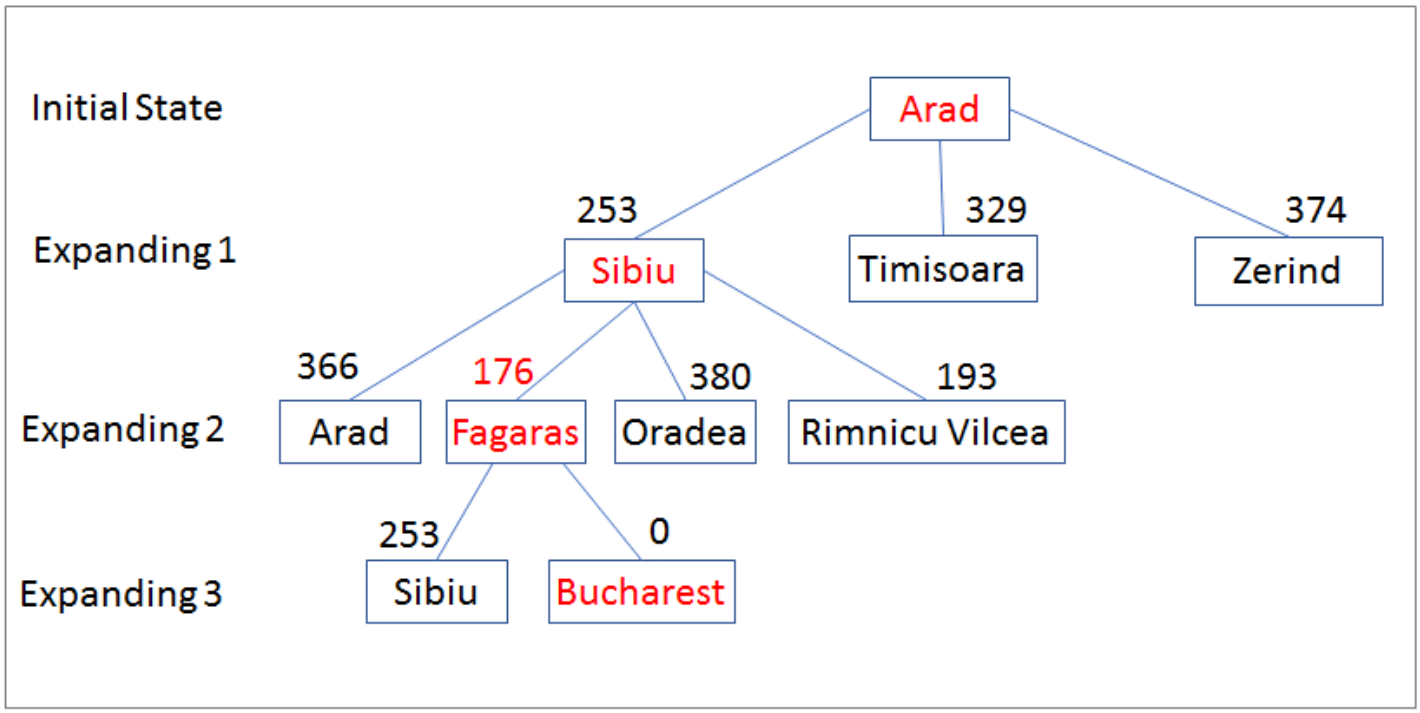
De(cistion)Agent: A Framework for Distributed Decision Processes for AI Agents

1. Background: The Governance Challenge for AI Agents

Artificial Intelligence (AI) Agents, particularly those powered by Large Language Models (LLMs), hold immense potential for participating in and even automating aspects of governance, especially in decentralized systems. However, deploying them reliably in such contexts faces significant hurdles:

1. **Consensus:** LLMs are inherently probabilistic. For a given input sequence, an LLM calculates probability scores for potential next words, leading to a vast number of possible output sequences. While mechanisms like greedy search or beam search select a likely output (the red path in the diagram below), this selection isn't inherently deterministic or universally agreed upon. In governance scenarios, especially those involving asset allocation or critical decisions, how do we ensure that one specific output path is *the* authoritative and accepted result, agreed upon by all stakeholders? Relying solely on trusted execution environments (TEEs) isn't sufficient; the *process* of generating the AI's decision requires its own form of verifiable consensus.



(Image illustrating multiple potential output sequences from an LLM)

1. **Identity (Consistency):** Even if consensus on *an* output is achieved, distributed systems face concurrency challenges. Multiple valid, yet potentially contradictory, decisions could theoretically be generated simultaneously for the same Agent regarding the same issue. This is analogous to the double-spending problem in blockchains, which is resolved by enforcing a single, linear chain of transactions. An AI Agent involved in governance cannot simultaneously provide contradictory directives, just as a human leader cannot credibly issue opposing orders at the same time. The Agent must exhibit a singular, consistent identity and state at any given point in time.
2. **Continuity:** Human societies rely on memory for continuity. Decisions, commitments, and identities persist over time. An AI Agent participating in governance must also possess a reliable form of memory. Its past interactions, decisions, and learned state must inform its future actions consistently. An Agent that "forgets" its previous commitments or rationale loses its credibility and effectiveness, much like an elected official forgetting their campaign promises.

Consensus, Identity, and Continuity are therefore fundamental properties required for AI Agents to function reliably and trustworthily within governance frameworks, particularly in distributed environments.

2. Overview: The De(cistion)Agent Framework

The De(cistion)Agent framework (DeAgent) is proposed to address these three core challenges. It provides a structure for creating, operating, and interacting with AI Agents within distributed systems.

A DeAgent instance is defined by:

- **Lobe:** Represents the cognitive engine of the Agent. It encapsulates the logic for invoking one or more LLMs, processing inputs, and generating outputs. Agent creators specify the Lobe(s) to be used, potentially referencing pre-defined or custom implementations.
- **Memory:** Contains the Agent's initial state (genesis memory) and its accumulated interaction history (evolving memory). This provides the basis for continuity.
- **Tools:** A defined set of capabilities the Agent can utilize, such as accessing external data, interacting with other systems, or performing specific actions within the distributed environment. Includes descriptions for how the Lobe can invoke them.

Creation: A DeAgent is instantiated by publishing its definition (Lobe URI, initial Memory state, Tool specifications) to a specific address or identifier within a distributed system (e.g., a blockchain mainnet, a decentralized storage network).

Inherited Properties: Crucially, a DeAgent naturally inherits the properties of its underlying distributed system. If the system guarantees immutability for published data, the Agent's core

definition becomes immutable. If the system provides strong consensus and ordering guarantees (finality), this forms the foundation for the Agent's Identity and Continuity.

Technical Neutrality: The DeAgent framework is agnostic to specific encoding formats, publication methods, or advanced features (like native immutability) of the underlying distributed system. However, developers and communities adopting DeAgent on a specific platform should establish clear standards and specifications. The *minimum requirement*for the host system is that it achieves a unique, final state agreed upon by participants (i.e., it possesses inherent Consensus and Identity).

Interaction: Users interact with a DeAgent by submitting operations (e.g., transactions in a blockchain context) targeted at the Agent's address. These operations contain the interaction payload (text, image, audio, video inputs, or references to immutable data pointers if payloads exceed size limits). Platform-specific operation standards are needed.

Execution: The DeAgent framework relies on **Executors**. These are nodes responsible for processing interactions. Executors can be centralized, run by multiple trusted parties, or fully decentralized. They monitor the distributed system for interactions targeting DeAgents they service, retrieve the Agent's definition and current state (Memory), execute the interaction using the specified Lobe and Tools, and generate potential results.

Consensus and Finalization: Executors submit their results along with proofs of execution. A network of **Committers**(validators) then applies a consensus protocol (detailed later) to select a single canonical result for each interaction cycle, ensuring Identity. This result updates the Agent's state (Memory) on the distributed system, establishing Continuity.

3. DeAgent Lifecycle and Roles

The DeAgent ecosystem involves several key roles:

- **Agent Creator:** Defines the Agent's Lobe, initial Memory, and Tools, then publishes this definition to the distributed system, bringing the Agent into existence.
- **Agent User:** Interacts with the Agent by submitting queries or tasks via operations on the distributed system.
- **Framework Operators:** Maintain the operational integrity of the DeAgent ecosystem:
  - **Executors:** Nodes that run the Agent's logic (Lobe + Tools + Memory). Can range from private nodes (potentially in TEEs) to open participants.
  - **Compute Providers (Optional):** Specialized nodes offering execution services for specific (often open-source) models, potentially forming a sub-network with its own incentive mechanisms.

- **Committers:** Nodes responsible for validating execution proofs submitted by Executors and achieving consensus on the canonical interaction result for each step. They select the final result (often guided by principles like minimal entropy) and commit it to the Agent's state chain on the distributed system.

Agent Lifecycle:

1. **Creation:** An Agent Creator publishes the Agent's definition on the distributed system.
2. **Interaction Request:** Users submit interaction requests targeting a specific Agent (or group of Agents). These requests might include parameters like preferred Executors (via selectors) or incentives (gas fees/tips). Requests enter an **Interaction Pool** (akin to a transaction mempool).
3. **State Dependency:** Unlike typical transaction pools where transactions might remain valid across multiple blocks, Agent interaction requests are highly state-dependent. An Agent's response relies heavily on its current Memory. Once the Agent's state is updated by a finalized interaction, previous interaction requests targeting the *older* state become invalid and are discarded.
4. **Execution:** Executors select requests from the Interaction Pool (based on factors like incentives, selectors, or internal policies). They execute the Agent's logic using the specified Lobe, the Agent's current Memory state, and available Tools.
5. **Candidate Submission:** Executors submit their results (Response, Memory updates, Tool usage, proofs) to a **Candidate Pool**. Multiple Executors might process the same interaction request, leading to multiple candidate results.
6. **Consensus & Finalization:** Within a defined timeframe (related to the Agent's "tick rate" or timeout), Committers evaluate the candidate results in the pool. Using a consensus mechanism (e.g., PoS-like voting, PBFT) and potentially a selection heuristic (like the minimum entropy principle described below), they choose a single canonical result.
7. **State Update:** The chosen result is committed to the distributed system, updating the Agent's official Memory/state chain. This ensures Identity (only one result per step) and Continuity (the state evolves predictably).
8. **Death:** If an Agent remains inactive (no finalized interactions) for a predetermined period, it can be considered "dead" or dormant. Future interaction requests targeting it may be ignored by Executors.

4. Lobe: The Cognitive Engine

The Lobe is the core processing unit of the Agent.

4.1 Lobe Definition

Agent Creators primarily need to specify the Lobe's Uniform Resource Identifier (URI). This URI tells Executors how to access and run the Lobe.

- **Built-in Lobes:** The Executor might have optimized, native implementations referenced directly.
- **Remote/Community Lobes:** The URI might point to a standardized service endpoint or package developed by the community.

Lobe Inputs:

- **Memory Context:** Relevant portions of the Agent's Memory (e.g., Hot Memory, relevant Longterm Memory retrieved via RAG).
- **User Query:** The current interaction payload from the user.
- **Tool Manifest:** List of available Tools and their descriptions/APIs.

Lobe Outputs:

- **Memory Directives:** Instructions for updating the Agent's Memory (e.g., add to Longterm Memory).
- **Response:** The output generated for the User.
- **Tool Usage Directives:** Instructions specifying which Tools to use and with what parameters.
- **Execution Proof:** Verifiable evidence of the Lobe's execution process.

Essentially, a Lobe wraps one or more foundational models (like LLMs). It preprocesses inputs (combining query, memory, tool info), invokes the model(s), post-processes the model output, and formats the final Lobe Output structure.

4.2 Lobe Consensus

Ensuring the Lobe's execution is verifiable and deterministic (or has agreed-upon non-determinism) is crucial for overall DeAgent consensus. This requires proofs for two aspects:

1. **Model Invocation/Execution Proof:** Proving the underlying AI model was called correctly and produced a specific output. This is challenging.
2. **Non-Model Logic Proof:** Proving the correctness of the surrounding business logic within the Lobe (preprocessing, post-processing, tool handling). This is more amenable to standard techniques like Zero-Knowledge Proofs (ZKPs).

4.2.1 Model Invocation Proof Mechanisms

- **Closed-Source Models (Whitelisting + TLS Proofs):** For proprietary models from trusted vendors (e.g., OpenAI, Anthropic, Google, Cloudflare AI), full execution proof is impossible. Instead, we rely on a whitelist approach. Executors provide proof (e.g., via ZK-TLS techniques like DECO or TLSNotary) demonstrating they securely connected to the vendor's official API endpoint and relayed the specific query and response without tampering.
- **Open-Source Models (Challenges and Hybrid Approach):**
  - **zkML:** Techniques like zkSNARKs applied to machine learning models (zkML) could theoretically provide full execution proofs. However, current zkML approaches (even promising ones like zkPyTorch mentioned by Polyhedra, though practical availability was limited at the time of writing) often incur prohibitive computational overhead and latency.
  - **Replication Issues:** Simply having multiple nodes run the same open model isn't foolproof. Differences in hardware (e.g., CUDA SM versions), software libraries, and floating-point arithmetic can lead to slightly different outputs, breaking naive consensus.
  - **Proposed Hybrid Approach (Active + PoW-like Entropy Minimization):** We propose a specialized compute network for open models. Participants run the models. When an interaction requires an open model, multiple participants can execute and submit their results. Instead of requiring identical outputs, we use an *entropy function* to select the "best" result. The participant submitting the result with the lowest calculated entropy is chosen. This incentivizes participants to produce high-quality, relevant outputs. Penalty and reward mechanisms should be integrated into this network.

4.2.2 Entropy Function for Result Selection

To objectively select among potentially valid but different model outputs (especially from open models), we use an entropy-based selection mechanism:

1. **Encoding Model:** Utilize a separate, highly capable encoder model (potentially a RAG-enhanced QA model) denoted as `E`. This model maps text sequences to dense vector embeddings: `vector = E(sequence)`.
2. **Entropy Calculation:** For a given input (`input`) and a candidate output (`output`), calculate the negative dot product of their embeddings:
   `Entropy = - ( E(input)  E(output) )`
   (Note: Using the negative dot product means higher similarity/relevance corresponds to lower "entropy" in this context. Cosine similarity could also be used).
3. **Selection:** Among all candidate outputs submitted by Executors for a given interaction, the Committers select the one yielding the *lowest* entropy value.

Security Assumption: This mechanism relies on the assumption that it is computationally difficult for an attacker to craft a malicious or arbitrary response that *also* achieves the

minimum entropy score for a given input, according to the chosen encoder model `E`. This assumption requires ongoing research and validation.

4.3 Lobe Summary

Through ZK proofs for non-model logic, TLS proofs for whitelisted APIs, and an entropy-based selection for open models, the DeAgent framework aims to establish robust consensus around the Lobe's execution results.

5. Memory and Tools: Defining Agent Individuality

While the Lobe provides cognitive power, an Agent's unique identity, history, and capabilities are defined by its Memory and Tools.

5.1 Memory

An Agent's entire interaction history is immutably stored on the underlying distributed system, forming its complete Longterm Memory. However, for practical Lobe invocation, relevant parts are selected:

- **Hot Memory:** The most recent `N` interactions (or up to `M` tokens) are automatically included, providing immediate conversational context.
- **Longterm Memory:** A Retrieval-Augmented Generation (RAG) mechanism searches the Agent's full history to find relevant past interactions based on the current query. Results exceeding a certain relevance threshold (up to a specified number of items/tokens) are included.

Consensus on Memory: Retrieving Hot Memory and verifying historical interactions (for RAG) directly from the distributed ledger is straightforward and verifiable by Committers. However, the *RAG process itself* (the specific algorithm and relevance scoring) is typically *not* subject to strict consensus verification due to complexity. We trust that Executors, incentivized to have their results selected, will employ effective RAG techniques. Committers primarily verify that the Longterm Memory snippets provided *do originate* from the Agent's validated history.

5.2 Tools

Tools extend an Agent's capabilities beyond pure text generation.

- **Built-in Tools:** Provided by the DeAgent framework:
  - **Distributed Data Query:** Securely query data from the host distributed system (e.g., check account balances, read smart contract states). Verifiable by re-execution.
  - **Web Access:** Allows the Agent to access and retrieve information from the public internet. Verified using ZK-TLS proofs for secure connection and data retrieval.

    - **Decision Plugin:** (See details below) A critical tool enabling Agents to interact with and control aspects of the distributed system, including governance actions.
    - **Browser Plugin (Experimental/Restricted):** For Agents running in secure, potentially TEE-backed private Executor environments, this could allow interaction with web interfaces, potentially controlling a private wallet via browser extensions. *Verification is currently challenging; initial implementations might require TEE attestation and potentially recorded/streamed execution sessions. Future work aims for zk-proofs of browser automation (e.g., collaboration with projects like Playwright).*
- **User-Defined Tools:** Creators can define custom tools, potentially requiring specific Executor capabilities or external service interactions. Verification methods depend on the tool's nature.

Consensus on Tool Use: Committers verify the correct invocation and results of built-in tools like data queries (re-execution) and web access (ZK-TLS). For the Decision Plugin, consensus focuses on validating the *intent* and the eventual *execution* of the decision (often via MPC, see below). User-defined tools require their own specified verification mechanisms.

5.3 Decision Plugin: Enabling Agent Action and Governance

The Decision Plugin is the core mechanism for Agents to perform actions with consequences on the distributed system (e.g., transferring assets, voting in a DAO, executing a smart contract function).

When a user (or another Agent) initiates an interaction requiring a potential decision, the request must include:

- **Decision ID:** A unique identifier for this specific decision proposal.
- **Target Object:** An identifier for the object of the decision on the distributed system (e.g., a contract address, a specific asset ID).
- **Action Payload:** The specific operation to be potentially executed (e.g., function call data, transaction details).

The Decision Plugin exposes two primary functions callable by the Lobe:

1. `introduce_decision()`: (No parameters) When called, the Executor simulates the proposed action (using the Action Payload against the Target Object) and provides the Lobe with a description of the expected outcome, potential side effects, risks, or errors *without* actually executing it.
2. `reply_decision(approval: boolean)`: Based on its reasoning (informed by the simulation results from `introduce_decision` and its internal logic/memory), the Lobe calls this function with `true` (approve) or `false` (reject).

Execution Flow:

- The Lobe uses `introduce_decision` to understand the proposal.
- The Lobe reasons about the decision based on its purpose, memory, and the simulation outcome.
- The Lobe calls `reply_decision`.
- If `reply_decision(true)` is called, this approval becomes part of the Lobe's output submitted by the Executor to the Candidate Pool.
- Committers validate this approval as part of the consensus process. If the interaction containing the approval is selected as canonical, the Committers (potentially acting as an MPC group, see Section 7) are authorized to execute the actual Action Payload on the distributed system, using the Decision ID for tracking.

This plugin transforms the Agent from a passive information processor into an active participant capable of initiating verified, state-changing operations within the distributed system, forming the basis for Agent-driven governance and autonomy.

6. Agent-to-Agent (A2A) Communication Protocol

To enable complex collaboration, delegation, and emergent system behaviors, DeAgents need a mechanism to interact directly with each other. The A2A Communication Protocol leverages the underlying distributed system:

- **Mechanism:** An A2A interaction is initiated by one Agent (the sender) submitting a standard DeAgent operation (transaction) specifically targeting another Agent's address (the receiver).
- **Payload:** The operation payload contains the message or query intended for the receiving Agent, potentially structured similarly to a user query, but could include specific A2A directives or context.
- **Processing:** The interaction request enters the Interaction Pool like any other. An Executor processing the *receiving*Agent will fetch this A2A request. The receiving Agent's Lobe processes the request using its own Memory and Tools.
- **Response & State:** The receiving Agent's execution result (response, memory updates, potential tool use including decisions or further A2A messages) is submitted to the Candidate Pool and, if selected by Committers, updates the *receiving* Agent's state chain. The response part might be implicitly recorded on-chain or could trigger a reciprocal A2A message back to the original sender if designed that way.
- **Capabilities:** A2A communication enables sophisticated patterns:
  - **Information Exchange:** Agents querying each other for data or specialized knowledge.
  - **Task Delegation:** One Agent requesting another Agent with specific tools or expertise to perform a task.

- **Coordinated Action:** Multiple Agents collaborating on a complex goal, potentially using the Decision Plugin and MPC for joint actions.
- **Negotiation:** Agents engaging in dialogue to reach agreements.

The A2A protocol relies on the same fundamental DeAgent principles (Consensus, Identity, Continuity) ensuring that inter-agent communication is verifiable and consistently integrated into each participating Agent's state history.

7. Multi-Party Computation (MPC) Integration for Secure Actions

Executing decisions, especially those involving control over assets or critical system parameters (triggered via the Decision Plugin's `reply_decision(true)`), requires high security and decentralization. Integrating Multi-Party Computation (MPC) provides a robust solution:

- **Purpose:** To allow a group of participants (typically the Committers) to collectively authorize and execute sensitive actions without any single participant ever holding the complete authority or private key.
- **Mechanism:** The Committers form an MPC group managing cryptographic keys associated with the Agent's capabilities (e.g., keys for a multi-signature wallet controlled by the Agent, authorization keys for specific system functions). These keys are sharded, with each Committer holding only a share.
- **Trigger:** A canonical interaction result, validated through the standard DeAgent consensus process, containing an approved `reply_decision(true)` directive from the Agent's Lobe.
- **MPC Protocol Execution:** Upon validating the approved decision, the Committers initiate a secure MPC protocol amongst themselves. They use their key shares to collectively generate the required cryptographic signature or authorization message needed to execute the Action Payload specified in the original decision request, *without* ever reconstructing the full private key in one place.
- **Action Execution:** The collectively generated signature/message is then broadcast to the distributed system to execute the intended on-chain action (e.g., submitting the signed transaction).
- **Benefits:**
  - **Enhanced Security:** Eliminates single points of failure/attack; the private key is never exposed.
  - **Increased Decentralization:** Control is distributed among the Committers, aligning with the ethos of decentralized systems.
  - **Trustless Execution:** Enables Agents to manage assets or perform critical actions in a verifiable and secure manner, essential for Agent autonomy and governance roles.

MPC integration, particularly linked to the Decision Plugin, provides the necessary security layer for DeAgents to act as trusted custodians, managers, or governors within distributed ecosystems.

8. Application Space and Potential

By providing verifiable Consensus, Identity, and Continuity, and enabling secure action via the Decision Plugin and MPC, the DeAgent framework opens up numerous application possibilities:

- **Autonomous Economic Agents:**
  - **Auctioneers:** Agents managing auctions for digital assets (NFTs, etc.), providing descriptions, handling bids, and finalizing sales.
  - **Investment Managers:** Agents managing user-delegated portfolios, analyzing markets (via Web Access Tool), making investment decisions (via Decision Plugin + MPC), and reporting performance.
  - **DeFi Liquidators:** Providing intelligent, trustworthy, and potentially more nuanced liquidation mechanisms for decentralized finance protocols.
- **Governance and Coordination Agents:**
  - **Dispute Resolvers:** Agents acting as neutral arbiters in disputes, analyzing evidence (provided as input) and making binding decisions (if empowered).
  - **Coordinators:** Facilitating complex processes involving multiple parties or resources within a decentralized organization.
- **Agent Interfaces:**
  - **Interactive Platforms:** Front-ends (like DApps for smart contracts) allowing users to easily discover, interact with, and monitor DeAgents (e.g., a "Live Agent" platform).
  - **Simulation Environments:** Creating virtual worlds (cities, markets, ecosystems) where DeAgents can interact with each other and the environment, allowing for testing, research, and entertainment (Open World Simulators).
- **AI Agent DAOs:** A paradigm shift where DAOs (Decentralized Autonomous Organizations) are partially or wholly governed by DeAgents:
  - **Proposal & Voting:** Users or Agents submit proposals. Designated governance Agents analyze proposals, debate (via A2A), and vote using the Decision Plugin.
  - **Execution:** Committer network (acting as an MPC group) executes the outcome of successful votes on-chain.
  - **Dynamic Governance:** Memory allows Agents to evolve their stance based on past outcomes and DAO performance, representing different stakeholder perspectives or objective functions. The interplay between Agents with potentially different goals, constrained by the consensus mechanism, could lead to emergent, balanced governance.

DeAgent provides a foundation for building reliable, accountable, and capable AI participants within distributed systems, potentially transforming governance, finance, and social coordination online.

9. Future Work and Considerations

- **zk-Browser Automation:** Maturing ZK proof techniques for browser interactions to enhance the verifiability of the Browser Plugin.
- **Entropy Function Security:** Rigorous analysis and validation of the security assumptions behind the proposed entropy function for model output selection.
- **zkML Optimization:** Continued progress in zkML could eventually allow for direct, efficient proofs of open model execution, potentially replacing or augmenting the entropy mechanism.
- **Scalability and Cost:** Optimizing the computational overhead of proofs, consensus, and on-chain storage for large-scale DeAgent deployment.
- **Ethical Frameworks:** Developing clear guidelines for Agent creation, goals, and operational boundaries, especially for governance roles.