

Multi-agent Environment

Contents

1. Recap: LLM-based Agents
2. Multi-agent Environment
3. LLM-based Agents Collaborate
4. Business Applications
5. References

Recap: LLM-based Agents

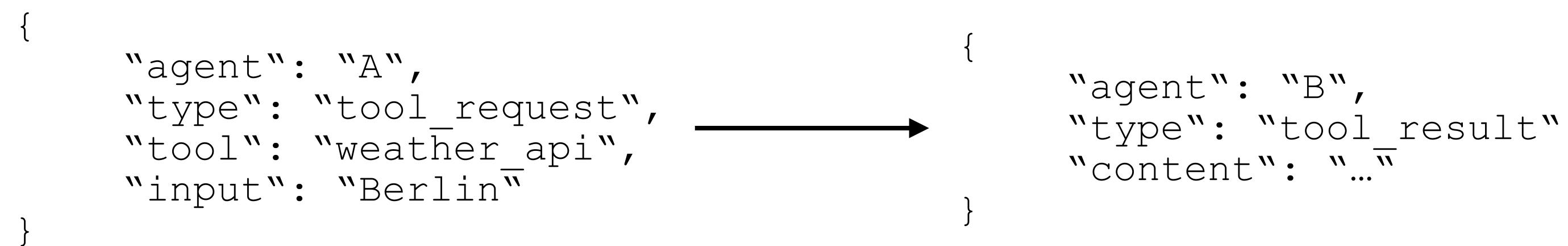
- LLM-based agent = an LLM paired with a structured loop for perception, planning, and action execution [1]
- Operates via an **Observe–Plan–Act** architecture: observes input, plans next steps, performs actions (e.g., tool use, response)
- Planning follows a **Chain-of-Thought** (CoT) paradigm: the LLM reasons step by step before producing actions or outputs [1]
- Can interact with environments and invoke external tools / APIs for information or task execution, which enables access to real-time or private knowledge [1]
- Often uses memory modules (short-term or long-term) to maintain state across turns [1]
- Capable of task decomposition, reflection, and adaptive behavior in dynamic tasks [1]
- LLM agent = base unit for more complex multi-agent systems [1]

Multi-agent Environment

- A **LaMA** (LLM-based Multi-Agent system) = a group of LLM agents working collaboratively on complex tasks; they communicate, share state, and distribute roles to solve problems that are too large for a single agent
- Agents may have a specialization, and work can be parallel or sequential
- Each agent operates independently using CoT-based reasoning and tool access; different agents may have access to different tools to support specialization [1]
- Coordination is handled via *communication protocols* and system-level orchestration [1], and the system can operate with or without human
- Widely adopted workflow [1]:
 - Planning: agents propose decompositions, strategies, task distribution
 - Voting: agents assess proposals and a strategy is chosen
 - Execution: selected agent(s) perform actions
 - Credit: performance is evaluated, and the agents are rewarded with credits
 - In more sophisticated systems, the agents can vote for intermediate results and adjust the strategy on the fly

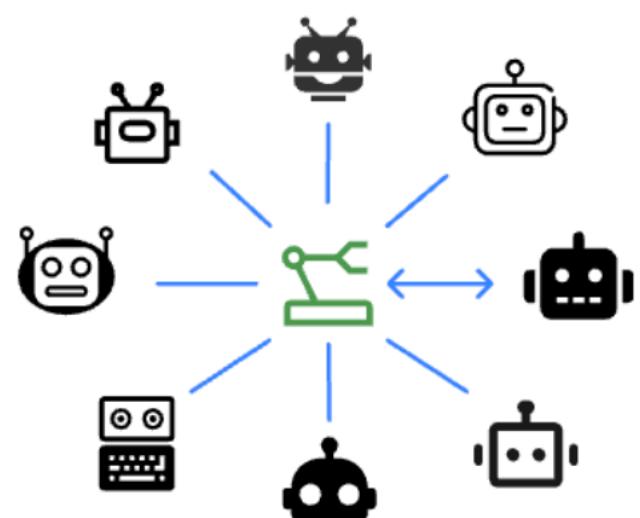
Multi-agent Environment

- *Communication protocols* define how agents exchange information – both in format and behavior; protocols are a kind of a script for conversation – if one agent asks, others know how to respond [1]
- A protocol includes:
 - The structure of messages (who speaks, when, and in what format)
 - The rules for interpreting and responding to them
- Technically, this can be implemented as:
 - JSON-style message schemas e.g. with fields like sender, task, content (cf. structured output)
 - Agent policies or functions that handle specific message types, i.e. behavioral rules
- Protocols ensure agent coordination, compatibility of response formats => system robustness [1]

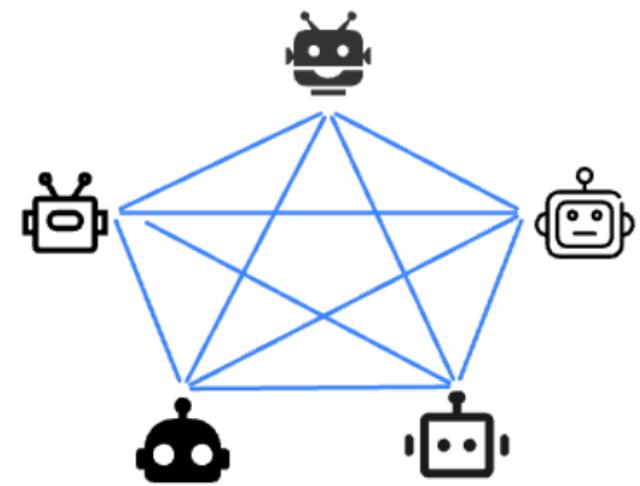


Multi-agent Environment

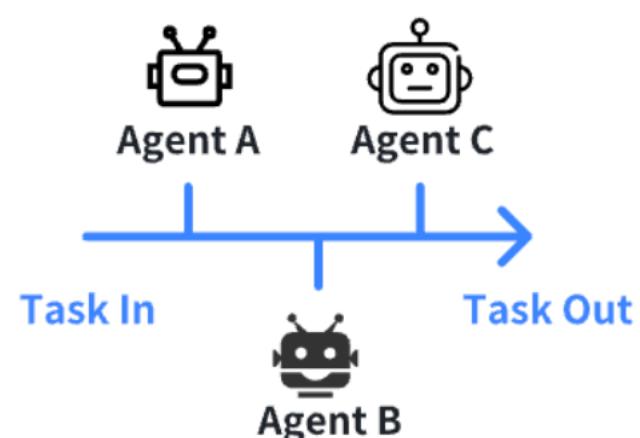
- Coordination design of LaMAs governs how the agents are connected, who coordinates them, and how they share responsibilities; system design should balance autonomy, control, and information access [1]



- Star** (centralized): one coordination agent supervises the work; after each step, the result is returned to it for it to require the next action [1]
- Best for tasks where a strict and consistent supervision is required (customer support, assistance on the websites etc.)



- Graph** (decentralized): a (fully) interconnected network of agents => full autonomy and peer-to-peer communication [1]
- Used for creative open-ended tasks (e.g. brainstorming)

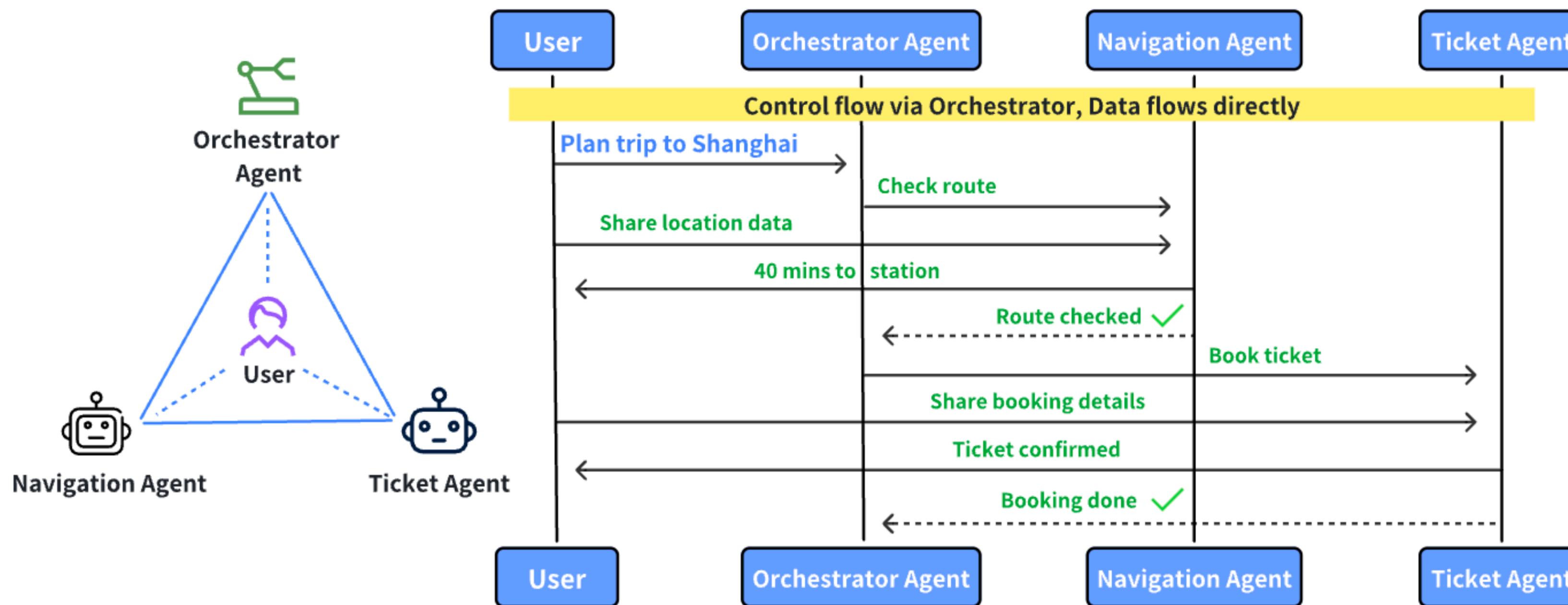


- Bus** (semi-centralized): agents have a shared communication channel called *bus* (kind of a group chat); one agent „posts“ a task, others pick it up and reply with results, votes, or new questions; the coordination succeeds by triggers (so each agent knows when it should step in) [1]
- Balances between full centralization and full decentralization

- In real complex applications, a hybrid approach is often used

Multi-agent Environment

- Moreover, LaMA architecture is also responsible for data privacy as it configures what access to user data the agents have: e.g. only planner sees confidential client info and executors get abstract tasks [1]
- In the example above [1], the coordinator in opposite has no access to user data and sees only current status of the task completion



Multi-agent Environment

- To dynamically adjust the system to maximize the performance, a credit allocation system may be adopted [1]
- Each agent maintains a “profile” representing its performance history across tasks
- After task completion (also possible on intermediate steps), agents receive “credit” based on their contribution
- Credit is tracked per skill or task domain: summarizing, planning, retrieval etc.
- Over time, agents become specialized: agents evolve to be strong at particular roles [1]
- Credit profiles can guide:
 - Agent selection: e.g. an agent that consistently earns high credit in planning is more likely to be selected again for planning
 - Voting: credits can be used as weights in collective decision-making
 - Experience sharing: outputs of a high-credit agent can become teaching material
 - A simple form of self-organization: the system “learns” who is good at what

LLM-based Agents Collaborate

- In [2], agents were placed in a virtual town called “Smallville”; each agent had memory and preferences, and could plan and replan its future actions and interact with other agents via natural language
- 25 agents simulated daily life in an open-ended setup, i.e. they had no particular goal; no script for coordination was given — only individual memory and planning [2]
- Emergent „social“ behavior was observed: agents began forming social groups, planning events, and inviting others — all autonomously; they started forming opinions and impressions of each other [2]
 - As a substrate for communication, the authors seeded two events: agent Isabella wanted to throw a party, and agent Sam wanted to propose his candidacy for the mayor elections; no other agent except Isabella / Sam knew about their plans; the simulation ran for two days
 - The number of agents who knew about Sam’s plans increased from one (4%) to eight (32%), and about Isabella’s party — from one (4%) to thirteen (52%), all without any user intervention [2]
 - 5 out of 12 agents who were invited to Isabella’s party actually showed up [2]
 - The network density (custom formula from 0 to 1) grew from 0.167 to 0.74 [2]

LLM-based Agents Collaborate

- Paper [3] explores how multiple LLM agents can improve reasoning and factual accuracy via open-ended debates.
- Each agent first generates its own answer independently; then, in 2–4 rounds, agents read each other's answers, critique them, and revise their own
- Debate structure: Propose → Read & Reflect → Revise → Consensus
- Debate improved not only accuracy but also confidence: agents dropped hallucinated facts after hearing others' doubts
- Emergent behaviors appeared that were not scripted: error correction, consensus building, and self-correction
- Results: on several datasets, the system significantly improved accuracy for math problems, determining the optimal next move in chess, QA, bibliographical factuality (compared to a single agent)
 - GSM8K (school math) accuracy increased from 77% to 85%
 - MMLU factual QA increased from 63.9% to 71.1%
 - Biographical fact accuracy increased from 66.0% to 73.8%

Business Applications

- LinkedIn launched a Hiring Assistant: a collaborative AI system embedded in Recruiter and LinkedIn Jobs; the assistant helps write job descriptions, discover candidates, and draft personalized messages — reducing effort and improving speed
- Features:
 - End-to-end hiring workflows: from job creation to candidate outreach
 - Integrates live context: job type, past hiring behavior, market data etc.
 - Recruiters can ask follow-ups or revise outputs iteratively
- Utilizes a centralized architecture:
 - Conversation Coordinator Agent routes requests to specialized agents
 - Job Post Agent drafts descriptions using templates + reasoning
 - Retrieval Agent queries LinkedIn's talent graph for best-fit candidates
 - Messaging Agent generates custom messages based on candidate background and role
- Tech stack: Microsoft Azure, LangGraph, LinkedIn internal embeddings and API tools

Business Applications

- Uber addresses the problem of testing its enormous (100M+ lines) multilingual codebase with AutoCover: a LaMA that automates unit test generation and validation
- The system mimics human heuristics to improve efficiency and coverage
- Features:
 - Built as an IDE plugin integrated into developer workflows
 - Supports streaming test generation and validation while keeping the developer in the loop
- Apart from LaMA for auto testing, Uber has an LLM-based copilot and a tool to migrate from Java to Kotlin
- Built with LangGraph

Business Applications

- Elastic built Attack Discovery — a tool integrated in their security analytics platform that helps with identifying and analyzing threats
- Enables a single analyst to conduct investigations that previously required entire teams, significantly reducing the time to detect and respond to threats
- Features:
 - Alert Prioritization: assesses alerts considering factors like severity, risk scores, and asset criticality to surface the most significant threats
 - Orchestrates LLM-based agents to interpret and contextualize alert data, providing actionable insights for analysts
 - Includes RAG to provide the system access to the most relevant and up-to-date information, enhancing the accuracy of threat assessments
 - Supports various LLMs, including OpenAI's GPT-4, Anthropic's Claude, and Google's Gemini, giving organizations the flexibility to choose models that best fit their needs
- Built with LangGraph

Business Applications

- Replit introduced [Replit Agent](#), an AI-powered assistant designed to help users build applications from scratch using natural language prompts; the agent (with LaMA under the hood) assists in coding, setting up environments, and deploying applications, streamlining the development process
- Features:
 - Natural Language Interface: the agent translates plain English into functioning code
 - Human-in-the-Loop: the verifier agent ensures continuous user engagement, promoting collaborative development
 - Prompt Engineering: utilizes few-shot examples, dynamic prompt construction, and structured formatting (e.g., XML tags) to enhance model performance
 - Instead of traditional API function calls, Replit Agent generates code to perform tasks, leveraging the Replit environment's capabilities (tools are built on the fly)
- Centralized architecture:
 - Manager Agent oversees the workflow and coordinates tasks
 - Editor Agents handles specific coding tasks
 - Verifier Agent checks code quality and interacts with the user for feedback
- Built with LangGraph

References

- [1] [LLM-based Multi-Agent Systems: Techniques and Business Perspectives](#), Shanghai Jiao Tong University & OPPO Research Institute
- [2] [Generative Agents: Interactive Simulacra of Human Behavior](#), Stanford, Google Research & DeepMind
- [3] [Improving Factuality and Reasoning in Language Models through Multiagent Debate](#), MIT & Google Brain
- [4] [Exploring Collaboration Mechanisms for LLM Agents: A Social Psychology View](#), Zhejiang University, National University of Singapore & DeepMind
- [5] [AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation](#), Microsoft Research et al.
- [6] [How real-world businesses are transforming with AI – with more than 140 new stories](#), Microsoft (blog post)
- [7] [Built with LangGraph](#), LangGraph (website page)
- [8] [Plan-Then-Execute: An Empirical Study of User Trust and Team Performance When Using LLM Agents As A Daily Assistant](#), Delft University of Technology & The University of Queensland