Mastering AI Agents

Presented by:
Aditya Kakkar (zjq5mr), Aryan Sawhney (ryd2fx), Yagnik Panguluri (yye7pm),
Nina Chinnam (fhs9af), Anisha Patrikar (gjq2yf), Mihika Rao (xsw5kn)

Aditya Kakkar (zjq5mr)

Presentation Outline

- ❖ Introduction
- ❖ Background & Motivation
- ❖ What are AI agents?
- ❖ Types of AI Agents & Use Cases

Presentation Outline

- Introduction
- Background & Motivation
- What are AI agents?
- Use Cases & Types of Agents

Chapter 1 - What are AI Agents?

Introduction

Mastering AI Agents: From Theory to Real-World Implementation

Background

Motivation

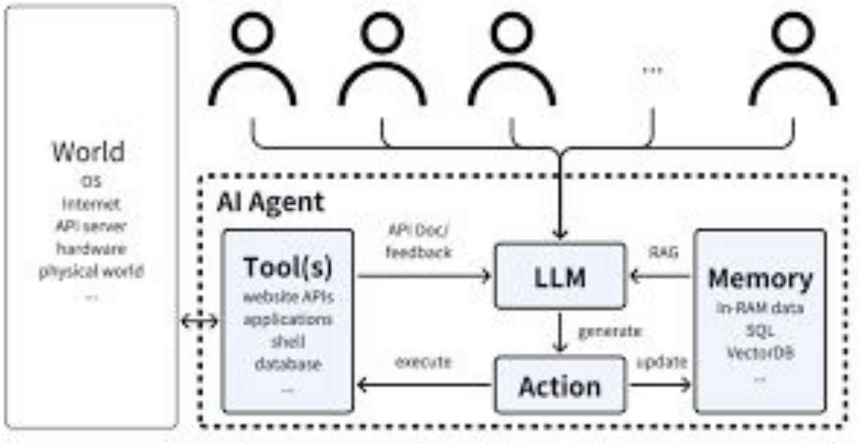
- Companies have quickly adapted, adopted, and integrated AI agents into their workflows.
- Capgemini's research found that over **50%** of companies plan to use AI Agents in 2025 and **82%** plan to integrate them within the next three years.

What are AI Agents?

How do we use LLMs to accomplish end-to-end tasks?

- AI agents take LLMs to the next level by adding decision-making and action-taking capabilities—like making API calls.
- Think of an LLM as the brain that understands and generates text, while an AI agent is the body that takes action based on that intelligence.

When should I use an AI Agent?



They’re incredibly useful for tasks that demand complex decision-making, autonomy, and adaptability, especially helpful in dynamic environments where the workflow involves multiple steps or interactions that could benefit from automation.

Salesforce estimates that salespersons spend **71%** of their time on non-selling tasks (like administrative tasks and manually entering data).

Use Cases

1. **Customer Interaction**

A customer messages your service asking, “When will my order ship?”

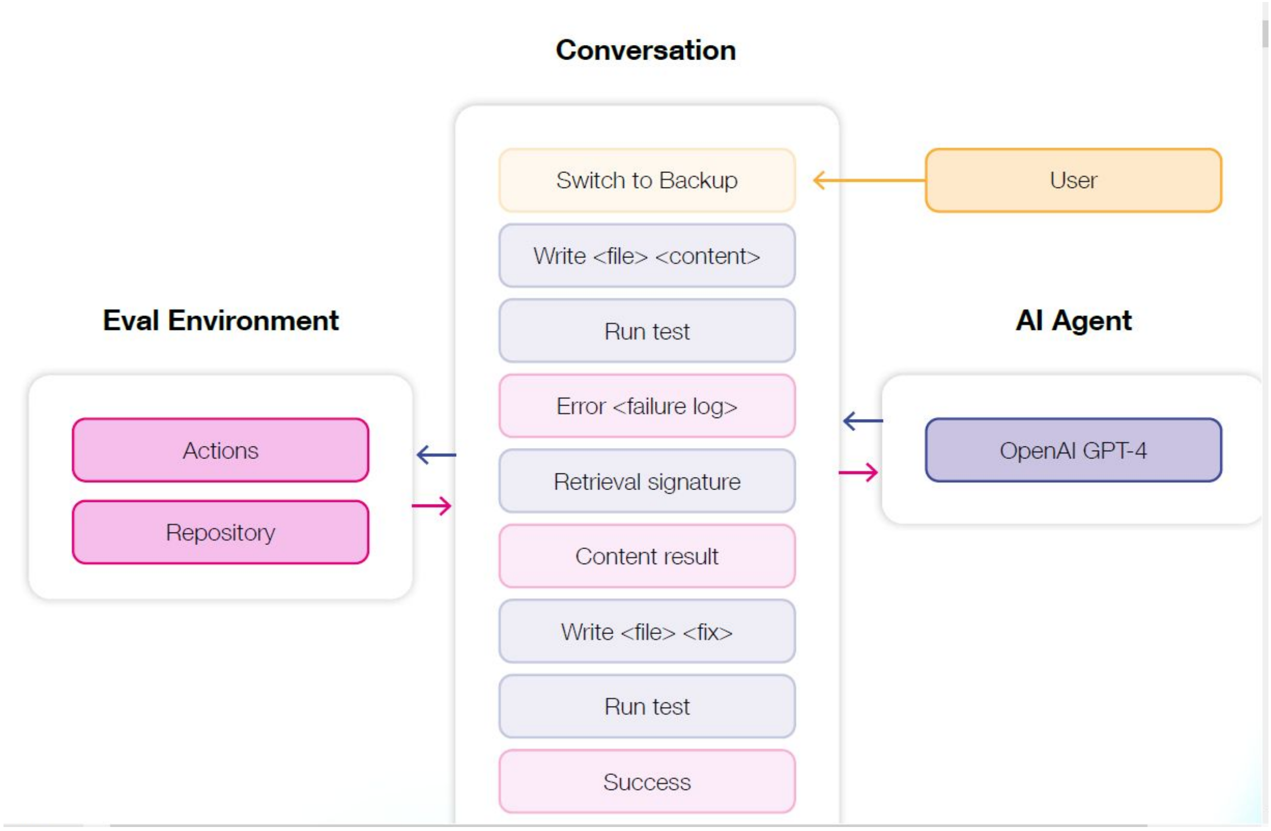
2. **Data Retrieval**

The AI agent accesses the order management system to find the specific order details.

3. **Response Generation**

Based on the data retrieved, the agent automatically provides an updates to the customer, such as sending “Your order will ship tomorrow and you’ll receive a tracking link via email once it’s on its way.”

Use Cases(code)

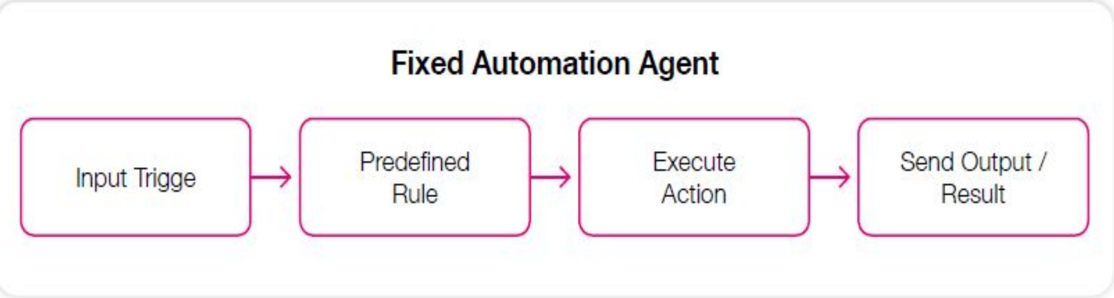


Types of AI Agents

Name of the agent	Key Characteristics	Examples	Best For
Fixed Automation: The Digital Assembly Line	No intelligence, predictable behavior, limited scope	RPA, email autoresponders, basic scripts	Repetitive tasks, structured data, no need for adaptability
LLM-Enhanced: Smarter, but Not Einstein	Context-aware, rule-constrained, stateless	Email filters, content moderation, support ticket routing	Flexible tasks, high-volume/low-stakes, cost-sensitive scenarios
ReAct: Reasoning Meets Action	Multi-step workflows, dynamic planning, basic problem-solving	Travel planners, AI dungeon masters, project planning tools	Strategic planning, multi-stage queries, dynamic adjustments
ReAct + RAG: Grounded Intelligence	External knowledge access, low hallucinations, real-time data	Legal research tools, medical assistants, technical support	High-stakes decisions, domain-specific tasks, real-time knowledge needs
Tool-Enhanced: The Multi-Taskers	Multi-tool integration, dynamic execution, high automation	Code generation tools, data analysis bots	Complex workflows requiring multiple tools and APIs
Self-Reflecting: The Philosophers	Meta-cognition, explainability, self-improvement	Self-evaluating systems, QA agents	Tasks requiring accountability and improvement
Memory-Enhanced: The Personalized Powerhouses	Long-term memory, personalization, adaptive learning	Project management AI, personalized assistants	Individualized experiences, long-term interactions
Environment Controllers: The World Shapers	Active environment control, autonomous operation, feedback-driven	AutoGPT, adaptive robotics, smart cities	System control, IoT integration, autonomous operations
Self-Learning: The Evolutionaries	Autonomous learning, adaptive/scalable, evolutionary behavior	Neural networks, swarm AI, financial prediction models	Cutting-edge research, autonomous learning systems

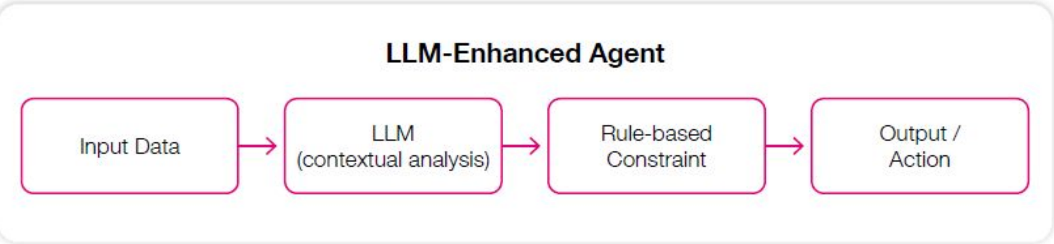
Fixed Automation Agent

Feature	Description
Intelligence	No learning, adaptation, or memory.
Behavior	Predictable and consistent, follows pre-defined rules.
Scope	Limited to repetitive, well-defined tasks. Struggles with unexpected scenarios.
Best Use Cases	Routine tasks, structured data, situations with minimal need for adaptability.



LLM-Enhanced – Smarter, but Not Exactly Einstein

Feature	Description
Intelligence	Context-aware; leverages LLMs to process ambiguous inputs with contextual reasoning.
Behavior	Rule-constrained; decisions are validated against predefined rules or thresholds.
Scope	Stateless; no long-term memory; each task is processed independently.
Best Use Cases	Tasks requiring flexibility with ambiguous inputs, high-volume/low-stakes scenarios, and cost-sensitive situations where "close enough" is sufficient.

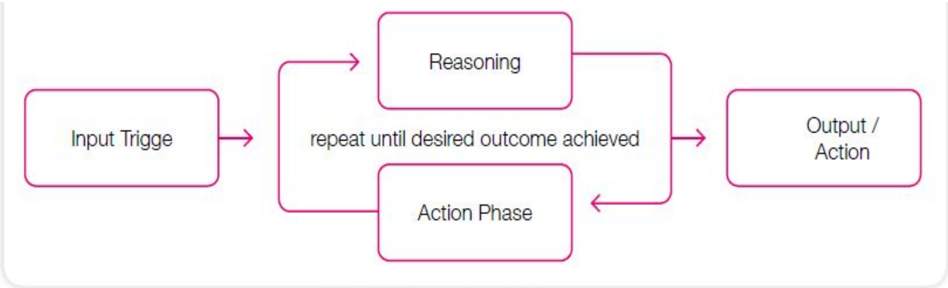


"Press 1 for English, Press 2 for Spanish, Press 3 for Billing…" – that's a basic rule-based IVR (Interactive Voice Response) system.

*"I need help with my bill" → classified as a **billing inquiry***
*"My internet is down" → classified as a **technical issue***

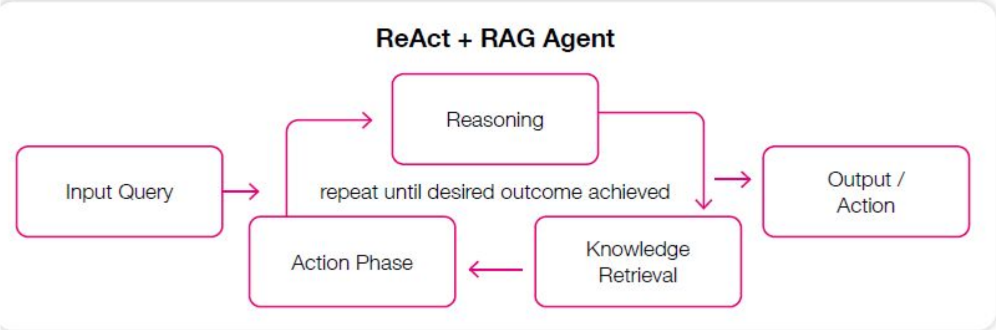
ReAct - Reasoning Meets Action

Feature	Description
Intelligence	Reasoning and action; mimics human problem-solving by thinking through a problem and executing the next step.
Behavior	Handles multi-step workflows, breaking them down into smaller, actionable parts. Dynamically adjusts strategy based on new data.
Scope	Assists with basic open-ended problem-solving, even without a direct solution path.
Best Use Cases	Strategic planning, multi-stage queries, tasks requiring dynamic adjustments, and re-strategizing.



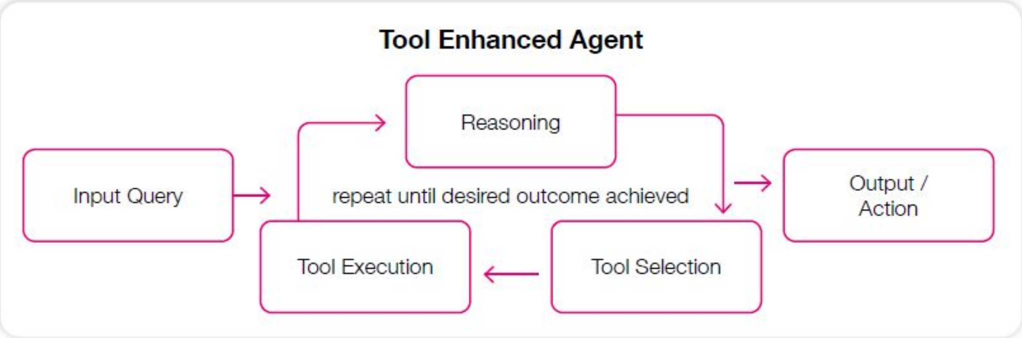
ReAct + RAG – Grounded Intelligence

Feature	Description
Intelligence	Employs a RAG workflow, combining LLMs with external knowledge sources (databases, APIs, documentation) for enhanced context and accuracy.
Behavior	Uses ReAct-style reasoning to break down tasks, dynamically retrieving information as needed. Grounded in real-time or domain-specific knowledge.
Scope	Designed for scenarios requiring high accuracy and relevance, minimizing hallucinations.
Best Use Cases	High-stakes decision-making, domain-specific applications, tasks with dynamic knowledge needs (e.g., real-time updates).



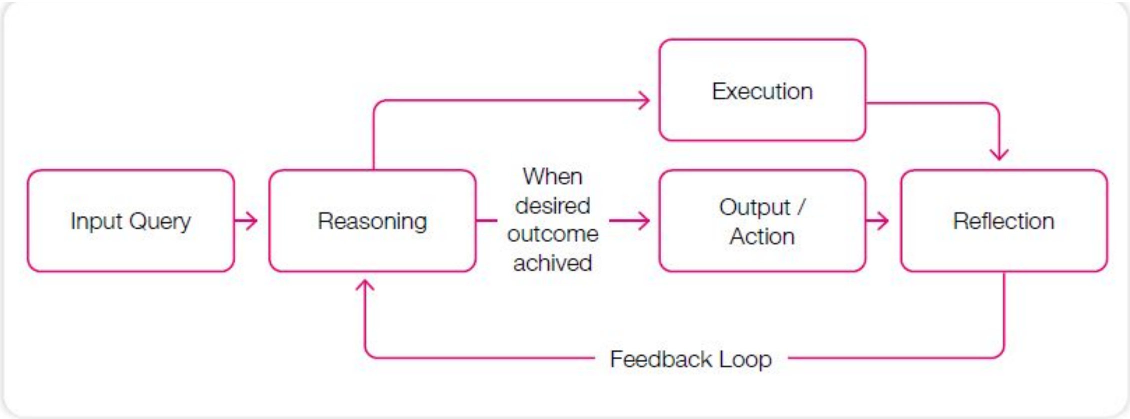
Tool-Enhanced – The Multi-Taskers

Feature	Description
Intelligence	Leverages APIs, databases, and software tools to perform tasks, acting as a multi-tool integrator.
Behavior	Handles multi-step workflows, dynamically switching between tools based on task requirements.
Scope	Automates repetitive or multi-stage processes by integrating and utilizing diverse tools.
Best Use Cases	Jobs requiring diverse tools and APIs in tandem for complex or multi-stage automation.



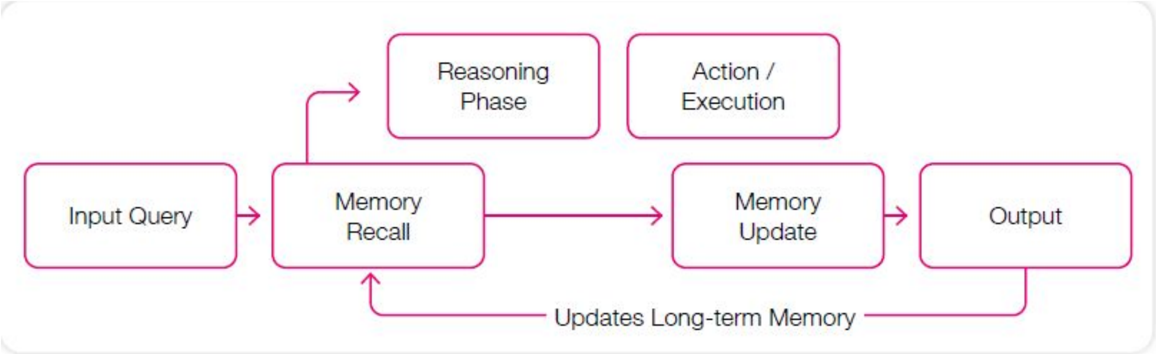
Self-Reflecting – The Philosophers

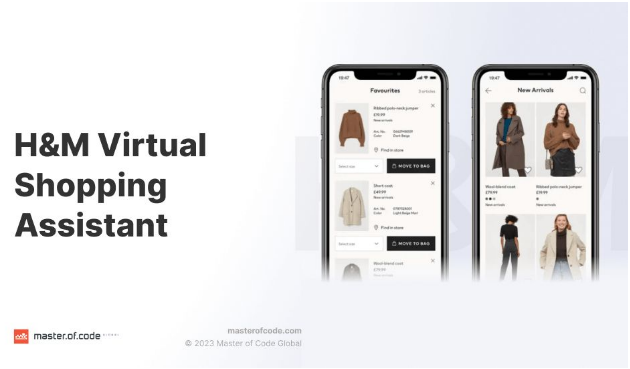
Feature	Description
Intelligence	Exhibits meta-cognition, evaluating its own thought processes and decision outcomes.
Behavior	Provides explanations for actions, offering transparency into its reasoning. Learns from mistakes and improves performance over time.
Scope	Suited for tasks requiring accountability and continuous improvement.
Best Use Cases	Quality assurance, sensitive decision-making where explainability and self-improvement are crucial.



Memory-Enhanced – The Personalized Powerhouses

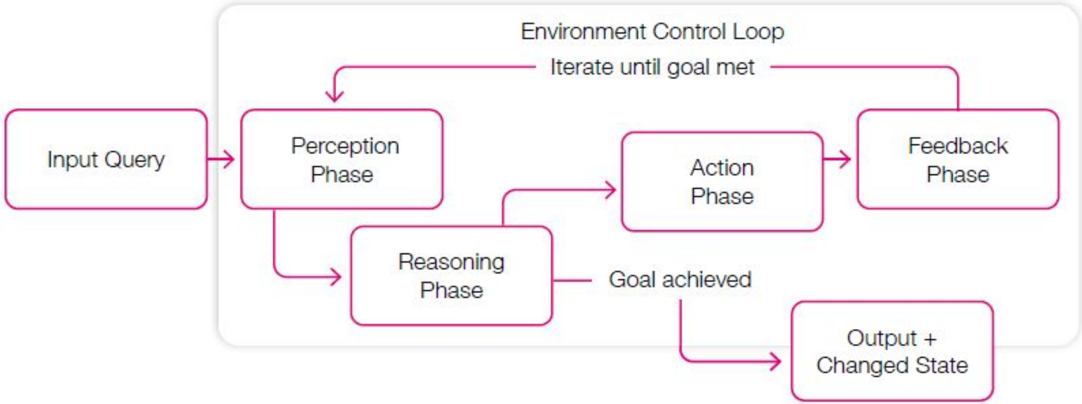
Feature	Description
Intelligence	Possesses long-term memory, storing and recalling past interactions, preferences, and task progress.
Behavior	Provides context-aware personalization, adapting decisions and actions based on user-specific data and history. Learns and improves over time.
Scope	Excels at tasks requiring individualized experiences, tailored recommendations, and maintaining consistency across multiple interactions.
Best Use Cases	Personalized assistance, long-term interactions, tasks spanning multiple sessions.





Environment Controllers – The World Shapers

Feature	Description
Intelligence	Autonomous learning; refines models and processes based on feedback, data, or environmental changes without manual updates.
Behavior	Adaptive and scalable, adjusting to changing conditions and new tasks. Exhibits evolutionary behavior, improving performance over time.
Scope	Suited for cutting-edge research and autonomous learning systems, offering high potential but requiring careful monitoring.
Best Use Cases	Situations where autonomous learning and adaptation are crucial, such as complex research, simulation, or dynamic environments.



UNIVERSITY of VIRGINIA

Aryan Sawhney (ryd2fx)

Self-Learning – The Evolutionaries

Feature	Description
Intelligence	Autonomous learning; refines models and processes based on feedback, data, or environmental changes without manual updates.
Behavior	Adaptive and scalable, adjusting to changing conditions and new tasks. Exhibits evolutionary behavior, improving performance over time.
Scope	Suited for cutting-edge research and autonomous learning systems, offering high potential but requiring careful monitoring.
Best Use Cases	Situations where autonomous learning and adaptation are crucial, such as complex research, simulation, or dynamic environments.
Examples	Neural networks with evolutionary capabilities, swarm AI systems, autonomous robotics, financial prediction models.

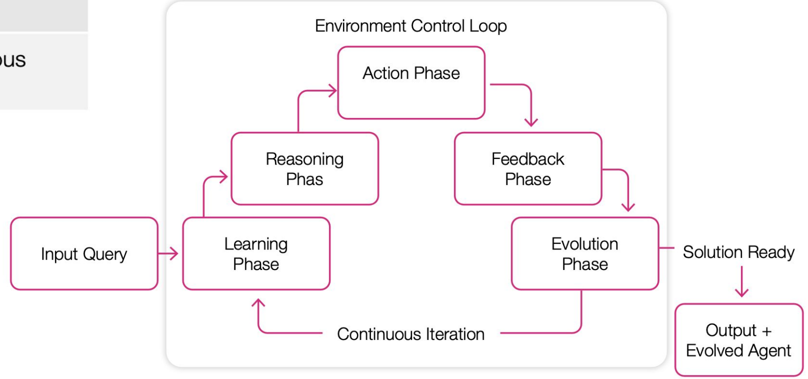


Fig 1.10: Workflow of a self-learning agent

When to Use Agents?

- AI Agents excel at tasks that require:
  - Complex decision-making
  - Autonomy
  - Adaptability

- AI Agents excel at environments where workflow is dynamic and involves multiple steps or interactions

Domain	Task	Benefits of Using AI Agents
Customer Support	Handling queries, providing real-time assistance, issue escalation	Agents enhance the efficiency and customer experience by offering timely and accurate responses, allowing human staff to focus on more complex issues.
Research and Data Analysis	Gathering, processing, and analyzing data	They autonomously provide deep insights from large datasets, helping you understand patterns without manual effort.
Financial Trading	Real-time data processing	Agents excel in making quick decisions based on rapidly-changing market conditions.
Education	Personalized learning experiences	These agents adapt to each student's learning pace, offering tailored feedback and supporting unique learning journeys effectively.
Software Development	Code generation, debugging, and testing	Agents streamline the development process by handling repetitive tasks like coding and testing, improving code quality, and reducing development time. They also learn and improve over time, which continually enhances their assistance.

Table 1.11: Domains and applications that can benefit from the use of AI agents

When Not to Use Agents?

- Tasks are too simple or infrequent
  - Minimal automation needed; traditional software is more efficient
  - Complexity and cost of AI agents may not be justified
- Requires deep domain expertise
  - Legal analysis, medical diagnosis, and high-stakes decision-making are better handled by professionals
  - Sole reliance on AI can lead to suboptimal or harmful outcomes
- Human emotion and creativity are essential
  - Fields like psychotherapy, counseling, and creative writing require a human touch
  - AI lacks the depth to fully understand emotions and creativity
- High implementation costs
  - Small businesses and budget-constrained projects may find AI agents too expensive
  - Development and maintenance costs may outweigh benefits
- Regulatory and compliance challenges
  - Highly regulated industries impose strict security and legal constraints
  - Ensuring AI agents meet compliance standards is resource-intensive

10 Questions to Ask Before You Consider an AI Agent

01 What is the complexity of the task?

02 How often does the task occur?

03 What is the expected volume of data or queries?

04 Does the task require adaptability?

05 Can the task benefit from learning and evolving over time?

06 What level of accuracy is required?

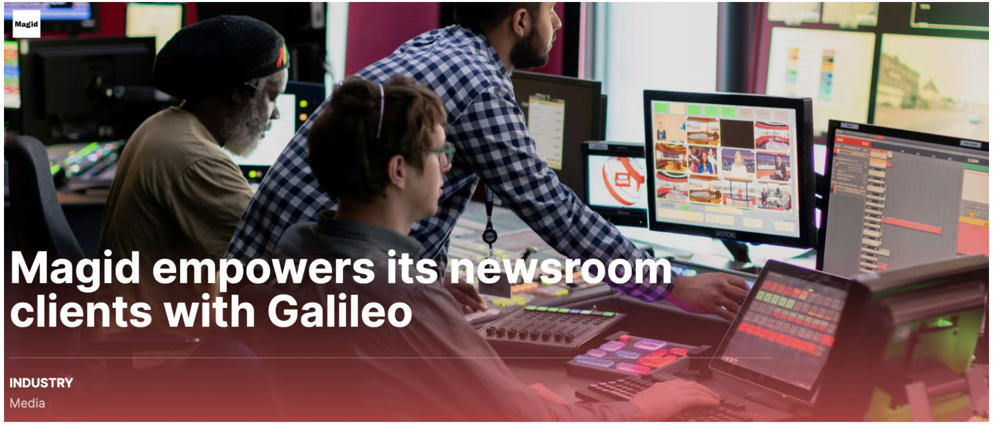
07 Is human expertise or emotional intelligence essential?

08 What are the privacy and security implications?

09 What are the regulatory and compliance requirements?

10 What is the cost-benefit analysis?

3 Interesting Real-World Use Cases of AI Agents



Wiley and Agentforce

Company:
Wiley

AI Agent:
Agentforce by Salesforce

Use Case:
Customer service automation

Problem:
Wiley faced challenges handling spikes in service calls during peak times, particularly at the start of new semesters when thousands of students use Wiley's educational resources.

Need:
The company needed an efficient customer service system to manage the increased volume and maintain positive customer experiences.

Solution:
Wiley invested in Salesforce's Agentforce, an AI agent designed to enhance customer service operations. This integration has significantly improved case resolution rates and faster resolution of customer queries, especially during peak times, such as the start of new semesters when demand spikes.

ROI:
A 40%+ increase in case resolution compared to their previous chatbot, a 213% ROI, and $230K in savings

Oracle Health and Clinical AI agent

Company:
Oracle Health

AI Agent:
Clinical AI Agen

Use Case:
Enhancing patient-provider interactions

Problem:
Healthcare providers faced documentation and time management challenges during patient visits, leading to burnout and reduced patient engagement.

Need:
There was a need for a solution that could streamline clinical workflows and improve documentation accuracy while allowing providers more time to interact with patients.

Solution:
Oracle Health developed its Clinical AI Agent, which automates documentation processes and enhances patient-provider interactions through a multimodal voice user interface. This allows providers to access patient information quickly and generate accurate notes efficiently.

ROI:
AtlantiCare, using the Clinical AI Agent, reported a 41% reduction in total documentation time, saving approximately 66 minutes per day, which translates to improved productivity and enhanced patient satisfaction.

Magid and Galileo

Company:
Magid

AI Agent:
RAG-based system powered with real-time observability capabilities

Use Case:
Empowering newsrooms with generative AI technology

Problem:
Magid, a leader in consumer intelligence for media brands, needed to ensure consistent, high-quality content in a fast-paced news environment. The complexity of diverse topics made it challenging to uphold accuracy, and errors could potentially lead to significant repercussions.

Need:
A robust observability system was essential for monitoring AI-driven workflows and ensuring the quality of outputs across various clients. This scalability was crucial for managing the daily production of numerous stories.

Solution:
Magid integrated Galileo's real-time observability capabilities into their product ecosystem. This integration provided production monitoring, relevant metrics for tracking tone and accuracy, and customization options tailored to Magid's needs.

ROI:
With Galileo, Magid achieved 100% visibility over inputs and outputs, enabling customized offerings as they scale. This visibility helps identify trends and develop client-specific metrics, enhancing the accuracy of news delivery.

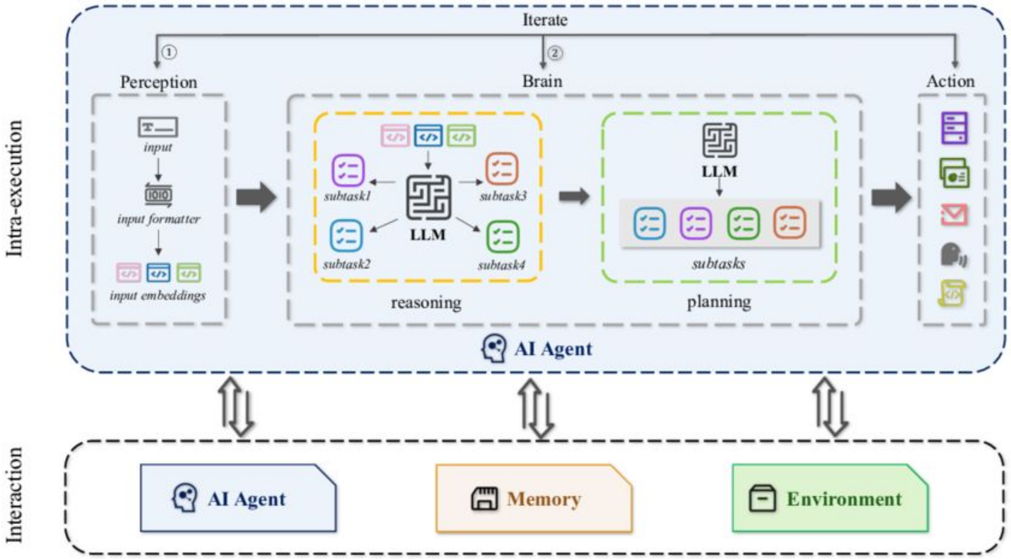
Nina Chinnam (fhs9af)

Chapter 2 - Frameworks for Building AI Agents

Presentation Outline

- Importance of AI Agent Frameworks
- Overview of Frameworks
- Deep Dive of 3 Frameworks
- Feature Comparison

Why do we need AI agent frameworks?



- State Management
- Tool Integrations
- Multi-Agent Coordination

Three AI Agent Frameworks

LangGraph

AG

crewai

High Level Comparison of Frameworks

Feature	LangGraph	Autogen	CrewAI
Execution Model	DAG-based execution	Conversational AI Modeling	Role-based multi agent system
Best For	Structured, DAG-based workflows	Chat-driven AI applications	Teams of agents working together
Memory Handling	Long-term, short-term, entity	Moderate, Conversation-based tracking	Shared-multi agent memory
Tool Support	Deep LangChain integration	Code execution, function calls	Customizable tools & LangChain support
Scalability	Highly scalable	Scales well for chat-bot like systems	Scales for teams & delegation

LangGraph Overview

LangGraph

Core Features

DAG-based execution flow

Structured, deterministic task execution

Ideal Use Cases

Data processing pipelines

Automated research workflows

AI-driven decision making systems

Example: Automating Research Pipelines

Workflow

- User inputs a research topic
- AI retrieves papers from multiple sources
- AI extracts insights from each source
- AI combines findings into a final structured report

LangGraph's DAG Approach

```
# Define Input Node (User provides research topic)
query_input = graph.add_node("User Inputs Topic")

# Define Parallel Retrieval Nodes (Fetching from multiple sources)
fetch_arxiv = graph.add_node("Fetch Papers from ArXiv")
fetch_ieee = graph.add_node("Fetch Papers from IEEE")
fetch_google = graph.add_node("Fetch Papers from Google Scholar")

# Define Processing Nodes (Running in Parallel)
analyze_arxiv = graph.add_node("Analyze ArXiv Papers")
analyze_ieee = graph.add_node("Analyze IEEE Papers")
analyze_google = graph.add_node("Analyze Google Scholar Papers")

# Define Final Aggregation & Report Generation Node
aggregate_findings = graph.add_node("Aggregate Insights & Generate Report")
```

```
# Connect Nodes (Creating Parallel Execution Paths)
graph.add_edge(query_input, fetch_arxiv)
graph.add_edge(query_input, fetch_ieee)
graph.add_edge(query_input, fetch_google)

graph.add_edge(fetch_arxiv, analyze_arxiv)
graph.add_edge(fetch_ieee, analyze_ieee)
graph.add_edge(fetch_google, analyze_google)

# All analyses feed into the final report generation
graph.add_edge(analyze_arxiv, aggregate_findings)
graph.add_edge(analyze_ieee, aggregate_findings)
graph.add_edge(analyze_google, aggregate_findings)
```

Autogen Overview

Core Features

Enables AI agents to interact via conversations

Ideal for AI assistants and chat-driven workflows

Function calling and code execution

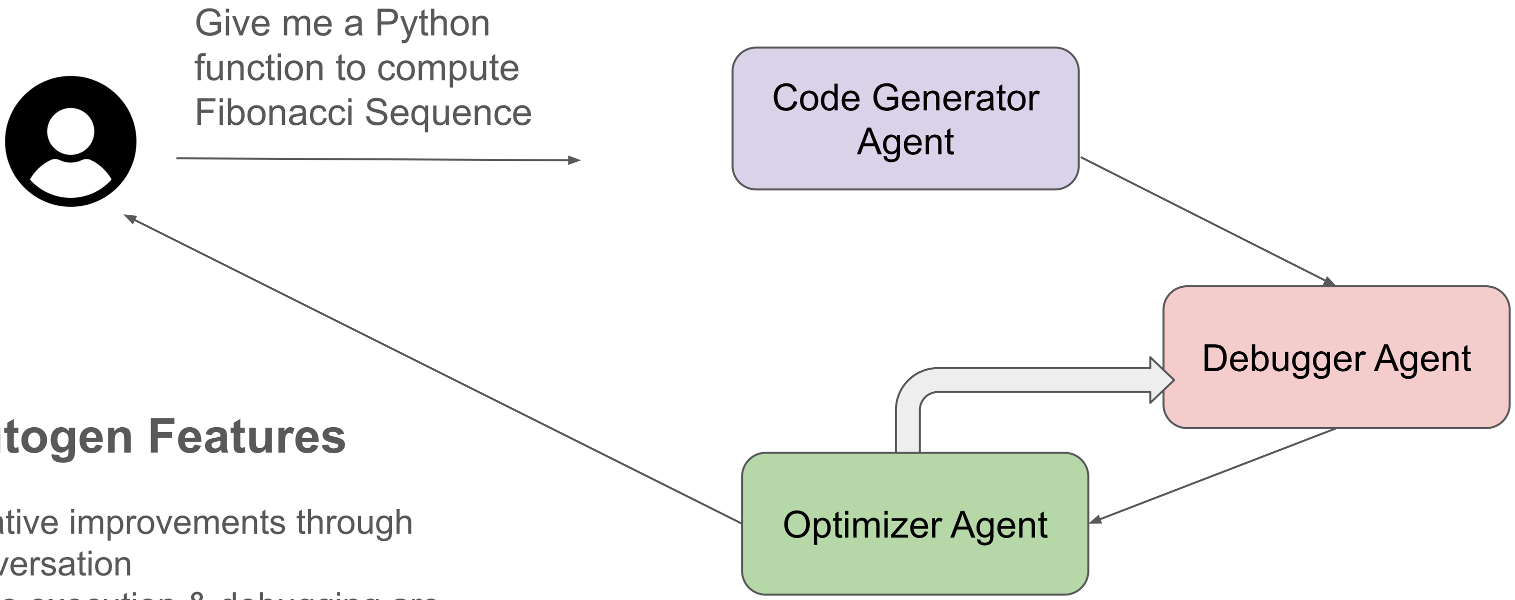
Ideal Use Cases

Conversational AI for customer support

AI Research Assistants

Collaborative Q&A Systems

Example: AI Coding Assistant



Key Autogen Features

- Iterative improvements through conversation
- Code execution & debugging are built-in
- Agents engage in dynamic problem-solving

CrewAI Overview

Core Features

Multiple AI agents work together with assigned roles

Structured Teamwork

Agents can be assigned hierarchical roles

Ideal Use Cases

AI-powered research teams

Automated content generation

Decision-making systems

Example: AI-Based Newsroom Workflow

Specific Workflow Roles

Research Agent: Gathers news stories

Writing Agent: Generates draft articles

Editing Agent: Fixes grammar, refines the draft

Fact-Checking Agent: Ensures accuracy before publishing

CrewAI Implementation

```python
from crewai import Agent, Task, Crew

# Define Agents
research_agent = Agent(role="Research", goal="Gather latest news")
writing_agent = Agent(role="Writer", goal="Draft news articles")
editing_agent = Agent(role="Editor", goal="Refine and fix writing")
fact_checker = Agent(role="Fact-Checker", goal="Verify information accuracy")

# Define Tasks
gather_news = Task(name="Find breaking news", agent=research_agent)
write_news = Task(name="Write article draft", agent=writing_agent, depends_on=[gather_news])
edit_article = Task(name="Edit the draft", agent=editing_agent, depends_on=[write_news])
verify_info = Task(name="Fact-check the article", agent=fact_checker, depends_on=[edit_article])

# Crew Orchestration
newsroom_crew = Crew(tasks=[gather_news, write_news, edit_article, verify_info])
newsroom_crew.kickoff()
```

Feature Comparison

Ease of Use	Autogen
Multi-Agent Support	CrewAI
Tool Support	LangGraph & CrewAI
Memory Handling	LangGraph & CrewAI
Scalability	All
Customization	All

Choosing the Right Framework

Structured execution and task control

LangGraph

AI agents to reason via conversations

Autogen

Multiple AI agents working in roles

CrewAI

Mihika Rao (xsw5kn)

Chapter 3 - How to Evaluate Agents

Presentation Outline

- ❖ Why Evaluate AI Agents?
- ❖ Case Study
- ❖ AI plan execution
- ❖ Galileo Callback

Why Evaluate AI Agents?

- Tasks are performed correctly and reliably
- Ensure accurate, relevant, and efficient responses

Case Study: Building a Financial Research Agent

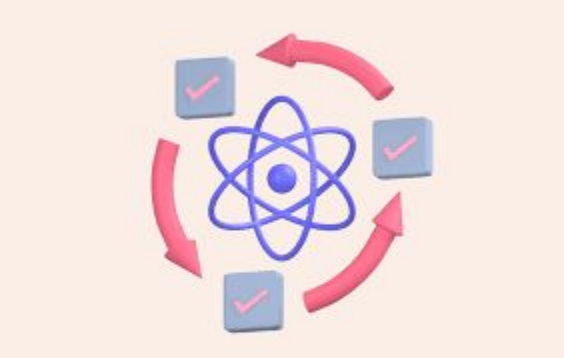
- Approach:
  - Break down research into smaller steps
  - Search for external data (Tavily)
  - Analyze findings with ReAct (Reasoning + Action)

```python
from langchain_openai import ChatOpenAI
from langchain_community.tools.tavily_search import TavilySearchResults
from langgraph.prebuilt import create_react_agent

system_prompt = "You are a helpful finance expert named Fred in year 2024. First of all you create a plan to get answer to the research query. Then you use tools to get answers to the questions. Finally you use the answers to each question in the plan to give your final verdict."

llm = ChatOpenAI(model="gpt-4o-mini")
tools = [TavilySearchResults(max_results=3)]
agent_executor = create_react_agent(llm, tools, state_modifier=system_prompt)
```

Understanding State Management in AI Agents



```python
import operator
from pydantic import BaseModel, Field
from typing import Annotated, List, Tuple
from typing_extensions import TypedDict


class PlanExecute(TypedDict):
    input: str
    plan: List[str]
    past_steps: Annotated[List[Tuple], operator.add]
    response: str


class Plan(BaseModel):
    """Plan to follow in future"""

    steps: List[str] = Field(
        description="different steps to follow, should be in sorted order"
    )
```

- How agent keeps track of its progress while solving a task.
- (action, result_action)
- Store input and response
- Benefits:
    - Avoids redundant work
    - Helps agent replan efficiently

    Helps track progress to final answer

Creating the Plan

```python
from langchain_core.prompts import ChatPromptTemplate

planner_prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """You are a finance research agent working in Oct 2024. For the given
objective, come up with a simple step by step plan. \
This plan should involve individual tasks, that if executed correctly will yield the
correct answer. Do not add any superfluous steps. The result of the final step should be
the final answer. Make sure that each step has all the information needed - do not skip
steps. At the end use the info collected to give the final answer to the main question
containing the facts.""",
        ),
        ("placeholder", "{messages}"),
    ]
)
```

Fig. 3.3: Guiding the agent to create a step-by-step plan that should lead to the correct answer for a given objective

```python
planner = planner_prompt | ChatOpenAI(
    model="gpt-4o-mini", temperature=0
).with_structured_output(Plan)

planner.invoke(
    {
        "messages": [
            ("user", "Should we invest in Tesla given the current situation of EV?")
        ]
    }
)
```

Fig. 3.4: Testing the agent with a question

Re-planning - Adjusting the Agent's Strategy

- Original question
- Initial Plan
- Completed Steps

```
replanner_prompt = ChatPromptTemplate.from_template(
    """For the given objective, come up with a simple step by step plan. \
This plan should involve individual tasks, that if executed correctly will yield the
correct answer. Do not add any superfluous steps. \
The result of the final step should be the final answer. Make sure that each step has all
the information needed - do not skip steps.

Your objective was this:
{input}

Your original plan was this:
{plan}

You have currently done the follow steps:
{past_steps}

Update your plan accordingly. If no more steps are needed and you can return to the user,
then respond with that. Otherwise, fill out the plan. Only add steps to the plan that still
NEED to be done. Do not return previously done steps as part of the plan."""
)
```

Executing the Plan

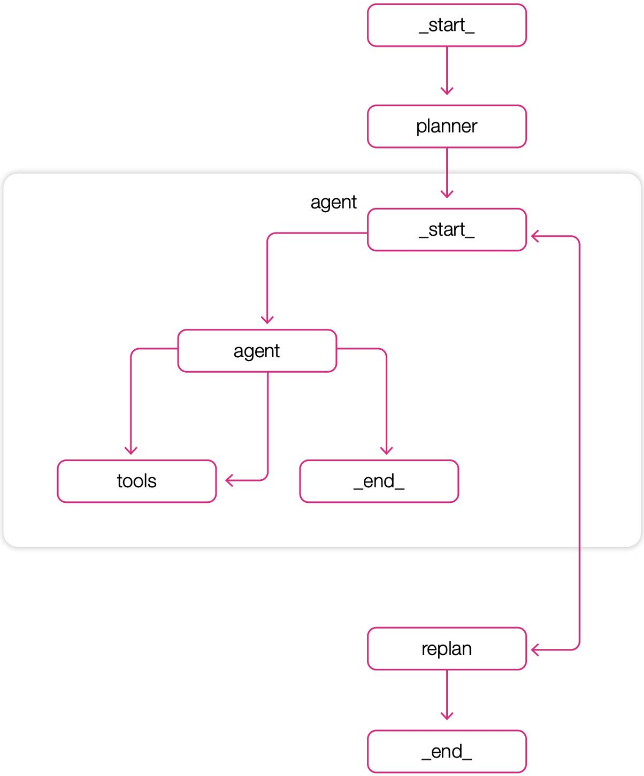
```python
async def plan_step(state: PlanExecute):
    plan = await planner.ainvoke({"messages": [("user", state["input"])]})
    return {"plan": plan.steps}
```

```python
async def execute_step(state: PlanExecute):
    plan = state["plan"]
    plan_str = "\n".join(f"{i+1}. {step}" for i, step in enumerate(plan))
    task = plan[0]
    task_formatted = f"""For the following plan:
{plan_str}\n\nYou are tasked with executing step {1}, {task}."""
    agent_response = await agent_executor.ainvoke(
        {"messages": [("user", task_formatted)]}
    )
    return {
        "past_steps": [(task, agent_response["messages"][-1].content)],
    }
```

```python
async def replan_step(state: PlanExecute):
    output = await replanner.ainvoke(state)
    if isinstance(output.action, Response):
        return {"response": output.action.response}
    else:
        return {"plan": output.action.steps}
```

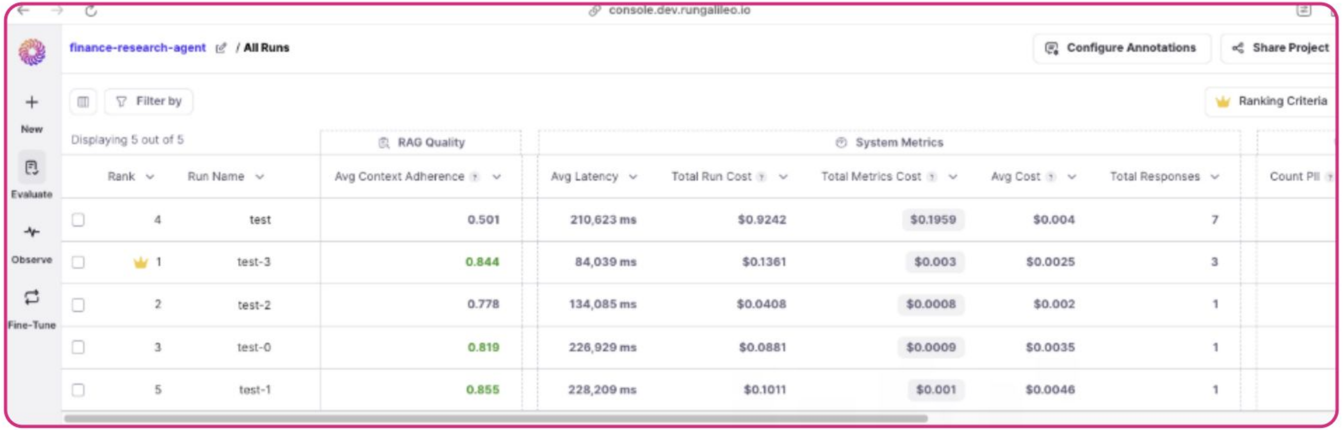
```python
def should_end(state: PlanExecute):
    if "response" in state and state["response"]:
        return END
    else:
        return "agent"
```

State Graph



Galileo Callback - Debugging and Optimizing

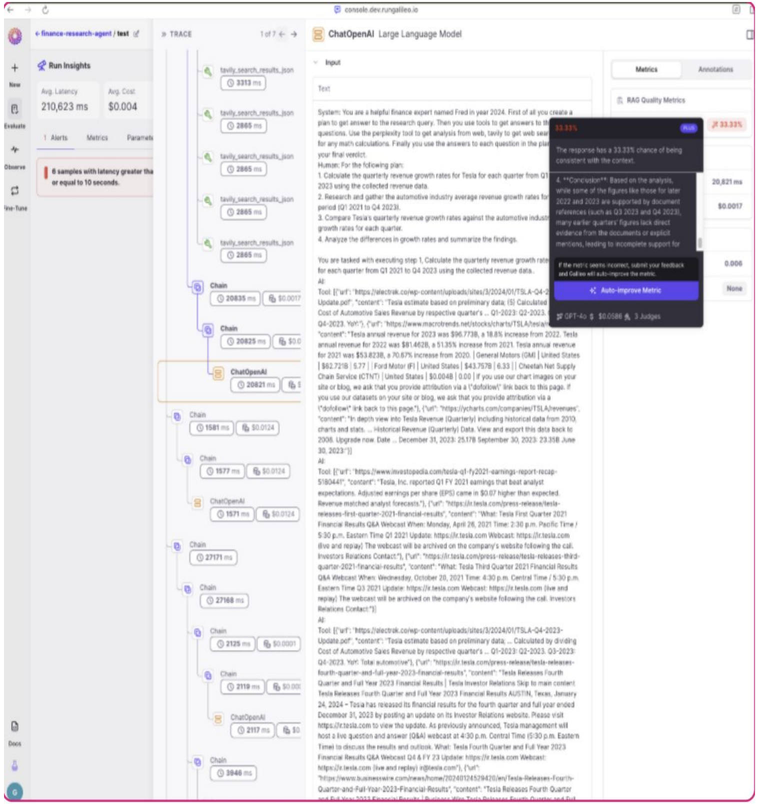
```
{'plan': ['Research the current market trends in the electric vehicle (EV) industry as of October 2024.', "Analyze Tesla's current financial
{'past_steps': [('Research the current market trends in the electric vehicle (EV) industry as of October 2024.', "### Current Market Trends i
{'plan': ["Analyze Tesla's current financial performance, including revenue, profit margins, and growth rates.", "Evaluate Tesla's competitiv
{'past_steps': [("Analyze Tesla's current financial performance, including revenue, profit margins, and growth rates.", "### Step 1: Analysis
{'plan': ["Evaluate Tesla's competitive landscape, identifying key competitors in the EV market and their market shares.", 'Assess the risks
{'past_steps': [("Evaluate Tesla's competitive landscape, identifying key competitors in the EV market and their market shares.", "### Step 1
{'plan': ['Assess the risks associated with investing in Tesla, including regulatory risks, market volatility, and technological changes.', '
{'past_steps': [('Assess the risks associated with investing in Tesla, including regulatory risks, market volatility, and technological chang
{'plan': ['Gather stock price forecast information for Tesla for 1 year, 3 years, and 5 years from reputable financial analysts and sources.'
{'past_steps': [('Gather stock price forecast information for Tesla for 1 year, 3 years, and 5 years from reputable financial analysts and so
{'plan': ['Compile the findings from the market analysis, financial performance, competition, risks, and stock forecasts into a comprehensive
{'past_steps': [('Compile the findings from the market analysis, financial performance, competition, risks, and stock forecasts into a compre
{'plan': ['Make a final recommendation on whether to invest in Tesla based on the compiled data.']}
{'past_steps': [('Make a final recommendation on whether to invest in Tesla based on the compiled data.', "Based on the gathered data regardi
{'response': 'The analysis and recommendation process for investing in Tesla has been completed. Based on the comprehensive overview and fina
Initial job complete, executing scorers asynchronously. Current status:
cost: Done ✅
toxicity: Computing 🚧
pii: Computing 🚧
protect_status: Done ✅
latency: Done ✅
groundedness: Computing 🚧
🔭 View your prompt run on the Galileo console at: https://console.dev.rungalileo.io/prompt/chains/1704927a-e6e7-4b22-9cff-890ebd8d32bf/be004
```



Rank	Run Name	Avg Context Adherence	Avg Latency	Total Run Cost	Total Metrics Cost	Avg Cost	Total Responses	Count PII
		RAG Quality	System Metrics					
4	test	0.501	210,623 ms	$0.9242	$0.1959	$0.004	7	
1	test-3	0.844	84,039 ms	$0.1361	$0.003	$0.0025	3	
2	test-2	0.778	134,085 ms	$0.0408	$0.0008	$0.002	1	
3	test-0	0.819	226,929 ms	$0.0881	$0.0009	$0.0035	1	
5	test-1	0.855	228,209 ms	$0.1011	$0.001	$0.0046	1	

Galileo Callback - Example

- Problem: AI agent has context adherence issues
  - Trace view shows 33.33% context adherence score
- System explanation: AI correctly cited some recent figures (Q3 2024, Q4 2023), earlier figures lacked explicit supporting evidence



Anisha Patrikar (gjq2yf)

Chapter 4 - Metrics for Evaluating AI Agents

Presentation Outline

- ❖ Key Performance Dimensions
- ❖ 5 Case Studies
- ❖ Common Challenges in AI Evaluation
- ❖ Best Practices for AI Optimization

Key Performance Dimensions

Fig 4.1: Four key performance dimensions to evaluate AI agents

Case Study 1: Advancing the Claims Processing Agent

AI deployed for automating claims processing

❌ Struggled with complex claims

❌ Increased compliance risks due to errors in complex cases

Challenges & Solutions

1. LLM Call Error Rate
2. Task Completion Rate
3. Human Intervention Requests
4. Token Usage per Interaction

Outcomes

• Faster claims processing
• Higher compliance accuracy
• Improved resource utilization
• Reduced rejection rates

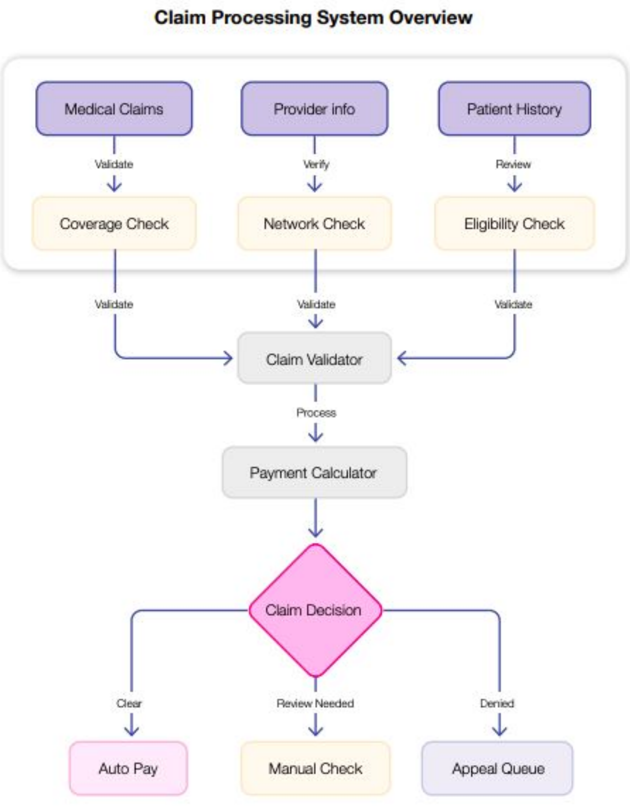


Fig 4.2: An overview of the Claims Processing System

Case Study 2: Optimizing the Tax Audit Agent

AI deployed for tax document processing

❌ Lengthy turnaround times for complex audits

❌ High computing costs from inefficient processing

❌ A backlog of partially completed audits requiring manual review

Challenges & Solutions

1. Tool Success Rate
2. Context Window Utilization
3. Steps per Task

Outcomes

• Faster audit completion
• Enhanced discrepancy detection
• Optimized processing efficiency

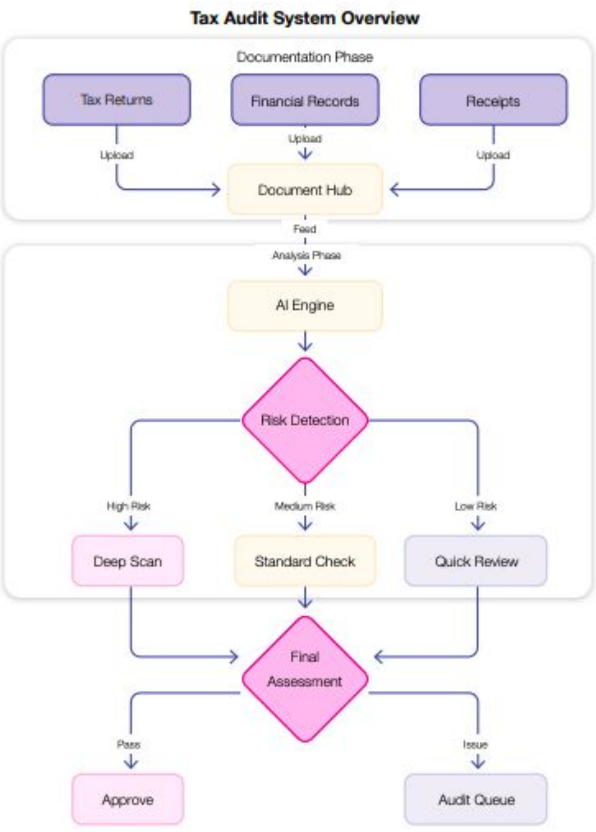


Fig 4.3: An overview of the Tax Auditing System

Case Study 3: Elevating the Stock Analysis Agent

AI deployed for investment analysis

❌ Redundant analysis requests
❌ Inconsistent reporting formats
❌ Failed to adjust analysis to market conditions

Challenges & Solutions

1. Total Task Completion Time
2. Output Format Success Rate
3. Token Usage per Interaction

Outcomes

• More precise market analysis

• Faster processing times

• Optimized resource utilization

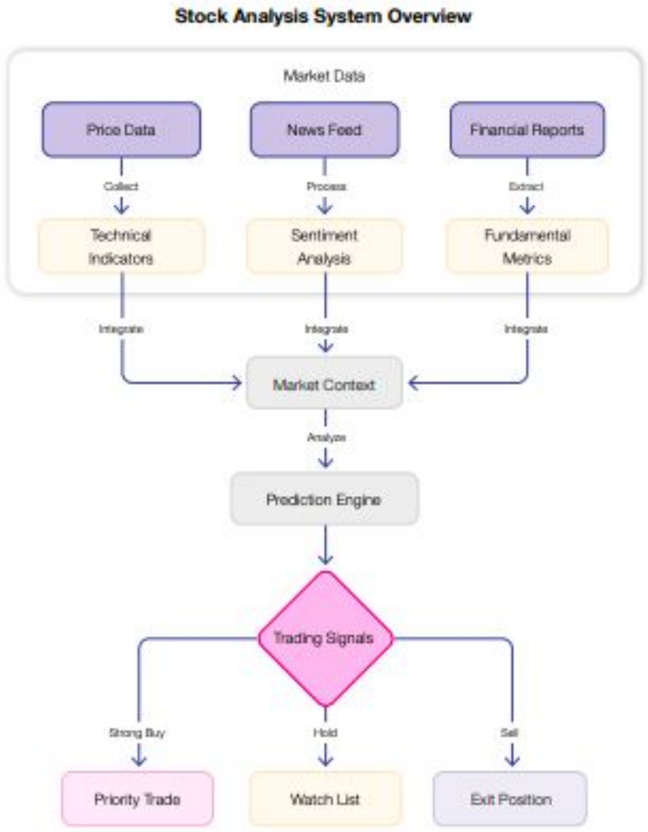


Fig 4.4: An overview of the Stock Analysis System

Case Study 4: Upgrading the Coding Agent

AI deployed to enhance developer productivity

❌ Frequent disruptions during sprint deadlines
❌ Struggled with large codebases, providing irrelevant suggestions
❌ Rising infrastructure costs due to inefficient resource usage

Challenges & Solutions

1. LLM Call Error Rate
2. Task Success Rate
3. Cost per Task Completion

Outcomes

- More accurate code analysis
- Improved suggestion relevance
- Optimized resource utilization

Development Assistant System Overview

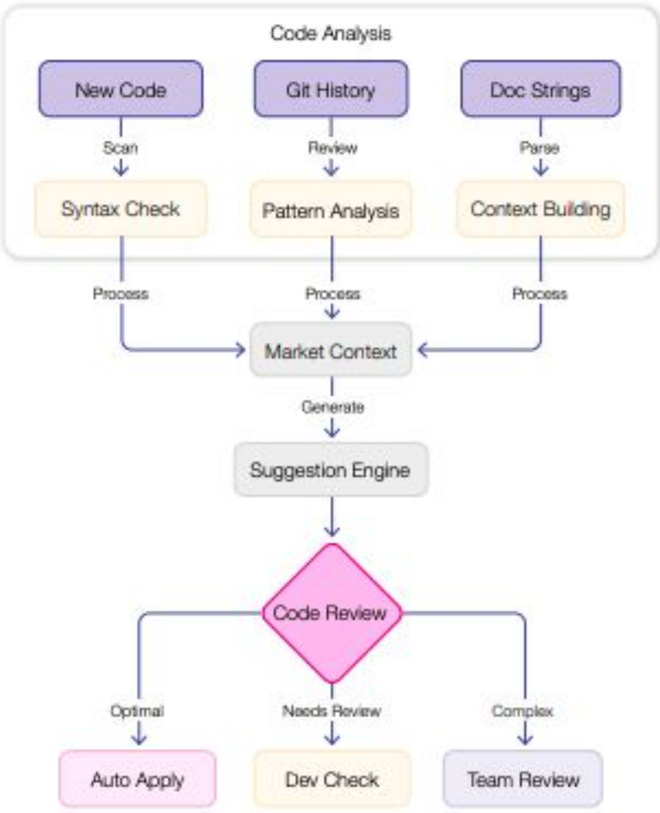


Fig 4.5: An overview of the Development Assistant System

Case Study 5: Enhancing the Lead Scoring Agent

AI deployed to enhance lead qualification efficiency

❌ Misclassification of prospects led to declining conversion rates
❌ Sales reps pursued low-quality leads due to inaccurate scoring
❌ Increased costs per qualified lead, impacting growth targets

Challenges & Solutions

1. Token Usage per Interaction
2. Latency per Tool Call
3. Tool Selection Accuracy

Outcomes

• Faster prospect analysis

• Higher lead qualification accuracy

• Optimized resource utilization

Lead Scoring System Overview

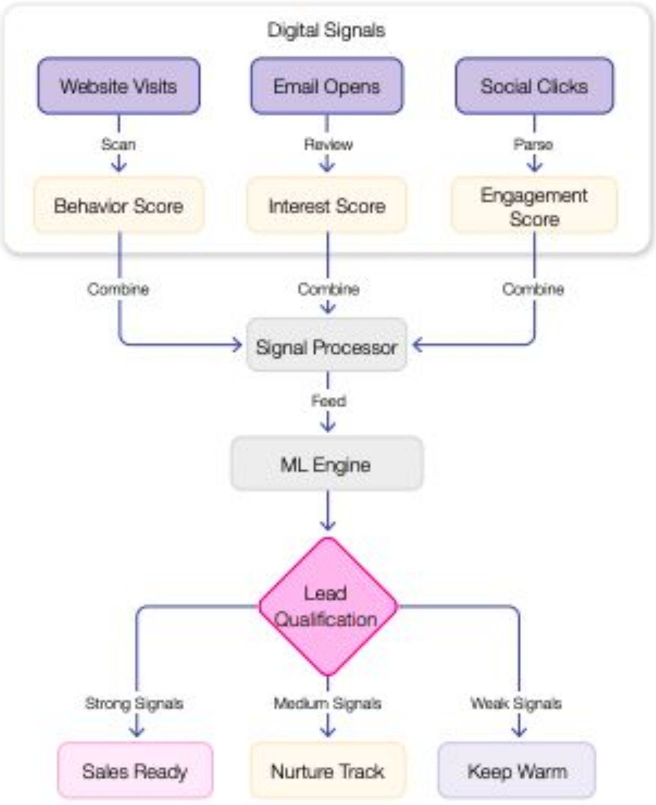
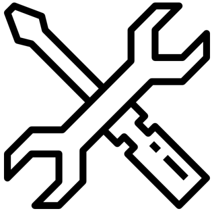


Fig 4.6: An overview of the Lead Scoring System

Common Challenges in AI Evaluation



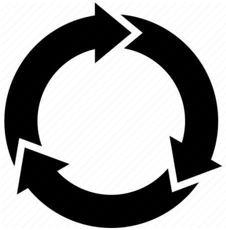
Ensuring consistency across test runs

Identifying model weaknesses

Improving real-time adaptation

Best Practices for AI Agent Optimization

Continuous
Monitoring



Adaptive
Evaluation
Methods

Strategic
Performance
Adjustments

Yagnik Panguluri (yye7pm)

Chapter 5 - Why Most AI Agents Fail and How To Fix Them

Why Do AI Agents Fail?

AI Agents are becoming widely used in industries like finance, healthcare, and customer support. Despite their potential, many fail due to poor design, lack of adaptability, and high operation costs.

DEVELOPMENT ISSUES

Poorly Defined Prompts
- Define Clear Objectives
- Craft Detailed Personas
- Use Effective Prompting

Evaluation Challenges
- Continuous Evaluation
- Use Real-World Scenarios
- Incorporate Feedback Loops

LLM ISSUES

Difficult to Steer
- Specialized Prompts
- Hierarchical Design
- Fine-Tuning Models

High Cost of Running
- Reduce Context Size
- Use Smaller Models
- Cloud-Based Solutions

Planning Failures
- Task Decomposition
- Multi-Plan Selection
- Reflection and Refinement

Reasoning Failures
- Enhance Reasoning Capabilities
- Fine-Tune LLMs with Feedback
- Use Specialized Agents

Tool Calling Failures
- Define Clear Parameters
- Validate Tool Outputs
- Tool Selectio VerificationLoops

PRODUCTION ISSUES

Guardrails
- Rule-Based Filters & Validation
- Human-in-the-Loop Oversight
- Ethical & Compliance Frameworks

Agent Scaling
- Scalable Architectures
- Resource Management
- Monitor Performance

Fault Tolerance
- Redundancy
- Automated Recovery
- Stateful Recovery

Infinite Looping
- Clear Termination Conditions
- Enhance Reasoning & Planning
- Monitor Agent Behavior

- Key failure points in AI agent development
- Evaluation and debugging challenges
- Performance, cost, and ethical concerns
- Practical solutions to improve AI agent reliability

Development Issues

Poorly Defined Task or Persona

Evaluation Issues

Development Issues - Poor Persona

A well-defined task or persona is essential for effectively operating your AI agents. Without it, agents may struggle to make appropriate decisions, leading to suboptimal performance.

Define Clear Objectives

You should specify the goals, constraints, and expected outcomes for each agent.

Craft Detailed Personas

Develop personas that outline the agents role, responsibilities, and behavior for you

Prompting

Use research-backed prompting techniques to reduce hallucinations for your agents

Development Issues - Evaluation Issues

> Evaluation helps you identify weaknesses and ensures your agents operate reliably in dynamic environments. Unlike traditional software, agents operate in dynamic environments which make it difficult to establish clear metrics for success

Continuous Evaluation

Implement an ongoing evaluation system to assess your agents performance and identify areas for improvement

Use Real-World Scenarios

Test your agents in real-world scenarios to understand their performance in dynamic environments

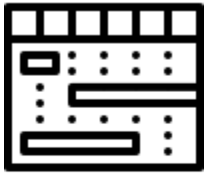
Feedback Loops

Incorporate feedback loops to allow for continuous improvement based on performance data

LLM Issues

Difficult to Steer

High Cost of Running



Planning Failures

Reasoning Failures

Tool Calling Failures

LLM Issues - Difficult to Steer

> You can steer LLMs towards specific tasks or goals for consistent and reliable performance. LLMs are influenced by vast amounts of training data which can lead to unpredictable behavior, and fine-tuning them for specific tasks require expertise and compute

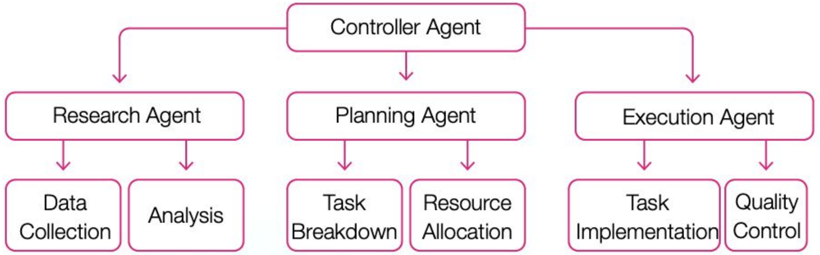


Fig 5.1: Hierarchical design with specialized agents performing specific tasks

Specialized Prompts - Use specialized prompts to guide the LLM toward specific tasks

Hierarchical Design - Implement a hierarchical design where specialized agents handle specific tasks, reducing the complexity of steering a single agent

Fine-Tuning - Continuously fine-tune the LLM based on task-specific data to improve performance

LLM Issues - High Cost of Running

Running LLMs, especially in production environments can be prohibitively expensive. This makes it difficult for organizations to scale their agent deployments cost-effectively.

Reduce Context - Introduce mechanisms to use as low context as possible to reduce the tokens

Use Smaller Models - Where possible, use smaller models or distill larger models to reduce costs

Cloud Solutions - Use cloud-based solutions to manage and scale computational resources efficiently

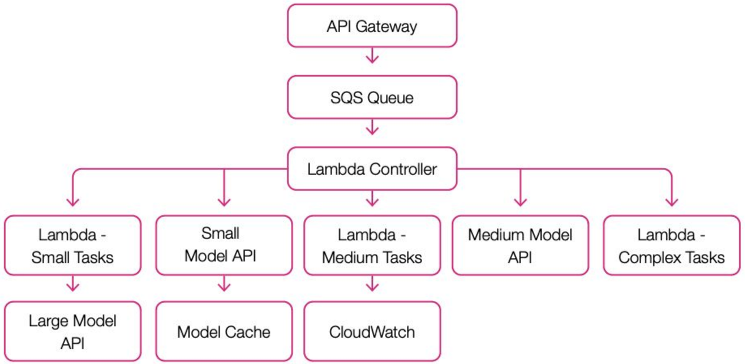


Fig 5.2: A serverless architecture where Lambda Controller makes intelligent decisions about request handling

Components of **Fig 5.2**

- The **SQS Queue** acts as our request buffer.
- The **Lambda Controller** makes intelligent decisions about request handling.
- **Small Model API** for simple completions and basic tasks
- **Medium Model API** for moderate complexity tasks
- **Large Model API** for complex reasoning tasks
- **Model Cache** for storing frequently used responses to reduce API calls
- **CloudWatch** to monitor system health and costs

LLM Issues - Planning Failures

Planning enables agents to anticipate future states, make informed decisions, and execute tasks in a structured manner. However, LLMs often struggle with planning, as it requires strong reasoning abilities.

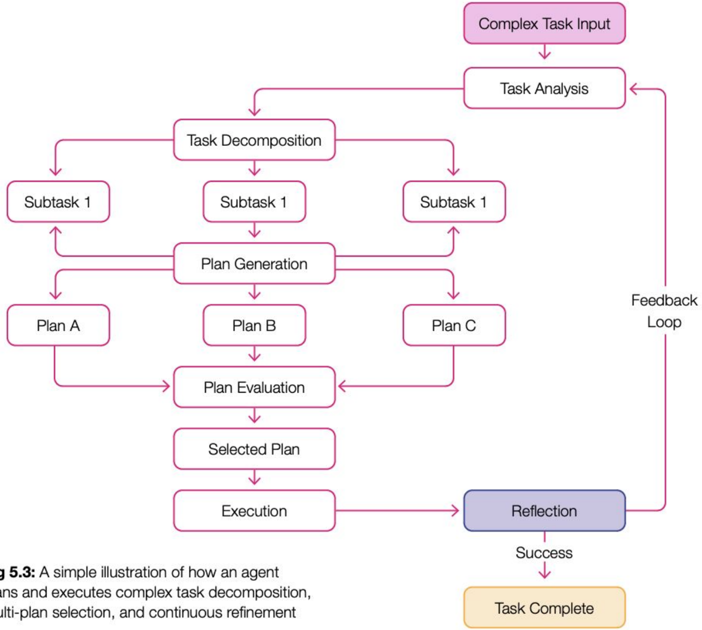


Fig 5.3: A simple illustration of how an agent plans and executes complex task decomposition, multi-plan selection, and continuous refinement

Task Decomposition - Break down tasks into smaller, manageable subtasks

Multi-Plan Selection - Generate multiple plan and select the most appropriate one based on the context

Reflection and Refinement - Continuously refine plans based on new information and feedback, and scale computational resources efficiently

LLM Issues - Reasoning Failures

> Reasoning is a fundamental capability that enables agents to make decisions, solve problems, and understand complex environments. LLMs lacking strong reasoning skills may struggle with tasks that require multi-step logic or nuanced judgement.

Enhance Reasoning Capabilities - Use prompting techniques like Reflexion to enhance the reasoning capabilities

Finetune LLM - Establish training with data generated with a human in the loop

Use Specialized Agents - Develop specialized agents that focus on specific reasoning tasks to improve overall performance

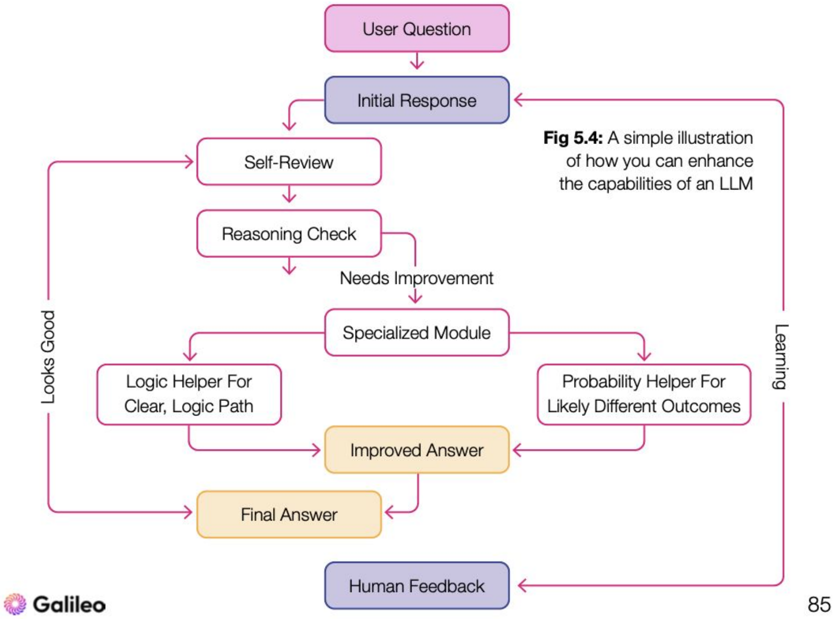


Fig 5.4: A simple illustration of how you can enhance the capabilities of an LLM

LLM Issues - Tool Calling Failures

Robust tool calling mechanisms ensure agents can perform complex tasks by leveraging various tools accurately and efficiently. Tool calling failures can occur due to incorrect parameter passing, misinterpretation of tool outputs, or failures in integrating tool results into the agent's workflow.

Define Clear Parameters

Ensure that tools have well-defined parameters and usage guidelines for you

Validate Tool Outputs

Implement validation checks to ensure that tool outputs are accurate and relevant

Tool Selection Verification

Use a verification layer to check if the tool selected is correct for the job

Production Issues

Guardrails

Agent Scaling

Fault Tolerance

Infinite Looping

Production Issues - Guardrails

Guardrails help ensure that agents adhere to safety protocols and regulatory requirements. Guardrails define the operational limits within which agents can function.

Rule-Based Filters & Validation

- Use predefined rules to filter offensive, harmful, or inappropriate content.
- Before processing, inputs received by the agent must be validated to meet criteria

Human-in-the-Loop Oversight

- Allow humans to provide feedback on the agent's performance and outputs
- Establish protocols for escalating complex or sensitive tasks to human operators

Ethical & Compliance Frameworks

- Establish ethical guidelines that outline the principles and values the agent must adhere to
- Implement compliance checks to ensure the agents actions align with regulation

Production Issues - Agent Scaling

Scaling agents to handle increased workloads or more complex tasks is a significant challenge. As the number of agents or the complexity of interactions grows, the system must efficiently manage resources, maintain performance, and ensure reliability.

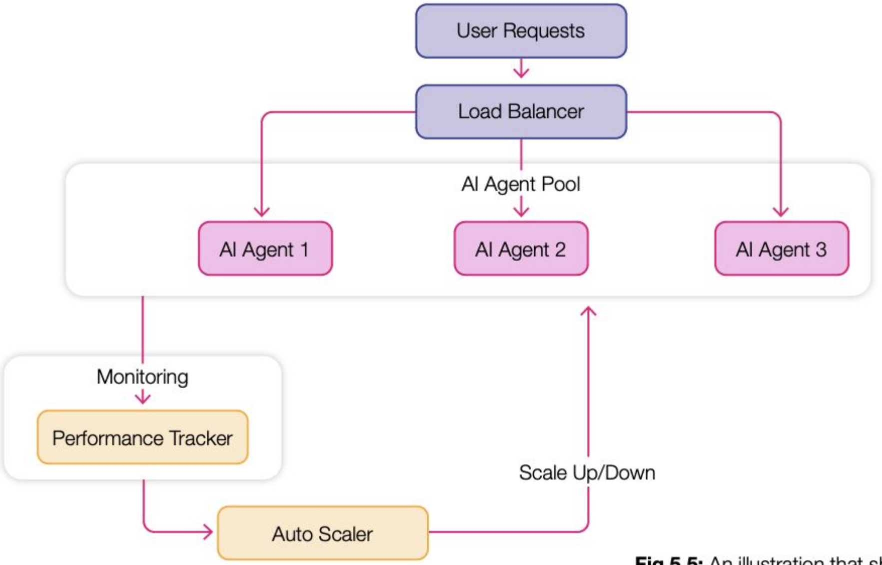


Fig 5.5: An illustration that shows how you can add monitoring and load balancers for easy scale-up and down

Scalable Architectures - Implement a microservice architecture where each agent or group of agents operates as an independent service

Resource Management - Integrate load balancers to distribute incoming requests evenly across multiple agents

Monitor Performance - Implement real-time monitoring tools to track each agent's performance

Production Issues - Fault Tolerance

AI agents need to be fault-tolerant to ensure that they can recover from eros and continue operating effectively. Without robust fault tolerance mechanisms, agents may fail to handle unexpected situations, leading to system crashes or degraded performance.

Redundancy - Deploy multiple instances of AI agents running in parallel

Automated Recovery - Incorporate intelligent retry mechanisms that automatically attempt to recover from transient errors

Stateful Recovery - Ensure the AI agents can recover their state after a failure

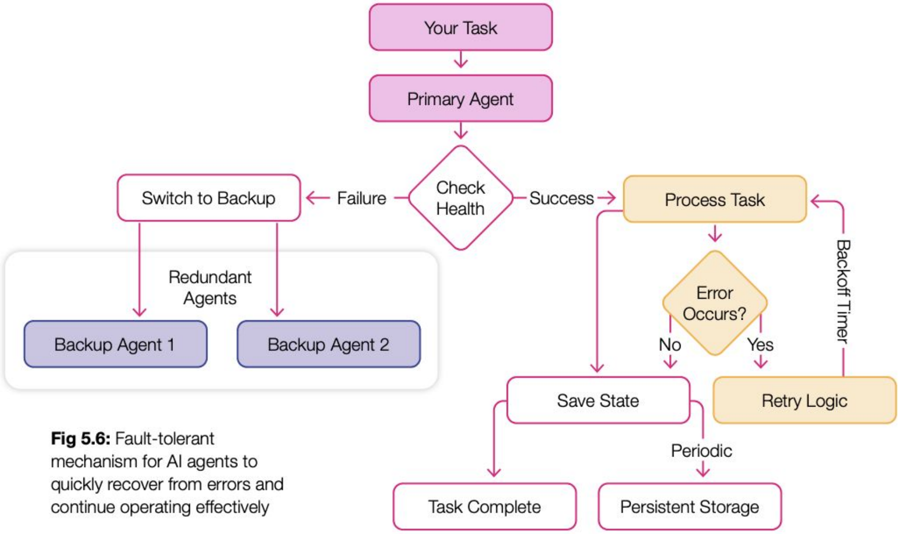


Fig 5.6: Fault-tolerant mechanism for AI agents to quickly recover from errors and continue operating effectively

Production Issues - Infinite Looping

> Looping mechanisms are essential for agents to perform iterative tasks and refine their actions based on feedback. Agents can sometimes get stuck in loops, repeatedly performing the same actions without progressing toward their goals.

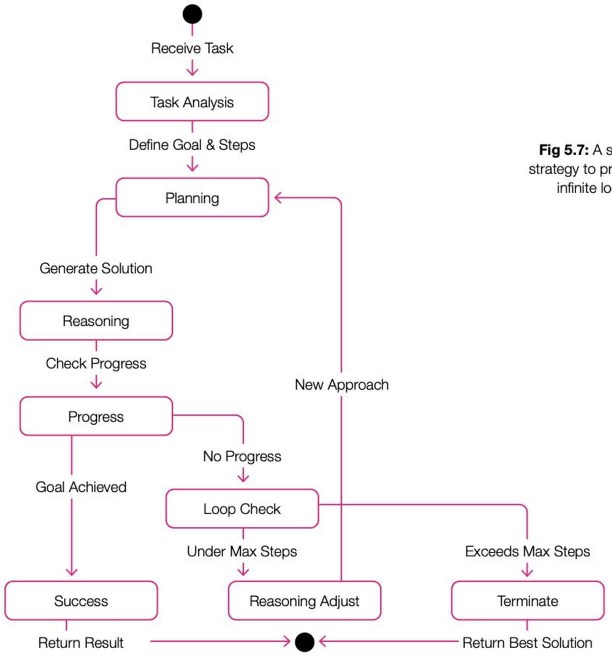


Fig 5.7: A simple strategy to prevent infinite looping

Clear Termination Conditions - Implement clear criteria for success and mechanisms to break out of loops

Enhance Reasoning and Planning - Improve the agents reasoning and planning capabilities to prevent infinite looping

Monitor Agent Behavior - Monitor agent behavior and adjust to prevent looping issues

Key Takeaways

- **Better task structuring and decomposition**
- **Continuous evaluation and monitoring**
- **Resource optimization and caching**
- **Security and compliance frameworks**

Paper Reference

Questions?