Introduction to Agentic AI

-- Multi-Agent Design

Instructor: Guangjing Wang

guangjingwang@usf.edu

Last Lecture

- Search for Planning
  - Monte Carlo Tree Search (MCTS)
- Workflow Design
- Process Formulation
  - Planning Domain Definition Language (PDDL)

This Lecture

- Multi-Agent Design Basics
- Pitfalls in Multi-Agent Design
- Agentic Collaboration
- Multi-Agent Memory

Multi-Agent System (MAS)

- An LLM-based agent is an artificial entity with prompt specifications (initial state), conversation trace (state), and ability to interact with the environments such as tool usage (action).

- A multi-agent system (MAS) is defined as a collection of agents designed to interact through orchestration.

- MAS achieves collective intelligence that surpasses individual capabilities through human-like collaboration and behaviors.

Multi-Agent System (Cont'd)

- MAS organized through carefully designed workflows can achieve superior performance in various domains:
  - Software engineering, scientific discovery.
  - Social simulation and business process automation.

- MAS is structured to coordinate efforts, enabling task decomposition, performance parallelization, context isolation, specialized model ensemble, and diverse reasoning discussions.

- MAS decomposes cognitive loads through a division of labor and collaborative debate approach.

Multi-(LLM) Agent Design

- Early multi-agent systems rely on manual configurations
  - **Agent profiling**: defining each agent's role and capabilities.
  - **Prompt engineering**: agent prompts must be carefully crafted and refined, as agent behavior is highly sensitive to prompt wording.
  - **Inter-agent topology**: the agentic workflow governs the communication structure and interaction topology among agents.
- A change in one often necessitates re-adjustment in the others.
- Evaluation performance beyond accuracy: token cost, calling round, communication overhead…

Automated MAS Design

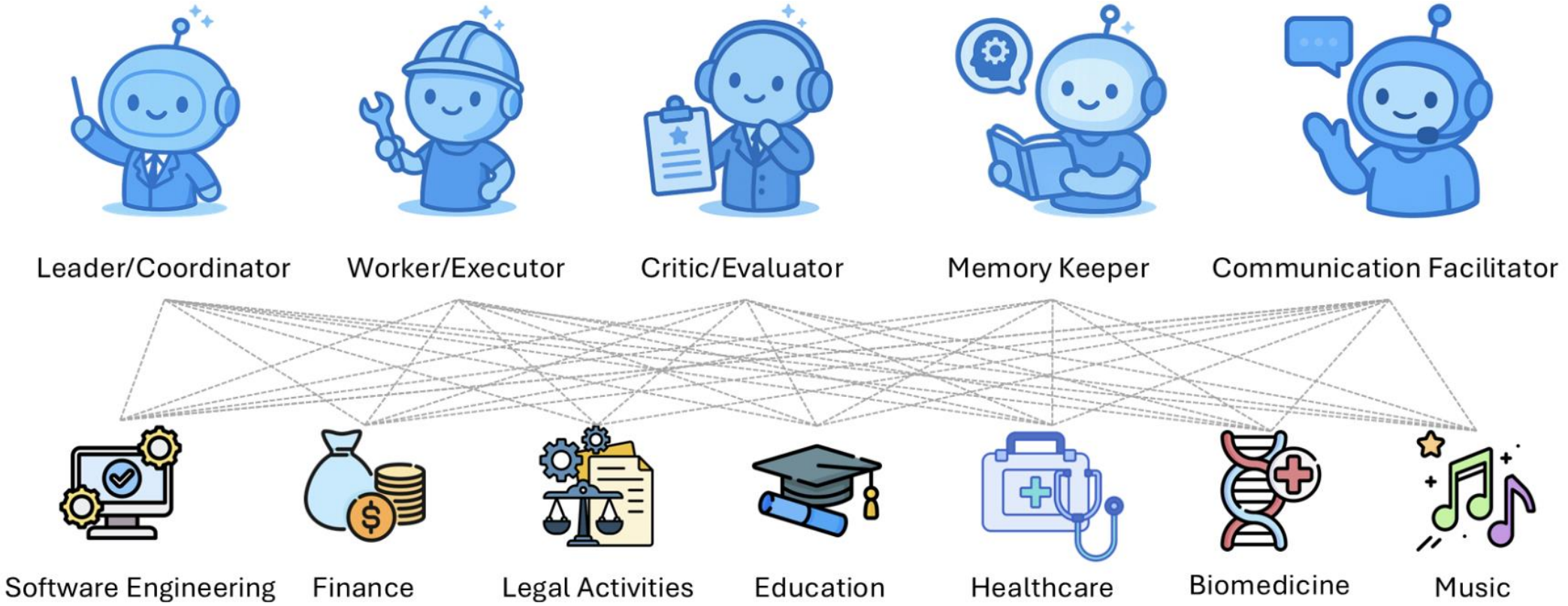
- The automation of multi-agent system design:
  - **Topology optimization**: discovering optimal communication structures between agents.
  - **Prompt optimization**: iteratively refining agent prompts.
  - **Agent profiling**: optimizing inherent agent characteristics.

- Optimizing a multi-agent workflow for a given dataset via different search paradigms, e.g., heuristic search, Monte Carlo tree search, and evolution.

Automated MAS Design Example

1. Let the parent agent create an agent configuration that specifies the metadata of the sub-agents it intends to create.
2. Parse the agent configuration file and create a sub-agent as a Python object and store it in a unified sub-agent pool.
3. Manage agents in the pool, including searching for existing sub-agents, registering new ones, and invoking stored sub-agents.
4. A graph-based memory for tracking and maintaining agent states.
5. During runtime, sub-agents can communicate with each other, and their results can be integrated through the agent ensemble mechanism.

Agent profiling in Multi-Agent Systems



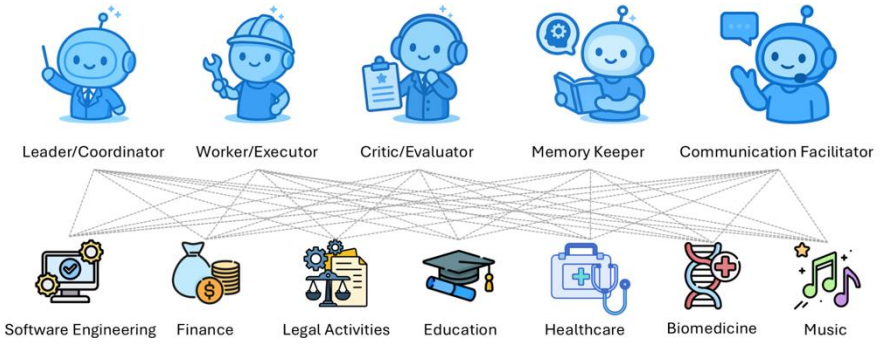
https://arxiv.org/pdf/2601.12538

Roles in Multi-Agent Systems (1)

- Leader: maintaining high-level coherence within the system.
  - This role involves setting global objectives, decomposing tasks into manageable subgoals, and assigning them to appropriate agents.
  - Arbitrating conflicts that emerge between agents with overlapping or contradictory outputs.
- Executor: engaging in concrete actions.
  - This role involves invoking external tools, writing or executing code, retrieving documents, or interfacing with the environment.
- Evaluator: centers on quality assurance.
  - This role includes verifying correctness, testing hypotheses, red-teaming responses, and surfacing potential risks.

Roles in Multi-Agent Systems (2)

- Memory Keeper: maintaining long-term knowledge structures such as episodic logs, semantic embeddings, or knowledge graphs
  - Persistent memory accumulates context, prevents repetitive failures, and enable learning across episodes.
- Communication Facilitator: governing protocols for inter-agent exchange
  - Defining message schemas, managing communication bandwidth, enforcing gating mechanisms, and orchestrating consensus-building.



Multi-Agent in Software Engineering

- Architects define system-level design principles and establish structural blueprints.
- Developers translate these abstractions into concrete implementations.
- Code reviewers and testers safeguard reliability, checking correctness, maintainability, and functional coverage.
- Continuous Integration (CI) orchestrators automate builds, testing, and artifact pipelines, reducing integration frictions.
- Release managers oversee deployment, aligning new versions with milestones and safety protocols.

Multi-Agent in Finance

- Analysts operate at different levels (e.g., fundamental, sentiment, or technical), each extracting distinct signals from raw market.
- Risk Managers then monitor portfolio exposure, apply stress tests, and enforce safeguards to prevent cascading vulnerabilities.
- Traders take responsibility for market interaction and ensure that orders are placed with efficiency under liquidity constraints.
- Compliance roles ensure that activities remain aligned with regulatory requirements, enabling traceable decision-making.

Multi-(LLM) Agent System

https://www.youtube.com/watch?v=sWH0T4Zez6I

Pitfalls in Multi-Agent Design (1)

- Some MAS failures can stem from fundamental limitations of current LLMs, such as hallucination or instruction following.
- Failure patterns where improvements in system design, agent coordination, and verification can offer room to improve the reliability of MAS.

FC1. System Design Issues. Failures originate from system design decisions, and poor or ambiguous prompt specifications.

Insight 1. MAS failure is not merely a function of challenges in the underlying model; a well-designed MAS can result in performance gain when using the same underlying model.

These include failing to follow task requirements (FM-1.1, 11.8%) or agent roles (FM-1.2, 1.5%), step repetitions (FM-1.3, 15.7%), context loss (FM-1.4, 2.80%), or not recognizing task completion (FM-1.5, 12.4%).

Pitfalls in Multi-Agent Design (2)

FC2. Inter-Agent Misalignment. Failures arise from a breakdown in critical information flow from inter-agent interaction and coordination during execution.
Insight 2. Solutions focused on context or communication protocols are often insufficient for FC2 failures, which demand deeper 'social reasoning' abilities from agents.

These include unexpected conversation resets (FM-2.1, 2.20%), proceeding with wrong assumptions instead of seeking clarification (FM-2.2, 6.80%), task derailment (FM-2.3, 7.40%), withholding crucial information (FM-2.4, 0.85%), ignoring other agents' input (FM-2.5, 1.90%), or mismatches between reasoning and action (FM-2.6, 13.2%).

Pitfalls in Multi-Agent Design (3)

FC3. Task Verification. Failures involve inadequate verification processes that fail to detect or correct errors, or premature termination of tasks.

Insight 3. Multi-Level Verification is Needed. Current verifier implementations are often insufficient; sole reliance on final-stage, low-level checks is inadequate.

These include premature termination (FM-3.1, 6.20%), no or incomplete verification (FM-3.2, 8.20%), or incorrect verification (FM-3.3, 9.10%).

More rigorous verification is needed, such as using external knowledge, collecting testing output throughout generation, and multi-level checks for both low-level correctness and high-level objectives.

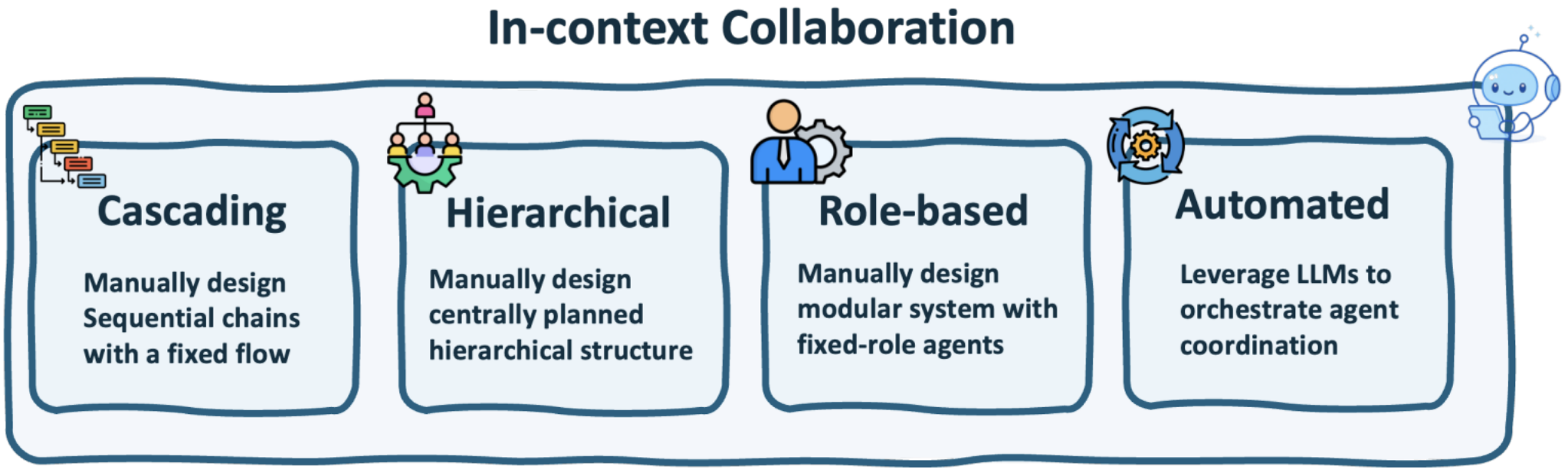
Pitfalls in Multi-Agent Design (4)

- Redundant data transmission when handling interactions among agents, leading to unnecessary network bandwidth and computational overhead.
- The lack of standardized communication mechanisms: session state, user identify, and task history is bundled with every message, consuming extra bandwidth.
- While individual agents can learn from their own interactions and memory, it is hard to effectively share learned experiences across the collective.

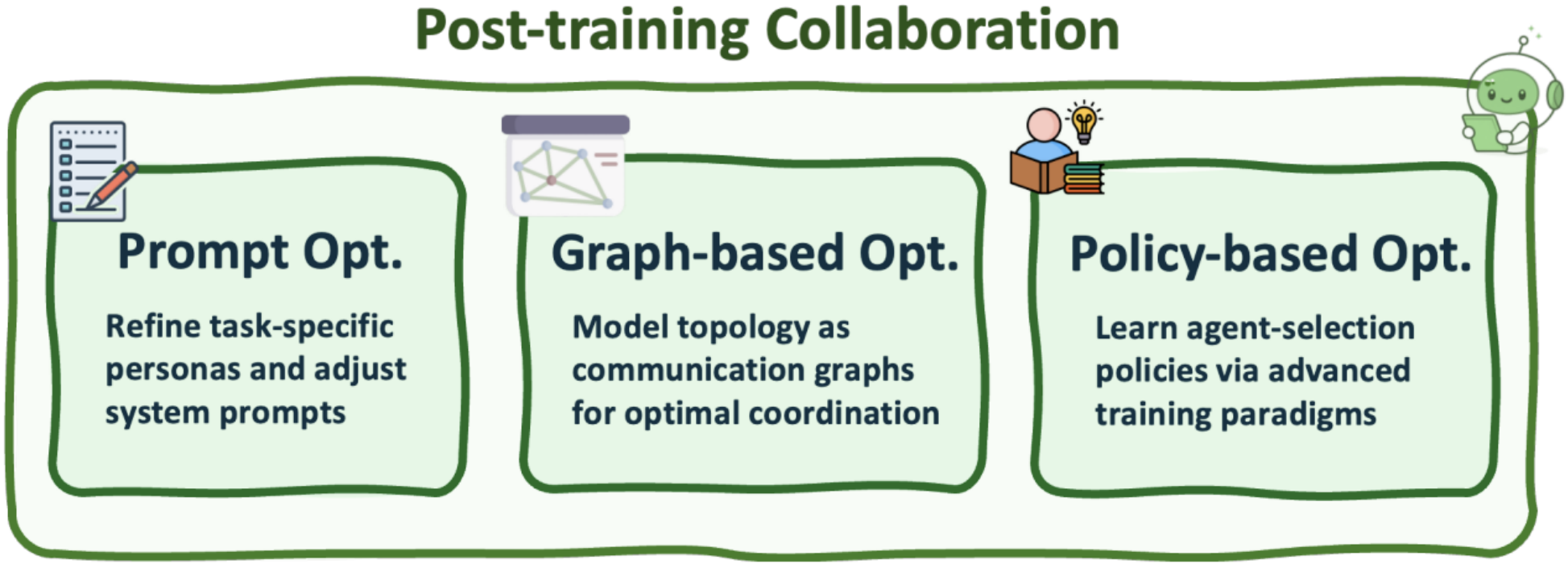
A2A Protocol

https://www.youtube.com/watch?v=Tud9HLTk8hg

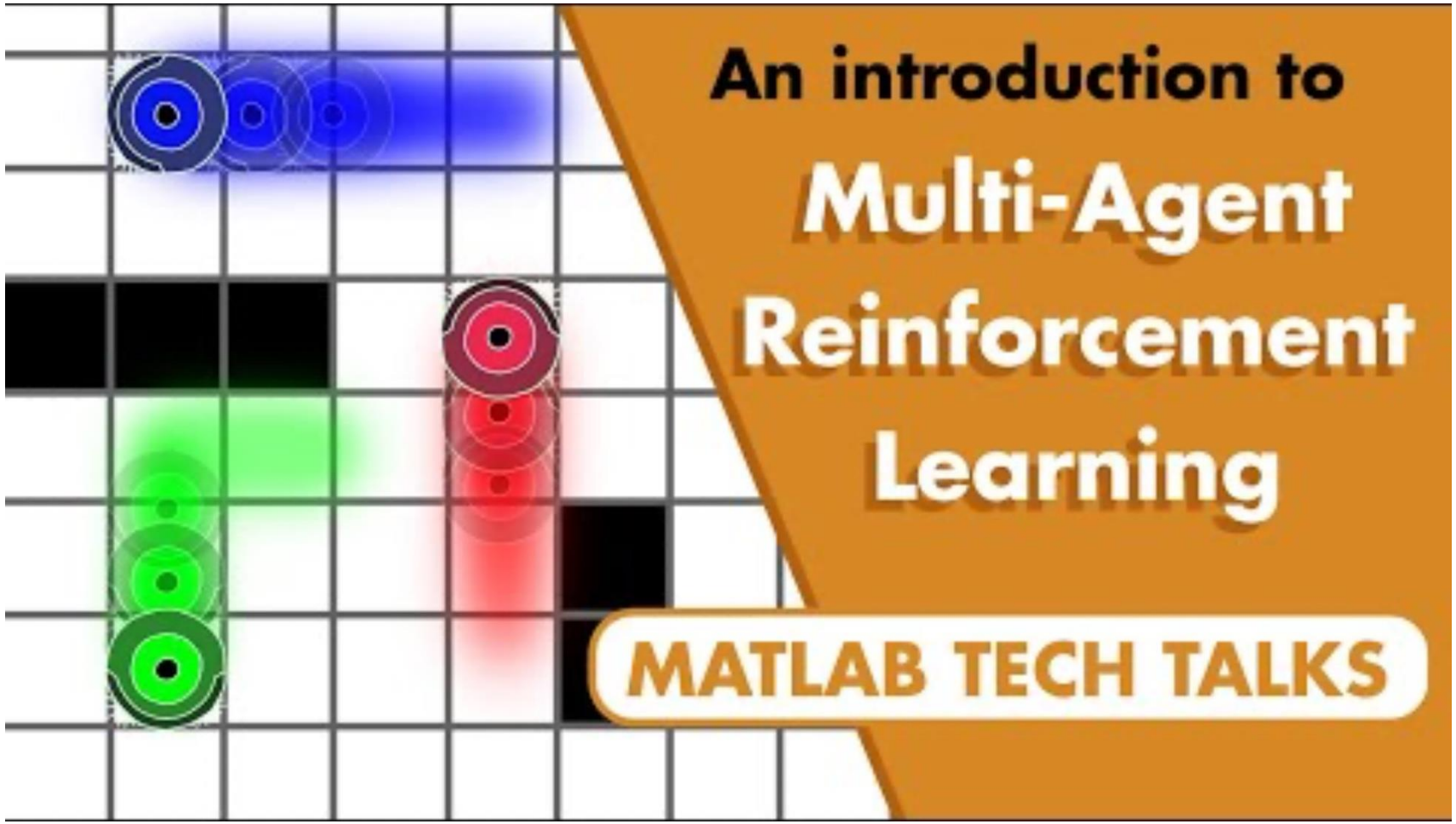
Agentic Collaboration



Agentic Collaboration



Multi-Agent Reinforcement Learning



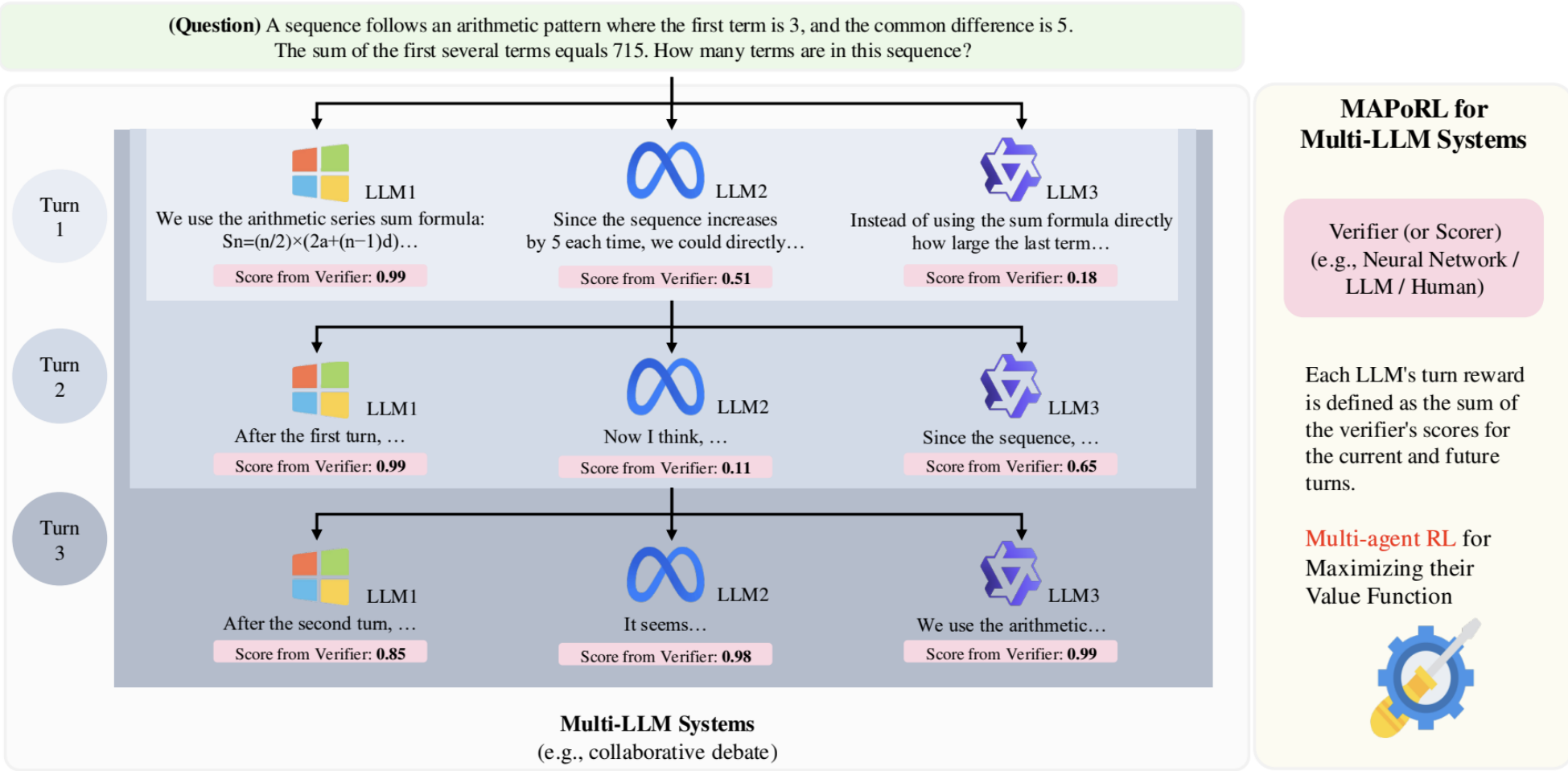
https://www.youtube.com/watch?v=qgb0gyrpiGk

Multi-Agent Post-co-training for collaborative LLMs with Reinforcement Learning (MAPoRL)

- In MAPoRL, **multiple LLMs** first generate their own responses independently and engage in a multi-turn discussion to collaboratively improve the final answer.
- **A MAPoRL verifier** evaluates both the answer and the discussion, by assigning a score that verifies the correctness of the answer, while adding incentives to encourage corrective and persuasive discussions.
- The **score** serves as the co-training reward and is then maximized through multi-agent RL.

https://aclanthology.org/2025.acl-long.1459.pdf

Multi-Agent Post-co-training for collaborative LLMs with Reinforcement Learning (MAPoRL)

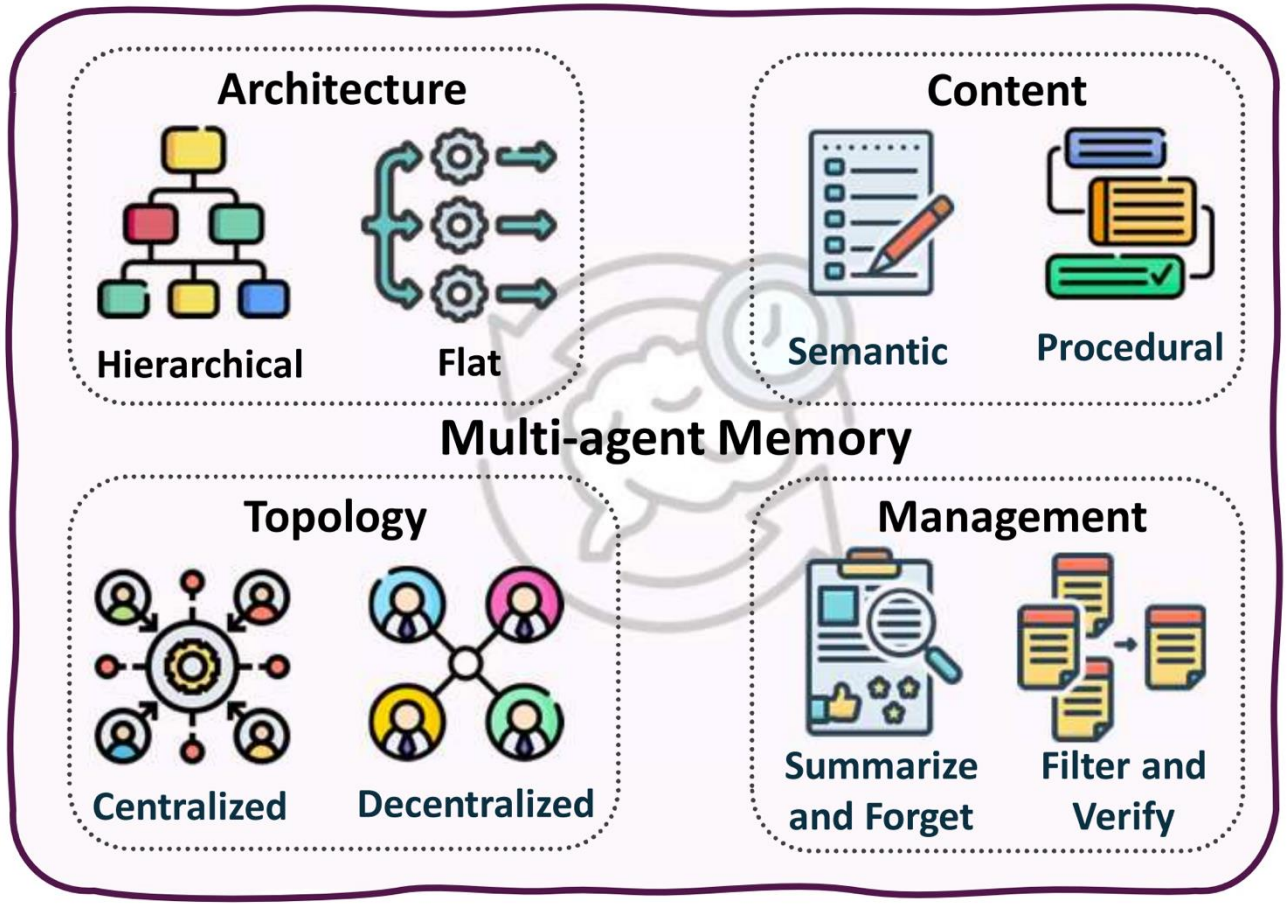


Multi-Agent Post-co-training for collaborative LLMs with Reinforcement Learning (MAPoRL)

- Multi-agent PPO extends PPO to multi-agent settings by training decentralized policies, either with a shared critic or through independent learning.
  - Each agent optimizes its **policy** based on local observations and rewards.
  - Defining the **state** as the concatenation of the multi-agent interaction history, allowing agents to condition their responses on past interactions.
  - **Reward** are aligned but not fully identical across agents, encouraging them to fulfill different roles while working toward solving the task.
  - The reward for each LLM is determined based on scores from a verifier, which may include both current and future pipeline evaluations.
  - Multi-Agent RL is employed to maximize each agent's value function.

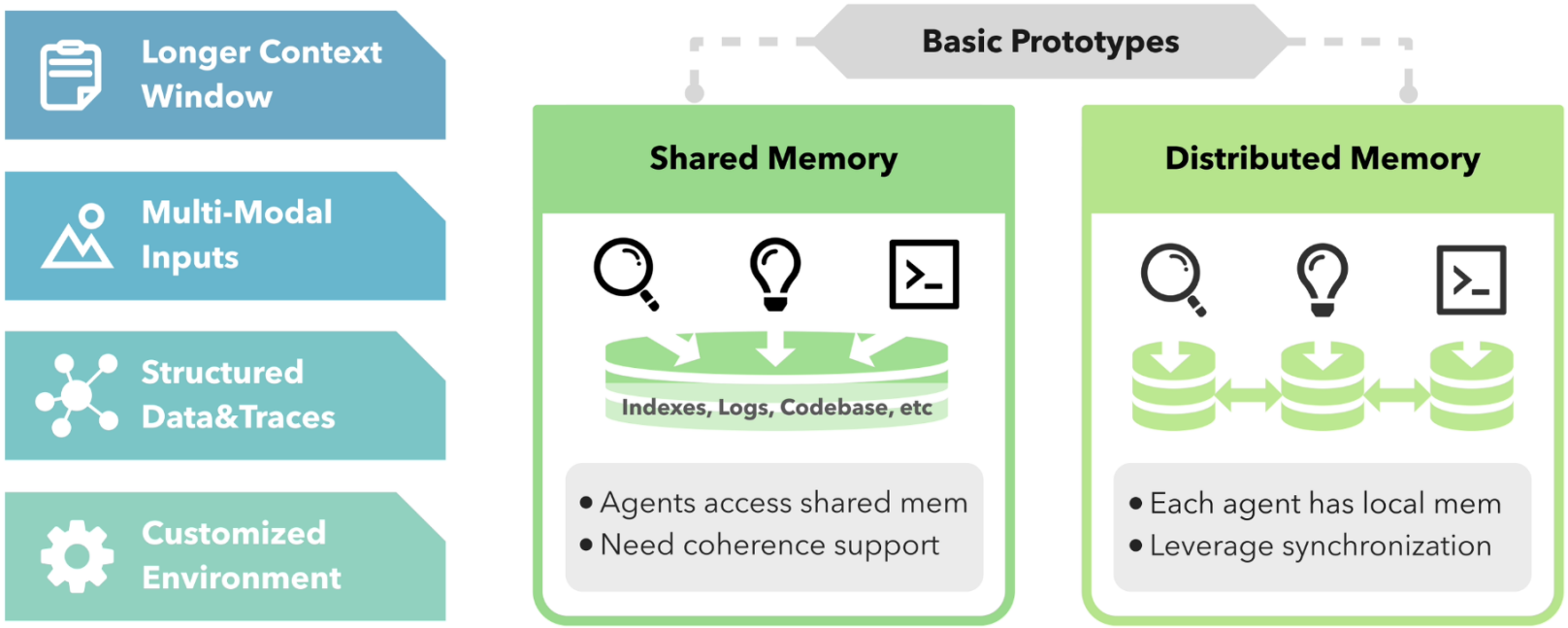
Multi-agent Memory Design

- Architecture, how memory is structured;
- Topology, where it is stored and shared;
- Content, what type of knowledge is stored;
- Management, how it is maintained and updated.



Multi-Agent Memory from a Computer Architecture Perspective (1)

- Complex Context: longer histories, multiple modalities, structured traces, and customized environments.



https://arxiv.org/abs/2603.10062

Multi-Agent Memory from a Computer Architecture Perspective (2)

- Shared memory makes knowledge reuse easy but requires coherence support.
  - Without coordination, agents overwrite each other, read stale information, or rely on inconsistent versions of shared facts.

- Distributed memory improves isolation and scalability but requires explicit synchronization.
  - State divergence becomes common unless carefully managed.
  - Most real systems sit between these extremes: local working memory with selectively shared artifacts.

Multi-Agent Memory Hierarchy

Agent I/O layer:

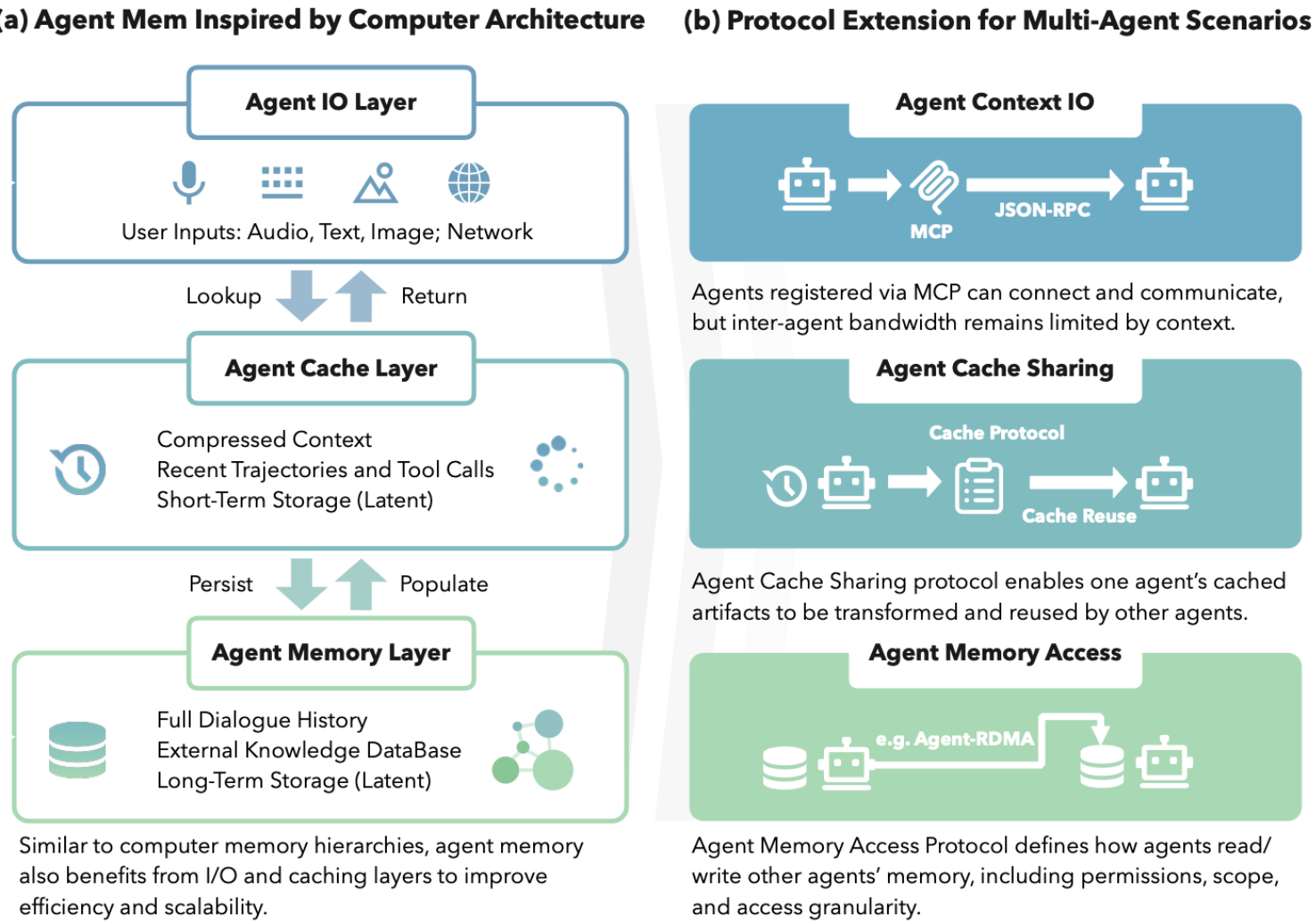
- Interfaces that ingest and emit information

Agent cache layer:

- Fast, limited-capacity memory for immediate reasoning

Agent memory layer:

- large-capacity, slower memory optimized for retrieval and persistence



Multi-agent Memory Consistency

- Multi-agent memory consistency has two requirements:
  - Read-time conflict handling under iterative revisions, where records evolve across versions and stale artifacts may remain visible.
  - Update-time visibility and ordering that determines when an agent's writes become observable to others and how concurrent writes may be observed in a permissible order.

- How to better handle multi-agent memory consistency is a key research problem.

References

- Agentic Reasoning for Large Language Models https://arxiv.org/abs/2601.12538
- Multi-Agent Post-Co-Training for Collaborative Large Language Models with Reinforcement Learning https://aclanthology.org/2025.acl-long.1459.pdf
- Why Do Multi-Agent LLM Systems Fail? https://arxiv.org/abs/2503.13657
- Multi-Agent Memory from a Computer Architecture Perspective: Visions and Challenges Ahead https://arxiv.org/abs/2603.10062