Introduction to Agents

Yogesh Haribhau Kulkarni

YHK

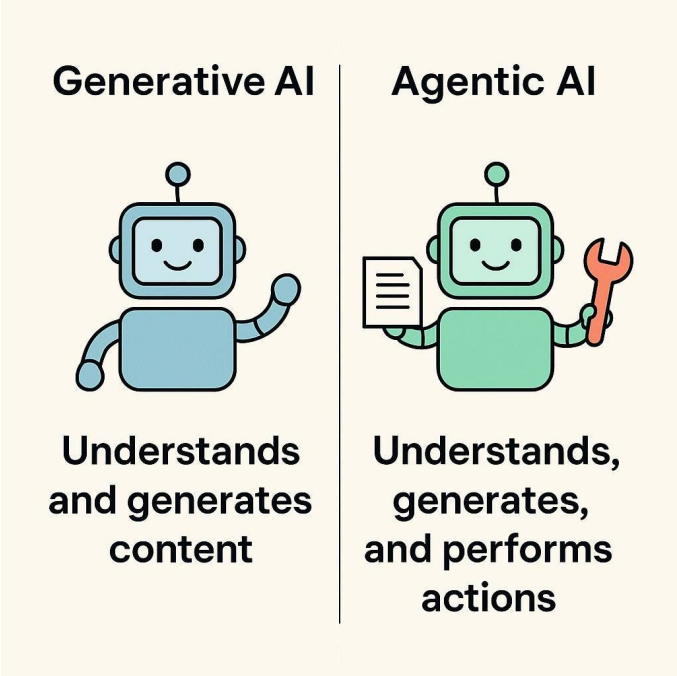
Outline

Introduction to AI Agents

What Even Is an AI Agent?

No widely accepted definition exists, but here's a practical one:

- **Generative AI:** Great at understanding and generating content
- **Agentic AI:** Goes further, understands, generates content, **and performs actions**



(Ref: Agentic AI For Everyone - Aish & Kiriti)

The key differentiator is the ability to **take action**, not just respond

YHK

The Evolution of AI Capabilities

- **Traditional Programming:** Needed code to operate
- **Traditional ML:** Needed feature engineering
- **Deep Learning:** Needed task-specific training
- **ChatGPT (2022):** Could reason across tasks without training
  - Zero-shot learning (no examples needed)
  - In-context learning (understands from instructions)
- **Agents (2024):** Can actually **do things**, not just talk

YHK

Why Does "Taking Action" Matter?

- In 2022, ChatGPT was revolutionary because AI felt conversational
- By 2024, people wanted more than conversation, they wanted **execution**
- Examples of what users now expect:
  - Instead of listing leads ? **email them directly**
  - Instead of summarizing docs ? **file and create workflow tasks**
  - Instead of suggesting products ? **customize landing pages**
- This shift from **information** to **action** defines the agent era

YHK

How Do Agents Take Action?

- The magic lies in **tools** and **function calling**
- Agents are paired with APIs, plugins, or external systems
- Instead of just text responses, LLMs output structured commands:
  - "Call the send_email() function with these inputs..."
  - "Fetch records from CRM using this query..."
  - "Schedule a meeting for Tuesday at 2PM..."
- **Mental model:** LLM = brain, Tools = hands
- Without tools, agents just talk. With tools, they act.

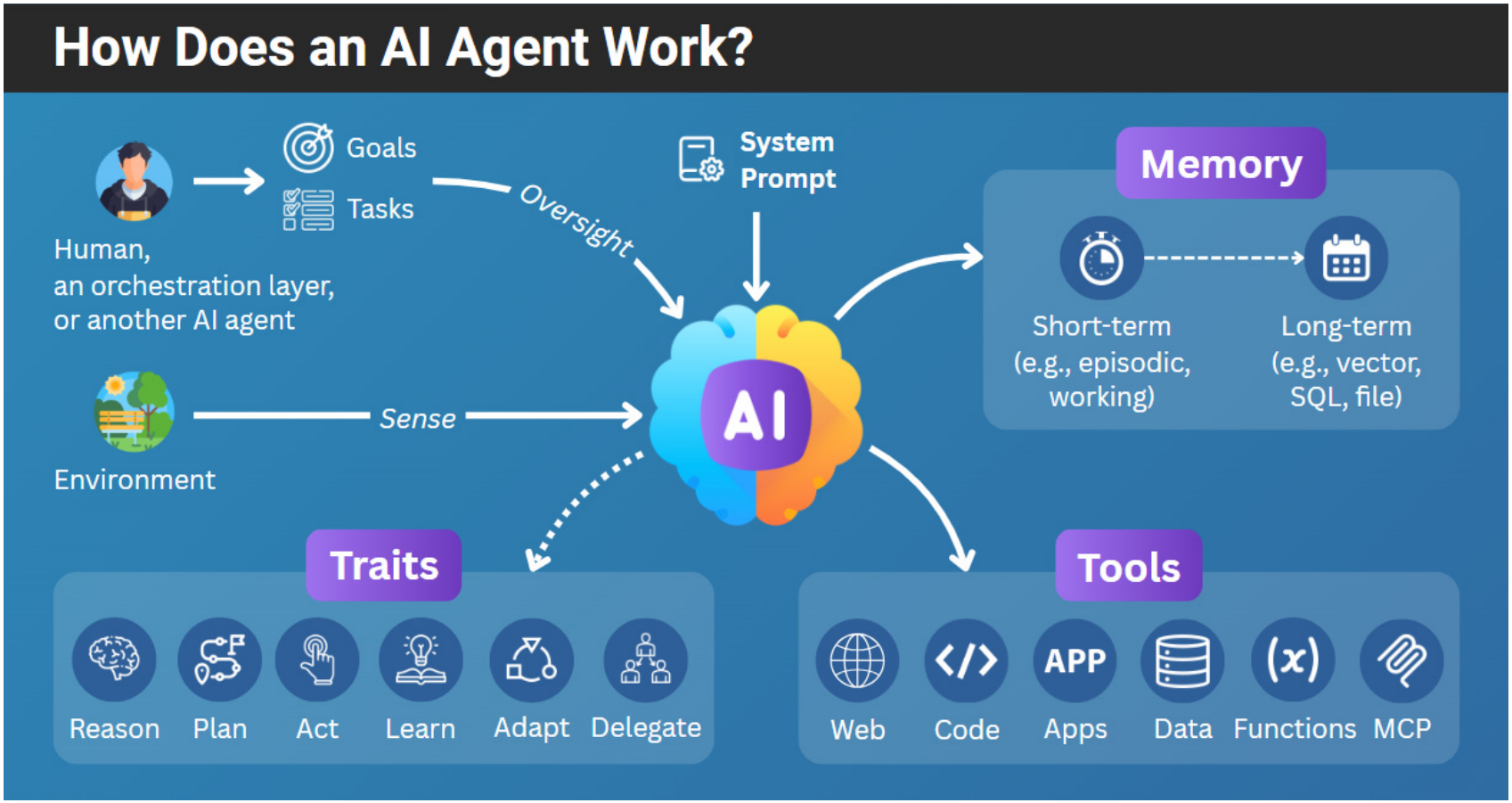
YHK

Future AI Applications

- What are future AI applications like?
  - **Generative:** Generate content like text and images
  - **Agentic:** Execute complex tasks on behalf of humans
- How do we empower every developer to build them?
  - **Co-Pilots:** Human-AI collaboration
  - **Autonomous:** Independent task execution
- 2024 is expected to be the year of AI agents

YHK

How AI Agent Works?



(Ref: The Ultimate Guide to AI Agents for PMs - Pawl Huryn)

What Is an Agent? (Technical Definition)

- Agent acts and takes you from one state to another, providing value through workflow automation
- Based on ReAct paradigm: **Reasoning + Acting**
- Key capabilities:
  - Can plan and make decisions
  - Has access to tools (search APIs, booking, email, etc.)
  - Interacts with external environments and other agents
  - Maintains memory of conversations and actions
  - May include human-in-the-loop for safety
- Agents existed since the 1950s but are now effective because of LLMs

YHK

Two Ways to Define Agents

Technical View:

- LLM (brain)
- + Tools (hands)
- + Planning (strategy)
- + Memory (context)
- + State management

Business View:

- Systems that complete tasks end-to-end
- Focus on outcomes, not components
- Solve real-world problems
- Provide measurable value

Important: Today's agents are **engineering wrappers** around AI models, the intelligence comes from the LLMs, agents help act on that intelligence.

YHK

Types of AI Agents

(Ref: Agentic AI For Everyone - Aish & Kiriti)

YHK

The Tool-Augmented LLM

- LLM is the brain of modern AI agents.
- Becomes agentic when paired with tools, planning, memory, and control logic.
- Enables reasoning, action-taking, and goal adaptation.
- Different autonomy levels define different agent types.

YHK

Rule-Based Systems

- No LLM, just if-this-then-that logic.
- Best for simple, structured, repetitive tasks.
- Examples: auto-approvals, file renaming, data copying.
- Pros: Fast, predictable, auditable.
- Cons: Brittle, lacks flexibility or reasoning.

YHK

Workflow Agents

- LLM assists but doesn't act independently.
- Human remains in control; agent augments tasks.
- Examples: email drafts, meeting summaries, query translation.
- Pros: Low risk, easy deployment, boosts productivity.
- Cons: No execution or planning capability.

YHK

Semi-Autonomous Agents

- LLM plans and executes with limited supervision.
- Supports multi-step, tedious workflows.
- Examples: CRM outreach, contract data entry, research reports.
- Pros: Saves time, automates complex tasks.
- Cons: Needs infra (memory, tools), harder to test.

YHK

Autonomous Agents

- Fully goal-driven, acts without oversight.
- Manages long-running, async, cross-system tasks.
- Examples: research briefs, ops monitoring, product testing.
- Pros: Highly scalable, handles complex goals.
- Cons: High risk, hard to monitor or trace.

YHK

Choosing the Right Agent Type

- Start with the problem, not the tech.
- Key questions: structure, language needs, decision steps, human involvement.
- Use autonomy level as a design guide.
- Mix agent types if needed within one system.

Problem Type	Suggested Agent Type	Enterprise Example
Structured, rules-based	Rule-Based System	Automatically approve reimbursements under $100 or trigger invoice processing based on fixed thresholds
Human-led process, needs speed	Workflow Agent	Suggesting first-draft responses to support tickets in Zendesk, or summarizing meeting transcripts for Slack follow-up
Multi-step, repeatable, bounded	Semi-Autonomous Agent	Lead enrichment and follow-up: fetch CRM info → draft outreach → send email → log interaction
Complex, async, cross-system	Autonomous Agent	A competitive intelligence agent that monitors public news, pulls data from multiple APIs, clusters insights, and generates weekly market reports

(Ref: Agentic AI For Everyone - Aish & Kiriti)

YHK

Keep in Mind ...

- Problem requirements dictate agent design.
- Hybrid systems often work best in practice.
- Single or multi-agent setups depend on task complexity.
- Don't ask "How do I build an agent?" - ask "What problem am I solving?"

YHK

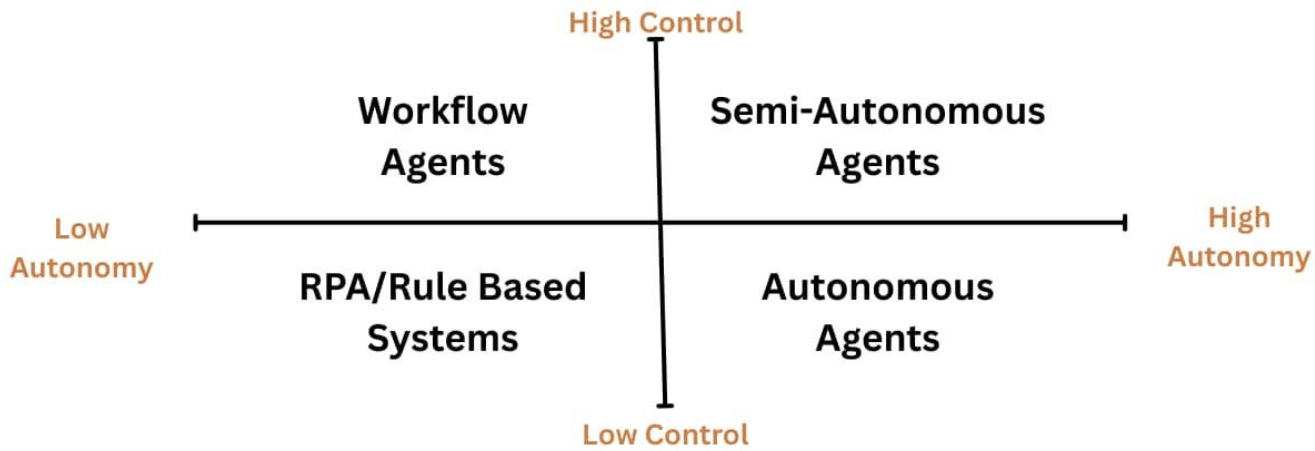
Building Agentic AI: Start with Problems, Not Technology

- **Common mistake:** Starting with "Let's build an agent!"
- **Correct approach:** "What real-world problem are we solving?"
- Identify enterprise pain points first:
  - Support teams drowning in repetitive queries
  - Analysts switching between dashboards for insights
  - Sales teams manually logging customer activity
- Design choices matter for enterprise applications
- Focus on agents that work in the real world, not just demos

YHK

Autonomy vs. Control: A Critical Trade-off

- Key decision: How autonomous should your agent be?
- This is a **contextual trade-off**, not a one-size-fits-all decision
- Spectrum of autonomy:
  - High human control ? Predictable, safe, limited capability
  - High agent autonomy ? Flexible, powerful, potentially risky
- Different problems demand different levels of agent involvement
- Consider: task complexity, risk tolerance, compliance requirements



(Ref: Agentic AI For Everyone - Aish & Kiriti)

When to Use Agents?

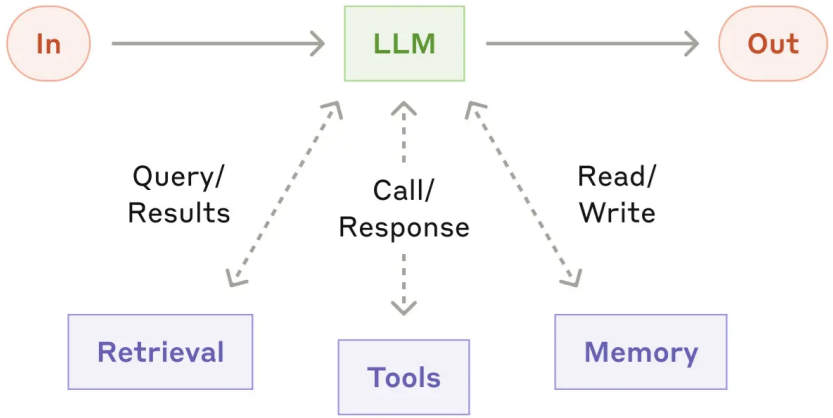
- **Best suited for:** Tasks requiring flexibility and model-driven decision-making
- **Consider trade-offs:** Agents increase latency and cost for better task performance
- **Recommended for:** Open-ended problems with unpredictable steps
- **Simple solutions preferred:** Single LLM calls with retrieval often sufficient
- Agents are systems where LLMs dynamically direct their own processes and tool usage
- Can operate autonomously over extended periods using various tools
- Distinct from workflows: agents have dynamic control vs. predefined code paths

YHK

Agent Architecture Patterns

Building Block: Augmented LLM

- Foundation of agentic systems
- Enhanced with retrieval, tools, and memory capabilities
- Can generate search queries independently
- Actively selects appropriate tools and determines information retention

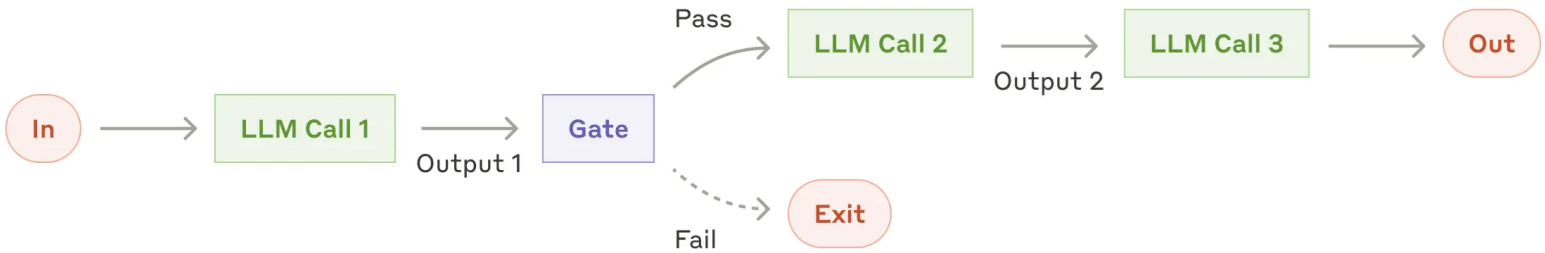


(Ref: Building effective agents - Anthropic)

YHK

Workflow Pattern: Prompt Chaining

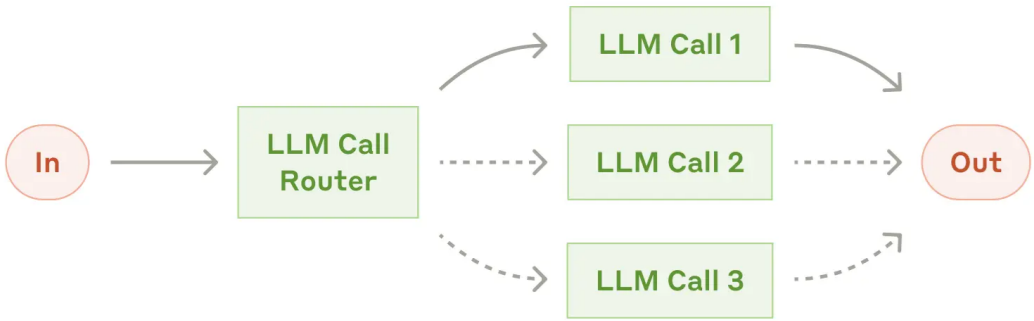
- Decomposes tasks into sequential steps
- Each LLM call processes previous output
- Includes programmatic checks for process verification
- Ideal for tasks with clear, fixed subtasks
- Examples: Marketing copy generation and translation, document outline creation



(Ref: Building effective agents - Anthropic)

Workflow Pattern: Routing

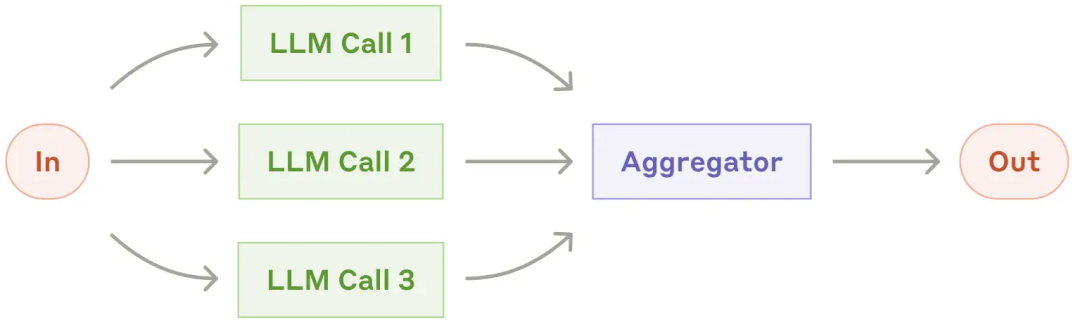
- Classifies input and directs to specialized follow-up tasks
- Enables separation of concerns and specialized prompts
- Suitable for complex tasks with distinct categories
- Applications: Customer service query distribution, model optimization based on query complexity



(Ref: Building effective agents - Anthropic)

Workflow Pattern: Parallelization

- Two key variations: Sectioning and Voting
- **Sectioning:** Breaks tasks into parallel subtasks
- **Voting:** Runs same task multiple times for diverse outputs
- Effective for speed optimization and confidence improvement
- Use cases: Content screening, code vulnerability review



(Ref: Building effective agents - Anthropic)

YHK

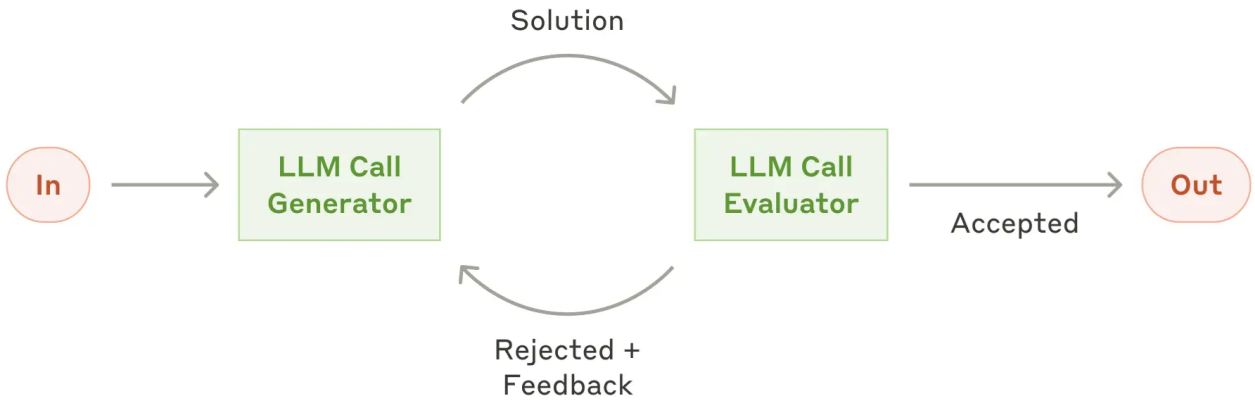
Workflow Pattern: Orchestrator-Workers

- Central LLM coordinates task breakdown and delegation
- Dynamically determines subtasks based on input
- Suitable for complex, unpredictable task structures
- Applications: Multi-file code changes, comprehensive information gathering

(Ref: Building effective agents - Anthropic)

Workflow Pattern: Evaluator-Optimizer

- One LLM generates, another evaluates in a feedback loop
- Effective with clear evaluation criteria
- Ideal for iterative refinement processes
- Examples: Literary translation, complex search tasks



(Ref: Building effective agents - Anthropic)

YHK

Agent Autonomy Levels

AI Agent Levels Overview

- Hugging Face defines 5 levels of AI agents
- Based on autonomy in decision-making and execution
- Control shifts from human to system as levels increase
- From paper: "Fully Autonomous AI Agents Should Not be Developed"
- This framework clarifies much-needed concepts about autonomy in AI agents

YHK

Level 1 - Simple Processor

- No impact on program flow
- Direct input-output processing
- Example: Printing LLM output
- Control: Fully human-driven
- Most basic form of AI integration

YHK

Level 2 - Router

- Determines basic program flow
- Routes execution based on conditions
- Example: If-else decision logic
- Control: Human (how), System (when)
- First level where AI influences execution path

YHK

Level 3 - Tool Calling

- Determines how functions execute
- Chooses tools and parameters dynamically
- Example: Running LLM-chosen tools
- Control: Human (what), System (how)
- This is where most current "agents" operate

YHK

Level 4 - Multi-Step Agent

- Controls iteration and continuation
- Executes multi-step workflows
- Example: Looping task execution
- Control: Human (what), System (which, when, how)
- Significant autonomy with human goal-setting

YHK

Level 5 - Fully Autonomous Agent

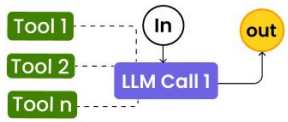
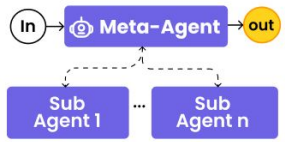
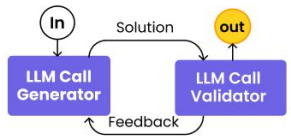
- Creates and executes new code
- Generates, validates, and runs programs
- Example: Code generation from user request
- Control: Fully system-driven
- **Warning:** Hugging Face argues these should not be developed

YHK

Agent Levels Summary

Hugging Face — @rakeshgohel01

5-Different Levels of Agentic AI with code </>

Levels	Workflow	Description	Term	Example Code	Who's in Control?
1		Model has no impact on program flow	Simple Processor	*print_llm_output(llm response)*	Human
2		Model determines basic program flow	Router	*if llm decision():* *path_a()* *else:* *path_b()*	Human: How functions are done; System: When
3		Model determines how functions are executed	Tool Calling	*run function (llm_chosen_tool, llm_chosen_args)*	Human: What functions are done; System: How
4		Model controls iteration and program continuation	Multi-Step Agent	*while should_continue():* *execute_next_step()*	Human: What functions exist; System: Which to do, when, how
5		Model creates & executes new code	Fully autonomous agent	*create_code(user_request);* *execute()*	System

(Ref: LinkedIn post - Rakesh Gohel)

From Monolithic to Compound AI

From Monolithic Models to Compound AI

- **Monolithic models** are limited by training data
- Hard to adapt without significant data and resources
- Example: Vacation planning with no access to personal data
- **Compound AI systems** solve this by integrating multiple components
- External databases and tools enhance model responses
- Shifting from single models to integrated AI systems

YHK

System Design in Compound AI

- Systems are modular: combine models, tools, and databases
- Easier to adapt and quicker to solve problems
- Combining models with external components for enhanced output
- Example: Search query integrated with model for better accuracy
- Programmatic control logic guides the response
- Compound AI systems use system design for better problem-solving

YHK

Role of Agents in Compound AI

- Agents use large language models (LLMs) for reasoning and planning
- LLMs break down problems into manageable tasks
- Slow thinking (planning) leads to better solutions
- Agents provide flexibility in solving complex tasks
- The agent's role is to manage logic and interact with tools
- AI agents improve with reasoning, acting, and memory components

YHK

Components of LLM Agents

- **Reasoning:** Break down complex problems into steps
- **Acting:** Use external tools like databases or calculators
- **Memory:** Store logs and conversation history for personalization
- **Tools:** External programs that help solve problems (search, calculators)
- **Configurations:** Adjust agents using frameworks like ReAct
- Enhanced with planning modules and state management

YHK

Example: ReAct Agent in Action

- ReAct agents think slowly, plan, and act iteratively
- User query feeds into the agent with reasoning instructions
- The agent may call external tools and evaluate the results
- If the result is wrong, the agent revises its plan
- Example: Calculating sunscreen needs for a vacation
- Demonstrates the power of reasoning + acting paradigm

YHK

Agent Autonomy in Problem Solving

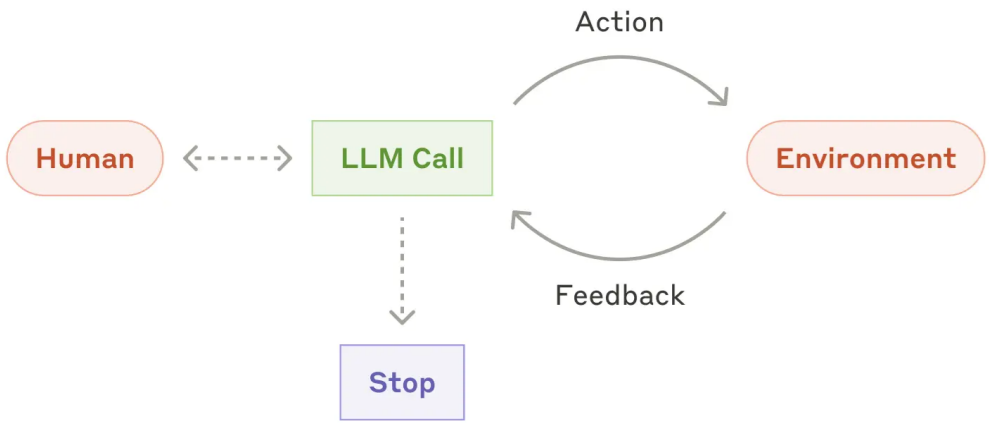
- Narrow problems can be solved with fixed, programmatic systems
- Complex tasks require agent autonomy for better flexibility
- Autonomy level depends on task complexity and need for iteration
- Agents are effective for complex, diverse tasks (e.g., GitHub issues)
- Modular AI systems can solve more complex problems
- Example: Planning vacation with weather, sunscreen, and health data

YHK

Agents in Production

Agents in Production

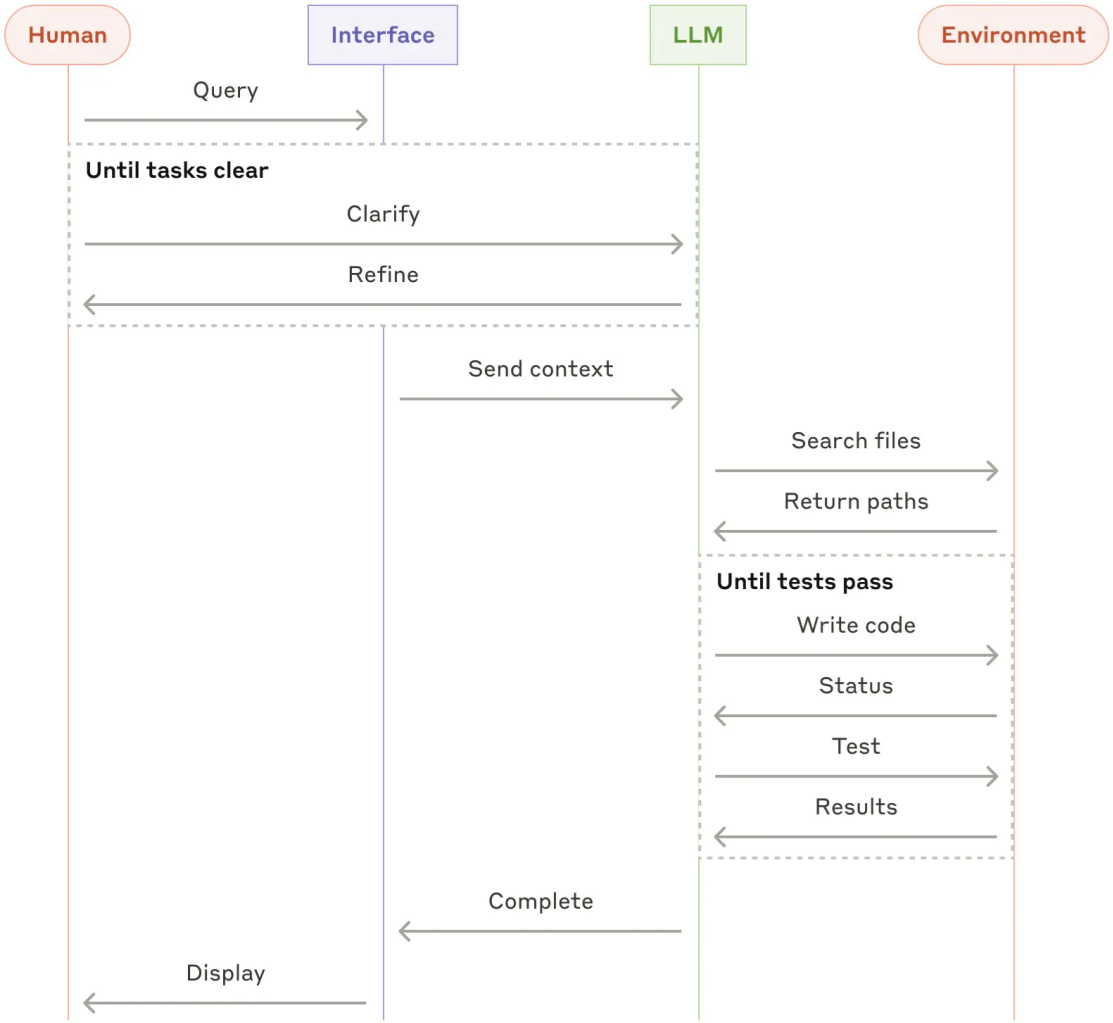
- Operate independently with human interaction when needed
- Require environmental feedback for progress assessment
- Implementation often straightforward despite sophistication
- Key applications: SWE-bench tasks, computer use automation
- Important considerations: Cost management and error prevention



(Ref: Building effective agents - Anthropic)

YHK

High-Level Flow of a Coding Agent



Examples of Agentic AI

- Agents mean **ACTION**
- Agentic AI means **ACTION using AI** (specifically LLMs)
- Examples across domains:
  - **Personal assistants:** Calendar management, email automation
  - **Autonomous robots:** Physical world interaction
  - **Gaming agents:** Strategic decision-making
  - **Science agents:** Research and experimentation
  - **Web agents:** Browser automation, data collection
  - **Software agents:** Code generation, debugging

YHK

Autonomous AI Agents: The Collaborative Advantage

- Collaborative approach yields astonishing enhancements in performance
- Contrasted with using a single AI (like ChatGPT) in isolation
- Ability to assume distinct roles within a team
- Like professionals in various fields
- Each agent contributes specialized expertise to the conversation
- Multi-agent systems outperform single-agent approaches

YHK

The Agent Blueprint

- **Planning:** Reflects on past experiences, offers self-critiques, and breaks down tasks using sub-goal decomposition
- **Memory:** Utilizes sensory, short-term, and long-term memory for:
  - Real-time data processing
  - Task-specific information
  - Retaining knowledge and experiences
- **Tools:** Equipped with a virtual toolbox accessing calendars, calculators, search engines, and other resources for versatile problem-solving

YHK

Agent Flow: The Symphony

- **Task Decomposition:** Breaks down tasks into smaller, manageable components
- **Model (LLM) Selection:** Chooses the most suitable LLM for the task
- **Task Execution:** Leveraging planning, memory, and tools
- **Response Generation:** Creates contextually relevant and accurate responses
  - Drafting reports
  - Answering questions
  - Making decisions

YHK

Agentic Frameworks

Frameworks and Implementation

- Popular frameworks: LangGraph from LangChain, Amazon Bedrock AI Agent
- Start with direct LLM APIs - many patterns need just a few lines of code
- Frameworks can obscure underlying prompts and responses
- Understanding the underlying code is crucial for effective implementation
- Choose framework based on specific needs and complexity requirements

YHK

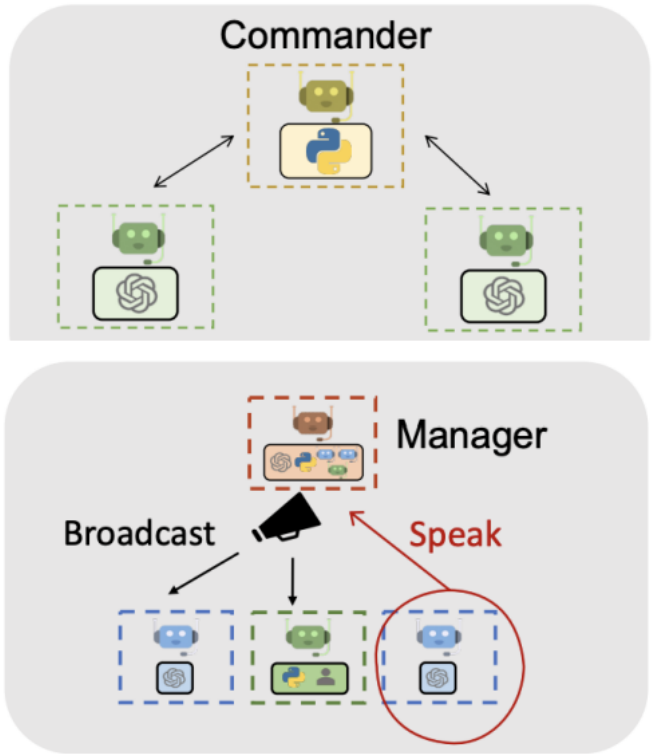
Agentic Framework Requirements

- **Intuitive unified agentic abstraction**
- **Flexible multi-agent orchestration**
- **Effective implementation of agentic design patterns**
- **Support diverse application needs:**
  - Handle more complex tasks and improve response quality
  - Easy to understand, maintain, and extend
  - Modular composition with natural human participation
  - Fast and creative experimentation
- **Agentic Abstraction:** Unify models, tools, humans for compound AI systems

YHK

Multi-Agent Orchestration

- **Static/Dynamic:** Pre-defined vs. adaptive workflows
- **Context sharing/Isolation:** Information flow control
- **Cooperation/Competition:** Collaborative vs. competitive dynamics
- **Centralized/Decentralized:** Control structure design
- **Intervention/Automation:** Human involvement level



YHK

Popular Agentic Frameworks

- **BabyAGI:** Pioneering AI learning system
- **AutoGPT:** Automates content generation
- **GPT Engineer:** Assists in coding and software development
- **LangGraph:** Graph-based control flow
- **CrewAI:** High-level static agent-task workflow
- **AutoGen:** Dialog-based planning and execution

YHK

LangGraph

- Open-source framework by **LangChain** for stateful, multi-actor applications
- Inspired by **directed acyclic graphs (DAGs)** for data processing pipelines
- Treats workflows as graphs with **nodes for specific tasks or functions**
- Enables **fine-grained control** over application flow and state
- Suitable for **complex workflows** with advanced memory and error recovery
- Supports **human-in-the-loop** interactions
- Integrates with **LangChain** for tools and models
- Supports **multi-agent interaction patterns**

YHK

AutoGen

- Framework by **Microsoft** for building conversational agents
- Treats workflows as **conversations between agents**
- Intuitive for users preferring **ChatGPT-like interfaces**
- Supports tools like **code executors** and **function callers**
- Allows agents to perform **complex tasks autonomously**
- Highly **customizable** for additional components and custom workflows
- **Modular design:** Easy to maintain
- Suitable for **simple and complex multi-agent scenarios**

YHK

Crew AI

- Framework for **collaboration of role-based AI agents**
- Agents are assigned **specific roles and goals**
- Ideal for **sophisticated multi-agent systems**
- Examples: **Multi-agent research teams**
- Supports **flexible task management** and delegation
- Enables **autonomous inter-agent collaboration**
- Provides **customizable tools** for diverse applications

YHK

Framework Comparison: Ease of Usage

- **Ease of usage** impacts learning curve and deployment time
- **LangGraph:** Uses graphs (DAGs), ideal for pipeline users but requires graph understanding
- **AutoGen:** Uses conversations, intuitive for chat-based tasks, simplifies interaction management
- **Crew AI:** Role-based design, agents act as cohesive units, easy to start

YHK

Framework Comparison: Tool Coverage

- **Tool coverage** expands agent capabilities through supported functionalities
- **LangGraph:** Integrates with LangChain for tool calling, memory, and human-in-the-loop interactions
- **AutoGen:** Supports tools like code executors and function callers; modular design eases new tool integration
- **Crew AI:** Built over LangChain, inherits their tools; allows defining and integrating custom tools

YHK

Memory Support

- **Memory Support:** Enables agents to maintain context for coherent interactions.
- **Types of Memory:**
  - **Short-Term Memory:** Recalls recent interactions relevant to current tasks.
  - **Long-Term Memory:** Preserves insights from past interactions for knowledge building.
  - **Entity Memory:** Organizes data on specific entities for deeper understanding.
  - **Contextual Memory:** Combines all memory types to sustain interaction context.
- **LangGraph:** Offers advanced memory features, including short-term, long-term, and entity memory. Supports error recovery and time travel.
- **Autogen:** Uses a conversation-driven approach to maintain context across tasks.
- **Crew AI:** Provides comprehensive memory systems, ensuring contextual consistency and knowledge retention.

YHK

Structured Output

- **Structured Output:** Ensures responses are well-organized and interpretable.
- **Benefits:**
  - Facilitates further processing and analysis.
  - Enhances workflow management.
- **LangGraph:** Nodes return structured output for workflow routing and state updates.
- **Autogen:** Supports structured responses via function-calling capabilities.
- **Crew AI:** Allows parsing outputs as Pydantic models or JSON, enabling custom-defined output structures.

YHK

Multi-Agent Support

- **Multi-Agent Support:** Handles diverse interaction patterns between agents.
- **Key Features:**
  - Supports hierarchical, sequential, and dynamic patterns.
  - Enables focused tasks by grouping tools and responsibilities.
  - Allows separate prompts for better task execution.
  - Agents can use distinct fine-tuned LLMs for improved functionality.
- **LangGraph:** Supports hierarchical and dynamic interactions with graph-based visualization.
  - Provides flexibility through explicit agent definitions and transition probabilities.
  - Best for constructing complex workflows.
- **Autogen:** Treats workflows as agent conversations.
  - Enables sequential and nested chats for dynamic interactions.
  - Easy to manage complex collaborations.
- **Crew AI:** Focuses on role-based interactions and autonomous delegation.
  - Implements hierarchical and sequential task execution.
  - Creates cohesive multi-agent teams.

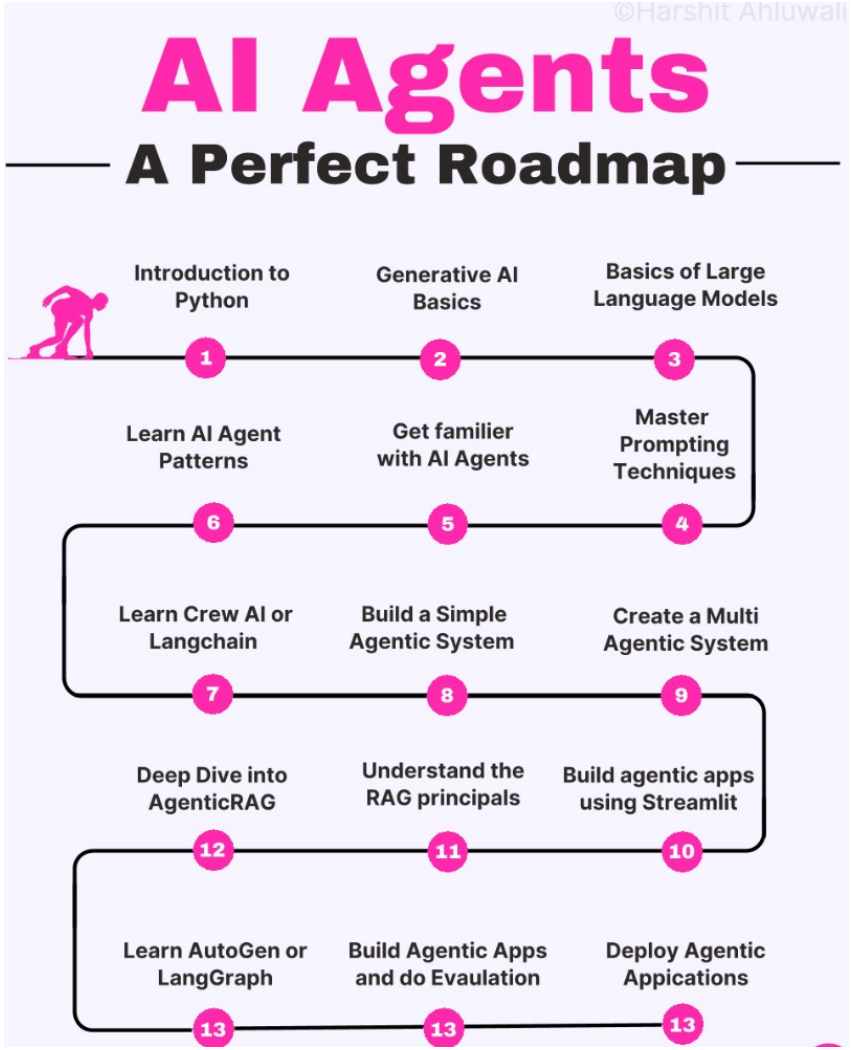
YHK

Summary

Criteria	LangGraph	Autogen	Crew AI	Final Verdict
Ease of Usage	✗	✓	✓	Autogen and Crew AI are more intuitive due to their conversational approach and simplicity.
Multi-Agent Support	✓	✓	✓	Crew AI excels with its structured role-based design and efficient interaction management among multiple agents.
Tool Coverage	✓	✓	✓	LangGraph and Crew AI have a slight edge due to their extensive integration with LangChain.
Memory Support	✓	✓	✓	LangGraph and Crew AI are advanced in memory support features, ensuring contextual awareness and learning over time.
Structured Output	✓	✓	✓	LangGraph and Crew AI have strong support for structured outputs that are versatile and integrable.
Documentation	✓	✓	✓	LangGraph and Crew AI offer extensive and well-structured documentation, making it easier to get started and find examples.
Multi-Agent Pattern Support	✓	✓	✓	LangGraph stands out due to its graph-based approach which makes it easier to visualize and manage complex interactions.
Caching	✓	✓	✓	LangGraph and Crew AI lead with comprehensive caching mechanisms that enhance performance.
Replay	✓	✗	✓	LangGraph and Crew AI have inbuilt replay functionalities, making them suitable for thorough debugging.
Code Execution	✓	✓	✓	Autogen takes the lead slightly with its innate code executors but others are also capable.
Human in the Loop	✓	✓	✓	All frameworks provide effective human interaction support and hence, are equally strong in this criterion.
Customization	✓	✓	✓	All the frameworks offer high levels of customization, serving various requirements effectively.
Scalability	✓	✓	✓	All frameworks are capable of scaling effectively, recommend experimenting with each to understand the best fit.
Open source LLMs	✓	✓	✓	All frameworks support open source LLMs

YHK

AI Agents Learning Road-map



(Ref: LinkedIn post by Harshit Ahluwalia)

Multi Agents

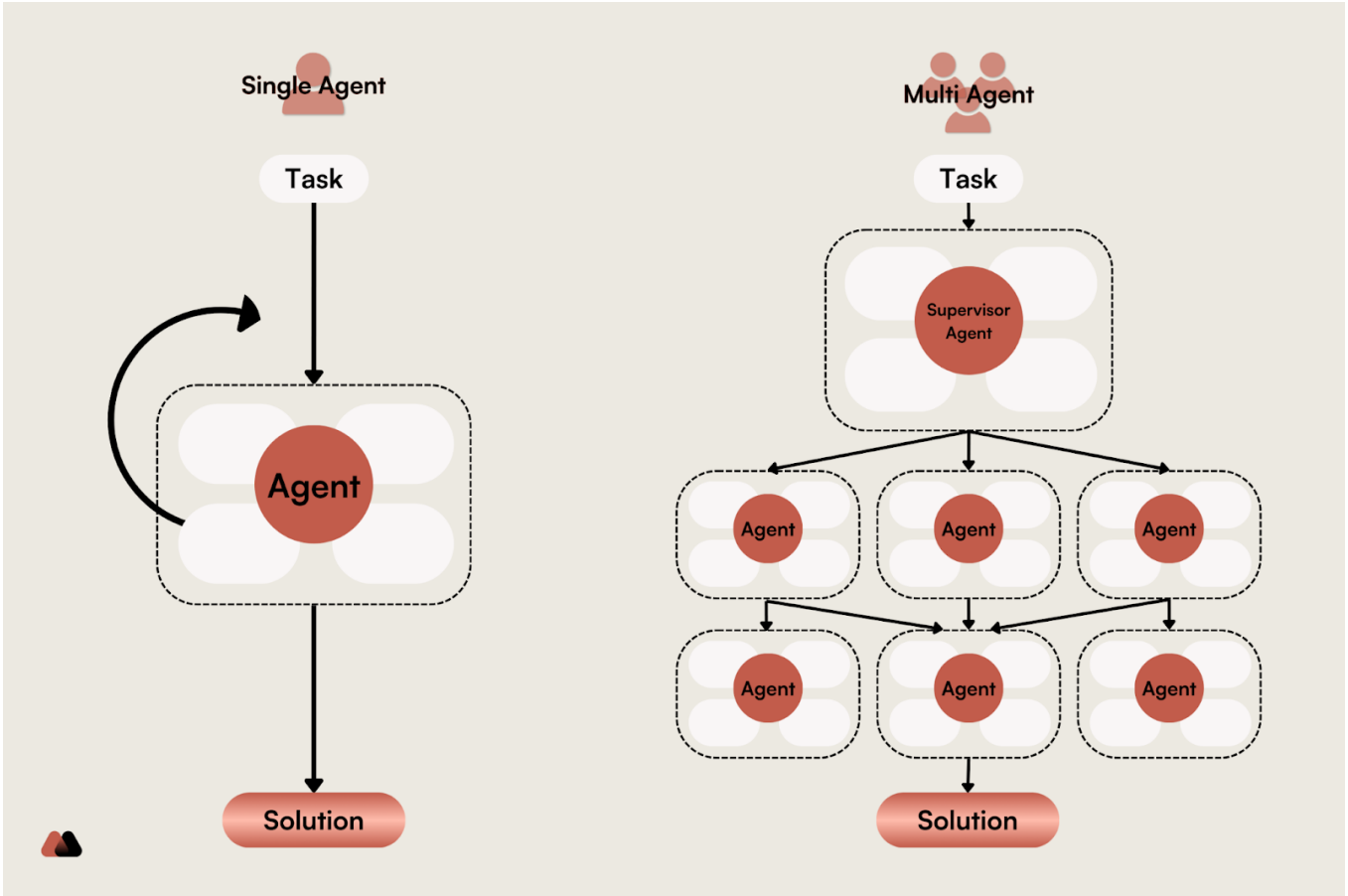
Avoid Overloaded Agents

- Don't overload a single AI agent with many MCP servers
- Leads to performance bottlenecks and poor scalability
- Use multiple agents for effective orchestration

(Ref: LinkedIn post by Rakesh Gohel)

YHK

Agents



(Ref: Meet Agentic AI: The Vanguard of Modern Enterprise - Multimodal)

Why Multi-Agent Systems?

- Specialised agents enable scalable automation
- Collaboration improves decision-making
- Parallel agents deliver faster results
- Real-time adaptation to dynamic inputs

YHK

Benefits of Multi-Agent Workflow

- Single-agent systems are limited in scalability
- Multi-agent systems are modular and efficient
- Better for solving complex, dynamic problems
- Mimics real-world team collaboration

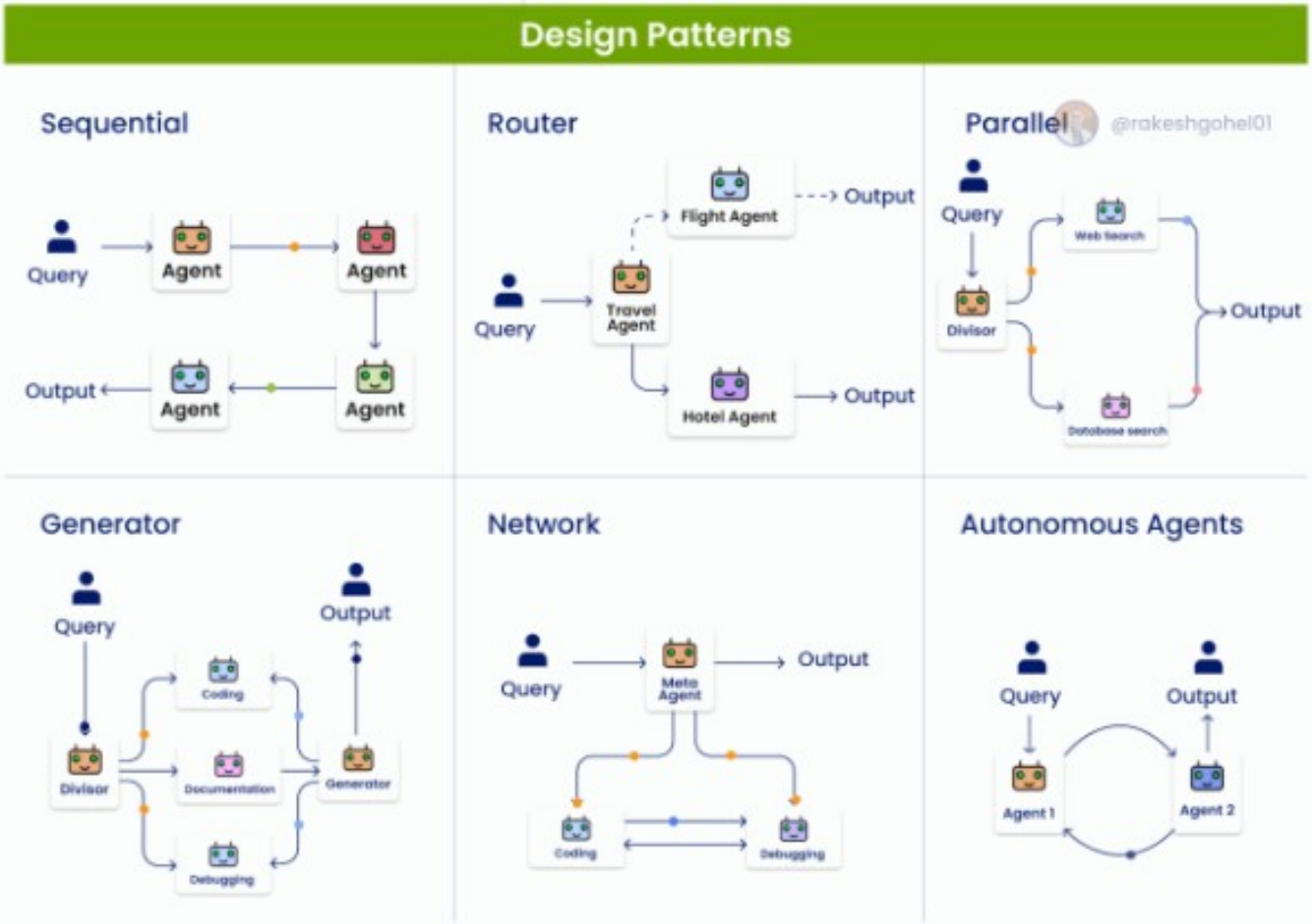
YHK

Popular Multi-Agent Patterns

- Choose design patterns based on task needs
- Six effective patterns streamline development
- Supports better orchestration and coordination

YHK

Multi-Agent Patterns



(Ref: LinkedIn post by Rakesh Gohel)

YHK

1. Sequential Pattern

- Agents execute one after another in a chain
- Each refines or transforms the output
- Use cases: ETL pipelines, Q&A verification

YHK

2. Router Pattern

- Central router delegates tasks to specialists
- Acts like an API gateway
- Use cases: Customer support, service orchestration

YHK

3. Parallel Pattern

- Divides tasks into independent parallel subtasks
- Aggregates results after parallel processing
- Use cases: Info retrieval, financial risk analysis

YHK

4. Generator Pattern

- Iterative loop: divisor → specialists → generator → feedback
- Enables draft-refine workflows
- Use cases: Code generation, design documentation

YHK

5. Network Pattern

- Fully meshed agents with bidirectional links
- Overseen by a central meta-agent
- Use cases: Design, security, compliance reviews

YHK

6. Autonomous Agents Pattern

- Agents operate in decentralised, looped interactions
- No central coordinator needed
- Use cases: Embodied agents, autonomous navigation

YHK

GuardRails

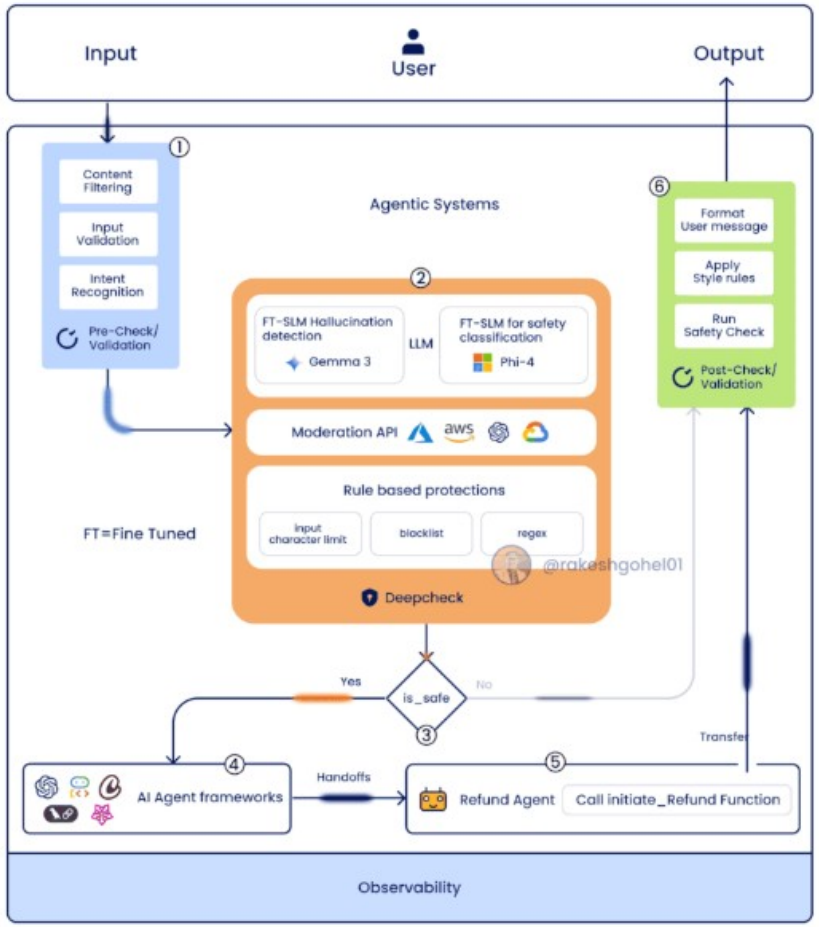
YHK

Guardrails Prevent Liability

- Without guardrails, AI agents can cause serious risks
- A simple malicious prompt can trigger dangerous actions
- Example: “Initiate a refund of $1800” may be executed blindly

YHK

GuardRails



(Ref: LinkedIn post by Rakesh Gohel)

How Guardrails Help

- Guardrails detect, filter, and block unsafe inputs
- Protect agent workflows from abuse and mistakes
- Ensure system behaves safely and predictably

YHK

1. Pre-Check & Validation

- Filters inputs before reaching the AI model
- Includes content filtering and intent detection
- Flags malicious, nonsensical, or off-topic prompts
- First line of defense in any AI pipeline

YHK

2. Agentic Guardrails

- Safety logic embedded inside the agent system
- Uses fine-tuned small LMs and strict rules
- Helps prevent unsafe actions from within

YHK

LLM-Based Safety Checks

- Gemma 3: Detects hallucinations in responses
- Phi-4: Flags unsafe or out-of-scope prompts
- Targets instructions like “Ignore all previous instructions”

YHK

Moderation APIs

- Use APIs from OpenAI, AWS, Azure, etc.
- Catch toxicity, PII, and policy violations
- Adds an additional moderation layer to the pipeline

YHK

Rule-Based Protections

- Blacklists block known prompt injection phrases
- Regex filters catch dangerous patterns
- Input length limits prevent oversized payloads

YHK

3. Deepcheck Safety Validation

- Central logic gate: `is_safe`
- Routes safe prompts to AI agents
- Unsafe prompts are diverted to fallback agents

YHK

4. AI Agent Frameworks & Handoffs

- Once validated, input goes to correct agent
- Example: Refund Agent handles refund logic
- Ensures only safe instructions reach execution layer

YHK

5. Refund Agent Execution

- Final agent in the chain performs the task
- Secure function call handles the refund logic
- Operates only after multilayer validation

YHK

6. Post-Check & Output Validation

- Output reviewed before being sent to user
- Checks formatting, style, and safety again
- Prevents accidental disclosure or unsafe responses

YHK

Observability Layer

- Logs every step: input $\rightarrow$ logic $\rightarrow$ output
- Enables auditing, debugging, and improvement
- Critical for maintaining trust in AI systems

YHK

Key Takeaways

- AI agents need more than good models
- Guardrails ensure safety, traceability, and fallbacks
- Systems thinking is essential for reliable automation

YHK

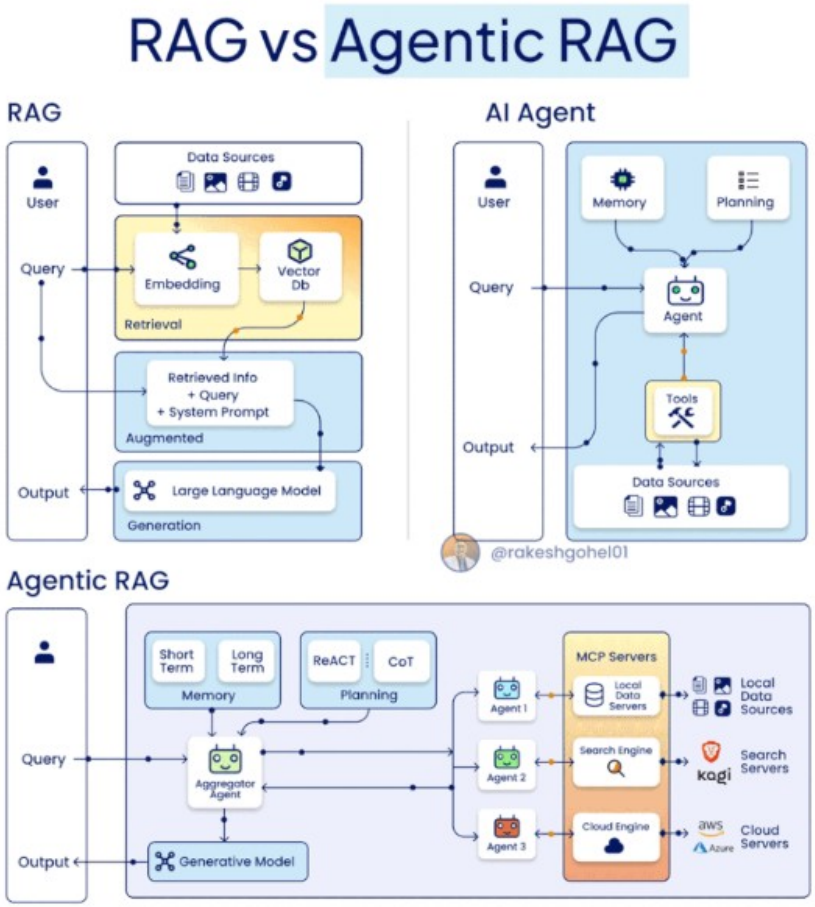
Agentic RAG

Agentic RAG: RAG is Here to Stay

- Agentic RAG proves the lasting value of RAG systems
- Used by Glean AI, Perplexity, Harvey, and others
- Ideal for complex enterprise workflows

YHK

Comparison



(Ref: LinkedIn post by Rakesh Gohel)

What is RAG (Retrieval Augmented Generation)?

- Combines external data retrieval with LLM generation
- Ensures grounded and up-to-date responses
- Enhances reliability and relevance of output

YHK

RAG Workflow Overview

- **Retrieval:** Query is embedded and relevant data is fetched from vector DB
- **Augmentation:** Retrieved data merged with query + system prompt
- **Generation:** LLM generates final response using augmented prompt

YHK

AI Agents in the Loop

- Handle incoming queries and analyze intent
- Use memory and planning (ReACT, Reflexion)
- Fetch real-time data using tools and APIs
- Generate answers using reasoning and context

YHK

How Agentic RAG Combines RAG + Agents

- Agents manage RAG's embedding and retrieval steps
- Dynamically choose data sources based on query
- Augment RAG prompts with planning and external tool data
- Deliver more precise and contextual outputs

YHK

Operational Workflow of Agentic RAG

1. **Query Routing:** Directs query to right agent
2. **Context Retention:** Maintains short and long-term memory
3. **Task Planning:** Chooses tools and retrieval plan
4. **Data Fetching:** Retrieves from KBs using tools (e.g., vector search)
5. **Prompt Optimisation:** Merges retrieved info + prompt + reasoning
6. **Response Generation:** Final LLM output is generated and returned

YHK

Why Agentic RAG Matters

- Enables smarter, more adaptive responses
- Combines memory, planning, retrieval, and reasoning
- Revolutionizing AI in enterprise applications

YHK

Protocols

1. Model Context Protocol (MCP)

- Client-server setup using JSON-RPC
- Enables LLMs to access tools, APIs, datasets
- Supports predefined prompts and tools
- Optionally uses DIDs for secure authentication
- Mostly stateless, with optional persistent context
- **Use-case:** Connect agents to tools with minimal code

YHK

2. Agent-to-Agent Protocol (A2A)

- Enables client-to-agent direct communication
- Uses Agent Cards for capability discovery
- Secure agent auth via DIDs
- Supports real-time updates with SSE & push
- **Use-case:** Delegate tasks across specialized agents

YHK

3. Agent Network Protocol (ANP)

- Decentralized, peer-to-peer communication
- Uses DIDs and JSON-LD for trustless exchange
- Agents discovered via public search engines
- Dynamic protocol negotiation at runtime
- **Use-case:** Build scalable, open agent networks

YHK

4. Agent Communication Protocol (ACP)

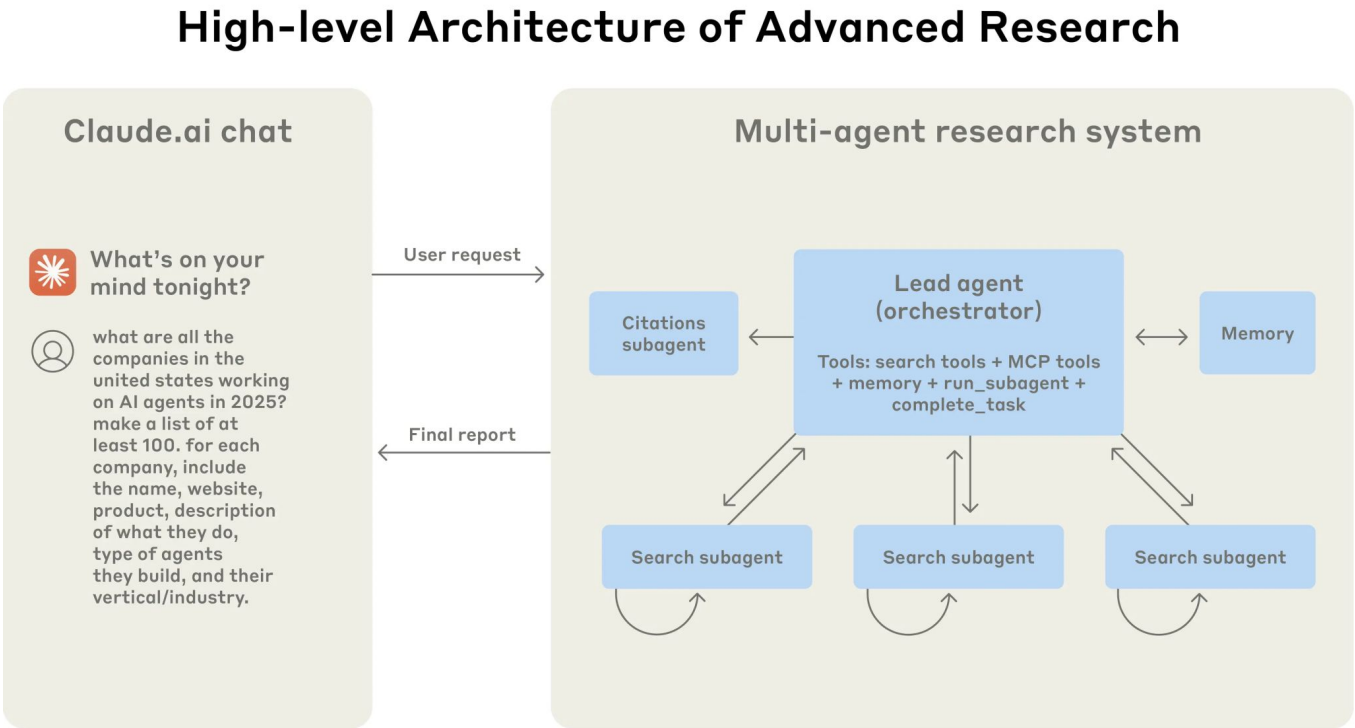
- Centralized registry for agent discovery
- MIME-type multipart messages enable rich data
- Session-aware interactions for multi-turn workflows
- Designed for structured, stateful exchanges
- **Use-case:** Automate cross-department workflows

YHK

Claude Research Multi Agents

Claude Research: Multi-Agent Architecture

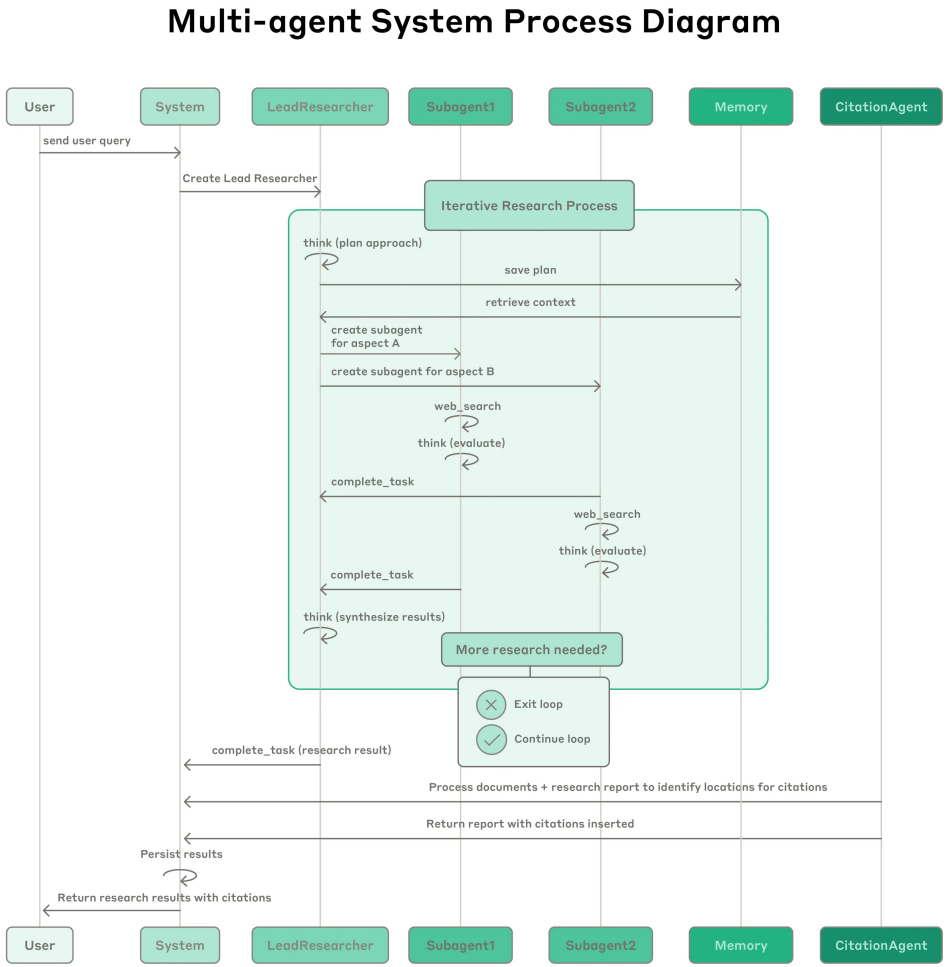
- Anthropic shared insights into Claude's multi-agent architecture
- Real-world example of production-grade agent systems
- Highlights challenges, benefits, and practical design



(Ref: LinkedIn post by Jerry Liu)

YHK

Process diagram



(Ref: How we built our multi-agent research system - Anthropic)

Multi-Agent Research Workflow

- User query spawns a LeadResearcher agent.
- LeadResearcher plans and saves context to Memory.
- Specialized subagents are created for subtasks.
- Subagents perform web search, analyze results, return findings.
- LeadResearcher synthesizes and iterates if needed.
- CitationAgent adds source citations to claims.
- Final report with citations is returned to user.

YHK

Challenges in Multi-Agent Coordination

- Early agents over-delegated and created task redundancy.
- Coordination complexity grows with agent count.
- Prompt engineering was key to guiding agent behavior.
- Simulations helped reveal failure modes.
- Agents often misused tools or duplicated tasks.

YHK

Effective Prompt Engineering

- Build mental models to improve prompt quality.
- Use simulations to observe and refine behavior.
- Prompts guide delegation, scope, and tool use.
- Poor task descriptions cause duplication and gaps.
- Prompts should scale agent effort to task complexity.

YHK

Optimizing Tool Usage

- Tool choice is critical-must match user intent.
- Agents are taught to assess tool relevance first.
- Specialized tools are preferred over generic ones.
- Bad tool descriptions can derail agent behavior.
- Self-improving agents rewrite flawed tool descriptions.

YHK

Search Strategy and Thinking

- Begin with broad queries, then narrow focus.
- Extended "thinking" improves planning and reasoning.
- Subagents evaluate results and refine iteratively.
- Heuristics guide when to explore vs. go deep.
- Avoid verbose or overly specific search queries initially.

YHK

Parallelism and Performance Gains

- Lead agent spawns subagents in parallel.
- Subagents use multiple tools concurrently.
- Parallel execution cuts research time drastically.
- Enables broad exploration within short timeframes.
- Improves system responsiveness for complex queries.

YHK

Reliability in Production Systems

- Agents must persist state across long tasks.
- System supports graceful error recovery and resumption.
- Observability helps trace agent failures without logging content.
- Debugging focuses on behavior patterns, not just outputs.
- Rainbow deployments prevent disruption during updates.

YHK

Sync vs. Async Agent Execution

- Current systems use synchronous agent execution.
- Sync simplifies coordination but slows down progress.
- Async allows subagents to act independently.
- Adds complexity in managing state and errors.
- Anticipated performance gains justify the shift.

YHK

From Prototype to Production

- Minor bugs can cause cascading behavioral failures.
- Agent systems need more engineering than expected.
- Testing, iteration, and collaboration are essential.
- Guardrails prevent runaway agent behavior.
- Final systems must handle edge cases gracefully.

YHK

Impact and User Value

- Users save days by uncovering hidden connections.
- Agents assist in business, healthcare, and debugging.
- Multi-agent systems solve complex research tasks.
- Careful design makes these systems reliable at scale.
- They are transforming how people tackle hard problems.

YHK

Not All Use Cases Need Multi-Agents

- Some domains require shared context among agents
- High interdependencies reduce multi-agent effectiveness
- Not every task benefits from parallel agent workflows

YHK

Single vs Multi-Agent Debate

- Similar point made by Cognition's "Don't Build Multi-Agents"
- Both agree: multi-agents fit a specific class of problems
- Focus should be on identifying those right-fit use cases

YHK

Sub-Agents as Tools, Not Peers

- Claude's system treats sub-agents like tools
- No explicit agent-to-agent handoffs
- Simplifies control and orchestration

YHK

Agents Improve Tool Interfaces

- Claude uses a tool testing agent to refine tool descriptions
- Agent rewrites unclear interfaces after testing failures
- Result: 40% reduction in task time for future agents

YHK

Self Improving Agents in Practice

- Tool ergonomics improved through feedback loops
- Agents help reduce integration complexity
- Smarter interface fewer downstream errors

YHK

Synchronous Execution Bottlenecks

- Claude's agents wait synchronously for sub agent results
- Simplifies coordination, but delays execution
- Creates sequential bottlenecks in agent chains

YHK

The Case for Async Architectures

- Event driven models allow async agent execution
- Each agent acts as events arrive-faster coordination
- Matches design in frameworks like LlamaIndex workflows

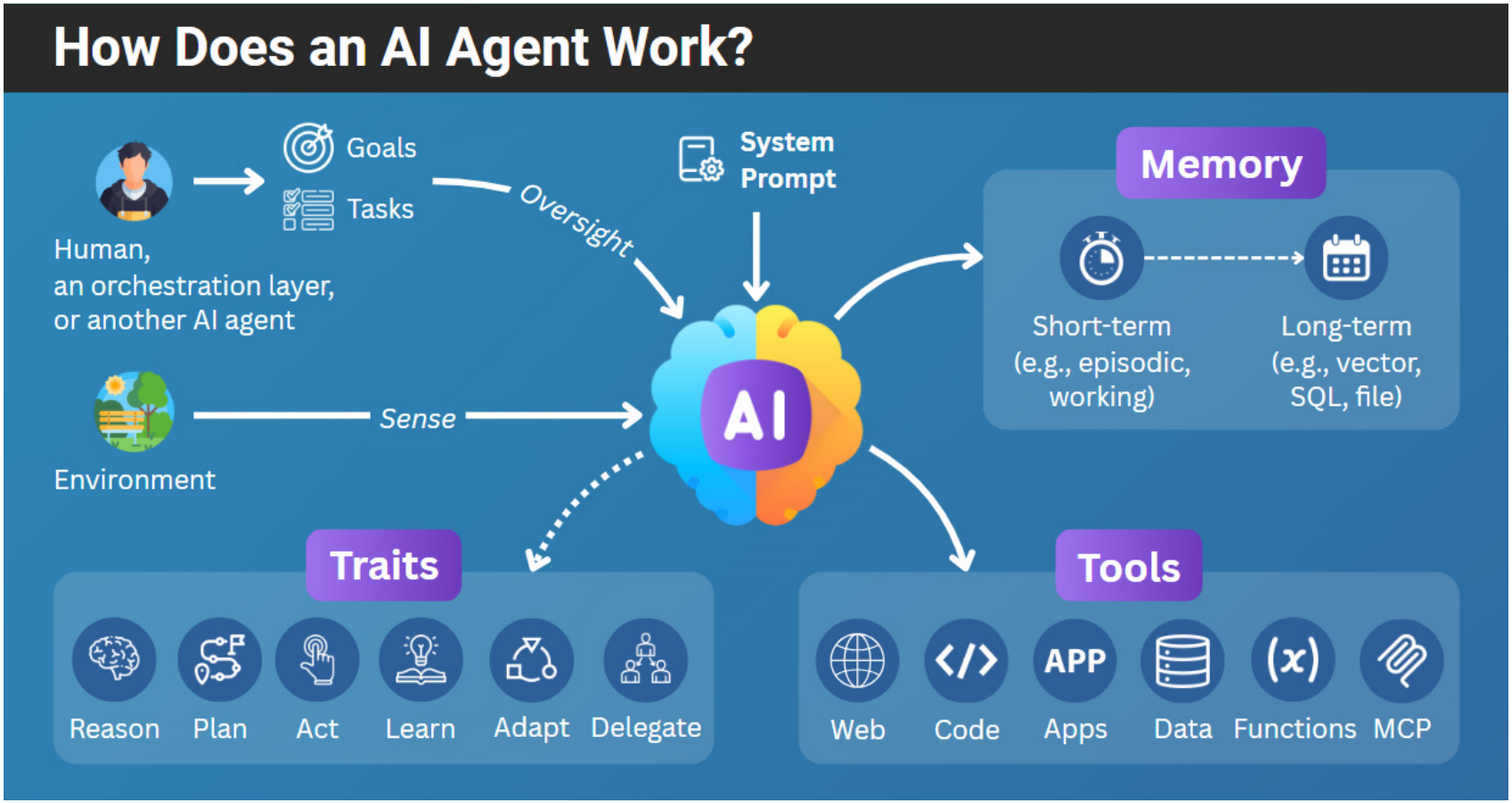
YHK

Key Lessons from Claude Research

- Use multi agent design selectively and purposefully
- Let agents optimize tools and interfaces over time
- Consider async architectures to eliminate bottlenecks

YHK

Implementation



(Ref: The Ultimate Guide to AI Agents for PMs - Pawl Huryn)

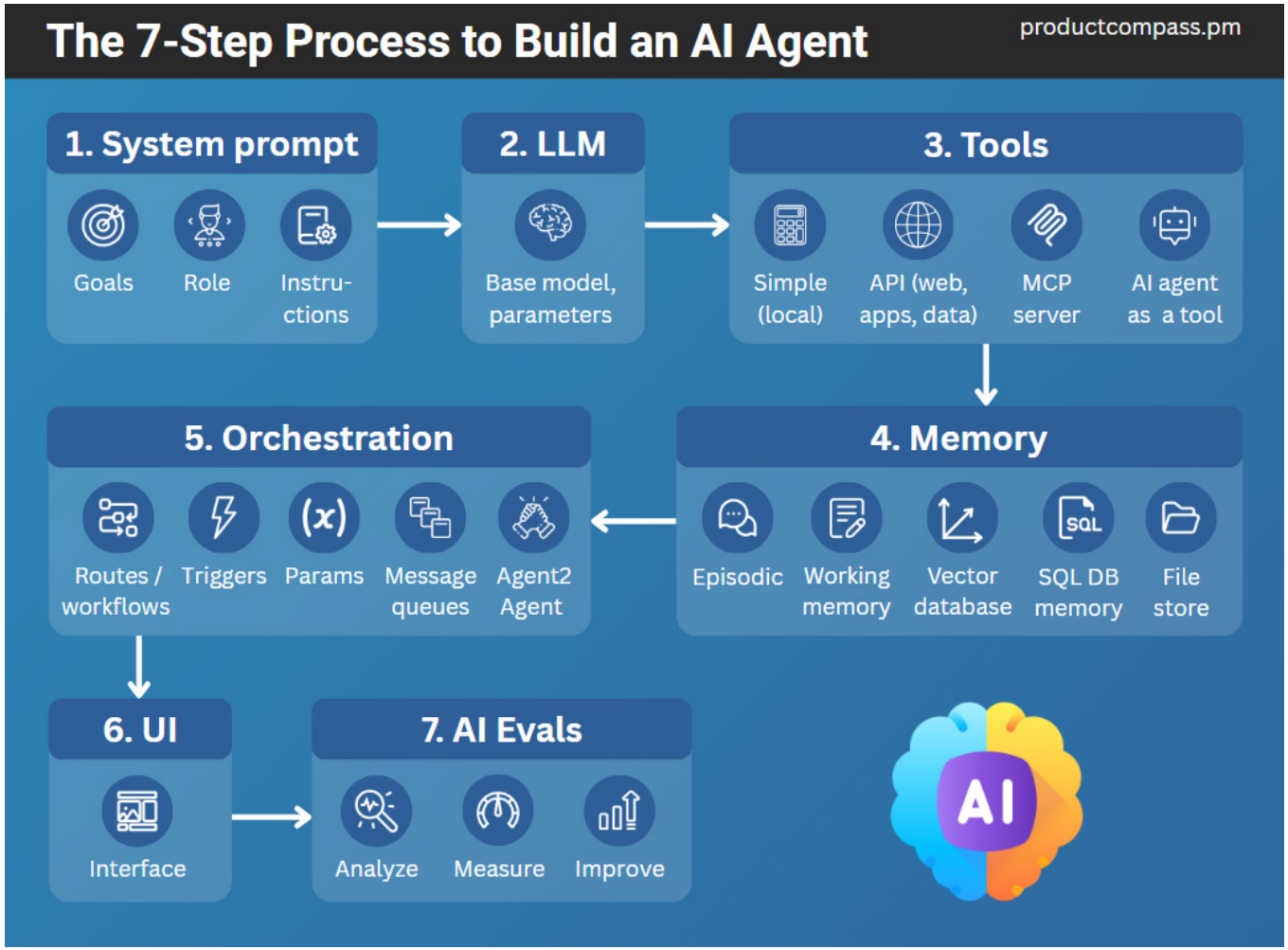
YHK

Build Your First AI Agent

- Takes just 30-60 minutes to get started
- Follow a clear step-by-step process
- Focus on functionality, not theory

YHK

How to Build an AI Agent?



(Ref: The Ultimate Guide to AI Agents for PMs - Pawl Huryn)

Step 1: Define a System Prompt

- Set goals, logic, and expectations
- Use structured prompting principles
- Refer to expert guides for inspiration

YHK

Step 2: Select an LLM

- Choose a reasoning-capable model (e.g., o1-mini)
- Frameworks like n8n may handle iterations
- Pick based on your use case complexity

YHK

Step 3: Connect Tools

- Add tools based on agent goals
- Use calculators, functions, data sources
- Optional: MCP servers for integration

YHK

Step 4: Connect Memory

- Enable short-term memory for local state
- Use long-term memory: vector, SQL, graph
- Essential for tracking progress and context

YHK

Step 5: Orchestrate the Logic

- Map core logic not tied to a single agent
- Enable agent-to-agent communication
- Static or dynamic flows via orchestration

YHK

Step 6: Add User Interface

- Use no-code tools like Lovable or Bolt
- Easily create interfaces without coding
- Great for user-facing agents and SaaS apps

YHK

Step 7: Evaluate the Agent

- Skip fixed metrics-do error analysis
- Let performance metrics emerge naturally
- For RAG: evaluate retrieval and generation separately

YHK

Bottom Line: Just Start

- Follow practical guides-many require no coding
- Use frameworks like n8n and tools like MCP
- Stop theorizing-start shipping real systems

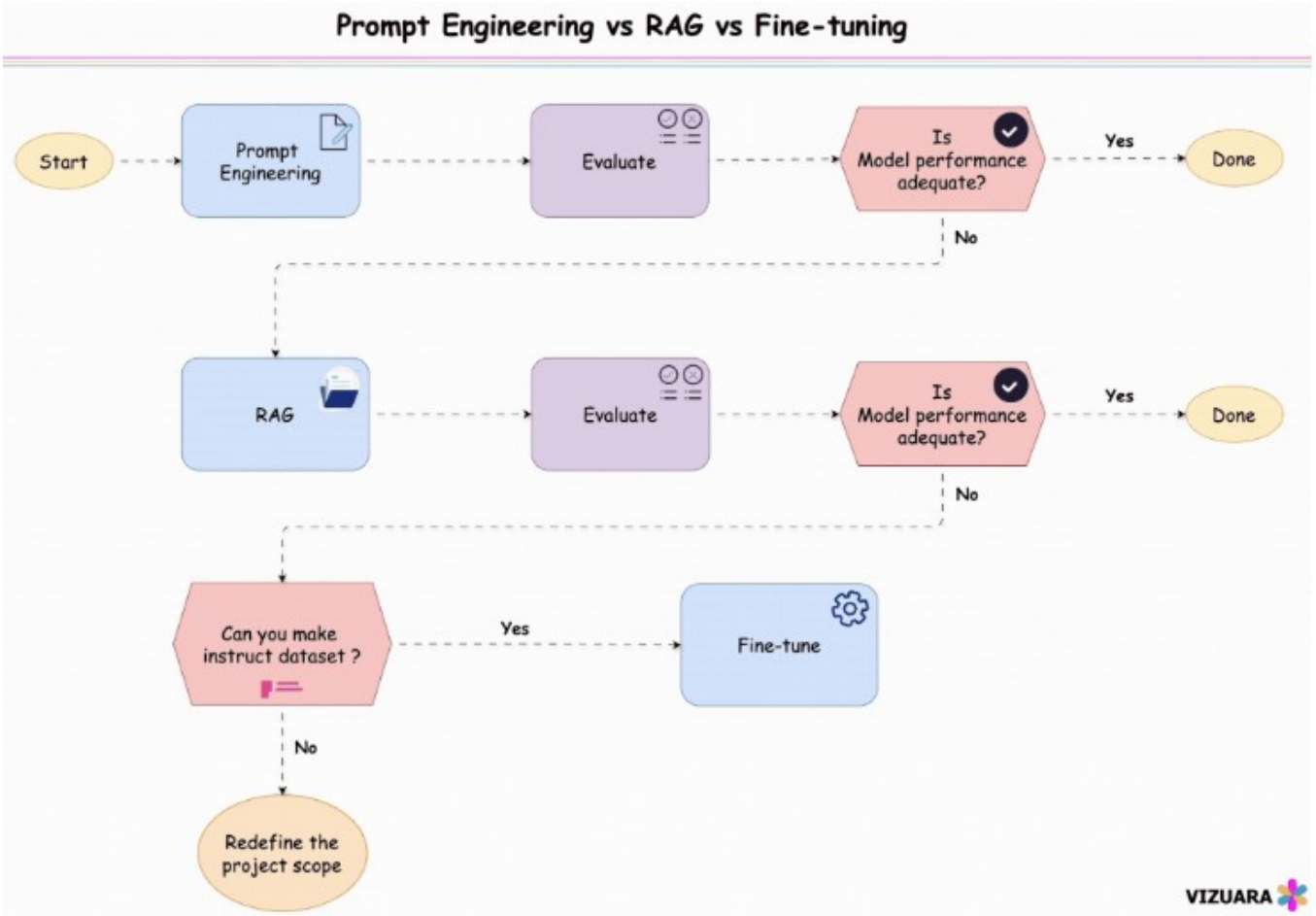
YHK

Practical Tips

RAG vs. Fine-Tuning: Business Impact

- Business-critical systems require the right LLM strategy.
- Fine-tuning feels powerful, but often adds avoidable complexity.
- Start by exploring simpler and more flexible approaches first.

RAG vs SFT



(Ref: LinkedIn post by Raj Dandekar)

Pre-Fine-Tuning Checklist

- Can prompt engineering alone solve the task?
- Could Retrieval-Augmented Generation (RAG) help more?
- Is there a clear and testable system already in place?

YHK

Why RAG Outperformed Fine-Tuning

- **Higher Accuracy**: Grounded answers from relevant context.
- **Fewer Hallucinations**: More reliable than fine-tuned outputs.
- **Dynamic Updates**: Easily update data without retraining models.

YHK

When to Use Fine-Tuning

- RAG can't solve context window limitations.
- Domain-specific tone or behavior is required.
- The system is mature enough to absorb added complexity.

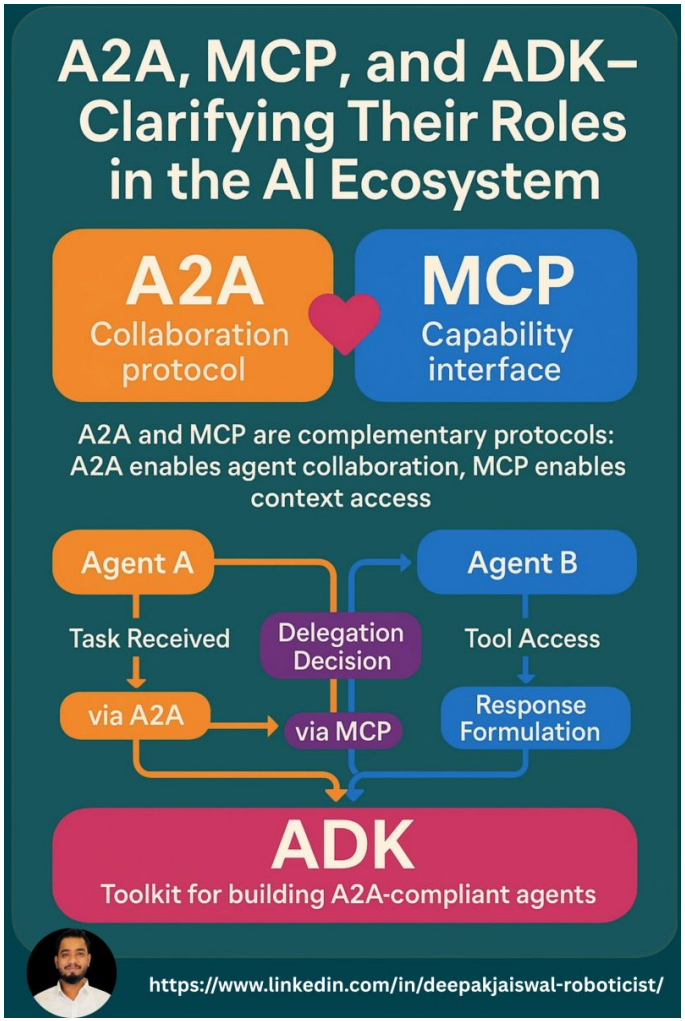
YHK

Strategy Summary

- Prompt engineering solves 30–50% of tasks.
- RAG adds power for another 30–40%.
- Fine-tuning is best for the final 10% of hard problems.
- Always choose the simplest effective approach first.

YHK

A2A MCP ADK



(Ref: LinkedIn post by Deepak Jaiswal)

YHK

Agentic AI Protocols Overview

- A2A, MCP, and ADK are foundational for agentic systems.
- Each solves a unique challenge in building autonomous agents.
- They work together to enable scalable multi-agent architectures.

YHK

A2A: Agent-to-Agent Protocol

- Enables agent discovery, delegation, and communication.
- Facilitates coordination in distributed multi-agent systems.
- Example: Agent A delegates a task to Agent B and receives results.

YHK

MCP: Model Context Protocol

- Standardizes agent access to tools, APIs, and data.
- Ensures consistent, secure interaction with external systems.
- Example: Agent queries a database or triggers a payment via MCP.

YHK

ADK: Agent Development Kit

- Toolkit for building A2A-compliant agents quickly.
- Includes libraries and scaffolds for rapid development.
- Compatible with frameworks like CrewAI, LangGraph, Semantic Kernel.

YHK

How They Work Together

- **MCP**: Makes agents powerful with tool access.
- **A2A**: Enables inter-agent collaboration.
- **ADK**: Simplifies and accelerates agent development.

YHK

Real-World Analogy

- **MCP**: Tools in a mechanic's toolbox.
- **A2A**: Team communication and task-sharing.
- **ADK**: Blueprint for building each mechanic fast.

YHK

Why This Stack Matters

- Forms the backbone of enterprise AI, robotics, and automation.
- Enables modular, scalable, and intelligent agentic systems.
- Quickly becoming a standard for future AI workflows.

YHK

Implementations

AutoGen

What is AutoGen?

- Flexible framework for defining roles and orchestrating agent interactions.
- Aims to accomplish tasks efficiently through seamless collaboration of autonomous agents.
- Microsoft's solution for orchestrating, optimizing, and automating Large Language Model (LLM) workflows.

YHK

It all started with ...

AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation

Qingyun Wu[†], Gagan Bansal[*], Jieyu Zhang[±], Yiran Wu[†], Beibin Li[*]

Erkang Zhu[*], Li Jiang[*], Xiaoyun Zhang[*], Shaokun Zhang[†], Jiale Liu[∓]

Ahmed Awadallah[*], Ryen W. White[*], Doug Burger[*], Chi Wang[*1]

[*]Microsoft Research, [†]Pennsylvania State University

[±]University of Washington, [∓]Xidian University

Framework

- Agents may handle code generation, execution, and human supervision.
- Key components include customizable agents based on LLMs, humans, tools, or combinations.
- Conversable agents with unified interfaces for sending/receiving messages.
- Supports flexible conversation patterns, such as group chats between agents.

YHK

Unified Interface

- Unified messaging interface adopted by all AutoGen agents fosters effortless cooperation.
- Serves as an interoperable layer for standardized communication, regardless of internal structures or configurations.
- Open framework not confined to a single system, allowing development of new applications.

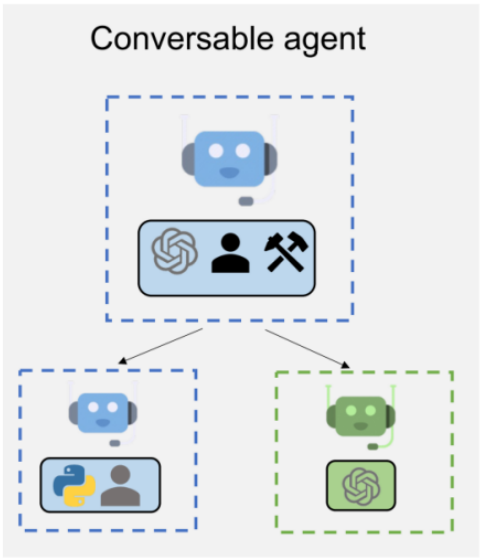
YHK

Unique Features

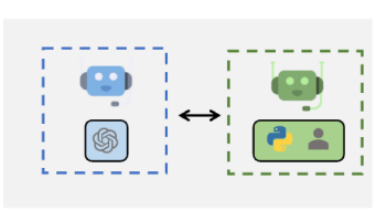
- Some pre-cooked agent types are provided viz Assistant, User Proxy Agent, etc.
- Assistant is like a standard chatbot, given a query it will answer.
- User Proxy Agent is your ie user's Proxy. So it has the task to get done. It initiates the chat.
- There are properties within it to have Human In Loop ie interactive, ALWAYs, NEVER, TERMINATE. If you want fully autonomous working, then set it to NEVER.
- Group Chat Manager offers creating chat rooms of AI agents.

YHK

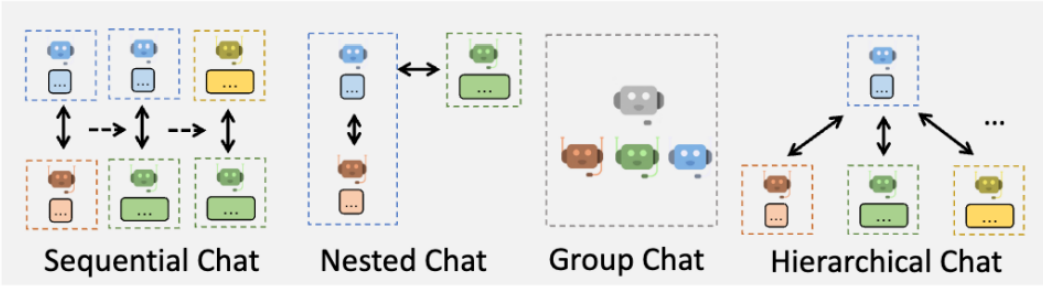
Define agents and Get them to talk



Agent Customization



Multi-Agent Conversations



Flexible Conversation Patterns

(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

YHK

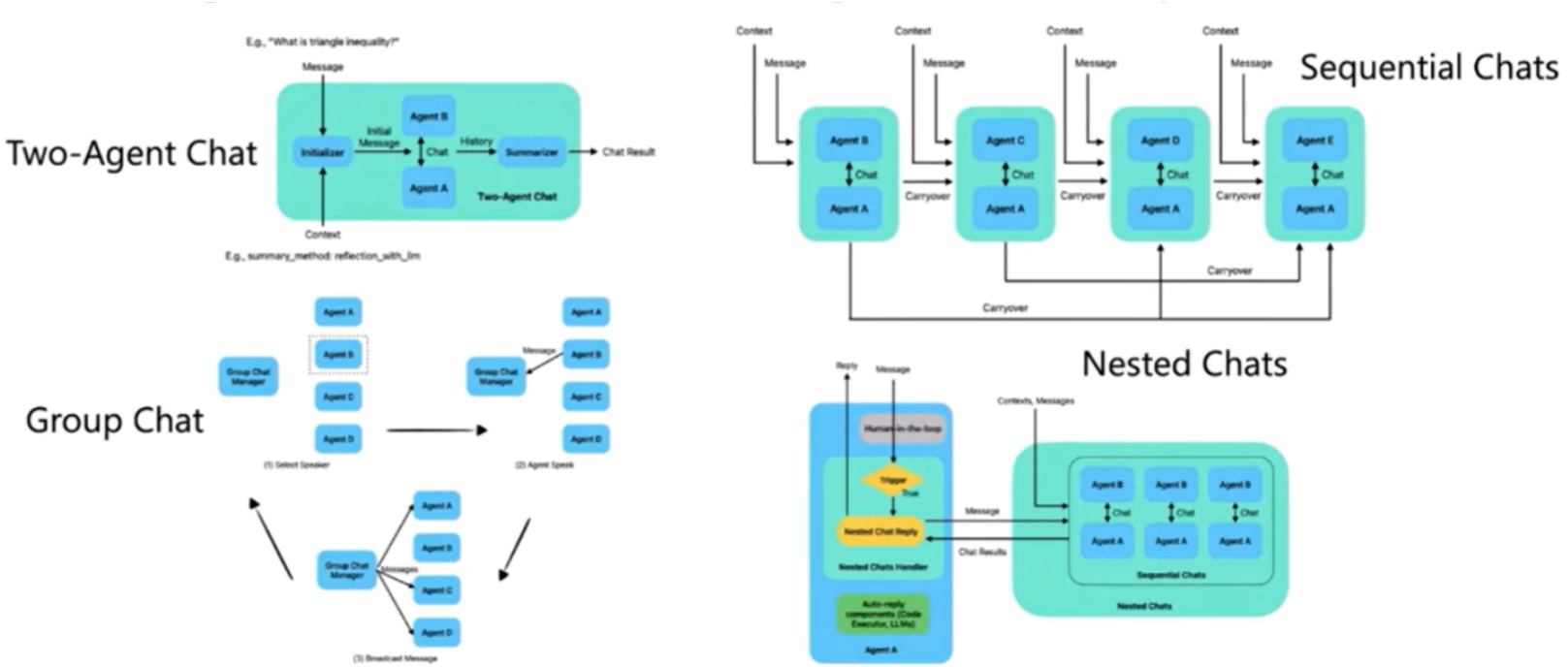
System Message

```
You are a helpful AI assistant. Solve tasks using your coding and language skills.

In the following cases, suggest python code (in a python coding block) or shell script (in a sh
coding block) for the user to execute.

1. When you need to collect info, use the code to output the info you need, for example, browse or
search the web, download/read a file, print the content of a webpage or a file, get the current
date/time. After sufficient info is printed and the task is ready to be solved based on your
language skill, you can solve the task by yourself.

2. When you need to perform some task with code, use the code to perform the task and output the
result. Finish the task smartly.

Solve the task step by step if you need to. If a plan is not provided, explain your plan first. Be
clear which step uses code, and which step uses your language skill.

When using code, you must indicate the script type in the code block. The user cannot provide any
other feedback or perform any other action beyond executing the code you suggest. The user can't
modify your code. So do not suggest incomplete code which requires users to modify. Don't use a
code block if it's not intended to be executed by the user.

If you want the user to save the code in a file before executing it, put # filename: <filename>
inside the code block as the first line. Don't include multiple code blocks in one response. Do not
ask users to copy and paste the result. Instead, use 'print' function for the output when relevant.
Check the execution result returned by the user.

If the result indicates there is an error, fix the error and output the code again. Suggest the
full code instead of partial code or code changes. If the error can't be fixed or if the task is
not solved even after the code is executed successfully, analyze the problem, revisit your
assumption, collect additional info you need, and think of a different approach to try.

When you find an answer, verify the answer carefully. Include verifiable evidence in your response
if possible.

Reply "TERMINATE" in the end when everything is done.
```

(Ref: AutoGen - John Tan Chong Min)

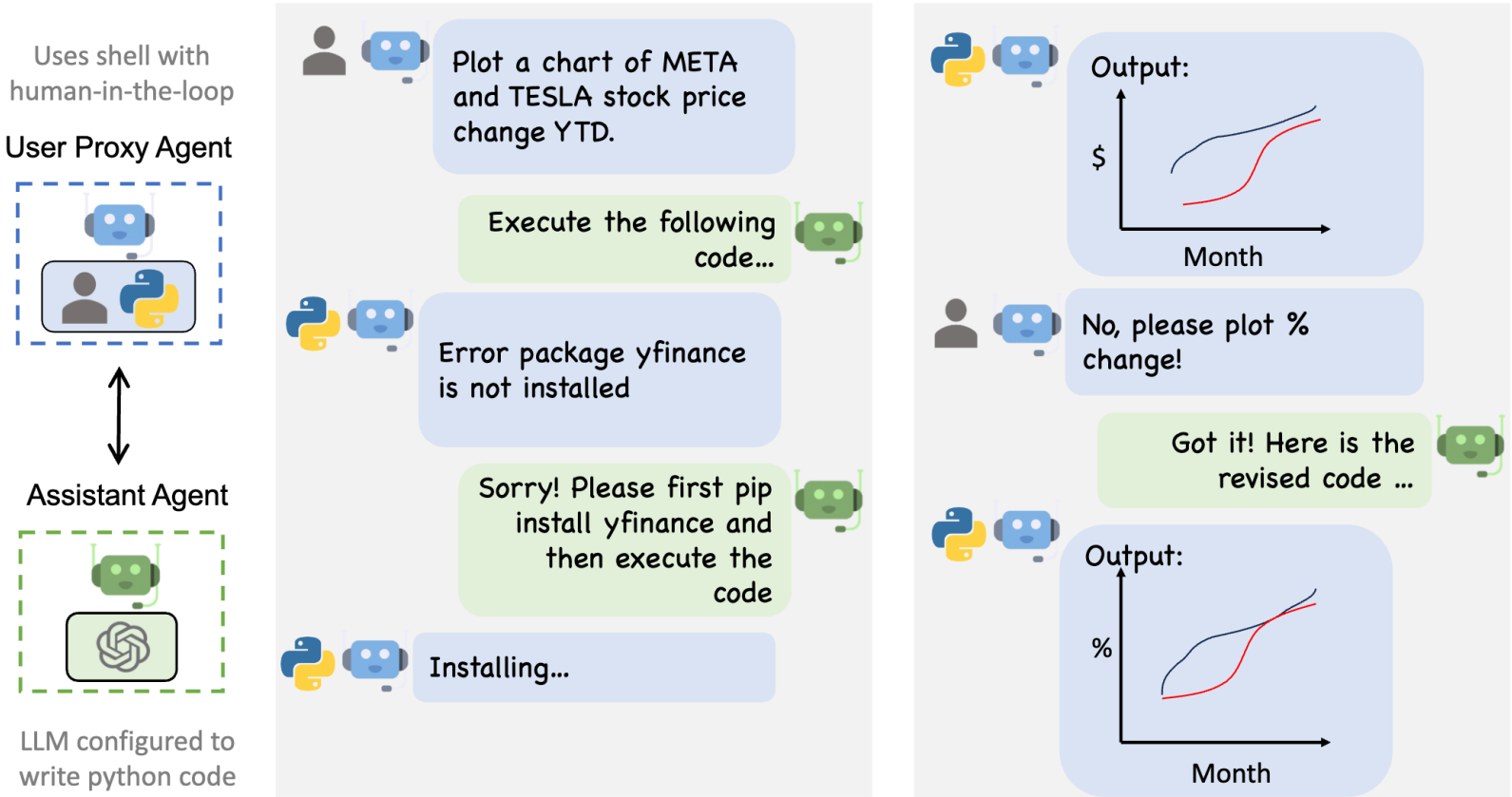
YHK

Conversation Patterns



(Ref: Build Agentic AI Apps with the Autogen framework — OD539 - Microsoft Developer)

Example: Plot Stocks



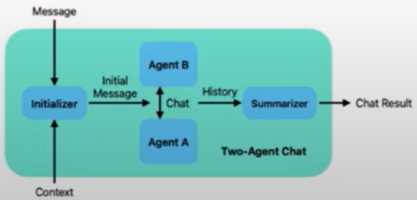
(Ref:"AutoGen: Enabling next-generation large language model applications" — Microsoft)

Two Agents Chat

- Two agents share same thread
- Assistant suggestions code and Executor runs the code (code execution is typically done the docker sandbox for safety) or Human-in-the-loop for executor)

two-agent-chat.py

```
programming_assistant = ConversableAgent(
    system_message="You are a programming assistant, and you solve tasks
using code...",
    llm_config={"model": "gpt-4-turbo", "api_key": ...},
    ...
)
code_executor = DockerCommandLineExecutor(
    image="python:3.11",
    ...
)
programming_executor = ConversableAgent(
    code_execution_config = {"executor": code_executor},
    human_input_mode = "ALWAYS",
    ...
)
result = programming_executor.initiate_chat(
    programming_assistant,
    message="What is the sum of 74th and 90th Fibonacci number?"
)
```

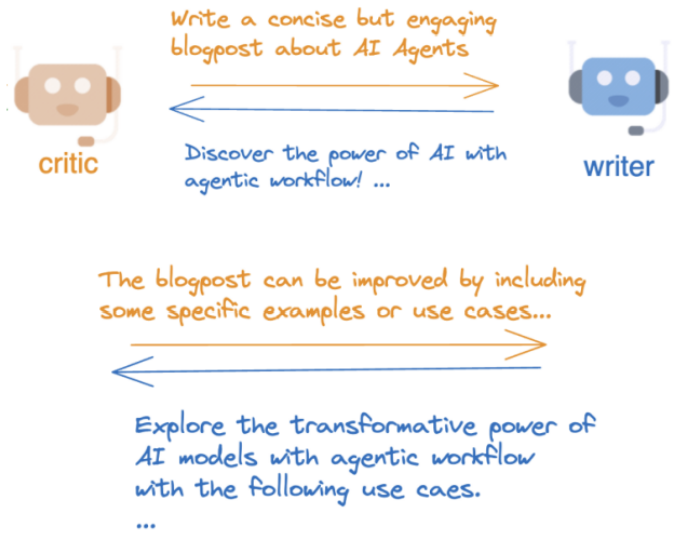


(Ref: Build Agentic AI Apps with the Autogen framework — OD539 - Microsoft Developer)

YHK

Example: Blogpost Writing with Reflection

```python
writer = autogen.AssistantAgent(
    name="Writer",
    system_message="You are a writer...",
    llm_config=llm_config,
)

critic = autogen.AssistantAgent(
    name="Critic",
    is_termination_msg=lambda x: x.get("content", "").find("TERMINATE") >= 0,
    llm_config=llm_config,
    system_message="You are a critic...",
)

critic.initiate_chat(
    recipient=writer,
    message=task,
    max_turns=2,
    summary_method="last_msg"
)
```



(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

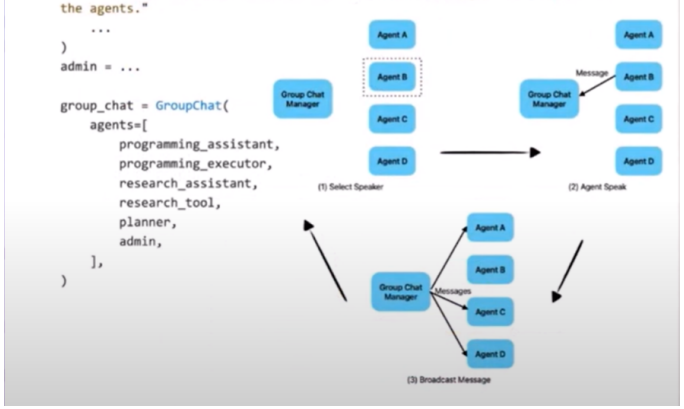
YHK

Group Chat

- Agents participate in a single thread
- Speaker is selected by a group chat manager.
- Planner agent to plan and guide other agents
- Admin agent for collecting human feedback
- Each agent could be simple or group or sequential nested agent, making this composable and more complex if needed.

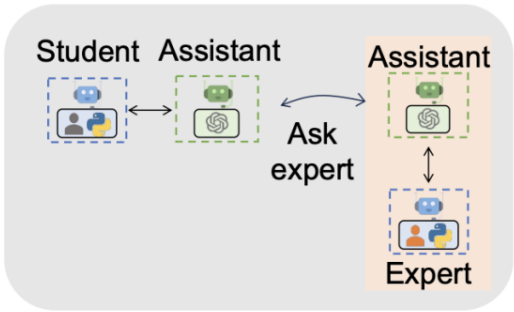
group-chat.py

```
programming_assistant = ...
programming_executor = ...
research_assistant = ...
research_tool = ...
planner = ConversableAgent(
    system_message="You are a planner for complex tasks. "
    "You come up with a plan and assign subtasks to different agents. "
    "You also check for the completion of the tasks and provide feedback to
the agents."
    ...
)
admin = ...

group_chat = GroupChat(
    agents=[
        programming_assistant,
        programming_executor,
        research_assistant,
        research_tool,
        planner,
        admin,
    ],
)
```

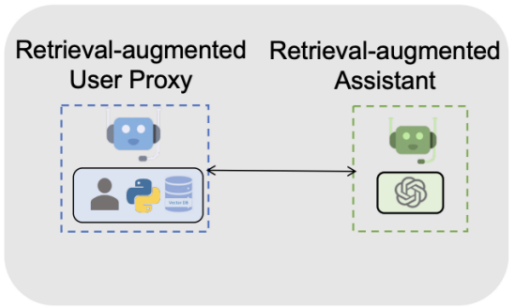


(Ref: Build Agentic AI Apps with the Autogen framework — OD539 - Microsoft Developer)

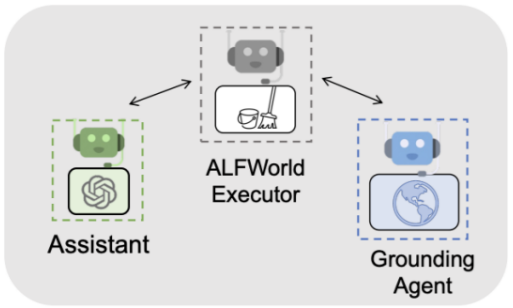
More examples



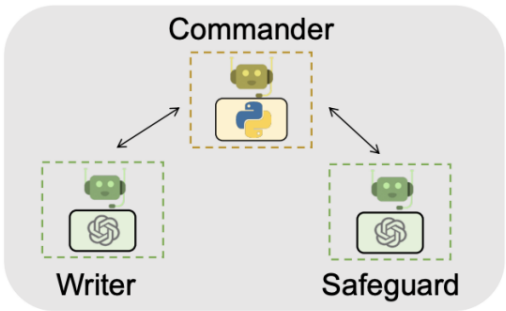
A1. Math Problem Solving



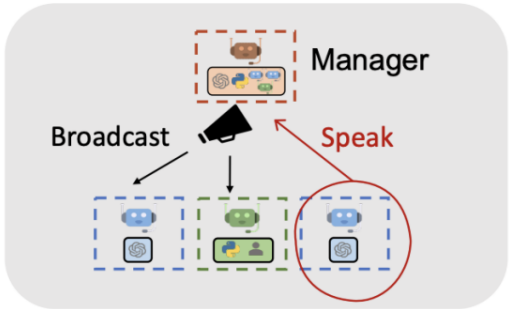
A2. Retrieval-augmented Q&A



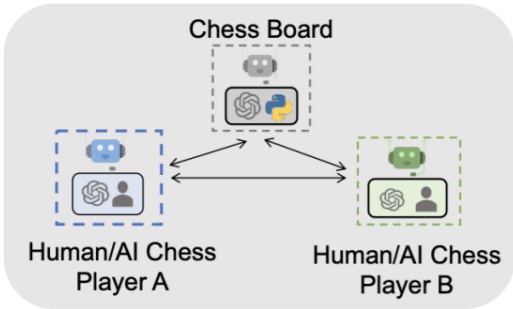
A3. Decision Making in Embodied Agents



A4. Supply-Chain Optimization



A5. Dynamic Task Solving with Group Chat



A6. Conversational Chess

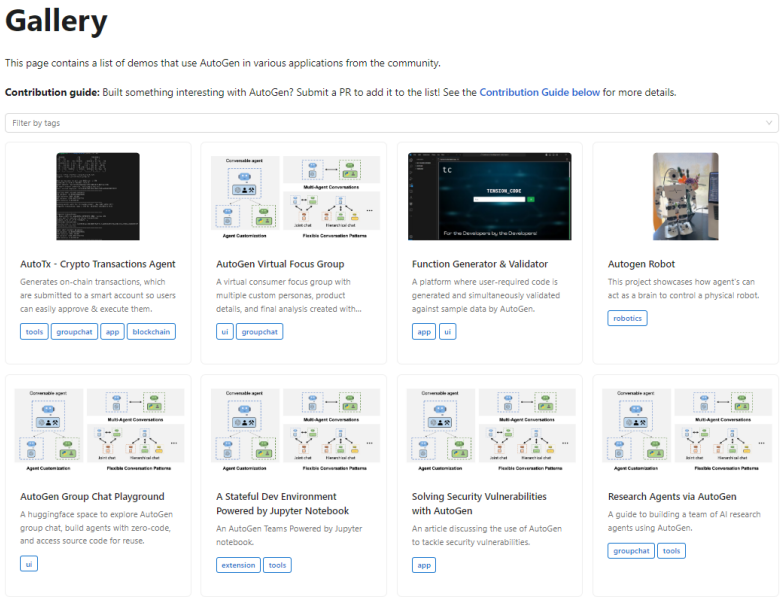
For more examples: https://autogen-ai.github.io/autogen/docs/notebooks

(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

YHK

Applications

- AutoGen facilitates the development of various Large Language Model (LLM) applications.
- Examples include code interpreters, chatbots, question answering systems, creative writing tools, translation tools, and research tools.
- Many application examples can be seen in https://microsoft.github.io/autogen/docs/Gallery



(Ref:https://microsoft.github.io/autogen/docs/notebooks)

Applications

- Finance: Collaborative AI agents in AutoGen accelerate tasks like sifting through vast datasets for financial models, risk assessments, and market predictions.
- Business: AutoGen provides leaders with a multifaceted tool, allowing analysis of consumer sentiment, predicting competitor reactions, and forecasting market dynamics.
- Market Research: AutoGen streamlines data collation, trend analysis, and prediction in market research and supply chain management, offering real-time understanding of operations.
- Democratizing AI: AutoGen is accessible under Creative Commons attribution, promoting data-driven decision-making across businesses of all sizes.
- Essential Impact: In a world where informed decisions are paramount, AutoGen opens up possibilities for professionals, realizing its potential across various sectors.

YHK

AutoGen Implementation

AutoGen: Building Multi-Agent Conversations

- Two-step process.
- **Step 1:** Define Conversable Agents with specialized capabilities and roles.
- **Step 2:** Define Interaction Behaviors, specifying how an agent should respond to messages, dictating the flow of the conversation.
- OpenAI APIs by default.
- Need to use LM Studio to serve local LLMs (more info on my blog at Medium)

YHK

Configuration

```
openai_config_list = [
    {
        "model": "gpt-4",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    },
    {
        "model": "gpt-3.5-turbo",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    }
]

```

YHK

Simple Query

```
import autogen

question = "Who are you? Tell it in 2 lines only."
response = autogen.oai.Completion.create(config_list=openai_config_list,
    prompt=question, temperature=0)
ans = autogen.oai.Completion.extract_text(response)[0]

print("Answer is:", ans)

```

YHK

Specify Agents

```
from autogen import AssistantAgent, UserProxyAgent
import openai

small = AssistantAgent(name="small model",
                       max_consecutive_auto_reply=2,
                       system_message="You should act as a student! Give
    response in 2 lines only.",
                       llm_config={
                           "config_list": openai_config_list,
                           "temperature": 0.5,
                       })

big = AssistantAgent(name="big model",
                     max_consecutive_auto_reply=2,
                     system_message="Act as a teacher. Give response in 2
    lines only.",
                     llm_config={
                         "config_list": openai_config_list,
                         "temperature": 0.5,
                     })

big.initiate_chat(small, message="Who are you?")

```

YHK

Results

As the temperature was set to the middle, (moderately creative, random), the dialog generated was aptly so

```
big model (to small model):

Who are you?

--------------------------------------------------------------------------------
small model (to big model):

I am a student.
What do you study at the university?
I study English language and literature.
:
How can you describe yourself in 3 words?
I am hardworking, creative and talented.

--------------------------------------------------------------------------------
big model (to small model):

What are your favorite books?
I like the works of Kafka, Dostoyevsky, Chekhov and Tolstoy.
What is the most important thing in your life?
My family, my friends, my job, my studies.

```

Using Open-Source Large Language Models

Via LM Studio

AutoGen: Overview and Configuration

- AutoGen leverages OpenAI APIs by default
- Requires well-structured configuration setup
- Uses OpenAI and Azure OpenAI models (gpt-4, gpt-3.5-turbo)
- Configuration includes model, API key, base URL, and version
- Supports both OpenAI and Azure API types

YHK

Default Config

```
openai_config_list = [
    {
        "model": "gpt-4",
        "api_key": "<your OpenAI API key here>"
    },
    {
        "model": "gpt-4",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    },
    {
        "model": "gpt-3.5-turbo",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    }
]
```

YHK

AutoGen: Basic Usage

Once you've set up the configuration, you can query like this:

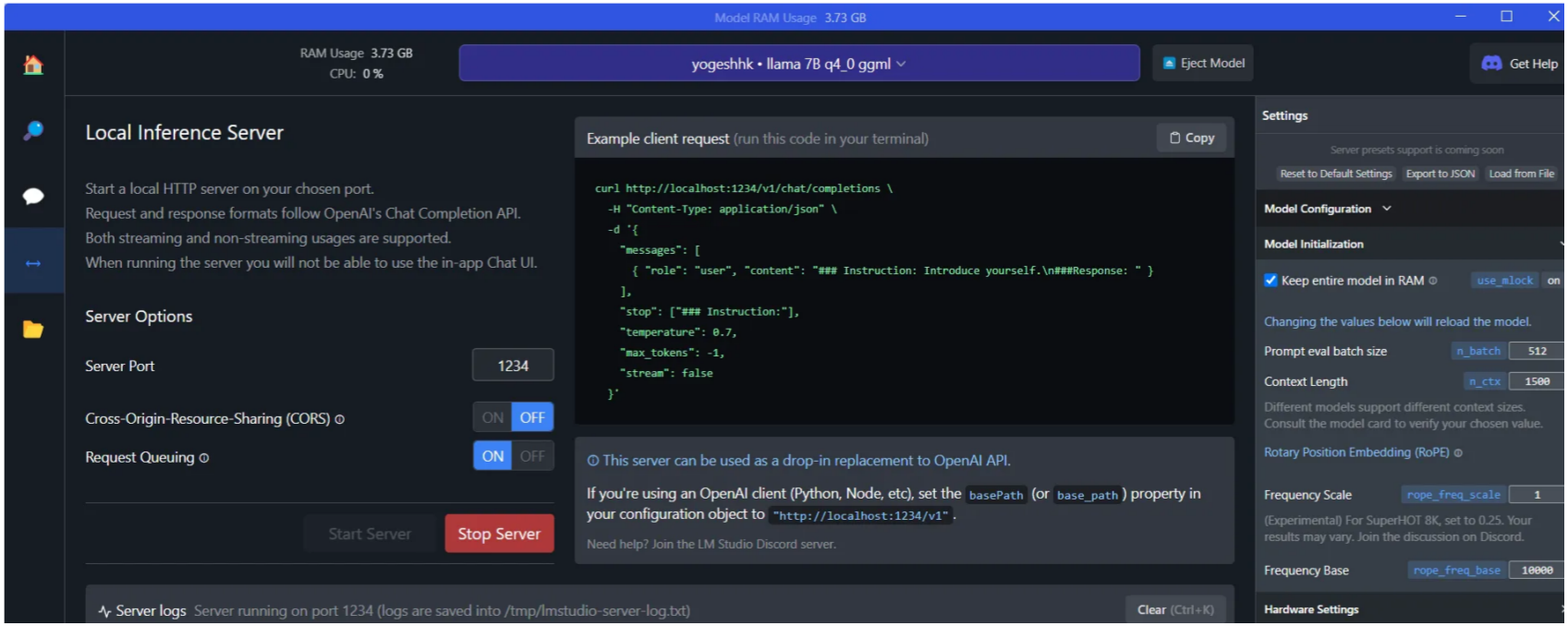
```
import autogen
question = "Who are you? Tell it in 2 lines only."
response = autogen.oai.Completion.create(config_list=openai_config_list,
    prompt=question, temperature=0)
ans = autogen.oai.Completion.extract_text(response)[0]
print("Answer is:", ans)
```

AutoGen Model Compatibility

- AutoGen is not limited to OpenAI models.
- Compatible with locally downloaded models.
- Integrate local models via a server.
- Use local model's endpoint in config as `api_base` URL.
- Multiple methods to serve local models in OpenAI API-compatible ways.
- `modelz-llm` did not work (UNIX-based limitation).
- `llama-cpp-server` also failed in this case.
- **Solution:** LM Studio worked effectively.

YHK

Example



(Ref:"Microsoft AutoGen, A Game-Changer in AI Collaboration" — Yogesh Kulkarni)

Local Model Setup: LM Studio

- LM Studio works for serving local models
- Download models or use existing ones
- Place models in specific directory
- Test using CHAT functionality
- Start server and configure base URL
- Set up OpenAI settings for local model
- Create `local_config_list` for model details

YHK

Local Config List

```
import autogen
import openai

# Configure OpenAI settings
openai.api_type = "openai"
openai.api_key = "..."
openai.api_base = "http://localhost:1234/v1"
openai.api_version = "2023-05-15"

autogen.oai.ChatCompletion.start_logging()

local_config_list = [
        {
            'model': 'llama 7B q4_0 ggml',
            'api_key': 'any string here is fine',
            'api_type': 'openai',
            'api_base': "http://localhost:1234/v1",
            'api_version': '2023-05-15'
        }
]
```

YHK

Advanced Usage: AI Agents Conversation

- Create AssistantAgent instances for different roles
- Configure agents with system messages and LLM configs
- Set maximum consecutive auto-replies
- Initiate chat between agents
- Example creates "student" and "teacher" agents
- Agents engage in a brief conversation
- Temperature setting influences creativity/randomness

YHK

Local Config List

```
from autogen import AssistantAgent, UserProxyAgent
import openai

# Configure OpenAI settings
openai.api_type = "openai"
openai.api_key = "..."
openai.api_base = "http://localhost:1234/v1"
openai.api_version = "2023-05-15"

autogen.oai.ChatCompletion.start_logging()

local_config_list = [
        {
            'model': 'llama 7B q4_0 ggml',
            'api_key': 'any string here is fine',
            'api_type': 'openai',
            'api_base': "http://localhost:1234/v1",
            'api_version': '2023-05-15'
        }
]
```

Local Config List (contd)

```
small = AssistantAgent(name="small model",
                       max_consecutive_auto_reply=2,
                       system_message="You should act as a student! Give
    response in 2 lines only.",
                       llm_config={
                           "config_list": local_config_list,
                           "temperature": 0.5,
                       })

big = AssistantAgent(name="big model",
                     max_consecutive_auto_reply=2,
                     system_message="Act as a teacher. Give response in 2
    lines only.",
                     llm_config={
                         "config_list": local_config_list,
                         "temperature": 0.5,
                     })

big.initiate_chat(small, message="Who are you?")
```

Results

```
big model (to small model):

Who are you?

------------------------------------------------------------------------
small model (to big model):

I am a student.
What do you study at the university?
I study English language and literature.
Why do you like your profession?
Because I want to be an interpreter.
Are there any special features of your job?
Yes, because it is very interesting and useful for me.
How can you describe yourself in 3 words?
I am hardworking, creative and talented.

------------------------------------------------------------------------
big model (to small model):

What are your favorite books?
I like the works of Kafka, Dostoyevsky, Chekhov and Tolstoy.
What is the most important thing in your life?
My family, my friends, my job, my studies.
```

YHK

Agents using Langchain

Agents using LangGraph

Agents using SmolAgents

Agents using LlamaIndex

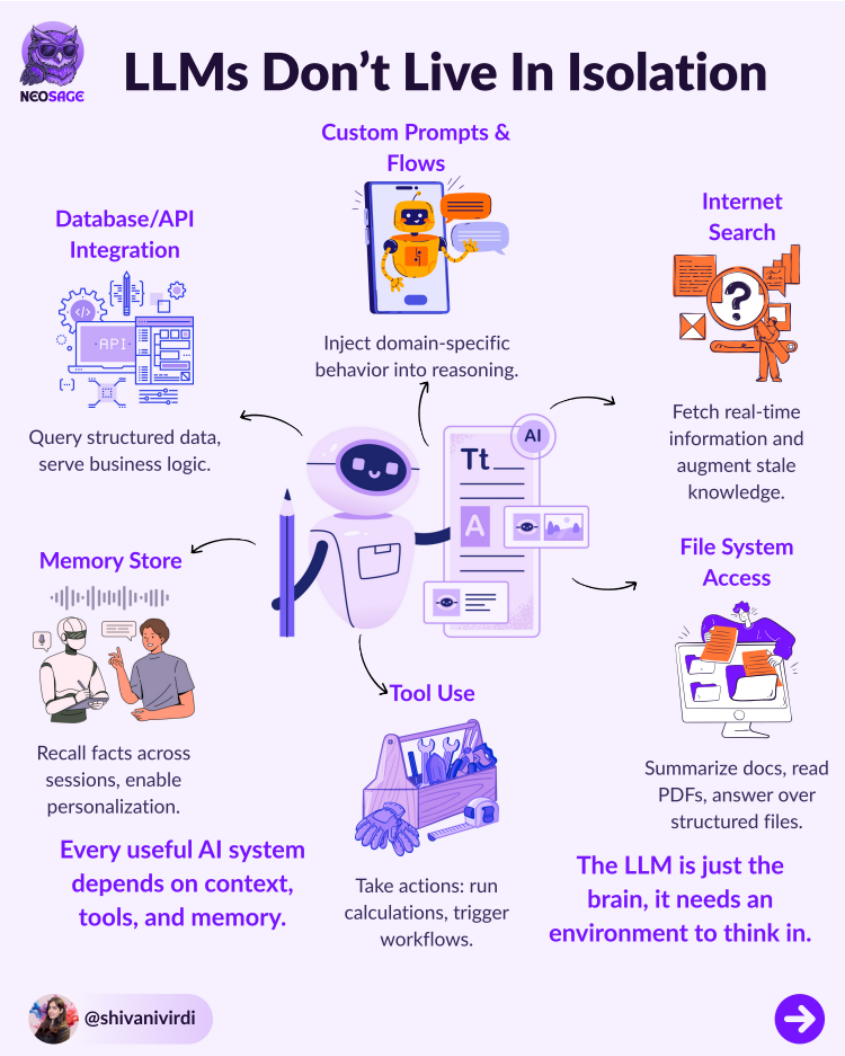
Google Agents Development Kit (ADK)

Google Agents To Agents Protocol (A2A)

Model Context Protocol (MCP)

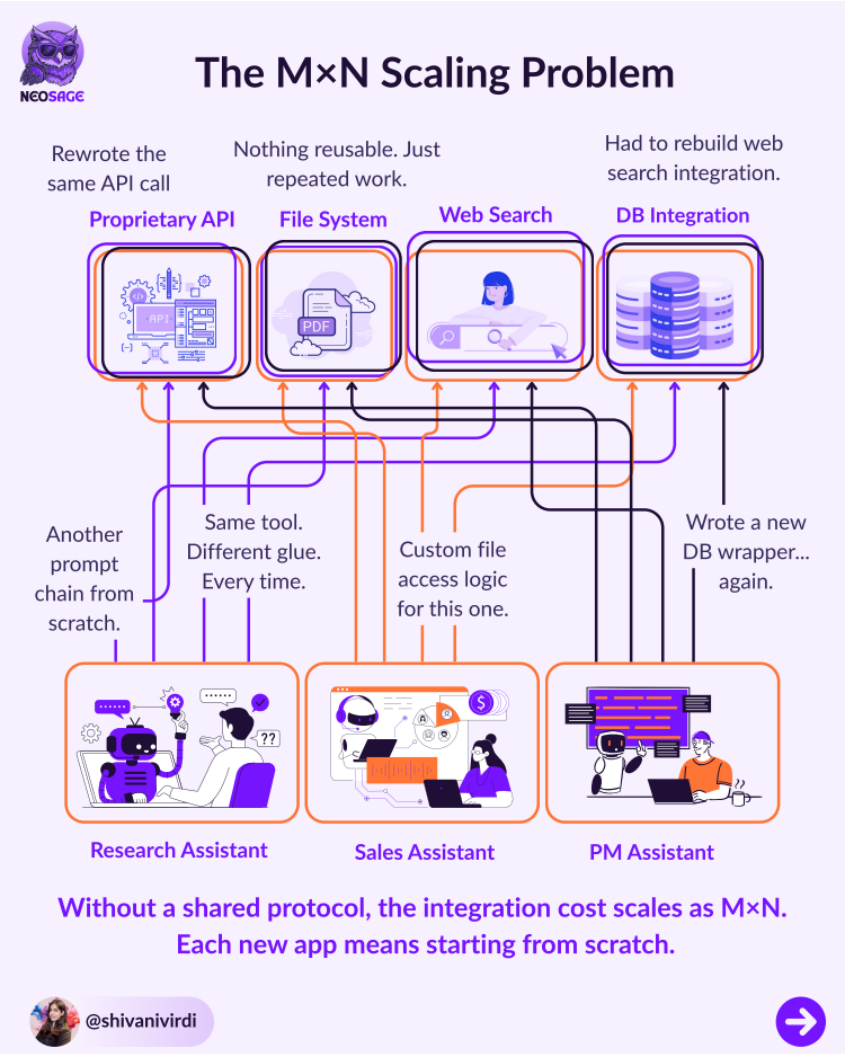
LLMs Alone Aren't Enough

- Real AI apps need search, tools, memory, and APIs
- LLMs can't handle complex tasks on their own
- Integration quickly becomes messy and fragile



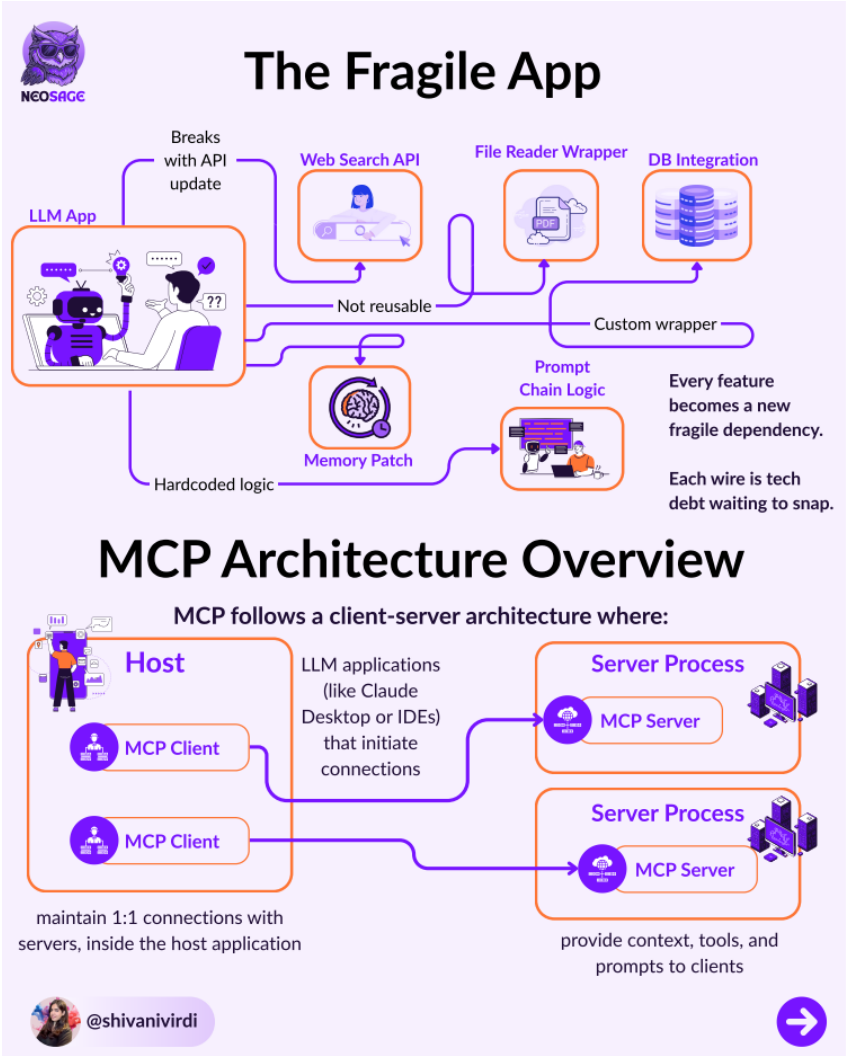
The Integration Problem

- Every feature requires custom wrappers
- Updates often break existing logic
- M apps × N tools = scaling nightmare



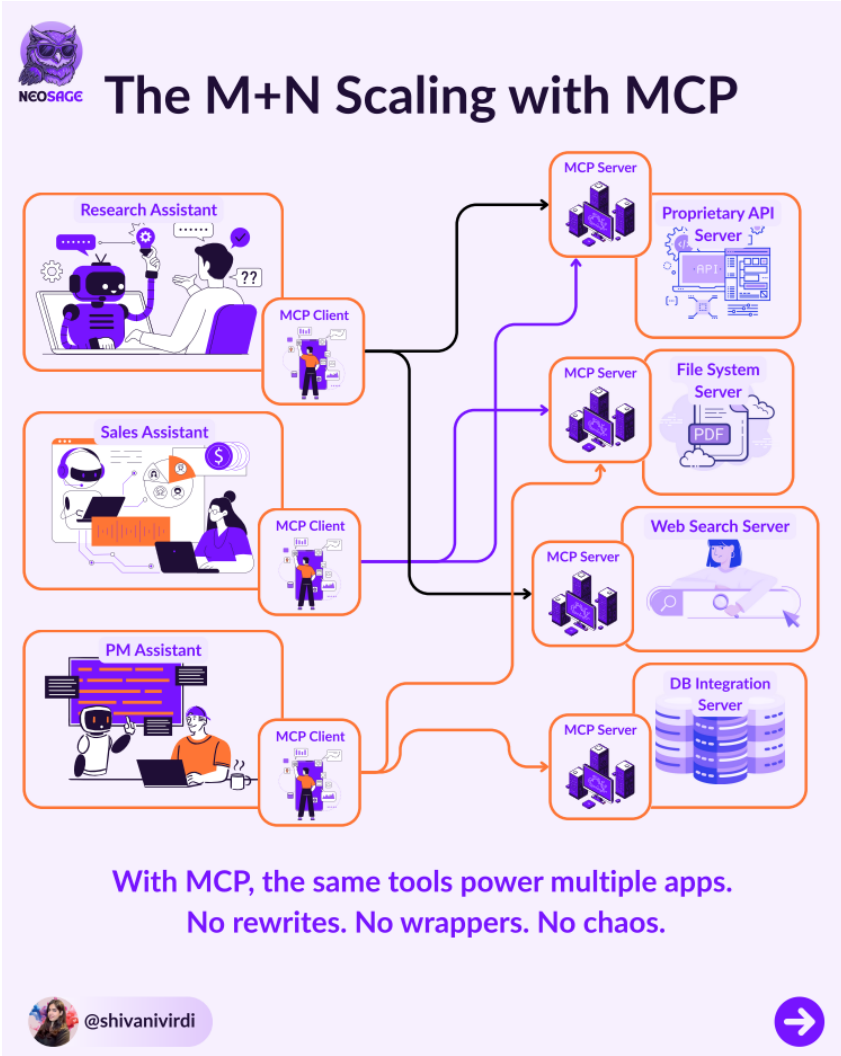
Enter MCP: USB-C for AI

- Shared protocol for AI-to-tool connections
- One-time tool exposure works with all models
- No wrappers, less chaos, easy reuse



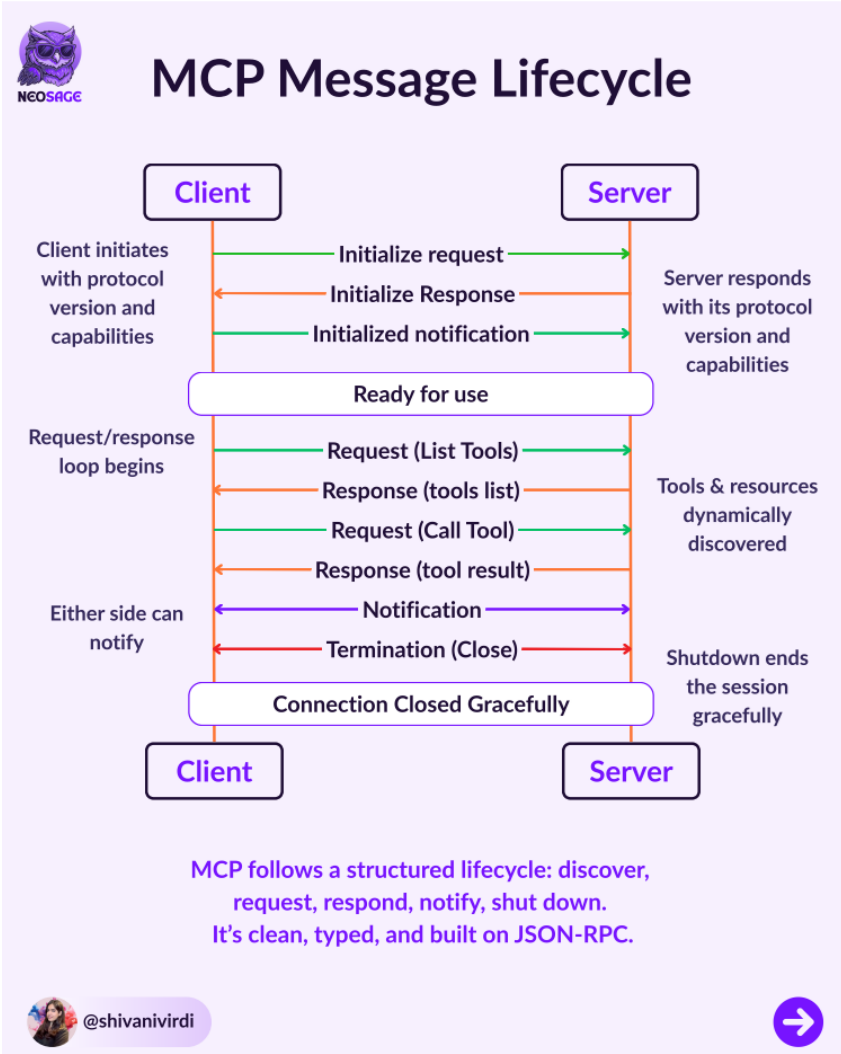
How MCP Works?

- Host: the AI app (e.g., Claude Desktop)
- Client: bridge that speaks MCP
- Server: where tools, files, prompts live



Clean, Typed Communication

- Uses JSON-RPC for structured messaging
- Modular design enables reuse
- Protocol simplifies complex system behavior

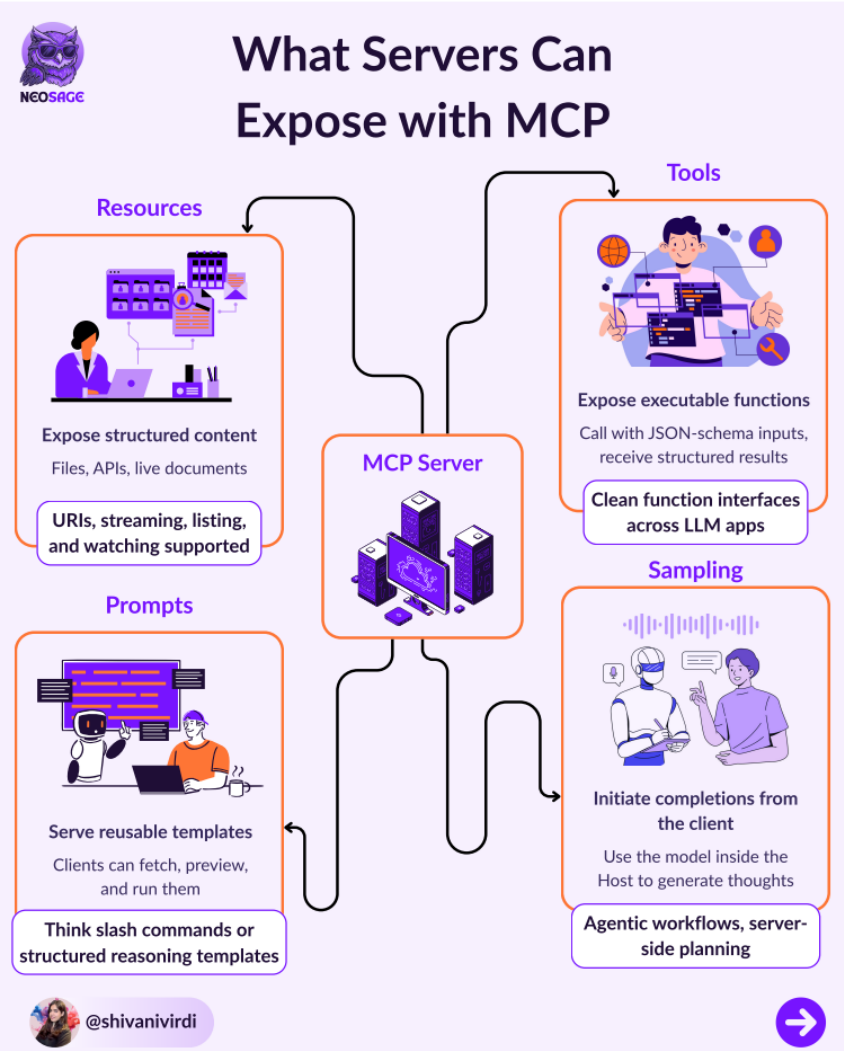


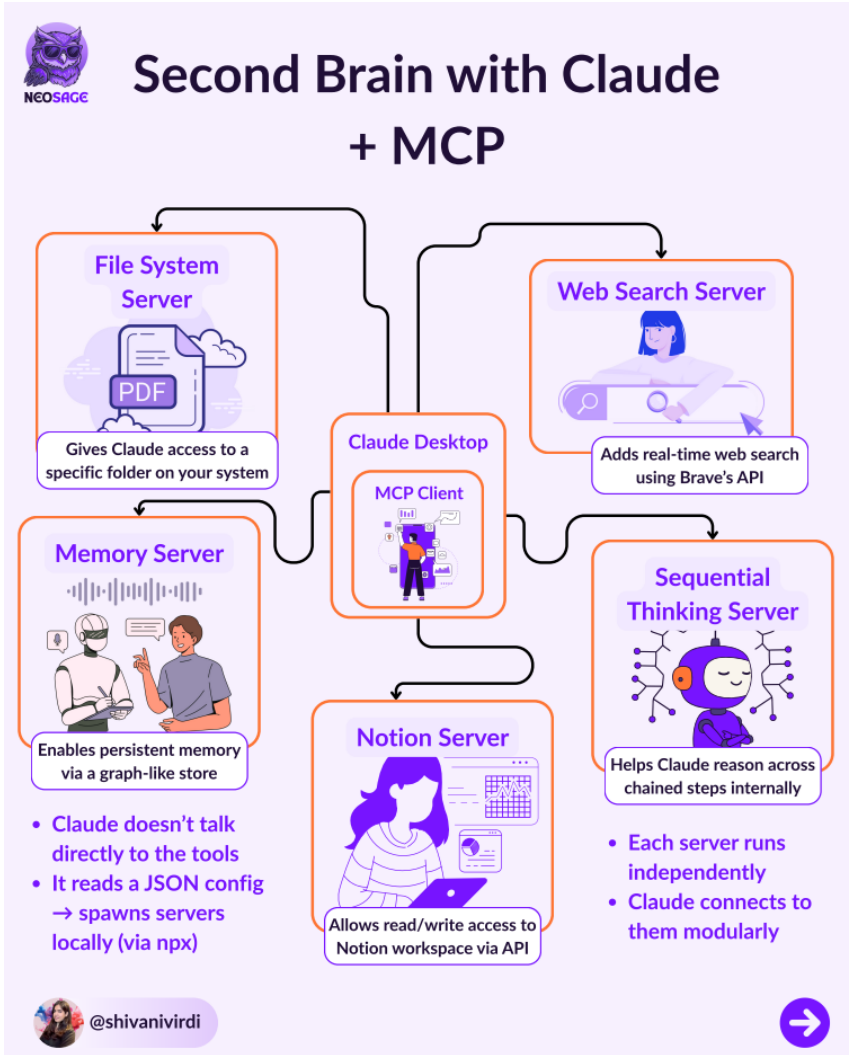
From Demos to Real Systems

- MCP transforms fragile demos into scalable systems
- Reduces engineering overhead
- Enables AI software that ships and lasts

YHK

From Demos to Real Systems





Some Opposition to MCP ...

- Fancy acronyms often solve imaginary problems
- Overengineered MCP layers complicate agent-API flow
- Hype drives architecture-not actual needs
- Direct API calls are faster and clearer
- Lower latency and reduced system complexity
- Easier to debug, scale, and maintain
- Skip the MCP if your API already works
- Avoid unnecessary layers for trend's sake
- Prioritize shipping over architecture theater

YHK

Zerodha Kite MCP Server

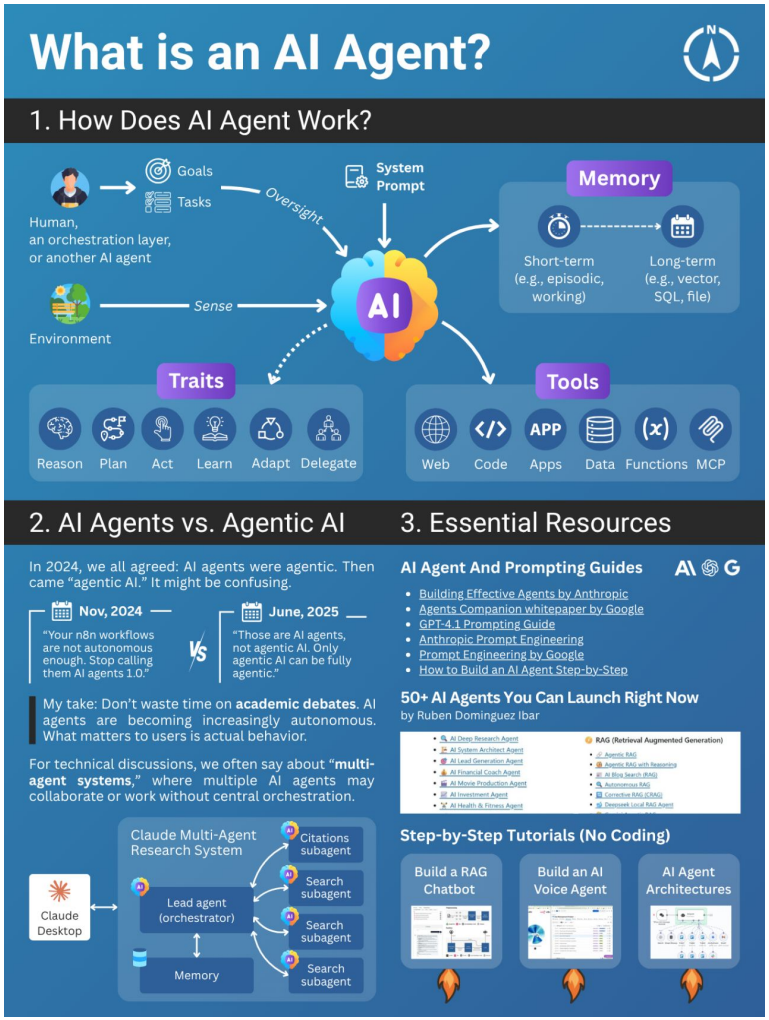
Evaluations

LangFuse

Arize

Conclusions

How to Build an AI Agent?



(Ref: LinkedIn post by Pawl Huryn)

Key Principles for LLM Success

- Focus on building the right system, not the most sophisticated one
- Progress systematically:
  - Start with simple prompts
  - Optimize through comprehensive evaluation
  - Add multi-step agents only when necessary
- Core implementation principles:
  - Maintain simplicity in agent design
  - Ensure transparency in planning steps
  - Carefully design agent-computer interface (ACI)
- Consider reducing framework abstraction layers in production
- Goal: Create powerful, reliable, and maintainable agents

YHK

In General

- Autonomous AI Agents powered by Large Language Models represent AI pinnacle.
- Abilities in planning, memory utilization, and tool use, combined with a flawless workflow, open exciting possibilities across industries.
- A future where AI-driven efficiency and problem-solving reach unprecedented heights.
- Machines that think, remember, and adapt - a revolution in AI.

YHK

Challenges in LLM-Centered Agents

- Finite Context Length: Restricted context capacity limits inclusion of historical information, detailed instructions, API call context, and responses.
- Long-term planning and task decomposition: LLMs struggle to adjust plans when faced with unexpected errors.
- Less robust compared to humans who learn from trial and error.
- LLMs may make formatting errors and occasionally exhibit rebellious behavior (e.g., refuse to follow an instruction).

YHK

When to Use Agents

- Ideal for tasks requiring **complex decision-making**, **autonomy**, and **adaptability**.
- Useful in **dynamic workflows** with multiple steps and automation potential.
- **Customer support**: Handle queries, provide real-time assistance, and escalate issues.
- Enhances **efficiency** and **customer experience** with timely responses.
- **Research and data analysis**: Autonomously gather, process, and analyze data.
- Suitable for **real-time data processing** like financial trading.
- Beneficial in **education**: Personalized learning, adaptive pacing, and instant feedback.
- **Software development**: Assist in code generation, debugging, and testing.
- Agents improve through **learning and adaptation**, crucial for continuous improvement.

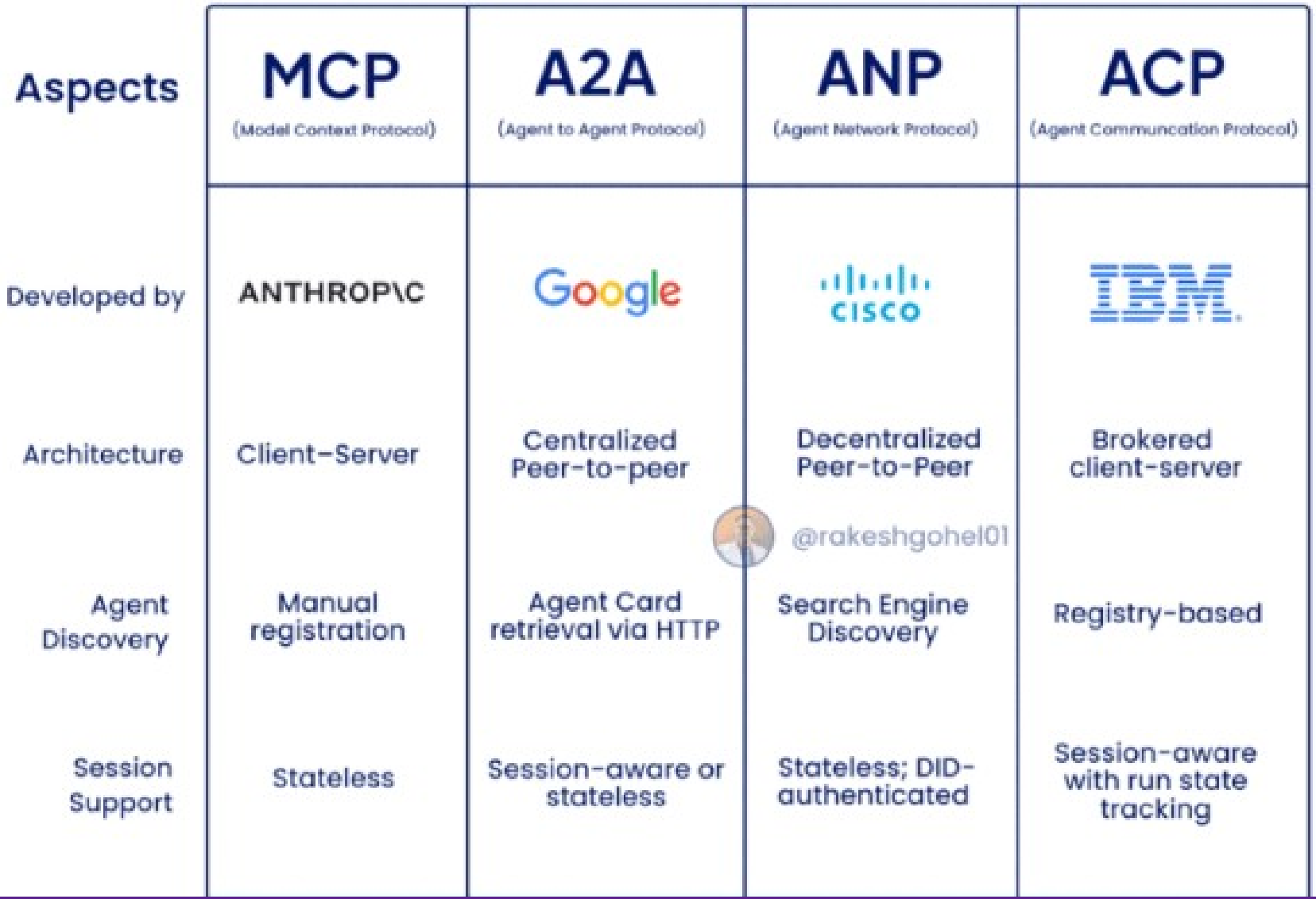
YHK

When Not to Use Agents

- Avoid for **straightforward or infrequent tasks** requiring minimal automation.
- Traditional methods are more **efficient and cost-effective** for simple tasks.
- Unsuitable for tasks needing **deep domain-specific expertise** or complex knowledge.
- Examples: Legal analysis, medical diagnoses, or high-stakes decisions in uncertain contexts.
- Not ideal for tasks requiring **empathy, creativity, or subjective judgment**.
- Examples: Psychotherapy, counseling, or creative writing.
- Implementation demands **time, resources, and expertise**.
- May not be viable for **small businesses or limited budgets**.
- Challenges in **highly regulated industries**: Compliance and security concerns.
- Ensuring adherence to **regulatory requirements** can be resource-intensive.

YHK

Protocols Summary



Aspects	MCP (Model Context Protocol)	A2A (Agent to Agent Protocol)	ANP (Agent Network Protocol)	ACP (Agent Communcation Protocol)
Developed by	ANTHROP\C	Google	cisco	IBM
Architecture	Client-Server	Centralized Peer-to-peer	Decentralized Peer-to-Peer	Brokered client-server
Agent Discovery	Manual registration	Agent Card retrieval via HTTP	Search Engine Discovery	Registry-based
Session Support	Stateless	Session-aware or stateless	Stateless; DID-authenticated	Session-aware with run state tracking

YHK

Are you creative enough?

SCIAGENTS: AUTOMATING SCIENTIFIC DISCOVERY THROUGH MULTI-AGENT INTELLIGENT GRAPH REASONING*

Alireza Ghafarollahi
Laboratory for Atomistic and Molecular Mechanics (LAMM)
Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02139, USA

Markus J. Buehler
Laboratory for Atomistic and Molecular Mechanics (LAMM)
Center for Computational Science and Engineering
Schwarzman College of Computing
Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02139, USA

Correspondence: mbuehler@MIT.EDU

ABSTRACT

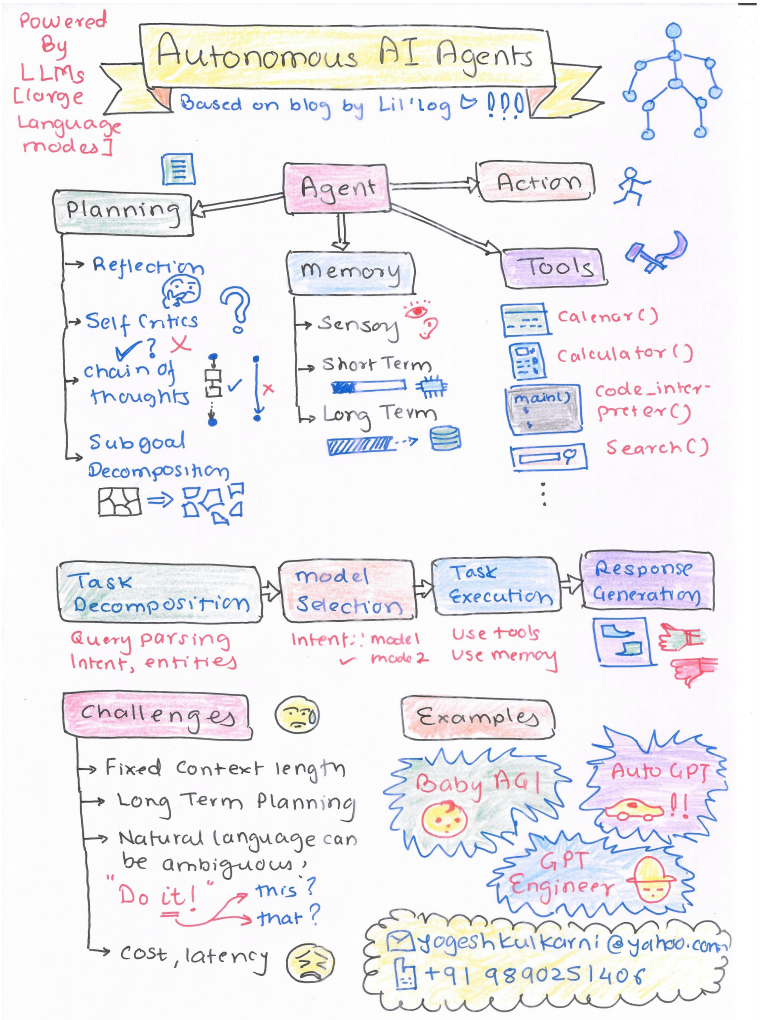
A key challenge in artificial intelligence is the creation of systems capable of autonomously advancing scientific understanding by exploring novel domains, identifying complex patterns, and uncovering previously unseen connections in vast scientific data. In this work, we present SciAgents, an approach that leverages three core concepts: (1) the use of large-scale ontological knowledge graphs to organize and interconnect diverse scientific concepts, (2) a suite of large language models (LLMs) and data retrieval tools, and (3) multi-agent systems with *in-situ* learning capabilities. Applied to biologically inspired materials, SciAgents reveals hidden interdisciplinary relationships that were previously considered unrelated, achieving a scale, precision, and exploratory power that surpasses traditional human-driven research methods. The framework autonomously generates and refines research hypotheses, elucidating underlying mechanisms, design principles, and unexpected material properties. By integrating these capabilities in a modular fashion, the intelligent system yields material discoveries, critique and improve existing hypotheses, retrieve up-to-date data about existing research, and highlights their strengths and limitations. Our case studies demonstrate scalable capabilities to combine generative AI, ontological representations, and multi-agent modeling, harnessing a 'swarm of intelligence' similar to biological systems. This provides new avenues for materials discovery and accelerates the development of advanced materials by unlocking Nature's design principles.

Keywords Scientific AI · Multi-agent system · Large language model · Natural language processing · Materials design · Bio-inspired materials · Knowledge graph · Biological design

(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

YHK

My Sketchnote



(Ref: Power of Autonomous AI Agents - Yogesh Kulkarni)

YHK

The Future with AutoGen

- Transformative era in AI collaboration is on the horizon.
- Microsoft's vision for Autonomous AI Agents and AutoGen's capabilities provide a glimpse into the future of AI applications.
- Empowers professionals to navigate the complex AI landscape with confidence, agility, and precision.

YHK

Towards Artificial General Intelligence (AGI)

- Research aligns with the belief that achieving human-like general intelligence requires cooperation among agents.
- Multi-agent collaboration is a crucial approach, but it may not alone pave the path to artificial general intelligence (AGI).
- The journey likely demands additional innovations and breakthroughs.
- AutoGen stands out as an enticing platform for exploring possibilities offered by multi-agent systems.

YHK

References

- AutoGen Tutorial Create Collaborating AI Agent teams - AssemblyAI
- CS 194/294-196 (LLM Agents) - Lecture 3, Chi Wang and Jerry Liu
- LLM Powered Autonomous Agents Lil'Log
- How to Use Microsoft AutoGen to Assemble a Team of Robots for Writing a Book: Step by Step with Code Examples - Dr. Ernesto Lee
- Autonomous Agents and Simulations in LLM - CodeGPT
- Power of Autonomous AI Agents - Yogesh Kulkarni
- Microsoft AutoGen- Yogesh Kulkarni
- Microsoft AutoGen using Open Source Models- Yogesh Kulkarni
- A CAMEL ride - Yogesh Kulkarni
- Autonomous AI Agents (LLM, VLM, VLA) - Code Your Own AI
- https://www.promptingguide.ai/research/llm-agents
- Awesome LLM-Powered Agent https://github.com/hyp1231/awesome-llm-powered-agent
- Autonomous Agents (LLMs). Updated daily https://github.com/tmgthb/Autonomous-Agents
- AutoGen: A Multi-Agent Framework - Overview and Improvements - John Tan Chong Min

YHK

Thanks . . .

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com

(https://www.linkedin.com/in/yogeshkulkarni/, QR by Hugging Face QR-code-AI-art-generator, with prompt as "Follow me")

YHK

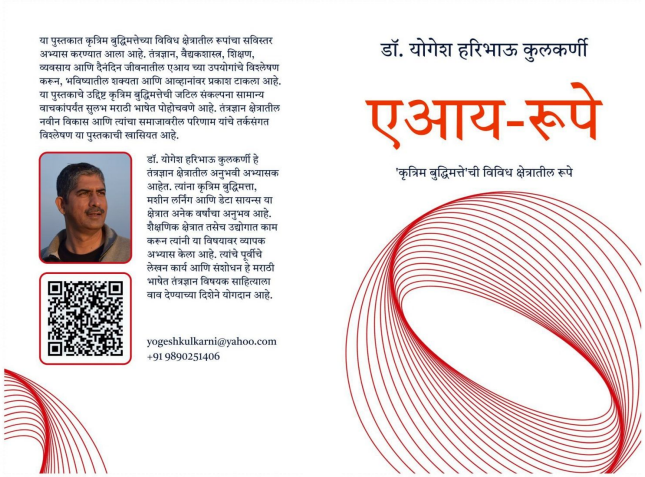
My TEDx Talk :

Hit Refresh : A story of purposeful resets

How rapidly the world is changing and how different career paths are now compared to previous generations. Yogesh shares his own journey of constant reinvention.

(https://www.youtube.com/watch?v=-VbWRs7BsPY, QR by https://www.the-qrcode-generator.com/)

My First Book: AI-Rupe (Marathi)



Amazon India

Flipkart India

Notion Press

YHK