Chapter 7. From One Agent to Many

Scaling systems from a single agent to multiple agents can significantly enhance their ability to solve complex tasks. However, this scalability introduces new challenges that require careful consideration. In this chapter, we explore the advantages and drawbacks of transitioning from one agent to many, discuss how to determine the optimal number of agents for various scenarios, examine different coordination strategies, and review frameworks that facilitate multi-agent system development.

How Many Agents Do I Need?

Determining the optimal number of agents is crucial for effective system performance. This decision hinges on understanding the complexity of the tasks, the environment, and the interactions between agents. The goal is to balance the benefits of multi-agent collaboration against the potential complications it introduces.

Single-Agent Scenarios

Single-agent systems are suitable for tasks that are well-defined, relatively simple, or isolated. In these scenarios, a solitary agent is responsible for performing tasks and making decisions independently. The primary benefits include:

- **Simplicity**: Easier implementation and management.
- **Lower Resource Requirements**: Less computational overhead.
- **Direct Control**: Clear understanding of the agent's behavior.

However, single-agent systems may struggle with tasks requiring diverse skill sets, parallel processing, or adaptability to complex environments. They are best suited for straightforward tasks or environments with minimal complexity. If a single agent is adequate for the use case and performance requirements, there is no need to add further complexity to the system. For most use cases, though, the key bottleneck arises when the number of skills and responsibilities increases. When an agent is expected to choose the correct skill from a set, performance degrades as the potential number of skills increases. At this point, it often makes sense to encapsulate multiple skills into larger groupings, and then choose from the skillset first, and then subsequently choose the skill within the skillset. When the number of skills exceeds the ability to correctly choose a correct skill, then decomposing the skills to distinct agents with appropriate responsibilities can improve reliability and performance.

Multi-Agent Scenarios

In multi-agent systems, multiple agents collaborate to achieve shared goals, an approach that is especially advantageous when tasks are complex and require varied skill sets, parallel processing, or adaptability to dynamic environments. A key benefit of multi-agent systems is specialization: each agent can be assigned specific roles or areas of expertise, allowing the system to leverage each agent's strengths effectively. This division of labor enables agents to focus on defined aspects of a task, which improves efficiency and ensures that specialized skills are applied where they are most needed. By distributing skills and responsibilities across agents, multi-agent systems address the limitations faced by single-agent systems, especially when tasks require expertise across different domains or when the number of skills required exceeds what a single agent can manage reliably.

Parallel processing is another significant advantage. In multi-agent systems, tasks can be split and processed concurrently by different agents, reducing overall execution time and increasing throughput. This parallelism is essential for applications where real-time performance is critical, such as monitoring systems, data processing pipelines, or autonomous vehicle coordination. By operating simultaneously on different parts of a task, agents can accelerate progress and meet the demands of time-sensitive environments more effectively.

Multi-agent systems also offer robustness through redundancy. When multiple agents are available to complete a task, the system can tolerate individual agent failures without disrupting overall functionality. Redundancy ensures that if one agent fails or encounters an obstacle, other agents can take over its responsibilities, allowing the system to maintain stable performance even in unpredictable conditions. This enhanced fault tolerance makes multi-agent systems particularly valuable in mission-critical applications, where uninterrupted operation is essential.

Adaptability is another core advantage, as multi-agent systems can respond dynamically to changing conditions. By coordinating their actions, agents can reallocate roles and responsibilities as needed, adapting to new information or environmental changes in real time. This adaptability allows the system to remain efficient and effective in complex and unpredictable scenarios, where static, single-agent approaches may struggle to keep up.

However, multi-agent systems are not without challenges. With multiple agents interacting, the complexity of coordination increases, requiring sophisticated communication and synchronization mechanisms to ensure agents work harmoniously. Communication overhead is another challenge, as agents must frequently exchange information to stay aligned and avoid duplicating efforts. This need for communication can slow down the system and introduce additional resource demands, especially in large-scale applications. Additionally, conflicts between agents may arise if they pursue overlapping goals or fail to prioritize effectively, necessitating protocols for conflict resolution and resource allocation.

In sum, while multi-agent systems offer powerful advantages in handling complex, multi-faceted tasks, they also require careful planning to manage the additional complexity and coordination requirements they introduce. By assigning agents distinct roles, enabling parallel processing, and incorporating adaptability and redundancy, multi-agent systems can achieve high levels of performance, reliability, and flexibility, particularly in scenarios where a single-agent approach would fall short.

Principles for Adding Agents

When expanding a system by adding more agents, a strategic approach is essential to ensure the system remains efficient, manageable, and effective. The following principles serve as guidelines for optimizing agent-based design and functionality.

Task Decomposition

> Task Decomposition is a foundational principle, emphasizing the importance of breaking down complex tasks into smaller, manageable subtasks. By decomposing tasks, each agent can focus on a specific aspect of the workload, simplifying its responsibilities and improving efficiency. Clear task boundaries reduce overlap and redundancy, ensuring that each agent's contribution is valuable and that no effort is wasted. This decomposition not only enhances individual agent performance but also makes the system easier to coordinate and scale.

Specialization

> Specialization allows agents to be assigned roles that match their strengths, thereby maximizing the system's collective capabilities. When each agent is tasked with activities that align with its specific

functions, the system operates with greater precision and effectiveness. Specialized agents are more adept at handling particular types of work, which translates to improved performance and faster task execution overall. By designing agents with distinct responsibilities, the system can leverage diverse expertise to address complex or multidisciplinary tasks.

Parsimony

Parsimony is a guiding principle that encourages adding only the minimal number of agents necessary to achieve the desired functionality and performance. This principle emphasizes simplicity and efficiency, reminding developers that each agent added to the system introduces additional communication overhead, coordination complexity, and resource demands. By adhering to parsimony, developers avoid unnecessary agent proliferation, which can lead to increased maintenance burdens and potential performance bottlenecks. Parsimony requires careful assessment of each agent's role and a disciplined approach to agent allocation, ensuring that each addition provides clear value to the system. Before adding an agent, developers should consider whether its responsibilities could be fulfilled by existing agents or by enhancing current capabilities. This focus on simplicity results in a streamlined, more manageable system that performs effectively without excessive redundancy. Ultimately, parsimony promotes an efficient, lean multi-agent system that maximizes functionality while minimizing the risks and costs associated with complexity.

Coordination

Coordination is critical for the harmonious operation of multi-agent systems. To maintain alignment among agents, robust communication protocols must be established, facilitating efficient information sharing and reducing the risk of conflicts. Coordination mechanisms should also include protocols for conflict resolution, particularly when agents have overlapping tasks or resource requirements. When agents can exchange information seamlessly and resolve issues autonomously, the system is more resilient and adaptable, capable of responding efficiently to dynamic scenarios.

Robustness

Robustness is essential for enhancing fault tolerance and resilience. Redundancy involves adding agents that can take over if others fail, providing backup support that ensures uninterrupted operation. In high-stakes environments, redundancy is invaluable for maintaining system stability and reliability. Robustness also encompasses designing agents and workflows that can withstand unexpected disruptions, such as network failures or agent downtime. By embedding redundancy and robustness into the system, developers can ensure that it remains functional even in adverse conditions.

Efficiency

Efficiency helps in assessing the trade-offs between adding agents and the potential complexity or resource demands that come with them. Each additional agent increases computational requirements and coordination overhead, so it is crucial to weigh the advantages of expanded functionality against these costs. By carefully evaluating the costs and benefits of each agent addition, developers can make informed decisions that balance system performance, resource efficiency, and scalability.

By following these principles, developers can determine the optimal number and configuration of agents required to achieve the desired balance of performance, efficiency, and complexity. This thoughtful approach enables the creation of multi-agent systems that are both capable and sustainable, maximizing the benefits of additional agents while minimizing potential downsides.

Multi-Agent Coordination

Effective coordination among agents is critical for the success of multi-agent systems. Various coordination strategies can be employed, each with its advantages and challenges. This section explores five primary coordination strategies.

Democratic Coordination

In democratic coordination, each agent within the system is given equal decision-making power, with the goal of reaching consensus on actions and solutions. This approach is characterized by decentralized control, where no single agent is designated as the leader. Instead, agents collaborate and share information equally, contributing their unique perspectives to collectively arrive at a decision. The key strength of democratic coordination is its robustness; because no agent holds a dominant role, the system has no single point of failure. This means that even if one or more agents experience failures, the overall system can continue functioning effectively. Another advantage is flexibility: when agents collaborate openly, they can quickly adapt to changes in their environment by updating their collective input. This adaptability is essential in dynamic settings where responsiveness to new information is crucial.

Moreover, democratic coordination promotes equity among agents, ensuring that all participants have an equal voice, which can lead to fairer and more balanced outcomes.

However, democratic coordination comes with its own set of challenges. The process of reaching a consensus often requires extensive communication between agents, leading to significant communication overhead. As each agent must contribute and negotiate their perspective, the decision-making process can also be slow, potentially causing delays in environments where quick responses are necessary. Furthermore, implementing a democratic coordination protocol is often complex, as it requires well-defined communication and conflict-resolution mechanisms to facilitate consensus-building. Despite these challenges, democratic coordination is particularly well-suited for applications that prioritize fairness and robustness, such as distributed sensor networks or collaborative robotics, where each agent's contribution is valuable and consensus is essential for system success.

Manager Coordination

Manager coordination adopts a more centralized approach, where one or more agents are designated as managers responsible for overseeing and directing the actions of subordinate agents. In this model, managers take on a supervisory role, making decisions, distributing tasks, and resolving conflicts among agents under their guidance. One of the primary advantages of manager coordination is its streamlined decision-making. Since managers have the authority to make decisions on behalf of the group, the system can operate more efficiently, bypassing the lengthy negotiation process required in democratic systems. This centralization also allows managers to clearly assign tasks and responsibilities, ensuring that agents focus on specific objectives without duplicating efforts or causing conflicts. Additionally, manager coordination simplifies communication pathways, as subordinate agents primarily communicate with their designated manager rather than with every other agent, reducing coordination complexity.

However, the reliance on managers introduces certain vulnerabilities. A single point of failure exists because if a manager agent fails or is compromised, the entire system may experience disruptions. Additionally, scalability becomes a concern as the system grows; managers can become bottlenecks if they cannot handle the increased volume of tasks or interactions required in larger networks. Finally, the centralized nature of decision-making in manager coordination can reduce adaptability, as managers may not always be able to make the most informed decisions based on real-time changes within each subordinate's environment. This type of coordination is particularly effective in structured, hierarchical settings like manufacturing systems or customer support centers, where centralized control allows for optimized workflows and quicker conflict resolution.

Hierarchical Coordination

Hierarchical coordination takes a multi-tiered approach to organization, combining elements of both centralized and decentralized control through a structured hierarchy. In this system, agents are organized into multiple levels, with higher-level agents overseeing and directing those below them while allowing subordinate agents a degree of autonomy. This approach provides significant scalability benefits, as the hierarchical structure enables coordination responsibilities to be distributed across multiple levels. By doing so, the system can manage a large number of agents more efficiently than a fully centralized model. The layered design also introduces redundancy, as tasks can be managed at different levels, improving fault tolerance. Clear lines of authority within the hierarchy streamline operations, with higher-level agents handling strategic decisions and lower-level agents focusing on tactical execution.

Despite these advantages, hierarchical coordination presents its own challenges. The complexity of designing a hierarchical system can be substantial, as each level must be carefully structured to ensure smooth coordination between layers. Communication delays can arise due to the need for information to propagate through multiple levels before reaching all agents, which can slow down responsiveness to urgent changes. Additionally, decision-making at higher levels may introduce latency, as lower-level agents may need to wait for instructions before acting. Despite these challenges, hierarchical coordination is well-suited for large, complex systems such as supply chain management or military operations, where different levels of coordination can handle both high-level planning and on-the-ground execution.

Actor-Critic Approaches

Actor-critic approaches, originally developed in the context of reinforcement learning, involve a dual-agent setup where one agent, the “actor,” makes decisions, while another agent, the “critic,” evaluates those decisions and provides feedback. This approach allows agents to learn and adapt over time, improving their performance through feedback loops. A key advantage of actor-critic methods is that they enable adaptive learning, allowing agents to refine their behavior based on evaluative feedback from the critic. This framework supports decentralized decision-making, as each actor can make choices independently while still benefiting from the critic’s feedback. Due to its decentralized nature, the actor-critic model scales well, making it suitable for systems with large numbers of agents that need to operate autonomously but benefit from structured feedback to enhance their performance.

However, implementing actor-critic approaches can be challenging. The underlying algorithms are often sophisticated, requiring considerable computational resources and a nuanced understanding of reinforcement learning techniques. Ensuring that all actors align with the system's overarching goals can also be difficult, especially as each actor seeks to optimize its own decisions based on feedback that may be influenced by local conditions rather than global objectives. Stability is another concern, as learning processes may become erratic or even destabilize the system if the feedback signals are noisy or inconsistent. Nevertheless, actor-critic approaches are highly effective in dynamic environments, such as financial trading systems or adaptive traffic control, where agents benefit from the ability to learn and improve in response to changing conditions.

Automated Design of Agent Systems

Automated Design of Agentic Systems (ADAS) represents a transformative approach to agent development, shifting away from hand-crafted architectures and toward systems that can design, evaluate, and iteratively improve themselves. As articulated by Shengran Hu, Cong Lu, and Jeff Clune[1], the central idea of ADAS is that, rather than manually constructing each component of an agent, we can enable a higher-level "meta-agent" to automatically create, assess, and refine agentic systems. This approach opens up a new research frontier, one that could yield agents capable of adapting to complex, shifting environments and continually improving their own capabilities without direct human intervention. ADAS builds on the idea that, historically, hand-designed solutions in machine learning have often been replaced by learned or automated alternatives, suggesting that agentic systems, too, may benefit from this transition.

In ADAS, Foundation Models serve as flexible, general-purpose modules within an agent's architecture. These models, which already power strategies such as Chain-of-Thought reasoning, Self-Reflection, and Toolformer-based agents, form a base upon which more specialized or task-specific capabilities can be layered. Yet, ADAS seeks to advance beyond these traditional approaches by enabling agents to invent entirely new structures and modules autonomously. The versatility of Foundation Models provides an ideal starting point, but ADAS leverages automated processes to push beyond pre-defined capabilities, enabling agents to evolve novel prompts, control flows, and tool use. These building blocks are not static; rather, they are generated dynamically by the meta-agent, which can continuously experiment with new designs in response to changing requirements or opportunities for improvement.

The backbone of ADAS is the concept of defining agents through code. By utilizing programming languages, which are Turing-complete, this framework theoretically allows agents to invent any conceivable structure or behavior. This includes complex workflows, creative tool integrations, and innovative decision-making processes that a human designer may not have foreseen. The power of ADAS lies in this code-based approach, which treats agents not as static entities but as flexible constructs that can be redefined, modified, and optimized over time. The potential of this approach is vast: in principle, a meta-agent could develop an endless variety of agents, continually refining and combining elements in pursuit of higher performance across diverse tasks.

Central to ADAS is the Meta Agent Search (MAS) algorithm, which demonstrates how a meta-agent can autonomously generate and refine agentic systems. In MAS, the meta-agent acts as a designer, writing code to define new agents and testing these agents against an array of tasks. Each successful design is archived, forming a continuously growing knowledge base that informs the creation of future agents. MAS operates through an iterative cycle: the meta-agent evaluates each agent’s performance, learns from the outcomes, and applies that knowledge to generate new agents with potentially superior capabilities. The meta-agent is thus both a creator and a curator, balancing exploration of new designs with exploitation of successful patterns. This process mirrors the evolution of biological systems, where successful traits are preserved and iteratively modified to adapt to new challenges.

The results of MAS reveal an intriguing property of agents designed through ADAS: they tend to maintain high levels of performance even when applied to new domains and models. For instance, agents developed in specific coding environments have demonstrated surprising efficacy when transferred to other fields, such as scientific or mathematical problem-solving. This robustness across domains suggests that agents created through MAS are not merely optimized for one-off tasks; rather, they embody more general principles and adaptive structures that enable them to excel even when the specifics of the environment change. This cross-domain transferability reflects a fundamental advantage of automated design: by generating agents that are inherently flexible, MAS produces solutions that can generalize more effectively than those designed for narrow, specialized contexts.

ADAS holds significant promise, yet its development requires careful consideration of both ethical and technical dimensions. The potential to automate the design of ever-more-powerful agents introduces questions about safety, reliability, and alignment with human values. While MAS offers a structured and exploratory approach, it is crucial to ensure that the evolving agents adhere to ethical standards and do not develop unforeseen behaviors that could be misaligned with human intentions. Ensuring that these systems are beneficial necessitates a balance between autonomy and constraint, allowing agents the freedom to innovate while guiding them to operate within safe and predictable bounds.

The trajectory of ADAS suggests a future where agentic systems can autonomously adapt, improve, and tackle an expanding range of tasks with minimal human intervention. As ADAS advances, the ability of agents to develop more sophisticated designs will likely become a cornerstone of AI research, providing tools that can address increasingly complex, evolving challenges. In this way, ADAS offers a glimpse into a future of intelligent systems capable of self-improvement and innovation, embodying a shift from static, pre-designed agents to adaptive, autonomous systems that grow alongside our expanding needs.

Multi-Agent Frameworks

Several frameworks facilitate the development and deployment of multi-agent systems. Selecting the appropriate framework depends on the specific needs and constraints of the project.

Do-It-Yourself (DIY)

A do-it-yourself (DIY) approach to multi-agent systems offers the ultimate flexibility, allowing developers to build a system tailored to their specific needs without the constraints or assumptions of pre-built frameworks. This approach is ideal for highly specialized applications or experimental research where the particularities of agent interaction, communication protocols, and coordination mechanisms need to be closely controlled and fine-tuned. By developing a custom multi-agent infrastructure, developers can optimize performance for specific hardware configurations, integrate unique communication standards, or test novel agent behaviors that existing frameworks may not support.

However, the DIY approach is not without challenges. Building a robust multi-agent system from scratch requires a deep understanding of distributed systems, network protocols, and software engineering best practices. The lack of built-in support for common tasks, such as agent lifecycle management or fault tolerance, means that every aspect of the system—from the architecture to the communication protocols—must be carefully designed and implemented. Despite these challenges, the DIY approach is invaluable for pushing the boundaries of multi-agent systems and exploring new possibilities in agent coordination.

LangGraph

LangGraph is a multi-agent framework focused on language-based interactions, leveraging natural language as a primary means of communication and decision-making among agents. In LangGraph, agents communicate through structured language exchanges, making it particularly well-suited for applications that rely on rich, context-sensitive information exchange, such as collaborative problem-solving, instructional systems, and conversational AI.

The framework provides tools to facilitate language-based negotiation, goal setting, and knowledge sharing among agents, making it possible for agents to dynamically adjust their strategies based on the evolving context of their language interactions. LangGraph also includes features for tracking conversational history and context, enabling agents to maintain coherence in their interactions over extended dialogues. While LangGraph's language-centered approach offers powerful capabilities for collaborative applications, it can be limited in scenarios where real-time performance or direct action coordination is essential. Nevertheless, LangGraph's unique focus on language provides a valuable platform for exploring multi-agent systems in domains where communication and interpretability are as important as raw computational power.

Autogen

Autogen stands out as a leading framework for managing complex multi-agent interactions, especially in environments where agents must work together to achieve goals through coordinated actions and decision-making. Autogen's strength lies in its versatility, providing built-in support for various types of agent coordination, such as task sharing, parallel processing, and hierarchical management. The framework is highly extensible, allowing developers to define custom agent behaviors, decision-making protocols, and reward structures. Autogen also offers robust logging and monitoring tools, which facilitate debugging and provide insights into agent behavior patterns, making it particularly valuable for research applications. Additionally, Autogen's modular architecture enables easy integration with machine learning models, reinforcement learning algorithms, and other AI components, allowing agents to incorporate advanced learning capabilities into their interactions. Autogen's extensive support for multi-agent workflows, coupled with its flexible configuration options, makes it an ideal choice for both research-oriented projects and real-world applications where coordination efficiency and agent scalability are paramount.

Crew AI

CrewAI is a framework designed to orchestrate role-playing, autonomous AI agents, fostering collaborative intelligence to tackle complex tasks. By enabling agents to assume specific roles, share goals, and operate cohesively, CrewAI facilitates the development of sophisticated multi-agent systems.

The framework offers a range of features to streamline the creation and management of AI workflows. Developers can utilize CrewAI's framework or UI Studio to build multi-agent automations, whether coding from scratch or leveraging no-code tools and templates. Once built, these "crews" can be deployed confidently with powerful tools for different deployment types, including autogenerated user interfaces. CrewAI also provides monitoring capabilities to track the performance and progress of each crew, enabling continuous improvement through testing and training tools. (CrewAI)

CrewAI's modular architecture supports seamless integration with various large language models (LLMs) and cloud platforms, offering flexibility in deployment. The framework's emphasis on role-based task allocation allows agents to focus on specialized aspects of a workload, enhancing efficiency and collaboration. Additionally, CrewAI's support for real-time communication protocols ensures that agents can update one another on task progress and make necessary adjustments dynamically.

By providing a structured yet flexible approach to multi-agent orchestration, CrewAI serves as a valuable resource for developers and researchers interested in exploring the dynamics of agent coordination and execution. Its comprehensive suite of tools and features makes it an ideal choice for both research-oriented projects and real-world applications where coordination efficiency and agent scalability are paramount.

Swarm

Swarm is an experimental educational framework developed by OpenAI's Solutions team, designed to explore ergonomic and lightweight multi-agent orchestration. It focuses on making agent coordination and execution highly controllable and easily testable. Swarm achieves this through two primary abstractions: Agents and handoffs. An Agent encompasses instructions and tools, and can at any point choose to hand off a conversation to another Agent. These primitives are powerful enough to express rich dynamics between tools and networks of agents, allowing developers to build scalable, real-world solutions while avoiding a steep learning curve.

Swarm is entirely powered by OpenAI's Chat Completions API and is stateless between calls. It is not intended for production use and does not have official support. The primary goal of Swarm is to showcase the handoff and routines patterns explored in OpenAI's "Orchestrating Agents: Handoffs & Routines" cookbook. It is not meant as a standalone library and is primarily for educational purposes.

Swarm's client.run() function is analogous to the chat.completions.create() function in the Chat Completions API—it takes messages and returns messages without saving state between calls. Importantly, it also handles Agent function execution, handoffs, context variable references, and can take multiple turns before returning to the user.

By providing a structured yet flexible approach to multi-agent orchestration, Swarm serves as a valuable resource for developers and researchers interested in exploring the dynamics of agent coordination and execution.

Conclusion

The transition from single-agent to multi-agent systems offers significant advantages in addressing complex tasks, enhancing adaptability, and increasing efficiency. Yet, as we've explored in this chapter, the scalability that comes with adding more agents brings challenges that demand careful planning. Deciding on the optimal number of agents requires a nuanced understanding of task complexity, potential task decomposition, and the cost-benefit balance of multi-agent collaboration.

Coordination is critical to success in multi-agent systems, and a variety of coordination strategies—such as democratic, manager-based, hierarchical, actor-critic approaches, and automated design with ADAS—provide different trade-offs between robustness, efficiency, and complexity. Each coordination strategy offers unique advantages and limitations, suited to particular scenarios, and careful selection can significantly enhance a system's effectiveness and reliability.

Choosing an appropriate framework is equally important. Custom-built solutions allow for maximum flexibility, but established frameworks like Autogen, Crew AI, and Swarm offer structured, scalable architectures and powerful tools for managing agent interactions and task orchestration. Selecting the right framework ensures not only that the multi-agent system can handle current requirements but also that it remains adaptable for future expansion and unforeseen challenges.

By understanding these factors and applying them thoughtfully, developers can create multi-agent systems that are not only robust and capable but also prepared to meet the demands of increasingly complex, dynamic tasks in real-world applications. This strategic approach enables multi-agent systems to evolve as powerful solutions that drive meaningful advancements across various domains.

[1] Automated Design of Agentic Systems *https://arxiv.org/pdf/2408.08435*