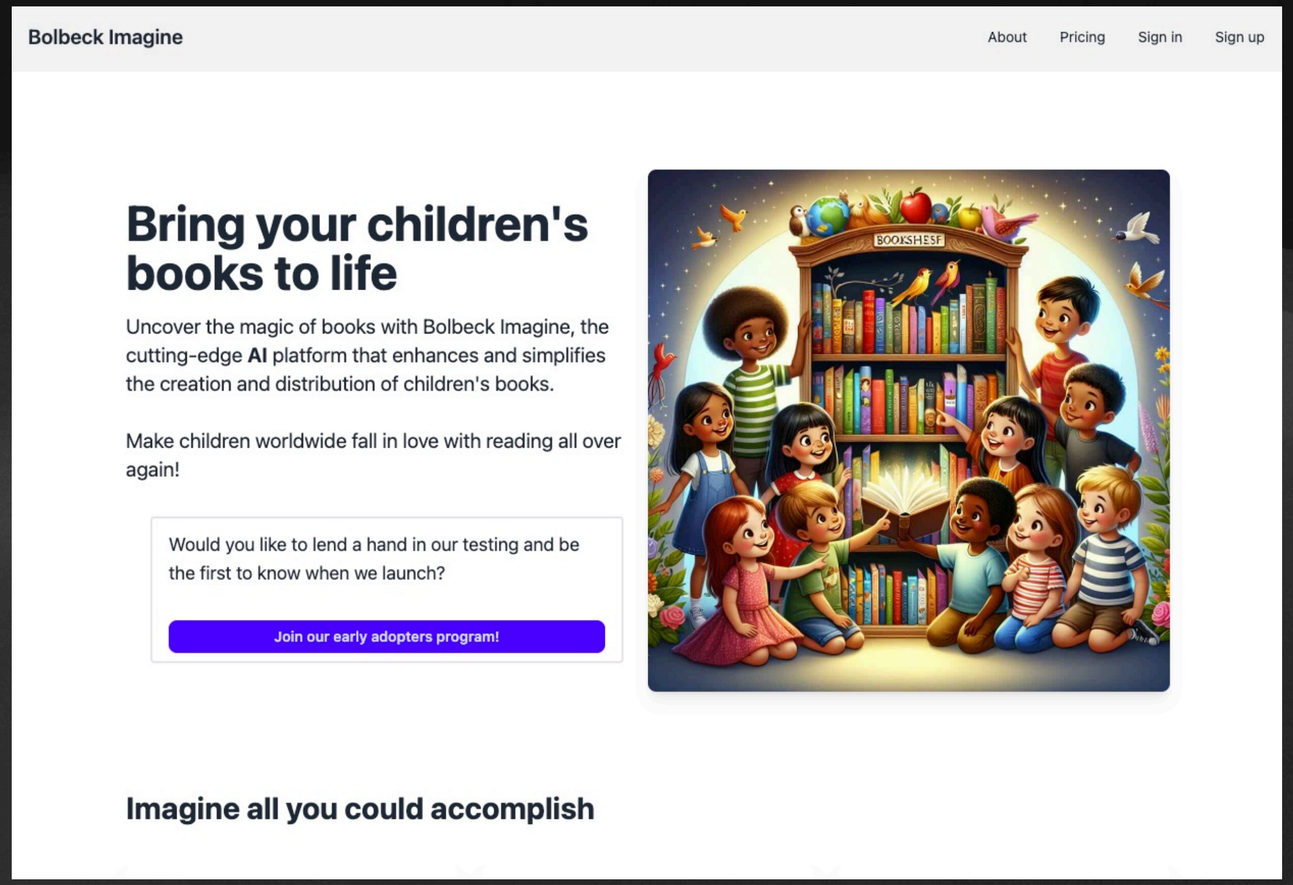
Juan Peredo - Your guide in this journey

- Founder / architect / consultant / developer & everything in between
- Linkedin: linkedin.com/in/juanperedotech
- Over 15 years of IT experience in companies like:
  - Bolbeck LLC
  - AWS
  - Strategy& (PWC)
  - Booz & Co

A look into the trenches of App dev with agents

On the menu for today:

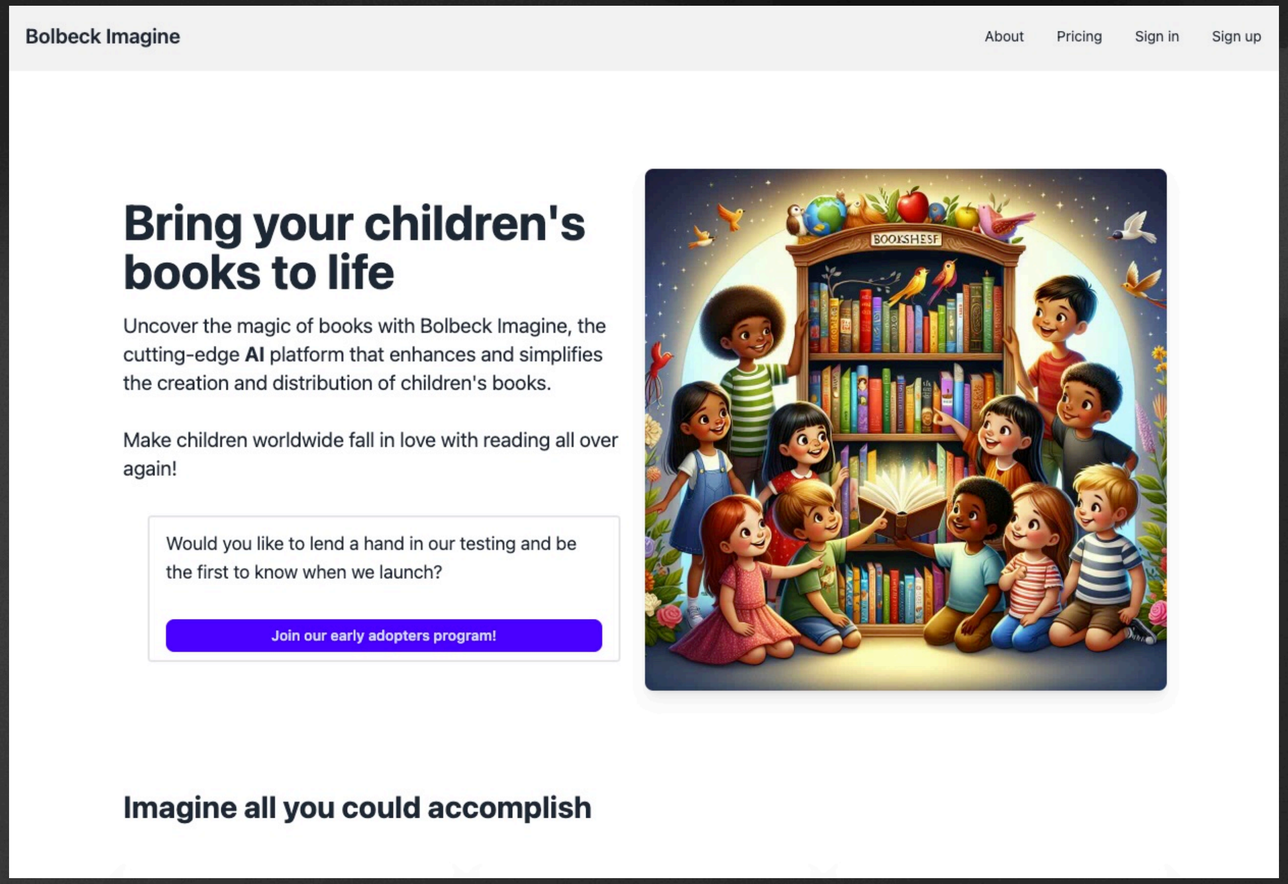
- ✔ The good
- ✘ The bad
- ϟ The “what the heck is it doing” moments
- 👽 The “wow, that is cool” surprises



This is my journey. Your mileage may vary.

Let’s take a quick look at the app…

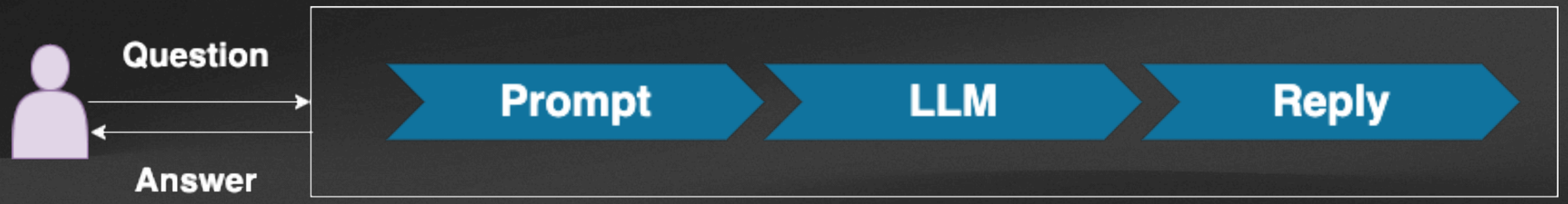
It’s demo time!



Going back to basics

How it all got started…

Going back to basics



Building an AI chatbot is easy

However, validating and moderating the chatbot content is hard

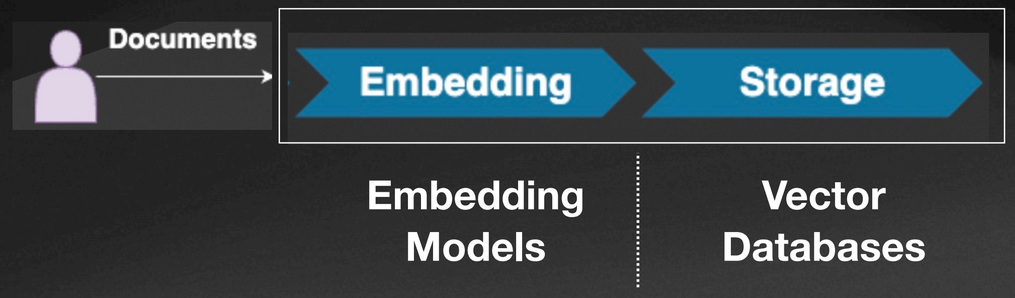
There are techniques to validate LLM output

Technique	Description	Technique Issues
Prompt Engineering	Provide directions to LLM in the prompt	• Need to be very precise • LLM may choose to ignore directions
Guardrails	Specialized LLMs that classifies answer as ‘bad’	• Adds additional latency to every LLM call • Adds cost and Is not always correct
RAG	Provide additional context to LLM as part of the prompt	• Need to add RAG store to solution • Highly dependent on quality of data provided • RAG may not provide enough info to get answers
Fine tuning	Provide additional training to an LLM	• Very expensive when compared to other methods • Time consuming, could degrade LLM

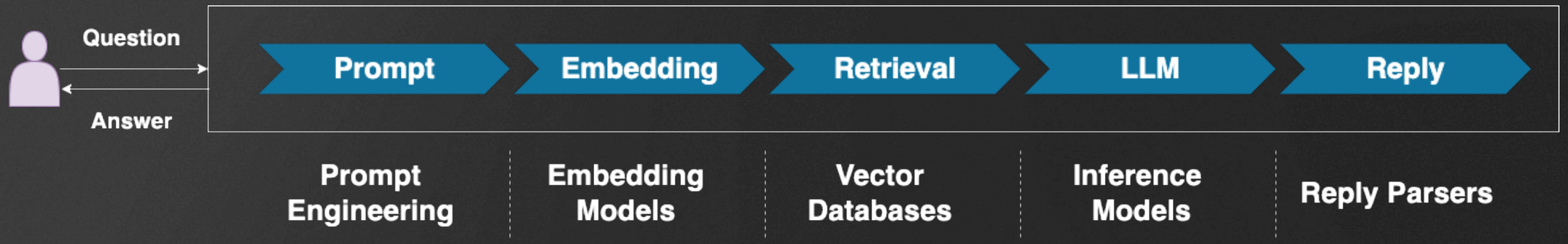
There are no techniques that guarantee chatbot answer is ‘good’

✓ RAG makes AI be more personal

Step 1: Data load



Step 2: Answer retrieval



RAG is fallible and thus the arrival of its friends

Simple RAG

Context Stuffing

Cache Aug. Gen.

Goal:
Get more precise and useful answers

Graph RAG

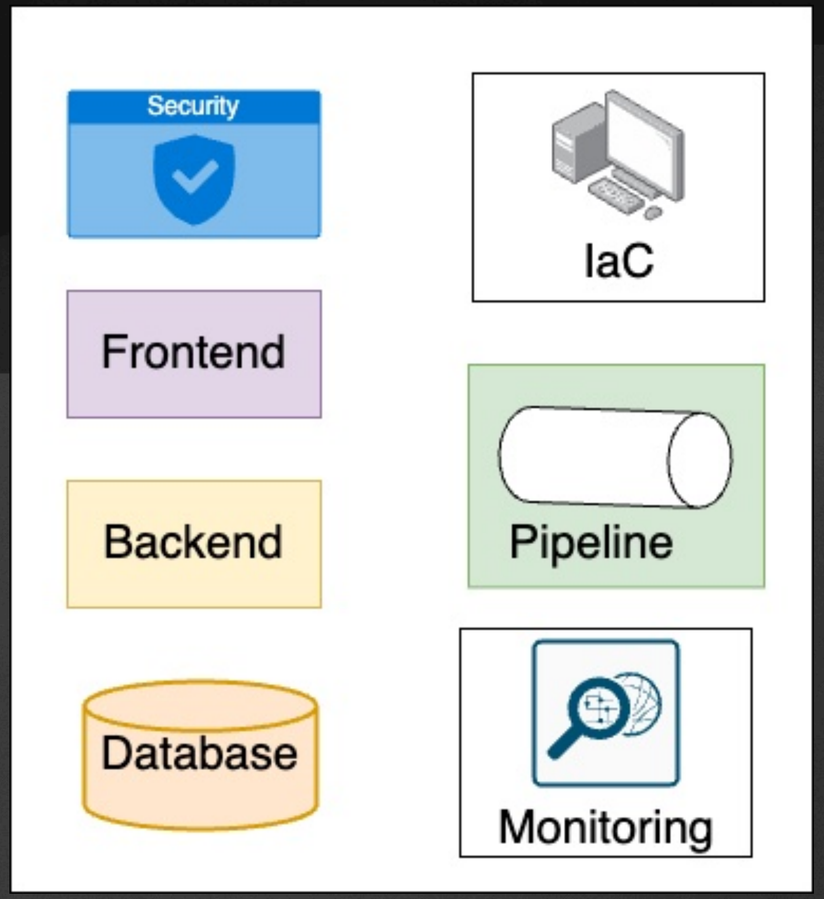
Multi-modal RAG

Hybrid RAG

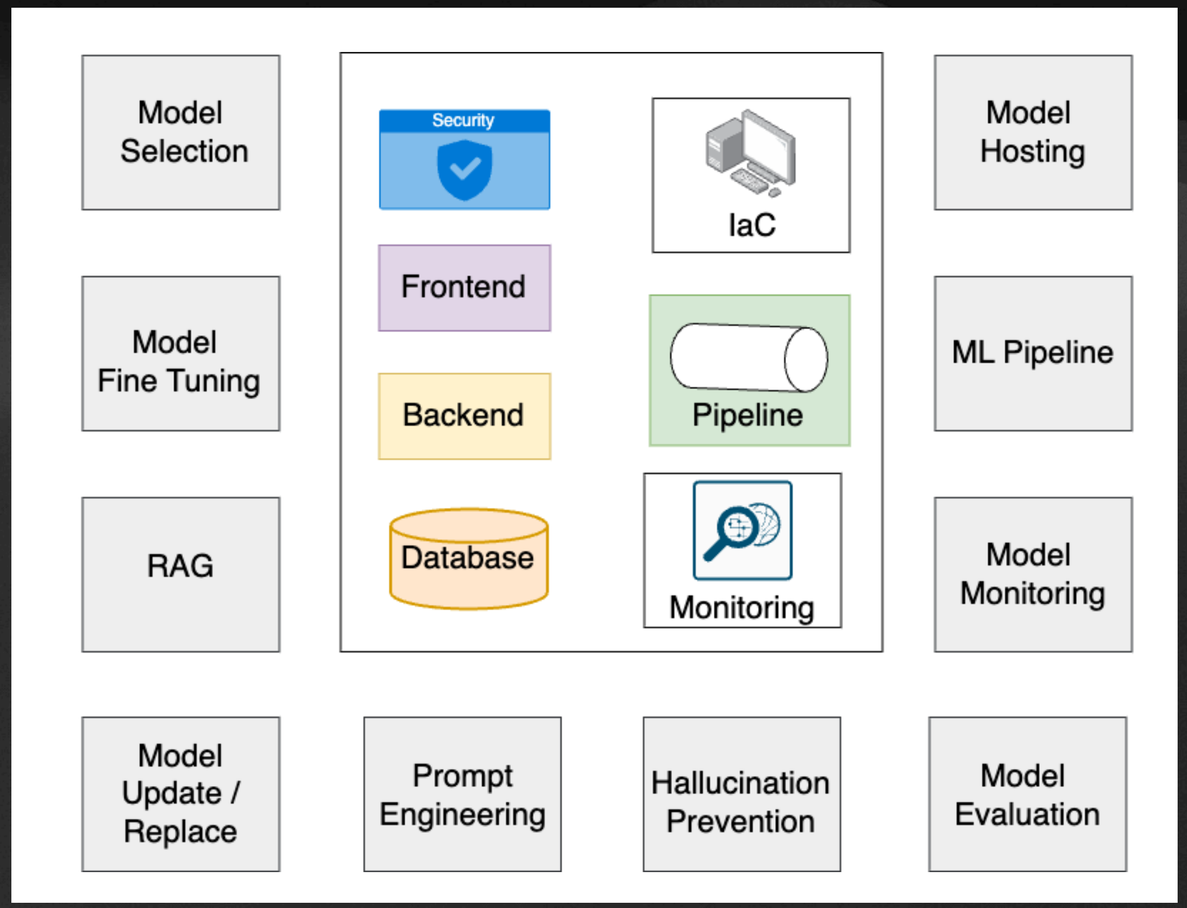
And there are so many more RAG variants!

✗ AI models increase your app complexity

Typical app dev components



Typical AI app dev components



Choose your models wisely

Text Generation

- Hundreds of models available
- Performance varies widely
- Switching LLMs requires re-evaluation of prompts

Image generation

- Quality of images dependent on prompt & model
- Text to image, image to image and others
- More expensive that LLMs

Video generation

- Generation can take minutes
- Text to video, image to video and others
- More expensive than image generation

Sound generation

- Not as common or widely used
- Less models to choose from

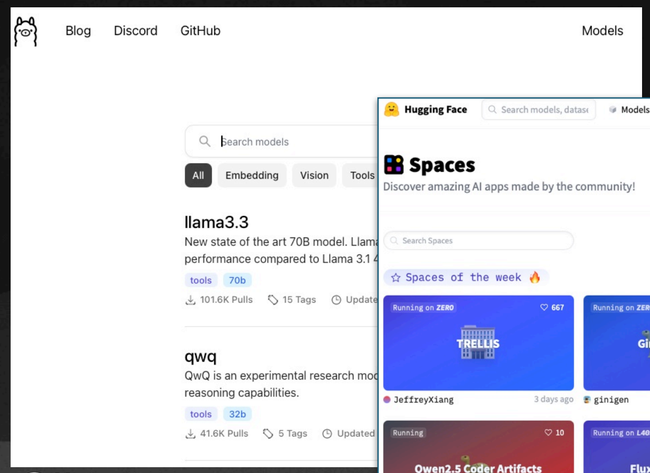
Music generation

- Quality depends on prompt and/or original audio
- Text to Audio, audio to audio

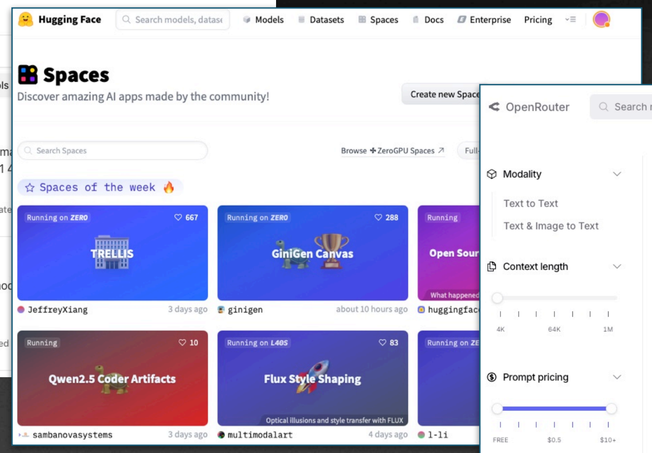
Speech generation

- Must cleanup input text
- Performance varies widely. Newer models are great
- Trade off between speed and intonation/quality

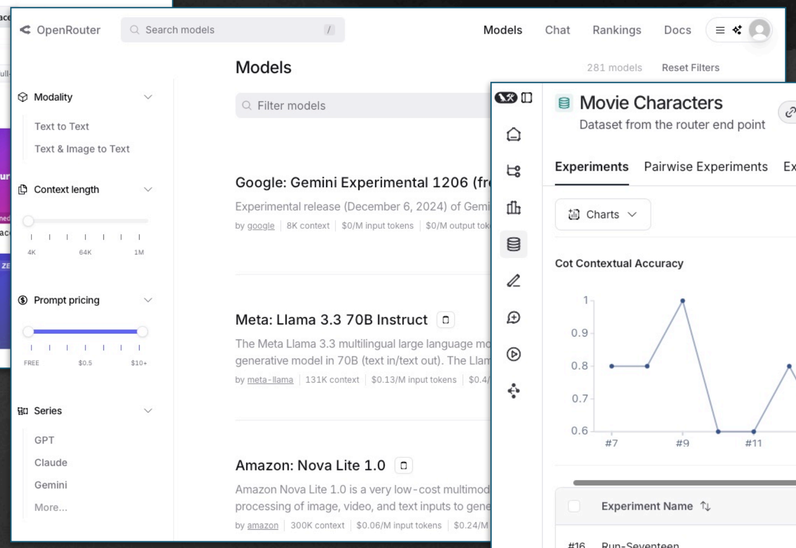
Evaluate models & choose the best for the use case



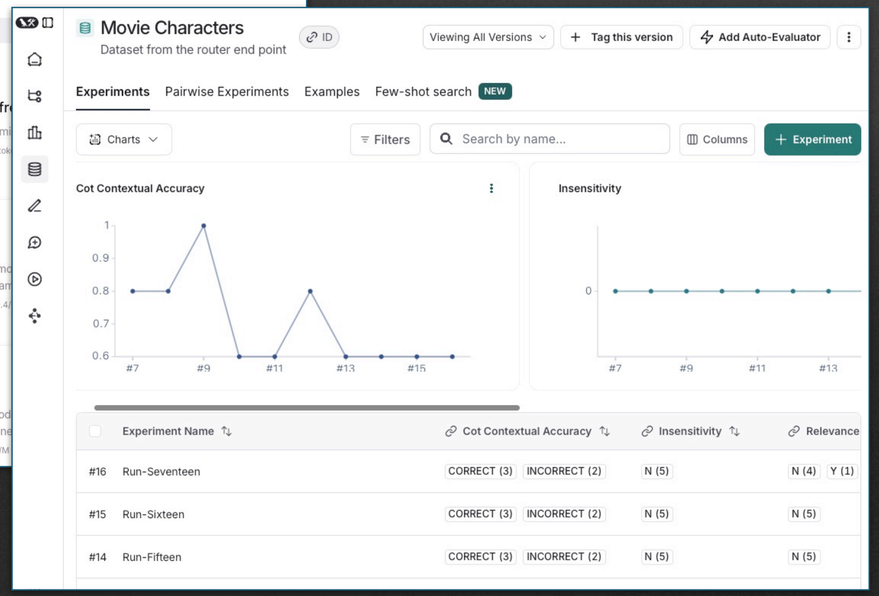
Ollama



Hugging Face



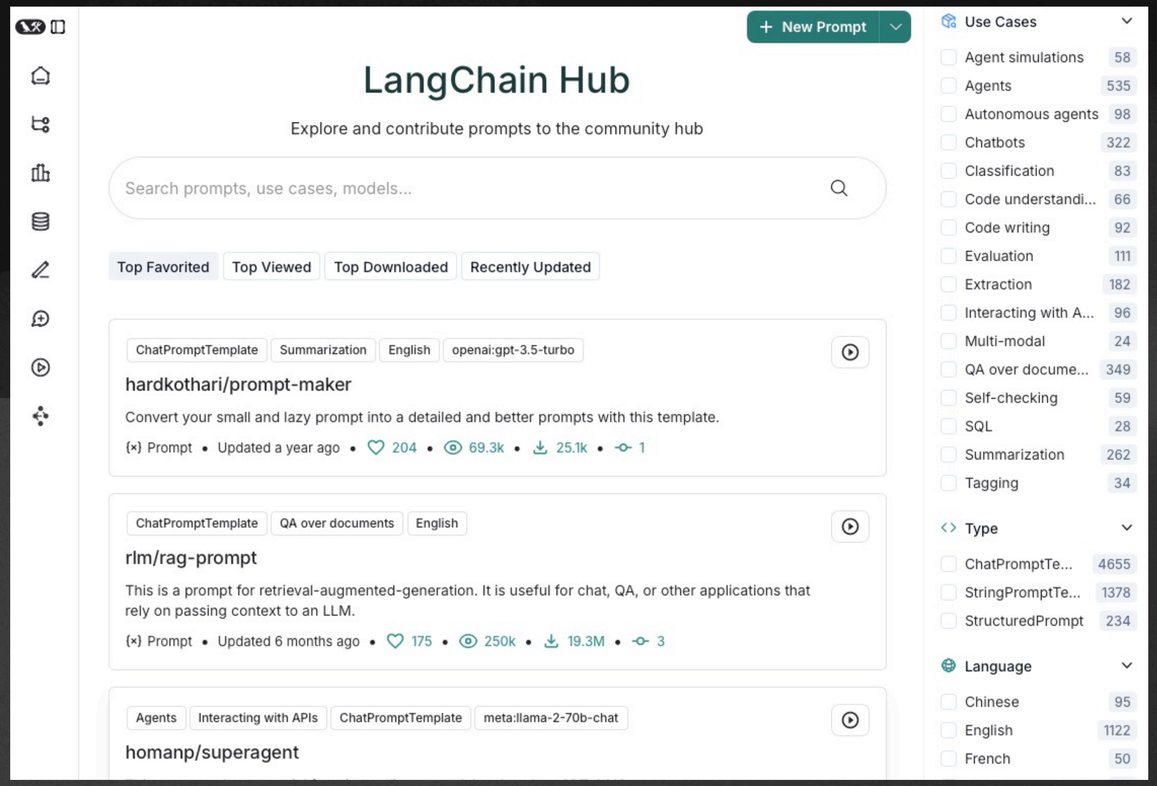
OpenRouter



LangSmith

Evaluation must take place at each step of the way

Externalize your prompts

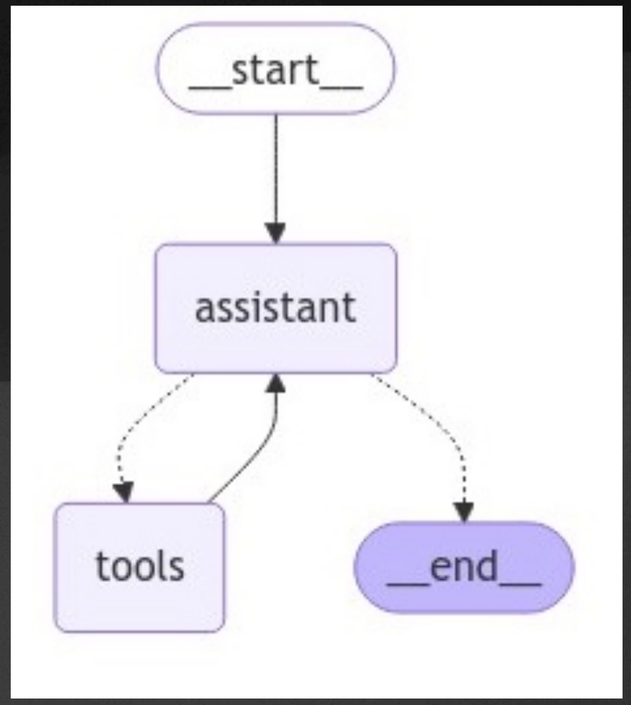


Externalizing prompts facilitates:

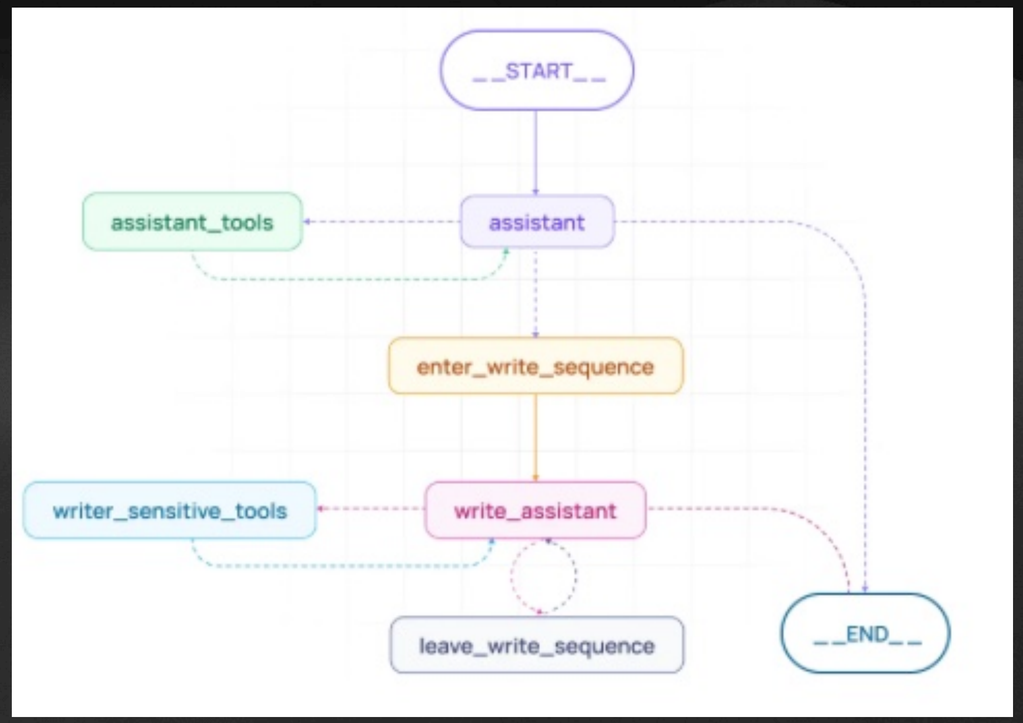
- Expert input
- Future proofing
- Faster development

Prompts should not be embedded in the codebase

Use agents to go beyond the chatbot



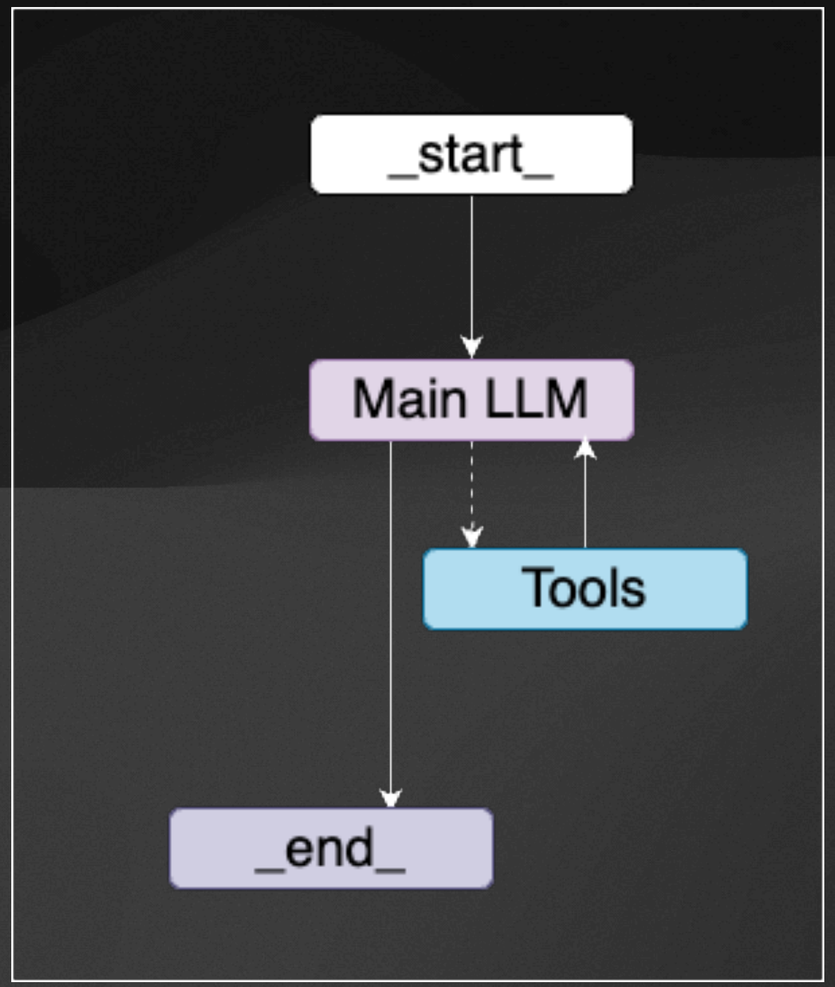
Simple tool calling agent



Simple assistant agent

Agents allow LLMs to interact with the real world

Interacting with the world via a reAct agent



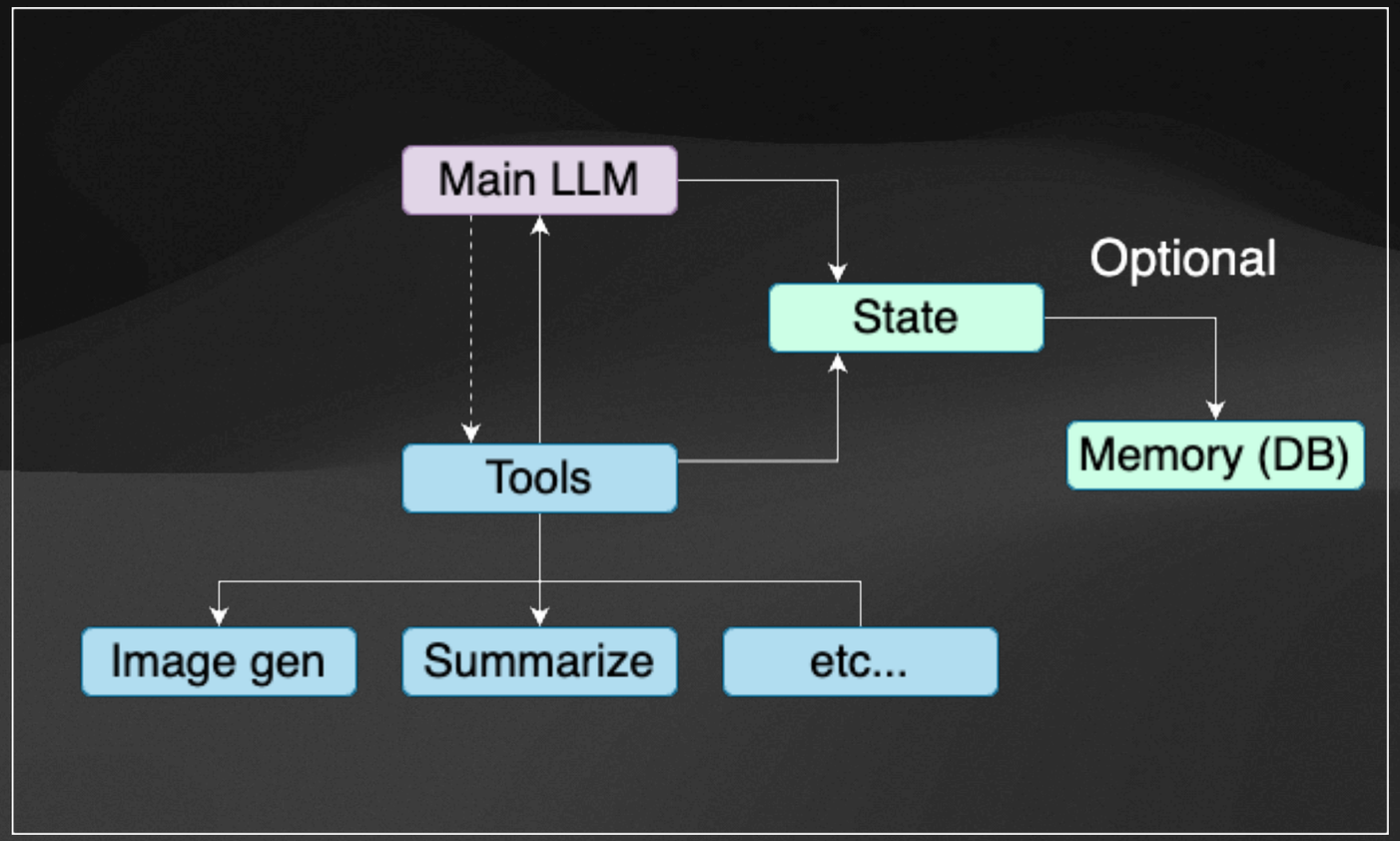
User calls agent

Agent calls main LLM

LLM use tools to perform task(s)

Agent returns result

Tools: fancy name for functions the agents can use



Tool can be as simple as a function that adds 2 numbers

Tools are called by LLM

Tools share info via a state

Memory can store helpful information

- User Settings
- Personalization
- Preferences
- Conversation facts
- Etc…

Use memory to help smooth out user interactions

Beware of storing user's personal info

- May subject you to additional regulations
- Could make you a target of a malitious actor

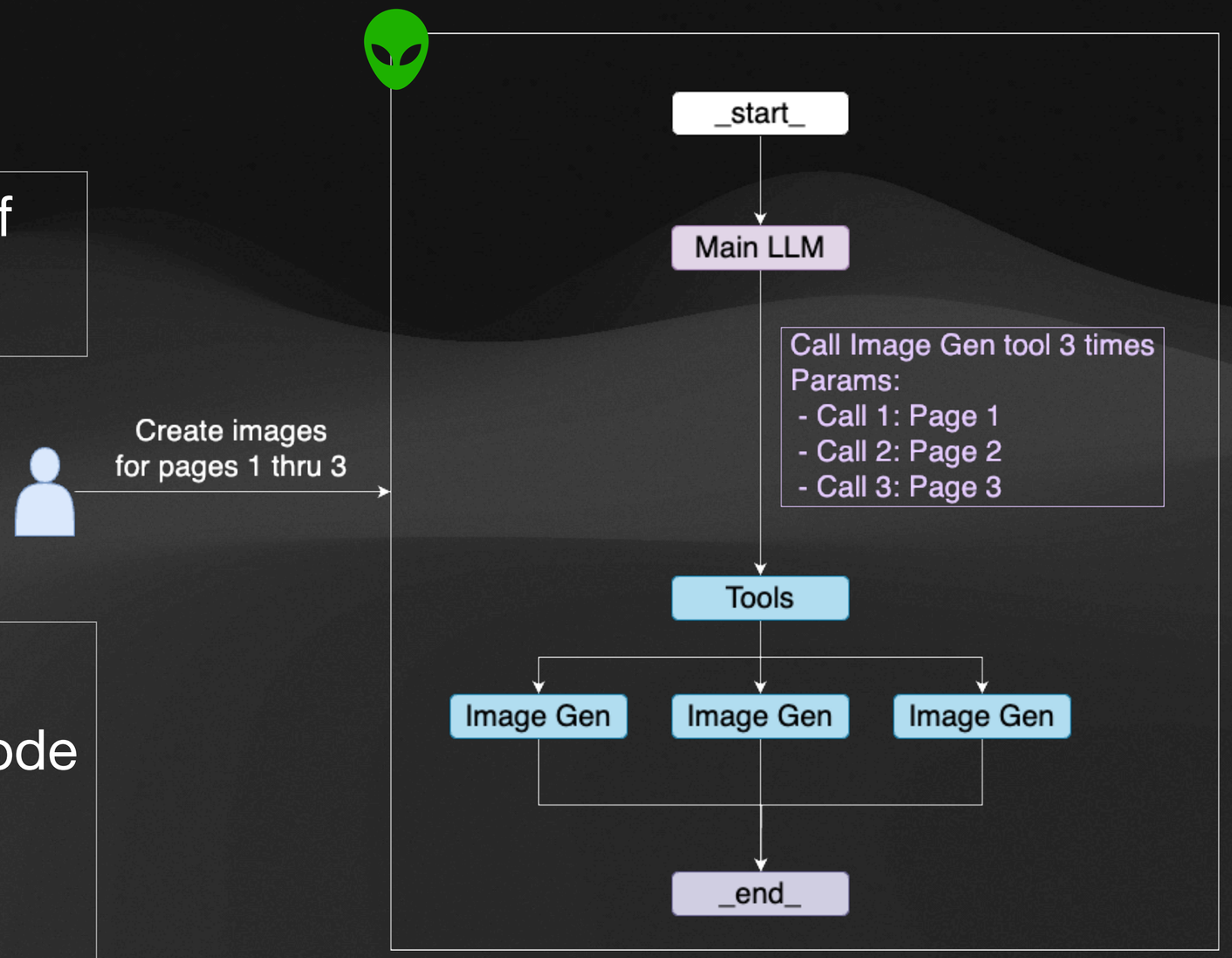
Store what you need and nothing else

Take advantage of platforms to speed up agents

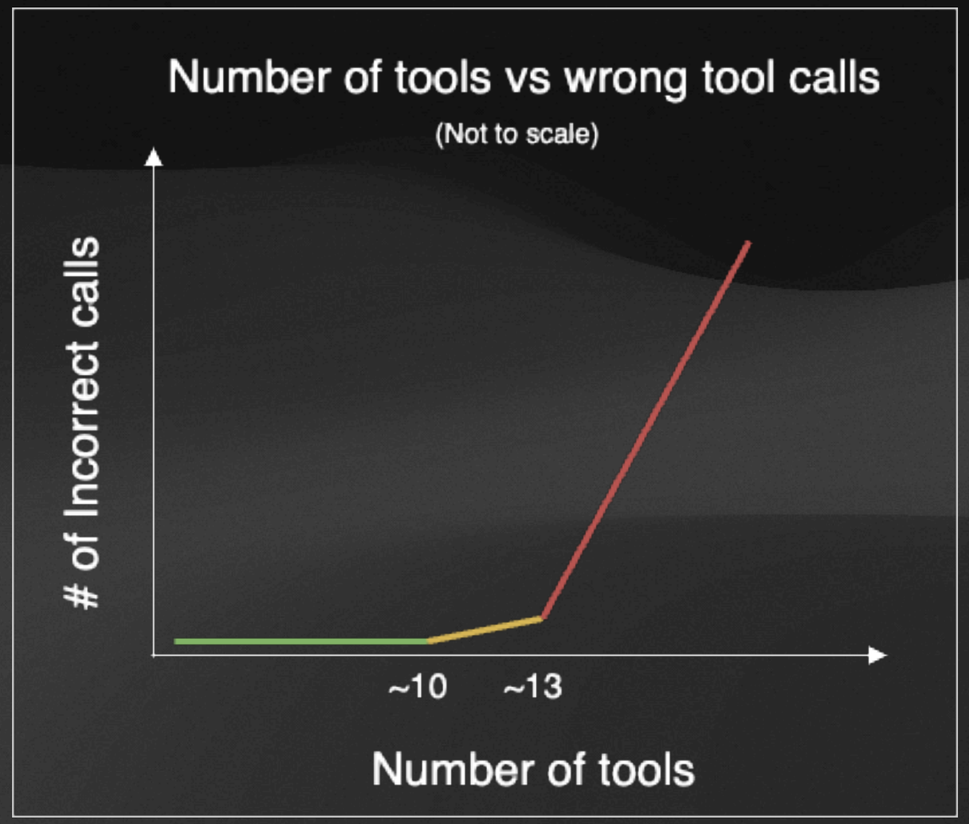
Models are the biggest source of latency

Platforms like LangGraph allow complex chaining of models & code

- Concurrent calls
- Branching



Crucial: # of functions and their descriptions



LLMs decide which tools to call based on their names, parameters & descriptions

They are not always right !

Be precise on your tools metadata for better accuracy

✗ Bad

```
def myFunction(a, b)
```

```
def generate(a: str, b: str)
```

✓ Better

```
def generate_image_for_location
(location_name: str, image_quality:
ImageQuality)

"""
Generate an image for a location

Args:
    location_name: name of the location
   image_quality: quality of the image to be
generated
"""
```

Precise metadata improves chances of LLM doing the right thing

LLMs can follow directions on the metadata

```
class ImageQuality(Enum):
    """Quality of the image to be generated. Use low as default"""
    LOW = "low"
    MEDIUM = "medium"
    HIGH = "high"
    EXTREME = "extreme"
```

Note the default quality to be used by LLM

When something goes wrong, LLMs go off the rails

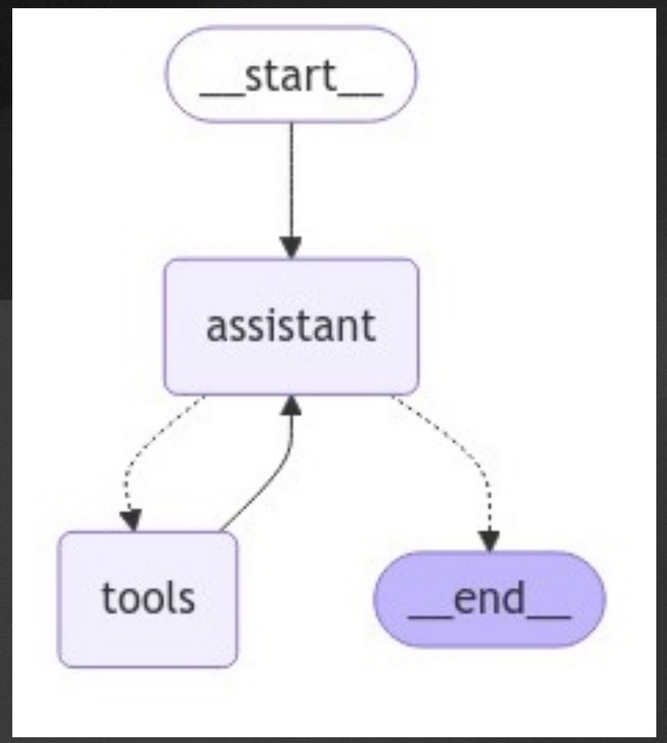
If a tool fails, LLM will try to call all other tools to fix the issue using brute force

Always check how many steps it takes to solve users' request

Set the **maximum** number of iterations per call for the agent

Be aware of the cost when running agents

Call Center example: Agent that uses OpenAI o1 as the LLM



Simple tool calling agent

Assumptions:

3,000 Calls / day	15 Tool calls / request
$15 / 1M Tokens in	$60 / 1M Tokens out

1,500 Input tokens
3,000 Output tokens

Pricing source: OpenAI API pricing webpage as of Mar 31st, 2025

Costs can add up quickly

Example call center agent using OpenAI o1 as the LLM

$ 3.24
Per call

$ 9,270
Per day

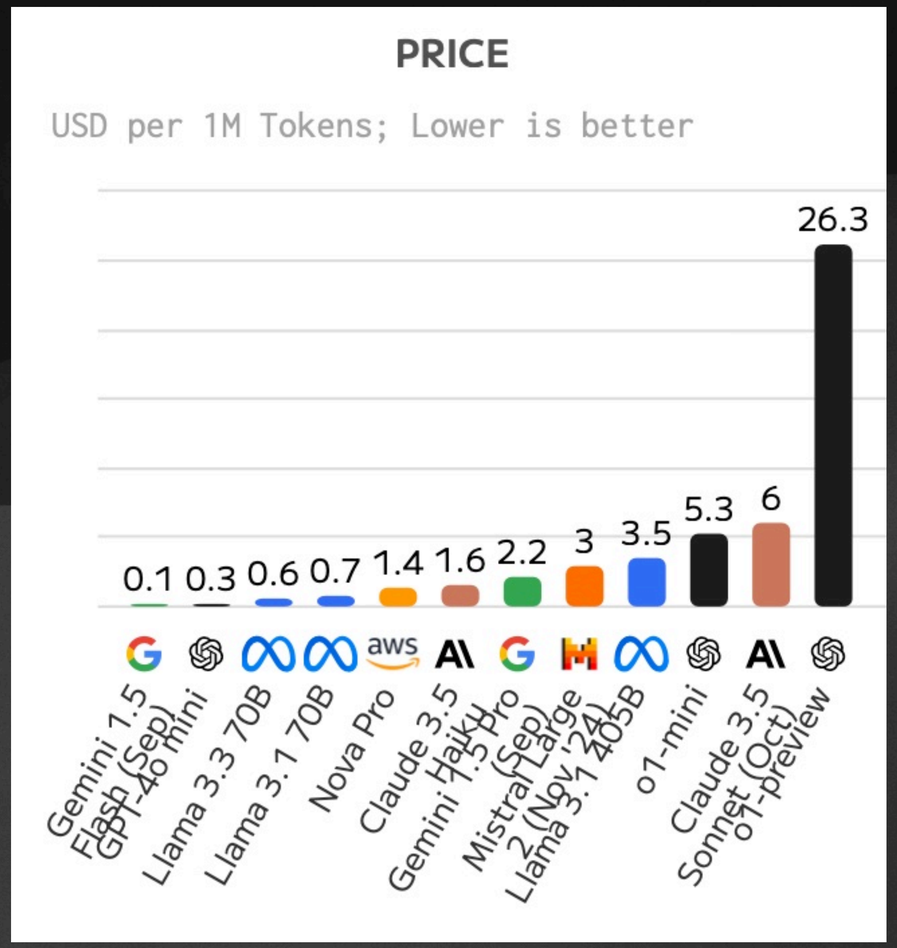
$ 291,600
per month

- 16 o1 calls (1per tool call + initial call)
- 0.000015 *1500 Input token cost
- 0.00006 * 3000 Output token cost
- (16 *(0.000015 *1500 + 0.00006 * 3000)) = **$ 3.24**

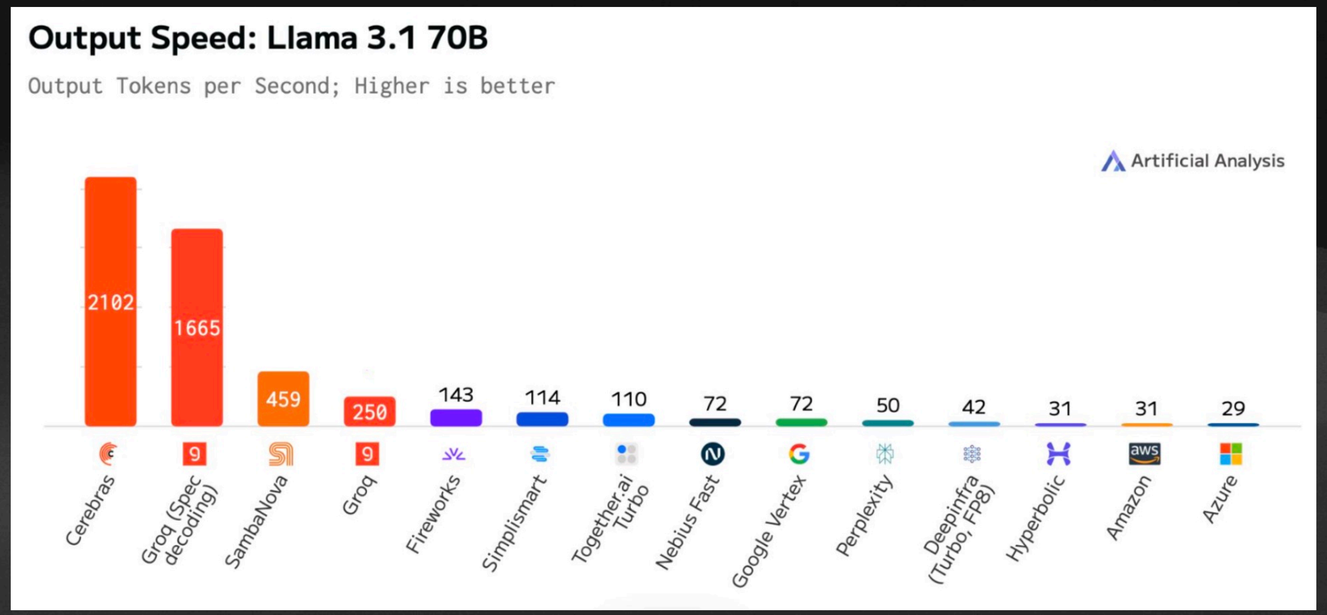
3.24 * 3000 calls/day = **$ 9,720**

9,270 $/day * 30 days = **$ 9,720**

Choice of model and provider has a big impact



Pricese are represented as a blend of token prices (3:1 ratio)



Small is beautiful and better for the environment (not to mention your wallet)

Charts source: Artificial Analysis, Feb 2025

The right tools for the job can minimize cost

Example call center agent using Llama 3.3 70B as the LLM running on Groq

$ 0.052
Per call

$ 1,56
Per day

$ 4,680
per month

- 16 o1 calls (1per tool call + initial call)
- 0.00000059 *1500 Input token cost
- 0.00000079 * 3000 Output token cost
- (16 *(0.00000059 *1500 + 0.00000079 * 3000)) = **$ 0.052**

0.052 * 3000 calls/day = **$ 156**

156 $/day * 30 days = **$ 4,680**

Pricing source: Groq API pricing webpage as of Mar 31st, 2025

The right tools for the job can minimize cost

Llama 3.1 8B, Llama 3.3 70B, OpenAI o1

$ 453
Per month

$ 4,680
Per month

$ 291,600
per month

Pricing source: Groq API and Open AI pricing webpages as of Mar 1st, 2025

Open AI o1: $15 in & 60 out per 1M , Llama 3.3 70B: $0.59 in & 0.79 out per 1M , Llama 3.1 8B: $0.05 in & 0.08 out per 1M

Remember to keep your LLM protected at all times

Having a chatbot as your input source increases the potential attack vectors to your data

Treat LLMs inputs and outputs just as potential bad actors trying to access your data

Increase Agent success rate with a classifier

Reduce the number of tools needed by your LLM at any one point

Break up the task into smaller problems by putting a classifier in front of your agent. Have the classifier direct requests to different agent nodes instead of overloading your agent with too many tools

Stop tool calling rampage by escaping the loop

Build an escape clause into the LLM tool calling loop If your app throws an error, ensure the LLM is not called again. Just report error to the user directly

Stops sticker shock and makes the app more user friendly

Make sure you have fallbacks

LLM endpoints are pretty robust, but not infallible

What do you do when a provider goes down?

Each Agent call can result in a dozen of LLM calls

Each LLM call comes with an additional cost

When using LLM endpoints, each user added to app increases the cost linearly. Economies of scale do not apply.

When renting GPUs, the cost of each additional GPU is pretty high and GPUs are hard to come by

Hard to come up with an appropriate pricing structure

Sometimes it is better to run agentic workflows

No need for an orchestrating LLM to decide which tool to call when the task path is pre-determined

Can still use LLMs to perform the work

Be nice to your wallet

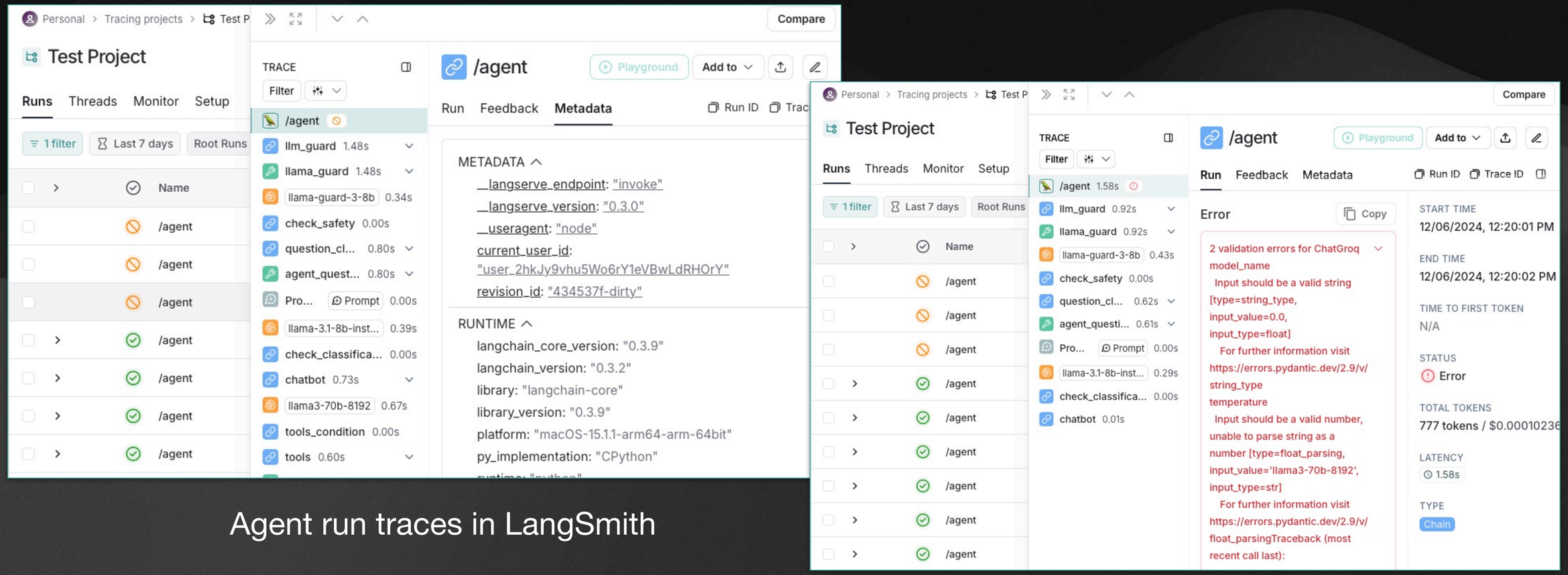
Use agents and LLMs only when you need them

AI APIs providers are convenient and efficient but cost is linear per user

GPUs are really expensive and hard to optimize for multi tenancy

Always keep an eye on your wallet and budget

Get proper observability for your Agents



Agent run traces in LangSmith

Tracing model output, specially with agents, is hard without the necessary tooling

Questions ?

Juan Peredo

linkedin.com/in/juanperedotech