Agents 101

An Introduction to Intelligent AI Agents

From concepts to code to career

Abinaya Mahendiran
CTO, Nunnari Labs
LinkedIN: https://www.linkedin.com/in/abinayamahendiran/

February 10, 2026 | Nunnari Labs

Roadmap

What we'll cover today

01 **What Are AI Agents?**

Definitions, analogies & mental models

02 **Core Building Blocks**

Perception, reasoning, memory & action

03 **Agents vs. Chatbots vs. ML Pipelines**

Understanding the spectrum

04 **The Agent Loop**

How agents think & act — step by step

05 **Types of Agents**

From simple reflex to multi-agent systems

06 **Real-World Use Cases**

Coding, research, customer support & more

07 **Frameworks & Tools**

LangChain, CrewAI, AutoGen, OpenAI SDK & more

08 **Career Opportunities**

Roles, skills & growth paths

01

What Are AI Agents?

Let's start with the fundamentals

An AI Agent is a software system that can:

- Perceive its environment (read data, receive instructions)
- Reason about what to do next (using an LLM as its "brain")
- Take actions autonomously (call APIs, write code, search the web)
- Learn & adapt from feedback (memory, reflection, iteration)

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

The Analogy: From Thermostat to AI Agent

Agents exist on a spectrum of intelligence

Reflex Agent

Simple Thermostat

- Senses temperature
- Follows fixed rules
- “If cold → turn on heat"
- No learning, no memory

Goal-Based Agent

Smart Assistant (Siri)

- Understands speech
- Looks up information
- Follows user commands
- Limited autonomy

Autonomous Agent

AI Agent (e.g. Devin)

- Reads a task description
- Breaks it into sub-tasks
- Writes & tests code
- Iterates until done

02

Core Building Blocks

The four pillars of every AI agent

Brain (LLM)

Perception → Reasoning

The reasoning engine. GPT-4, Claude, Llama that decides what to do.

Tools

Action Layer

APIs, web search, code execution, databases — how agents act on the world.

Memory

State Management

Short-term (context window) & long-term (vector DBs) for persistent knowledge.

Planning

Strategy Engine

Task decomposition, chain-of-thought, reflection & self-correction.

03

Agents vs. Chatbots vs. ML Pipelines

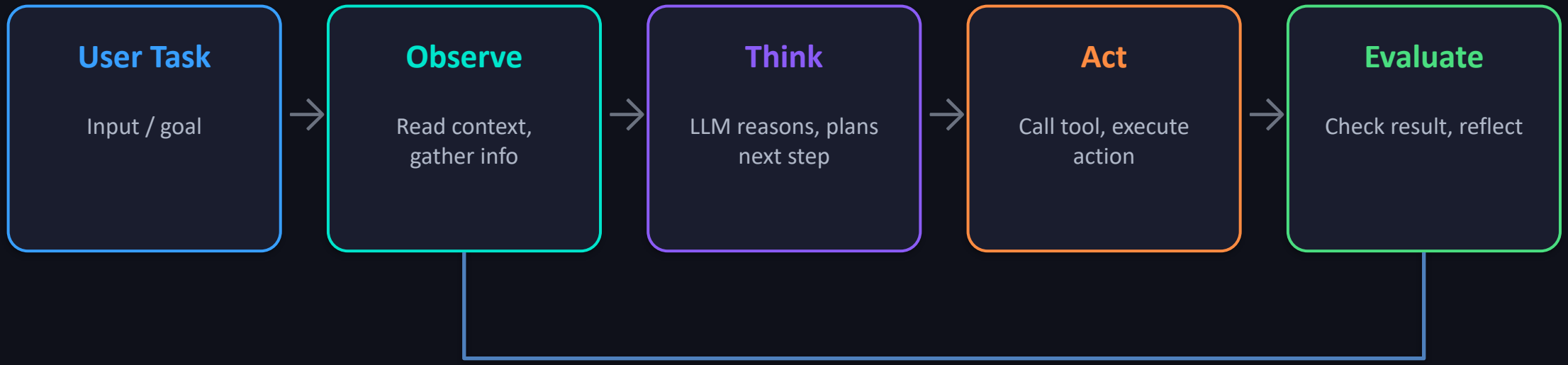
Understanding the spectrum of AI systems

Feature	Chatbot	ML Pipeline / Workflows	Agent
Autonomy	None — waits for each prompt	Pre-defined steps executed in order	Decides its own next steps
Tool Use	No tool access	Fixed tool sequence	Dynamically picks tools as needed
Memory	Only current conversation	Passed between steps explicitly	Short + long-term memory systems
Error Handling	User must re-prompt	Fails or retries fixed logic	Self-corrects, retries, adapts
Example	FAQ bot on any website	RAG pipeline	Devin, Claude Code, AutoGPT

04

The Agent Loop

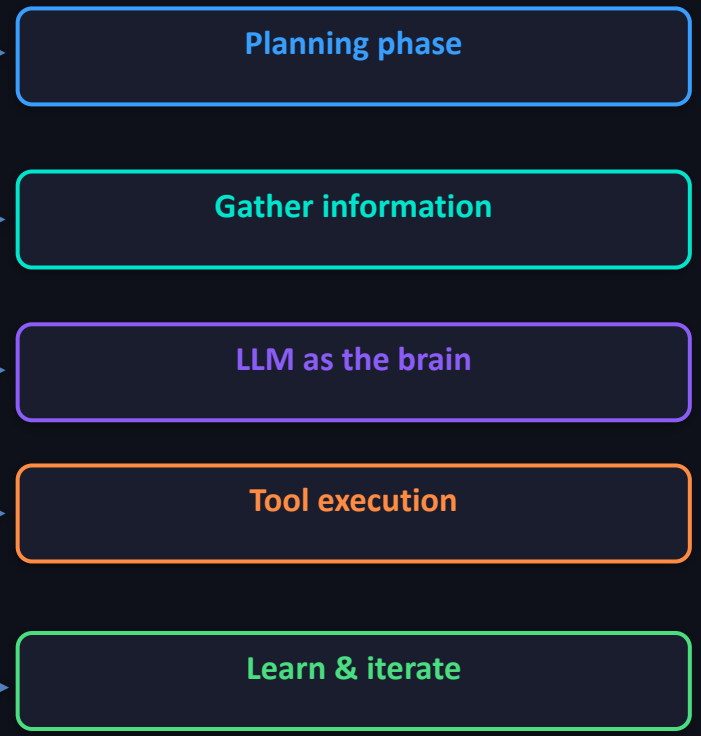
How agents think and act — the core execution cycle



The Agent Loop — In Code

Pseudocode for the core execution cycle

```
def agent_loop(task: str):
    memory = []
    plan = llm.think(f"Break this into steps: {task}")

    while not is_complete(plan):
        # 1. OBSERVE — gather current context
        context = observe(environment, memory)

        # 2. THINK — ask LLM what to do next
        action = llm.decide(context, plan, memory)

        # 3. ACT — execute the chosen action
        result = execute_tool(action)

        # 4. EVALUATE — check and learn
        memory.append({"action": action, "result": result})
        plan = llm.reflect_and_update(plan, result)

    return plan.final_output
```



05

Types of AI Agents

From simple to sophisticated

Simple Reflex

Acts on current perception only. If-then rules.

Example

Spam filter, thermostat

Model-Based

Maintains internal state of the world. Tracks changes.

Example

Self-driving car

Goal-Based

Works toward a specific objective. Plans actions.

Example

GPS navigation, game-playing AI

Utility-Based

Maximizes a utility function. Handles trade-offs.

Example

Recommendation systems

Learning

Improves from experience. Adapts over time.

Example

AlphaGo, modern LLM agents

Simple

Sophisticated →

06

Real-World Use Cases

Where agents are making an impact right now

Coding Agents

Devin, Claude Code, GitHub Copilot Agent, Cursor → Write, debug & deploy code autonomously

Research Agents

Deep Research (OpenAI, Gemini, Perplexity) → Multi-step web research with cited reports

Customer Support

Intercom Fin, Sierra, Decagon → Resolve tickets end-to-end, not just answer FAQs

Data Analysis

Julius AI, Claude Artifacts → Ingest data, run analysis, produce visualizations

DevOps & SRE

PagerDuty AI, Shoreline → Detect incidents, diagnose root cause, auto-remediate

Personal Assistants

Rabbit R1, Multi-On, Zapier AI → Book flights, fill forms, manage workflows

Key Concept: Tool Use & Function Calling

How agents interact with the outside world

What is Function Calling?

The LLM doesn't just generate text — it can output structured requests to call external functions.

Analogy: A Chef in a Kitchen

- The LLM is the chef (decides what to cook)
- Tools are kitchen equipment (oven, mixer, knife)
- The chef calls for the right tool at the right time
- Results come back → chef decides next step

Example Flow

```
User: "What's the weather in Tokyo?"

LLM thinks → I need the weather tool

LLM outputs:
  {"tool": "get_weather",
   "args": {"city": "Tokyo"}}

System executes → calls weather API

Tool returns: {"temp": "8°C", ...}

LLM: "It's currently 8°C in Tokyo"
```

Key Concept: Memory & RAG

How agents remember and retrieve knowledge

Short-Term Memory

The LLM's context window. Holds the current conversation and recent tool results.

Like your working memory when solving a math problem.

~128K–1M tokens

Long-Term Memory

Vector databases (Pinecone, Chroma, Weaviate) store past interactions as embeddings.

Like your brain's long-term storage — searchable.

Unlimited, persistent

RAG (Retrieval-Augmented Generation)

Before answering, the agent searches a knowledge base and injects relevant docs.

Like open-book exam — look up, then answer.

Dynamic knowledge

Key Concept: The ReAct Pattern

Reason + Act — the most widely used agent design pattern

ReAct (Reason + Act)

Introduced by Yao et al., 2022 — now the default pattern in most frameworks

```
Thought: I need to find the population of France
Action: search("population of France 2025")
Observation: France population: ~68.4 million
Thought: Now I have the answer, I can respond
Answer: France has approximately 68.4 million people
```

Why ReAct Works

- Forces the LLM to show its reasoning before acting
- Observations ground the agent in real data — reduces hallucination
- Easy to debug — you can read the full thought trace

Analogy: A Detective Solving a Case

- Think: "The suspect was last seen near the docks"
- Act: Go to the docks, interview witnesses
- Observe: Witness says suspect boarded a boat
- Think: "I should check the marina records next"
- Each clue leads to the next action — just like an agent!

Planning Strategy: Chain-of-Thought (CoT)

"Let's think step by step" — the simplest yet most powerful prompting technique

What is Chain-of-Thought?

Instead of jumping to an answer, the LLM is prompted to break its reasoning into explicit intermediate steps.

Analogy: Showing Your Work in Math Class

- Without CoT: "What is 17 × 24?" → "408"
- With CoT: "17 × 24 = 17 × 20 + 17 × 4 = 340 + 68 = 408"
- The step-by-step trace makes errors visible and fixable

Key Insight:

CoT doesn't add new knowledge — it unlocks reasoning the LLM already has by forcing it to "think out loud".

Prompt Example

```
system_prompt = """
You are a helpful assistant.
Always think step-by-step before
giving your final answer.

Format:
Step 1: ...
Step 2: ...
Final Answer: ...
"""
```

When to Use CoT

- Math, logic, and multi-step reasoning tasks
- Any task where accuracy > speed
- Foundation of almost every agent's system prompt

Planning Strategy: Plan-and-Execute

Make the full plan first, then execute each step — like a project manager

Phase 1: Planning (Big Model)

A powerful LLM creates the full task breakdown

1. Research competitor pricing models
2. Gather our current cost data from DB
3. Build comparison spreadsheet
4. Generate analysis report
5. Draft executive summary email

Phase 2: Execution (Worker Agents)

Smaller/cheaper agents execute each step independently

Step 1	✓ Done	Browsed 5 competitor sites
Step 2	✓ Done	Queried PostgreSQL, got cost CSV
Step 3	⟳ Running	Building spreadsheet with pandas...
Step 4	◯ Pending	Waiting for Step 3 output
Step 5	◯ Pending	Waiting for Step 4 output

Analogy: A general doesn't fight — they plan the battle, then troops execute. | Used in: LangGraph, BabyAGI, many production systems

Planning Strategy: Reflexion

Learn from failure — agents that write themselves post-mortems

The Reflexion Loop

1. Attempt

Agent tries to solve the task

↓

2. Evaluate

Check: did the output pass tests / meet criteria?

↓

3. Reflect

If failed: "What went wrong? What should I do differently?"

↓

4. Retry

Agent tries again with the reflection as added context

Example: Code-Writing Agent

```
Attempt 1: def sort(lst): return lst.sort()
Test: ✗ Failed — returns None, not sorted list

Reflection: "lst.sort() modifies in-place and
returns
 None. I should use sorted() instead."

Attempt 2: def sort(lst): return sorted(lst)
Test: ✓ Passed — all test cases green
```

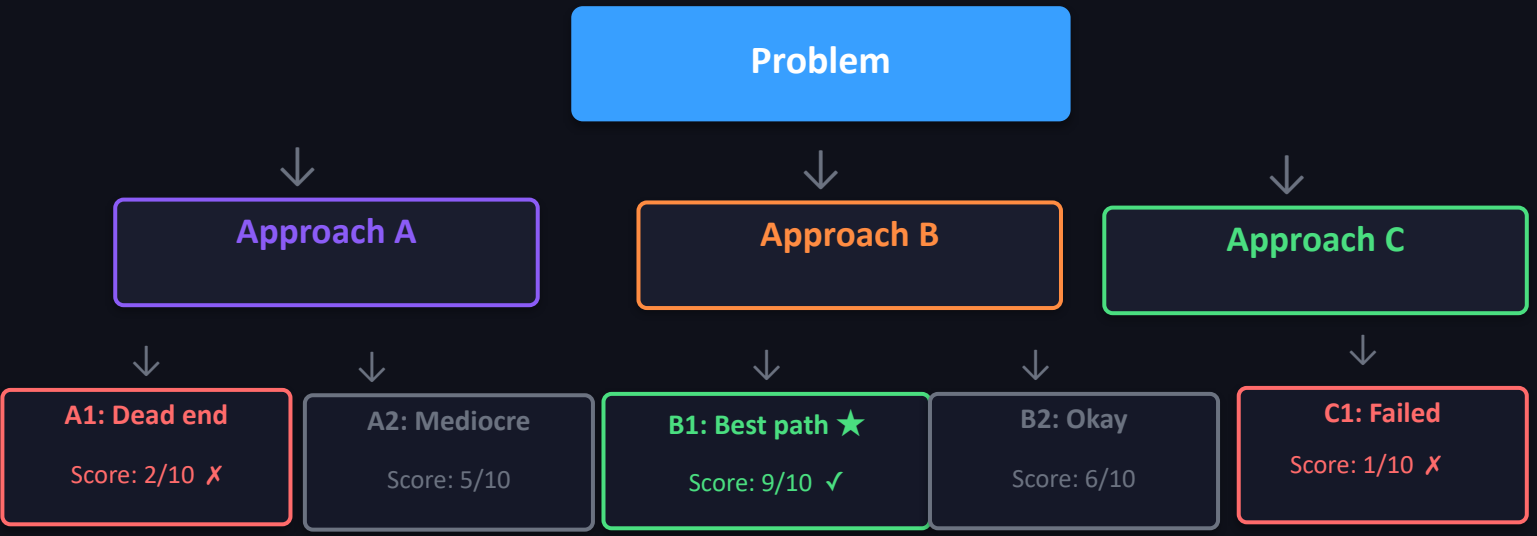
Analogy: Athlete Reviewing Game Film

- After each game, review what went wrong
- Write down lessons: "I should have passed, not shot"
- Next game, adjust strategy using those lessons

Planning Strategy: Tree of Thoughts (ToT)

Explore multiple reasoning paths, evaluate each, and pick the best

Visual: Branching Thought Paths



→ Selected: Approach B, Path B1 (Score 9/10)

How It Works

- Generate multiple candidate reasoning paths
- Evaluate each path with a scoring heuristic
- Prune bad paths, expand promising ones
- Select the highest-scoring complete solution

Analogy: A Chess Grandmaster

- Considers several opening moves mentally
- Plays each forward a few moves in their head
- Picks the move with the strongest position

Best For:

Creative tasks, puzzle-solving, code generation, tasks with many valid approaches

Multi-Agent Systems

When one agent isn't enough — teams of specialists

Analogy: A Software Engineering Team

Just like a real team has a PM, designers, frontend & backend devs, and QA — multi-agent systems have specialized agents that collaborate, delegate, and review each other's work.

Orchestrator

Assigns tasks, coordinates agents, merges outputs

Researcher

Searches the web, finds information, summarizes docs

Coder

Writes code, fixes bugs, runs tests

Reviewer

Reviews outputs, catches errors, ensures quality

Writer

Drafts reports, formats output, communicates results

Frameworks: CrewAI • AutoGen (Microsoft) • LangGraph • OpenAI Swarm • Anthropic's Multi-Agent SDK

07

Frameworks & Tools

The open-source ecosystem for building agents

LangChain / LangGraph

Most popular agent framework. Chains, tools, memory, and graph-based orchestration.

Python, JS

CrewAI

Multi-agent framework with role-based agent teams. Easy to define & orchestrate.

Python

AutoGen (Microsoft)

Multi-agent conversations. Agents chat with each other to solve complex tasks.

Python

OpenAI Agents SDK

Official OpenAI framework. Built-in tools, handoffs, guardrails & tracing.

Python

Anthropic Agent SDK

Claude-native agent toolkit. Tool use, multi-agent, client & server implementations.

Python, TS

Hugging Face smolagents

Lightweight agent framework. Code-based actions, built on Transformers ecosystem.

Python

Build Your First Agent — in ~20 Lines

Using the OpenAI Agents SDK

```python
from agents import Agent, Runner, function_tool

@function_tool
def get_weather(city: str) -> str:
    """Get weather for a city."""
    return f"The weather in {city} is sunny, 22°C"

agent = Agent(
    name="Weather Assistant",
    instructions="You help users check the weather.",
    tools=[get_weather],
)

result = Runner.run_sync(agent, "What's the weather in Paris?")
print(result.final_output)
# → "It's sunny and 22°C in Paris right now!"
```

1. Define Tools

Decorate Python functions as tools the agent can call

2. Create Agent

Give it a name, instructions, and available tools

3. Run It

Pass a user message and the agent handles the rest

That's it!

The framework handles the agent loop, tool calling, and response generation

Challenges & Limitations

What to watch out for when building agents

Hallucinations

Agents can confidently produce wrong information. Mitigation: grounding with RAG, tool verification, human-in-the-loop.

Cost & Latency

Multiple LLM calls + tool calls = expensive & slow. Mitigation: caching, smaller models for sub-tasks, efficient prompting.

Reliability & Looping

Agents may get stuck in loops or fail silently. Mitigation: max-iteration limits, fallback strategies, observability.

Security Risks

Prompt injection, tool misuse, unintended actions. Mitigation: sandboxing, guardrails, permission systems, input validation.

Evaluation

Hard to measure if an agent is "good enough". Mitigation: benchmark suites, human eval, automated test cases.

Context Window Limits

Long tasks can exceed context window capacity. Mitigation: summarization, memory management, chunking strategies.

08 Career Opportunities

The job market for agent builders is exploding

AI/ML Engineer

~ $120K-$250K+

Build & optimize agent systems. Design tool integrations & orchestration pipelines.

Prompt Engineer

~ $80K-$180K+

Craft agent instructions & system prompts. Design reliable reasoning chains.

AI Agent Developer

~ $130K-$280K+

Specialize in autonomous systems. Multi-agent design, evaluation & deployment.

AI Solutions Architect

~ $150K-$300K+

Design end-to-end AI agent solutions for enterprises. Integrate with existing systems.

Key Skills: Python • LLM APIs • Prompt Engineering • Vector Databases • System Design • Evaluation & Testing

Your Action Plan: Getting Started

A concrete roadmap to become an agent builder

1 Week 1-2

Learn the Basics

Get comfortable with LLM APIs (OpenAI, Anthropic).
Understand prompting, tokens, and tool use.

2 Week 3-4

Build Simple Agents

Use LangChain or OpenAI Agents SDK.
Create a ReAct agent with 2-3 tools.
Add memory with a vector DB.

3 Month 2

Multi-Agent Projects

Build a CrewAI team or LangGraph workflow.
Tackle a real project:
blog writer, code reviewer, etc.

4 Month 3+

Go Advanced

Add evaluation, observability (LangSmith), deploy to production.
Contribute to open-source.

Essential Resources

- **DeepLearning.AI**
  Free courses on agents, LangChain, and multi-agent systems
- **LangChain Academy**
  Official LangGraph & agent orchestration courses
- **Anthropic Cookbook**
  Hands-on examples for building with Claude
- **OpenAI Documentation**
  Agent SDK guides, best practices, and examples

Thank You!

Questions & Discussion

Key Takeaways

- Agents = LLM + Tools + Memory + Planning
- Start simple (ReAct pattern), then layer complexity
- The ecosystem is mature — frameworks make it accessible
- Huge career opportunity — the field is just getting started