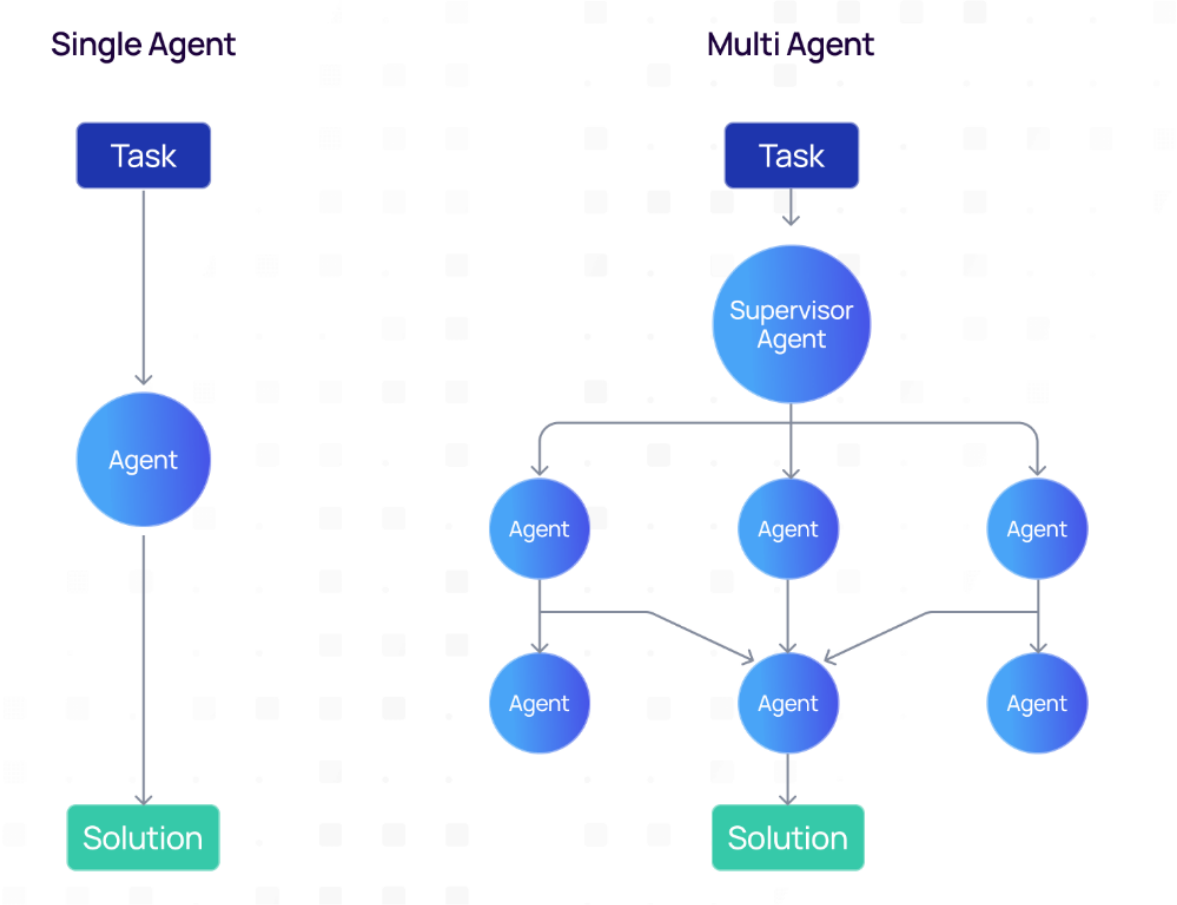
Single-Agent vs Multi-Agent Systems

Table of Contents

Introduction

Agent-based artificial intelligence (AI) systems can be designed as single-agent systems (SAS) or multi-agent systems (MAS). A single-agent system employs one central AI agent to perform tasks, often invoking various tools or functions under a unified reasoning process. In contrast, a multi-agent system consists of multiple autonomous agents that cooperate or compete to accomplish tasks. Multi-agent AI systems allow large or complex problems to be decomposed into subtasks handled by specialized agents. As the scope of AI applications expands, understanding the trade-offs between SAS and MAS architectures is critical for designing effective systems.

Figure 1 below illustrates a high-level comparison of single agent versus multi-agent workflows.



For example, a single-agent AI might sequentially call different tools to complete a task, whereas in a multi-agent setup the task is distributed

among parallel AI specialists. This document provides a thorough comparison of single-agent and multi-agent paradigms defining each, detailing architecture and communication, and summarizing key similarities and differences – and then reviews real-world use cases in healthcare, mobility, and customer service. Each section includes examples, diagrams, and citations from recent research or industry to ensure depth and clarity.

Single-Agent Systems

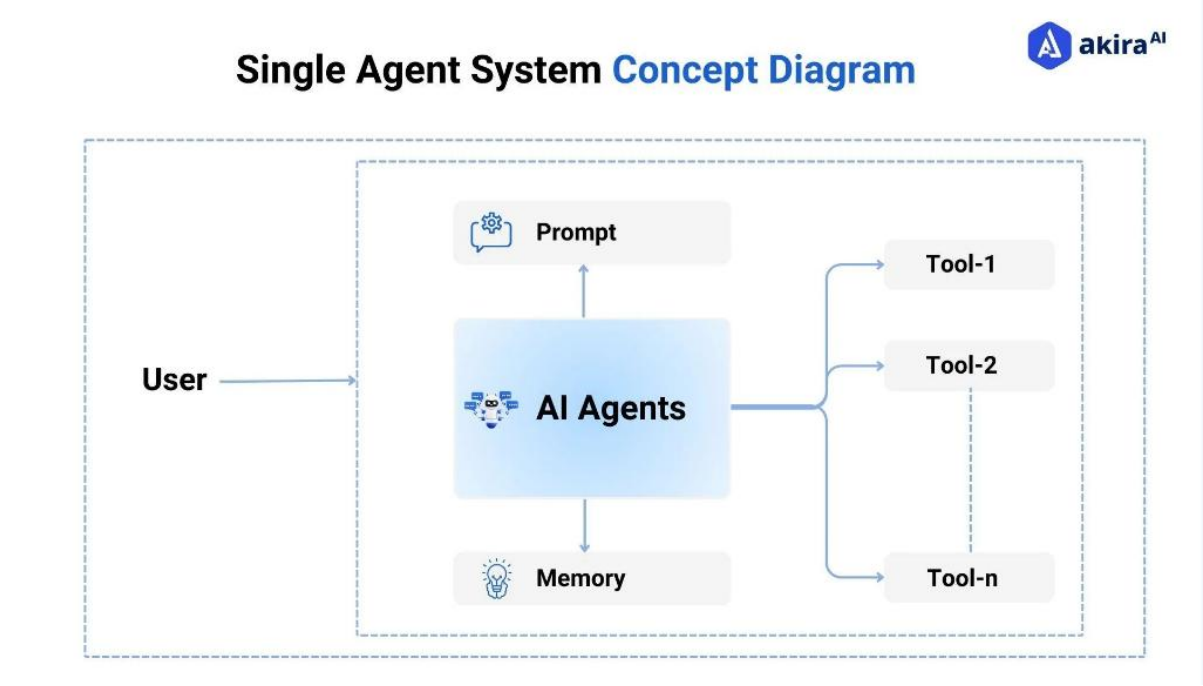
Definition

A single-agent system (SAS) is one in which a lone intelligent agent orchestrates all decision-making and actions for a given task. The agent may invoke multiple tools or API calls, but *itself* remains the sole "brain" of the system. For instance, a single agent chatbot might integrate image recognition, web search, and translation, but these are all under control of one core agent. SAS are centralized: one reasoning thread handles everything. The advantage is a unified context and simpler architecture. Examples of modern single-agent systems include conversational assistants like ChatGPT's "o3" agent, which internally decides when to call web browsing, code execution, or vision tools, all within one reasoning process. In essence, a single-agent system is *monolithic* in structure – one AI handles perception, reasoning, and action.

Architecture and Design

Architecturally, a single-agent system typically follows a straightforward pipeline: input (e.g. user query) → the single agent's reasoning (often a large language model or neural network) → output or action. Design principles include maintaining a unified knowledge base or context window, and planning or execution all within one agent loop. The system may have modular **tools** (such as a database query tool or code interpreter), but these are not independent agents; they are called by the agent when needed. As described in alliance, SAS design emphasizes simplicity and efficiency: "no need for complex communication protocols, as all context

resides in one agent that maintains one coherent perspective". This simplicity makes the system easier to debug: there is one reasoning chain to trace. Because there is only one agent, the architecture avoids overhead of inter-agent messaging.



Communication and Coordination

Single-agent systems require no peer-to-peer communication protocols, since by definition there are no other agents. The sole agent handles all coordination internally. In effect, coordination is trivial: the agent simply schedules its own subtasks. This contrasts sharply with MAS, which must define languages or protocols for agents to exchange messages. For an SAS, the main "communication" concerns are with external tools and databases (APIs, search engines, knowledge graphs). But these do not count as agent-to-agent communication. Since all decisions are centralized, there is no issue of synchronizing state across agents. In summary, SAS architectures eliminate the complexities of inter-agent messaging and mutual goal alignment, at the cost of having one agent do all the work.

Autonomy and Control

In a single-agent system, the one agent is fully autonomous within the system's domain. It possesses a global view of the problem (within its

context window) and makes all strategic decisions. That agent can utilize tools autonomously, but human operators or external triggers are not needed for its decision loops (though operators may still provide high-level goals). Autonomy here means the agent acts without coordinating with "peers." As notes, an SAS agent is the "center of decision making". Because all autonomy is vested in one entity, there is no division of labor. This can simplify reasoning: the agent doesn't need to guess what another agent might do. However, it also means that any blind spots or errors in that single agent will affect the entire system. Autonomy in SAS is straightforward, but *monolithic* – every component in the system depends on that one agent's correctness and capacity.

Use Cases

Single-agent systems excel in scenarios where tasks are moderately complex and primarily sequential and where having one consistent world model is beneficial. Common use cases include:

- **Chatbots and Virtual Assistants:** A single conversational agent handles user requests end-to-end, using internal tools as needed (e.g. GPT-based assistants that generate responses, execute queries, etc.). About 37% of small businesses now use chatbots for routine customer queries.
- **Automation Scripts:** A single AI agent that automates workflows (e.g. booking flights, scheduling appointments) by invoking various APIs one after another.
- **Single-Robot Control:** A lone service robot in a home or factory, controlled by one onboard agent, taking sensor inputs and acting (e.g. a vacuum robot).
- **Interactive Game AI:** Non-player characters driven by one decision-making agent (for simple games).

In each case, the single agent can often handle all required tasks more simply than a coordinated swarm. As GrowthJockey notes, SAS are "efficient for defined tasks" and cost-effective since only one model runs. They are a natural fit when the problem is limited in scope and complexity.

Limitations

While simple, single-agent systems have important limitations. Chief among them is scalability: there is a limit to what one agent can handle. A single context window (for an LLM) constrains the amount of information the agent can consider. Tasks that require diverse expertise or long-term, parallel processes may overwhelm one agent. Single agents are also a single point of failure: if the agent makes a critical mistake or crashes, the whole system stops. Further, SAS can struggle in real-time or highly dynamic environments, because they cannot parallelize. points out that when tasks become long-running or involve multiple domains, single agents may fall short due to “capability constraints” and “processing limitations”. In summary, SAS limitations include limited robustness (no redundancy) and difficulty handling very large, distributed tasks.

Multi-Agent Systems

Definition

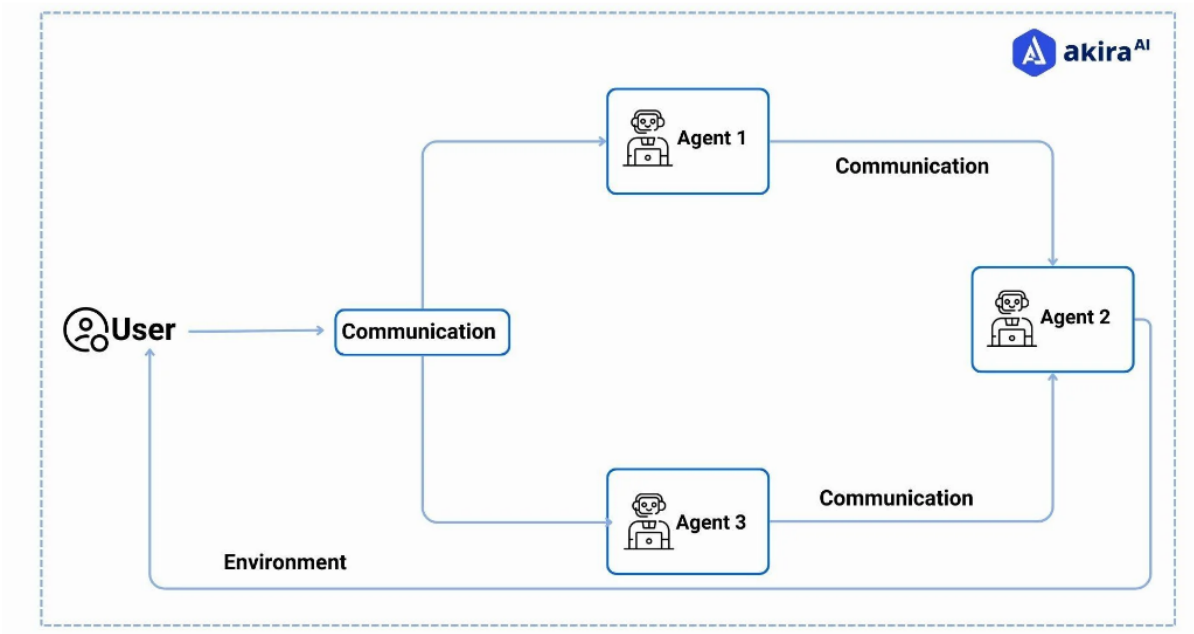
A multi-agent system (MAS) comprises two or more autonomous agents that interact in a shared environment to achieve individual or common goals. Each agent has its own local perspective or specialty, but they work as a team to solve problems. Formally, an MAS “involves several AI agents working together to perform tasks,” where agents may share information and collaborate towards shared objectives. Multi-agent systems are inherently distributed and collaborative. Examples include fleets of autonomous drones coordinating to map an area, or a set of customer-service bots each handling different aspects of a conversation. The MAS approach is particularly useful for *large-scale or complex tasks* that exceed a single agent’s capacity. Because MAS involves multiple decision-makers, it opens up possibilities for specialization and parallelism that single-agent systems cannot match.

Architecture and Design

Multi-agent architectures come in various designs, but typically include multiple autonomous components plus mechanisms for their interaction. Design patterns include centralized coordinators (a master agent

delegating subtasks), fully decentralized networks (peer-to-peer collaboration), or hierarchical arrangements (teams with leader agents). Regardless of pattern, a MAS architecture defines how agents are instantiated, how they sense environment and goals, and how they share information. Key design principles are *modularity* and *specialization*: each agent can be optimized for a particular subtask (e.g. one agent for planning routes, another for diagnosing images). This modularity allows the system to grow by adding new agents for new functions.

To orchestrate cooperation, MAS often include middleware or orchestration layers. For example, Google's new "A2A" (Agent-to-Agent) communication protocol provides standardized messaging and collaboration rules between agents. In a MAS, agents might also use a shared blackboard or memory to post updates. The architecture must balance autonomy with coordination: agents must have their own decision logic, yet agree on common goals and protocols. emphasizes that in MAS "agents must communicate and coordinate their actions," often via established protocols. In practice, MAS architectures can be complex; they need to handle agent discovery, dynamic team formation, task allocation, and conflict resolution. However, the trade-off is flexibility. As Alliance notes, MAS architectures are *flexible* and modular, enabling emergent solutions from agent interactions.



Communication and Coordination

Communication is central in multi-agent systems. Agents exchange messages to share knowledge, request actions, or negotiate. Well-known communication languages and protocols (e.g. FIPA-ACL, KQML, or more recent standards like Google's A2A) are used to structure these exchanges, ensuring interoperability. The choice of communication mechanism depends on the MAS style: a centralized coordinator might communicate via direct calls (one agent tells another exactly what to do), whereas decentralized agents might broadcast intentions or vote on actions.

Coordination strategies in MAS range from collaborative planning to market-based task allocation. For instance, agents might collectively schedule subtasks so that dependencies are respected, or use auctions where agents bid on tasks they are best suited for. In collaborative settings, agents can check each other's work and adjust plans if needed. The V7labs analysis highlights that MAS architectures include an orchestration layer to manage workflow and defined protocols for information exchange. For example, one agent might extract data from a document, then pass it to another agent for analysis, coordinated by a master control. Coordination also handles synchronization: ensuring that agents have a consistent view of shared data. Agents may use a shared database or messaging queue to stay in sync. Thus, multi-agent communication and coordination add considerable overhead – but they are what enable MAS to function as a coherent system. Without communication, the agents would effectively be isolated, defeating the purpose of MAS.

Autonomy and Collective Behavior

Each agent in a MAS is itself an autonomous entity: it can perceive its environment, make decisions, and act on its own. However, unlike SAS, an agent's autonomy is local; it operates with partial information and limited scope. The system's overall behavior emerges from the interactions of these autonomous agents. For example, in a fleet of autonomous cars, each car-agent controls its own speed and route, but all cars together optimize traffic flow. Agents often have individual goals in addition to any group goal. In cooperative MAS, agents share common goals and thus

coordinate to achieve them; in competitive MAS, agents may have different or opposing goals. The MAS must be designed so that agents' autonomy is guided by incentives or protocols that lead to the desired global outcome.

This balance of autonomy and cooperation is a defining feature of MAS. notes that specialization and *distributed* decision-making allow MAS to solve problems that single agents cannot. For instance, one agent might autonomously diagnose an image, while another autonomously generates a patient report, both working towards the goal of patient care. The collective intelligence arises when agents check, refine, or complement each other's work. In essence, MAS harness the autonomy of many agents to create a higher-level, emergent intelligence that can adapt to complex tasks.

Use Cases

Multi-agent systems are suited for complex, long-running, or highly parallel tasks. Some typical use cases include:

- **Robotic Teams:** Multiple robots collaboratively perform tasks (e.g. warehouse robots cooperatively retrieving items, or search-and-rescue drones coordinating to survey an area).
- **Distributed Sensor Networks:** Agents on sensors share data to monitor large environments (e.g. environmental monitoring, smart home systems with many IoT devices).
- **Automated Vehicles:** Swarms of autonomous cars or drones coordinating movement, such as platooning vehicles on a highway or drones forming ad-hoc networks.
- **Gaming and Simulation:** Non-player characters or economic agents in simulations that interact, compete, and adapt (e.g. multi-agent reinforcement learning in game environments).
- **Enterprise Systems:** Multiple software agents handle different functions: one for customer queries, another for inventory, another for ordering – all integrating into a unified workflow.

- **Complex Problem Solving:** Breaking down a large problem (like planning a space mission) into sub-problems handled by specialized agent teams.

In industry, MAS can tackle tasks that require combining expertise from different domains. For example, a multi-agent approach might integrate natural language understanding, image analysis, and predictive analytics through different agents, allowing parallel processing. highlights the example of Manus AI, which uses specialized agents for business functions, and Devin (an "AI software engineer" composed of planning, coding, testing, and debugging agents). These applications demonstrate how MAS can achieve "adaptive collaboration" and resilience by letting agents specialize and check each other's work.

Limitations

Multi-agent systems introduce significant complexity. Each added agent means more potential communication paths and more things that can fail. The system must manage agent coordination overhead and keep goals aligned. warns that MAS require "continuous resource management and goal alignment," as well as careful debugging of many interacting components. MAS generally need more computation and development effort than SAS. They must address issues such as latency in communication, consistency of shared data, and fault isolation when one agent fails. There is also the challenge of emergent behavior: unpredictable interactions between agents can produce unintended consequences. Because of these challenges, MAS are often chosen only when their benefits outweigh the costs. According to Alliance, for about 80% of use cases a single-agent system suffices, with multi-agent reserved for very complex, long-running, or highly scalable scenarios.

Despite these drawbacks, MAS can be far more powerful for suitable applications. The strengths of MAS scalability through added agents, fault tolerance via redundancy, and specialized parallel processing can justify the overhead in mission-critical or large-scale systems. The key is careful system design to handle communication and coordination efficiently, which leads to the detailed comparisons below.

Key Differences and Similarities

While SAS and MAS both involve intelligent “agents,” their architectures differ markedly. For quick reference, single-agent systems have one centralized decision-maker, whereas multi-agent systems distribute decision-making among multiple agents.

- **Architecture:** *SAS* – one agent owns the entire task; *MAS* – multiple agents each handle parts of the task, possibly with a coordinator or in peer-to-peer fashion.
- **Decision-Making:** In *SAS*, one agent generates a complete plan. In *MAS*, agents may negotiate or jointly decide on actions.
- **Scalability:** *SAS* is limited by the agent’s capabilities. *MAS* can scale by adding more agents to cover more tasks or data.
- **Fault Tolerance:** In *SAS*, if the sole agent fails, the system stops. In *MAS*, other agents can compensate if one fails.

- **Complexity:** *SAS* is simpler to implement and maintain. *MAS* requires more complex infrastructure (communication protocols, synchronization).

Despite these differences, SAS and MAS share the goal of autonomous problem-solving with AI. Both rely on AI models and potentially similar technologies (e.g. LLMs, neural networks, knowledge bases). Both require careful design of state representation and action mechanisms. In practice, the two approaches can sometimes blend: a system might use a main agent that spawns helper agents as needed (a hybrid). But in pure SAS, only one entity reasons, whereas MAS leverages multiple minds. Choice between SAS and MAS depends on factors like task complexity, required parallelism, and system constraints.

Real-World Applications

Healthcare

Healthcare systems have begun leveraging both single-agent and multi-agent AI to improve care delivery. For example, a single-agent diagnostic chatbot (based on an LLM) might assist doctors by answering patient queries or suggesting likely diagnoses from symptoms. In contrast, multi-agent AI is emerging to tackle broader healthcare tasks. MAS can coordinate complex workflows such as patient triage, diagnostics, and treatment planning by having specialized agents for each function.

Recent reviews note that MAS are being explored in clinical decision support systems, robotic surgery, and critical care monitoring. MAS allows sharing information among agents (e.g. an agent analyzing lab results, another monitoring imaging, another handling scheduling), which can lead to better decision-making and scale. mentions that MAS enable “better decision-making, improved scale, and smarter responses” based on situational awareness. However, healthcare MAS are still largely experimental; there are concerns about reliability, safety, and ethics. The

systematic review by Bizarro and Mustafa (2025) highlights that while MAS can improve accuracy and reduce workload (for example, by assisting surgical teams or continuously monitoring ICU patients), they also raise issues around patient safety, data privacy, and the need for human oversight.

Diagnostic Assistants

In medical diagnostics, both SAS and MAS approaches have appeared. A single-agent diagnostic assistant might use one large model that ingests patient data, medical history, and images, then outputs a possible diagnosis or suggestion. Google's Med-PaLM or IBM's Watson used to operate similarly on single vast models. In a multi-agent approach, different diagnostic tasks (e.g., interpreting blood tests, analyzing radiology images, evaluating symptoms) could be handled by separate AI agents, each expert in its domain. These agents would then share findings via an orchestration layer to produce a unified assessment. This can mimic a medical team's workflow. For instance, one MAS prototype might have one agent specialized in cardiology, another in neurology, etc., collaborating to diagnose complex cases. The potential benefits are parallel processing and specialization, but integration is challenging.

Robotic Surgery Coordination

Robotic surgery provides a clear illustration of multi-agent coordination. Modern operating rooms often involve multiple robotic devices: surgical robots manipulating instruments, robotic arms controlling cameras, and monitoring systems. These constitute multiple "agents" that must work synchronously under a surgeon's direction. Even if the human surgeon is considered one agent, the system acts like a multi-agent team: the da Vinci surgical system, for example, has separate controllers for each instrument. Research platforms (e.g., Raven-II, Raven's open-source surgical robot) model these as MAS for studying human–robot collaboration. In an MAS design, one agent could manage force-feedback and haptics, another could ensure safety limits, while a third maintains a map of the surgical site. Coordination protocols ensure none of the robotic arms collide and that they execute complex maneuvers safely. While each robotic

component is not intelligent in the AI sense, advanced research is developing autonomous surgical assistants that would indeed act as agents (e.g., an agent controlling suturing while another follows the surgeon's commands), illustrating how MAS principles apply to healthcare robotics.

Case Study: Multi-Agent Healthcare App

A concrete example is provided by a 2025 industry case study of a multi-AI healthcare app (Nirmitee Inc.). In this case, a startup built a patient-care platform using *multiple AI agents* to provide personalized recommendations and diagnostics. Key challenges emerged: *inter-agent communication*, *data synchronization*, *real-time processing*, and *security* For instance, each agent specialized in a domain (e.g. symptoms analysis, medication scheduling, patient Q&A) needed a way to communicate results so that no data silos formed. The development team solved this by creating an orchestration layer (a centralized hub) where agents exchanged data in real time, and by maintaining a shared database so all agents used the same up-to-date patient information. With this MAS design, the app could, for example, let one agent perform symptom evaluation and immediately pass results to another agent that updates medication plans, all transparently to the user. According to the case report, this multi-agent approach led to significant improvements in response time and personalization (e.g. 95% reduction in patient query response time). This example shows how MAS in healthcare can provide more advanced, interactive services than a single-agent model could, but also underscores the need for careful system design and data management.

Mobility

In transportation and mobility, agent-based architectures are increasingly important for autonomous vehicles and traffic systems.

Autonomous Vehicles

Each autonomous vehicle can be viewed as an intelligent agent controlling perception, decision-making, and actuation. In a single-agent perspective, each car uses one onboard agent (e.g. a neural network) to process sensor

data and navigate. Modern self-driving cars largely follow this approach – each vehicle is autonomous and operates independently. However, there is growing research on *vehicle-to-vehicle (V2V) multi-agent systems*, where multiple cars cooperate. For example, *platooning* is when cars drive closely together at high speed, communicating accelerations and braking to act as a convoy; each car is an agent sharing data for safety and efficiency. Similarly, MAS approaches are used in coordinating fleets of delivery drones or ride-sharing vehicles (each vehicle is an agent that negotiates routes or tasks with others to optimize overall system performance). A recent multi-agent study even proposed using a central LLM to facilitate communication among car-agents, allowing them to share situational awareness and coordinate lane changes or merges. These MAS in mobility can improve safety and throughput, but require robust communication networks and protocols.

Smart Traffic Systems

Urban traffic management is a classic MAS application. Instead of one controller for all traffic lights, a MAS can assign an intelligent agent to each intersection or traffic signal. These agents observe local traffic and coordinate with neighbors to optimize flow. For example, Olusanya et al. (2025) implemented a MAS for traffic signal control using multi-agent reinforcement learning. Each intersection's traffic light was an agent with its own actor-critic model; agents shared traffic state to learn a joint policy. The result was dramatic: in simulation, the MAS reduced vehicle queue lengths and delays by over 60% compared to baseline strategies. Each traffic light agent has its own policy (actor network) and value estimator (critic network), and all agents collectively interact with the SUMO traffic simulator environment. The agents' collaborative decisions adapt in real time to traffic conditions. This case exemplifies how MAS can handle the scale and complexity of city-wide systems, outperforming single-agent control. The trade-offs include the complexity of training many agents and the need for fast inter-agent communication, but the benefits in throughput and adaptability can be substantial.

Overall, mobility applications show that while single-agent autonomous control works for an individual vehicle, multi-agent coordination is key for

system-level efficiency (traffic flow, fleet management). SAS can handle tasks like perception and immediate control for one car, but MAS is needed when vehicles must negotiate or cooperate (e.g. unplanned events, dense traffic, multi-vehicle platooning).

Customer Service

Customer support is another domain seeing both single-agent and multi-agent AI. Traditional chatbots and virtual assistants (e.g. company helpdesk bots, banking chat AIs) are usually single-agent: one chatbot model converses and uses internal logic to answer questions. These systems handle scripted or simple queries effectively.

However, advanced customer service AI increasingly adopts a multi-agent mindset. In a multi-agent customer support system, different AI modules (agents) specialize in tasks like intent recognition, knowledge retrieval, sentiment analysis, and response generation. A central orchestration agent routes queries to the appropriate specialized agents. For instance, one agent might be an NLP engine that classifies a customer's issue, another might fetch data from a CRM, and a third might generate the final reply. These agents communicate and build on each other's outputs. The V7labs analysis of multi-agent AI notes that such systems involve an "AI concierge" or orchestrator that breaks a user's request into subtasks handled by specialist agents.

A practical example is a support ticketing system: the AI might automatically classify a ticket (agent A), prioritize it (agent B), and draft a solution (agent C), each step by a different agent, rather than one giant model doing everything. BoldDesk (2023) describes AI agents that automate workflows, remembering user context across channels and integrating with business tools. For example, Bank of America's virtual assistant "Erica" uses context-aware AI to personalize financial advice by tracking user interactions over time. Such systems illustrate how MAS principles (memory, learning agents, and tool integration) are applied in customer service.

The advantages in this domain include 24/7 automation of repetitive tasks (e.g. ticket sorting, FAQ answering) and more human-like interactions.

According to Gartner, AI agents could resolve up to 80% of routine customer issues by 2029, freeing human agents for complex cases. BoldDesk notes that AI agents can personalize support (remembering user profiles and sentiment) and maintain consistent tone across languages. Multi-agent architectures facilitate this by letting one agent focus on language translation, another on compliance, another on knowledge base retrieval, etc. Challenges include ensuring the agents cooperate seamlessly so the customer sees a unified experience rather than a disjointed handoff.

In summary, in customer service a single intelligent assistant can handle basic dialogue, but a multi-agent setup allows an ecosystem of specialized virtual assistants and services, which can yield richer and more robust customer support. The line between SAS and MAS in this space is blurrier, as many systems fall somewhere in between, but the trend is toward greater modularity and agent collaboration to manage growing user demands.

Conclusion

Single-agent and multi-agent systems represent two complementary approaches to agentic AI design. Single-agent systems favor simplicity and unified reasoning, making them ideal for straightforward, interactive tasks where one model can handle the job. They require no inter-agent communication and are easier to develop and deploy. In contrast, multi-agent systems enable specialization, parallelism, and fault tolerance by distributing a task across multiple autonomous agents. MAS can tackle more complex, large-scale, or long-term tasks that exceed a single agent's capacity. They are more flexible and scalable, but come with greater communication and coordination overhead.

In practice, many real-world AI solutions blend these paradigms, starting with a powerful single agent and evolving toward a multi-agent structure as needs grow. For most customer-facing applications (e.g. chatbots, personal assistants), a single robust agent suffices. But in domains like healthcare coordination, smart transportation, or large-scale automation, multi-agent systems unlock capabilities that single agents cannot provide.

As research and industry continue to explore “agentic” AI, understanding these architectures and their trade-offs is essential. The examples and case studies above illustrate how designers can choose and implement SAS or MAS to meet specific goals, whether it’s the rapid response of a single chatbot or the distributed intelligence of a swarm of autonomous agents.

References

Alliance.xyz. (2024). *Single-Agent vs. Multi-Agent systems*. Alliance Essays. Retrieved October 2025, from https://alliance.xyz/essays/single-agent-vs.-multi-agent

BoldDesk. (2023). *AI Agents in Customer Service: Smarter Support with Virtual Assistants*. BoldDesk Blog. Retrieved October 2025, from https://www./blogs/ai-agents-in-customer-service .

GrowthJockey. (2023). *Single-Agent vs. Multi-Agent Systems: Key Differences*. GrowthJockey Blog. Retrieved October 2025, from https://www./blogs/singleagent-vs-multiagent .

Nirmitee Inc. (2025, February 14). *Case Study: Integrating Multi-AI Agents into Healthcare Apps*. Nirmitee Knowledge Base. Retrieved October 2025, from https:///case-study/integrating-multi-ai-agents-into-healthcare-apps/

Olusanya, O. O., Owosho, Y., Daniyan, I., Elegbede, A. W., Sodipo, Q. B., Adeodu, A., et al. (2025). Multi-agent reinforcement learning framework for autonomous traffic signal control in smart cities. *Frontiers in Mechanical Engineering*, 11:1650918. https://doi.org/10.3389/fmech.2025.1650918 .

Telus Digital. (2023). *Multi-Agent AI Systems: Orchestrating AI Workflows*. V7 Labs Blog. Retrieved October 2025, from https://www./blog/multi-agent-ai