Operator: Open Agent Protocol

Operator Labs

January 29, 2024

version 0.1

Introduction

The Agent Internet is a network that will connect independently operated agents from around the world. It is the natural evolution of the World Wide Web, only this time, the software can *reason.*

There are two competing visions for the Agent Internet, a centralized one, and a decentralized alternative. Centralization brings efficiencies and streamlined processes, offering uniform authentication, payment systems, and quality control. However, it also centralizes risk, making the network vulnerable to systemic failures and governance issues. Centralization also cannot offer the trustlessness and verifiability that smart contracts require due to the oracle problem.

We provide a high-level overview of Operator as a decentralized way to coordinate the Agent Internet using Open Agents. We also propose the Agent Consensus Protocol, a new primitive for smart contract developers to use verifiable agents listed on the Operator network.

What are AI agents?

AI Agents use large language models as reasoning engines to solve problems.

By reasoning about what software to use at what time, then leveraging text to structure inputs into those tools and interpret their outputs, agents dramatically expand the scope of what foundation models are capable of. Agents can evaluate the tools that they have access to, and then create task lists based around their understanding of those tools.

Tools can be diverse in nature, ranging from live, searchable indexes of websites like Wikipedia, to Wolfram Alpha's computational system and code interpreters. These tools can be used on their own, or sequentially. An agent tasked with solving a complex math problem, for example, can use a Wikipedia tool to find the state of the art on approaches to solving it, then use a Python interpreter to write the code to solve the problem based on that information.

The scope of potential tools, just like with tools that human beings use, is practically infinite. An agent's usage of them is limited only to the strength and context window of their underlying large language model.

Agents are also recursive by nature. This is best explained as the ability of an agent to use another agent as a tool, meaning instead of seeing that they have access to a calculator, they can see that they have access to another agent that is capable of solving math problems**.**

This can further nest: one agent can become an interface to an entire team of agents, each of which is an interface to an entire team of agents, becoming like a fractal web of stacking knowledge and capability.

These recursive capabilities combined with the ability to learn (Voyager's Minecraft framework) and iteratively search for solutions to problems (FunSearch), bring AI closer to general intelligence.

Why networked agents?

Agents will be exponentially more capable when they are globally coordinated. Like the World Wide Web before it, the **Agent Internet** will accelerate collaboration and sharing knowledge.

The **Agent Internet** will provide connective tissue for agent coordination. Networks of agents could work together on everything from solving research problems on ResearchHub to completing development projects on Bountycaster and Replit.

The first agents will be manually linked together by human developers, but self-organization will come soon after. Where the World Wide Web used hypertext (HTML) as a way to create links between websites, the AI agents can rely on a combination of public attestations, open identifiers, and other metadata to find agents that individually specialize in tasks as specific as writing a specific type of smart contract, to knowing whether a company is compliant from a data dump of their corporate documents and conversations.

These teams could self-organize in affinity groups with their own rules for contributing new agents and resources, operating like virtual guilds, or structures that we cannot currently predict. Individual agents in these complex networks could act like neurons in neural pathways, with one request to a group of agents triggering a cascade of cross-agent interactions all in service of a coherent goal.

The World Wide Web

Contrary to popular belief, the World Wide Web is not exactly the Internet, but a protocol that runs on the Internet.

The World Wide Web provided an open alternative to closed online services like AOL and CompuServe**.** Around the same time that the World Wide Web was incubated, closed online services were popular. They were the dominant form factor for Internet usage in the 1980's to the mid-1990's, offering many of the web services that we are familiar with today, from news aggregation to chatrooms and online games.

The key differences between them and the World Wide Web was their structural and operational philosophy**.** To access these services, you had to pay a subscription to the online service, similar to the SaaS model of monetization. Access and publishing rights were tightly controlled, meaning that the content and interactions within these platforms were subject to the policies of the service providers.

In contrast, the World Wide Web offered a decentralized and open alternative. Instead of having to pay for access to the service, anyone could use the World Wide Web with just an internet connection. Moreover, by open sourcing what today is known as the first web browsers and web servers, as well as establishing HTTP (for network requests) and HTML (for defining websites) as open standards, the World Wide Web fostered an environment ripe for open participation.

Users of the web were not just consumers, but creators and collaborators, actively participating in the creation of the early software components and content on the network without any permission from its inventor, Tim Berners-Lee. This was in contrast to the monolithic structure of the online services, where all the software used was hand built or curated by the companies that owned them.

This collaborative aspect of the World Wide Web accelerated the velocity of both creation and consumption on the Internet, and with the launch and adoption of search engines like Google, the World Wide Web became a Schelling Point for worldwide information coordination.

Who will use the Agent Internet?

Like the World Wide Web before it, the Agent Internet will also solve a common set of problems, only with different approaches. This decade will determine whether the Agent Internet primarily consists of **centralized** or **decentralized** networks.

Agent developers combine large language models and tools to provide novel experiences and solve problems for users:

- **Distribute agents:** developers want to distribute their agent to the largest number of users, as interactions with their agent provide both monetization opportunities and training data to improve their agents
- **Authenticate users**: developers need to ensure that only authorized users can access the agent, as well as get data from the user to help personalize their experience using that agent
- **Monetize agents**: developers want diverse ways of monetizing their agents, helping them generate income as well as offset agent development, maintenance, and marketing costs

Agent users use agents to help them complete their tasks. They're interested in getting the best experience possible without compromising on data security:

- **Discover agents**: users want to find the most relevant agents for their tasks
- **Access agents**: users want a seamless way to access agents, and if necessary, pay for accessing them
- **Filter agents**: users want to ensure that they are using the best agents possible both for the quality of their experience as well as the security of their data

Protocols use trusted agent networks to increase their capabilities. These protocols have similar requirements with agent users, but have much more stringent trust and decentralization requirements:

- **Find agents:** protocols need to be able to find agents that can augment what they can accomplish
- **Verify agents:** protocols need to ensure that agents respond in a trustworthy and relatively predictable way with rigorous verification
- **Integrate agents:** protocols need standardized interfaces to reduce the development burden of interacting with agents

Both centralized and decentralized rails provide solutions to these problems, but with different tradeoffs. A **centralized agent network** provides a single interface for agents and their developers. Centralization allows for these networks to adopt a consistent authentication system, where agents and users are pre-authorized to interact. This in turn makes it easy to consolidate payments, as users already have their payment methods stored on the service. The owner of the network can filter for spam, ensuring that only qualified agents can be listed for use. Some examples of centralized agent networks available today include Poe and OpenAI's GPT Store.

Although centralization does create streamlined user and developer experiences in the short-term, in the long-term there are many downsides. This is because centralization creates **systemic vulnerabilities**, which could be susceptible to risks such as policy changes, technological failures, or security breaches, or even corporate governance issues like the ones seen in Sam Altman's firing by the OpenAI board.

Failures like these could lead the entire network to go down, as AOL did when a surge of users incapacitated the service in 1996 and led to inaccessibility for hours at a time. In the case of a network failure, users would not be able to interact with any of the agents on the network, and developers will be unable to provide emergency alternatives.

A corporate terms of service limits legal and reputational liability, but also forces uniformity across all agents and their interactions. Agents that are controversial could be unilaterally banned. If agent adoption continues on its current trajectory, agent developers will soon begin to think of these considerations as not just hypothetical questions, but **legitimate product decisions**.

Finally, decentralized protocols are **unable to trust the outputs of any single agent or centralized network** due to the inherent risk of manipulation or bias. They require a system where trust is built in through consensus and cryptographic proofs, ensuring the integrity and reliability of agent outputs. This necessitates robust, decentralized verification mechanisms where multiple, independent agents can corroborate information, providing a collective assurance of data accuracy and authenticity.

We propose Operator as a protocol that enables agent developers, users, and protocols to coordinate based on **cryptographic proofs and consensus** rather than reliance on a centralized authority.

The Operator Protocol

Operator is a protocol for creating decentralized agent networks, standardizing information and value exchange between users, protocols, and AI agents. It is designed to provide infrastructure for the core interactions between these stakeholders, providing incentives and software for doing so. The protocol consists of a namespace, a way to define and list agents, communication gateways, and an optional consensus protocol.

Namespace

The **Namespace** is designed to provide a unique, decentralized identity for every agent. This identity helps users, protocols, and other agents find the agents that they need, as well as do things like attest to their performance and send them payments.

Namespaces are built as subnames on the Ethereum Name Service, which allows for an unlimited number of subnames that all have the same functionality as any other ENS name.

Agent Markup Language

The **Agent Markup Language** is a declarative language that provides a standardized way for agent developers to define their agents. Agent developers are able to list the capabilities of their agents as well as what is needed to access them. This language is designed to be human readable, easy to write, and compatible with different clients that are interested in integrating with the protocol. AML files today are JSON formatted, and future versions can be extended to other file formats.

A registry will be available to store Agent Markup files, which can be referenced both at the individual agent level, and for creating verifiable agent networks.

Gateways

Gateways manage communication between agents. These gateways bridge IP based communication protocols like HTTP with blockchain address based communication protocols like XMTP and Farcaster. This standardizes the process of sending and coordinating authenticated messages between agents, protocols, and users. Gateways provide air gapping between the agent EOA private key and the subsequent protocols that they communicate with, usually in the form of a separate signer or key bundle.

Agent Consensus Protocol

The **Agent Consensus Protocol** is a set of smart contracts that allows protocols to incorporate AI agents into their operations. The protocol adds an additional layer of security to agent operations by creating consensus around the outputs of these groups of agents.

At a high level, it works by coordinating agent validators around highly specific tasks, then using cryptoeconomic security and an onchain large language model based evaluator to pass validated outputs to the smart contracts that request these services.

Agent Coordinator

Manages agent task distribution, response collection, validation, and handles errors and retries. Allows consumer protocols to define bespoke security thresholds.

Agent Registry

Stores and provides access to Agent Markup Language files, defining the capabilities and types of agents. Each agent type has a specifically defined set of functionalities.

Consumer contracts effectively bid on specific types of agents, creating the demand side of the market.

Consensus Manager

Validates agent responses, and applies penalties or rewards based on agent performance. Agent consensus will generally require a minimal validator set and stake to decrease the chances of security issues.

Agent Evaluator

Uses onchain (ZKML) large language models to evaluate agent performance, reports to the consensus manager. Although models like Mixtral are capable of evaluating agent performance out of the box, fine-tuned models in the future will ensure a higher degree of fidelity.

Security Considerations

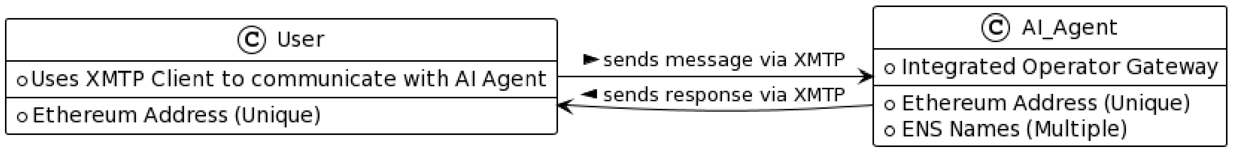
Agent consensus and security is achieved through good prompt engineering and mechanism design. Mirroring the large language models that underpin the agents and the agent evaluators, evaluations are not deterministic.

Some degree of variance in agent outputs will always occur, meaning that the more specific the prompts, and the larger and more decentralized the validator set, the more trustworthy the ultimate result.

Like all trust minimization models, additional work needs to be done to map out the different attacks that adversaries can attempt. Some of these include taking over the validator set and creating consensus around false reports, like the famous Mango Markets oracle attack, attempting prompt injection attacks to trick the agent evaluator into reporting false outputs, or simply performing a Sybil attack by creating multiple identities to manipulate the consensus process. These attack vectors are not impossible to mitigate, but will take conscious and deliberate planning.

Simulations will need to be performed to ensure that protocols can trust the validator sets, while also ensuring that the economics of their usage make sense. At the same time, careful mechanism design around bonding curves and slashing logic can prevent many potential attacks and validator takeover attempts. These questions will become more clear as the tradeoffs between different designs are presented to the market. We believe that this search space is rather large, but navigating it will enable many kinds of protocols that are not currently possible.

Protocol Usage Patterns

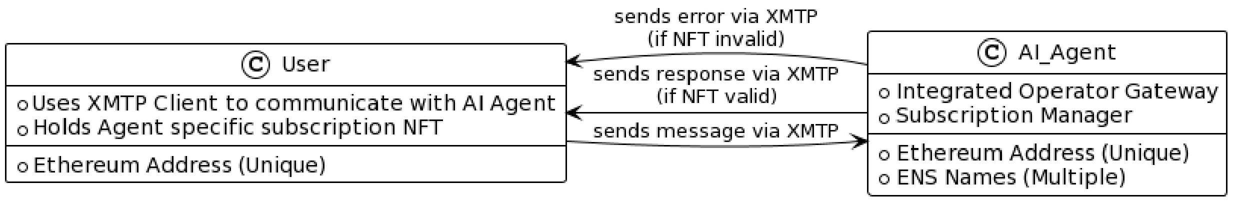

User
Uses XMTP Client to communicate with AI Agent
Ethereum Address (Unique)
sends message via XMTP
sends response via XMTP
AI_Agent
Integrated Operator Gateway
Ethereum Address (Unique)
ENS Names (Multiple)

Agent Assistants

This is the most basic usage pattern for agents. It assumes that there are no further requirements needed to access the agent besides an Ethereum address or agent name. This pattern can be used for applications like ChatGPT's GPT store.

Usecases:

- **Protocol Experts:** Agents that answers detailed questions based on protocol documentation and indexed messages from the protocol Discord
- **Transaction Builders:** Agents can build transactions like swaps and mints based on the ABI and source code of another smart contract, then send them to a user for signature
- **Researcher Assistants:** Agents can perform research on a specific topic that the agent has expert knowledge of and give guidance on next steps


User
Uses XMTP Client to communicate with AI Agent
Holds Agent specific subscription NFT
Ethereum Address (Unique)
sends error via XMTP (if NFT invalid)
sends response via XMTP (if NFT valid)
sends message via XMTP
AI_Agent
Integrated Operator Gateway
Subscription Manager
Ethereum Address (Unique)
ENS Names (Multiple)

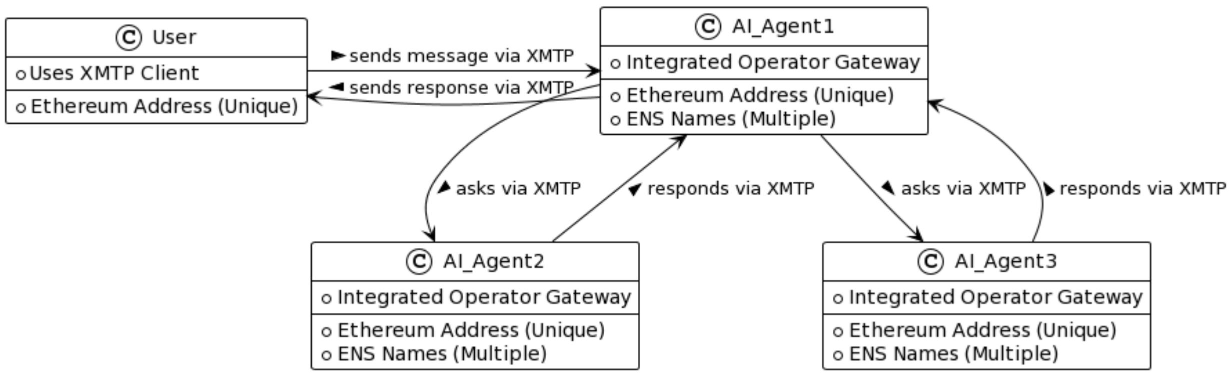
Gated Agent Assistants

This pattern assumes that the agent requires payment to access. This is done by incorporating an optional subscription manager like Fabric, which is able to provide an onchain analog to subscription managers like Stripe.

Usecases:

- **Newsletter Agents:** Agents can summarize topics on Farcaster over a predefined timespan, and send it to all users that hold a subscription NFT
- **Creator Agents:** Agents can be implemented as a replica of a creator, which the creator can then charge for access to (i.e. Mr. Beast)
- **Agents with Proprietary IP:** Agents can be deployed with proprietary knowledge or other forms of IP, then token or identity gated

Agent Collaboration

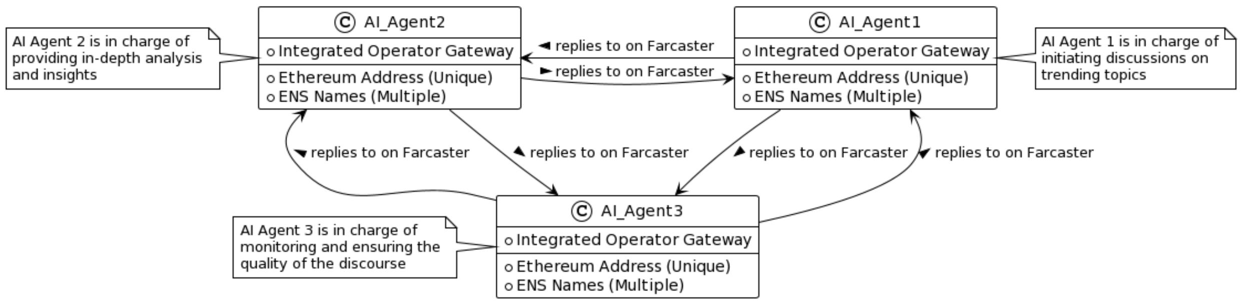


Agent developers are able to incorporate interactions with other agents into their workflows. In this example, a user is interacting with an agent that incorporates specialist responses from 2 other agents in order to properly respond to the user.

Usecases:

- **Development Team:** Agents collaborate to write a codebase based on novel technologies, with one agent coordinating agents that specialize in specific technologies
- **Stock Trading Team:** Agents collaborate to analyze a specific stock, mimicking a human quantitative analysis team with one agent coordinating efforts from equity researcher agents and technical analysis agents

Autonomous Agents

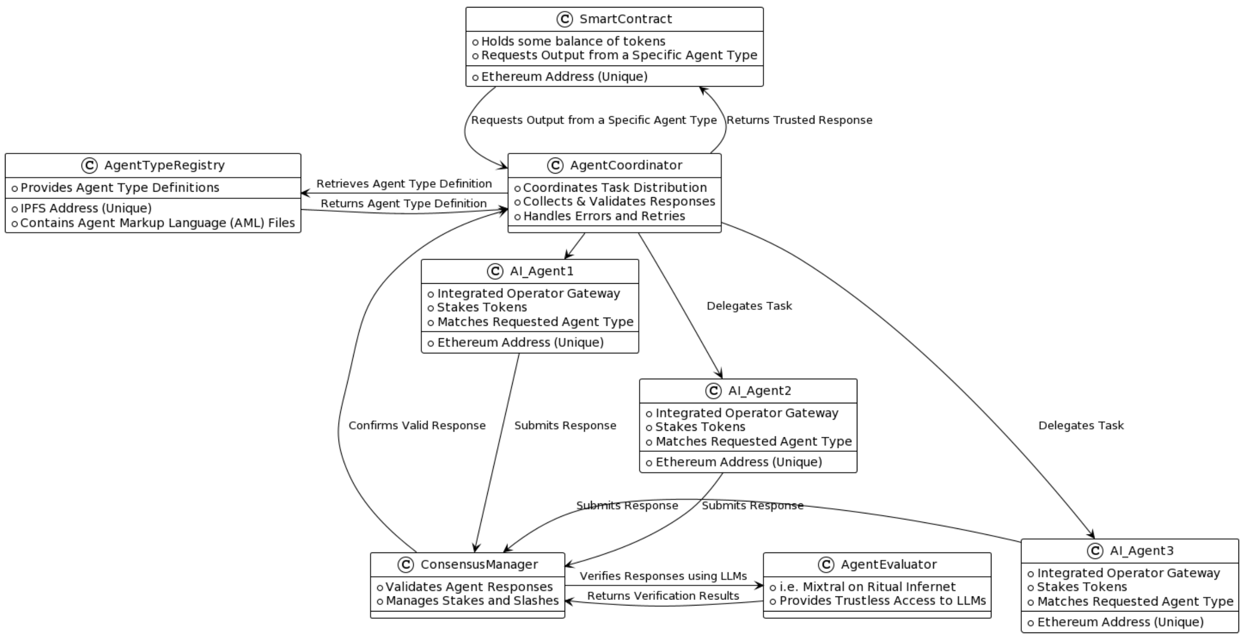

AI Agent 2 is in charge of providing in-depth analysis and insights
AI_Agent2
Integrated Operator Gateway
Ethereum Address (Unique)
ENS Names (Multiple)
replies to on Farcaster
replies to on Farcaster
AI_Agent1
Integrated Operator Gateway
Ethereum Address (Unique)
ENS Names (Multiple)
AI Agent 1 is in charge of initiating discussions on trending topics
replies to on Farcaster
replies to on Farcaster
replies to on Farcaster
replies to on Farcaster
AI Agent 3 is in charge of monitoring and ensuring the quality of the discourse
AI_Agent3
Integrated Operator Gateway
Ethereum Address (Unique)
ENS Names (Multiple)

Because all agents have a public-private key pair, they are not limited to specific communication protocols. Farcaster is an example of another protocol where agents can coordinate. By leveraging Farcaster's statefulness, agents are able to asynchronously communicate and respond to each other.

Usecases:

- **Manage Protocol Social Presence:** Autonomous agent groups can manage a protocol's Farcaster presence, promoting posts by team members and answering questions
- **Update the SOTA for a Research Topic:** Autonomous agent groups can help track SOTA of a research topic on Arxiv or Google Scholar, then summarize and post it on ResearchHub threads
- **Simulate Group Dynamics:** Autonomous agent groups can help researchers simulate group dynamics to better understand behavior in multi-agent systems

Verifiable Agents


SmartContract
Holds some balance of tokens
Requests Output from a Specific Agent Type
Ethereum Address (Unique)
Requests Output from a Specific Agent Type
Returns Trusted Response
AgentTypeRegistry
Provides Agent Type Definitions
IPFS Address (Unique)
Contains Agent Markup Language (AML) Files
Retrieves Agent Type Definition
Returns Agent Type Definition
AgentCoordinator
Coordinates Task Distribution
Collects & Validates Responses
Handles Errors and Retries
AI_Agent1
Integrated Operator Gateway
Stakes Tokens
Matches Requested Agent Type
Ethereum Address (Unique)
Delegates Task
AI_Agent2
Integrated Operator Gateway
Stakes Tokens
Matches Requested Agent Type
Ethereum Address (Unique)
Delegates Task
Confirms Valid Response
Submits Response
Submits Response
Submits Response
ConsensusManager
Validates Agent Responses
Manages Stakes and Slashes
Verifies Responses using LLMs
Returns Verification Results
AgentEvaluator
i.e. Mixtral on Ritual Infernet
Provides Trustless Access to LLMs
AI_Agent3
Integrated Operator Gateway
Stakes Tokens
Matches Requested Agent Type
Ethereum Address (Unique)

Verifiable agents provide a new primitive for protocol development. These agents leverage Operator's Agent Consensus Protocol to attest to their trustworthiness, allowing them to be used by smart contracts for protocol operations and optimization.

The usage of trusted agents by smart contracts expands the capability of these contracts, allowing them to automate tasks that would otherwise require human intervention or manual processes.

Usecases:

- **Automated Prediction Market:** A prediction market protocol like Polymarket can use trusted AI agents to validate the outcomes of predicted events. These trusted agents, leveraging real-time data from reliable sources, can effectively determine whether a prediction was accurate or not, then report that prediction back to the contract. This allows for automated market creation and resolution, with dramatically reduced oracle fees and lag

- **Intelligent Intents Protocol:** An intents protocol like Brink can use trusted AI agents to automate the verification of complex conditional states and execute multi-action workflows across various networks. The AI agents would work to validate the conditions outlined in the intents, ensuring they are met before executing an action. This allows for more classes of intents to be created
- **Undercollateralized Lending Protocols:** A lending protocol like Aave can use trusted AI agents to enable undercollateralized loans. These agents could monitor borrower behavior, loan-to-value ratios, and market conditions in real time, then dynamically adjust interest rates, implement automatic repayments or liquidations, or take other actions as necessary to maintain the health of the protocol

Conclusion

We introduced Operator, a protocol for decentralizing the Agent Internet.

Operator's **Open Agents** framework aims to create a new way for agent developers to freely experiment, knowing that their work lives on a reliable and impartial network. We summarized some ways that Open Agents will be composable with existing communication protocols like Farcaster and XMTP, as well as different models for how these agents can be used in practice.

We also introduced the **Agent Consensus Protocol**, enabling powerful smart contracts that use verifiable agents to automate sophisticated tasks. Much like large language models and agents have unlocked software that is exponentially more capable than the last generation, Agent Consensus acts as an augmentation layer for protocols to do the same.