

AI Agents (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688P, Spring Semester 2026

October 10, 2026

Overview

- 1 Introduction to AI Agents
 - Definition, Framework, Strengths/Limitations
- 2 Basic Workflow Patterns
 - Sequential, Parallel, Loop, ...
- 3 Agent/LLM Workflow Patterns
 - Single-Shot, Chaining, Routing, Orchestrator, ...
- 4 Agent Design Patterns
 - Reason and Act, Reflection, Tool Use, Planning, ...
- 5 Multi-Agent Applications
 - Travel Planning, Shopping Mall, Scheduling System
- 6 Appendix A: Cheatsheets
- 7 Appendix B: Software Considerations

AI Agents

AI Agent

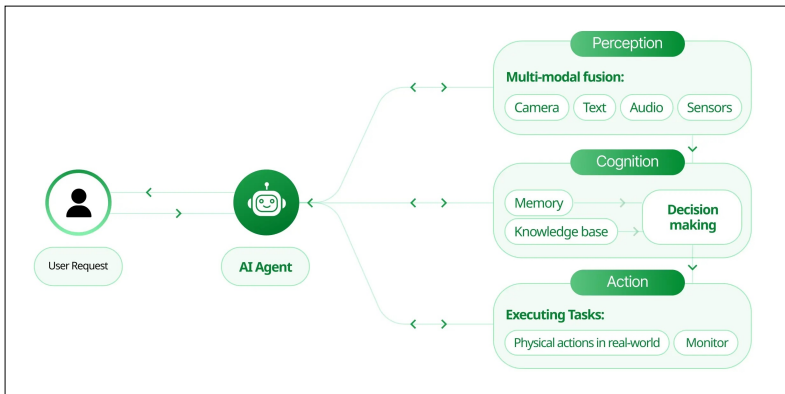
A **system or program** that can independently **plan** its **workflow** and **utilize resources** to carry out **tasks** on behalf of a user or another system.

AI agents can perform various functions, including decision-making, problem-solving, interacting with external environments, and executing actions.

A **single-task worker** designed to **automate one repetitive job** with **speed** and **reliability**.

Definition

AI Agent Concepts:



Definition

Perception:

- ...

Cognition:

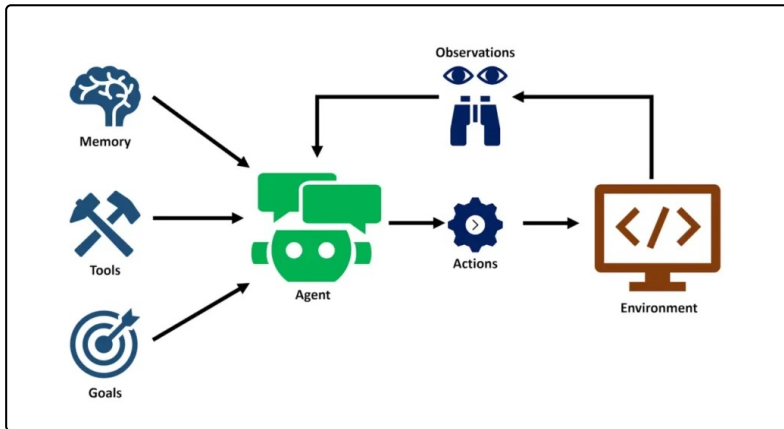
- • •

Action:

- ■ ■

Definition

AI Agent Components:



Memory:

- ...

• • •

- ...

● ...

- • •

Definition

Observations:

- ...

Actions:

- • •

Environment:

- ...

AI Agent Guidelines

AI agents are **appropriate** when:

- Problems are open-ended ...
- Need for flexibility.
- Model decision making is needed at scale.
- Performance is valued over cost and latency.

AI agents are **not appropriate** when:

- Problem (or task) is well defined.
- When simpler solutions (an algorithm) will work.
- Strict cost and latency requirements apply.
- High risk of compounding errors.

Source: Rao P., Building Effective Agents.

AI Agent Construction

Agent Building Blocks

Agent **building blocks** are the **core foundation** of an **agent**, usually an LLM enhanced with augmentations such as retrieval, tools, and memory.

Agent Assembly

The most **successful imlementations** do not use complex frameworks or specialized libraries – instead, they **build** with **simple composable patterns**.

Types of AI Agents

Simple Rule-based Agents:

- ...

Learning Agents:

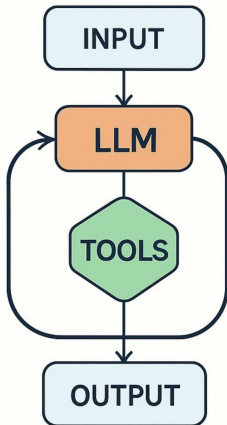
- ...

Goal-based Agents:

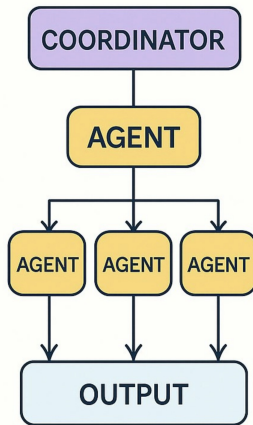
- ...

Single- and Multi-Agent Architectures

Single-Agent



Multi-Agent



Single- and Multi-Agent Architectures

Single-Agent System

A **single-agent system** employs an AI model, a predefined set of tools, and a **comprehensive system prompt** to autonomously handle a user request or to complete a specific task.

Simplified Architecture:



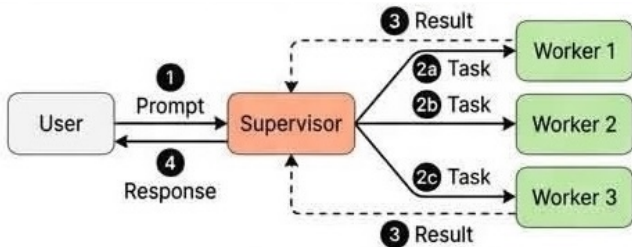
The agent relies on the model's reasoning capabilities to interpret a request, plan a sequence of steps, and decide which tools to use from a defined set.

Single- and Multi-Agent Architectures

Multi-Agent System

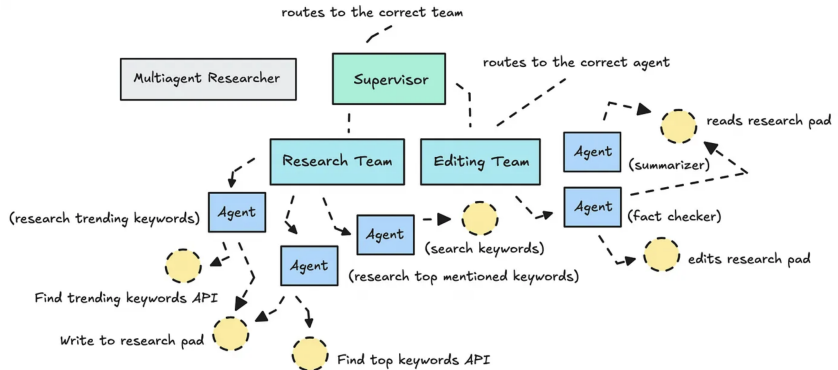
A multi-agent system comprises **multiple intelligent autonomous agents** that **interact** in a **shared environment** to solve a **complex task** that cannot be easily handled by a single agent.

Simplified Architecture:



Single- and Multi-Agent Architectures

Multi-Agent System Example:



AI Agent Benefits and Limitations

Benefits:

- Automates repetitive time-consuming tasks.
- Reliable in predictable environments.
- Easy to test and refine within specific rules.

Limitations:

- Struggle with ambiguity or new situations.
- Fragile if inputs are unclear or data is missing.
- Requires human oversight for complex decision making and/or for agent to work alongside other agents.

Remark: The inability of AI agents to adapt in real-time to differing circumstances spawned Agentic AI.

Basic Workflow Patterns

Workflows

Workflows are where the LLM follows a predefined, hardcoded, step-by-step procedure.

Core Foundational Patterns:

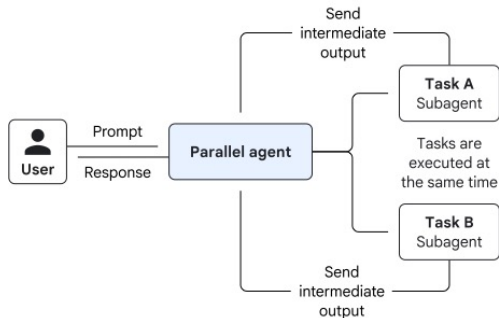
- Sequential, Parallel, and Loop Patterns.
- Review and Critique Pattern.
- Iterative Refinement Pattern.
- Coordinator Pattern.
- Hierarchical Task Decomposition.
- Swarm Pattern.

Basic Workflow Patterns

Sequential Pattern

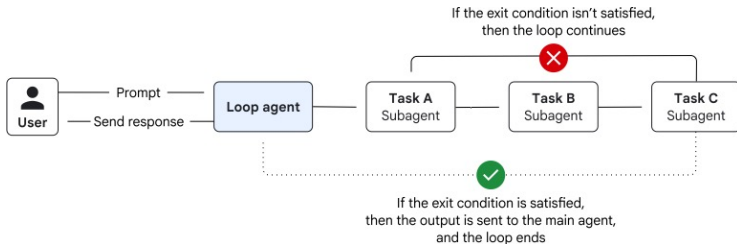


Parallel Pattern

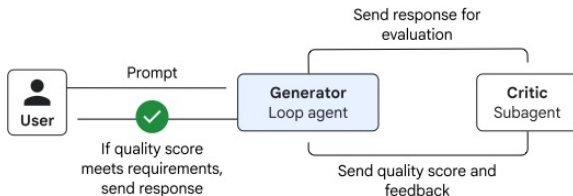


Basic Workflow Patterns

Loop Pattern

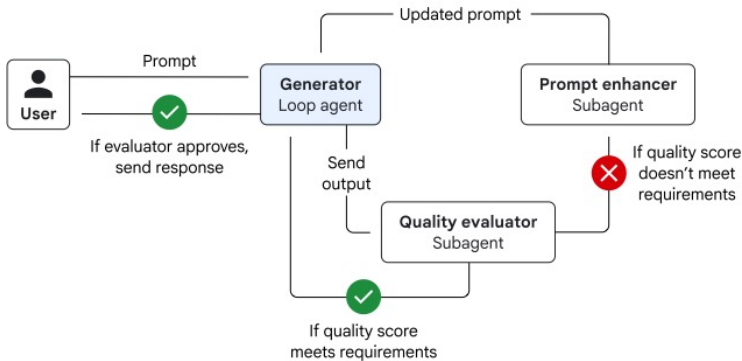


Review and Critique Pattern



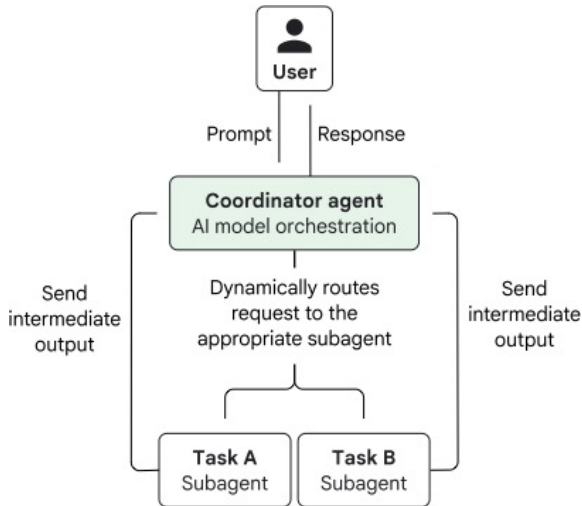
Basic Workflow Patterns

Iterative Refinement Pattern



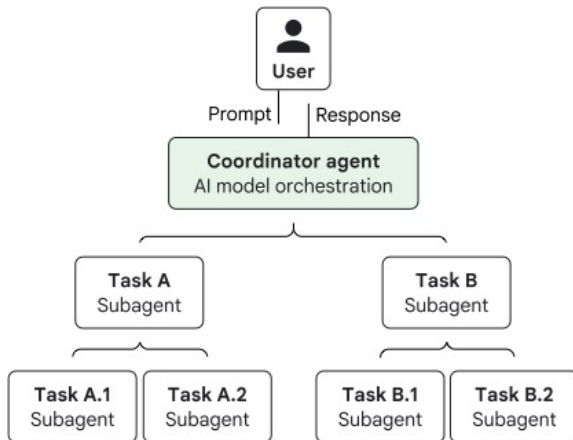
Basic Workflow Patterns

Coordinator Pattern



Basic Workflow Patterns

Task Decomposition

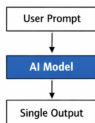


Agent/LLM

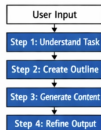
Workflow Patterns

AI Workflow Patterns

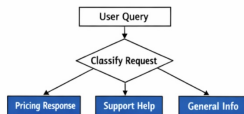
1. Single-Shot Pattern



2. Chaining Pattern



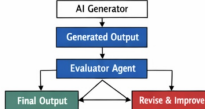
3. Routing Pattern



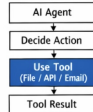
4. Orchestrator Pattern



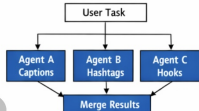
5. Evaluator Pattern



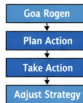
6. Tools Pattern



7. Parallel Pattern



8. Autonomous Pattern



AI Workflow Patterns

Single-Shot Workflow:

- A [single-shot workflow](#) has one prompt and one response. It's simple and direct.

Chaining Workflow:

- A [chaining workflow](#) breaks down complex tasks into sequential steps.

Routing Workflow:

- The [routing workflow](#) intelligently directs requests to the right destination.

Orchestrator Workflow:

- An [orchestrator workflow](#) coordinates multiple AI agent to solve complex tasks.

AI Workflow Patterns

Evaluator Workflow:

- The **evaluator workflow** generates, evaluates, and refines for higher quality output.

Tools Use Workflow:

- The **tools use** workflow lets AI accomplish tasks by loading and executing external tools.

Parallel Workflow:

- **Parallel workflow** runs multiple tasks simultaneously for speed and scale.

Choose the Right Pattern

Guidelines:

- Use simple workflows for quick and direct AI tasks.
- Combine multiple steps when tasks require deeper reasoning.
- Use tools and agent to automate complex real-world actions.
- Choose the autonomous systems pattern for adaptive self-managed workflows.

Agent Design Patterns

Agent Design Patterns

Agent Design Patterns

Agent design patterns are reusable architectural frameworks used to structure how AI agents perceive their environment, reason, use tools, and collaborate to achieve a well-defined goal.

Core Foundational Patterns:

- ReAct (Reason and Act) Pattern.
- Reflection (Self-Critique) Pattern.
- Planning and Tool Use Patterns.

Multi-Agent and Orchestration Patterns:

- Multi-Agent (Supervisor) Pattern.
- Swarm and Human-in-the-Loop Patterns.

Agent Design Patterns

Core Objectives:

- Reduce errors and wrong tool calls.
- Enable systems to make dynamic, multi-step decisions.
- Handle large or cross-functional workflows using modular components.
- Enforce human oversight and prevent runaway loops or budget overruns.

Foundational Objectives:

- Tool Use – Connect the model to external APIs, databases and/or code interpreters to perform real-world actions.
- Reflection – Allows an agent to inspect, critique and revise its own output to improve quality.

Agent Design Patterns

Foundational Objectives: continued ...

- Planning – Break a large goal down into structured, ordered subtasks with clear dependencies before execution.
- ReAct (Reason and Act) – Interleaves chain-of-thought reasoning with tool observation in a continuous feedback loop.

Production Objectives:

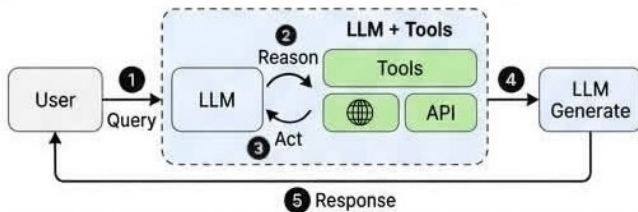
- Observability – log every plan, tool call, and decision for easy debugging.
- Cost control – Cap the number of total iterations and spend per user request.
- Safety gates – Keep irreversible actions behind human approval or deterministic checks.

Reason and Act Design Pattern

Reason and Act Design Pattern

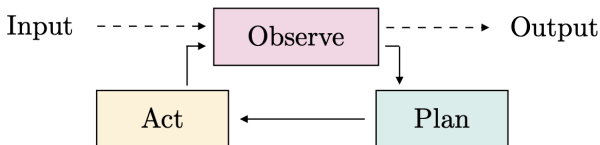
Alternates between reasoning and taking actions until a task is complete.

Framework:



Reason and Act Design Pattern

ReAct (Reason+Act): Chains of LLM complete complex tasks.



The framework:

- Observe: synthesize past actions and explicitly determine what is currently known.
- Plan: determine that tasks need to be accomplished.
- Action: perform an action via an API, or look for relevant information in the system knowledge base.
- Repeat: As needed.

Reason and Act Design Pattern

ReAct (Reason+Act): Example ...

User: Calculate $25 * 17$, then tell me if it's a prime number

Thought 1: I need to calculate $25 * 17$

Action 1: `calculator(expression=25 * 17)`

Observation 1: 425

Thought 2: I need to check if 425 is prime

Action 2: `Evaluate is_prime(number=425)`

Observation 2: False (divisible by 5)

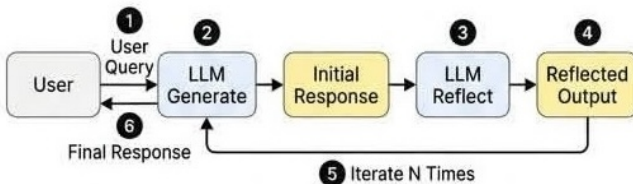
Final Answer: "25 * 17 equals 425, which is not a prime number because it's divisible by 5."

Reflection Design Pattern

Reflection Design Pattern

Output is fed back into the model so it can critique and refine its own response.

Framework:



Reflection Design Pattern

Key Points:

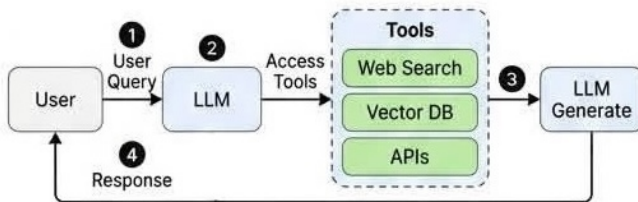
- ...
- ...
- ...
- ...
- ...

Tool Use Design Pattern

Tool Use Design Pattern

The model decides which tools to call, retrieves the results, and assembles the answer.

Framework:



Tool Use Design Pattern

Key Points:

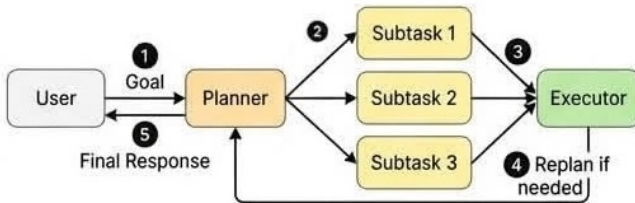
- ...
- ...
- ...
- ...
- ...

Planning Pattern

Planning Design Pattern

A planning design pattern breaks the goal into subtasks, executes them as required, and adjusts the plan when obstacles are encountered.

Framework:



Planning Pattern

Key Points: The planning pattern is useful when:

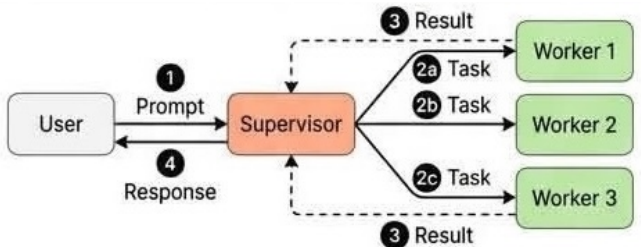
- The objective requires multiple actions.
- Actions have dependencies or a specific order.
- The agent needs to make decisions before acting.
- There are constraints on time, budget, resources, or rules.
- A task can be divided into subtasks.
- The environment can change during execution.
- The agent needs to adapt or replan when obstacles appear.
- The goal is to make the system more autonomous, rather than requiring step-by-step instructions from the user.

Multi-Agent (Supervisor) Pattern

Multi-Agent Design Pattern

A supervisor agent routes tasks to specialized worker agents and compiles the final response.

Framework:



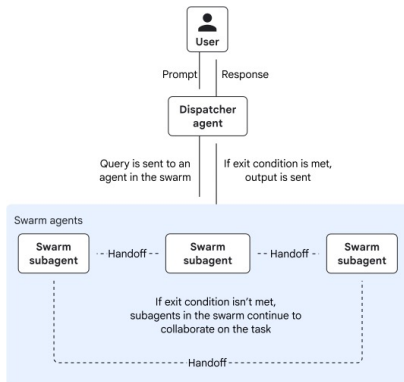
Key Points:

- [illegible]

Swarm Pattern

Swarm Pattern/Group Chat

Loosely coupled agents engage in a shared discussion to handle ambiguous or complex problems through consensus.



Swarm Pattern

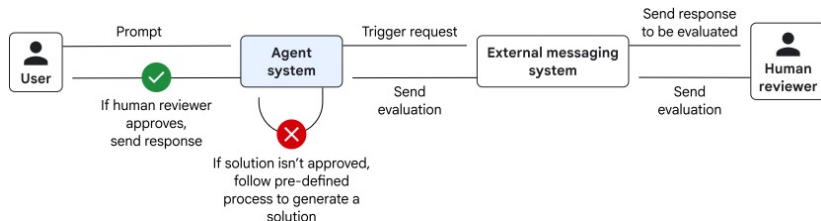
Key Points:

- ...
- ...
- ...
- ...
- ...

Human-in-the-Loop Pattern

Human-in-the-Loop Pattern

A **cross-cutting safety gate** that requires human approval before executing risky action, tool calls, or final outputs.



Human-in-the-Loop Pattern

Key Points:

- ...
- ...
- ...
- ...
- ...

Multi-Agent Travel Planning System

Travel Planning System

Travel Planning System

Use [multi-agent architecture](#) to implement a [travel planning assistant](#). Individual agents are designed for a specific aspect of travel planning.

Architecture:

Travel Planning System

Step-by-step Procedure:

- Gather data sources – weather, hotel options, etc.
- Provide destination, preferences and duration.
- **Weather agent** predicts the best time to visit.
- **Hotel agent** finds matching accommodation.
- **Itinerary agent** creates daily plans.
- **Summary agent** calculates costs and generates an e-mail.

Source: ...

Travel Planning System

Weather Data:

```
[{'latitude': 41.9028, 'longitude': 12.4964, 'month': 1, 'weather_score': 0.4},  
 {'latitude': 41.9028, 'longitude': 12.4964, 'month': 2, 'weather_score': 0.6},  
  
 ... lines of output removed ...  
  
 {'latitude': 41.9028, 'longitude': 12.4964, 'month': 11, 'weather_score': 0.6},  
 {'latitude': 41.9028, 'longitude': 12.4964, 'month': 12, 'weather_score': 0.4}]
```

Hotel Accommodation Data:

```
[{'description': 'Luxury hotel in city center with pool, spa and restaurant',  
  'name': 'Grand Hotel',  
  'price': 300},  
  
 ... lines of output removed ...  
  
 {'description': 'Modern hotel with panoramic city views',  
  'name': 'City View Hotel',  
  'price': 200} ]
```

Travel Planning System

Weather Analysis Agent

- Analyze historical weather data to predict optimal travel periods.

Python Code:

```

1  # -----
2  # WeatherAnalysisAgent: Analyze historical weather data (random forest model) to
3  # predict optimal travel periods.
4  #
5  # Note: n_estimators = 100 <--- model consists of 100 decision trees.
6  # -----
7
8  class WeatherAnalysisAgent:
9      """
10     An agent that analyzes historical weather data to predict optimal travel periods.
11
12     Attributes:
13         model (RandomForestRegressor): A random forest model trained on historical
14         weather data.
15     """
16
17     def __init__(self):

```

Travel Planning System

Python Code: continued ...

```

18         # Uses RandomForest for weather predictions based on historical data
19         print("--- Enter WeatherAnalysisAgent.__init__(self) ...");
20         self.model = RandomForestRegressor(n_estimators=100)
21         print("--- Leave WeatherAnalysisAgent.__init__(self) ...");
22
23     def train(self, historical_data: Dict):
24         # Training on historical weather data
25         X = np.array([[d['month'], d['latitude'], d['longitude']] for d in historical_data])
26         y = np.array([d['weather_score'] for d in historical_data])
27         # Fit model to (X, y) data ...
28         self.model.fit(X, y)
29
30     def predict_best_time(self, location: Dict) -> Dict:
31         # Predicts the best time to visit a location based on weather patterns
32         predictions = []
33         for month in range(1, 13):
34             # predict returns a 2D array, we take the first (and only) element
35             prediction = self.model.predict([[ month, location['latitude'], location['longitude']]
36             ]).item() # .item() converts numpy array to scalar
37             predictions.append({'month': month, 'score': float(prediction)})
38
39         return {
40             'best_months': sorted(predictions, key=lambda x: x['score'], reverse=True)[0:3],
41             'location': location
42         }

```

Travel Planning System

Hotel Recommender Agent

- Use semantic (transformer) embeddings to recommend hotels that best match user and hotel descriptions.

Python Code:

```

1  # -----
2  # HotelRecommenderAgent: Use semantic embeddings to recommend hotels that
3  # best match user preferences.
4  # -----
5
6  class HotelRecommenderAgent:
7      """
8      An agent for recommending hotels that best match user preferences using
9      semantic embeddings.
10
11     Attributes:
12         encoder (SentenceTransformer): Sentence embedding model for semantic matching.
13         hotels_db (List[Dict]): Database of hotel information.
14         hotels_embeddings (np.ndarray): Precomputed embeddings of hotel descriptions.
15     """

```

Travel Planning System

```

16
17 def __init__(self):
18     # Uses SentenceTransformer for hotel description embeddings
19     print("--- Enter HotelRecommenderAgent.__init__(self) ...");
20     self.encoder = SentenceTransformer('all-MiniLM-L6-v2')
21     self.hotels_db = []
22     self.hotels_embeddings = None
23     print("--- Leave HotelRecommenderAgent.__init__(self) ...");
24
25 def add_hotels(self, hotels: List[Dict]):
26     # Updates the hotel database and computes embeddings
27     self.hotels_db = hotels
28     descriptions = [h['description'] for h in hotels]
29     self.hotels_embeddings = self.encoder.encode(descriptions)
30
31 def find_hotels(self, preferences: str, top_k: int = 5) -> List[Dict]:
32     # Finds hotels most similar to preferences using semantic similarity
33     pref_embedding = self.encoder.encode([preferences])
34     similarities = np.dot(self.hotels_embeddings, pref_embedding.T).flatten()
35
36     top_indices = similarities.argsort()[-top_k:][::-1]
37     return [
38         {**self.hotels_db[i], 'similarity_score': float(similarities[i])}
39         for i in top_indices
40     ]

```

Travel Planning System

Itinerary Planner Agent

- Use GPT-2 (text generation pipeline) to generate personalized travel itineraries based on provided prompts.

Python Code:

```

1  # -----
2  # ItineraryPlannerAgent: Create travel itineraries by generating descriptive
3  # text using GPT-2.
4  # -----
5
6  class ItineraryPlannerAgent:
7      """
8          An agent that creates travel itineraries by generating descriptive text using GPT-2.
9
10         Attributes:
11             planner (pipeline): Text-generation pipeline from Hugging Face transformers.
12         """
13         def __init__(self):
14             # Uses a language model for generating itineraries

```

Travel Planning System

```

16     print("--- Enter ItineraryPlannerAgent.__init__(self) ...");
17     print("--- Execute: self.planner = pipeline(...) ...");
18
19     # In production, use a more powerful model
20
21     self.planner = pipeline( "text-generation", model="gpt2",
22                             max_new_tokens=50,
23                             truncation=True, pad_token_id=50256 )
24
25     print("--- Leave ItineraryPlannerAgent.__init__(self) ...");
26
27     def create_itinerary(self, destination_info: Dict, weather_info: Dict,
28                         hotel_info: Dict, duration: int) -> Dict:
29
30         print("--- ");
31         print("--- Enter ItineraryPlannerAgent.create_itinerary() ... ");
32         print("--- ----- ... ");
33
34         print("--- ");
35         print("---     Part 01: Create prompt ... ");
36
37         prompt = self._create_prompt(destination_info, weather_info, hotel_info, duration)
38         pprint.pprint(prompt)
39
40         # Generate the itinerary
41
42         print("--- ");
43         print("---     Part 02: Generate itinerary: response = self.planner(prompt)[0]['g

```

Travel Planning System

```

45     response = self.planner(prompt)[0]['generated_text']
46     pprint.pprint(response)
47
48     print("--- ----- ... ");
49     print("--- Leave ItineraryPlannerAgent.create_itinerary() ... ");
50     print("--- ");
51
52     return { 'itinerary': response, 'duration': duration,
53             'destination': destination_info['name'] }
54
55     def _create_prompt(self, destination_info: Dict, weather_info: Dict,
56                       hotel_info: Dict, duration: int) -> str:
57         return f"""Create a {duration}-day itinerary for {destination_info['name']}.
58 Weather: {weather_info['best_months'][0]['month']} is the best month.
59 Hotel: Staying at {hotel_info[0]['name']}.
60 Attractions: {'', '.join(destination_info['attractions'])}."""

```

Travel Planning System

Summary Agent

- Estimate the total cost of the trip.
- Use GPT-2 to generate personalized e-mail the trip details, cost breakdown, and itinerary highlights.

Python Code:

```

1  # -----
2  # SummaryAgent
3  # -----
4
5  class SummaryAgent:
6      """
7      An agent that generates summarized travel plans in the form of personalized emails.
8
9      Attributes:
10         llm (pipeline): Text-generation pipeline from Hugging Face transformers for email
11         """
12     def __init__(self):
13         print("--- Enter SummaryAgent.__init__(self) ...");
14         self.llm = pipeline( "text-generation", model="gpt2",
15                             max_new_tokens=100, truncation=True,
16                             pad_token_id=50256 )
17         print("--- Leave SummaryAgent.__init__(self) ...");

```

Travel Planning System

```
18
19 def calculate_total_price(self, hotel_info: Dict, duration: int) -> float:
20
21     # Calculate total trip price
22
23     hotel_cost = hotel_info[0]['price'] * duration
24
25     # Estimate additional costs (activities, meals, transport)
26
27     daily_expenses = 100 # Simplified example
28     additional_costs = daily_expenses * duration
29
30     return hotel_cost + additional_costs
31
32 def create_email(self, trip_data: Dict, client_name: str) -> Dict:
33
34     total_price = self.calculate_total_price(
35         trip_data['recommended_hotels'],
36         trip_data['itinerary']['duration']
37     )
38
39     prompt = f"""
40     Dear {client_name},
41
42     Based on your preferences, I'm pleased to present your travel plan:
43
44     Destination: {trip_data['itinerary']['destination']}
45     Duration: {trip_data['itinerary']['duration']} days
46     Best time to visit: Month {trip_data['weather_analysis']['best_months'][0]['month']}
```

Travel Planning System

```

46         Best time to visit: Month {trip_data['weather_analysis']['best_months'][0]['mont
47
48         Recommended Hotel: {trip_data['recommended_hotels'][0]['name']}
49
50         Itinerary Overview:
51         {trip_data['itinerary']['itinerary']}
52
53         Estimated Total Cost: \${total_price}
54
55         Please let me know if you would like any adjustments.
56         """
57
58         # Generate email using LLM
59
60         response = self.llm(prompt)[0]['generated_text']
61
62         return {
63             'email_content': response,
64             'total_price': total_price,
65             'summary_data': {
66                 'destination': trip_data['itinerary']['destination'],

```

Travel Planning System

Python Code:

```

1  # -----
2  # TravelPlanningSystem
3  # -----
4
5  class TravelPlanningSystem:
6      def __init__(self):
7          print("--- Enter TravelPlanningSystem.__init__(self) ...");
8          self.weather_agent = WeatherAnalysisAgent()
9          self.hotel_agent = HotelRecommenderAgent()
10         self.itinerary_agent = ItineraryPlannerAgent()
11         self.summary_agent = SummaryAgent()
12         print("--- Leave TravelPlanningSystem.__init__(self) ...");
13
14         def setup(self, historical_weather_data: Dict, hotels_database: List[Dict]):
15
16             # Initialize and train the models
17             self.weather_agent.train(historical_weather_data)
18             self.hotel_agent.add_hotels(hotels_database)
19
20         def plan_trip(self, destination: Dict, preferences: str, duration: int, client_name: str):
21
22             print("--- ");
23             print("--- Enter TravelPlanningSystem.plan_trip() ...");
24
25             # 1. Weather analysis and best time prediction

```

Travel Planning System

```

27         print("--- ");
28         print("---      Part 1: Weather Analysis .plan_trip() ...");
29         print("--- ");
30
31         weather_analysis = self.weather_agent.predict_best_time(destination)
32         pprint.pprint( weather_analysis, width=80 )
33
34         # 2. Hotel search
35
36         print("--- ");
37         print("---      Part 2: Hotel Search ...");
38         print("--- ");
39
40         recommended_hotels = self.hotel_agent.find_hotels(preferences)
41         pprint.pprint( recommended_hotels, width = 80 )
42
43         # 3. Itinerary creation
44
45         print("--- ");
46         print("---      Part 3: Itinerary Creation ...");
47         print("--- ");
48
49         itinerary = self.itinerary_agent.create_itinerary( destination, weather_analysis
50         pprint.pprint( itinerary, width = 80 )
51
52         # 4. Create summary email and calculate price

```

Travel Planning System

```
53
54     print("--- ");
55     print("---      Part 4: Create Summary and Calculate Price ...");
56     print("---      ----- ...");
57     print("--- ");
58
59     trip_data = { 'weather_analysis': weather_analysis,
60                  'recommended_hotels': recommended_hotels,
61                  'itinerary': itinerary }
62
63     summary = self.summary_agent.create_email(trip_data, client_name)
64
65     print("--- Leave TravelPlanningSystem.plan_trip() ...");
66
67     return { **trip_data, 'summary': summary }
```

Travel Planning System: main() ...

```
1  def main():
2      print("--- Enter Test-Travel-Planning-System01.main()      ... ");
3      print("--- ===== ... ");
4
5      # Example data with a full year of weather information
6
7      print("--- ");
8      print("--- Part 01: List of historical weather data ...");
9      print("--- ----- ...");
10
11     historical_weather_data = [
12         {'month': 1, 'latitude': 41.9028, 'longitude': 12.4964, 'weather_score': 0.4},
13         {'month': 2, 'latitude': 41.9028, 'longitude': 12.4964, 'weather_score': 0.6},
14
15         ... lines of data removed ...
16
17         {'month': 11, 'latitude': 41.9028, 'longitude': 12.4964, 'weather_score': 0.6},
18         {'month': 12, 'latitude': 41.9028, 'longitude': 12.4964, 'weather_score': 0.4},
19     ]
20
21     pprint.pprint( historical_weather_data, width=80 )
22
23     # Sample hotel database
24
25     print("--- ");
26     print("--- Part 02: Sample hotel database ... ");
27     print("--- ----- ...");
28
29     hotels_database = [
```

Travel Planning System: main() ...

```
30         { 'name': 'Grand Hotel',
31           'description': 'Luxury hotel in city center with pool, spa and restaurant',
32           'price': 300
33         }, { 'name': 'Novotel',
34             'description': 'Luxury hotel in city center with spa and conference center',
35             'price': 150
36         }, { 'name': 'Boutique Resort',
37             'description': 'Intimate boutique hotel with personalized service',
38             'price': 250
39         }, { 'name': 'City View Hotel',
40             'description': 'Modern hotel with panoramic city views',
41             'price': 200
42         } ]
43
44 pprint.pprint( hotels_database, width=80 )
45
46 # Initialize the system
47
48 print("--- ");
49 print("--- Part 03: Initialize Travel Planning System      ...");
50 print("--- -----");
51
52 print("--- Instantiate TravelPlanningSystem() ...");
53
54 system01 = TravelPlanningSystem()
55 system01.setup(historical_weather_data, hotels_database)
56
57 print("--- ");
58 print("--- Part 04: Plan a trip                                ...");
```

Travel Planning System: main() ...

```

59     print("--- ----- ...");
60
61     destination = { 'name': 'Rome',
62                     'latitude': 41.9028,
63                     'longitude': 12.4964,
64                     'attractions': ['Colosseum', 'Vatican', 'Trevi Fountain'] }
65
66     pprint.pprint( destination, width=80 )
67
68     # Setup preferences and client name ...
69
70     print("--- ");
71     print("--- Part 05: Set up preferences and client name      ...");
72     print("--- ----- ...");
73
74     preferences = """Looking for a luxury hotel in the city center,
75                  preferably with spa facilities and fine dining options"""
76
77     client_name = "John Smith"
78
79     print("--- Preferences: {:s} ...".format(preferences));
80     print("--- Client name: {:s} ...".format(client_name));
81
82     # Generate trip plan
83
84     print("--- ");
85     print("--- Part 06: Generate trip plan ... ");
86     print("--- ----- ...");

```

Travel Planning System: main() ...

```

88     trip_plan = system01.plan_trip( destination, preferences,
89                                     duration=3, client_name=client_name)
90
91     # Print results in a readable format
92
93     print("--- ");
94     print("---- Print results in a readable format ...");
95
96     print("--- ")
97     print("---- Part 07: Travel Planning Results:")
98     print("---- -----")
99     print(f"---- Client: {client_name}")
100    print(f"---- Destination: {destination['name']}")
101    print("---- ")
102    print("---- Generated Email:")
103    print("-" * 20)
104    print(trip_plan['summary']['email_content'])
105    print("---- ")
106    print("---- Estimated Total Price:")
107    print(f"\${trip_plan['summary']['total_price']}")
108
109    print("---- ===== ... ");
110    print("---- Leave Test-Travel-Planning-System01.main() ... ");
111
112    # call the main method ...
113
114    if __name__ == "__main__":
115        main()

```

Travel Planning System Output

Program Output: (abbreviated) ...

```

--- Enter Test-Travel-Planning-System01.main()      ...
--- =====                                         ...

--- Part 01: List of historical weather data (lines of output removed) ...
--- Part 02: Sample hotel database (lines of output removed) ...
--- Part 03: Initialize Travel Planning System      ...
--- Part 04: Plan a trip                            ...

{'attractions': ['Colosseum', 'Vatican', 'Trevi Fountain'],
 'latitude': 41.9028, 'longitude': 12.4964, 'name': 'Rome'}

--- Part 05: Set up preferences and client name      ...
--- -----                                         ...

--- Preferences: Looking for a luxury hotel in the city center,
                    preferably with spa facilities and fine dining options ...
--- Client name: John Smith ...

--- Part 06: Generate trip plan ...
--- -----                                         ...

---
--- Enter TravelPlanningSystem.plan_trip() ...
---
--- Part 1: Weather Analysis .plan_trip() ...

```

Travel Planning System Output

```
{'best_months': [{'month': 7, 'score': 0.9330000000000014},
                  {'month': 8, 'score': 0.9070000000000001},
                  {'month': 6, 'score': 0.8875000000000007}],
 'location': {'attractions': ['Colosseum', 'Vatican', 'Trevi Fountain'],
              'latitude': 41.9028,
              'longitude': 12.4964,
              'name': 'Rome'}}
```

--- Part 2: Hotel Search ...

```
[{'description': 'Luxury hotel in city center with spa and conference center',
  'name': 'Novotel',
  'price': 150,
  'similarity_score': 0.8830808401107788},
 {'description': 'Luxury hotel in city center with pool, spa and restaurant',
  'name': 'Grand Hotel',
  'price': 300,
  'similarity_score': 0.8809388875961304},
 {'description': 'Intimate boutique hotel with personalized service',
  'name': 'Boutique Resort',
  'price': 250,
  'similarity_score': 0.5769646763801575},
 {'description': 'Modern hotel with panoramic city views',
  'name': 'City View Hotel',
  'price': 200,
  'similarity_score': 0.5353888273239136}]
```

Travel Planning System Output

```

--- Part 3: Itinerary Creation ...
---
--- Enter ItineraryPlannerAgent.create_itinerary() ...
---
--- Part 01: Create prompt ...

('Create a 3-day itinerary for Rome.\n'
 ,
   Weather: 7 is the best month.\n'
 ,
   Hotel: Staying at Novotel.\n'
 ,
   Attractions: Colosseum, Vatican, Trevi Fountain.')

--- Part 02: Generate itinerary: response = self.planner(prompt)[0]['generated_text'] ...

('Create a 3-day itinerary for Rome.\n'
 ,
   Weather: 7 is the best month.\n'
 ,
   Hotel: Staying at Novotel.\n'
 ,
   Attractions: Colosseum, Vatican, Trevi Fountain.\n'
 ,
   ')
--- Leave ItineraryPlannerAgent.create_itinerary() ...

{'destination': 'Rome',
 'duration': 3,
 'itinerary': 'Create a 3-day itinerary for Rome.\n'
 ,
   Weather: 7 is the best month.\n'
 ,
   Hotel: Staying at Novotel.\n'
 ,
   Attractions: Colosseum, Vatican, Trevi Fountain.\n'

```

Travel Planning System Output

--- Part 07: Travel Planning Results:

--- -----

--- Client: John Smith

--- Destination: Rome

--- Generated Email:

Dear John Smith,

Based on your preferences, I'm pleased to present your travel plan:

Destination: Rome

Duration: 3 days

Best time to visit: Month 7

Recommended Hotel: Novotel

Itinerary Overview:

Create a 3-day itinerary for Rome.

Weather: 7 is the best month.

Hotel: Staying at Novotel.

Attractions: Colosseum, Vatican, Trevi Fountain.

Estimated Total Cost: \$750

Please let me know if you would like any adjustments.

--- Estimated Total Price: \$750

Travel Planning System (Streamlit Implementation)

AI Travel Planner

Find the best time to travel and discover the perfect hotel!

Enter your destination (e.g., Rome):

Rome

Describe your ideal hotel:

Luxury hotel in city center with spa.

Trip duration (days):

7

Travel Planning System (Streamlit Implementation)

Generate Travel Plan ✨



Best Months to Visit ⇄

Month 1: Score 0.72

Month 5: Score 0.65

Month 4: Score 0.57



Recommended Hotel

Grand Hotel - Luxury hotel in city center with spa.



Generated Itinerary

Day 1: Arrival in Rome.

- Check in at Grand Hotel.
- Freshen up, then take a brief tour around the hotel to familiarize yourself.
- Welcome dinner at the hotel restaurant.

Day 2: Exploring the heart of Rome.

- Breakfast at the Grand Hotel restaurant.
- Guided tour of the city, visiting iconic sights such as the Colosseum, Palatine Hill, and the Roman Forum.
- Lunch at a traditional Roman Trattoria.
- Afternoon visits to the Pantheon, Piazza Navona, and Campo de' Fiori.
- Return to the hotel for a relaxing dinner and evening.

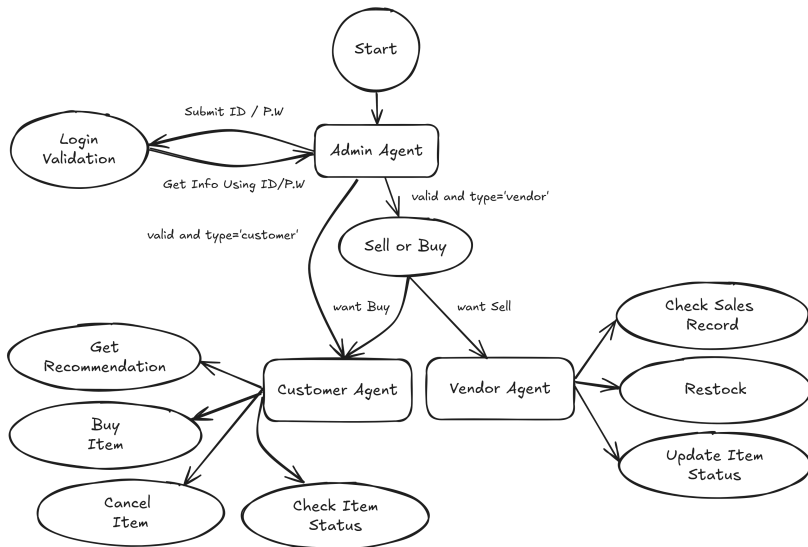
Travel Planning System (Streamlit Implementation)

Destination Map



Multi-Agent Shopping Mall System

Multi-Agent Shopping Mall System



Admin Agent

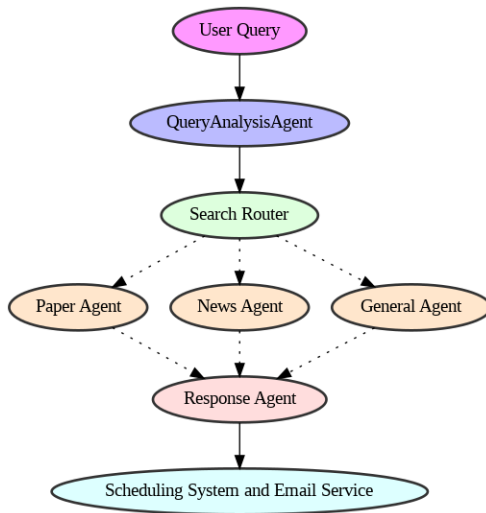
Customer Agent

Vendor Agent

Shopping Mall System Output

Multi-Agent Scheduler System

Multi-Agent Scheduler System



User Query

Query Analysis Agent

Search Router

Paper Agent

News Agent

General Agent

Response Agent

Scheduler System and E-mail Service

Scheduler System Output

References

- Amidi A., and Amidi S., Super Study Guide: Transformers and Large Language Models (First Edition), ISBN 979-8-836693-31-2, 2024.
- Guilli A., Agentic Design Patterns (Hands-on Guide to Building Intelligent Systems), Springer, 2025.
- Lakshmanan V., and Hapke H., Generative AI Design Patterns: Solutions to Common Challenges when building GenAI Agents and Applications, O'Reilly, 2026.
- Raieli S., and Luculano G., Building AI Agents with LLMs, RAG and Knowledge Graphs, Packt Publishing, 2025.

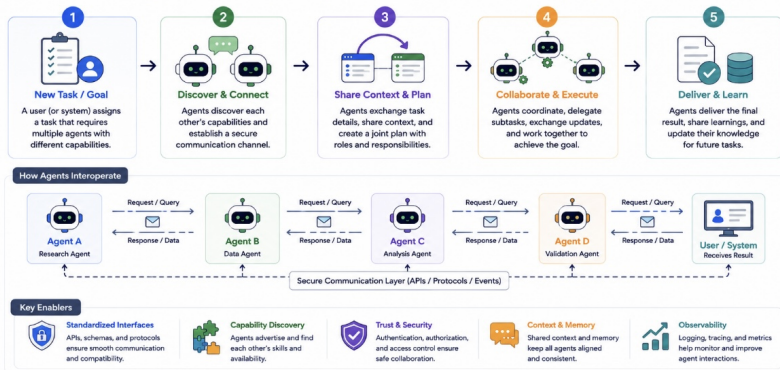
Appendix A:

Cheatsheets

Agent-to-Agent Design Pattern

Agent-to-Agent Interoperability Pattern

Agents communicate, share capabilities, and collaborate to solve complex problems.

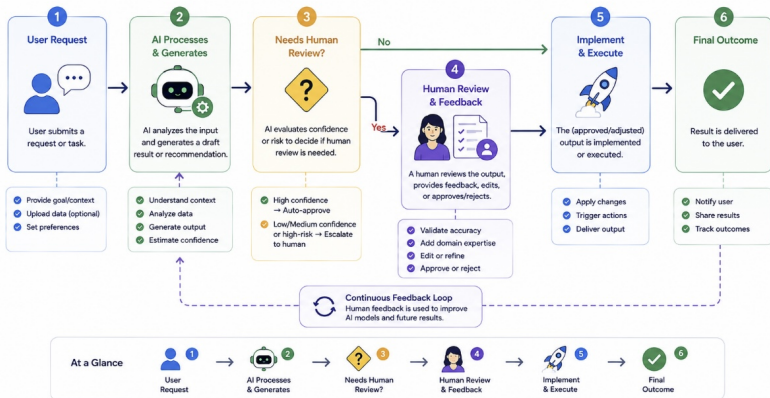


Goal: Enable agents to seamlessly communicate, collaborate, and combine their strengths to solve complex problems more effectively.

Human-in-the-Loop Design Pattern

Human-in-the-Loop Pattern

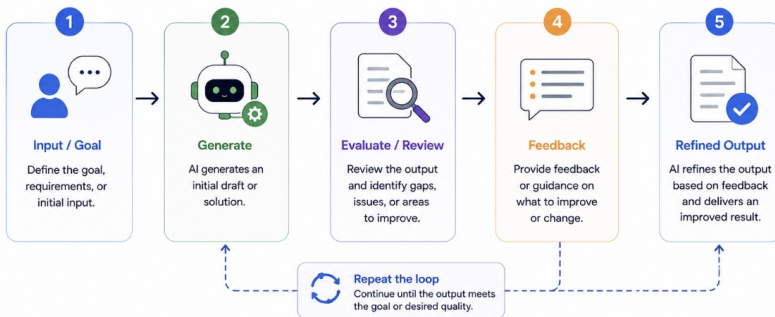
AI handles the work, humans guide, review, and make the final call.



Iterative Refinement Design Pattern

Iterative Refinement / Loop Pattern

Repeat, refine, and improve until the output is just right.



How the Loop Works

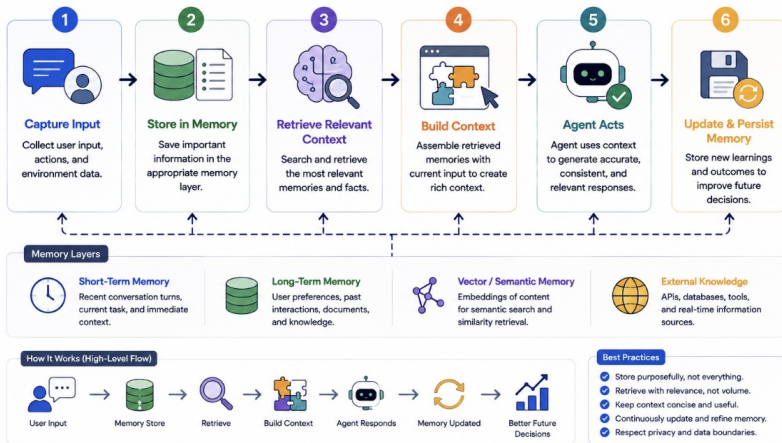


Goal: Continuous improvement through feedback and iteration leads to higher quality results.

Memory-and-Context Design Pattern

Memory & Context Engineering Pattern

Give agents the right memory, context, and recall — so they stay consistent and effective.

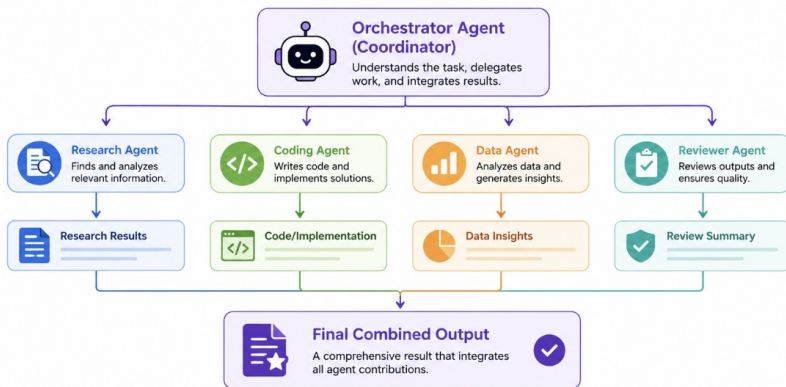


Goal: Build agents that remember what matters, understand context deeply, and get smarter over time.

Multi-Agent Design Pattern

Multi-Agent Pattern

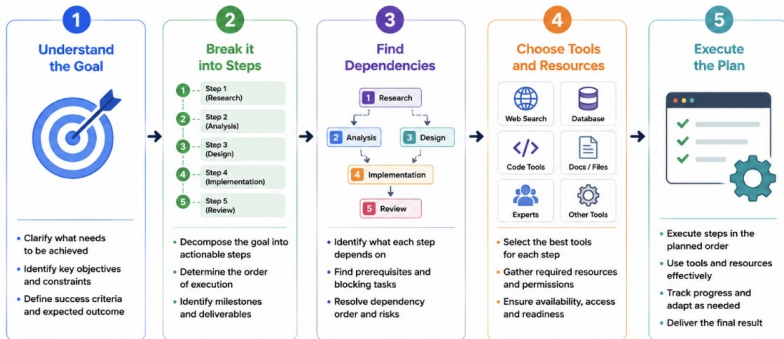
Multiple specialized agents work together to solve one task.



Planning Design Pattern

PLANNING PATTERN

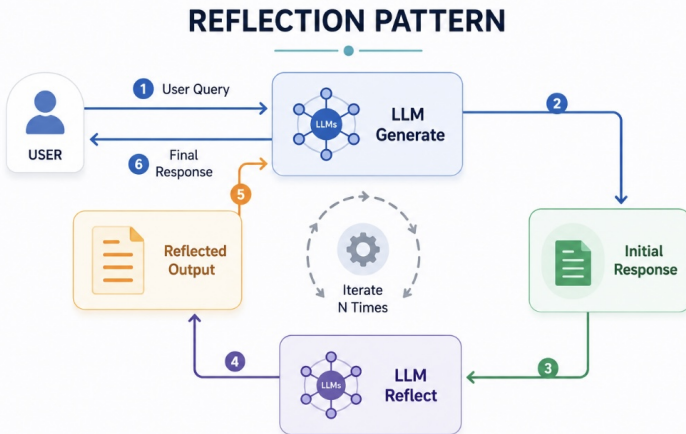
A step-by-step approach to plan smartly and execute successfully.



PLANNING FLOW AT A GLANCE



Reflection Design Pattern



1 User Query
User asks a question.

2 Generate
LLM generates an initial response.

3 Reflect
LLM reflects on the response.

4 Refine
Reflection produces improvements.

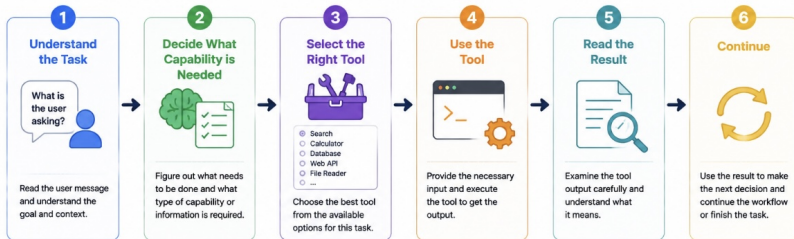
5 Reflected Output
Improved output is produced.

6 Final Response
Deliver the best final response.

Tool-Use Design Pattern

TOOL USE PATTERN

How an AI agent thinks, chooses, uses tools, and continues until the task is complete.



KEY IDEA

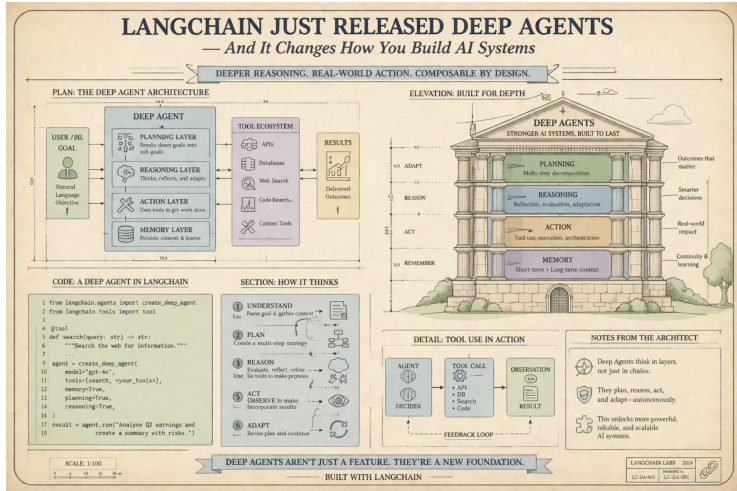
The agent doesn't do everything on its own. It thinks, chooses the right tool, uses it, learns from the result, and keeps going until the task is complete.

BEHIND THE SCENES



Appendix B: Software Considerations

Deep Agent Architecture



Building Agents with `create_agent()`

Benefit of `create_agent()`:

- Manages the ReAct loop

Thought $\longrightarrow$ Action $\longrightarrow$ Observation $\longrightarrow$ Repeat

- Handles message history automatically
- Implements iteration limits to prevent infinite loops
- Provides production-ready error handling
- Returns clean, structured responses

Create agent, get response ...

```
agent = create_agent( model, tools=[calculator], )  
response = agent.invoke({"messages": [HumanMessage(content=query)]})
```