UNIVERSITY of VIRGINIA

AI Agents 2024 Rewind: A Year of Building and Learning

Presented by:
Nina Chinnam (fhs9af) Mihika Rao (xsw5kn)

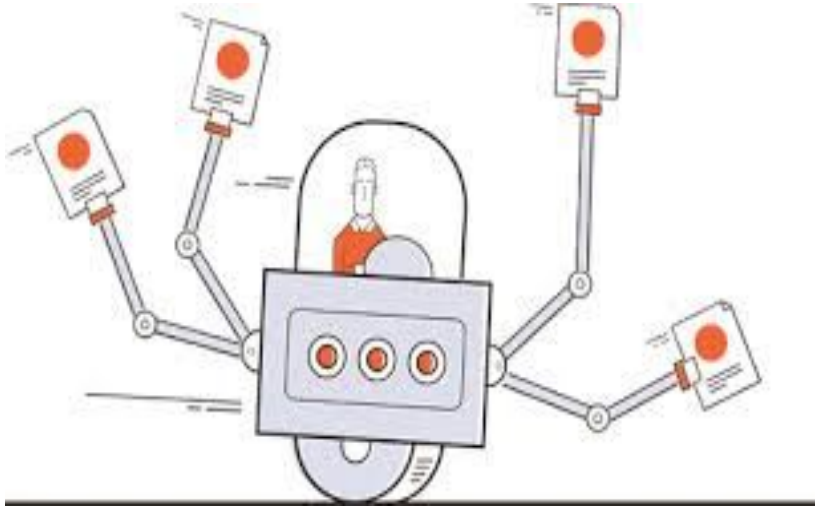
Mihika Rao (xsw5kn)

Presentation Outline

- Enterprise Adoption of Agents in Products
- “Agent Native” Foundation Models
- Interface Agents Take Center Stage
- Shift to Complex Tasks and Rise of Frameworks
- Realizations from Benchmarks
- Looking Forward to 2025

Enterprise Adoption of Agents in Products .. With Caveats

Enterprise Adoption of AI Agents - Trends and Challenges 2024



Growing Adoption of AI Agents in Enterprises

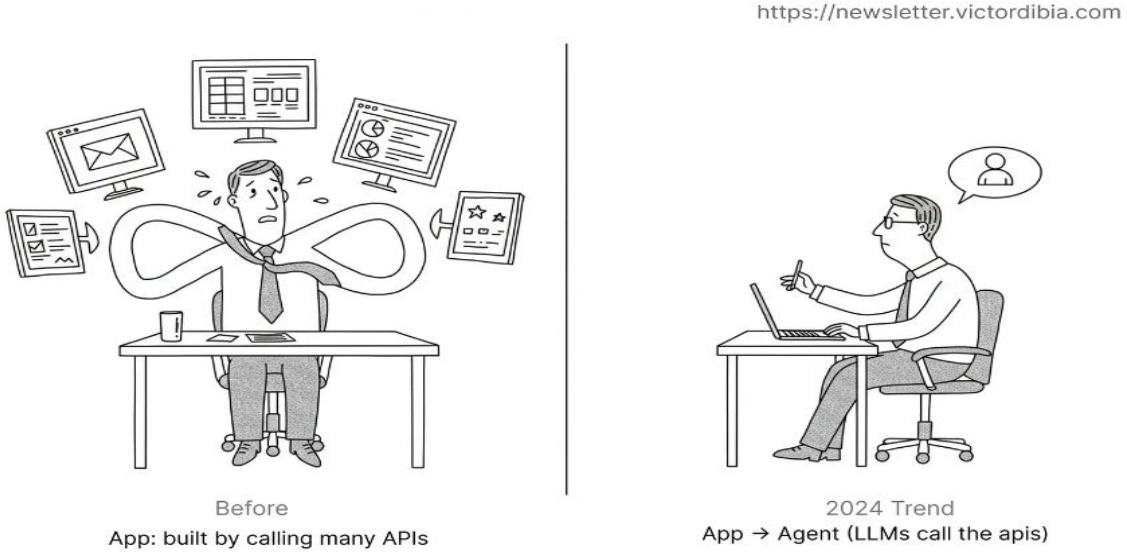
Microsoft Copilot

Google Workspace

Company	AI Agent System	Functionality	Limitations
Microsoft	Copilot Agents	Automates document writing, summarization, and Excel analysis	Still needs human review for accuracy
Salesforce	Agentforce	Enhances CRM tasks, customer support automation	Lacks deep decision-making abilities
Google	Workspace AI	Helps with email drafts, meeting setups, and documentation	Primarily text-based, no complex workflow automation
SAP AI Assistants	Business Process Automation	AI-driven insights in enterprise resource planning	Limited to predefined functions
OpenAI API Adoption	Embedded into various SaaS products	Automates text-based tasks	Does not perform complex decision-making

AI Agents as Orchestration Layers: Key Caveat

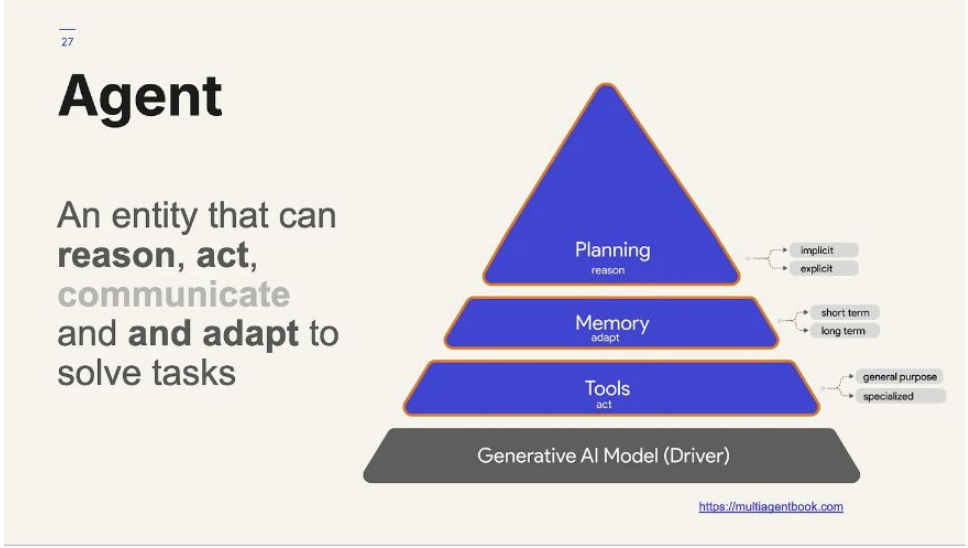
- AI Agents enhance efficiency but don’t replace human workflows
- Not fully autonomous AI agent - these agents don’t decide which tasks need to be done without being explicitly told

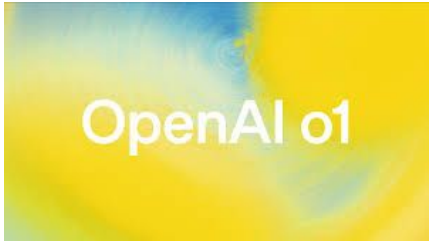


“Agent-Native” Foundation Models

What are Agent-Native Foundation Models

- Designed specifically for AI Agents
- Handle long-term memory & planning
- Enable autonomous task execution



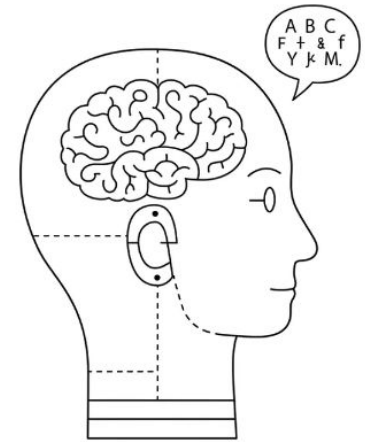


Key Features of Agent-Native Models

Feature	Traditional LLMs (e.g., GPT-4, Claude)	Agent-Native Models (o1, Gemini 2.0, OpenAgent)
Response Generation	One-time text-based outputs	Plans multi-step tasks autonomously
Memory & Context	Limited, forgets past conversations	Retains long-term memory for better decision-making
Tool & API Integration	Requires external plugins	Built-in ability to call APIs & use external tools
Autonomy	Requires human input for every step	Can act independently after receiving an initial command

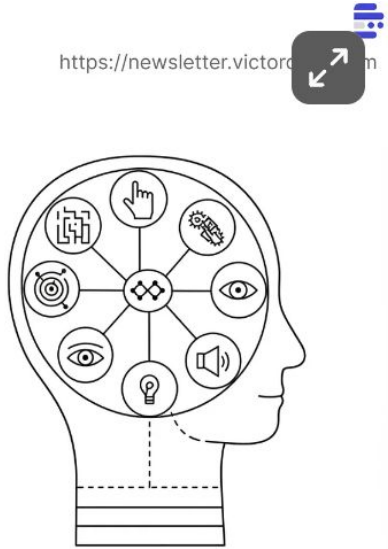
Shift Towards Agent-Native

- **Before**: Language modeling - generating text-based responses
  - Can’t plan or break down complex tasks
  - Could not use tools without external help
- **Trend**: Capabilities are built in.
  - Have memory for long-term interactions
  - Can interact with APIs
  - Better for autonomous AI agents



Before

Models focused on language modeling



2024 Trend

More capabilities (reasoning, planning, tool use etc) “lifted” into Models

Interface Agents Take Center Stage

What are Interface Agents?

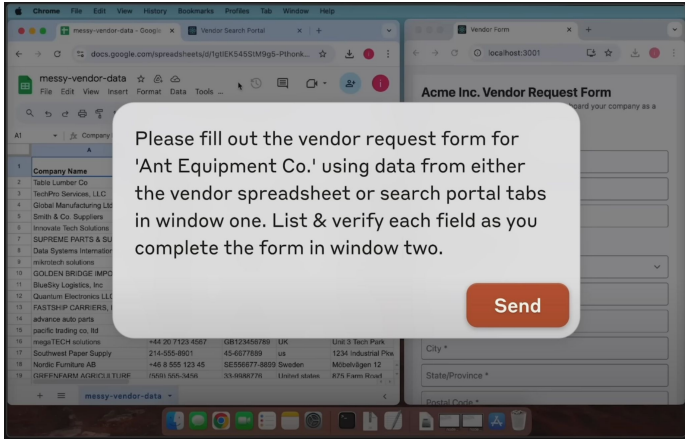
- Interface agents can control and interact with UIs
- Automation techniques
- Real-world actions
- Ex: AutoGPT, Adept ACT-1, and OpenAI's Code Interpreter

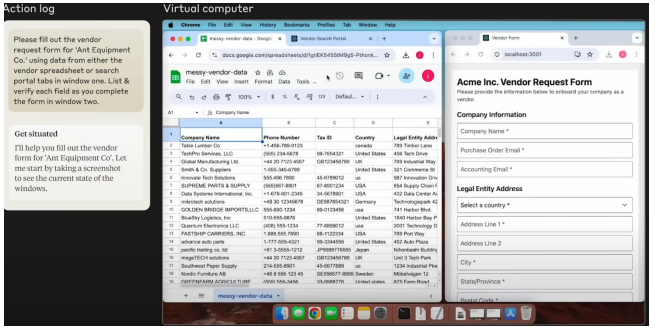
Real-World Examples of Interface Agents

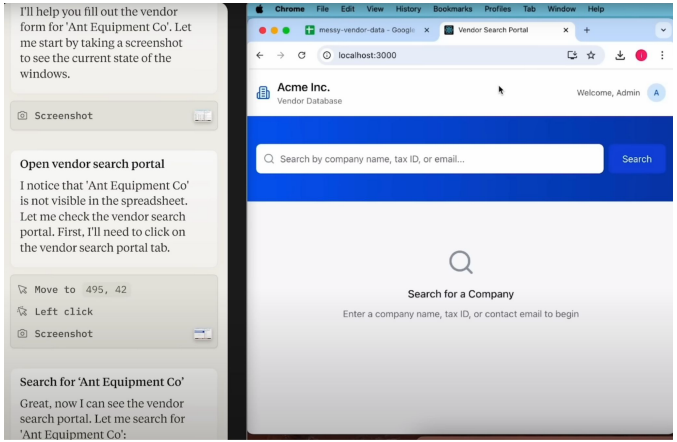
- **Kura AI** - AI Agents for Web Automation
- **Microsoft OmniParser** - AI for GUI Interactions
- **AutoGPT WebSurfer Agent**

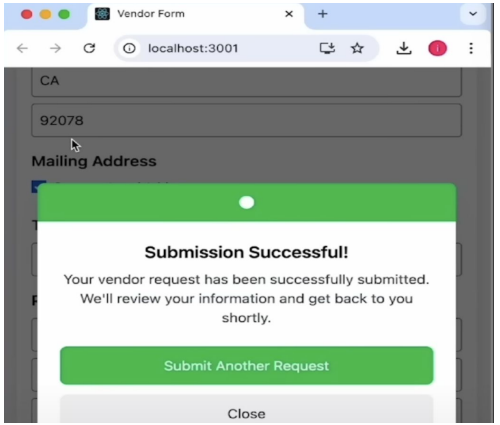
```
driver.get("https://www.google.com")
search_box = driver.find_element(By.NAME, "q")
search_box.send_keys("Latest AI research papers")
search_box.submit()

first_result = driver.find_element(By.XPATH, "//h3")
print("Top Result:", first_result.text)

client = OpenAI(api_key="YOUR_API_KEY")
response = client.ChatCompletion.create(
    model="gpt-4",
    messages=[{"role": "system", "content":
f"Summarize this article: {first_result.text}"}]
)

print("Summary:",
response["choices"][0]["message"]["content"])

driver.quit()
```

Anthropic Claude Computer Use





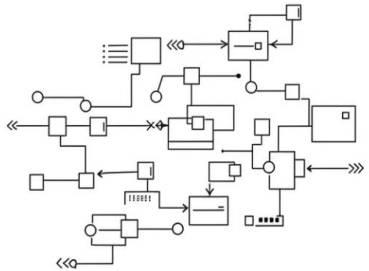




Shift Towards Interface Agents

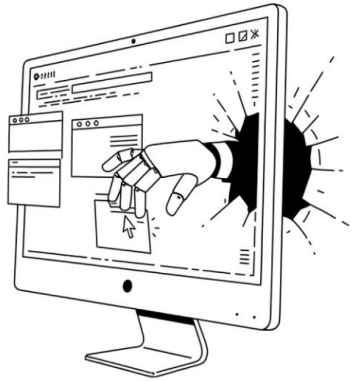
- **Before**: Programmatic Tools & Code Execution
- **Trend**: AI Agents Now Directly Manipulate User Interfaces

https://newsletter.victordibia.com



Before

Agents act via code (functions/apis)



2024 Trend

Agents act by driving interfaces (web, OS etc)

Nina Chinnam (fhs9af)

Presentation Outline

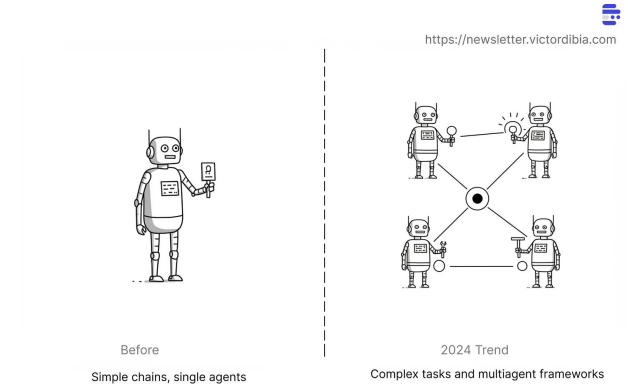
- Enterprise Adoption of Agents in Products
- “Agent Native” Foundation Models
- Interface Agents Take Center Stage
- Shift to Complex Tasks and Rise of Frameworks
- Realizations from Benchmarks
- Looking Forward to 2025

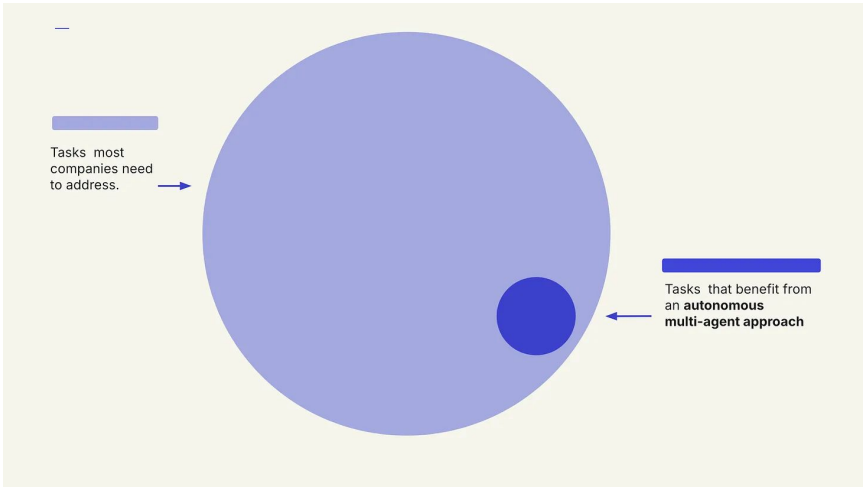
Presentation Outline

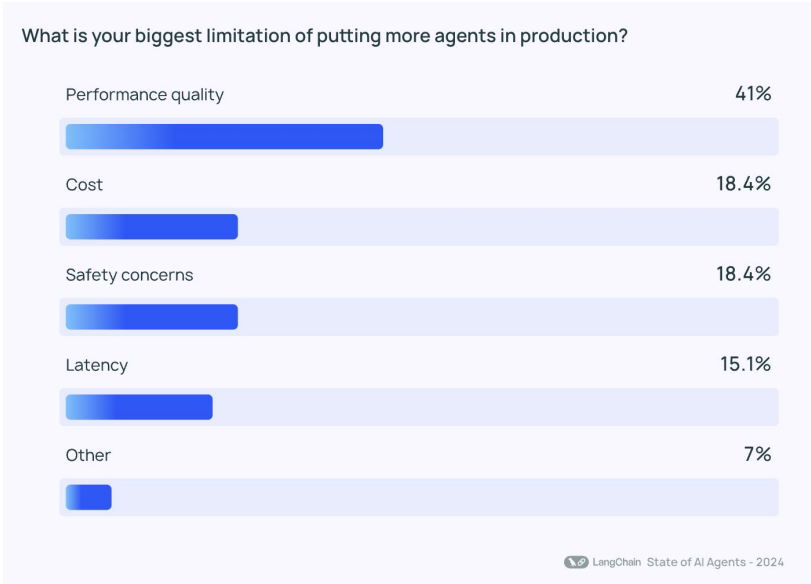
- Enterprise Adoption of Agents in Products
- “Agent Native” Foundation Models
- Interface Agents Take Center Stage
- Shift to Complex Tasks and Rise of Frameworks
- Realizations from Benchmarks
- Looking Forward to 2025

Shift to Complex Tasks and Rise of Frameworks

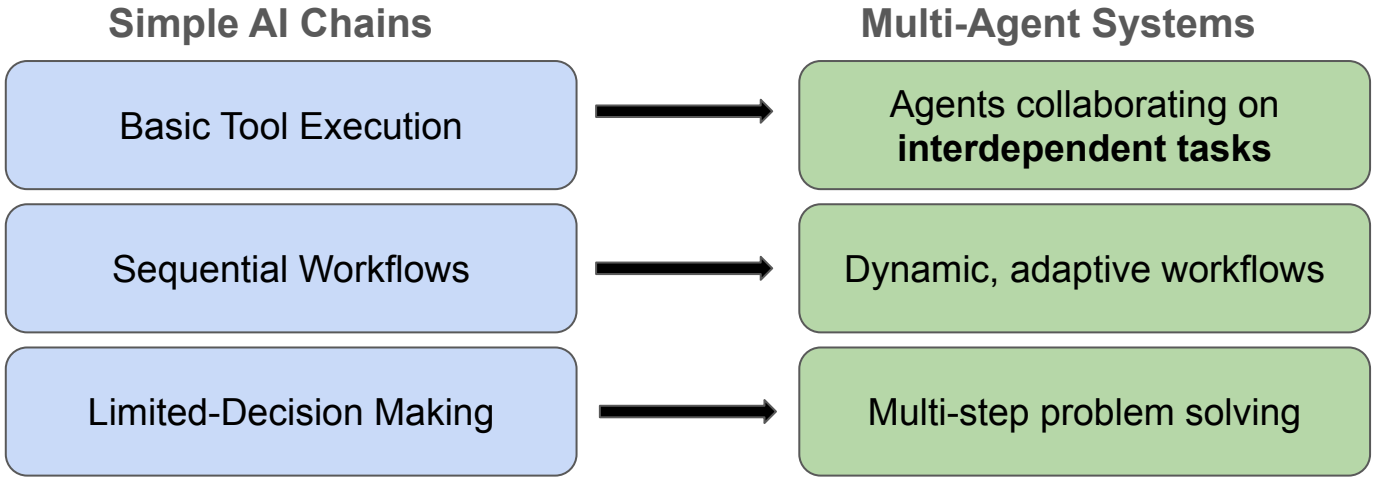
The Evolution of AI Agents







Shift To Complex Tasks: Transition to Complexity



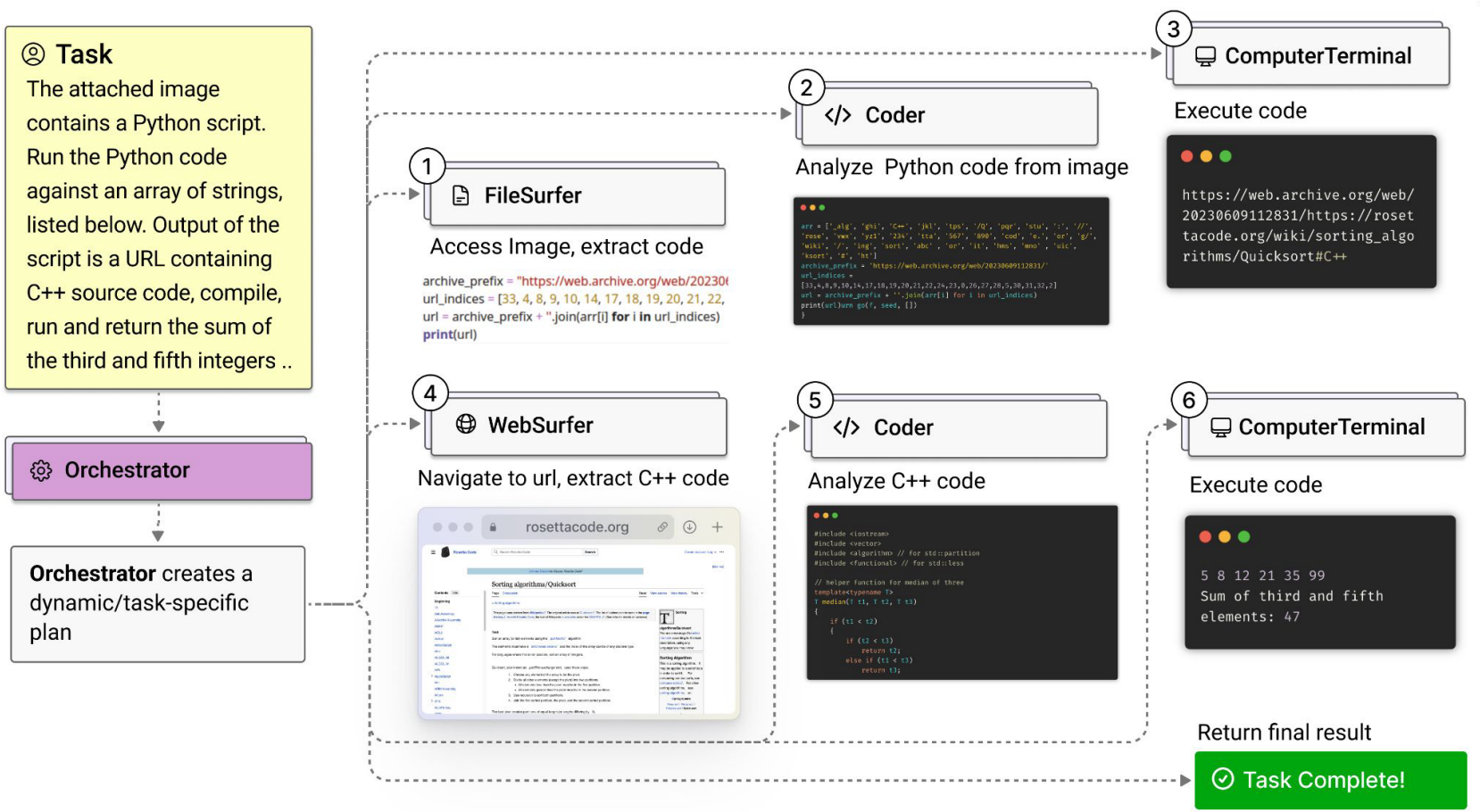
Challenge: Selecting the right Multi-Agent Pattern

- **Branching logic**
- **Reflection**
- **Metacognition**

Solution: Multi-Agent Frameworks

Framework	Key Features
Autogen	open-source framework for building AI agent systems
Magnetic One	**General-purpose agentic system** using an orchestrator model
Autogen Studio	**No-code tool** for working with multi-agentic frameworks
LangGraph	**Graph-based** for structured workflows
OpenAI Swarm	**Lightweight framework** focused on multi-agent collaboration and orchestration
CrewAI	Multi-agent, **role-based** collaboration
Pydantic AI	Develop **production-grade** AI applications through **Python**

Multi-Agent Frameworks: Magnetic One



Multi-Agent Frameworks: Autogen Studio

Multi-Agent Frameworks: OpenAI Swarm

Problem: User needs real-time weather updates and wants system to automatically send alerts via email

```
from swarm import Swarm, Agent

# Define agent functions
def get_weather(location, time="now"):
    # Function to get weather information
    return f"The weather in {location} at {time} is sunny."
def send_email(recipient, subject, body):
    # Function to send an email
    return "Email sent successfully."
# Define agents
weather_agent = Agent(
    name="Weather Agent",
    instructions="Provide weather updates.",
    functions=[get_weather, send_email],
)
# Initialise Swarm client and run conversation
client = Swarm()
response = client.run(
    agent=weather_agent,
    messages=[{"role": "user", "content": "What's the weather in New York?"}],
)
print(response.messages[-1]["content"])
```

Solution:

1. Defining Specialized Agent Functions
2. Creating an AI Weather Agent
3. Run the Agent
4. Display Response

Key Features of OpenAI Swarm

- seamless AI task delegation
- scalable, efficient, and modular

Multi-Agent Frameworks: Pydantic AI

Problem: Bank wants to automate its customer support

```python
@dataclass
class SupportDependencies:
    customer_id: int
    db: DatabaseConn


class SupportResult(BaseModel):
    support_advice: str = Field(description='Advice returned to the customer')
    block_card: bool = Field(description="Whether to block the customer's card")
    risk: int = Field(description='Risk level of query', ge=0, le=10)

support_agent = Agent(
    'openai:gpt-4o',
    deps_type=SupportDependencies,
    result_type=SupportResult,
    system_prompt=(
        'You are a support agent in our bank, give the '
        'customer support and judge the risk level of their query.'
    ),
)
@support_agent.system_prompt
async def add_customer_name(ctx: RunContext[SupportDependencies]) -> str:
    customer_name = await ctx.deps.db.customer_name(id=ctx.deps.customer_id)
    return f"The customer's name is {customer_name!r}"


@support_agent.tool
async def customer_balance(
    ctx: RunContext[SupportDependencies], include_pending: bool
) -> float:
    """Returns the customer's current account balance."""
    return await ctx.deps.db.customer_balance(
        id=ctx.deps.customer_id,
        include_pending=include_pending,
    )
```

Solution:

1. Create dependencies
2. Define structured AI output
3. Create AI Agent
4. Add context awareness
5. Define AI Balance Tool

Key Features of Pydantic AI

- Ideal for structured, production-grade AI application
- Prevents unpredictable AI behavior through enforcing strict validation.

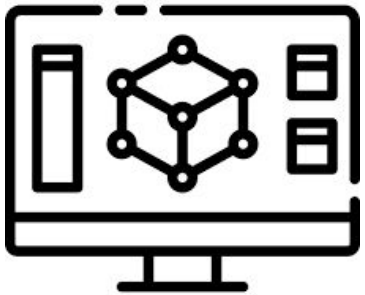
Realizations from Benchmarks

Key Benchmarks introduced in 2024

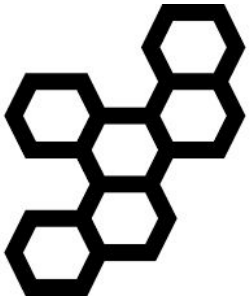
Benchmark	Purpose	Findings
CORE-Bench	Measures computational reproducibility in AI-generated outputs	AI models struggle with consistent output generation across different runs
WebArena	Evaluates web-based task completion	AI agents complete structured tasks well but fail at adaptive reasoning
Windows Agent Arena	Tests AI capabilities within desktop environments	Low success rates for multi-step workflows
ARC-AGI Benchmark	Measures general intelligence against human cognition	OpenAI's o3 model scored 87.5, nearing the human benchmark of 85

Looking Forward into 2025

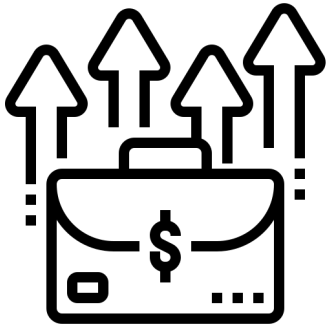
Future of AI Agents



More improvements in models



Patterns drive reliability



Agent Marketplace

UNIVERSITY of VIRGINIA

Agent Codebases

Presented by:
Anisha Patrikar (gjq2yf) and Yagnik Panguluri (yye7pm)

Anisha Patrikar (gjq2yf)

Presentation Outline

- Introduction of Codebases
- Why are they important?
- LangGraph
- Hugging Face Transformer Agents
- AG2
- OpenAI Operator
- Comparison of Codebases

Overview of Codebases

LangGraph

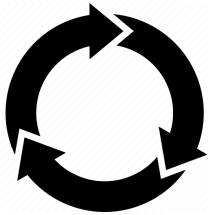
Hugging Face

OpenAI Operator

High Level Comparison

Feature	LangGraph	Transformer Agents	AG2	OpenAI Operator
Primary Purpose	Multi-agent workflows in LangChain	Single-agent AI with tool support	Multi-agent coordination	Managing OpenAI API workloads
Workflow Type	Graph-based	Linear	Dynamic AI collaboration	Containerized execution
Multi-Agent Support	Yes	No	Yes	No
Tool Use (e.g., search, code	No	Yes	Yes	No
Scalability	High	Moderate	High	High
Ease of Use	Requires setup	Easy	Complex	Easy

Why Are They Important?



Automate workflows with intelligent agents

Enable LLM-powered reasoning and collaboration

Scale AI applications across industries

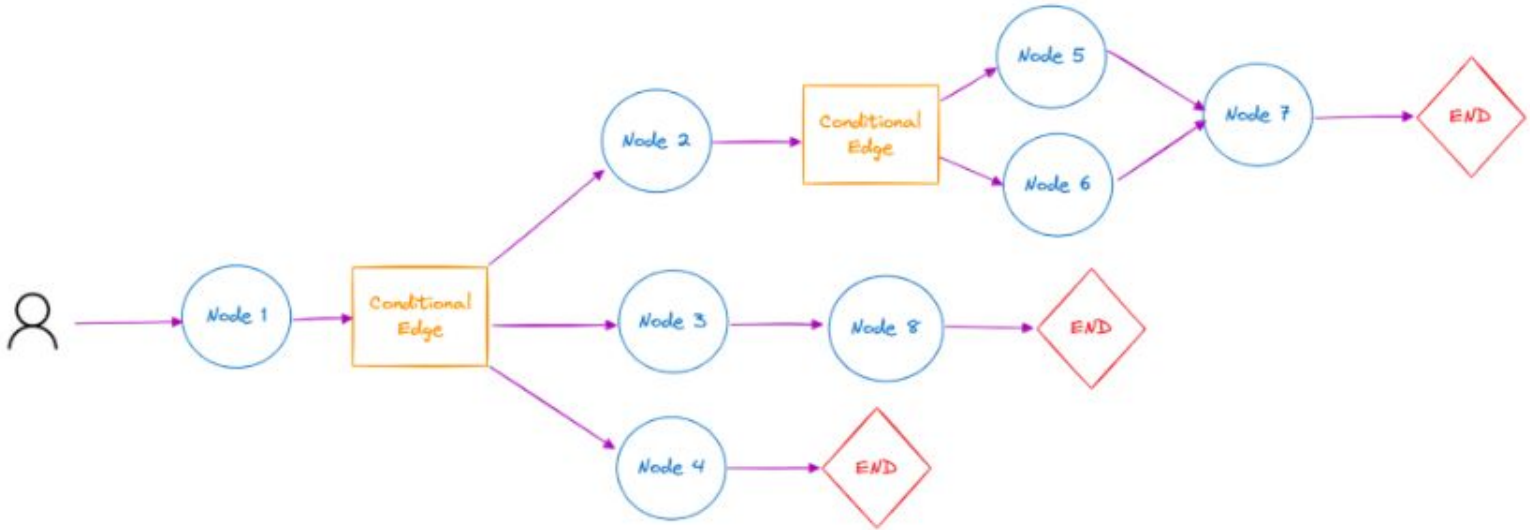
LangGraph - Purpose

LangGraph

- Framework for building multi-agent workflows
- Uses graph-based execution to define interactions between AI components
- **Ideal Use Case:** structured, modular, and scalable AI systems

LangGraph - Key Concepts

- **StateGraph:** Defines AI interactions
- **Nodes:** Individual Steps in workflow
- **Edges:** Define execution order
- **State Management:** Tracks messages between nodes



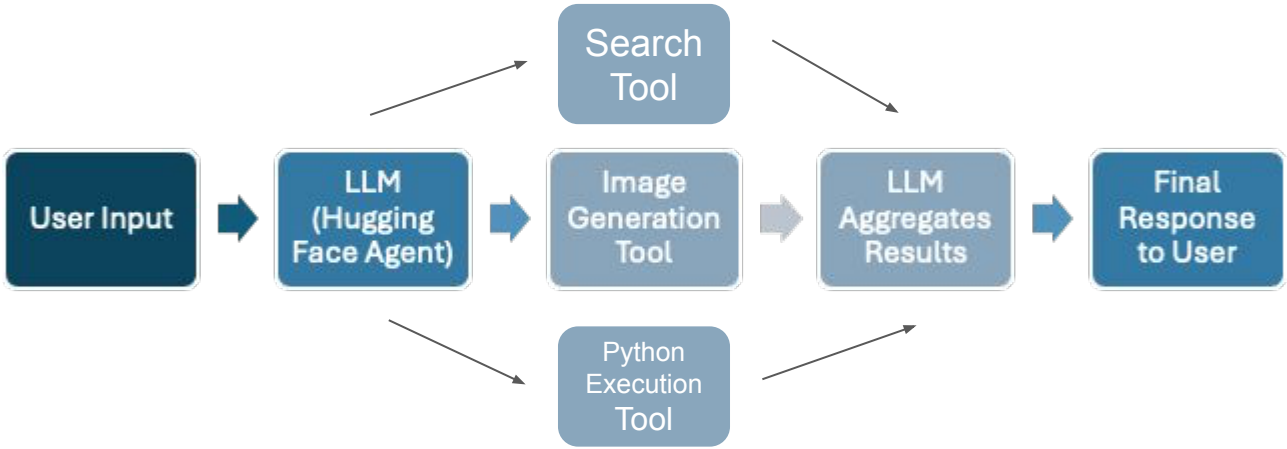
LangGraph Example

```python
# Define the message structure
class State(TypedDict):
    messages: list

# Initialize the LangGraph StateGraph
graph_builder = StateGraph(State)

# Set up the OpenAI API connection (replace with your actual API key)
llm = ChatOpenAI(
    openai_api_key=os.getenv("OPENAI_API_KEY"),
    model="gpt-4"
)

# Define chatbot function
def chatbot(state: State):
    response = llm.invoke(state["messages"])
    return {"messages": [response]}

# Add chatbot node to the graph
graph_builder.add_node("chatbot", chatbot)
graph_builder.add_edge(START, "chatbot")
graph_builder.add_edge("chatbot", END)

# Compile and run the graph
graph = graph_builder.compile()

# Example user interaction
user_input = "Hello, how can I assist you today?"
for event in graph.stream({"messages": [("user", user_input)]}):
    for value in event.values():
        print("Assistant:", value["messages"][-1].content)
```

Transformer Agents - Purpose

Hugging Face

- Allows LLMs to use tools (Python execution, search, image generation)
- Built on Hugging Face's Inference API
- Good for single-agent AI reasoning



Transformer Agents - Key Features

- Natural Language API
- Multimodal Capabilities (text, code, images)
- Agent-Oriented Execution

Available Tools for Transformer Agents

Tool	Functionality
Python Execution	Runs Python code dynamically and returns results
Image Generation	Generates images using Stable Diffusion models
Document Q&A	Answers questions based on a given document
Web Search	Fetches relevant information from the web
Text-to-Speech	Converts text into speech using AI voices
Translation	Translates text into different languages

Transformer Agents - Example

```python
from transformers import HfAgent

agent = HfAgent("https://api-inference.huggingface.co/models/bigcode/starcoder")

response = agent.run("Generate a Python function to calculate the factorial of a number.")
print(response)
```

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

print(factorial(5))
```

Yagnik Panguluri (yye7pm)

AG2 and OpenAI Operator

Autogen

A framework for multi-agent coordination where AI agents collaborate to solve tasks efficiently

Operator

A Kubernetes-based solution for managing OpenAI models at scale

AG2

In traditional AI workflows, a single model is responsible for completing a task. However, real-world problems are complex and often require a distributed intelligence approach.

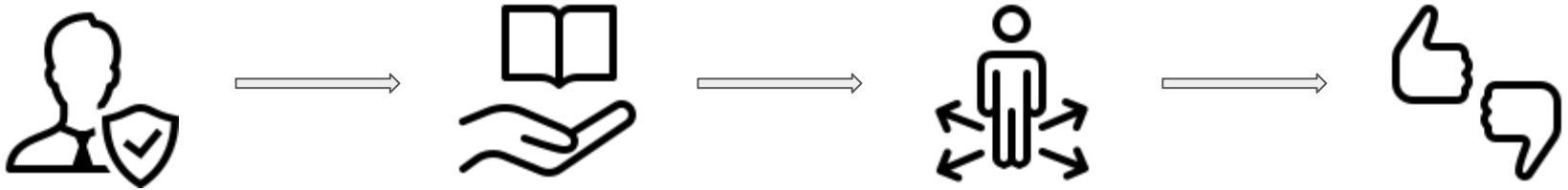
- **Decentralized decision-making -** Agents operate independently but coordinate when needed
- **Scalability-** The framework support hundreds or even thousands of agents working together
- **Modularity -** Different agents can be designed for different sub-tasks, making the system more flexible
- **Efficiency -** Collaboration leads to faster problem solving and improved AI performance

Autonomous Research Agents

Real-Time AI Teamwork

AI-Assisted Decision Making

How AG2 Works



Agent Initialization

Communication and Knowledge Sharing

Decision-Making and Execution

Feedback and Adaptation

AG2 - Code

```python
from autogen import ConversableAgent

# Create an AI agent
assistant = ConversableAgent(
    name="assistant",
    system_message="You are an assistant that responds concisely.",
    llm_config=llm_config
)

# Create another AI agent
fact_checker = ConversableAgent(
    name="fact_checker",
    system_message="You are a fact-checking assistant.",
    llm_config=llm_config
)

# Start the conversation
assistant.initiate_chat(
    recipient=fact_checker,
    message="What is AG2?",
    max_turns=2
)
```

AG2 - Code (Human in the Loop)

```python
from autogen import ConversableAgent

# Create an AI agent
assistant = ConversableAgent(
    name="assistant",
    system_message="You are a helpful assistant.",
    llm_config=llm_config
)

# Create a human agent with manual input mode
human = ConversableAgent(
    name="human",
    human_input_mode="ALWAYS"
)
# or
human = UserProxyAgent(name="human", code_execution_config={"work_dir": "coding", "use_docker"

# Start the chat
human.initiate_chat(
    recipient=assistant,
    message="Hello! What's 2 + 2?"
)
```

AG2 - Code (Multiple Operators)

```python
from autogen import ConversableAgent, GroupChat, GroupChatManager

# Create AI agents
teacher = ConversableAgent(name="teacher", system_message="You suggest lesson topics.")
planner = ConversableAgent(name="planner", system_message="You create lesson plans.")
reviewer = ConversableAgent(name="reviewer", system_message="You review lesson plans.")

# Create GroupChat
groupchat = GroupChat(agents=[teacher, planner, reviewer], speaker_selection_method="auto")

# Create the GroupChatManager, it will manage the conversation and uses an LLM to select the n
manager = GroupChatManager(name="manager", groupchat=groupchat)

# Start the conversation
teacher.initiate_chat(manager, "Create a lesson on photosynthesis.")
```

Comparison

Feature	LangGraph(LangChain)	HF Transformers Agents	AG2	OpenAI Operator
Purpose	Multi-agent workflows in LangChain	LLM-powered agents for reasoning	Multi-agent coordination & decision making	AI model deployment & scaling in Kubernetes
Core Functionality	Creates directed acyclic graphs (DAGs) for agent interactions	Enables Transformers-based agents to use external tools	Enables agents to collaborate and optimize decision-making	Deploys, scales, and manages AI models efficiently
Use Case Examples	Task automation, AI-driven workflows	AI-powered assistants, chatbots, agent-based automation	Financial analysis, robotics, multi-agent simulations	Chatbots, large-scale AI APIs, real-time analytics
Scalability	Supports chaining multiple agents but not inherently multi-agent	Designed for single-agent use cases	Supports thousands of AI agents collaborating in parallel	Dynamically scales AI workloads
Interaction with AI Models	Works well with LLMs but focuses on workflow structuring	Uses LLMs directly for intelligent task execution	Uses AI models to enhance multi-agent decision-making	Manages infrastructure to serve AI models at scale

Questions?