Decentralized LLM-based Multi-Agent Framework with DAG on the Internet Computer Protocol (ICP)

By Helion AI Lab

Contents

1. Abstract

This whitepaper delves into the design of multi-agent workflows utilizing language models within diverse and Directed Acyclic Graphs (DAG) on the Internet Computer Protocol blockchain. The document explores the benefits of multi-agent designs, the impact of structured versus unstructured loops, and the challenges posed by unstructured loops in multi-agent workflows. Techniques for optimizing performance, efficiency, and reliability in multi-agent workflows using third-party libraries like LangGraph are examined. The whitepaper provides insights into creating structured frameworks, defining clear communication protocols, implementing automation tools, and customizing agents to enhance collaboration and task execution. By leveraging LangChain's LangGraph library, organizations can optimize multi-agent workflows, improve task management, and achieve efficient performance within diverse and Directed Acyclic Graphs (DAG). The paper concludes by summarizing the key findings of the field of LLM-based multi-agent systems and their potential impact on Web3 applications. The whitepaper aims to provide a comprehensive overview of the current state of research in this area, highlighting the potential for further advancements in multi-agent workflows utilizing language models in diverse and Directed Acyclic Graphs (DAG).

2. Introduction

The development of multi-agent workflows utilizing language models (LLMs) has gained significant traction in recent years due to the remarkable capabilities of LLMs in various tasks. This whitepaper aims to provide an in-depth exploration of the creation of multi-agent workflows, focusing on the integration of LLMs in diverse and structured loops to enhance collaboration and task performance.

The development of Large Language Models like GPT-3, GPT-4, Claude, Gemini, Mistral, and Grok and their successors represents a significant milestone in the field of artificial intelligence. These models have been trained on vast datasets, enabling them to understand and generate human-like text with a high degree of coherence and relevance. Their capabilities span a wide range of applications, from generating creative content to providing detailed technical assistance, making them valuable components in automated systems and workflows. Multi-agent workflows are a crucial aspect of building complex applications, especially in the context of Web3 applications on the Internet Computer Protocol (ICP) blockchain. These workflows involve the collaboration of multiple agents, each with its own set of tools, prompts, and custom code, to achieve a common goal. In this whitepaper, we will explore the creation of multi-agent workflows utilizing language models in diverse and structured loops.

3. Problem with LLM's

LLMs are prone to generating inaccurate and nonsensical information, which can be a significant issue in various applications. This can be mitigated by employing controlled generation techniques, such as offering specific details and constraints for the input prompt.

LLMs are not reliable and consistent enough to ensure that the model output will be right or as expected every time. This can lead to inconsistent results, causing problems for users and customers. When building an application based on LLMs, prompt injection is a potential issue. Users may force the LLMs to produce unexpected output, which can be a security concern. Ensuring the accuracy and reliability of AI-generated content is crucial, as hallucinations are a real problem when working with LLMs, and inaccuracies can lead to misinformation.

There are several open challenges in LLM research, including reducing and measuring hallucinations, optimizing context length and context construction, incorporating other data modalities, making LLMs faster and cheaper, designing new model architectures, developing GPU alternatives, making agents usable, improving learning from human preference, and improving the efficiency of the chat interface.

The performance of LLMs is heavily dependent on the quality, diversity, and representativeness of the training data. Biases and inaccuracies in the data can be inadvertently learned and perpetuated by the model, leading to biased outputs and potential disparities in diagnoses.

4. The Future is Agentic

In a captivating talk, AI pioneer Dr. Andrew Ning, a leading mind in the field and co-founder of Coursera, explores the transformative potential of AI agents, envisioning a future where these agents play a central role in advancing artificial intelligence. This summary presents key insights from his presentation, emphasizing the significance of agentic workflows over traditional non-agentic interactions with AI models. Unlike traditional prompt-response models, agentic workflows involve iteration, where AI agents, endowed with diverse skills and tools, collaborate and refine outcomes through continuous feedback loops. By employing agents with specialized roles (e.g., writer, reviewer, fact-checker), the collective effort leads to superior results, mirroring the human approach of planning, iteration, and improvement. A case study involving a coding benchmark illustrated how GPT-3.5, when incorporated into an agentic workflow, outperformed even GPT-4 in certain tasks. This highlights the efficiency of agents in leveraging existing models to achieve remarkable results. Ning suggests that embracing agentic workflows could accelerate our progress toward artificial general intelligence (AGI) by expanding the range of tasks AI can perform, promoting innovation in AI application development. The talk underlines the necessity for AI researchers and developers to adopt agentic workflows, emphasizing planning, tool integration, and the orchestration of multi-agent systems for enhanced AI performance.

5. Why Internet Computer Protocol

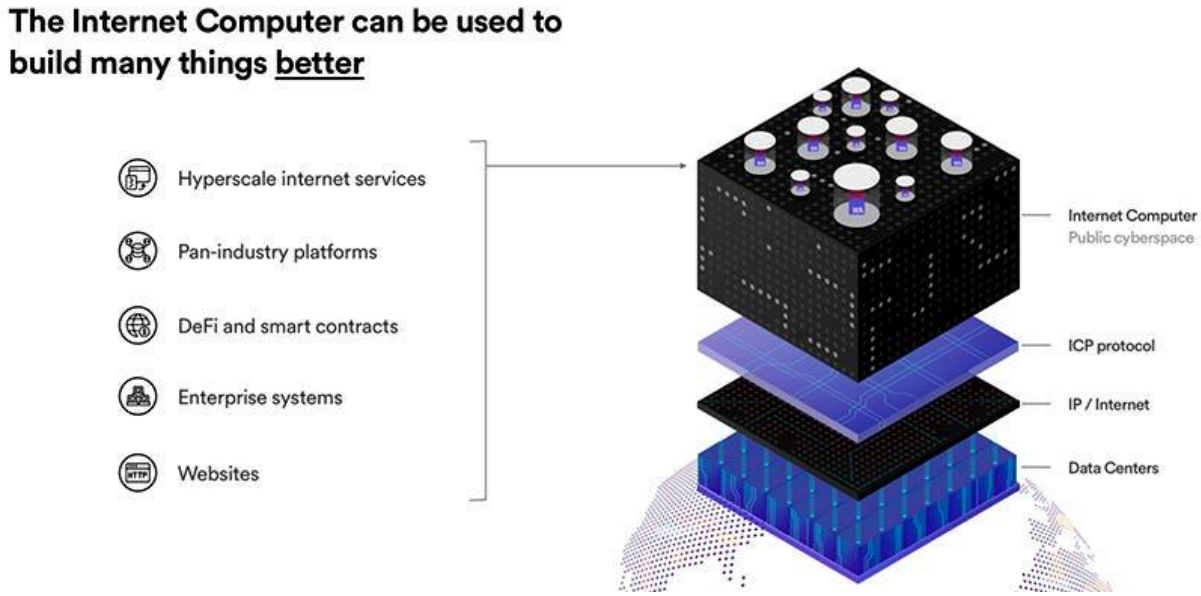
Building on the Internet Computer Protocol (ICP) offers several advantages and unique features that make it an attractive choice for developers and organizations looking to create 100% decentralized applications.

Some reasons to build on the Internet Computer Protocol include:

1) Decentralized Cloud Computing: ICP bridges the gap between traditional programming and blockchain-based development, offering a decentralized cloud infrastructure that enables developers to build and host applications without relying on centralized servers. This decentralized approach enhances security, resilience, and censorship resistance.
2) Smart Contract Capabilities: Smart contracts on ICP are expressive and scalable like traditional applications but benefit from the trust-less and decentralized execution of a blockchain. This allows developers to create sophisticated applications with the security and transparency of blockchain technology.
3) Scalability and Decentralization: ICP is designed to achieve a practical balance between scalability and decentralization. By sharding smart contracts over multiple instances called subnets, ICP can handle a high volume of transactions in parallel while maintaining decentralization and security.
4) Interoperability with Bitcoin: ICP seamlessly integrates with the Bitcoin blockchain, allowing smart contracts to function as decentralized users with their own Bitcoin addresses. This integration opens new possibilities for applications that leverage the power of Bitcoin while benefiting from a decentralized infrastructure layer.
5) Community and Ecosystem: The ICP community is vibrant and actively supports the growth and development of the protocol. With a thriving ecosystem of developers, projects, and venture studios, building on ICP provides access to a supportive community and a wide range of resources.
6) Innovative Features: ICP offers innovative features such as threshold signatures, deterministic decentralization, and advanced cryptography for

secure key management. These features enhance the security, efficiency, and flexibility of applications built on the protocol.

Overall, building on the Internet Computer Protocol provides developers with a robust and scalable platform for creating decentralized applications, leveraging the benefits of blockchain technology while offering unique features tailored for Web3 applications.



6. LLM-based Multi-Agent Systems and Web3 Applications

LLM-based multi-agent systems are poised to have a significant impact on Web3 applications by revolutionizing decentralized systems through enhanced interactions, decision-making, and task execution. These systems leverage Large Language Models (LLMs) to enable agents to communicate, reason, and collaborate effectively, leading to emergent collaborative behaviors and high-order Theory of Mind capabilities. In the context of Web3 applications, LLM-based multi-agent systems offer the following potential impacts:

1) Sophisticated Interactions: LLM-based multi-agent systems can facilitate sophisticated interactions within Web3 applications, enabling agents to communicate, reason, and collaborate effectively. This can lead to more intelligent and dynamic interactions within decentralized systems.
2) Complex Problem-Solving: By leveraging the diverse capabilities of individual agents within a multi-agent system, LLM-based systems can tackle complex tasks through collaboration. This can enhance problem-solving capabilities and enable the system to address intricate challenges in Web3 applications.
3) Optimized Task Execution: These systems can optimize task execution by efficiently allocating tasks among multiple agents, fostering robust reasoning through iterative debates, managing complex context information, and enhancing memory management to support intricate interactions within the system.
4) On-Chain Decision-Making: LLM-based multi-agent systems can enable decentralized decision-making processes within Web3 applications, allowing agents to autonomously collaborate, reason, and execute tasks based on shared objectives and contextual information.
5) Efficient Resource Utilization: By enhancing collaboration and coordination among agents, LLM-based multi-agent systems can improve resource utilization within Web3 applications, leading to more efficient and effective task execution.

7. Understanding Multi-Agent Workflows

Multi-agent workflows involve multiple independent agents collaborating to achieve a common goal. Each agent possesses its own prompt, LLM, tools, and custom code, enabling specialized contributions to the workflow. The key considerations in multi-agent workflows are identifying the independent agents and establishing connections between them. This approach aligns well with a graph representation, where each agent is a node connected by edges, managing control flow and communication within the system.

8. Leveraging Language Models in Multi-Agent Workflow

LLMs play a pivotal role in multi-agent workflows by providing advanced language processing capabilities. By integrating LLMs into diverse and structured loops, agents can leverage the reasoning and planning abilities of these models to enhance decision-making, interactions, and task performance within the workflow.

9. LangGraph vs DAG vs Agents

LangGraph is a module built on top of LangChain to improve the creation of cyclical graphs, often needed for agent runtimes. One of the value props of LangChain is the ability to create custom chains. These chains can be referred to as Directed Acyclic Graphs (DAG).

A Directed Acyclic Graph (DAG) is a directed graph with no directed cycles. Which consists of vertices and edges, with each edge directed from one vertex to another, such that following those directions will never form a closed loop.

Agents are a type of application running an LLM in a loop that helps create applications that are flexible and thus can accomplish tasks that are not predefined. The main interface to LangGraph functionality is stateGraph, nodes, and edges. After we define our graph, we can compile it into a runnable that can also be referred to as a chain.

10. Structured Loops vs Unstructured Loops

Structured loops in multi-agent workflows differ from unstructured loops in their organization and coordination of agents. In structured loops, workflows are organized as graphs with nodes representing agents and edges symbolizing communication channels between them, allowing for a systematic and coordinated execution of tasks. This structured approach enables the definition of interactions and dependencies between agents, leading to more efficient task execution and collaboration within the workflow. On the other hand, unstructured loops lack this organized framework, resulting in a less systematic and coordinated workflow where agents may not have clearly defined roles or communication channels, potentially leading to inefficiencies and difficulties in task management and collaboration.

11. Problem with Multi-Agent with Unstructured Loops

In multi-agent workflows, examples of unstructured loops include scenarios where there is incoherence from unstructured conversations, unproductive dialog loops, misdirected engagements, disorganized coordination, opacity into teamwork, and passive learning. Unstructured loops lack the organized framework found in structured loops, leading to issues such as ambiguities in conversations, lack of focus in dialogues, irrelevant engagements, unclear coordination, limited visibility into teamwork, and passive observation without active learning mechanisms. These unstructured loops highlight the challenges faced in multi-agent workflows when coordination and structure are lacking, impacting the efficiency and effectiveness of task execution and collaboration.

11.1 Scalability

Unstructured loops in multi-agent workflows can significantly impact scalability by introducing inefficiencies, ambiguities, and challenges in task management and collaboration. The lack of organization and structure in unstructured loops can lead to incoherent conversations, unproductive dialog loops, misdirected engagements, and disorganized coordination. These issues hinder the smooth flow of information and tasks between agents, making it difficult to scale the workflow effectively as the number of agents or complexity of tasks increases. In essence, unstructured loops can impede the scalability of multi-agent workflows by reducing efficiency, increasing errors, and complicating coordination among agents.

11.2 Reliability

Unstructured loops in multi-agent workflows can impact the reliability of the system by introducing inefficiencies, ambiguities, and challenges in task management and collaboration. The lack of organization and structure in unstructured loops can lead to incoherent conversations, unproductive dialog loops, misdirected engagements, and disorganized coordination. These issues

can result in errors, delays, and misunderstandings within the workflow, affecting the overall reliability of the multi-agent system. In essence, unstructured loops hinder the smooth flow of information and tasks between agents, potentially compromising the reliability of the workflow.

11.3 Performance

Unstructured loops in multi-agent workflows can negatively impact performance by introducing inefficiencies, errors, and challenges in task management and collaboration. The lack of organization and structure in unstructured loops can lead to incoherent conversations, unproductive dialog loops, misdirected engagements, and disorganized coordination. These issues can result in decreased productivity, delays in task completion, and difficulties in achieving optimal performance within the multi-agent system. In essence, unstructured loops hinder the smooth flow of information and tasks between agents, ultimately affecting the overall performance and efficiency of the workflow.

11.4 Efficiency

Unstructured loops in multi-agent workflows can significantly impact efficiency by introducing inefficiencies, errors, and challenges in task management and collaboration. The lack of organization and structure in unstructured loops can lead to incoherent conversations, unproductive dialog loops, misdirected engagements, and disorganized coordination. These issues can result in decreased productivity, delays in task completion, and difficulties in achieving optimal performance within the multi-agent system. In essence, unstructured loops hinder the smooth flow of information and tasks between agents, ultimately affecting the overall efficiency of the workflow.

12. Helion Application

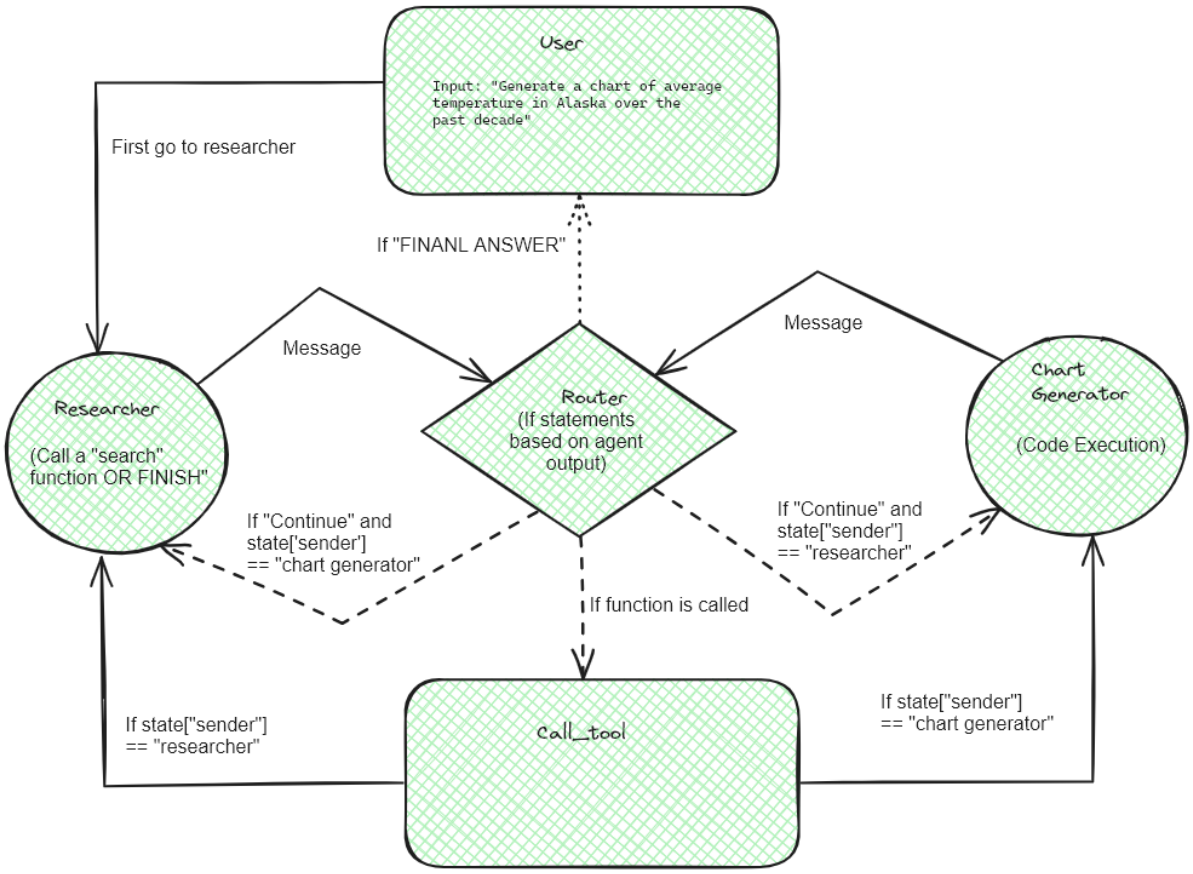
The benefits of using multi-agent workflows in structured loops include enhanced collaboration, improved task focus, and development flexibility. Multi-agent workflows allow for specialized contributions from different agents, leading to more accurate responses tailored to specific tasks. By breaking down complex tasks into manageable units entrusted to specialized agents, multi-agent workflows enhance task performance and efficiency. Additionally, the Directed Acyclic Graphs (DAG) in multi-agent workflows facilitate the evaluation and improvement of each agent independently without disrupting the larger application, providing flexibility in development and optimization.

A key benefit of using LangGraph for designing multi-agent systems is to enable the definition of agent-like workflows as graphs, allowing the dynamic creation of structured loops and cycles. It's conceptualized as a tool for defining state machines with labelled directed graphs, where nodes represent states or agents, and edges represent transitions or communications between agents.

In summary, integrating LLM-based multi-agent systems with Directed Acyclic Graphs (DAG) on a decentralized network like the Internet Computer Protocol can significantly enhance collaboration, decision-making, and task execution within Web3 applications. This approach leverages the strengths of both LLM-based systems and decentralized networks to create a robust and efficient framework for building intelligent and decentralized applications.

12.1 Implementation of Multi-Agent Framework with Directed Acyclic Graphs (DAG)

1) Collaboration: A multi-agent collaboration workflow is detailed, showcasing how multiple agents can work on the same state or message. This collaboration is facilitated by LangGraph's ability to track state over time, add nodes (agents), and define edges (communications) between nodes. This describes a system where different agents collaborate by sharing a scratchpad, allowing them to see each other's work. This transparency can be both beneficial for tracking the process and cumbersome due to excess information, sometimes making it unnecessary to share every detail except the outcome. These agents operate as part of a single large language model (LLM) call, utilizing specific prompt templates to structure inputs and responses according to predefined system messages.



The flow of operations between these agents is managed by a rule-based router, which determines the next step based on the output of each LLM call. If a tool is activated, the router directs the operation to that tool. If the LLM indicates a "FINAL ANSWER" without activating a tool, the process concludes, and the response is delivered to the user. Otherwise, the router transitions to another LLM for further processing, indicating a simple but effective system for managing task transitions based on predefined rules.

To encapsulate the process described above in a mathematical equation or algorithm, we can consider the following notation:

Let $\mathbf{LLM}(x)$ represent the call to the Large Language Model with input x, where x is the input prompt or question.

Let $\mathbf{Tool}(y)$ represent the call to an external tool with input y, determined by the output of $\mathbf{LLM}(x)$.

The router's decision-making process, $\mathbf{Router}\ (Output)$, can be defined as:

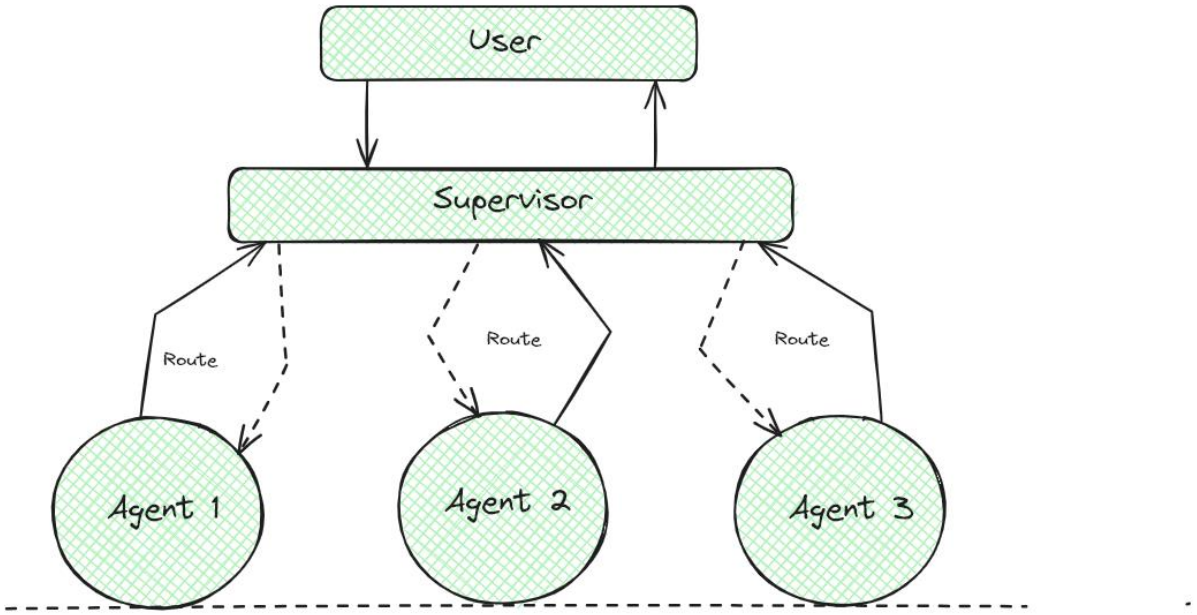
Consider the process initiated by an input x to a Large Language Model (LLM), denoted as $\mathbf{LLM}(x)$. The LLM processes this input and produces an output. The subsequent action taken by the system is determined by a router function, $\mathbf{Router}\ (Output)$, which evaluates the LLM's output. The decision logic of the router can be represented as follows:

$$Router(Output) = \begin{cases} Tool(y), & \text{if LLM output invokes a tool} \\ Return\ to\ User, & \text{if LLM output is "FINAL ANSWER"} \\ LLM(x'), & \text{otherwise} \end{cases}$$

Where:

- $Output$ represents the response generated by the LLM given the input x.
- $Tool(y)$ represents the invocation of an external tool, decided based on the content of Output.
- $Return\ to\ User$ signifies the process ends, and the output is presented to the user as the final answer.
- $LLM(x')$ represents another call to the LLM, with x' being a new input or a continuation of the discussion, necessitated when the output does not directly lead to a tool invocation or a conclusive answer.

2) Agent Supervisor: Another variant involves a supervisor managing independent agents, each operating in its loop with the supervisor only seeing the final output. This setup emphasizes a hierarchical structure where the supervisor routes tasks to different agents, illustrating a less collaborative but more controlled workflow. In this setup, multiple agents operate independently without a shared scratchpad, unlike the previously described collaborative model. Each agent has its private scratchpad for intermediate work, and only their final responses are compiled into a global scratchpad.



These agents are identified as LangChain agents, indicating they each utilize their specific prompts, LLM calls, and tools through a mechanism called the agent executor, rather than relying on a singular LLM call. An agent supervisor oversees the process, directing tasks to the appropriate individual agents. This supervisor acts as a meta-agent, with its "tools" being the ability to utilize other agents, orchestrating their contributions towards a cohesive output.

To represent the described system mathematically, let's define a few components and then formulate the process involving multiple independent agents and a supervisor.

Let $Agent_i(x)$ denote the i^{th} independent LangChain agent being executed with input x, where x is the specific prompt or input for that agent. This agent has its own LLM and possibly other tools it might call upon, encapsulated within what's referred to as an $AgentExecutor$.

Let $GlobalScratchpad$ be the shared global scratchpad where final responses from all agents are compiled.

The $AgentSupervisor$ oversees the distribution of tasks to the appropriate agents and compiles their final outputs into the $GlobalScratchpad$.

Given n independent agents, the process can be described as follows:

Agent Execution: For each agent $Agent_i$, with $i=1,2,...,n$, the process $Agent_i(x_i)$ is executed independently, where x_i is the input specific to $Agent_i$.

Final Response Compilation: The final output of each $Agent_i$, denoted as $Response_i$, is appended to the $GlobalScratchpad$.

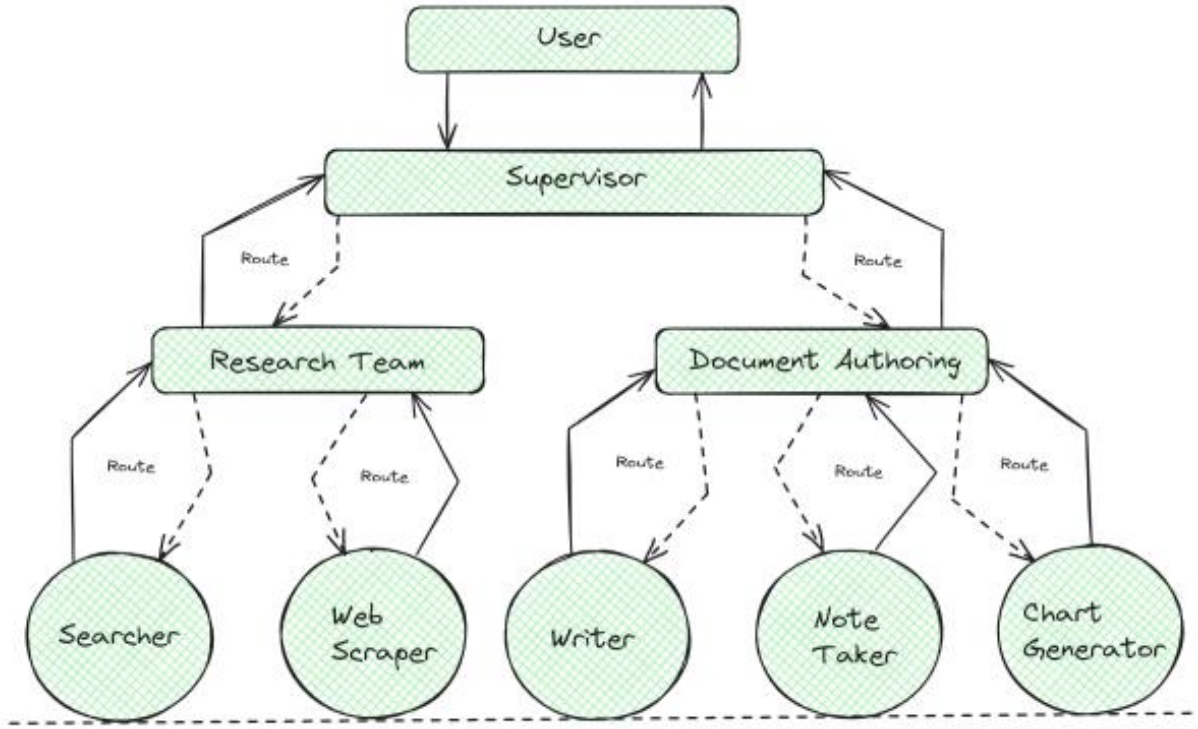
Global Compilation Process: This can be mathematically represented as $GlobalScratchpad = \bigoplus_{i=1}^{n} Response_i$

Where $\bigoplus$ symbolizes the operation of appending each agent's final response to the $GlobalScratchpad$.

Agent Supervisor's Role: Conceptually, the role of the $AgentSupervisor$ in directing inputs and compiling outputs can be represented as a function that maps inputs to the respective agents and then combines their outputs:

$$AgentSupervisor(\{x_1, x_2, \ldots, x_n\}) = GlobalScratchpad$$

3) Hierarchical Agent Teams: We further explore hierarchical agent teams, where each agent node is a supervisor for sub-agents, creating a layered approach to managing complex tasks. This setup allows for detailed task division and specialization among agents. In this enhanced setup, the agents within the nodes are now other LangGraph objects, offering greater flexibility compared to using the LangChain agent executor for agent runtime. This arrangement is termed "hierarchical teams" because the subagents, functioning as teams within the broader system, provide a layered structure of operation and decision-making.
The interconnected web of LangGraph agents operates under the oversight of a supervisor agent, who orchestrates their activities and ensures coherence and collaboration across the system.



This hierarchical approach allows for a more dynamic and scalable system, where complex tasks can be distributed and managed across different levels of agents. To mathematically represent the hierarchical team structure with LangGraph agents and a supervisor, we conceptualize the structure as follows:

Let $LangGraphAgent_i$ denote the i^{th} LangGraph agent, which itself can be a complex object capable of performing tasks independently or acting as a supervisor to further subagents.

Let $SubAgents_{i,j}$ represent the j^{th} subagent under the ith LangGraph agent, indicating the hierarchical nature of this system. Each $LangGraphAgent_i$ can have multiple $SubAgents_{i,j}$, demonstrating the team structure within each agent. The interaction and oversight of these agents by a supervisor can be formulated as:

Execution by LangGraph Agents and Their Subagents: For each LangGraph agent $LangGraphAgent_i$ and its subagents $SubAgents_{i,j}$, the process can be represented as $Process_{i,j}(x)$, where x is the specific task or input.

Hierarchical Team Operation: The operation within each LangGraph agent, considering its subagents, can be aggregated as:

$$LangGraphAgent_i(Operation) = \bigoplus_j Process_{i,j}(x)$$

Where $\bigoplus$ symbolizes the aggregation of operations or responses by the subagents within the LangGraph agent.

Supervisor Agent Coordination: The supervisor agent, which coordinates the overall process and integrates the responses from various LangGraph agents, can be represented as:

$$SupervisorAgent(\{LangGraphAgent_1, LangGraphAgent_2, \ldots, LangGraphAgent_n\}) = GlobalResponse$$

Where $GlobalResponse$ is the compiled output from all the LangGraph agents (and their subagents) coordinated by the supervisor.

12.2 Smart Contract LLM

The integration of Large Language Model (LLM) technology into on-chain smart contracts presents a significant advancement in the field of blockchain development, offering new possibilities for automating and enhancing the functionality of multi-agent workflow. Smart Contract LLM will enable individuals and enterprises to build autonomous agents and deploy on-chain products in a decentralized ecosystem. We aim to create an open on-chain multi-agent Economy, where developers can delegate Web3 tasks and product development to agents. The use of LLM in smart contracts opens possibilities for creating a more complex and sophisticated run-time engine that can handle a wide range of tasks autonomously. As LLM technology continues to evolve, smart contracts are expected to become increasingly capable of executing complex transactions and enforcing contractual agreements without the need for human intervention, thereby reducing execution and enforcement costs in the contracting process. In summary, the integration of LLM technology into on-chain smart contracts represents a significant step towards automating and decision-making process in the Web3 ecosystem. This advancement holds the potential to revolutionize the way smart contracts are created, executed, and managed, paving the way for more efficient and sophisticated blockchain applications.

12.3 Multi-Agent Marketplace

The features of Helion LLM-based multi-agent marketplace ecosystem include:

1) Agent Economy Manifestation: The LLM-based multi-agent marketplace enables the manifestation of an Agent Economy in the Web3 ecosystem. This marketplace allows individuals and enterprises to build autonomous agents that automate on-chain projects and tasks, fostering a collaborative environment where agents can interact and work together towards common goals.
2) Natural Language to Motoko Code Conversion: One key feature is the ability to convert natural language into Motoko code. This functionality empowers users to express their Web3 product ideas in plain language, which is then translated into executable code for smart contracts, arbitrage agents, NFTs, rollups, and more. This conversion process simplifies the development and deployment of on-chain products.

3) Agentic Experience: The LLM-based multi-agent marketplace offers an agentic experience where users can communicate their project requirements to the system, and the system autonomously generates and deploys the necessary code on-chain. This streamlined process accelerates development efforts and optimizes resource utilization by allowing users to focus on high-level project goals while the system handles the technical implementation.
4) On-Chain Agent Deployment: Users can deploy agents directly on-chain, enabling the agents to autonomously interact with the deployment infrastructure. These agents possess their own identity and can independently figure out ways to engage with the on-chain environment, enhancing their adaptability and operational efficiency. This direct on-chain deployment feature streamlines the execution of tasks and interactions within the marketplace.
5) Specialized Agents and Expertise: The marketplace offers specialized agents tailored for specific tasks and operations. Users can choose from a variety of agents, each proficient in distinct functions, such as Code Forker assistants for code customization. By selecting agents based on their expertise and trustworthiness, users can optimize their project development and execution processes within the marketplace.
6) Optimized Motoko Code: The LLM-based multi-agent marketplace produces gas-optimized Motoko code for various prompts, ensuring efficient execution of smart contracts and on-chain tasks.
This optimization enhances the performance and cost-effectiveness of deploying agents and executing transactions within the digital economy, promoting scalability and sustainability.

These features collectively define the functionality and capabilities of an LLM-based multi-agency marketplace in the digital economy, offering a user-friendly and efficient environment for developing, deploying, and managing on-chain projects and tasks.

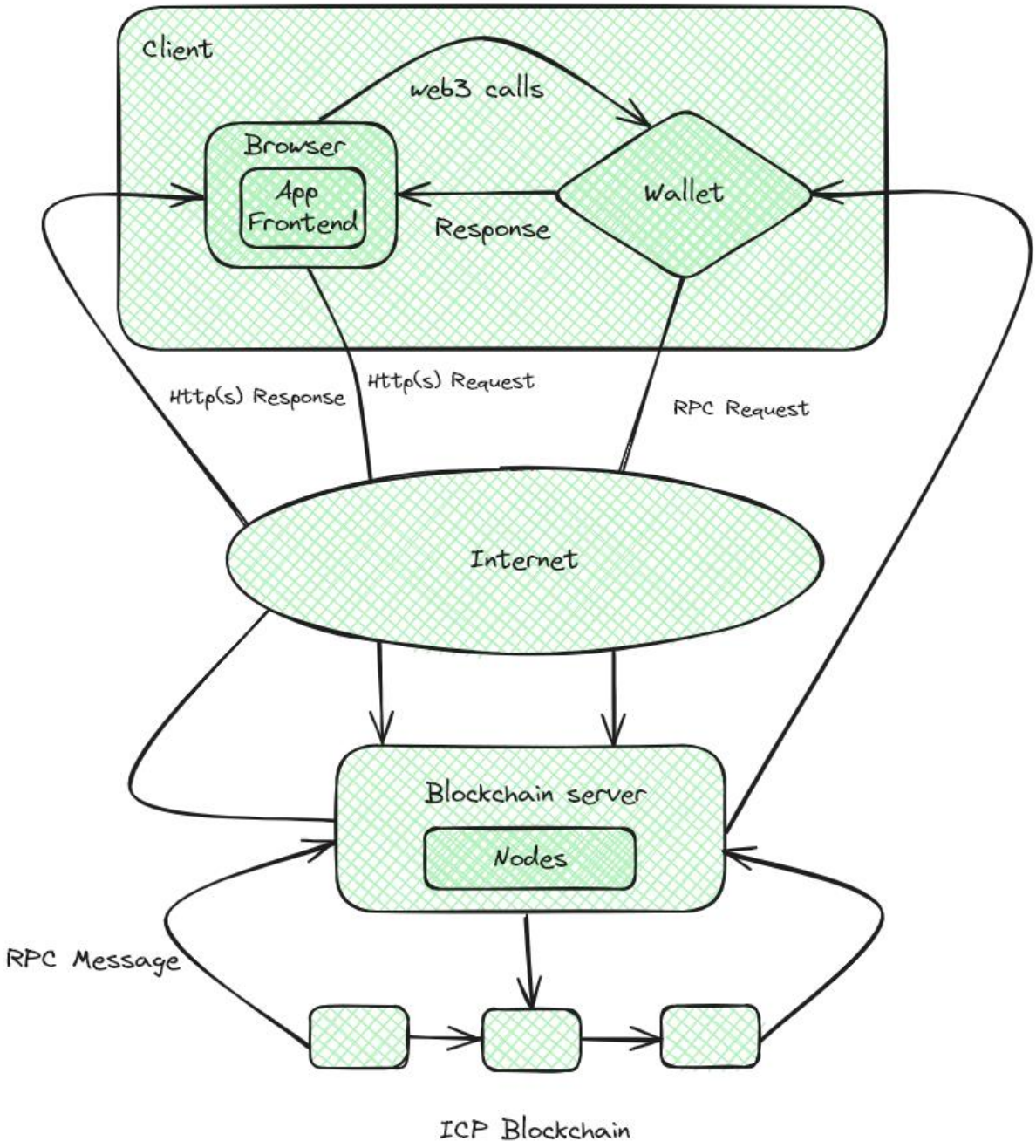
12.4 Custom Multi-Agent with Structured Loops

The concept of a Custom Multi-Agent with Directed Acyclic Graphs (DAG) on-chain involves creating a system where multiple agents interact in a structured manner within the Internet Computer blockchain environment. In the ecosystem, each agent operates within a loop structure where they observe, plan, and react to events iteratively. This loop drives the activity and decision-making process of each agent, ensuring continuous engagement and responsiveness within the simulation. The introduction of the LangGraph library enables more complex interactions between agents and tools, allowing for multi-faceted engagements. These tools facilitate the execution of diverse actions by agents, enhancing the adaptability and functionality of the system within the Internet Computer Protocol (ICP). Developers can create custom agents with structured tools by leveraging LangGraph templates, which provide downloadable and customizable components for quick integration. By examining pre-built agents and following a similar development approach, users can tailor agents to specific use cases, incorporating unique prompts, rules, and output parsers. The LLM-based multi-agent marketplace facilitates the creation of custom agents with Directed Acyclic Graphs (DAG) on-chain. This marketplace allows for the deployment of autonomous agents that can interact within the blockchain ecosystem, enabling users to define agent behaviours, interactions, and decision-making processes in a structured and customizable manner. By combining the capabilities of structured tools, agent loops, and customs agent creation within the blockchain environment, developers can design sophisticated multi-agent systems that operate efficiently and effectively on-chain, fostering dynamic interactions and intelligent decision-making processes.

12.5 dApp Frontend and ICP Cannister

Helion's decentralized application on Internet computer Protocol will use a front-end framework such as ReactJS or NextJS to build the application. These frameworks allow the team to create user-friendly interfaces that can communicate with the blockchain. As part of the application, it will have an identity service provided by the ICP blockchain to handle user authentication and authorization. This ensures a secure and privacy-preserving way for users to interact with your dApp. The application will also communicate with ICP canisters and use the appropriate client libraries, such as the Dfinity Canister SDK, to interact with the canisters that contain the business logic and data for the dApp. Boundary nodes act as the gateway between the front-end application and the ICP blockchain. They translate user requests into API calls to the canisters and handle the responses. For data storage and hosting, dApp will utilize decentralized Internet Computer's Protocol built-in storage capabilities. The ICP blockchain does provide a web-scale performance platform that can help optimize front-end applications to handle the latency and throughput characteristics of the network. We also plan to improve the application runtime build using techniques like caching and batching requests to help improve the user experience. The application will implement governance features, that can leverage the Network Nervous System (NNS), the decentralized autonomous organization (DAO) that governs the ICP network.

13. Web3.0 Architecture



14. Real World Use-Case

Real-world applications of Multi-Agent Systems span various domains and industries, leveraging the collaborative interaction of multiple agents to address complex challenges.

Here are some key real-world applications:

14.1 Graphical Applications

Multi-agent systems are applied in real-world scenarios to graphical applications such as computer games, showcasing their versatility and effectiveness in interactive environments.

14.2 Disaster Response

Multi-agent systems play a crucial role in disaster response scenarios by coordinating teams of agents to survey damaged locations, restore utilities, rescue injured individuals, and maximize points within a specified time frame. These systems help optimize resource allocation, task prioritization, and team coordination in emergencies.

14.3 Online Trading

Multi-agent systems research offers an appropriate approach for online trading applications, where agents can interact, negotiate, and make decisions autonomously to optimize trading strategies and outcomes.

14.4 Target Surveillance

Multi-agent systems are utilized in target surveillance applications to enhance monitoring, tracking, and surveillance activities by coordinating multiple agents to cover designated areas, gather information, and respond to potential threats effectively.

14.5 Social Structure Modelling

Multi-agent systems are employed in social structure modelling to simulate and analyse social interactions, behaviours, and dynamics within a given environment. These systems help researchers understand and predict social phenomena by modelling the interactions between autonomous agents.

14.6 Policy Simulation

Multi-agent systems are used in policy simulation applications to model and simulate the impact of different policies on complex systems. By leveraging the collective intelligence of multiple agents, these systems can analyse policy scenarios, predict outcomes, and inform decision-making processes.

14.7 Game Simulation

Multi-agent systems are applied in game simulation scenarios to create interactive and dynamic gaming environments. By utilizing multiple agents with specialized capabilities, these systems enhance the realism, complexity, and adaptability of game simulations, providing engaging and challenging experiences for players.

14.8 Task-Oriented Single Agents

LLM-powered agents excel in understanding natural language instructions and autonomously accomplishing well-defined goals. They can break down high-level objectives into sub-tasks, dynamically form plans, and adapt to new environments, showcasing versatility in tasks like household chores, administrative workflows, educational systems, programming, healthcare, and engineering design.

14.9 Human-Agent Collaboration:

In scenarios requiring human involvement, LLM-based multi-agents collaborate with humans in various ways. This collaboration can involve humans providing high-level instructions, and feedback, or engaging in empathetic conversations to complement the capabilities of the agents. Different paradigms like Instructor-Executor and Equal Partnership demonstrate the flexibility and adaptability of these systems in working alongside humans.

15. The Helion Ecosystem

The Helion ecosystem employs two distinct tokens, each with its specific roles and functionalities. They include HLN and PSM tokens.

HLN acts as the governance token, enabling holders to partake in vital decision-making processes regarding the platform's development, management, and operation. This participation ensures that HLN holders have a significant influence on the platform's future direction. On the other hand, PSM functions as the utility token within the Helion platform, facilitating all transactions and activities. It's utilized for deploying and using custom multi-agents on the platform, accessing these multi-agents in the marketplace, and distributing royalties to developers and creators within the marketplace. PSM token generation comes from burning HLN tokens, a mechanism designed to maintain its value stable and independent of market changes. Moreover, HLN tokens can be acquired through various methods aside from their initial issuance. These include direct purchases on exchanges, receipts as compensation for creating agents, or contributions to the Helion ecosystem.

16. Conclusion

In conclusion, multi-agent workflows are a powerful tool for building complex applications, especially in the context of Web3 applications on the Internet Computer Protocol blockchain.

By structuring diverse loops of agents with specialized skills and leveraging the capabilities of LLMs, organizations can streamline complex processes, improve decision-making, and achieve more efficient outcomes in various domains. In summary, the integration of LLM-based multi-agent systems in Web3 applications holds the promise of enhancing interactions, decision-making, and task execution within decentralized systems, ultimately leading to more intelligent, efficient, and reliable decentralized applications. Large Language Model (LLM) based on multi-agent systems are being applied in various domains and applications, including problem-solving, world simulation, and conversational systems. These systems have shown considerable progress in complex tasks and have the potential to revolutionize the way we approach problem-solving and decision-making in Web3 applications.

17. Reference

LangChain. (2024, January 23). LangGraph: Multi-Agent Workflows. Retrieved from https://blog.langchain.dev/langgraph-multi-agent-workflows/

Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., & Zhang, X. (2024). Large Language Model based Multi-Agents: A Survey of Progress and Challenges. arXiv. Retrieved from https://arxiv.org/abs/2402.01680v1

Singh, M. P. (n.d.). Multi-Agent Systems for Workflows. Retrieved from https://www.csc2.ncsu.edu/faculty/mpsingh/papers/databases/mas-for-workflows.pdf

Sahu, S. (2024, January 24). The next generation of AI software workflow agents in the multi-modal era. Siddhant's Substack. Retrieved from https://sidstage.substack.com/p/the-next-generation-of-ai-software

Coenen, F., & Bench-Capon, T. (2009). Argumentation in artificial intelligence. *Artificial Intelligence Review, 30*(3), 195-215. https://doi.org/10.1007/s00146-009-0193-6

Authors. (2022). Human-in-the-loop online multi-agent approach to increase trustworthiness in ML models through trust scores and data augmentation. *arXiv*. Retrieved from https://arxiv.org/pdf/2204.14255.pdf

Haddadi, A., & Sundermeyer, K. (2007). Escape and intervention in multi-agent systems. *Proceedings of the International Conference on Autonomous Agents*. Retrieved from https://www.researchgate.net/publication/227004807_Escape_and_intervention_in_multi-agent_systems

Microsoft AutoGen Team. (2023, December 1). AutoGen Studio: Innovations in Automation. Retrieved from https://microsoft.github.io/autogen/blog/2023/12/01/AutoGenStudio/

Singh, M. P. (n.d.). Multi-Agent Systems for Workflows. Retrieved from https://www.csc2.ncsu.edu/faculty/mpsingh/papers/databases/mas-for-workflows.pdf

Finkbeiner, B., & Müller, C., & Seidl, H., & Zălinescu, E. (2017). Verifying Security Policies in Multi-agent Workflows with Loops. *arXiv*. Retrieved from https://arxiv.org/abs/1708.09013

Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., & Wohed, P. (2005). An Analysis and Taxonomy of Unstructured Workflows. Retrieved from https://www.researchgate.net/publication/221586043_An_Analysis_and_Taxonomy_of_Unstructured_Workflows

AHLBERG, S. (2019). *Human-in-the-Loop Control Synthesis for Multi-Agent Systems under Metric Interval Temporal Logic Specifications*. Retrieved from https://people.kth.se/~sofa/Publications/LicThesis.pdf

DFINITY Foundation. (n.d.). Overview of the Internet Computer Protocol (ICP). Retrieved from https://internetcomputer.org/docs/current/developer-docs/getting-started/overview-of-icp

Kampik, T., Najjar, A. (2020). Simulating, Off-Chain and On-Chain: Agent-Based Simulations in Cross-Organizational Business Processes. *Information*, 11(1), Article 34. https://www.mdpi.com/2078-2489/11/1/34

LangChain. (n.d.). GP-Team: A Multi-Agent Simulation. Retrieved from https://blog.langchain.dev/gpteam-a-multi-agent-simulation/

LangChain. (n.d.). Structured Tools. Retrieved from https://blog.langchain.dev/structured-tools/