# Multi-agent Decision Systems

Yogesh Haribhau Kulkarni

YHK

# Outline



YHK

# About Me

# Yogesh Haribhau Kulkarni

Bio:

- 20+ years in CAD/Engineering software development
- Got Bachelors, Masters and Doctoral degrees in Mechanical Engineering (specialization: Geometric Modeling Algorithms).
- Currently doing Coaching in fields such as Data Science, Artificial Intelligence Machine-Deep Learning (ML/DL) and Natural Language Processing (NLP).
- Feel free to follow me at:
  - Github (github.com/yogeshhk)
  - LinkedIn (www.linkedin.com/in/yogeshkulkarni/)
  - Medium (yogeshharibhaukulkarni.medium.com)
  - Send email to yogeshkulkarni at yahoo dot com

Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.

YHK

# Introduction

# Classical idea of Agents

- Agents are autonomous
- Can look at their environment and analyze the situation
- Make comprehensive plans to achieve specific goals
- Actually take action to execute those plans
- Agents bridge the gap between answering and doing

YHK

# Welcome to AI Agents

- Agents were there from 1950's but they are effective because of LLMs.
- Agents are systems where LLMs dynamically direct their own paths and tool usage
- Agents can have autonomous-ness or predefined workflow paths
- Essential component in modern AI systems with varying degrees of autonomy
- Unlike LLMs that just respond to prompts, agents do things
- Not just everyday chatbots, systems that reason, plan, and take action
- Can take on complex multi-step tasks autonomously or with human-in-loop
- Technology is advancing rapidly from conversational to agentic AI
- AI agents represent one of the most exciting frontiers in AI
- 2025 is the year of AI agents.

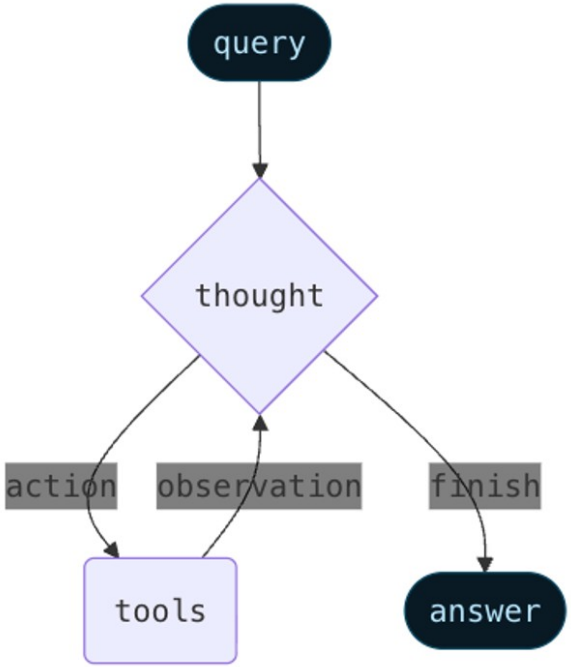
YHK

# The Impossible Job

- If your boss tasks you with planning a massive get-together
- Must prepare a guest list and plan fancy menu
- Needs to find entertainment for the event
- A simple chatbot (with answering LLM, not reasoning LLM) cannot handle these complex requirements
- You need an AI agent, not just responses, but actions
- Agents have a Reasoning LLM as brain and tools as its hands-legs.
- AI agents improve with reasoning, acting, and memory components. (ReAct = Reasoning + Acting)

YHK

# ReAct (Reasoning + Acting) Agent




A ReAct agent selects tools by thinking about the request, selecting a tool, observing its result, and iterating until the request is fulfilled.

Based on paper:
ReAct: Synergizing Reasoning and Acting in Language Models
https://arxiv.org/abs/2210.03629

Diagram from Langchain docs: https://docs.langchain.com/oss/python/langchain-agents

(Ref: Python + AI Agents - Microsoft)

YHK

# The Evolution of AI Capabilities

- **Traditional Programming:** Needed code to operate
- **Traditional ML:** Needed feature engineering
- **Deep Learning:** ML with Neural networks, no feature engineering
- **Foundational Models:** Many tasks single model
- **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 Agents Work?

- Agent acts, take you from one state to the other state(ReAct paper: Reasoning and Action),
- It can plan and make decisions, provides value by workflow automation.
- Agents have access to tools (ToolFormer paper) e.g. Search APIs, booking, send email etc.
- Interacting of external environment and other Agents, etc.
- Memory to keep the history of conversations/actions done so far.
- May have human-in-loop to keep it sane in the wild-world.

YHK

# The Agent's Fundamental Game Loop

- Not a one-and-done action, but a continuous reasoning loop
- Similar to a programming while loop that keeps iterating
- **Thought**: Analyzes situation and plans next step
- **Action**: Calls specific tools to execute the plan
- **Observation**: Examines results of the action taken
- Cycle repeats: Thought $\rightarrow$ Action $\rightarrow$ Observation
- Continues until the task is completely accomplished
- This loop enables continuous adaptation and problem-solving

YHK

# Agent's Inner Monologue

- Agents have visible thought processes before taking action
- Example: "User wants weather in New York. I have a tool for that".
- "My first move is to call the weather API".
- Internal planning step makes agents more than reactive programs
- Reasoning through problems before execution
- This deliberation distinguishes agents from simple scripts
- Shows intelligent decision-making rather than blind execution

YHK

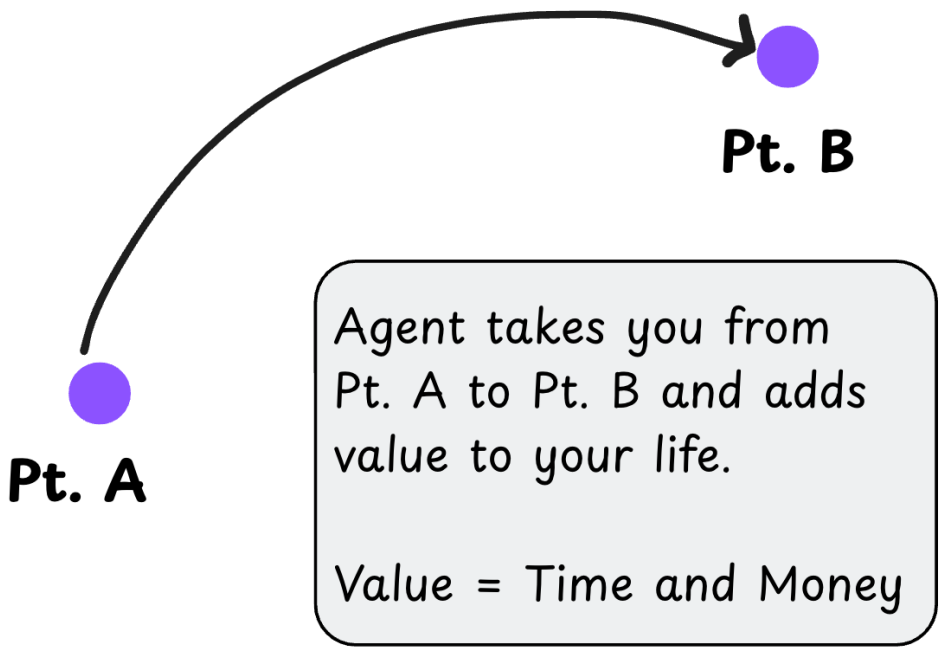
# How Do Agents Take Action?

- 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

# Defining AI Agents with an example

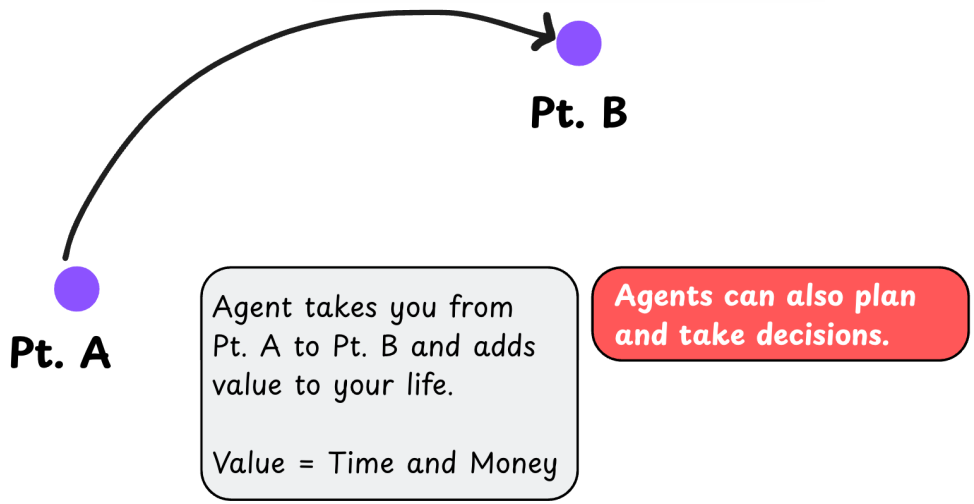
- Planning a trip involves many complex tasks
- Point A: Just discussing the trip
- Point B: All bookings and itinerary ready
- AI Agents aim to take you from A to B
- First requirement: Agent adds value by saving time/money




(Ref: Vizuara AI Agents Bootcamp)

# Evolving Definition of Agents

- Not all tools from A to B are agents (e.g., cars)
- Agents must plan and make decisions
- Example: Choosing flights based on budget
- Planning daily itinerary needs contextual judgment
- Second requirement: Agent includes decision-making ability

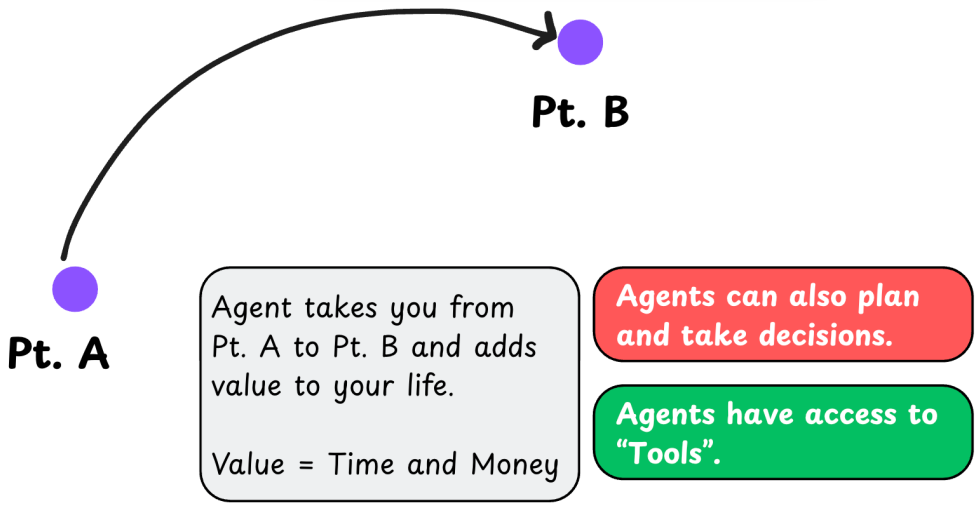



(Ref: Vizuara AI Agents Bootcamp)

YHK

# Agents Need Tools

- Even self-driving cars plan but are not agents
- Agents need access to external tools
- Tools = Access to services (e.g., Gmail, Booking)
- Agents perform tasks using these tools
- Third requirement: Agent adds tool access to capabilities

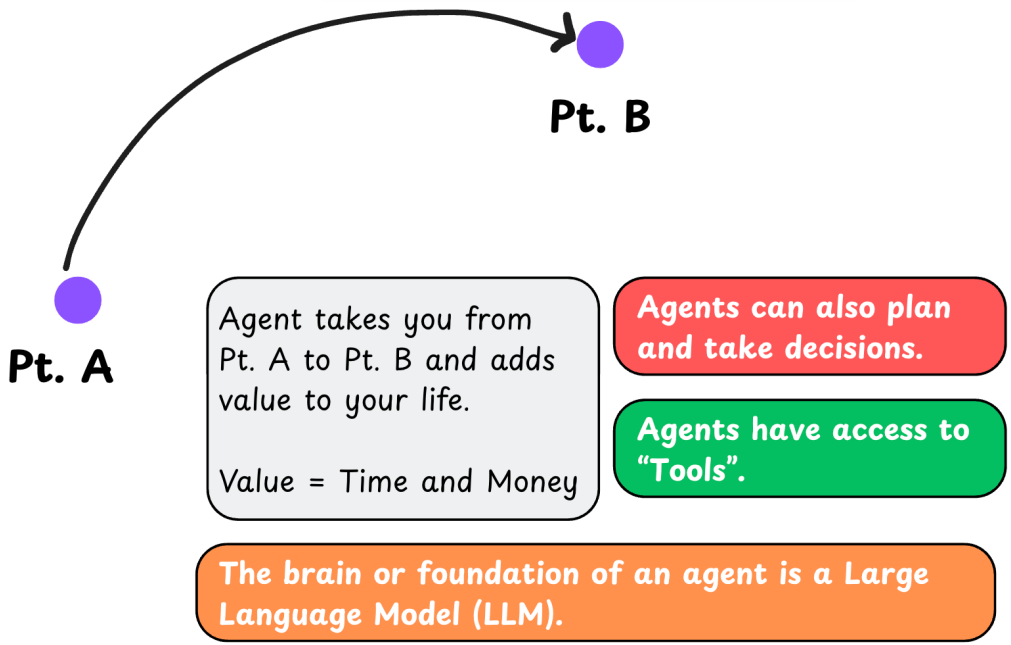



(Ref: Vizuara AI Agents Bootcamp)

YHK

# Rise of LLMs in Agents

- Transformers (2017) enabled powerful LLMs
- LLMs understand and generate human language
- Agents use LLMs for reasoning and planning
- LLMs enable understanding of webpages and writing emails
- Fourth requirement: Agents are LLMs with tools and planning ability




(Ref: Vizuara AI Agents Bootcamp)

YHK

# 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:**

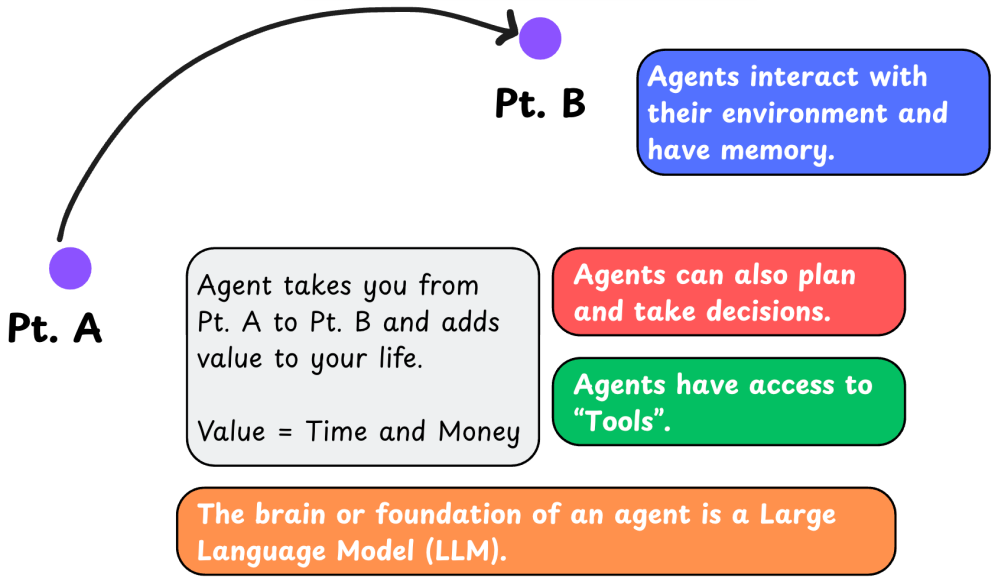
- Systems that complete tasks end-to-end
- Focus on outcomes, not components
- Solve real-world problems
- Provide measurable value

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

YHK

# Final Definition of Agents

- Agents can learn from feedback and environment
- Agents interact with tools, humans, and websites
- They improve with experience (memory)
- Agents evolve over time via memory and feedback
- Fifth requirement: LLMs + Tools + Planning + Learning




(Ref: Vizuara AI Agents Bootcamp)

YHK

# Understanding Agency

- Agency = Level of autonomy an agent has
- Low agency → less value
- High agency → high value
- More autonomous agents can handle complex tasks
- Agency is key to measuring agent usefulness

| Agency Level | Description | Name | Example |
|---|---|---|---|
| ●○○○○ | Agent does not influence what happens next | Simple Processor | Grammar checker that rewrites sentences |
| ●●○○○ | Agent determines basic control flow | Router | Customer Query → Tech Support or Sales |
| ●●●○○ | Agent determines function execution | Tool Caller | A smart calendar assistant that spots "let's meet on Tuesday" and books the meeting |
| ●●●●○ | Lays out a short plan and carries it step by step | Multi-step Agent | Personal travel planner that gathers flight options, hotels, local activities |
| ●●●●● | One agentic workflow starts another agentic workflow | Multi-agent | Travel planner agent → Booking agent ← Email agent |

(Ref: Vizuara AI Agents Bootcamp)

YHK

# Tools: The Agent's Hands

- LLM is the agent's brain; tools are its hands
- Tools are functions agents call to interact with the world
- Can search web, run calculations, or query databases
- Bridge between thinking and doing in the real world
- Enable agents to move from planning to execution
- Without tools, agent thoughts would be useless
- Tools provide the interface to external systems and data

YHK

# Two Ways Agents Use Tools

- **JSON Agent**: Writes structured work orders for other systems
- JSON approach requires external system to read and execute
- **Code Agent**: Directly writes and runs code blocks
- Code approach is more direct and powerful
- Code is naturally more expressive than JSON
- Can handle complex logic like loops and conditionals
- Modular, easier to debug, and taps into existing libraries
- Code agents can access thousands of APIs directly

YHK

# Code Agent in Action

- Alfred needs a gala menu - agent has “suggest_menu” tool
- Agent doesn’t just make up suggestions randomly
- Generates and runs actual code to call the specific tool
- Gets real results from the tool execution
- Super direct, efficient, and powerful way to take action
- Code generation enables precise tool interaction
- Results are based on actual tool capabilities, not hallucination

YHK

# Advanced Pattern: Agentic RAG

- Traditional RAG: Retrieval Augmented Generation fetches info before answering
- Agentic RAG supercharges this with intelligent multi-step processes
- Turns retrieval itself into an agent-driven task
- Like having a master researcher on staff
- Doesn't just do one search - runs complete research processes
- Rewrites queries for better results and runs multiple searches
- Uses findings to inform next searches and validates accuracy
- Pulls from both private data and public web sources

YHK

# Multi-Agent Systems: Digital Teams

- Complex problems like finding the missing Batmobile need teams
- Single agents can't handle web searches, calculations, and visualization
- Solution: Build teams of specialized agents
- Manager agent acts as project lead breaking down big tasks
- Delegates work to specialist agents with specific skills
- Web agent handles online searching while manager coordinates
- Manager focuses on big picture and final integration
- Digital division of labor for complex problem solving

YHK

# Papers that Shaped AI Agents

- Core research papers laid the foundation
- Introduced key frameworks and architectures
- Sparked recent boom in agent development
- Include Transformer and Agentic frameworks
- Major driving force in LLM-based agent systems

**Chain-of-Thought Prompting Elicits Reasoning in Large Language Models**

Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou
Google Research, Brain Team
{jasonwei,dennyzhou}@google.com

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao*[,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]
[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

**Toolformer: Language Models Can Teach Themselves to Use Tools**

Timo Schick Jane Dwivedi-Yu Roberto Dessì[†] Roberta Raileanu
Maria Lomeli Luke Zettlemoyer Nicola Cancedda Thomas Scialom
Meta AI Research [†]Universitat Pompeu Fabra

**Generative Agents: Interactive Simulacra of Human Behavior**

Joon Sung Park
Stanford University
Stanford, USA
joonspk@stanford.edu

Joseph C. O'Brien
Stanford University
Stanford, USA
jobrien3@stanford.edu

Carrie J. Cai
Google Research
Mountain View, CA, USA
cjcai@google.com

Meredith Ringel Morris
Google DeepMind
Seattle, WA, USA
merrie@google.com

Percy Liang
Stanford University
Stanford, USA
pliang@cs.stanford.edu

Michael S. Bernstein
Stanford University
Stanford, USA
msb@cs.stanford.edu

(Ref: Vizuara AI Agents Bootcamp)

YHK

# Frameworks

# Python AI Agent Frameworks Overview

- Python offers several frameworks for building single and multi-agent AI systems.
- These frameworks support workflows, tool use, reasoning, and human-in-the-loop capabilities.
- Common applications include autonomous assistants, research copilots, and workflow orchestration.

YHK

# LangChain v1

- **Type:** Multi-agent, graph-based framework.
- **Built on:** LangGraph for agent state management and orchestration.
- **Features:**
  - Agent-centric architecture.
  - Workflow graphs for reasoning and tool execution.
  - Human-in-the-loop integration via LangSmith.
  - Supports OpenAI, Anthropic, Gemini, and local models.

YHK

# Pydantic-AI

- **Type:** Lightweight single/multi-agent framework.
- **Focus:** Strong type safety using Pydantic for structured inputs/outputs.
- **Features:**
  - Declarative agent definition with validation.
  - Easy LLM orchestration and chaining.
  - Built for developers needing data integrity.
- **Ideal for:** AI applications requiring schema consistency and typed reasoning.

YHK

# Crew AI

- **Type:** Multi-agent coordination framework.
- **Concept:** "Crew" of specialized agents working collaboratively.
- **Features:**
  - Role-based agent interactions.
  - Shared memory and task assignment.
  - Supports autonomous workflow creation.
  - Integrates with LangChain, OpenAI, and custom tools.
- **Use Case:** Complex multi-role task solving (research, planning, content generation).

YHK

# Autogen

- **Developed by:** Microsoft Research.
- **Type:** Multi-agent conversational framework.
- **Features:**
  - Agent-to-agent and human-in-loop communication.
  - Supports function calling and code execution.
  - Built-in memory and orchestration logic.
  - Extendable with custom agents and tools.
- **Example Code:**

```
from autogen import AssistantAgent, UserProxyAgent

assistant = AssistantAgent("assistant", llm="gpt-4")
user = UserProxyAgent("user", code_execution=True)

user.initiate_chat(assistant, message="Build a weather app.")
```

YHK

# OpenAI Agents

- **Type:** Hosted agent platform by OpenAI (2024+).
- **Features:**
  - Create, configure, and deploy GPT-based agents.
  - Access via OpenAI API and GPTs interface.
  - Integrates with files, APIs, tools, and memory.
  - Supports human feedback and evaluation.
- **Example Use:** Deploy a custom assistant with knowledge base and tools.

YHK

# Google ADK (Agent Development Kit)

- **Developed by:** Google DeepMind / Gemini ecosystem.
- **Type:** Multi-agent and workflow orchestration framework.
- **Features:**
  - Integrates Gemini 1.5 models with tool-use capabilities.
  - Supports perception, memory, planning, and dialogue.
  - Human-in-the-loop and autonomous modes.
  - Built for scalable, production-grade AI agents.
- **Use Case:** Building Gemini-powered assistants and reasoning systems.

YHK

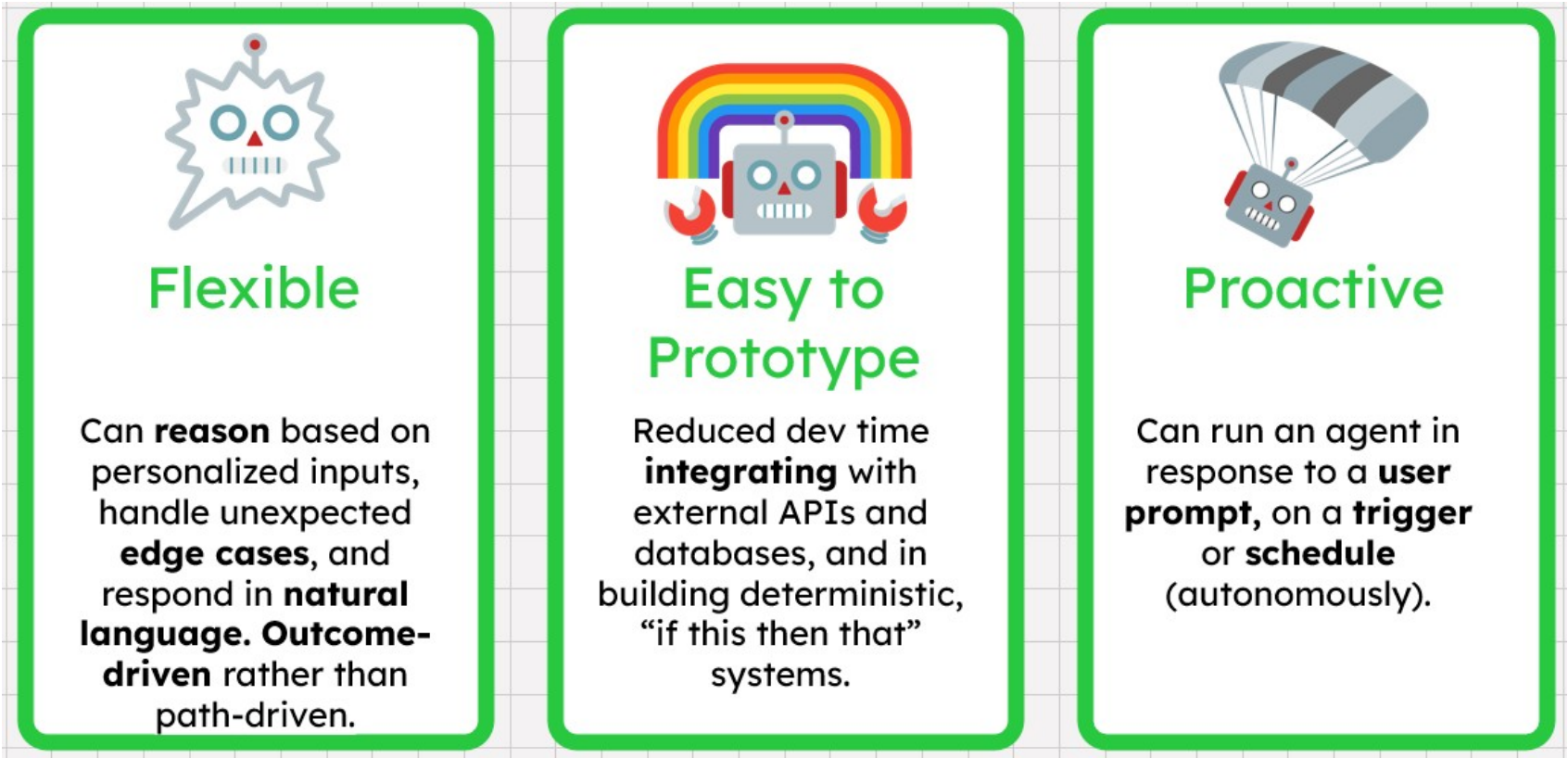
# Comparison Summary

- **LangChain v1:** Graph-based, modular, integrates with LangSmith.
- **Pydantic-AI:** Strong typing, data-safe orchestration.
- **Crew AI:** Role-based multi-agent collaboration.
- **Autogen:** Conversational multi-agent system by Microsoft.
- **OpenAI Agents:** Hosted GPT-based no-code agent deployment.
- **Google ADK:** Gemini-native multi-agent orchestration suite.

YHK

# Final Thoughts

# Why Agents?




(Ref: Google Cloud Labgs H2 India Agents for All)

YHK

# When to Use Agents?

- Best suited for tasks requiring flexibility and model-driven decision-making
- Consider tradeoffs: 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


(Ref: Google Cloud Labgs H2 India Agents for All)

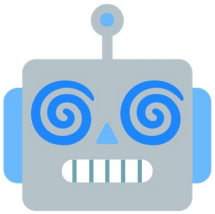
YHK

# When NOT to Use Agents?


(Ref: Google Cloud Labgs H2 India Agents for All)

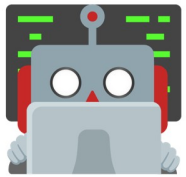
YHK

# Agent Challenges



## Predictability

Agents use LLMs to reason and respond. **LLMs are nondeterministic.**



## Stability

**Complexity** of using many tools, **error-handling, security, privacy**...

+ Framework + protocol landscape is **evolving rapidly.**

## Operations

Tracking calls through a **distributed system** is hard, debugging is tricky, tracking **token** throughput is important (**costs**), **quota,** LLM reasoning can be **opaque**...

(Ref: Google Cloud Labgs H2 India Agents for All)

YHK

# Mitigating Predictability challenges

- choose reasoning models, when possible. (eg. gemini 2.5 flash)
- ground LLM in trusted data to reduce hallucinations (RAG, search)
- guide your agent to think step by step (chain of thought or few-shot prompting)
- use state management and memory.
- implement checks in the agent's workflow: tool-call request validation, safety filters, logic checks...
- test and evaluate your agent.

(Ref: Google Cloud Labgs H2 India Agents for All)

YHK

# Mitigating Stability challenges

- use consistent tool abstractions via Model Context Protocol (MCP)
- when writing your agent's system prompt, be as detailed as possible. (tool descriptions)
- implement or leverage error-handling in your agent (circuit-breakers, retries, hand off to a human in the loop...)
- utilize state management, especially the workflow status field (recover from interruptions, prevent infinite loops)
- be willing to refactor and adjust with new framework features, tool interfaces...

(Ref: Google Cloud Labgs H2 India Agents for All)

YHK

# Mitigating Operations challenges

- in the dev phase, log verbosely.
- build robust test and evaluation datasets. automate these tests on every commit.
- gather necessary metrics, like token throughput to your agent's models, and response code distribution. create alerts if these trip certain thresholds.
- leverage agent framework's tracing features to track model and tool calls, find + address latency bottlenecks.

(Ref: Google Cloud Labgs H2 India Agents for All)

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

# The Big Question

- Agentic AI technology is moving incredibly fast
- Personal AI assistants will soon be capable of complex tasks
- Think beyond simple queries to multi-step accomplishments
- Consider what "impossible" tasks you want to delegate
- What complex, party-of-the-century level challenge will you tackle?
- The era of AI that truly does rather than just discusses
- Prepare for AI assistants that can handle your biggest challenges

YHK

# Introduction to Multi Agent Systems

# From Single to Multi-Agent Systems

- The Evolution of AI Systems:
  - QnA (Atomic Prompts) → Chatbot (Context)
  - RAG (Own Data) → Agents (Actions/Tools)
  - **Multi-Agent OS** (Orchestrated Intelligence)
- Multi-agent systems represent the pinnacle of AI orchestration
- Specialized AI agents collaborate to solve complex problems
- Each agent masters one framework deeply
- Together they achieve what no single expert could produce alone

YHK

# Why Multiple Agents Beat Single Intelligence

- **Cognitive Bias**: Diverse agents challenge assumptions vs single perspective
- **Bounded Rationality**: Each agent specializes, synthesis integrates
- **Expertise Breadth**: Deploy 25+ specialist agents simultaneously
- **Consistency**: Agents maintain analytical rigor across perspectives
- **Parallel Processing**: Simultaneous analysis from multiple frameworks
- **Emotional Detachment**: Pure evaluation on merits without personal stakes
- **Scenario Analysis**: Each agent explores different timelines/strategies

YHK

# The Wisdom of Artificial Crowds

- Based on **Society of Mind** (Marvin Minsky)
- Intelligence emerges from multiple simple processes
- Strategic wisdom emerges from:
  - **Specialized Analysis**: Each agent masters one framework
  - **Diverse Perspectives**: Game theorist vs mental model thinker
  - **Synthesis**: Oracle routes intelligently, Synthesizer integrates
  - **Emergent Insight**: The whole exceeds sum of parts
- Example: Business strategy analyzed by Nash, Prisoner's Dilemma, Coalition, and Mental Model agents
- Result: Strategic recommendation no single expert could produce alone

YHK

# Multi-Agent Systems: Background

# History of Multi-Agent Systems

- **1950s-1970s**: Early AI research on distributed problem solving
- **1980s**: Distributed Artificial Intelligence (DAI) emerges
  - Focus on coordinating multiple AI systems
  - Research in distributed reasoning and planning
- **1990s**: Multi-Agent Systems (MAS) becomes distinct field
  - Agent communication languages (KQML, FIPA-ACL)
  - Belief-Desire-Intention (BDI) architectures
- **2000s-2010s**: Practical applications emerge
  - Robotic swarms, game AI, trading systems
- **2020s**: LLM revolution transforms multi-agent systems
  - Natural language coordination becomes possible
  - Agents can reason, plan, and communicate effectively

YHK

# Why Multi-Agent Systems?

**Limitations of Single Agent Systems**:

- Limited expertise scope - cannot excel at everything
- Single point of failure - no redundancy
- Bounded cognitive capacity - context window limits
- Lack of diverse perspectives - prone to bias

**Advantages of Multi-Agent Systems**:

- **Specialization**: Each agent masters specific domain
- **Parallelization**: Simultaneous processing of subtasks
- **Robustness**: System continues if one agent fails
- **Scalability**: Add agents for new capabilities
- **Modularity**: Easy to update or replace individual agents

YHK

# Multi-Agent System Architectures: Overview

**Three Main Architectural Patterns**:

1. **Hierarchical/Manager-Worker**
   - Central coordinator delegates to specialist agents
   - Clear authority structure, predictable flow
2. **Peer-to-Peer/Collaborative**
   - Agents communicate directly, negotiate solutions
   - Democratic decision-making, emergent behavior
3. **Pipeline/Sequential**
   - Each agent performs stage in workflow
   - Output of one becomes input of next

YHK

# Architecture 1: Hierarchical (Manager-Worker)

**Structure**:

- Manager agent coordinates
- Worker agents specialize
- Clear task delegation
- Centralized synthesis

**Examples**:

- AutoGen's group chat with manager
- CrewAI hierarchical mode
- MADS Oracle-Agent-Synthesizer

**Pros**:

- Clear control flow
- Easy to debug
- Predictable behavior
- Prevents agent conflicts

**Cons**:

- Manager bottleneck
- Less emergent behavior
- Single point of failure
- Limited agent autonomy

YHK

# Architecture 2: Peer-to-Peer (Collaborative)

**Structure**:

- Agents communicate directly
- Democratic coordination
- Emergent solutions
- Consensus building

**Examples**:

- AutoGen's autonomous chat
- MetaGPT role-playing
- ChatDev software team

**Pros**:

- High agent autonomy
- Creative solutions
- No bottlenecks
- Robust to failures

**Cons**:

- Unpredictable behavior
- Hard to control
- May not converge
- Higher token costs

YHK

# Architecture 3: Pipeline (Sequential)

**Structure**:

- Linear workflow stages
- Each agent processes then passes
- Assembly line pattern
- Clear input/output contracts

**Examples**:

- LangGraph sequential chains
- Content creation pipelines
- Data processing workflows

**Pros**:

- Simple to understand
- Easy to optimize stages
- Predictable flow
- Low coordination cost

**Cons**:

- Inflexible
- No parallel processing
- Errors cascade
- Limited adaptability

YHK

# Communication Patterns in Multi-Agent Systems

- **Shared Memory/Blackboard**
  - Agents read/write to common state
  - LangGraph's state management
  - Good for: Coordinated information sharing
- **Message Passing**
  - Direct agent-to-agent communication
  - AutoGen's conversational approach
  - Good for: Negotiation, debate, collaboration
- **Publish-Subscribe**
  - Agents subscribe to topics of interest
  - Event-driven coordination
  - Good for: Decoupled, scalable systems

YHK

# Coordination Mechanisms

**How agents coordinate their actions**:

- **Centralized Planning**
  - Manager creates overall plan, assigns tasks
  - Example: MADS Oracle routing to specialists
- **Distributed Planning**
  - Each agent plans independently, negotiates conflicts
  - Example: Market-based task allocation
- **Reactive Coordination**
  - Agents respond to environment and each other
  - Example: Swarm robotics, stigmergy
- **Contract Net Protocol**
  - Agents bid for tasks, coordinator assigns to best bidder
  - Example: Task allocation in heterogeneous systems

YHK

# Multi-Agent Frameworks Comparison

**AutoGen (Microsoft)**:

- Conversational agents
- Group chat patterns
- Code execution support
- Good for: Iterative tasks

**CrewAI**:

- Role-based agents
- Sequential/hierarchical modes
- Simple configuration
- Good for: Business workflows

**LangGraph**:

- Graph-based workflows
- Conditional routing
- State management
- Good for: Complex orchestration

**MetaGPT**:

- Software team simulation
- Standardized outputs (SOP)
- Document-driven
- Good for: Software development

YHK

# Challenges in Multi-Agent Systems

- **Coordination Overhead**
  - More agents = more communication = higher latency and cost
  - Need efficient routing and synthesis mechanisms
- **Consistency and Conflict Resolution**
  - Agents may produce contradictory recommendations
  - Need robust conflict resolution and voting mechanisms
- **Non-Determinism**
  - LLM variability multiplied across agents
  - Difficult to guarantee reproducible results
- **Debugging and Observability**
  - Hard to trace decisions through multi-agent interactions
  - Need comprehensive logging and visualization tools

YHK

# Best Practices for Multi-Agent Systems

- **Start Simple**: Begin with 2-3 agents, scale gradually
- **Clear Roles**: Define distinct, non-overlapping agent responsibilities
- **Explicit Communication**: Use structured formats (JSON, XML)
- **State Management**: Maintain shared context effectively
- **Termination Conditions**: Prevent infinite loops with clear stopping criteria
- **Human-in-Loop**: Include human oversight for critical decisions
- **Evaluation Metrics**: Define success criteria for system performance
- **Error Handling**: Graceful degradation when agents fail

YHK

# Multi Agent Decision Systems (MADS)

# MADS Core Philosophy

- **Configuration Over Code**
  - Zero code required to build sophisticated systems
  - Define everything in JSON configuration
  - Domain-agnostic framework
- **Multiple Domain Support**
  - Game Theory: 25+ agents for strategic decisions
  - Mental Models: 10+ agents for personal decisions
  - TRIZ Innovation: 40 inventive principles
  - Business Strategy: Porter's Forces, Blue Ocean, etc.
  - Custom Domains: Your expertise, your agents

YHK

# Three-Stage Intelligence Architecture

**Oracle (Classifier) → Specialist Agents (Analyzers) → Synthesizer (Integrator)**

1. **Oracle Agent**: Analyzes problem, routes to 3-7 most relevant specialists
2. **Specialist Agents**: Each applies unique framework in parallel
3. **Synthesizer Agent**: Evaluates, ranks, integrates into unified recommendation

This three-agent core enables:

- Intelligent problem classification
- Parallel expert analysis
- Coherent strategy synthesis

YHK

# How Problems Flow Through MADS

1. **Intake**: User describes strategic situation in natural language
2. **Classification**: Oracle extracts key dimensions (players, timing, uncertainty)
3. **Routing**: Oracle selects most relevant 3-7 specialist agents
4. **Parallel Analysis**: Selected agents simultaneously analyze from unique frameworks
5. **Evaluation**: Synthesizer scores each output on multiple criteria
6. **Integration**: Synthesizer resolves conflicts, finds synergies
7. **Output**: Executive summary + strategic guidance + implementation plan

YHK

# Use Cases: Real-World Applications

# Company-Level Strategic Decisions

**Problem**: "Should we acquire Competitor X or build feature Y internally?"

**MADS Analysis**:

- Game Theory agents evaluate competitive dynamics
- Coalition agent assesses M&A value distribution
- Opportunity cost agent quantifies what you give up
- Second-order thinking agent maps long-term consequences

**Result**: Nuanced recommendation with phased approach and decision triggers

YHK

# Nation-Level Policy Analysis

**Problem**: "How should we respond to trade partner imposing tariffs?"

**MADS Analysis**:

- Repeated game agent models long-term diplomatic dynamics
- Nash equilibrium agent finds stable response strategies
- Bargaining agent identifies negotiation leverage
- Risk assessment agent evaluates escalation scenarios

**Result**: Multi-track strategy with clear red lines and off-ramps

YHK

# Product Innovation

**Problem**: "Design a better smartphone charging solution"

**MADS Analysis**:

- TRIZ agents apply 40 inventive principles
- Systems thinking agent maps user journey and pain points
- First principles agent questions "why charging at all?"

**Result**: Portfolio of innovations from incremental to radical

YHK

# Personal Life Decisions

**Problem**: "Should I take risky startup job or stay in stable corporate role?"

**MADS Analysis**:

- Opportunity cost agent quantifies what each path sacrifices
- Inversion agent identifies how each could fail
- Second-order consequences agent maps ripple effects
- Margin of safety agent recommends de-risking strategies

**Result**: Clear decision framework with contingency plans

YHK

# Domain Frameworks

# Game Theory Frameworks (25+ Agents)

**Cooperative Game Theory**

- Coalition Formation: Shapley values, core stability, optimal alliances
- Bargaining: Nash bargaining solution, BATNA analysis

**Non-Cooperative Game Theory**

- Nash Equilibrium: Pure/mixed strategies, stability analysis
- Dominant Strategy: Mechanism design

**Classic Scenarios**

- Prisoner's Dilemma, Stag Hunt, Chicken, Battle of Sexes

YHK

# Mental Models (10+ Agents)

Cognitive frameworks for clearer thinking:

- **First Principles**: Break down to fundamental truths
- **Second-Order Thinking**: Consequences of consequences
- **Inversion**: Work backwards from failure to avoid it
- **Opportunity Cost**: What you sacrifice by choosing
- **Margin of Safety**: Build buffers for uncertainty
- **Circle of Competence**: Know what you know
- **Compound Interest**: How small advantages accumulate
- **Feedback Loops**: How systems reinforce or balance

YHK

# TRIZ Innovation Principles (40 Agents)

Systematic innovation from studying 200,000+ patents:

- Segmentation, Asymmetry, Dynamization
- Preliminary Action, Cushion in Advance
- Self-Service, Copying, Cheap Disposables
- Nested Doll, Pneumatics, Flexible Shells
- Principle of universality, Mechanical vibration
- Thermal expansion, Strong oxidants
- And 25+ more systematic innovation principles

Each principle provides structured approach to innovation challenges

YHK

# Configuration & Implementation

# Creating Custom Configurations

**Step 1**: Define Your Domain Experts

- Identify specialist perspectives needed
- Example: Business Strategy domain
  - Porter's Five Forces Analyst
  - Blue Ocean Strategist
  - Disruptive Innovation Expert
  - Core Competency Evaluator

**Step 2**: Configure Agents in JSON

- Define agent ID, name, expertise areas
- Specify analysis framework and steps
- Set output format and prompt template

YHK

# Oracle Routing & Synthesizer Evaluation

**Step 3**: Define Oracle Routing Logic

- Set conditions based on keywords or problem features
- Map conditions to relevant agent selections
- Ensure 3-7 agents selected for optimal analysis

**Step 4**: Configure Synthesizer Evaluation

- Define evaluation criteria (relevance, depth, feasibility)
- Assign weights to each criterion
- Common criteria: strategic clarity, competitive insight, actionability, risk awareness

YHK

# Why LangGraph?

After evaluating AutoGen, CrewAI, and others:

- **Open Source**: Fully open with no vendor lock-in
- **Workflow Definition**: Excellent graph-based control
- **Open-Source LLMs**: Native support for local models
- **Conditional Routing**: Built-in intelligent routing
- **State Management**: Sophisticated state handling
- **Production Ready**: Deploy via LangServe
- **Community**: Large and active ecosystem

**Decision**: LangGraph for workflow control + open-source LLMs + production deployment

YHK

# The Lang* Ecosystem

- **LangChain**: Framework for LLM applications, prompt chaining
- **LangGraph**: Multi-agent workflows with conditional routing (MADS uses this)
- **LangFlow**: Visual drag-and-drop interface for prototyping
- **LangSmith**: Testing, monitoring, debugging LLM applications
- **LangServe**: Production deployment with scaling and monitoring

Complete ecosystem for building, testing, and deploying agentic systems

YHK

# Advanced Applications

# Real-World MADS Applications

- **Quality Assurance**: Test-case generators from requirements
  - Edge cases, happy paths, error conditions, security threats
- **Brainstorming & Innovation**: Generate diverse ideas
  - TRIZ agents propose innovations, mental models challenge assumptions
- **Negotiation Optimization**: Multi-agent deliberation
  - Bargaining agents model BATNA, game theory finds equilibria
- **Investment Advisory**: Personalized portfolio recommendations
  - Risk profile, asset allocation, tax optimization, behavioral analysis

YHK

# Current Limitations

**Not Mission-Critical Ready Yet**

- Non-deterministic outputs due to LLM variability
- Requires human-in-loop validation
- Best used in **Assistive/Co-pilot mode**, not full autonomy

**Technical Limitations**

- JSON parsing can fail with weaker LLMs
- Processing time: 30 seconds to 5 minutes per analysis
- Context window limits for very complex problems
- Rate limiting considerations for API-based LLMs

YHK

# Roadmap to Production

**Phase 1: Reliability** (Q1 2026)

- Structured outputs enforcement, validation layers

**Phase 2: Scale** (Q2 2026)

- Parallel agent execution, caching, streaming outputs

**Phase 3: Intelligence** (Q3 2026)

- Agent learning from feedback, meta-agent improvements

**Phase 4: Ecosystem** (Q4 2026)

- Visual configuration builder, agent marketplace, enterprise features

YHK

# Final Thoughts

# The Big Picture: Multi-Agent Future

- **Vertical AI Agents could be 10X bigger than SaaS** - Y Combinator
- Moving from conversational AI to orchestrated intelligence
- Configuration-driven development: zero code for complex systems
- Domain-agnostic framework applicable to any field
- Open-source foundation with production-ready path
- 2025: The year of AI agents becomes reality
- Multi-agent systems solve problems humans struggle with alone
- We're not looking for Success but for **Wonder**

YHK

# Key Takeaways

- Single agents are powerful; multi-agent systems are transformative
- Wisdom emerges from diverse perspectives working together
- Configuration over code enables rapid domain adaptation
- LangGraph provides production-ready orchestration
- Start simple: 3-7 agents for specific problems
- Scale complex: 25+ agents for strategic challenges
- Human-in-loop remains essential for reliability
- The future is collaborative intelligence, not singular AI

YHK

# Next Steps

**For Developers**:

- Explore LangGraph and multi-agent frameworks
- Start with pre-built configurations (Game Theory, Mental Models)
- Create custom agents for your domain expertise

**For Organizations**:

- Identify high-value strategic decision points
- Pilot MADS on non-critical problems first
- Build domain-specific agent configurations
- Establish human oversight workflows

**For Researchers**:

- Contribute novel frameworks and evaluation metrics
- Improve agent coordination and synthesis algorithms

YHK

# References


- CS 194/294-196 (LLM Agents) - Lecture 3, Chi Wang and Jerry Liu
- LLM Powered Autonomous Agents Lil'Log
- 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
- Awesome LLM-Powered Agent https://github.com/hyp1231/awesome-llm-powered-agent
- Autonomous Agents (LLMs). Updated daily https://github.com/tmgthb/Autonomous-Agents


YHK

# Thanks . . .

- Office Hours: Saturdays, 3 to 5 pm (IST); Free-Open to all; email for appointment to yogeshkulkarni at yahoo dot com
- Call + 9 1 9 8 9 0 2 5 1 4 0 6

(https://www.linkedin.com/in/yogeshkulkarni/)

(https://medium.com/@yogeshharibhaukulkarni )

(https://www.github.com/yogeshhk/ )

YHK

# Pune AI Community (PAIC)

- Two-way communication:
  - Website puneaicommunity dot org
  - Email puneaicommunity at gmail dot com
  - Call + 9 1 9 8 9 0 2 5 1 4 0 6
  - LinkedIn: https://linkedin.com/company/pune-ai-community
- One-way Announcements:
  - Twitter (X) @puneaicommunity
  - Instagram @puneaicommunity
  - WhatsApp Community: Invitation Link https://chat.whatsapp.com/LluOrhyEzuQLDr25ixZ
  - Luma Event Calendar: puneaicommunity
- Contribution Channels:
  - GitHub: Pune-AI-Community and puneaicommunity
  - Medium: pune-ai-community
  - YouTube: @puneaicommunity

Website

YHK

# Pune AI Community (PAIC) QR codes

Website

Medium Blogs

Twitter-X

LinkedIn Page

Github Repository

WhatsApp Invite

Luma Events

YouTube Videos

Instagram