Google

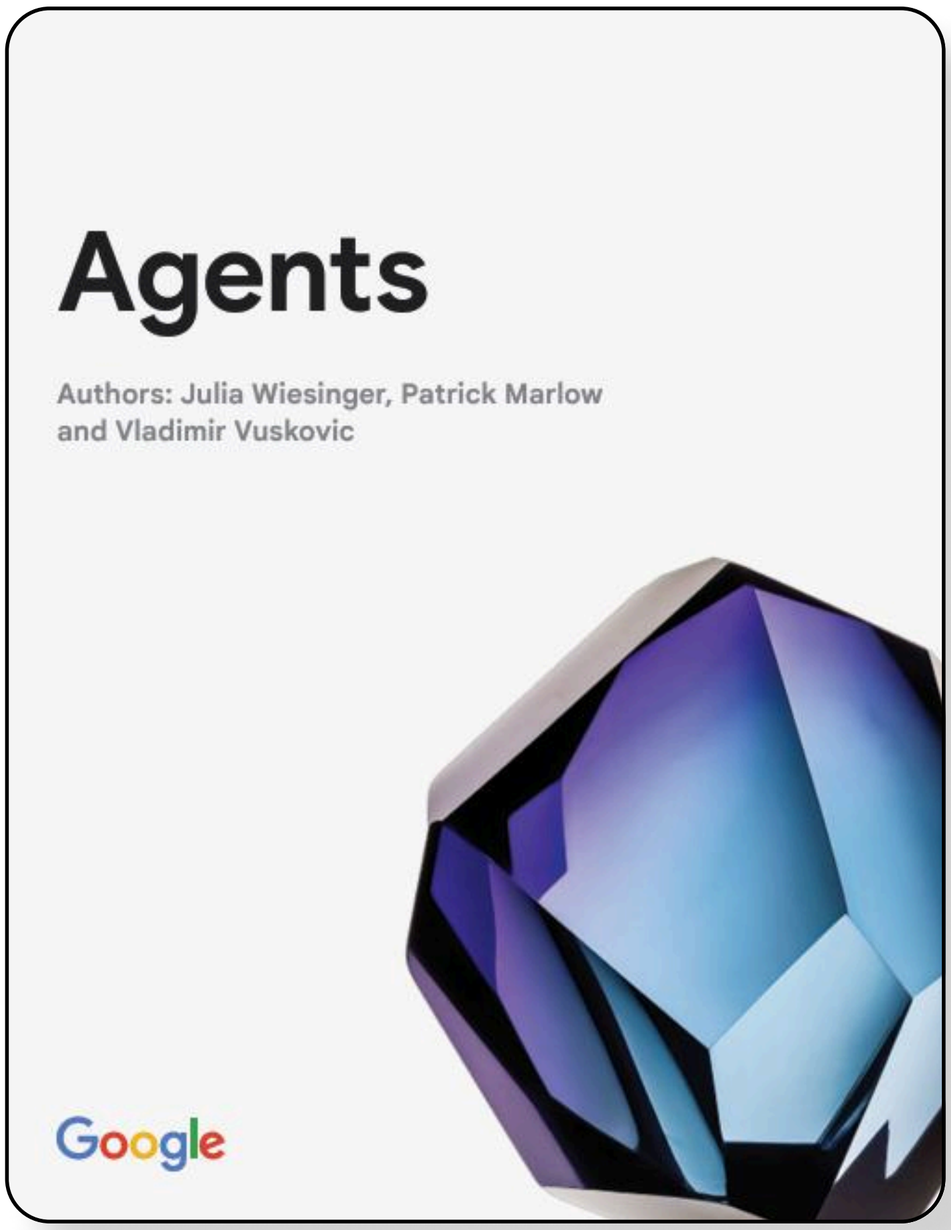
Agents

Whitepaper

Comprehensive summary of Agents whitepaper released by Google

Original Authors: Julia Wiesinger, Patrick Marlow and Vladimir Vuskovic

Summarized by
@rakeshgohel01



Brief

Summary of the a well researched whitepaper on AI Agents.

In this post, i have summarized all the important points talked within the paper in a concise manner, so that it will be easier to understand for different use-cases.

Let’s learn together,

1

What are AI Agents?

Google shares what they think about agentic System and AI Agents in general.

They share how the basic architecture of AI Agents looks like

What is an *Agent*?

According to Google,

A Generative AI agent is an autonomous application that observes and acts on the world to achieve goals without human intervention.

They are autonomous because of various components used within the architecture which makes them very different from other GenAI applications

The paper even lists basic parts of an agent which makes them autonomous.

Those parts are:

Model **Tools** **The Orchestrator**

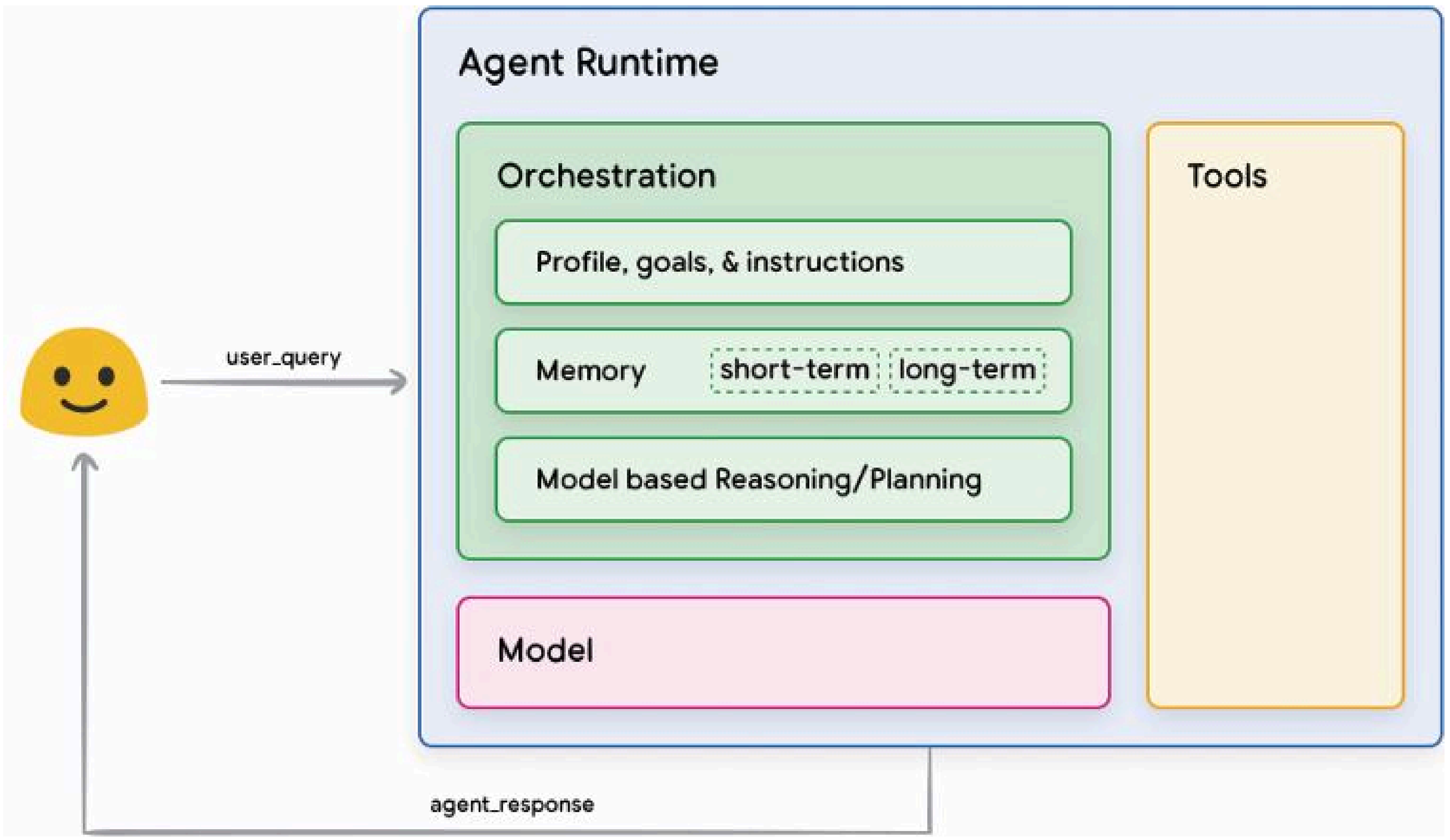


Fig: General agent architecture and components

Description of the core agent architecture:

Model

In an agent's context, a model is the language model (LM) that makes centralized decisions, which can be of any size and follow reasoning frameworks like ReAct or Chain-of-Thought.

For optimal results, choose a model that fits your application and is trained on relevant data, though it may not be trained with the agent's specific settings.

Tools

Foundational models can't interact with the outside world, but tools enable agents to do so by connecting to external data and services.

Tools, often using web API methods like GET and POST, allow agents to perform real-world tasks and support advanced systems like retrieval-augmented generation (RAG).

The Orchestrator

The orchestration layer is a cyclical process where an agent takes in information, reasons internally, and decides its next action until it reaches its goal.

This layer's complexity varies from simple calculations to advanced reasoning techniques, depending on the agent and task.

Since, models are so important in AI Agents.

You might be wondering what seems to be core differences between them?

Core Differences between Models and Agents

Models	Agents
Knowledge is limited to what is available in their training data.	Knowledge is extended through the connection with external systems via tools
Single inference / prediction based on the user query. Unless explicitly implemented for the model, there is no management of session history or continuous context. (i.e. chat history)	Managed session history (i.e. chat history) to allow for multi turn inference / prediction based on user queries and decisions made in the orchestration layer. In this context, a 'turn' is defined as an interaction between the interacting system and the agent. (i.e. 1 incoming event/ query and 1 agent response)
No native tool implementation.	Tools are natively implemented in agent architecture.
No native logic layer implemented. Users can form prompts as simple questions or use reasoning frameworks (CoT, ReAct, etc.) to form complex prompts to guide the model in prediction.	Native cognitive architecture that uses reasoning frameworks like CoT, ReAct, or other pre-built agent frameworks like LangChain.

Cognitive Architecture: How do Agents Operate?

They operate though a distinct Process:

- Gather information (e.g., user input, available data).
- Reason internally to plan actions based on gathered information.
- Execute actions and make adjustments as needed.

To perform, these core components:

- **Orchestration layer**: Maintains memory, state, reasoning, and planning.
- Uses prompt engineering frameworks for effective interaction and task completion.

Few Popular Frameworks they described to improve reasoning are:

- **ReAct:** Guides reasoning and action for user queries, improving human interoperability.
- **Chain-of-Thought (CoT):** Enables reasoning through intermediate steps, with various sub-techniques.
- **Tree-of-Thoughts (ToT):** Suited for exploration tasks, generalizing over CoT for problem-solving.

An example for ReAct

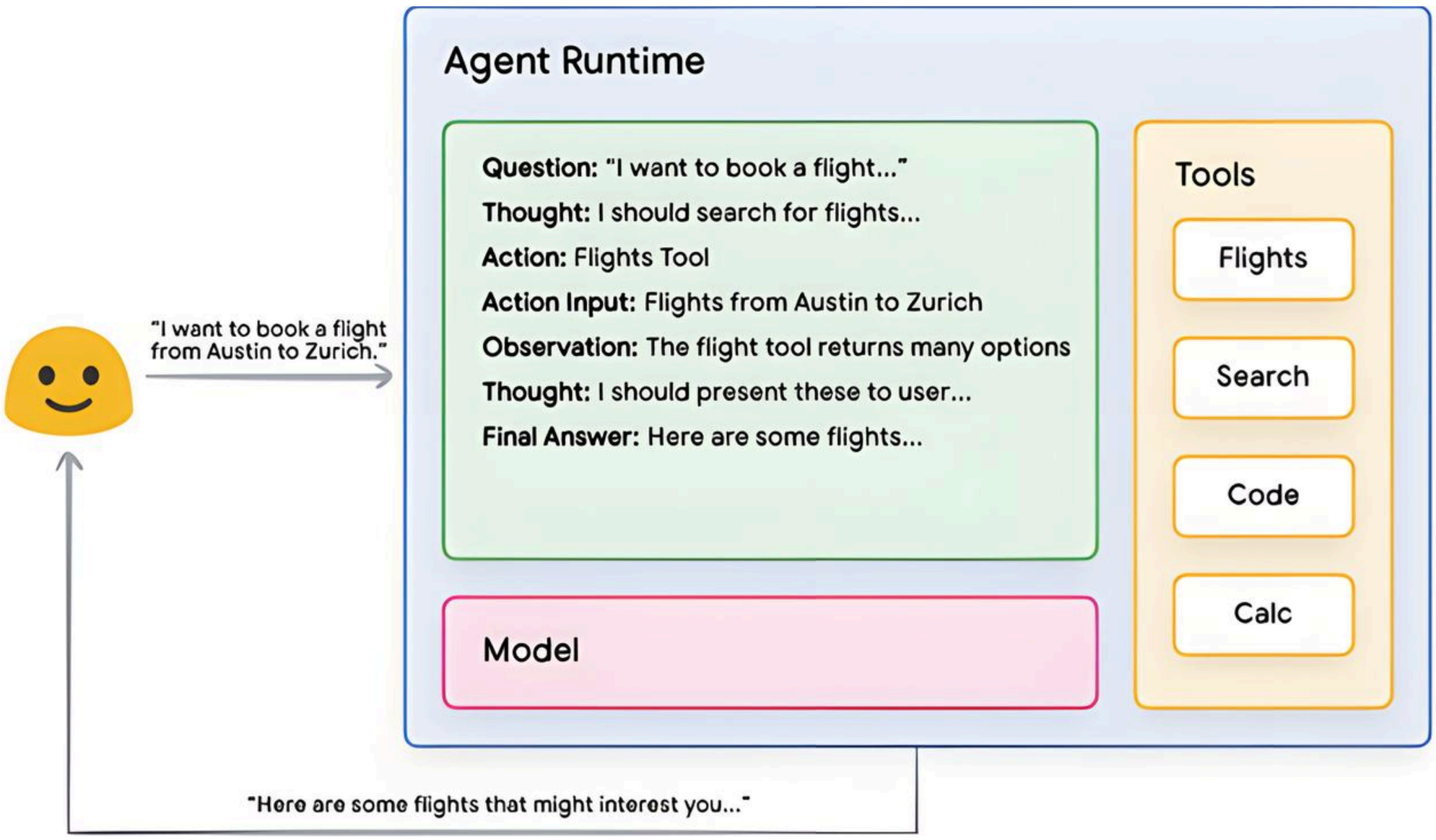


Fig: General ReAct agent architecture

ReAct summarized workflow:

1. The agent receives a user query and initiates the ReAct sequence.

2. It generates a thought, decides on an action (which may involve choosing a tool like Flights, Search, Code, or none), provides inputs to the tool if required, and observes the result.

3. The process repeats as necessary, culminating in a final answer to the user.

2

Tools: A power hand for Agents

Language models can't interact with the real world, limiting their usefulness in situations requiring external data or systems.

Tools like Functions, Extensions, and Data Stores bridge this gap, enabling agents to perform various tasks accurately and reliably.

Extensions

According to Google,

Extensions standardize the connection between APIs and agents, allowing seamless API execution.

For instance, an agent helping users book flights can easily interact with the Google Flights API using Extensions.

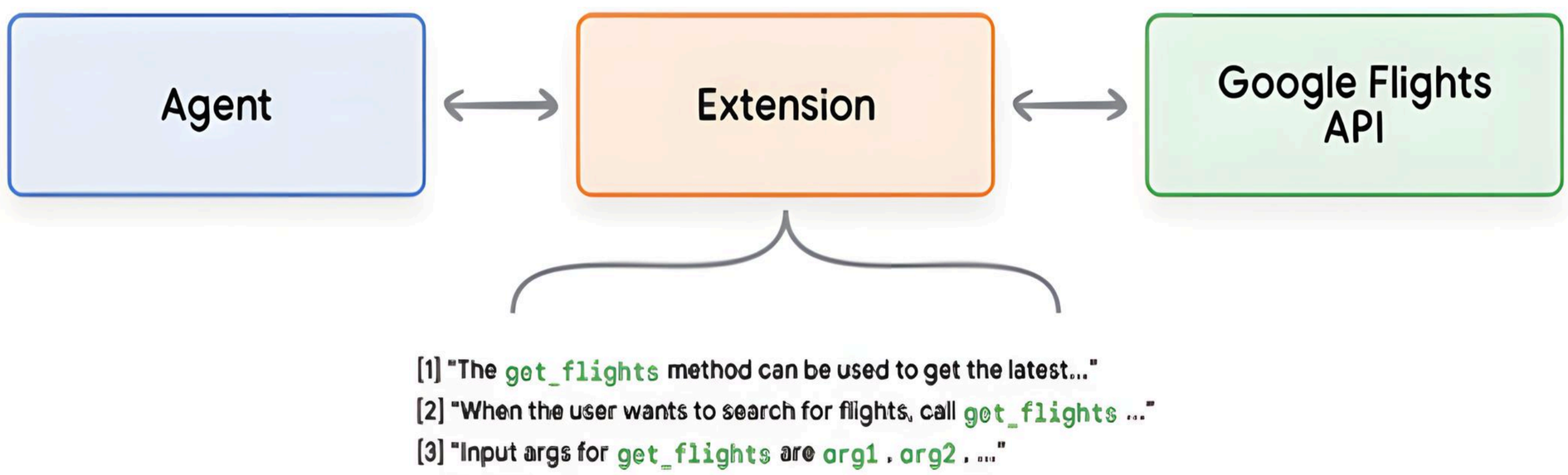


Figure: Extensions connect Agents to External APIs

This approach is more scalable and reliable than implementing custom code to handle various user queries and edge cases

Functions

According to Google,

Functions are reusable code modules that perform specific tasks, with the developer defining when and how to use them.

Unlike Extensions, functions are executed client-side and do not make live API calls, allowing developers more control over data flow.

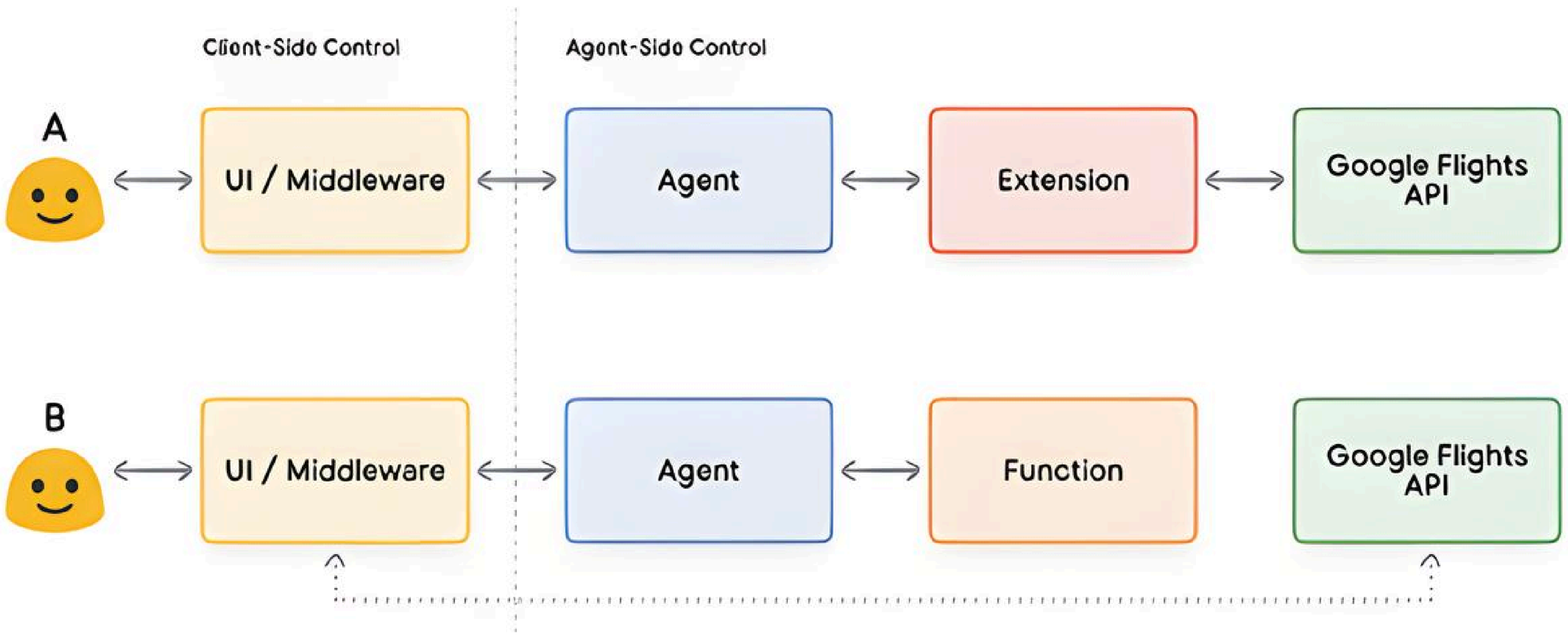


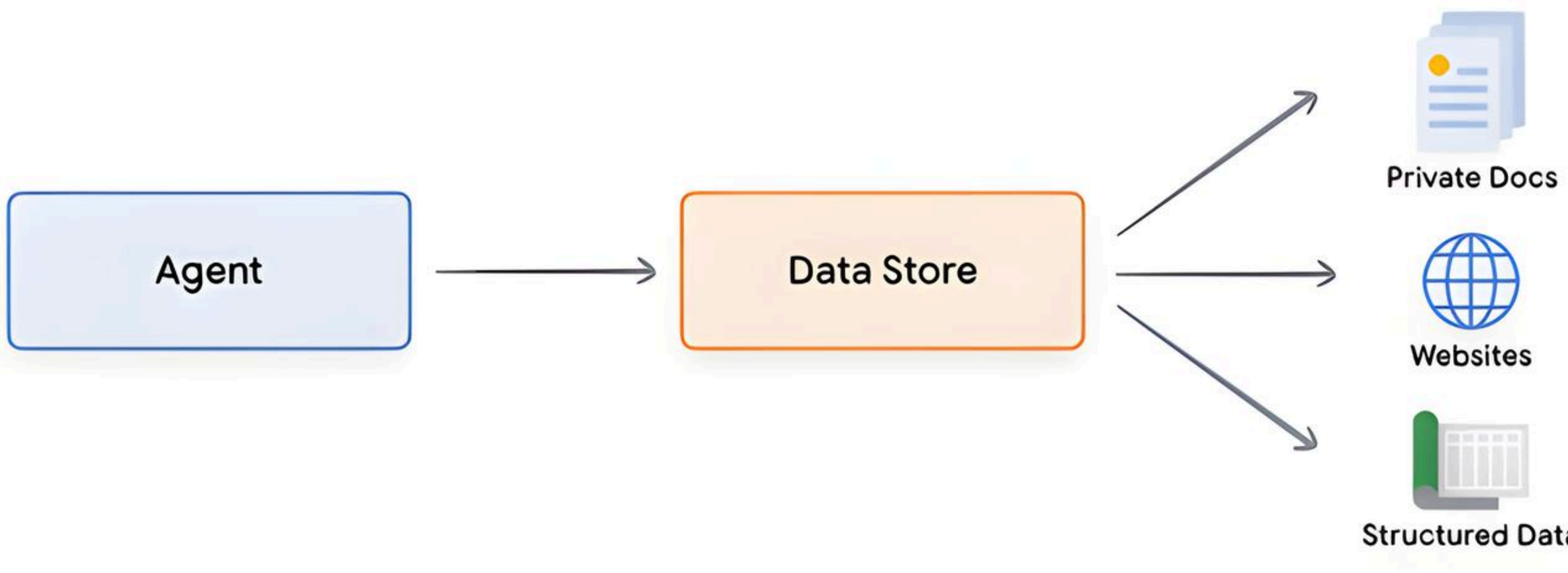
Figure: Delineating client vs. agent side control for extensions and function calling

Data Store

According to Google,

A language model's knowledge is static, like a library with no new books, making it challenging to stay current with evolving information.

Data Stores solve this by providing access to dynamic and up-to-date data, ensuring the model's responses remain factual and relevant.



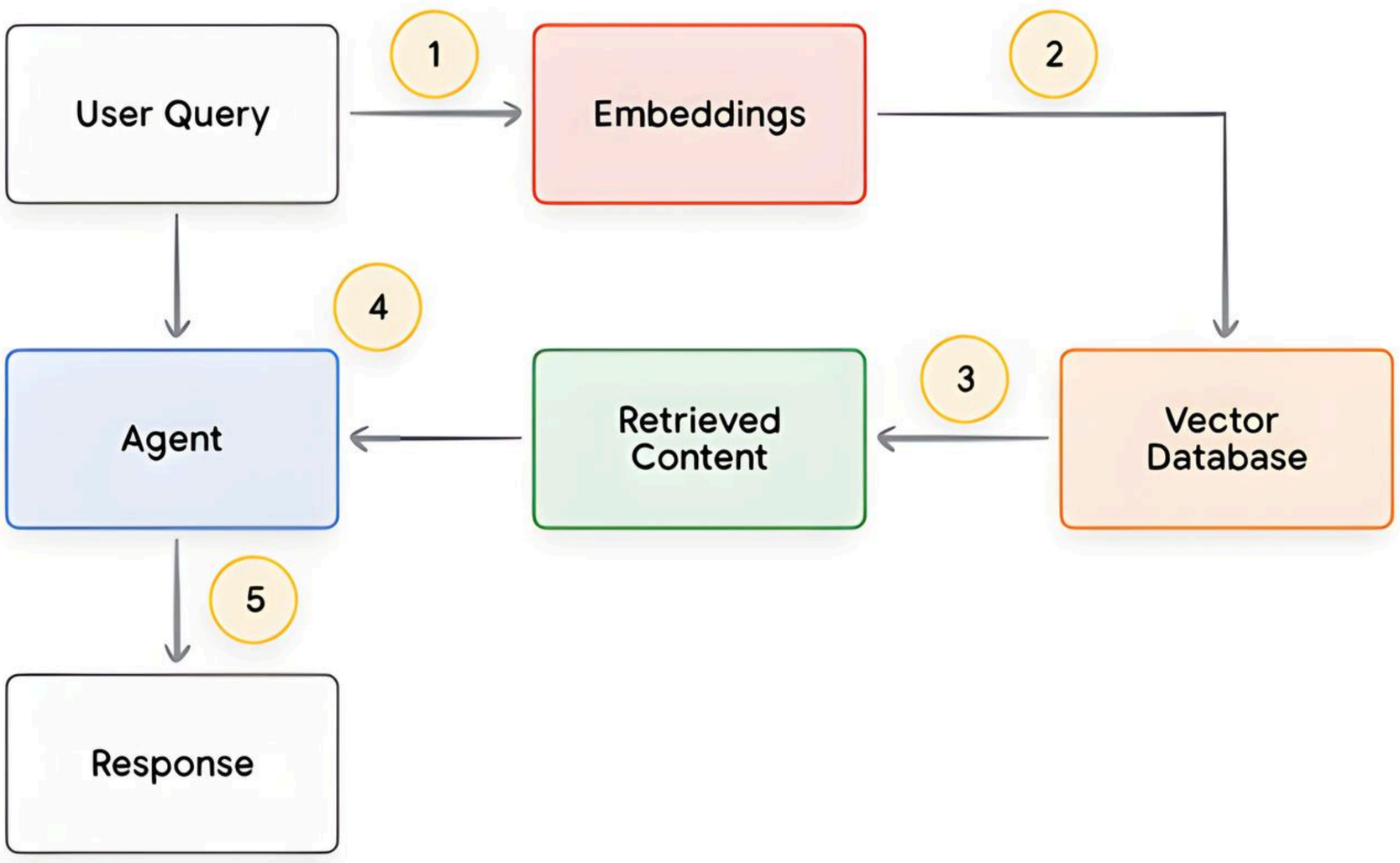
Developers can easily add data in its original format, such as spreadsheets or PDFs, without needing to retrain or fine-tune the mode.

Data Store

RAG Application example

In Generative AI agents, Data Stores are implemented as vector databases that store data as high-dimensional vector embeddings, accessible to the agent at runtime.

A prominent example is Retrieval-Augmented Generation (RAG) application



The process involves generating query embeddings, matching them against the vector database, retrieving relevant content, and formulating a response.

Tool use Summary

	Extensions	Function Calling	Data Stores
Execution	Agent-Side Execution	Client-Side Execution	Agent-Side Execution
Use Case	• Developer wants agent to control interactions with the API endpoints • Useful when leveraging native pre-built Extensions (i.e. Vertex Search, Code Interpreter, etc.) • Multi-hop planning and API calling (i.e. the next agent action depends on the outputs of the previous action / API call)	• Security or Authentication restrictions prevent the agent from calling an API directly • Timing constraints or order-of-operations constraints that prevent the agent from making API calls in real-time. (i.e. batch operations, human-in-the-loop review, etc.) • API that is not exposed to the internet, or non-accessible by Google systems	Developer wants to implement Retrieval Augmented Generation (RAG) with any of the following data types: • Website Content from pre-indexed domains and URLs • Structured Data in formats like PDF, Word Docs, CSV, Spreadsheets, etc. • Relational / Non-Relational Databases • Unstructured Data in formats like HTML, PDF, TXT, etc.

Targeted learning

Using few methods, you can leverage targeted learning which will help you get your desired result faster.

Few of those methods are;

1. **In-context learning:** Provides the model with a prompt, tools, and few-shot examples at inference time, allowing it to learn on the fly how and when to use tools for specific tasks, similar to a chef preparing a dish with limited information.

2. **Retrieval-based in-context learning:** Dynamically populates the model prompt with the most relevant information, tools, and examples from external memory, like a chef choosing ingredients and cookbooks from a well-stocked pantry.

3. **Fine-tuning based learning:** Involves training the model with a larger dataset of specific examples before inference, helping it understand tool application prior to user queries, akin to a chef learning a new cuisine through formal training.

3

Building your Agents with Google

Google has shared sample codes so that you can also get started for building AI agents

Getting started with Langchain

Google’s paper contains two full sample codes to build AI Agents, one of them being Langchain.

You can access these codes from paper linked in comments

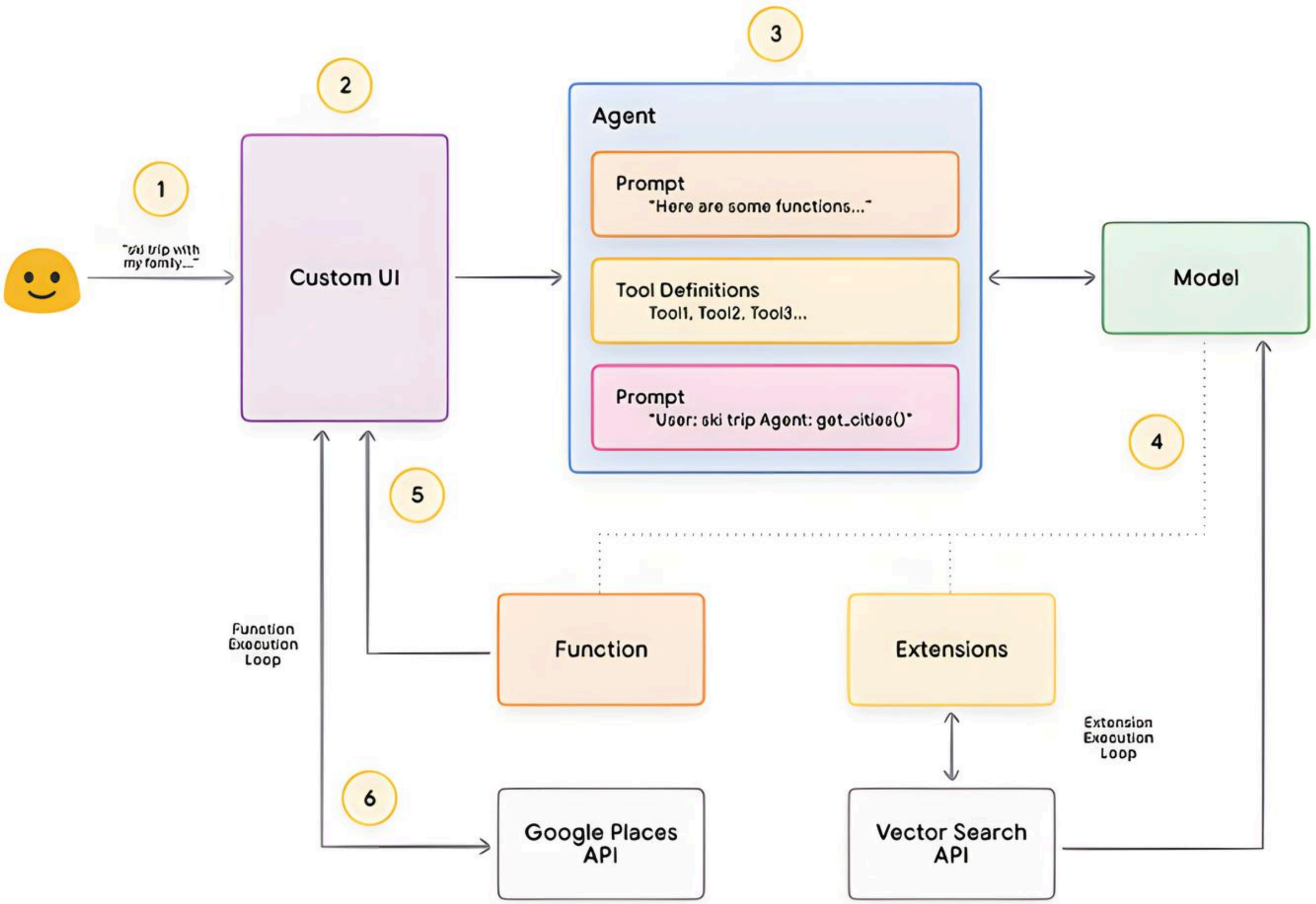
Python

```python
from langgraph.prebuilt import create_react_agent
from langchain_core.tools import tool
from langchain_community.utilities import SerpAPIWrapper
from langchain_community.tools import GooglePlacesTool

os.environ["SERPAPI_API_KEY"] = "XXXXX"
os.environ["GPLACES_API_KEY"] = "XXXXX"

@tool
def search(query: str):
    """Use the SerpAPI to run a Google Search."""
    search = SerpAPIWrapper()
    return search.run(query)

@tool
def places(query: str):
    """Use the Google Places API to run a Google Places Query."""
    places = GooglePlacesTool()
    return places.run(query)

model = ChatVertexAI(model="gemini-1.5-flash-001")
tools = [search, places]

query = "Who did the Texas Longhorns play in football last week? What is the
address of the other team's stadium?"
```

Production grade architecture with Vertex AI agents

Google also shares an architecture for designated workflow designed with dedicated UI and API's for a production ready AI Agents.

This architecture consists of all the required components for building AI Agents



Summary

This whitepaper explored the foundational components of Generative AI agents and their implementation as cognitive architectures.

Agents enhance language models by using tools to access real-time information, suggest actions, and execute complex tasks autonomously.

The orchestration layer is central to an agent's operation, using reasoning techniques like ReAct and Chain-of-Thought to guide decisions and actions.

Tools such as Extensions, Functions, and Data Stores enable agents to interact with external systems and data, while future advancements and 'agent chaining' promise even more sophisticated solutions.

Hi, I am
Rakesh Gohel

“We help businesses 10X their growth with Cloud and AI Agents”

FOLLOW TO LEARN MORE ABOUT AI AGENTS