Agents Unleashed: Concepts, Best Practices, and Industry Insights

@JohnAlexander,

- Content Lead, Azure AI App Development, AI Tools, Azure MCP Server - Microsoft

- AI Tutor - University of Oxford

KANSAS CITY DEVELOPER CONFERENCE
KCDC
2025

@JohnAlexander

Content Lead, Azure AI App Development, Microsoft
https://aka.ms/azai
Tutor, University of Oxford
7 years of AI/ML
25 years of Software Development, Architecture,
3 years of Generative AI

There is a resource guide for all links used in this session (and more for further exploration) at JRAlexander/kcdc-2025

Resource Guide

Agents

- Agents
- Building effective agents \ Anthropic
- AI Agents from First Principles
- Tips for building AI agents
- How We Build Effective Agents: Barry Zhang, Anthropic
- Andrej Karpathy: Software Is Changing (Again)
- [Session] MCP vs ACP vs A2A: Comparing Agent Protocols with Laurie Voss from LlamaIndex
- Four AI Agent Strategies That Improve GPT-4 and GPT-3.5 Performance
- microsoft/generative-ai-for-beginners: 21 Lessons, Get Started Building with Generative AI https://microsoft.github.io/generative-ai-for-beginners/
- microsoft/ai-agents-for-beginners: 11 Lessons to Get Started Building AI Agents
- VS Code: Open-Source AI Editor (**GitHub Copilot Chat open source**)
- [2505.22954] Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents
- How I program with Agents | crawshaw - 2025-06-08
- Choosing the right AI model for your task - GitHub Docs
- How to deploy AI safely | Microsoft Security Blog
- AI Strategy: A Practical Framework Using Jobs-to-be-Done (JTBD) | by Mike Boysen | Apr, 2025 | Medium

MultiAgents

- Lessons from Toyota for building durable multi-agent copilots
- Conceptual Guide: Multi Agent Architectures
- A2A Protocol - Agent-to-Agent Communication
- [1706.02275] Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments

Prompting

- State-Of-The-Art Prompting For AI Agents
- The Prompt Engineering Playbook for Programmers
- Enhance your prompts with meta prompting

Agents Unleashed Resource Guide – August 15th, 2025

- AI-assisted coding for teams that can't get away with vibes - nilenso blog
- AI-assisted coding (simon willison)
- GPT-4.1 Prompting Guide
- The rise of "context engineering"
- Claude 4 System Card
- A Systematic Survey of Prompt Engineering in Large Language Models:
- Techniques and Applications

Tool Calling

- How to call functions with chat models
- Introducing the Model Context Protocol \ Anthropic
- Introduction - Model Context Protocol
- Authorization - Model Context Protocol
- The New MCP Authorization Specification - Den Delimarsky
- Please Don't Write Your Own MCP Authorization Code - Den Delimarsky
- Announcing Model Context Protocol Support (preview) in Azure AI Foundry Agent Service | Azure AI Foundry Blog
- Build Agents using Model Context Protocol on Azure | Microsoft Learn
- microsoft/mcp-for-beginners: This open-source curriculum is designed to teach the concepts and fundamentals of the Model Context Protocol (MCP), with practical examples in .NET, Java, TypeScript, JavaScript and Python.
- Open Protocols for Agent Interoperability Part 1: Inter-Agent Communication on MCP | AWS Open Source Blog
- Open Protocols for Agent Interoperability Part 2: Authentication on MCP | AWS Open Source Blog
- Building Agents with Model Context Protocol - Full Workshop with Mahesh Murag of Anthropic
- How to call functions with chat models
- Build AI agent tools using remote MCP with Azure Functions | Microsoft Community Hub
- modelcontextprotocol/inspector: Visual testing tool for MCP servers
- NLWeb Pioneer Profiles: Customer Success Stories & Use Cases
- Agentic Coding Recommendations | Armin Ronacher's Thoughts and Writings

John Alexander - https://github.com/JRAlexander/kcdc-2025

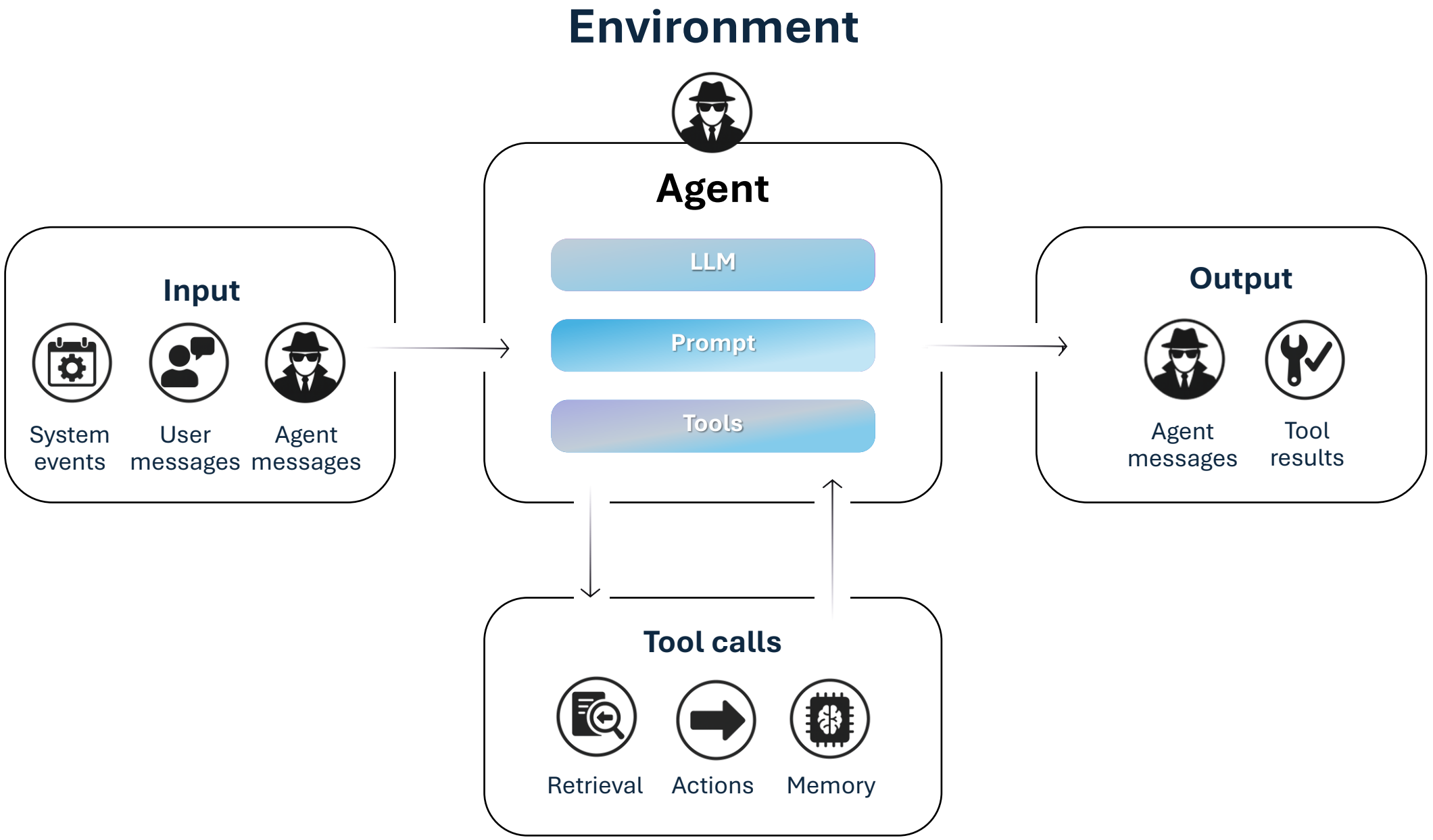
Learning objectives

- Understand the fundamental concepts and terminology related to AI agents.
- Learn essential design principles for creating effective AI agents.
- Gain insights into selected industry reports to understand the current state.

Part I: Level Set

Agent	A software entity that performs tasks autonomously, often using AI to make decisions and interact with the environment.
Workflow	A system where LLMs and tools are orchestrated through predefined code paths.
Metaprompting	A technique using an LLM to generate or improve prompts
Context engineering	Providing all the context for the task to be plausibly solvable by the LLM
Model Context Protocol	An open protocol for connecting AI apps to tools, APIs, and data sources.
Agent 2 Agent Protocol	An open standard for AI agent communication and collaboration across different platforms and frameworks, regardless of their underlying technologies.
Deterministic system	A system that always produces the same output for a given input.
Probabilistic system	A system that produces a probability distribution over possible outcomes and will likely have different outcomes given the input.
Multi-Agent	A system of multiple autonomous agents that interact or work together to achieve individual or shared goals.

Terms


Environment
Agent
LLM
Prompt
Tools
Input
System events
User messages
Agent messages
Output
Agent messages
Tool results
Tool calls
Retrieval
Actions
Memory

Agent Tool calling – Function calling

What is it?

- **Function calling** allows large language models (LLMs) to **invoke external functions or APIs** based on natural language input.

Why Does It Matter?

- Function calling transforms LLMs from **static text generators** into **dynamic, tool-using agents**.

Tool calling –
Model Context Protocol (MCP)

An open protocol for connecting AI apps (clients) to tools, APIs, and data sources (servers).

Why It Matters:

- Standardizes tool/data access
- Creates a single interface all apps and tools can use
- Inspired by APIs and LSP (Language Server Protocol)

MCP Architecture

- MCP Host: The environment where the agent logic lives (your own app, Azure OpenAI, Copilot Studio)
  - Agent: The LLM-based reasoning unit (e.g., Azure OpenAI GPT-4 model)
  - MCP Client: Communication layer that speaks the Model Context Protocol
- MCP Server: Provides access to tools, APIs, and data (e.g., Azure CLI, resource graphs)
- Data Sources:
  - Local
  - Remote

Flow: User → Agent → MCP client ↔ MCP Server → Services

Three MCP Interfaces

Provided by server, consumed by client:

- **Tools** – model-controlled - Functions that can be called by the LLM (with user approval)
- **Resources** – application-controlled - File-like data read by clients (like API responses or file contents)
- **Prompts** – user-controlled - Pre-written templates that help users accomplish specific tasks

Tool calling – MCP vs Function call

Use MCP when:

- You want reusable tool endpoints across agents
- You need **fine-grained control**, **auditing**, or **identity-aware access**
- You are building for **enterprise or secure Azure scenarios**

Use function calling when:

- You are prototyping or need **tight integration** within a single model call
- Your tools are **lightweight** and don’t need centralized governance

Multi-Agents

A system of multiple autonomous agents that interact or work together to achieve individual or shared goals.

Multi-agent systems are used in fields like artificial intelligence, distributed computing, economics, and robotics to model and solve complex problems.

Key Characteristics of Multi-Agent Systems

Autonomy:

- Each agent operates without direct human intervention, controlling its own actions based on its goals.

Decentralization:

- No single agent has complete control over the entire system.
- Agents work independently or collaboratively, contributing to the overall system behavior.

Interaction:

- Agents communicate and cooperate, compete, or negotiate with each other to achieve their goals.

Distributed Problem Solving:

- The system can solve problems that are too complex for an individual agent by dividing tasks among multiple agents.

Adaptation:

- Agents can learn and adapt their behavior over time based on experiences or environmental changes.

Agent 2 Agent (A2A) protocol

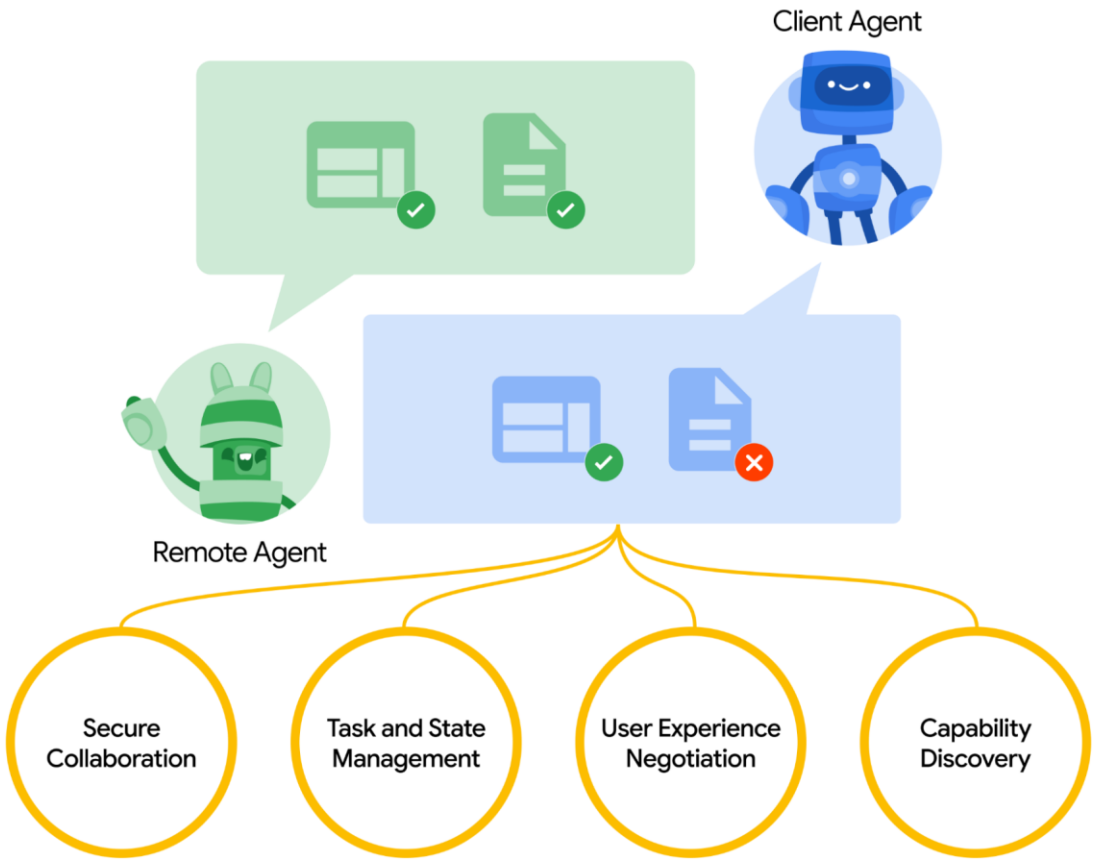
A2A facilitates communication between a 'client' agent and a 'remote' agent through a structured process

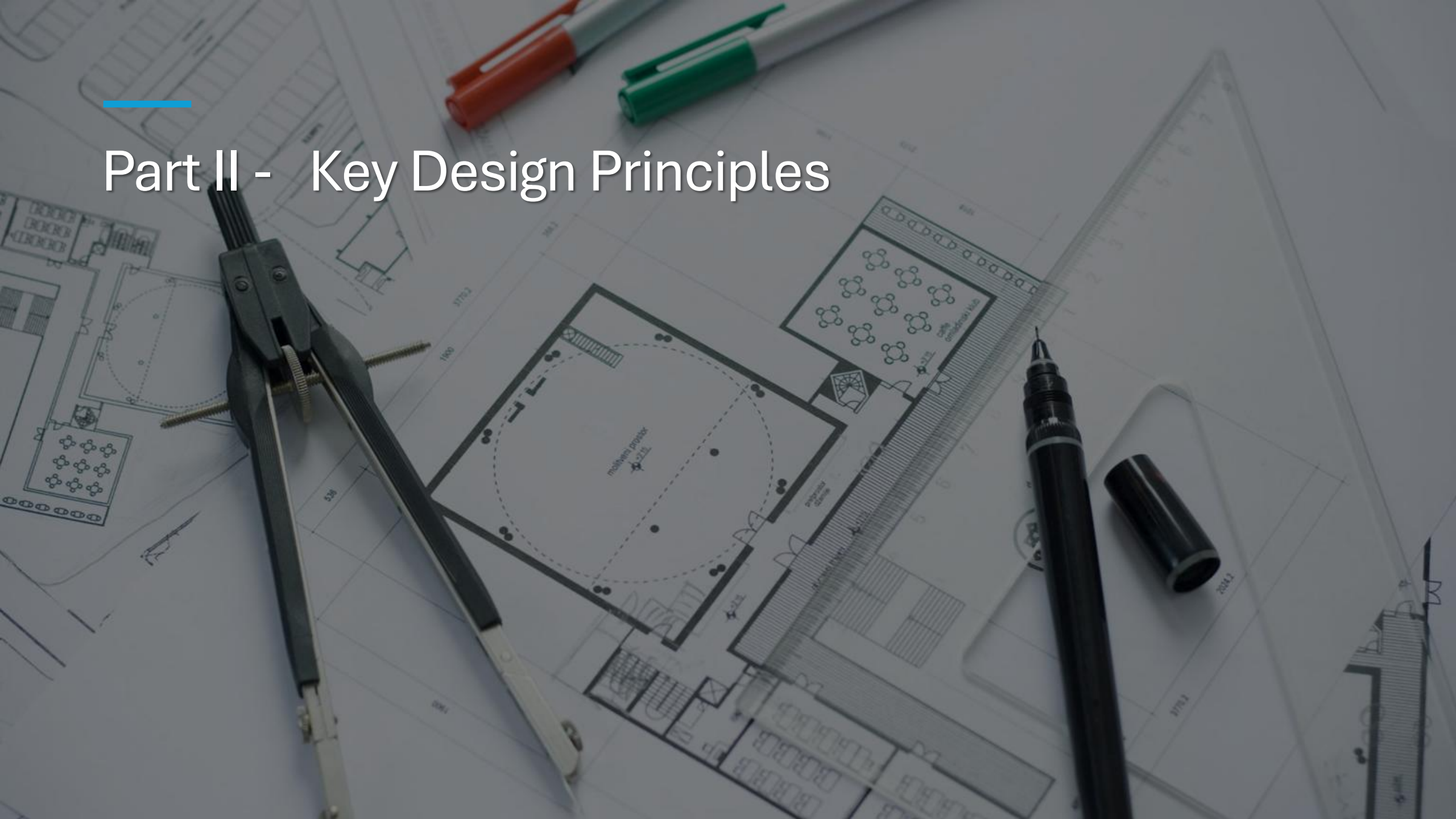
Capability Discovery
Agents advertise their capabilities using an 'Agent Card' in JSON format, enabling other agents to identify the best agent for a task.

Task Management
Communication is oriented towards task completion, with a defined lifecycle that can be completed immediately or over time.

Collaboration
Agents can send messages to communicate context, replies, artifacts, or user instructions.

User Experience
Messages include 'parts' with specified content types, allowing agents to negotiate the correct format and UI capabilities.





Part II - Key Design Principles

Don’t build agents for everything

Use agents only when the task *requires* exploration and autonomy.

- Is the task ambiguous?
- Does it justify high token cost?
- Can the agent act effectively?
- Can I detect/fix errors quickly?

How do I know if my task is worth building an agent for?

Use this checklist:

- Is the task ambiguous?
- Does it justify high token cost?
- Can the agent act effectively?
- Can I detect/fix errors quickly?

Iterate reality!

01

Sit with real users (e.g., in logistics or support centers).

02

Learn their actual workflows / processes.

03

Encode their logic into prompts and evals.

04

Iterate quickly with them in the loop. - FEEDBACK

Designing AI Agents with the Jobs To Be Done Framework – a unique approach

- **JTBD Core Principle**
  - "People don’t buy products; they hire them to get a job done.“
- **Applied to AI:**
  - "Users don’t engage with agents for AI’s sake; they want specific jobs completed."

Designing AI Agents with the Jobs To Be Done Framework

JTBD Step	AI Agent Design Equivalent
Define the core job	What's the agent hired to do?
Functional/emotional/social dimensions	What user outcomes must be met?
Desired outcomes	What defines agent success?
Struggling moments	Where do users get stuck today?
Capabilities design	Build features to resolve struggles
Hire/fire criteria	Why users continue or stop using the agent?

JTBD Checklist for AI Agents

- ✅ Have we defined the job the user is hiring the agent for?
- ✅ Do we understand emotional and functional needs?
- ✅ Are agent outputs tied to real progress?
- ✅ Does the agent reduce or eliminate user struggles?
- ✅ Can we measure success as job completion?

Start with the three core agent components

Environment — What the agent acts upon.

Tools — Interfaces for actions + feedback.

Prompt — Goals, constraints, behavioral guardrails.

Longer prompts (even 6+ pages) are okay if structured cleanly:

- Role definition
- Task plan
- Tool usage
- Output format
- Reasoning scaffolding
- Worked examples

Treat Prompts Like Code (Prompt as Spec)

Structure prompts using roles, tasks, constraints, and formats—just like defining a software interface.

Use clear markdown or XML-like formatting for parsing and stability.

Prompt should act as an API contract between your agent and your system.

Test Prompts and Tools with the Model Itself

Ask the model:	Is this prompt clear?
	Is the tool easy to use?
	Are parameters missing?

Example: keeping agents on the leash

Here's an example. This prompt is not unreasonable but not particularly thoughtful:

```
Write a Python rate limiter that limits users to 10 requests per minute.
```

I would expect this prompt to give okay results, but also miss some edge cases, good practices and quality standards. This is how you might see someone at nilenso prompt an AI for the same task:

```
Implement a token bucket rate limiter in Python with the following requirements:

- 10 requests per minute per user (identified by `user_id` string)
- Thread-safe for concurrent access
- Automatic cleanup of expired entries
- Return tuple of (allowed: bool, retry_after_seconds: int)

Consider:
- Should tokens refill gradually or all at once?
- What happens when the system clock changes?
- How to prevent memory leaks from inactive users?

Prefer simple, readable implementation over premature optimization. Use stdlib only (no Redis/external deps).
```

Source: Andrej Karpathy: Software Is Changing (Again) AI-assisted coding for teams that can't get away with vibes - nilenso blog

Think Like the Agent

Constrain yourself to its **context window**:

- Same token limits
- No memory beyond current state
- No visual continuity

Practical exercise:

Close your eyes and imagine you are the LLM. Try completing a task with only a screenshot + brief description. This will reveal missing context in your agent’s design. What’s missing?

This helps with context engineering!!!

Evaluation & Testing

- Start simple, with clear success metrics.
- Iterate and validate agent effectiveness early and often.
  - Run evals across tool versions
  - Test with fallback scenarios
- Compare agent output pre/post update

MCP Server – Getting Started

- Start with tools, then add prompts/resources
- Separate logic: model vs app vs user
- Favor dynamic interpolation
- Push business logic to server

MCP Server - Treat Tools Like Prompts

Write **descriptive tool documentation**.

Remember: Tool specs are part of the prompt.

Poor tool design = poor agent performance.

MCP Auth & Security

Use OAuth 2.0 (natively supported)

Let server manage access tokens

Trust but verify: vet external servers

This is rapidly changing – follow the MCP Spec

MCP Debugging & Observability

1

Use **Inspector** for logs and tool tracing

2

Return metadata from tools

3

Document capabilities clearly

4

Just like an API, a good MCP server includes clear tool names, capabilities, and optional annotations.

Mitigate Risk in Early Deployments

Use read-only or human-in-the-loop phases.

Scope tasks narrowly at first.

AI Agent Development Best Practices

✅ **Understand the Core Difference: Agent vs. Workflow**
Know when to use autonomous, iterative agents vs. fixed, linear workflows.

✅ **Think Like the Agent**
Simulate the model's limited context and reasoning ability when designing prompts and environments.

✅ **Treat Tools Like Prompts**
Provide clear, documented, and well-named tools—these are part of the prompt context too.

✅ **Measure Everything**
Build with evaluation in mind—test whether your agent actually improves outcomes.

✅ **Pick the Right Problems**
Use agents where tasks are complex and valuable, but the cost of error is low (e.g., search, coding).

✅ **Design for the Future**
Structure your systems so they improve as models improve, not break under smarter behavior.

Agent Insights Summary

Insight	Takeaway
Agents ≠ Workflows	Use agents for exploration, workflows for control.
Scope = Trust	Start with safe use cases; expand once validated.
Simplicity Wins	Start with a minimal architecture and optimize later.
Tool UX Matters	Agents benefit from intuitive, constrained tools.
Verifiability is Key	Build agents in domains where you can test outcomes.
Context is Everything	Understand agent failures by reproducing their context view.
Use the Model to Debug Itself	Agents can audit prompts and decisions when prompted.
Does it work in Multi-Agent use cases?	Plan for asynchronous agent-to-agent protocols.
Meta-Tools Enable Scale	Let agents help evolve their own infrastructure.

What essential skills should agent developers master?

Workflow Breakdown: Clearly defining business processes into manageable tasks.

Data Integration & Plumbing: Using frameworks and tools like MCP for smooth data integration.

Evaluation Frameworks: Quickly setting up systematic evals to trace and manage improvements efficiently.

Rapid Decision-Making: Developing instincts to identify which agent components to refine or abandon quickly.

Designing and managing context: Designing information flows to support task completion.

Key Design Principles for Multi-Agent vs Single-Agent Systems

Fundamental differences in designing single-agent and multi-agent systems.

Multi-agent systems require a shift from individual optimization to managing interactions.

Key principles include communication, coordination, and distributed decision-making

Addressing these principles leads to efficient, reliable systems capable of complex tasks through collective intelligence.

Communication and Interaction

Single-Agent Systems

- Interacts primarily with the environment.
- No need for inter-agent communication.

Multi-Agent Systems

- Agents must communicate with each other.
- Share information, negotiate, and coordinate actions.

Design Principle

- Establish robust communication protocols
- Ensure reliability, scalability, and security in communication

Coordination and Cooperation

Single-Agent Systems

- Focus on internal coordination for optimizing actions.

Multi-Agent Systems

- Agents need to coordinate actions to achieve shared goals or avoid conflicts.

Design Principle

- Implement coordination mechanisms (centralized planning, distributed consensus).
- Define clear protocols for cooperation and conflict resolution.

Distributed Decision-Making

Single-Agent Systems

- Centralized decision-making within the agent.

Multi-Agent Systems

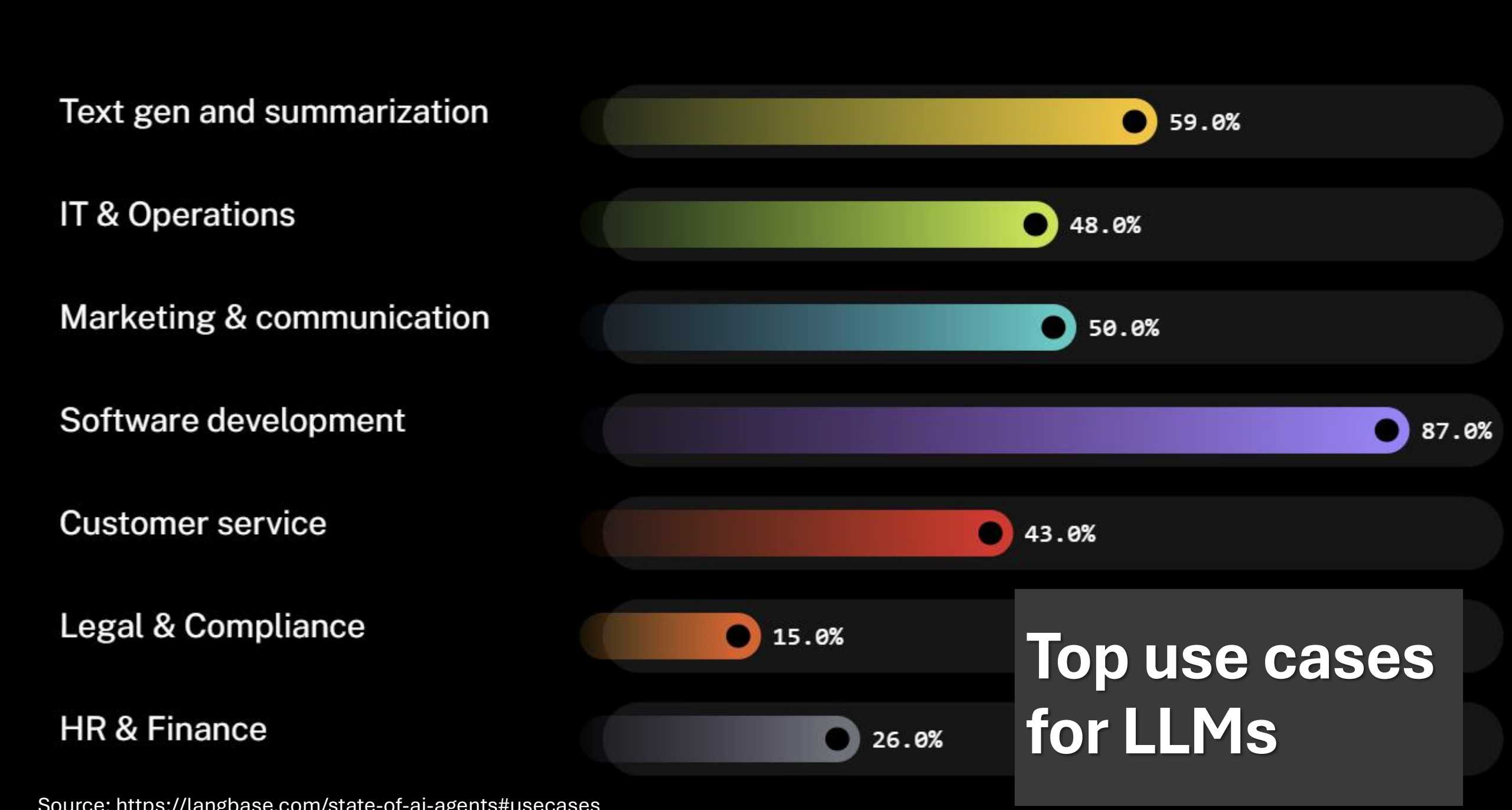
- Decentralized decision-making; each agent is autonomous.

Design Principle

- Design agents to make decisions based on local information.
- Use distributed algorithms for consistency and global objectives

Part III - Selected conclusions from industry reports

In your opinion, which tasks are agents best suited to perform today?



Top use cases for LLMs

Source: https://langbase.com/state-of-ai-agents#usecases

Industry Excerpts:
KPMG AI Quarterly Pulse Survey

- Investor pressure to demonstrate **ROI on GenAI** investment is important or very important for 68% of leaders.
- **Productivity** is now the top ROI metric (79%) for the first time since Q1 2024. Profitability is a close second and increased more than any other metric from Q1 to Q4, jumping from 35% to 73%.
- The **quality of organizational data** is the **biggest anticipated challenge** to AI strategies in 2025 (85%), followed by data privacy and cybersecurity (71%), and employee adoption (46%).

Questions?

All materials at
JRAlexander/kcdc-2025

Thanks!

@johnalexander

https://www.linkedin.com/in/thejohnalexander/

Resources

- Andrew Ng: State of AI Agents | LangChain Interrupt
- Tips for building AI agents
- Building Agents with Model Context Protocol - Full Workshop with Mahesh Murag of Anthropic
- AI prompt engineering: A deep dive
- State-Of-The-Art Prompting For AI Agents
- Choosing the right AI model for your task - GitHub Docs
- https://github.com/JRAlexander/kcdc-2025