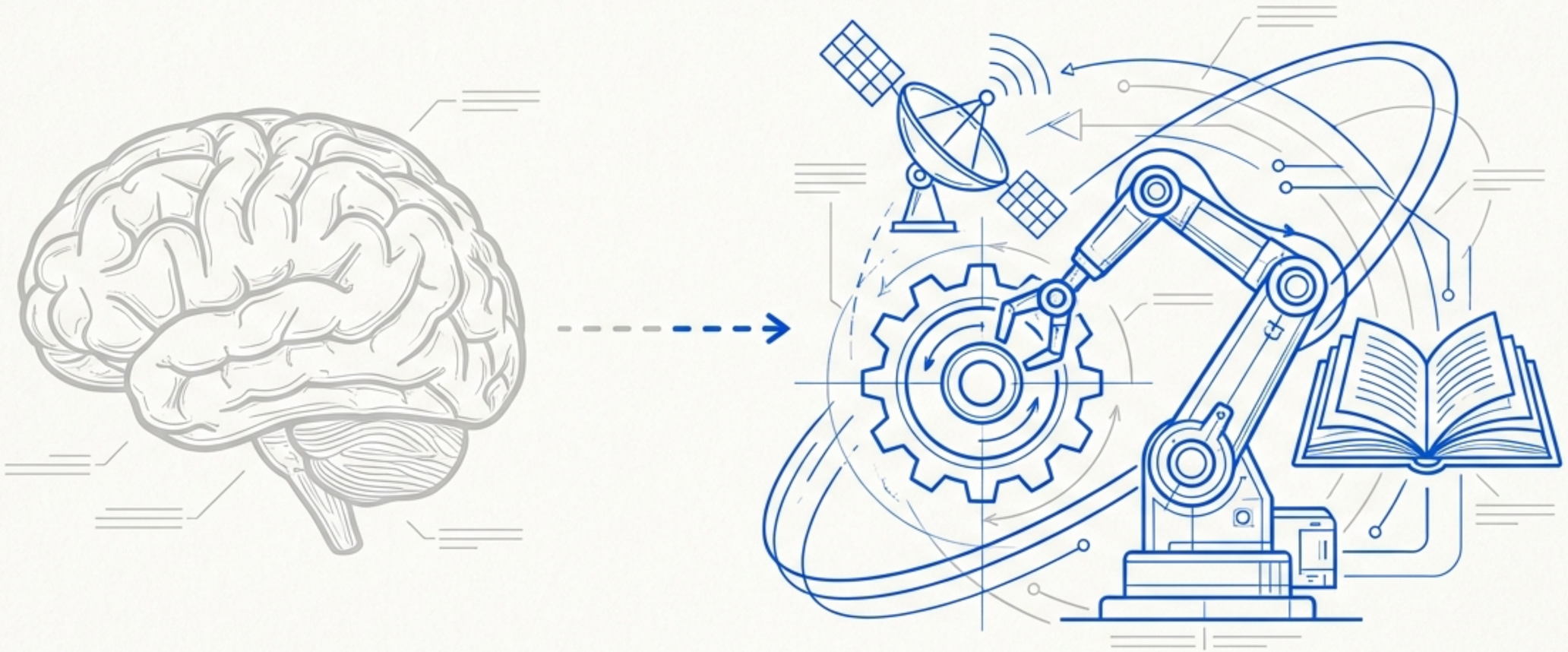
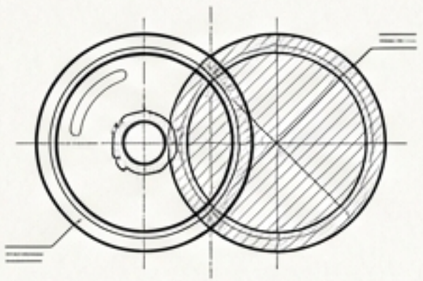
FROM LLMS TO AUTONOMOUS AGENTS

Agent Foundation



WHAT YOU WILL LEARN IN THIS SECTION

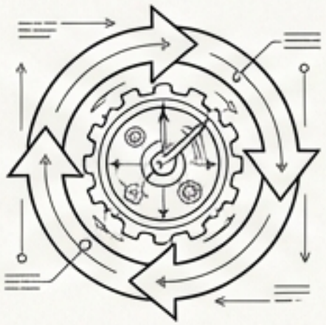
1.



The Critical Distinction

Understand the fundamental difference between Large Language Models and AI Agents.

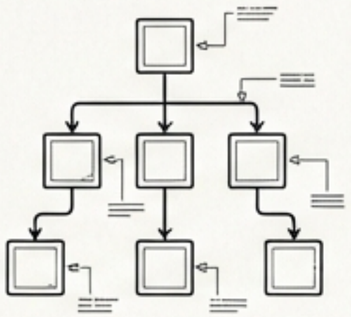
2.



The Engine of Autonomy

Deconstruct the agent's core operational cycle: The Think → Tool → Think → Answer loop.

3.



A Taxonomy of Agents

Explore the four primary types of AI Agents and their specific use cases.

4.

Your First Build

See how an agent is built using practical tools like a calculator and web search.

The Starting Point: The Large Language Model as a Passive Responder

Key Characteristics

- **Nature:** Passive responder.
- **Capability:** Text in → Text out.
- **Behaviour:**
  - Predicts the next token autoregressively.
  - Provides a single-turn response.
  - Has no memory between conversations.
  - Cannot take actions or use external tools.

Example

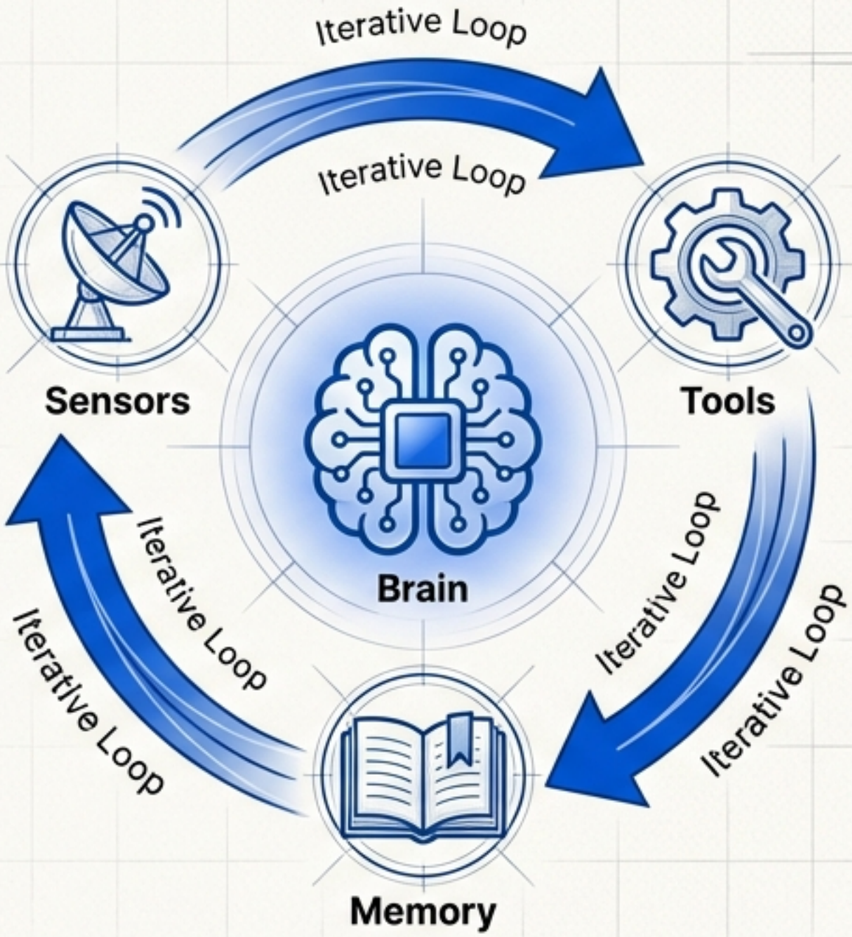
Query: What's 127 × 83?

LLM Process: Generates an answer based on pattern recognition from its training data.

The Evolution: The AI Agent as an Autonomous Actor

Key Characteristics

- **Nature:** Autonomous actor.
- **Capability:** Perceives + Acts + Reasons.
- **Behaviour:**
  - Uses an LLM as its ‘brain’ for reasoning.
  - Engages in a multi-turn, iterative process.
  - Maintains memory and context.
  - Takes actions in an environment using external tools.



Example

Query: What’s 127 × 83?

Agent Process: Decides to use a calculator tool to compute the exact answer.

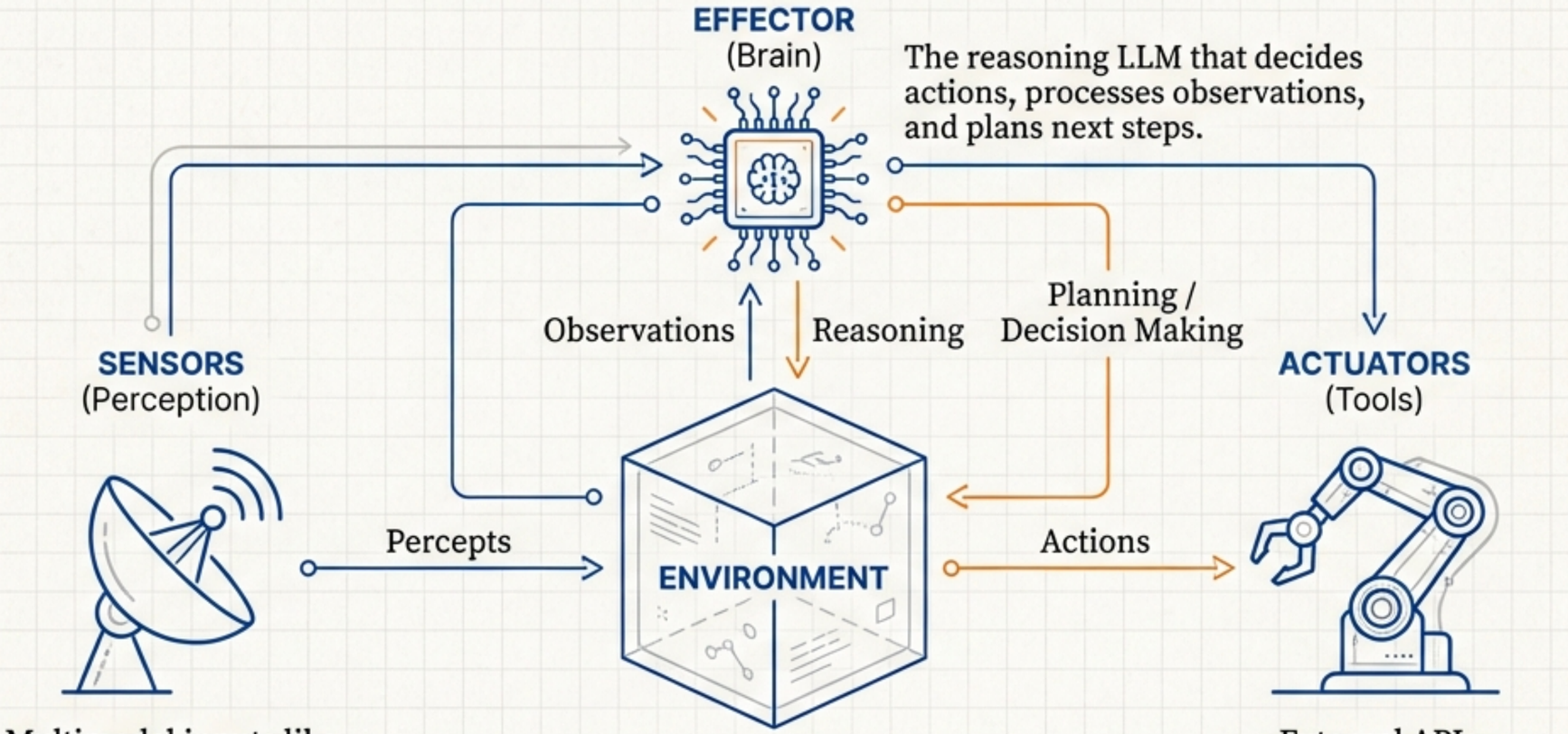
The Core Insight: An Agent Augments a Reasoning Engine with Action Capabilities

An Agent = LLM (brain) + Memory + Tools + Planning/Reflection

The LLM provides the reasoning, while the agent framework provides the ability to act.

ANATOMY OF AN AGENT: THE CORE COMPONENTS OF ACTION

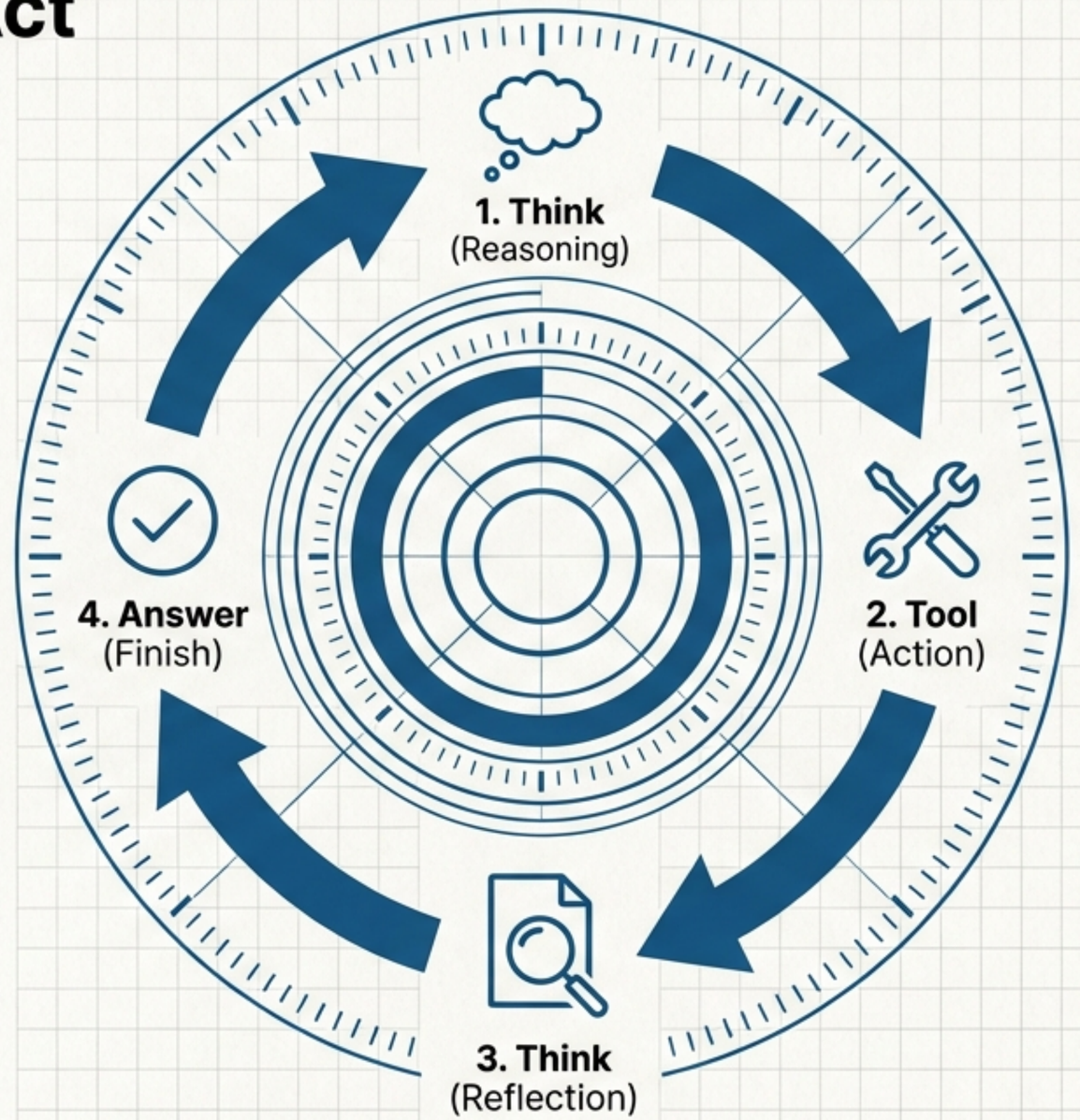
"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators." — Russell & Norvig, AI: A Modern Approach



The Agent’s Engine: The ReAct Pattern Powers the Loop of Thought and Action

Agents follow the ReAct pattern (Reasoning + Acting).

- **Iterative:** The loop can repeat multiple times.
- **Dynamic:** The agent decides when to stop based on goal completion.
- **Autonomous:** No human intervention is needed during the loop (unless configured).



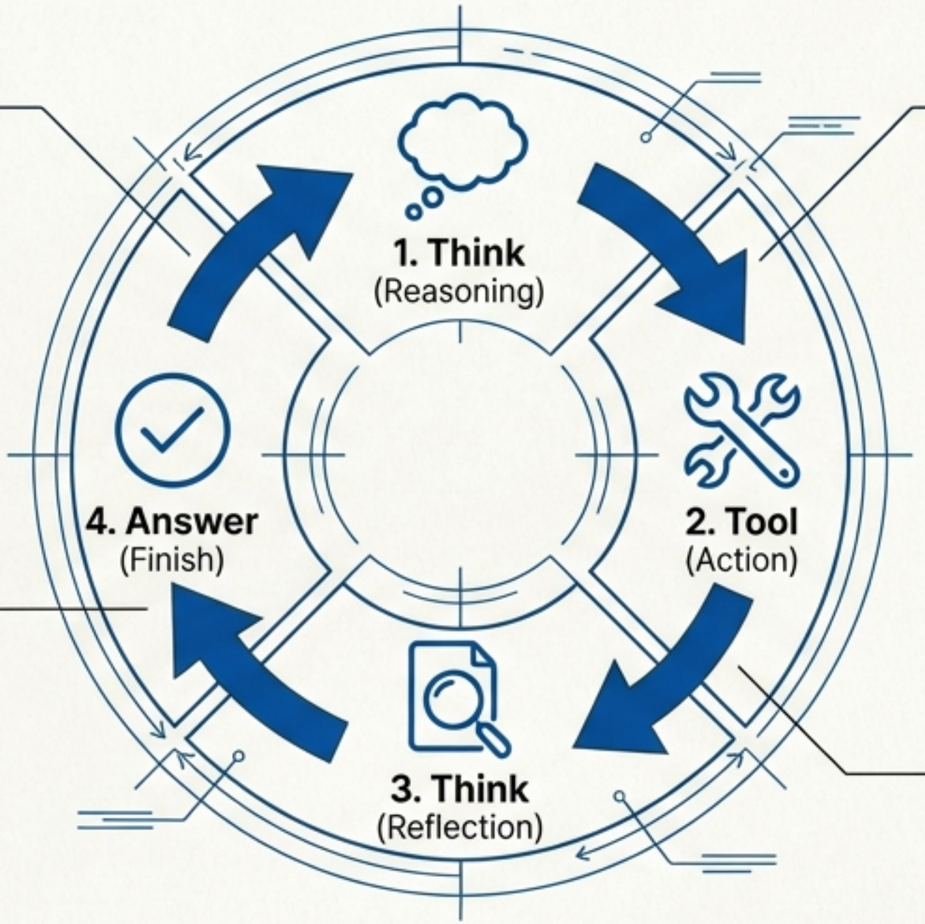
ReAct in Action: Solving a Multi-Step Query

User Query: What is 127 × 83 and is it a prime number?

1. 🤔 Think (Reasoning): The LLM analyses the query and determines it needs to perform a calculation first.

3. 🔍 Think (Reflection): The agent observes the result and reasons about the next step.

```
Reasoning: "Now I need to check if 10,541 is prime"
```



2. 🛠️ Tool (Action): The agent selects and executes the calculator tool.

```
Action: calculator(127 * 83)
Observation: 10,541
```

4. ✅ Answer (Finish): After a potential second tool use (e.g., a primality test function), the agent synthesises the information into a final response.

```
Result: "127 × 83 = 10,541. Based on my analysis, it is not a prime number."
```

A Taxonomy of Agents: Four Patterns for Different Tasks

While ReAct is the most common pattern, specialised agents are designed for more complex, diverse, or critical tasks. The choice of agent architecture directly impacts performance and reliability.

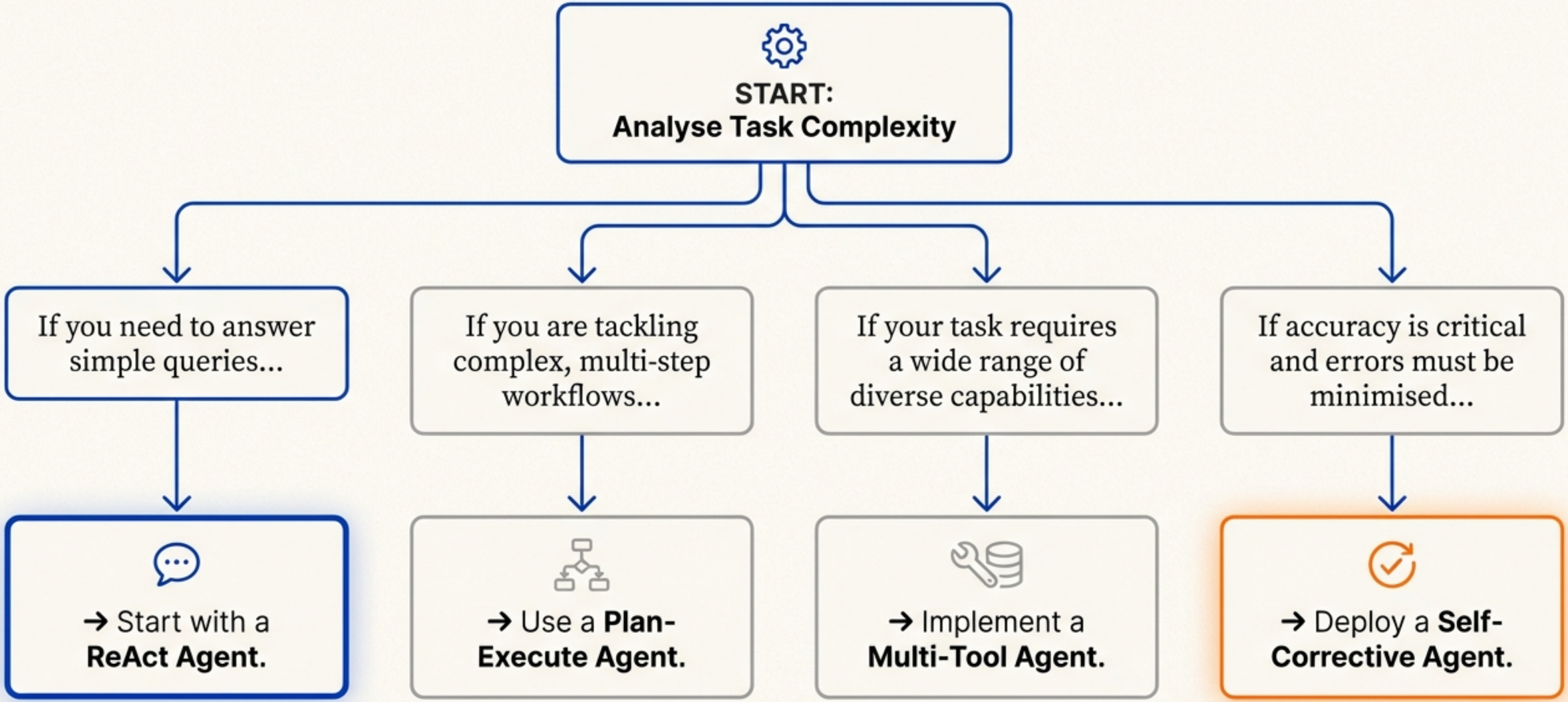
ReAct Agent

Plan-Execute Agent

Multi-Tool Agent

Self-Corrective Agent

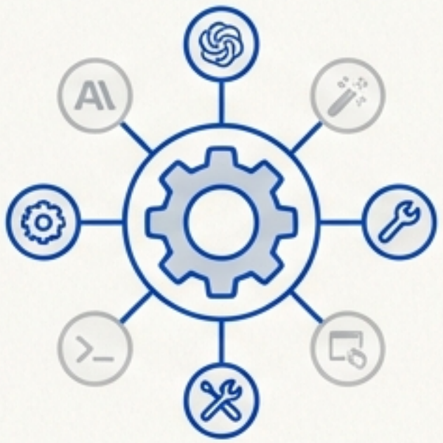
How to Choose the Right Agent for Your Task



THE BUILDER'S STACK: LANGCHAIN FRAMEWORK AND GEMINI MODELS

Why LangChain? The Agent Framework

- Open-source, most mature ecosystem (since 2022).
- LangChain 1.0 is the first stable release.
- Works with any LLM (Google, OpenAI, Anthropic, local).



Our Recommended Model Strategy: Two-Model Approach

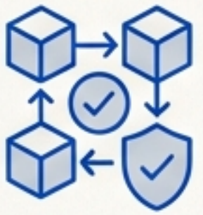
Primary: Gemini 3 Flash Preview

Fast, cost-effective, frontier intelligence.

Use for: Most agent tasks.

`gemini-3-flash-preview`

Alternative: Gemini 2.5 Flash

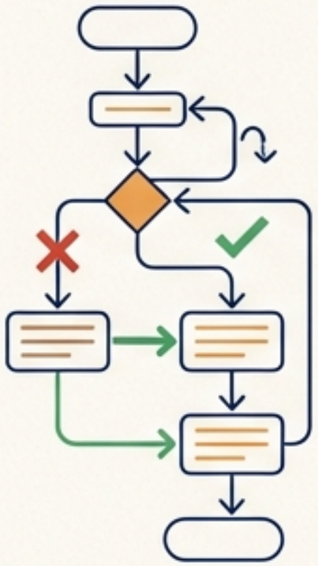


Stable, production-proven reliability.

Use for: Cost-sensitive workloads.

FROM BLUEPRINT TO PRODUCTION: ESSENTIAL ENGINEERING PATTERNS

1. Error Handling & Fallbacks



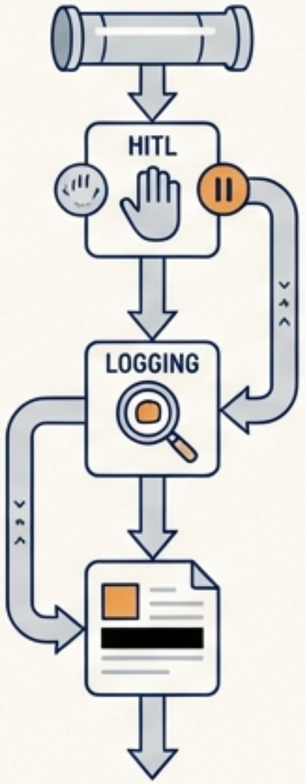
Strategies:

- Simple retry, exponential backoff, graceful degradation.

Fallback Strategy:

- Plan for model failure (e.g., Primary: Gemini 3 Flash → Fallback: Gemini 2.5 Gemini 2.5 Flash → Backup: Local Ollama).

2. Core Middlewares



Human-in-the-Loop (HITL):

- Pause for human approval on critical actions (payments, public posts).

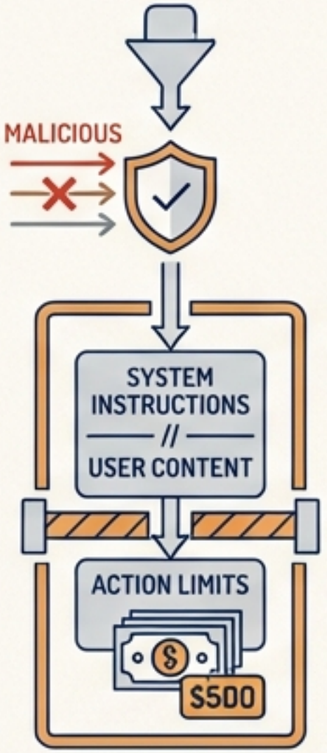
Logging & Tracing:

- Record all interactions for debugging, compliance, and analytics.

PII Redaction:

- Remove sensitive data (credit cards, SSNs) before logging.

3. Safety Guardrails



Input/Output Validation:

- Check user input for malicious patterns and model responses for safety.

Prompt Injection Defense:

- Separate system instructions from user content.

Action Constraints:

- Limit what an agent can do (e.g., Approve refunds only up to $500).

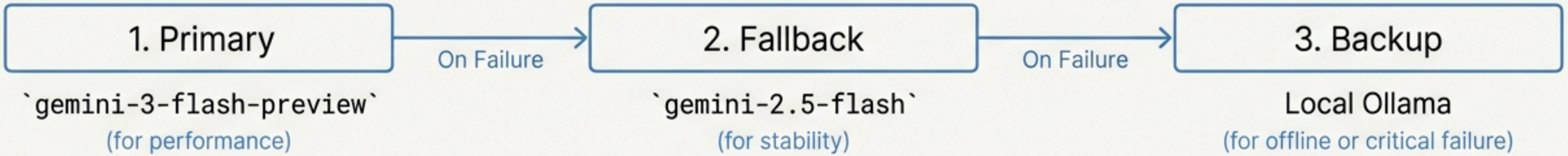
Powering Your Agent with a Strategic Two-Model Approach

A production-grade strategy involves using the right model for the right job and having a reliable fallback.

Model	Key Characteristics	Primary Use Case
Gemini 3 Flash	• Release: Jan 2026 (Preview) • Strength: Frontier intelligence at Flash speed • Model Name: `gemini-3-flash-preview`	Most agent tasks requiring speed and power.
Gemini 2.5 Flash	• Release: June 2025 (Stable) • Strength: Proven reliability • Cost: $0.30/M input, $2.50/M output tokens	Cost-sensitive workloads or as a fallback.

Fallback Strategy

Your agent's configuration should be robust:

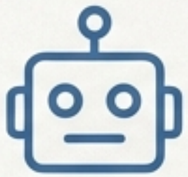


The Language of LangChain: Core Message Types

Every turn in a conversation is represented by a specific message class. Understanding these is key to building and debugging agents.

`HumanMessage`

Represents input from the user. Can contain text, images, or other file types.



`AIMessage`

Represents the model's response. Contains the output content and can include requests for tool calls and token usage metadata.

`SystemMessage`

Provides high-level instructions or **context to the agent**. This sets the **agent's** role, personality, and **constraints** (e.g., "You are a helpful financial assistant").

`ToolMessage`

Contains the results from a **tool execution**. It's explicitly linked back to the specific `AIMessage` that requested the tool call.

The Construction Plan: From a Simple Tool to a Sophisticated System

We will build our agent floor by floor, mastering fundamental concepts at each stage before adding complexity. This ensures a solid foundation for advanced skills.

Ground Floor: The YouTube Video Analyzer

Project Goal

Build an agent that takes a YouTube URL and provides a concise analysis.

Functionality

1. Accepts a YouTube video URL as input.
2. Uses a tool to fetch the video's transcript.
3. Summarizes the transcript content.
4. Extracts a bulleted list of key points.

Technical Details

- Model: Gemini 3 Flash Preview or 2.5 Flash (a simple task, good for either).
- Skills You'll Master: Basic agent structure, Integrating and calling a single tool, Fundamental error handling.
- Complexity: ⭐ Beginner

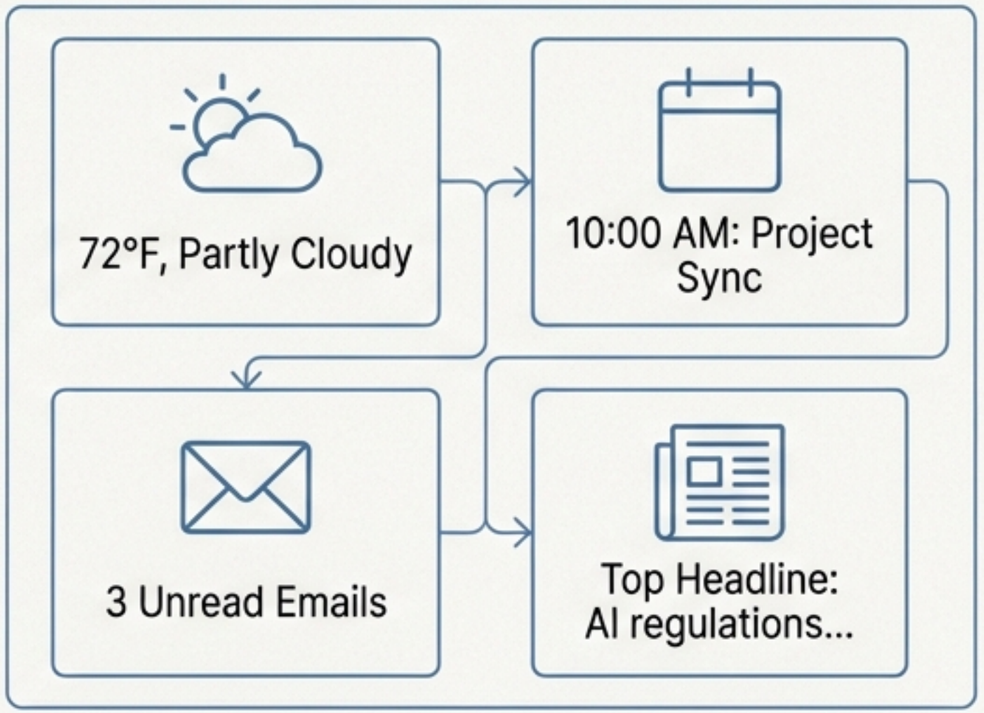
Ascending Complexity: The Daily Briefing Agent

Project Goal

Create a personalized morning briefing by orchestrating multiple tools.

Functionality

- Fetches today's weather forecast.
- Pulls events from your calendar.
- Summarizes unread emails.
- Gets the latest news headlines.
- Combines all information into a single, coherent briefing.



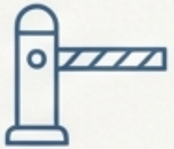
Technical Details

- ****Model****: Gemini 3 Flash Preview (ideal for its speed and cost-effectiveness in multi-step tasks).
- ****Skills You'll Master****: Multi-tool orchestration and planning, Using short-term memory to maintain context across steps, Streaming responses for a better user experience.
- ****Complexity****: ★★ Intermediate

The Professional's Toolkit: Crafting the Agent's Brain

Tools: The Agent's Capabilities	Prompt Engineering: The Agent's Instructions	Context Engineering: The Agent's Knowledge
External functions your agent can execute.	The art of writing clear, effective system instructions.	The data provided to the agent beyond the user query.
Examples: • Search APIs • Database Queries • Calculators • File I/O • Custom Functions	**Structure:** 1. **Role**: Who are you? 2. **Capabilities**: What can you do? 3. **Constraints**: What can't you do? 4. **Tools**: Here are your functions. 5. **Examples**: Show desired behavior.	**Types:** • **System**: Business rules, policies • **User**: Chat history, preferences • **Task**: Data for the current request • **Domain**: Business data
LangChain offers **100+ built-in tool integrations**.	**Tip:** Be specific. Test and iterate constantly.	**Challenge:** Balance providing enough information against token limits and cost.

Building for Reliability: Guardrails, Middlewares, and and Error Handling

 **Error Handling: Responding to Failure**	 **Middlewares: Intercepting the Flow**	 **Guardrails: Enforcing Safety**
What to do when things go wrong. **Strategies:** • Simple Retries • Exponential Backoff • Fallback Models • Graceful Degradation	Essential layers for control, security, and observability. **Key Types:** 1. **Human-in-the-Loop (HITL):** Pause for approval on critical actions (e.g., payments). 2. **Logging:** Record interactions for debugging. 3. **PII Redaction:** Remove sensitive data.	Proactive safety layers to constrain agent behavior. **Layers:** • **Input Validation:** Check user input for malicious patterns. • **Prompt Injection Defense:** Isolate system and user instructions. • **Output Validation:** Check responses before delivery. • **Action Constraints:** Limit agent actions (e.g., refund max of $500).

Your Production-Ready Checklist

Before deploying any agent, conduct a final inspection against these critical points.

- [x] **Model Configuration**: Primary model and fallbacks are set.
- [x] **Tracing**: LangSmith or another observability tool is enabled. You can't fix what you can't see.
- [x] **Streaming**: Responses are streamed for optimal user experience.
- [x] **Error Handling**: A robust retry and fallback strategy is in place.
- [x] **Memory**: The correct memory type for the use case has been chosen and configured.
- [x] **Middlewares**: HITL, Logging, and PII Redaction are implemented where necessary.
- [x] **Context**: Context retrieval is optimized for relevance and cost.
- [x] **Prompts**: Prompts have been tested against a variety of inputs.
- [x] **Guardrails**: Input, output, and action constraints are active.

Blueprints for Success: Critical Dos and Don’ts

Do ☑	Don’t ☒
Start simple and build incrementally.	Overcomplicate your prompts or agent from the start.
Test each component in isolation.	Ignore error handling until it’s too late.
Monitor everything with tracing.	Skip tracing—it makes debugging nearly impossible.
Fail gracefully with fallbacks.	Forget guardrails and safety layers.
Implement human oversight for critical actions.	Allow autonomous action on sensitive tasks without a HITL step.

Every Expert Started as a Beginner. Your Journey Starts Now.

Core Principles to Remember

1. **LangChain 1.0 is your foundation.** Build on a stable, mature framework.
2. **Start with the YouTube Analyzer.** Master the basics before adding complexity.
3. **Production-readiness is safety.** Guardrails and middlewares are not optional.
4. **Always trace.** You cannot improve what you cannot measure.

Your Next Steps

1. **Begin Week 1:** Clone the repository and build the YouTube Analyzer.
2. **Follow the Path:** Don't skip the intermediate steps; each one builds a critical skill.
3. **Practice Consistently:** Building is a craft that requires daily practice.
4. **Explore the Resources:** Dive into the official LangChain documentation and join the community.

Connect & Learn More

YouTube: KGP Talkie
Website: kgptalkie.com
Official Docs: docs.langchain.com