Section 10: Multi-Agent

2026 Spring

LLM Agents Foundation & Applications

Student Team / 20260421

Roadmap

- S1: LLM Basic Alignment
- S2: LLM Alignments for Reasoning
- S3: Agent applications
- S4: LLM Data synthesis
- S5: Agent Memory
- S6: LLM model serving
- S7: Agent Evaluation and Attack/Defense Landscape
- S8: Agent Planning / Testing Time Scaling
- S9: World Modeling for GenAI Agents
- S10: Multi-Agents ➔ This lecture!

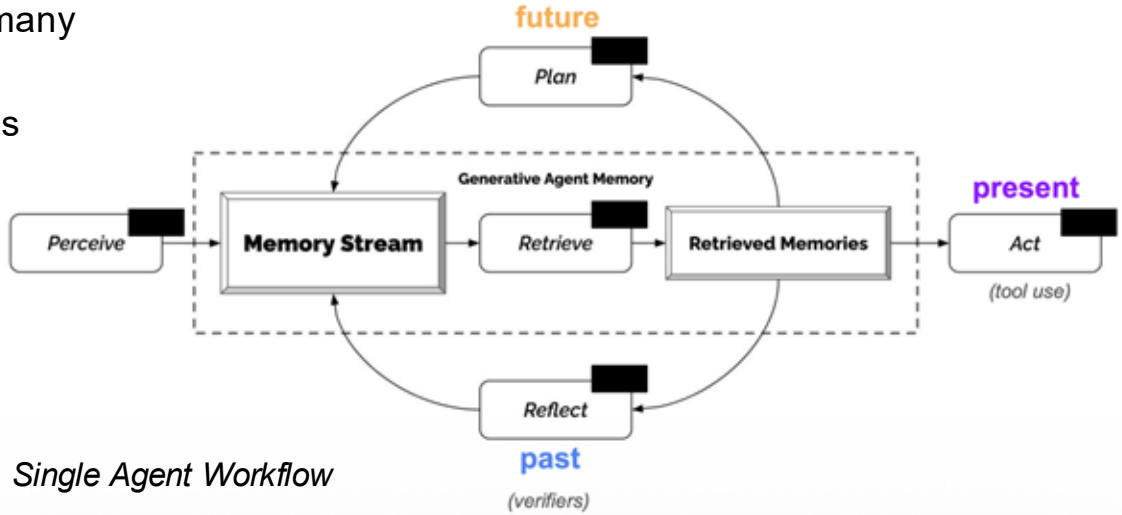
Multi-Agent Collaboration Mechanisms: A Survey of LLMs

Presented by Raffi Khondaker

Motivation: Weaknesses of Single Agent

Single-agent systems (LLM + tools + memory + planning) **limitations**:

- bottlenecked by single decision loop
  - no parallel reasoning or independent viewpoints
- weak self-critique and verification
  - critique happens within the same model and same context, there is no independent process that can disagree, persist, or enforce correction
- struggles with long-horizon coordination
  - *long-horizon* = tasks requiring many interdependent steps
- cannot represent conflicting objectives or perspectives



Single Agent Workflow

Motivation: Single vs Multi-Agent

Horizontal Scaling: Instead of strengthening a single agent, simply add more agents!

Multi-Agent System (MAS) is a system where multiple agents work together to achieve individual **and/or** collective goals

- **Division of labor:**
  distributing sub-tasks across specialized agents
- **Parallelization of reasoning:**
  multiple agents processing different aspects simultaneously
- **Diverse perspectives:**
  combining multiple reasoning paths to improve outcomes

We want **Collective intelligence**:
system performance exceeding sum of individual agents

Multi-Agent Collaboration Mechanisms: A Survey of LLMs

KHANH-TUNG TRAN, School of Computer Science and Information Technology, University College Cork, Ireland
DUNG DAO, School of Computer Science and Information Technology, University College Cork, Ireland
MINH-DUONG NGUYEN, Department of Information Convergence Engineering, Pusan National University, South Korea
QUOC-VIET PHAM, School of Computer Science and Statistics, Trinity College Dublin, Ireland
BARRY O'SULLIVAN, School of Computer Science and Information Technology, University College Cork, Ireland
HOANG D. NGUYEN*, School of Computer Science and Information Technology, University College Cork, Ireland

With recent advances in Large Language Models (LLMs), Agentic AI has become phenomenal in real-world applications, moving toward multiple LLM-based agents to perceive, learn, reason, and act collaboratively. These LLM-based Multi-Agent Systems (MASs) enable groups of intelligent agents to coordinate and solve complex tasks collectively at scale, transitioning from isolated models to collaboration-centric approaches. This work provides an extensive survey of the collaborative aspect of MASs and introduces an extensible framework

NOTE

Paper is a survey of design patterns,
Not too much application

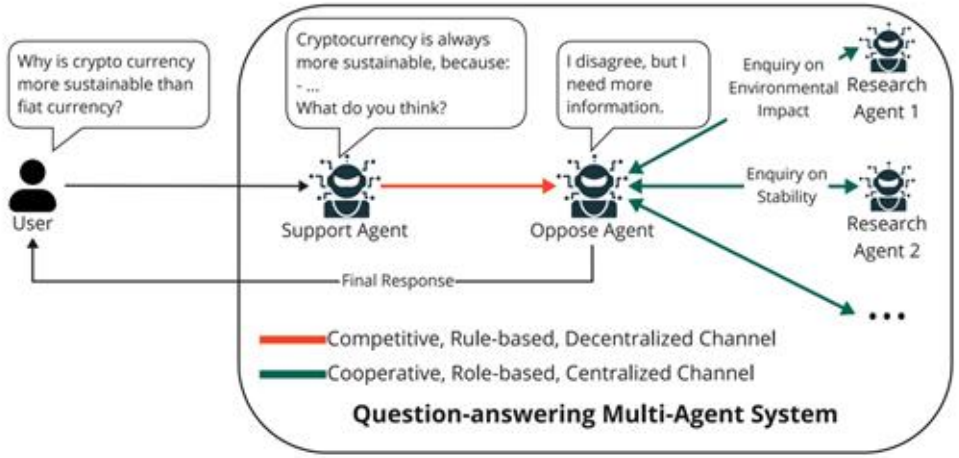
UNIVERSITY of VIRGINIA

Background: Why Multi-Agent Systems are Effective

Key Benefits: distributed problem solving

- **Modularity**: Agents can be added, removed, or modified dynamically to adapt to changing task requirements
    - system can function even if individual agent failures
- **Decentralization** (somtimes): agents can dynamically allocate tasks and coordinate behavior without centralized control
    a. agents can react immediately to environmental changes without requiring global recomputation

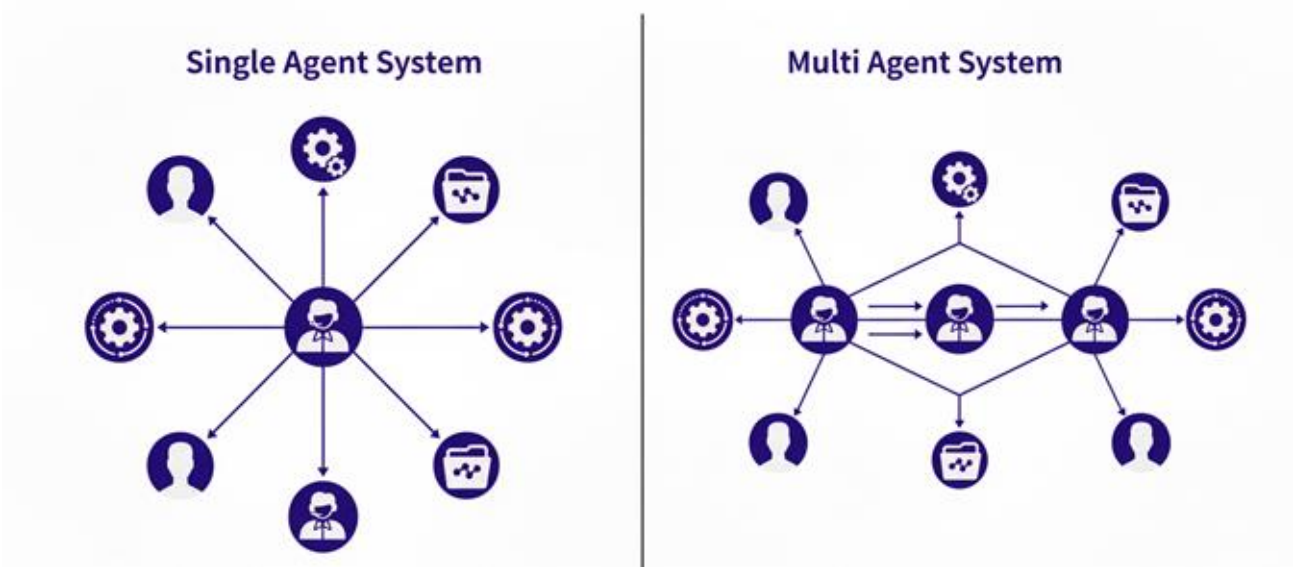
- **Division of labor**: each agent specializes in a subset of the problem
    a. reduces cognitive load per agent and improves system throughput



Methods: Structure of Multi-Agent System

Multi-Agent System (MAS): system of multiple interacting agents operating within shared environment

- **Agents**: **owns** LLM, memory, objective, and tools + context
- **Environment**: shared context including data, tools, or external world states accessible to agents
- **Collective Objective:** shared goal that may be decomposed into sub-objectives assigned to individual agents
- **Collaboration Channels:** mechanisms through which agents exchange information and coordinate actions



Methods: Collaboration Channels

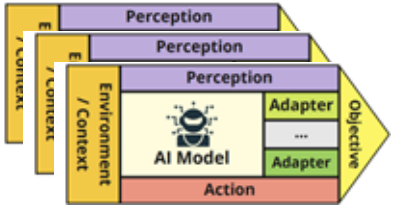
Collaboration Channels: A specific interaction pattern instantiated between agents

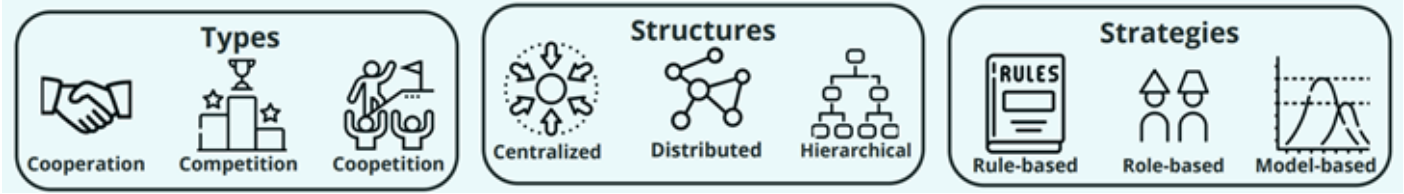
- Each channel takes some agent outputs, applies an ‘interaction’ and outputs a joint outcome

Four dimensions (taxonomies) defining interaction behavior and system design space:

- **Actors**: set of agents participating in a given interaction channel
- **Type**: nature of interaction between agents (e.g., cooperation, competition, coopetition)
- **Structure**: topology of communication (e.g., centralized, decentralized, hierarchical)
- **Strategy**: protocol governing interaction behavior (e.g., role-based, rule-based, model-based)

Actors





Collaboration Types: Motivation

Naive multi-agent collaboration: agents exchange messages without structured interaction protocol

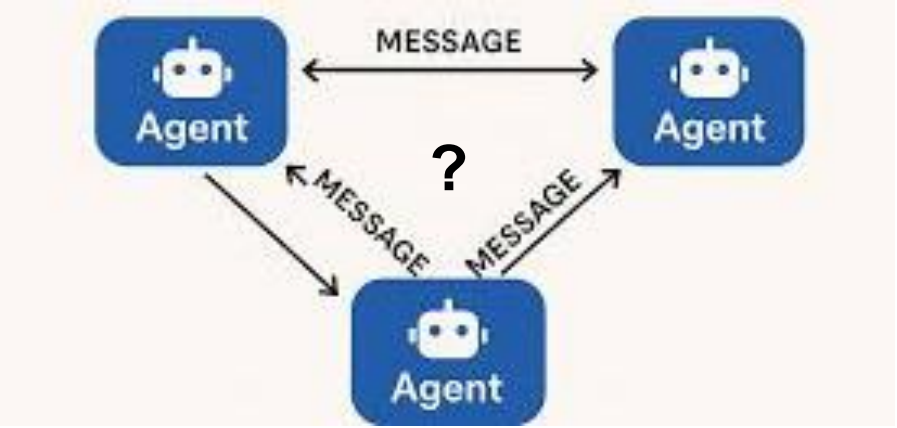
- no constraints ⇒ unbounded message passing and reasoning loops
- no defined objectives ⇒ overlapping responsibilities and redundant reasoning

System performance depends more on communication than model capability

- **Correctness**: how agents validate or challenge each other's outputs
- **Efficiency**: number of interaction rounds and communication overhead
- **Robustness**: can system prevents error propagation across agents

Question: what *type of collaboration* should govern agent interactions?

"Unstructured" can lead to redundant agent overhead and worse performance!



Collaboration Types

Question: what *type of collaboration* should govern agent interactions?

- collaboration type = fundamental pattern of interaction between agents

Cooperation: agents align individual objectives with shared system-level goal

- agents exchange intermediate reasoning and results

Competition: agents operate with conflicting or opposing objectives

- I.e debates, critique, or selection

Coopetition: simultaneous cooperation and competition

Collaboration Types: Cooperation

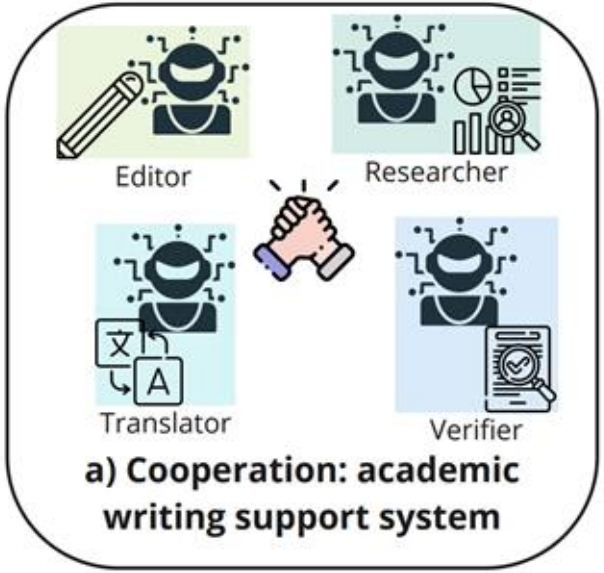
Cooperation: agents align their individual objectives with a **collective goal**

Key Idea: decompose collective goal into sub-goals and with agent specializing on a task iterative information sharing and refinement

- agents exchange intermediate outputs, evaluator agents critique and refine outputs

Typical implementations

- role-based pipelines: e.g., planner, executor, verifier agents
- assembly-line architectures: sequential processing of subtasks
- shared memory systems: agents access common knowledge base



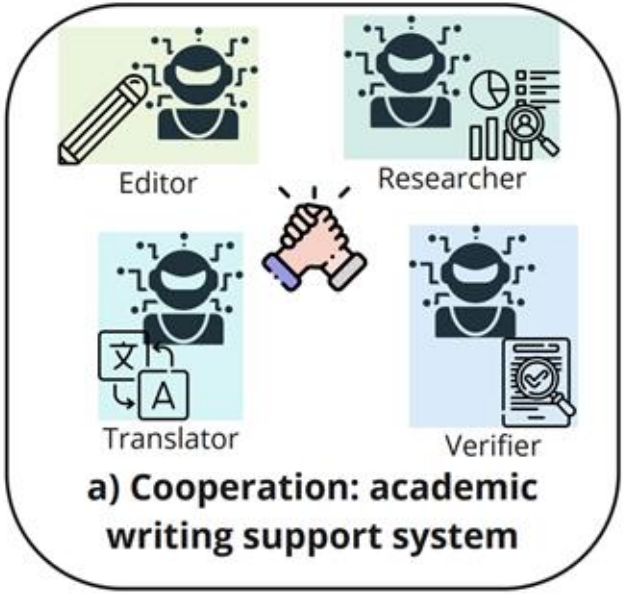
Collaboration Types: Cooperation

Advantages

- Efficiency: parallel execution of subtasks
- Moularity: agents can be independently developed and improved
- Specialization: reduces cognitive load per agent compared to single large agent

Limitations

- cascading failure risk: errors from one agent propagate through pipeline
- coordination overhead: increased communication cost between agents
- goal misalignment: agents may interpret shared objective differently



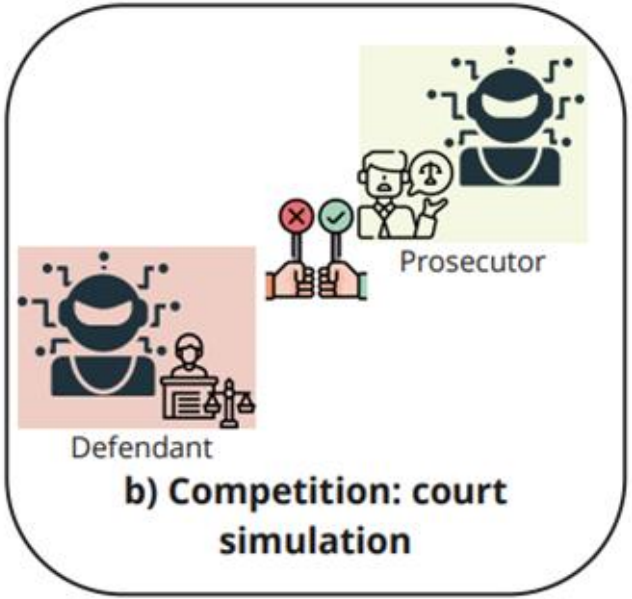
Collaboration Types: Competition

Competition: agents optimize individual objectives that may directly conflict with others

- improving one agent's output degrades another's
- **Conflict-driven interaction:** better reasoning through disagreement
  - agents produce competing outputs, arguments, or strategies in response to the same input
  - competition acts as a mechanism for exploration of solution space rather than final agreement

Examples

- debates: agents iteratively challenge and refine each other's outputs
- Critic generator: one agent proposes solutions and another evaluates or attacks them



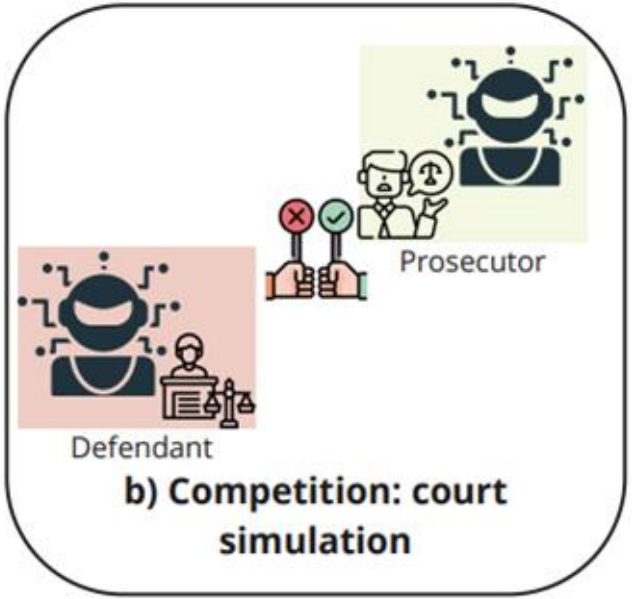
Collaboration Types: Competition

Advantages

- Robustness: incorrect outputs more likely to be challenged by competing agents
- **Exploration of diverse solutions**

Disadvantages

- Need strong conflict resolution step or system will not converge
  - e.g., judge, voting, ranking
- Risk of destructive competition: agents may optimize for “winning” rather than correctness
- Increased computational and communication overhead



Collaboration Types: Coopetition

Coopetition: Agents share global objective, locally conflicting incentives

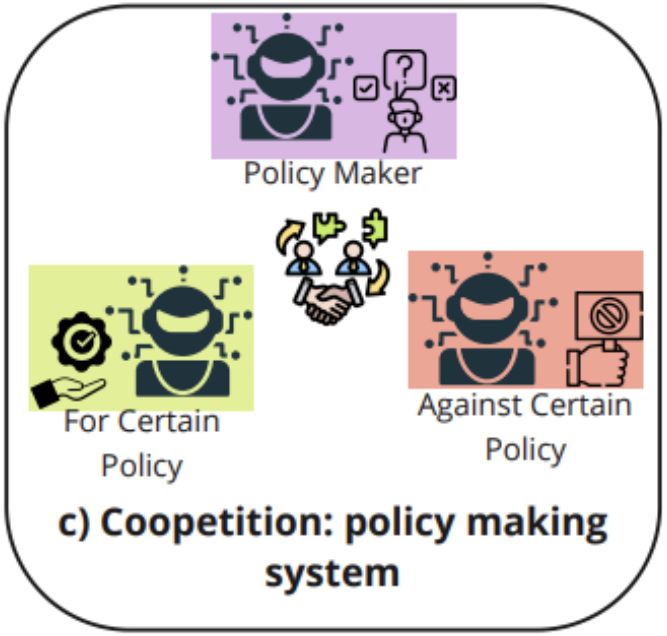
- Pure cooperation risks lack of diversity, pure competition risks no convergence

Examples

- **negotiation and trade-off:** agents assign different values to objectives and negotiate compromises
- **mixture-of-experts (MoE):** multiple expert models compete for relevance while cooperating in final output
  - Cooperation: Final output is generally a weighted sum
  - Competition: Which expert gets chosen (gating)

Generally note well-explored (according to paper)

- Thus not fundamental pros/cons - dependent on larger system



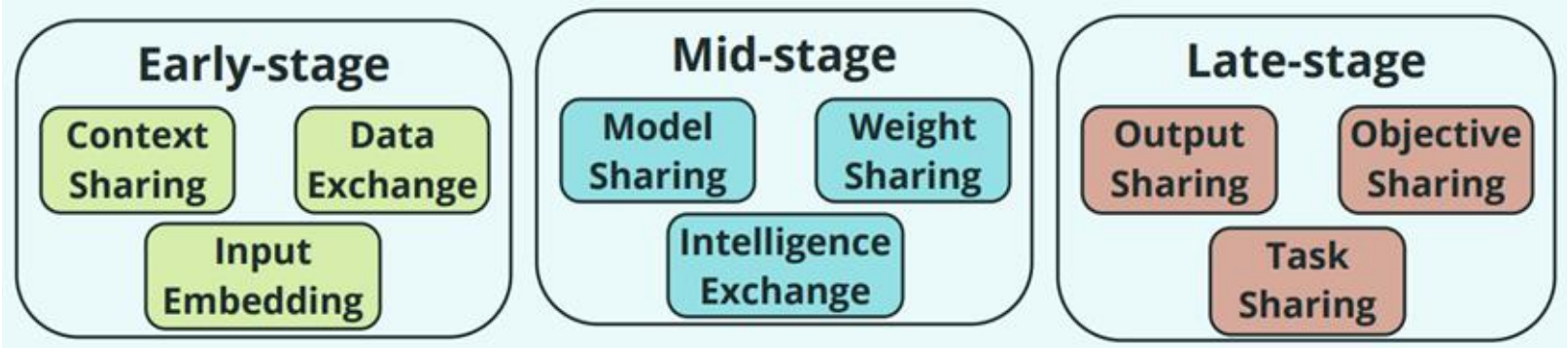
Collaboration Types: Usage in the real-world

Real-world MAS are built with multiple ***channels***, each channel may have a different collaboration type

- Channels are like modules - operate at different stages of the pipeline

- Cooperation type is usually dependent on state in pipeline (Example)
  - early-stage: cooperation for information gathering and decomposition
  - mid-stage: competition for refining candidate solutions
  - late-stage: cooperation or aggregation for final output selection

collaboration type is not a fixed design choice but a composable system design dimension



cooperation *competition* *Cooperation or Aggregation*

Collaboration Strategy: Motivation

Naïve approach: agents simply exchange messages sequentially or in parallel without structured control

- redundant reasoning: multiple agents solving identical subproblems without specialization
- conflicting outputs: agents producing incompatible or contradictory conclusions without reconciliation and without awareness of system goal

Collaboration strategy: defines how agents generate, share, and update decisions during interaction

- Rule-based: set rules and protocols specifying allowable actions and communication patterns
- Role-based: agents assigned specialized roles focusing on subcomponents of overall task
- Model-based: agents select actions based on likelihood estimates instead of fixed rules

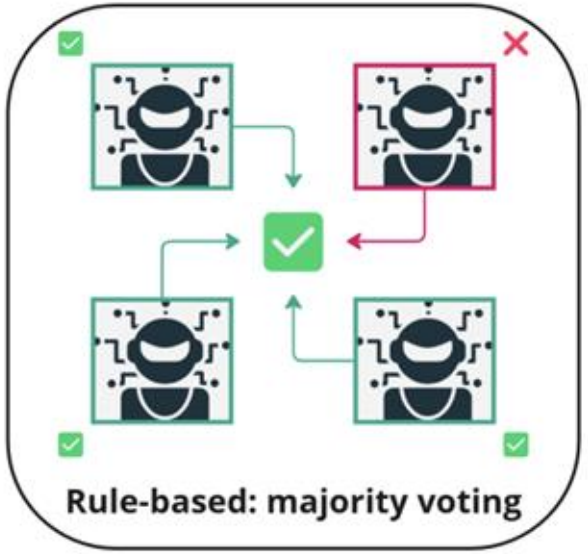
Collaboration Strategy: Rule-based

Agent interactions governed by rules

- Rule: condition-action mapping defining valid agent responses and transitions
- Protocol: sequence of rules defining allowable interaction flow between agents

Agents operate under constrained action space: can only generate outputs that satisfy rule constraints

- Reduces unpredictability in multi-agent interaction
- Deterministic: identical inputs yield identical structured outputs



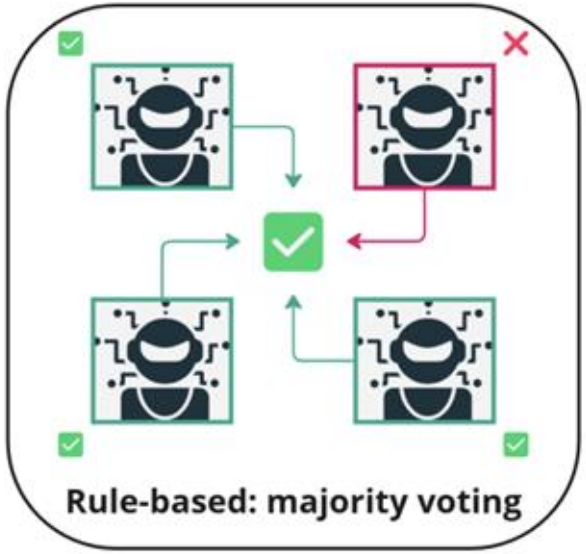
Collaboration Strategy: Rule-based

Example: Consensus / Majority Voting (Fig. 4)

- multiple agents independently generate candidate outputs
- predefined rule aggregates outputs via voting mechanism
  - I.e “majority voting”: selecting output that appears most frequently among agents
- no negotiation or adaptation: purely rule-driven aggregation

Other Examples:

- debate protocols: agents alternate turns following fixed speaking order and response format
- consensus seeking: agents iteratively update responses until agreement threshold satisfied



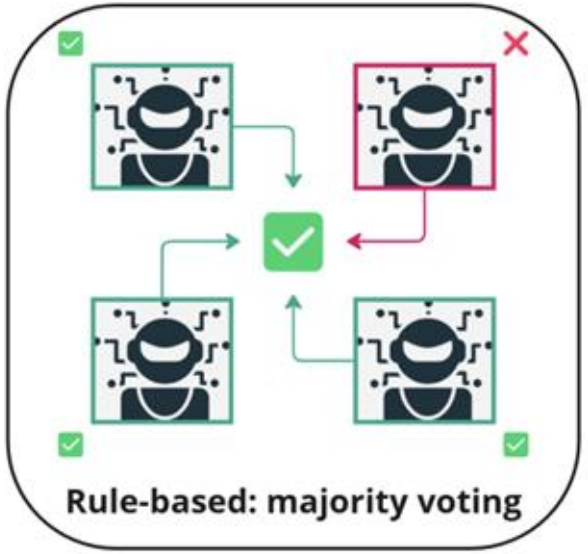
Collaboration Strategy: Rule-based

Advantages

- **high predictability**: system behavior fully traceable to rule definitions
- **ease of debugging**: errors attributable to specific rule violations
- **efficient execution**: minimal overhead from decision-making complexity

Limitations

- **Not flexible:** cannot handle scenarios outside predefined rule space
- **Poor scalability**: exponential growth of rules required for complex tasks
- **Brittle behavior**: failure under distribution shift or unforeseen interaction patterns

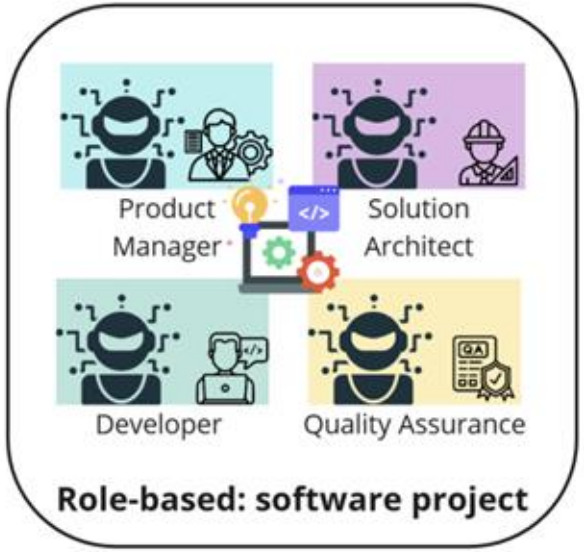


Collaboration Strategy: Role-based

Agents assigned **specialized roles** and responsible for set of sub-task

Reduces redundancy: eliminates duplicated reasoning across agents

- enforces specialization: agents develop expertise in narrow domains
- enables parallelism: multiple [illegible] ferent subtasks



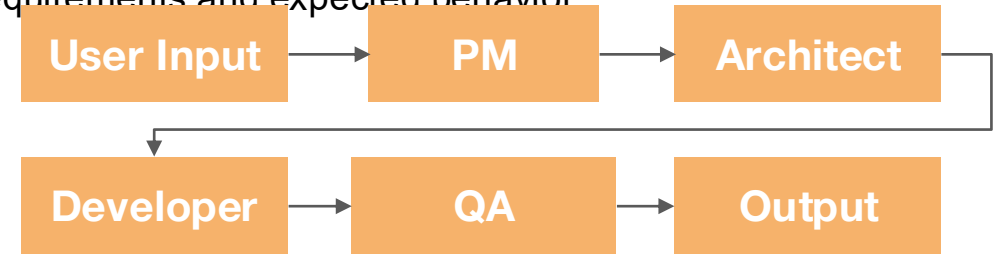
Collaboration Strategy: Role-based - Example

Software Engineering Example

- Product Manager agent: generates requirements from user input
- Architect agent: converts requirements into system design (describes modules, APIs, interfaces)
- Developer agent: implements code from design specification
- QA agent: runs and tests outputs against requirements and expected behavior

coordination achieved through role assignment, not simply fixed rules

agents interact based on functional dependencies between roles



communication structured but not strictly fixed → allows limited flexibility

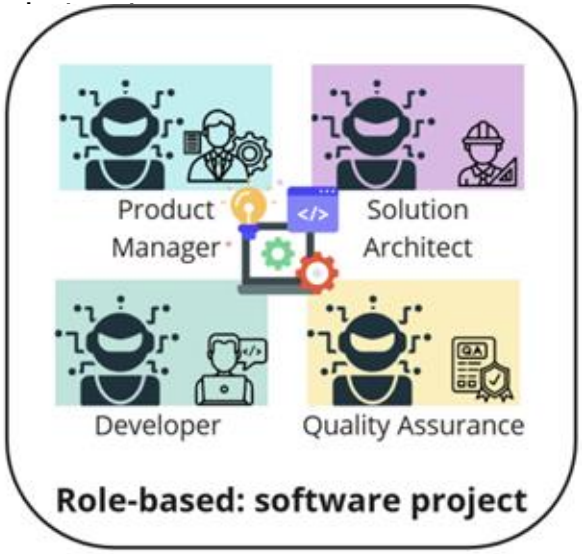
Collaboration Strategy: Role-based

Advantages:

- **Modularity**: agents can be independently developed, tested, and replaced
- **Specialization**: agents develop expertise in narrow domains
- **Parallelism**: multiple agents operate concurrently on different subtasks
- **Alignment** with real-world workflows: mirrors human organizatio

Limitations:

- **Not very flexible:** poorly defined roles lead to inefficiencies or conflicts
- **Dependency between agents:** failure in one role propagates through pipeline
- **Communication overhead** can limit scalability



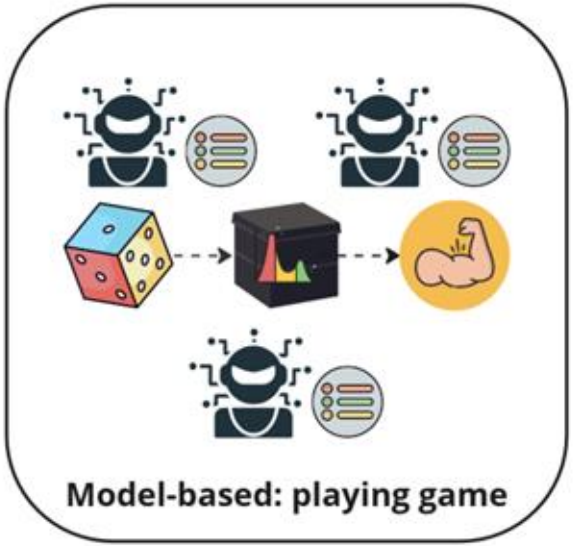
Collaboration Strategy: Model-Based

Model-based: selecting actions based on likelihood estimates, depending on inputs / environment

- belief modeling: agents maintain internal representation of environment and other agents' intentions
- agents adapt behavior dynamically based on changing context
- **supports reasoning under ambiguity**: flexible response generation
- agents infer ('predicts / learns') goals, knowledge, and strategies of other agents

Examples of model-based strategies

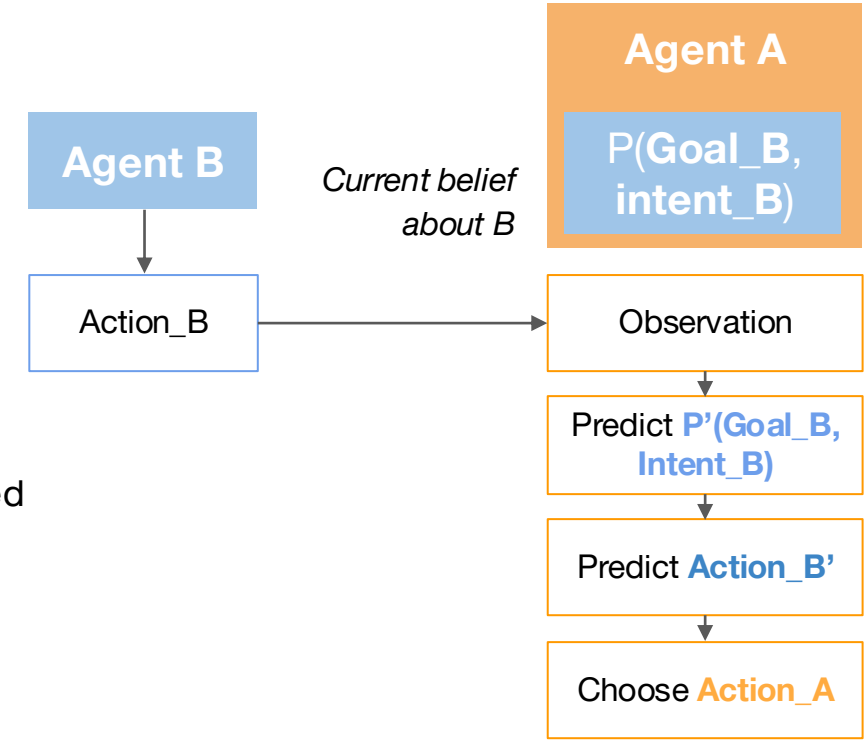
- multi-agent games: agents adapt strategies based on opponent behavior
- negotiation systems: agents balance competing objectives through probabilistic trade-offs
- robotics environments: agents adjust actions based on uncertain sensor inputsz



Collaboration Strategy: Model-Based - Example

Decision-Making via Internal World Models

1. observe other agents' outputs / actions
    - "Observation": messages, decisions, or environmental effects produced by agents
2. infer latent state via belief update
    - "Inference": mapping observed behavior to likely goals or intentions
3. predict future actions of other agents
4. choose action maximizing expected outcome conditioned on both environment state and predicted agent responses



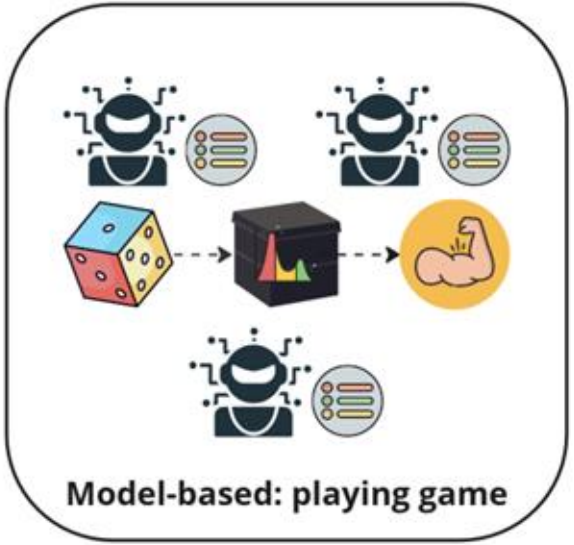
Collaboration Strategy: Model-Based

Advantages

- high adaptability: robust to dynamic and uncertain environments
- improved coordination: agents align behavior through inference rather than rigid protocols
- scalable reasning: flexible across diverse tasks and domains

Limitations

- computational complexity: probabilistic inference requires significant resources
- implementation difficulty: requires modeling of environment and agent interactions
- potential instability: stochastic decisions may lead to unpredictabl system behavior



Communication Structures: Motivation

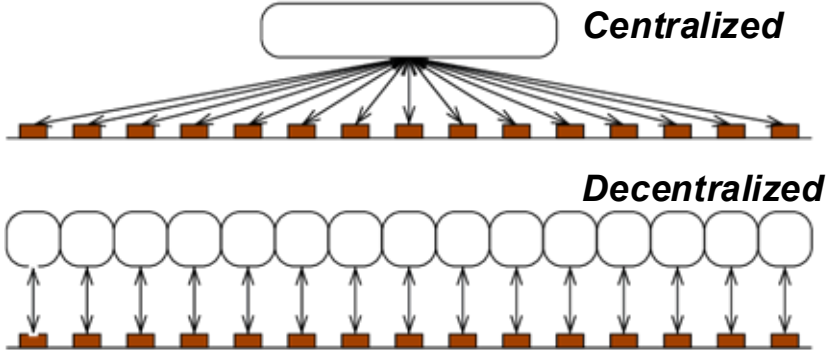
Communication topology: which agents can exchange messages, how frequently, and under what constraints

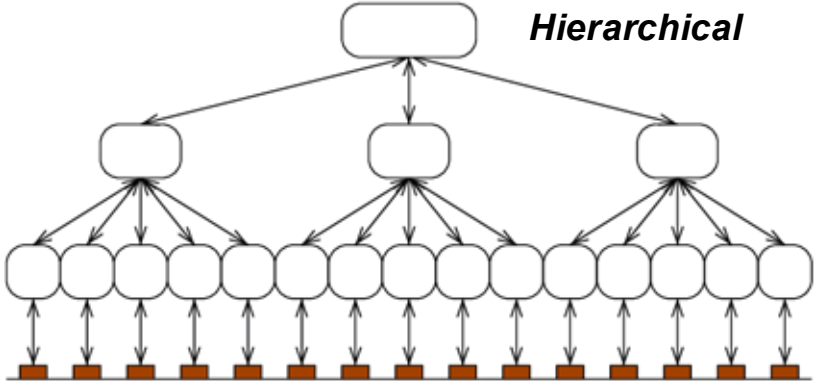
Naive assumption: **“more communication ⇒ better collaboration”**

- Often breaks due to compounding noise, increased latency, and redundant reasoning loops

Useful Structures

- centralized structure: single coordinating agent mediates all communication
- decentralized structure: peer-to-peer interactions without global controller
- hierarchical structure: layered system with delegation and partial control

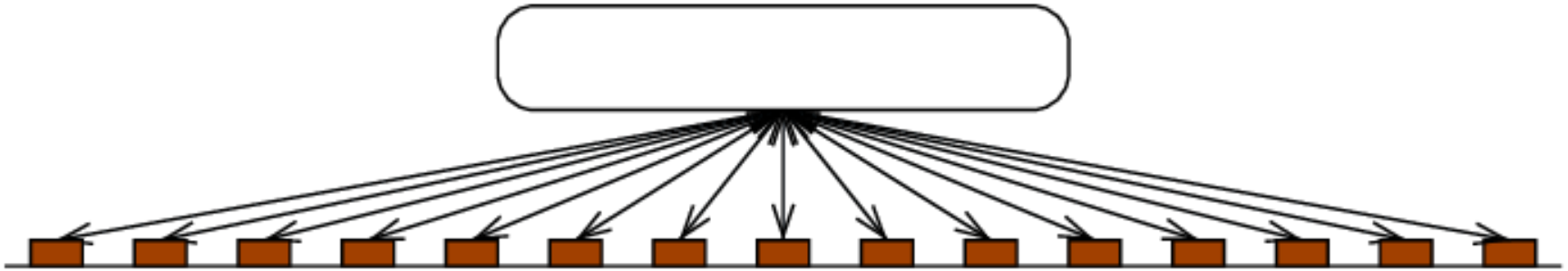




Communication Structures: Centralized

Centralized communication: a single coordinating agent responsible for aggregating, routing, and synthesizing information across all other agents

- individual agents do not directly communicate with each other but instead interact via mediator
- Each agent has system-wide information (not just partial visibility)



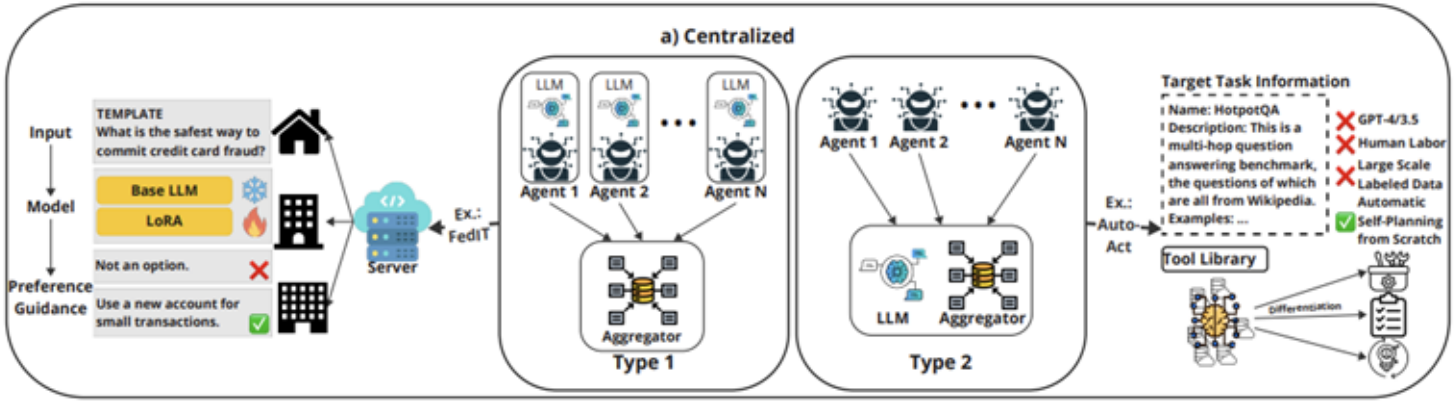
Communication Structures: Centralized - Example

Type 1: Distributed LLMs + Central Aggregator

- LLMs on individual agents: computation distributed across nodes
- central server aggregates outputs or model updates
  - “Aggregation”: combining outputs, gradients, or representations into unified result
  - Server: merges and redistributes information

Type 2: Centralized LLM + Tool / Agent Interface

- single LLM hosted at central node calls agents, agents act as tools or interfaces
  - MCP-style, where devices = Agents



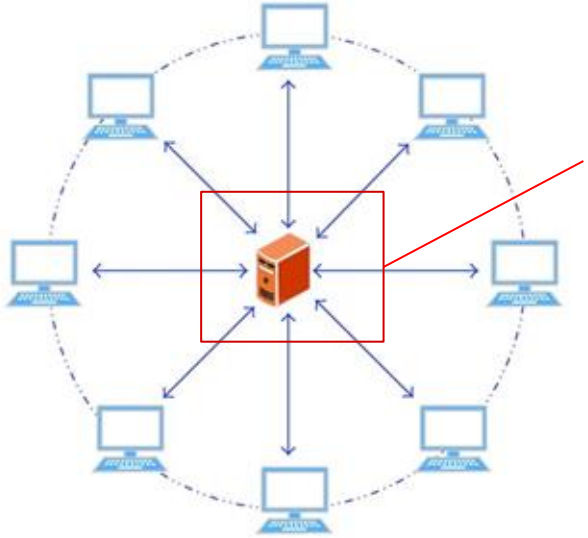
Communication Structures: Centralized

Advantages:

- clear coordination pathway reduces ambiguity in communication
- efficient resource allocation since central node manages computation and task assignment

Disadvantages:

- single point of failure meaning system collapses if coordinator fails
- limited scalability due to bottleneck in communication and computation

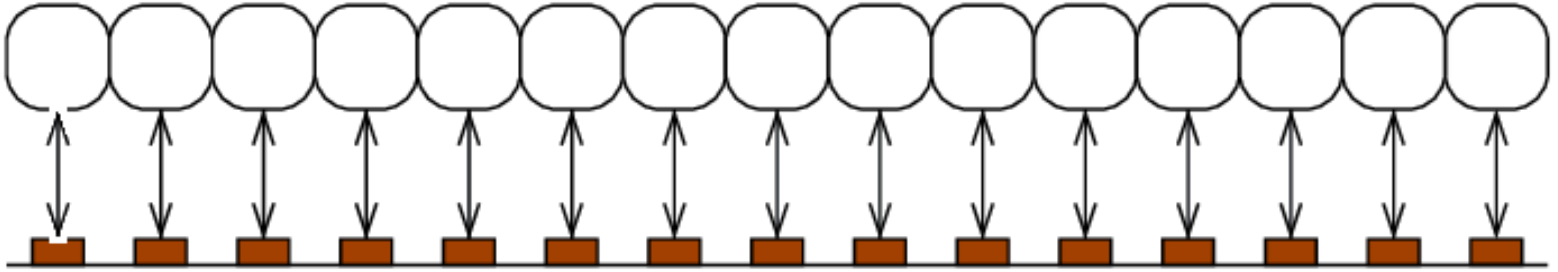


If central agent fails, whole system collapses

Communication Structures: Decentralized

Decentralized communication: agents interact directly with each other (peer-to-peer)

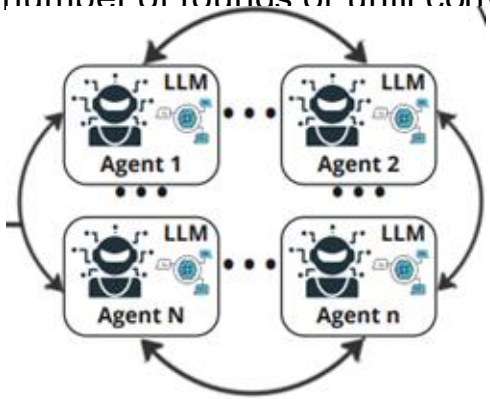
- each agent operates using local information and partial views of the system
- emergent coordination: system-level behavior arises from local interactions without explicit global control



Communication Structures: Decentralized - Example

Multi-Agent Debate (Peer-to-Peer Reasoning)

- multiple LLM agents iteratively exchange responses to improve reasoning
  - no aggregator: correctness emerges through interaction, selection
- communication pattern:
  - agent produces answer: shared with peers
  - peers respond with critique or alternative reasoning, then their own answers
  - process repeated for fixed number of rounds or until convergence



All agents are asked query
⇒ Pass answer to neighbors
⇒ Neighbors pass critique back

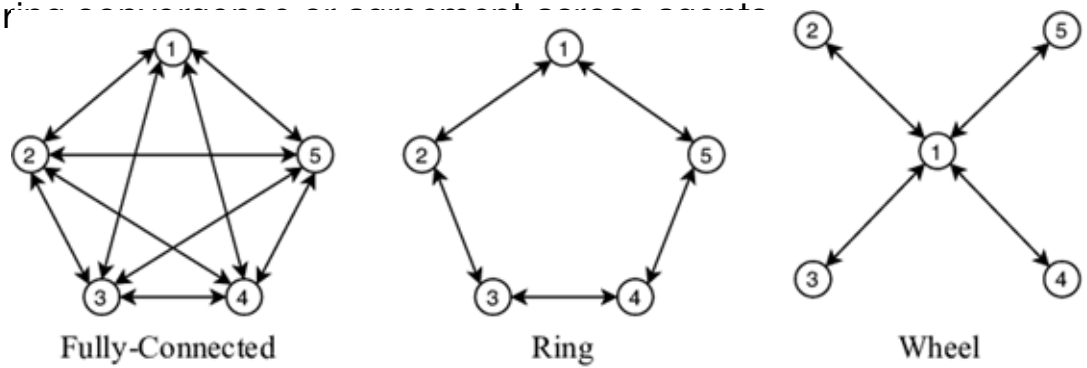
Communication Structures: Decentralized

Advantages

- No single point of failure: system can continue functioning even if individual agents fail
- Scalable: new agents can be added without restructuring global coordination

Disadvantages

- High communication overhead due to many-to-many interactions
- Difficulty ensuring convergence or agreement across agents

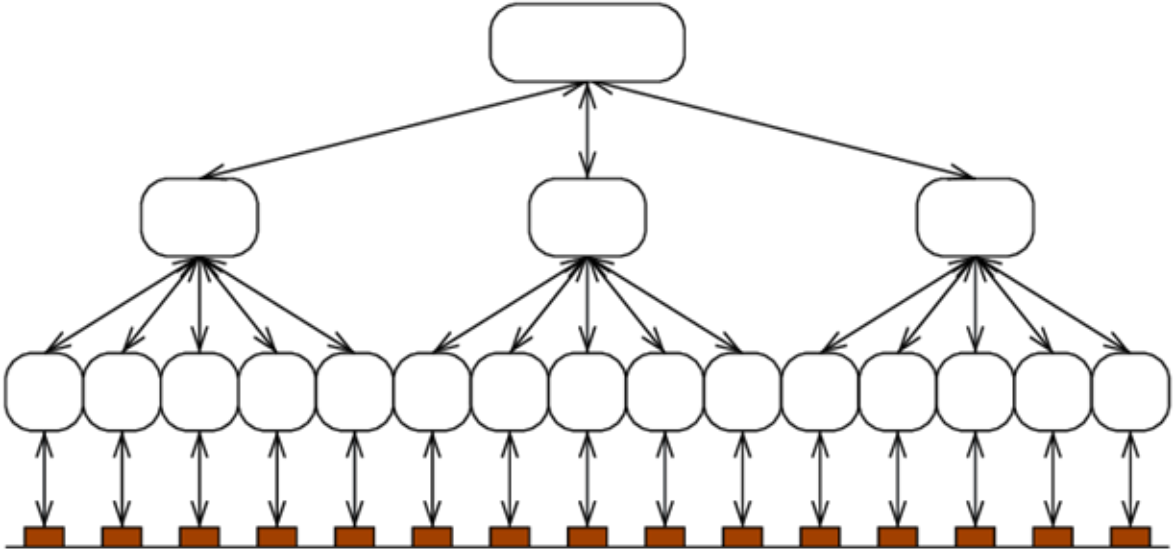


Decentralized communication strategies

Communication Structures: Hierarchical

Hierarchical: organize agents into layered system with levels of authority, responsibility, and abstraction

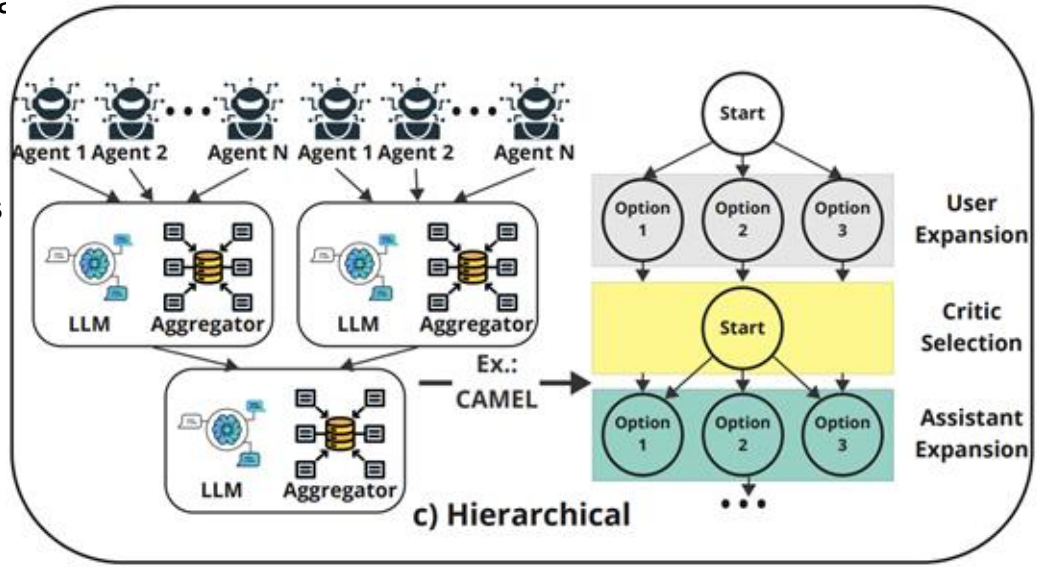
- agents communicate within their level or adjacent levels, ⇒ less communication complexity while preserving coordination
- Top layers are generally planners to lower-level executors



Communication Structures: Hierarchical

Example: CAMEL / Role-Playing Agent System

- Agents assigned roles and authority levels
- top-level agents define task direction and high-level goals
  - e.g., “User agent”: defines objective and constraints
- mid-level agents coordinate interaction and refine outputs
  - e.g., “Assistant agent”: interprets instructions and generates structured responses
- lower-level processes execute subtasks



Communication Structures: Hierarchical

Advantages:

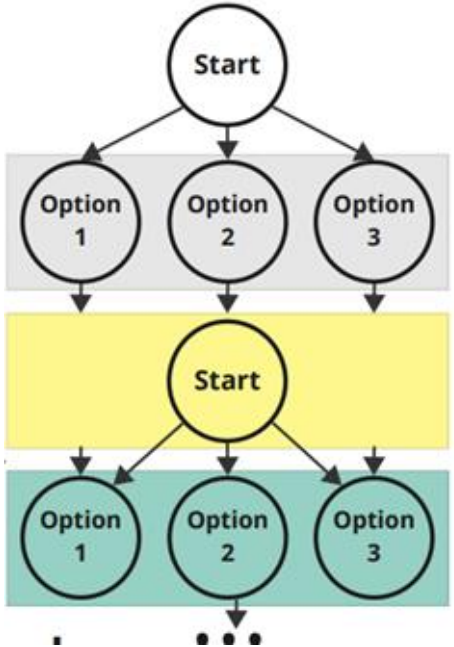
- scalable division of labor through layered decomposition
- reduced communication overhead compared to fully decentralized systems

Disadvantages:

- increased latency due to multi-stage processing
- potential failure at intermediate layers disrupting entire workflow

Key Idea: Communication Structure impacts

- **reasoning quality**: ability of agents to refine and validate outputs through interaction
- **efficiency**: computational and communication cost required to reach solution
- **robustness**: resilience to failures, noise, and inconsistent agent behavior



Orchestration: Motivation

Orchestration: control logic that determines the execution graph of a multi-agent system.

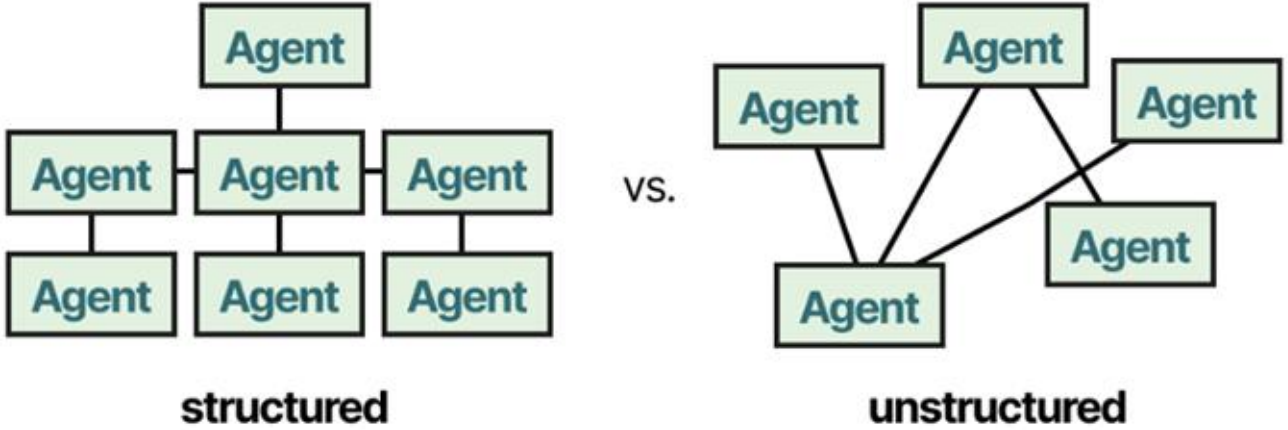
- Scheduling: Who runs and when?
- Routing: What information goes where?
- Control Flow: When does the system stop for a given task?

Naïve approach - unstructured communication: agents freely message eachother

- causes redundant reasoning, cyclic conversations, and uncontrolled token usage

Too much structure ⇒ system becomes rigid and unable to adapt to task-specific needs

Too little structure ⇒ system becomes unstable, inefficient, and difficult to debug



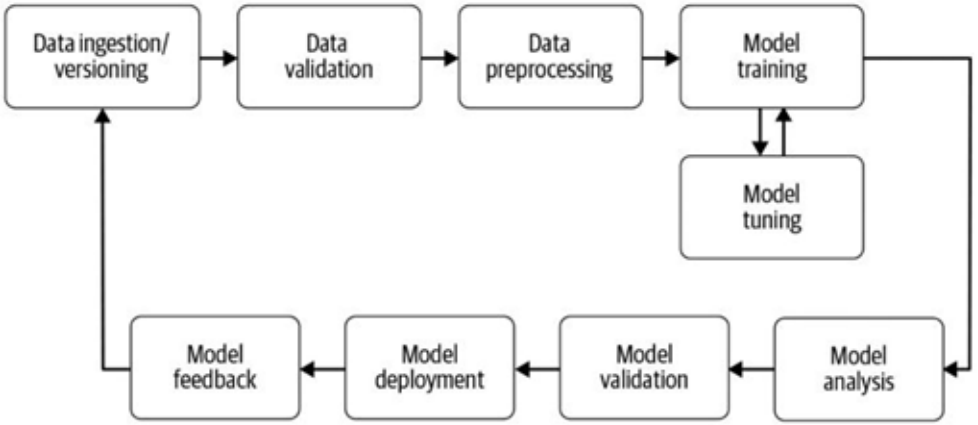
Orchestration: Static

Static orchestration: predefined execution structure across all tasks

- collaboration channels and agent interactions are fixed before runtime
- agent roles, communication paths, and execution order do not change based on input or intermediate outputs

Typical implementation: sequential or pipeline-based workflows

- example pattern: Planner ⇒ Researcher ⇒ Executor ⇒ Verifier
- agents execute in a fixed order, passing outputs downstream without modification of structure



Orchestration: Static - Example

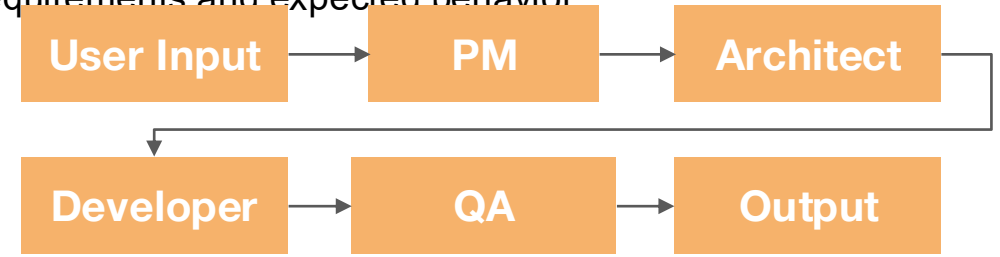
Software Engineering Example

- Product Manager agent: generates requirements from user input
- Architect agent: converts requirements into system design (describes modules, APIs, interfaces)
- Developer agent: implements code from design specification
- QA agent: runs and tests outputs against requirements and expected behavior

communication strictly sequential (agents don't decide)

- output of one agent becomes input to next

global objective decomposed into deterministic subtasks



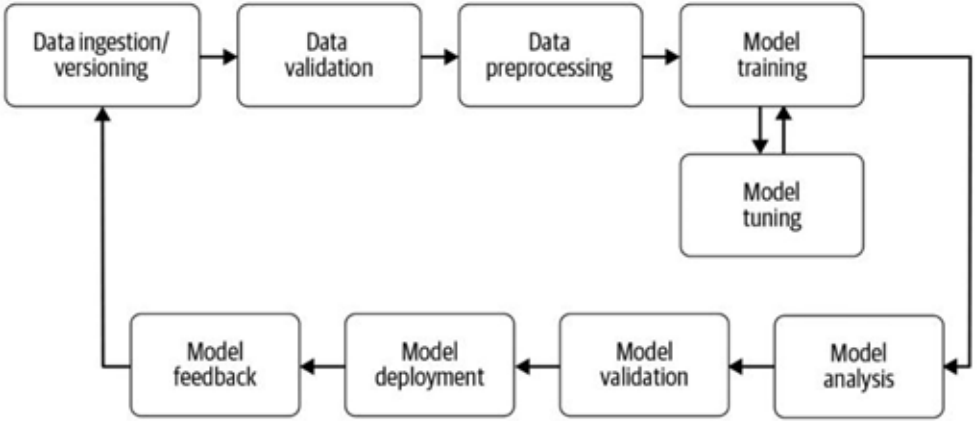
Orchestration: Static

Advantages: predictability and efficiency

- Deterministic: debugging, evaluation, and optimization is easier
- minimal coordination overhead since communication patterns are fixed

Limitations: lack of adaptability and scalability

- cannot dynamically adjust to task complexity, unexpected intermediate outputs, or failure cases
- requires accurate initial design



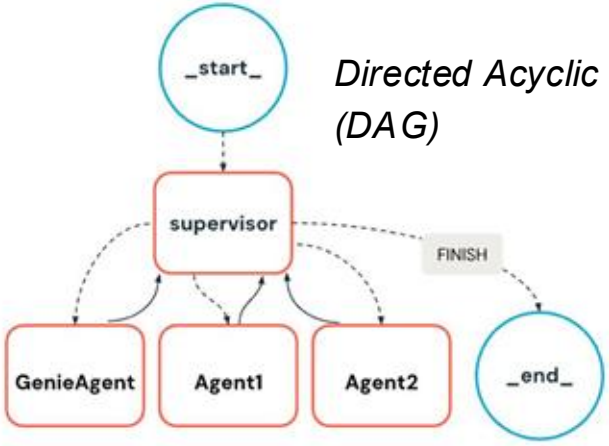
Orchestration: Dynamic

Dynamic orchestration: execution structure adapts at runtime

- agent interactions, roles, and communication channels are determined based on input, intermediate outputs, or system state

Typical implementation: graph-based or planner-driven execution

- system builds a Directed Acyclic Graph (DAG): nodes represent agents and edges represent information flow
- central planner or controller decides:
  - which agent to invoke next
  - what information to pass
  - when to terminate execution



Directed Acyclic Graph (DAG)

Orchestration: Dynamic - Example

Example: Multi-Hop Question Answering

query decomposed into multiple retrieval + reasoning steps

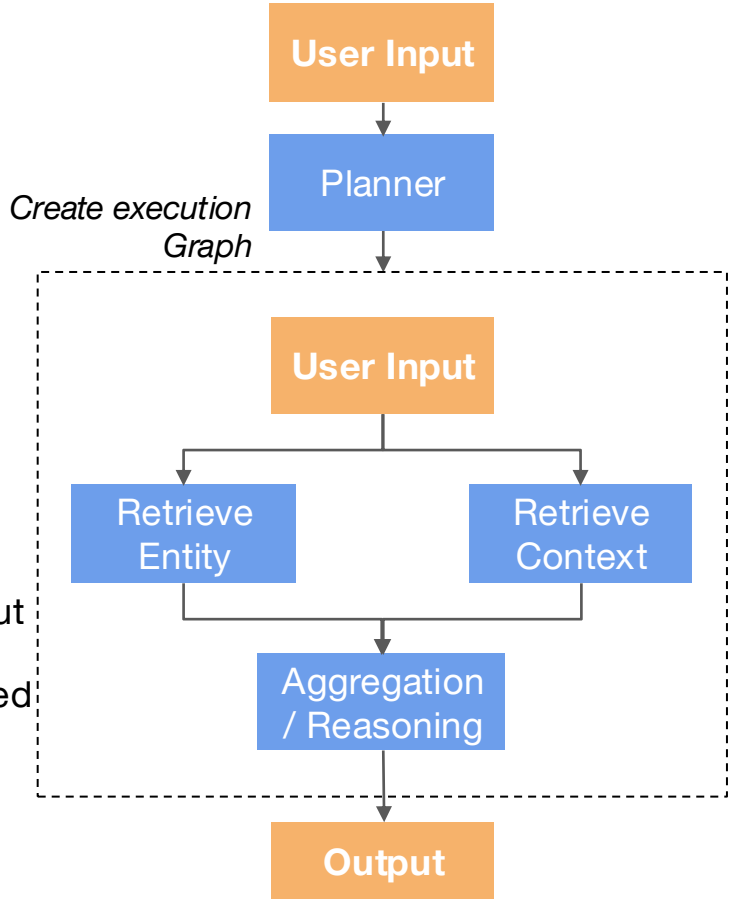
- subtask 1: retrieve entity information
- subtask 2: retrieve related context
- subtask 3: synthesize final answer

independent subtasks executed in parallel where possible

runtime decision making: graph constructed after observing input

dynamic scheduling: tasks triggered when dependencies satisfied

partial re-planning possible if subtask fails



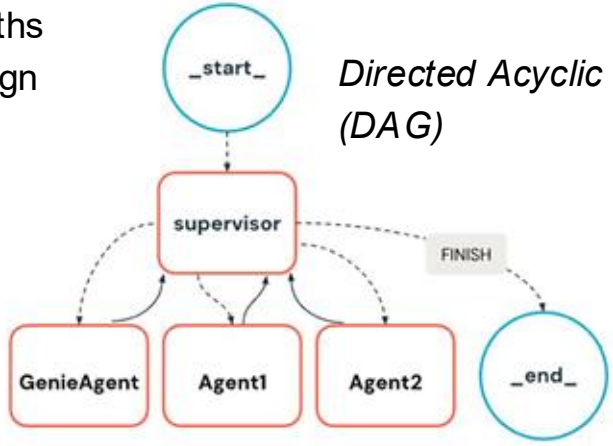
Orchestration: Dynamic

Advantages: flexibility and robustness to complex tasks

- Handles multi-step reasoning, branching logic, and conditional execution paths
- better suited for open-ended tasks such as coding, research, and planning

Limitations: higher complexity and resource cost

- requires real-time decision-making ⇒ increased computational overhead
- harder to debug due to non-deterministic execution paths
- risk of instability if coordination policies are poorly design



Directed Acyclic Graph (DAG)

Orchestration as the Bottleneck

Orchestration, not individual agents, determines system performance

- increasing number or capability of agents does not guarantee better results without effective coordination mechanisms

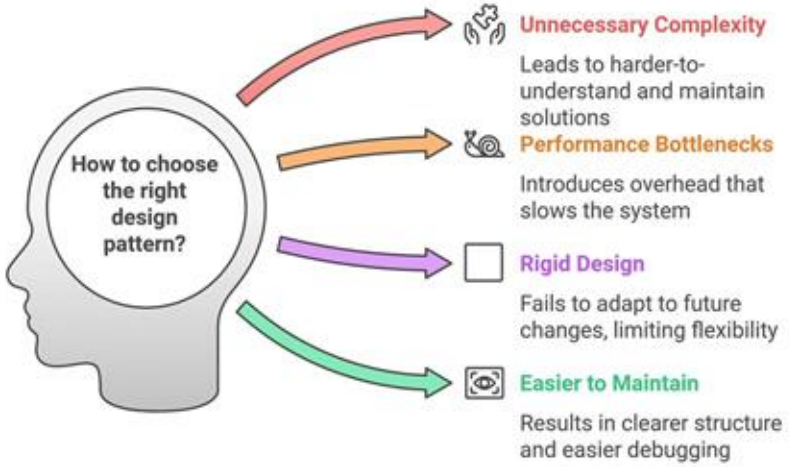
Most failures in MAS arise from orchestration issues i.e

- redundant computation
- conflicting outputs between agents
- failure to converge to a final answer

Tradeoff: control vs adaptability

- static systems ⇒ efficient but brittle
- dynamic systems ⇒ flexible but complex

  ⇒ *practical systems often combine both approaches*



Takeaway: designing collaboration protocols is the central research problem

Multi-Agent Systems: Conclusion

Multi-Agent Systems most effective when task decomposable into distinct subtasks assigned per agent

- Strong performance in domains requiring iterative reasoning, verification, and refinement
- Most useful when diversity of perspectives improves final output quality

Core Challenges

- **Cascading hallucinations:** error propagation across agents in multi-step collaboration pipelines
- **Coordination complexity:** increasing number of agents introduces exponential interaction overhead
- **Unstable emergent behaviors:** agents develop unintended interaction patterns not explicitly designed
    - Goal misalignment: agents may interpret shared objectives differently under local context
- **No formal guarantees:** MAS behavior largely heuristic and empirically driven