DEVELOPING AND MEASURING AI PLURALISM: STRATEGIES, MILESTONES, AND INNOVATIVE APPROACHES *

Devin Gonier
Columbia University
devin.gonier@university.edu

ABSTRACT

This paper presents a comprehensive overview of the development strategies and milestones necessary for realizing AI Pluralism...

***K*eywords** AI Pluralism · AGI · ASI · Multi-Agent Systems · Knowledge Graphs

1 Introduction

AI Pluralism represents a pivotal shift in the development of artificial intelligence, offering a pathway towards more individualized and adaptable AI agents. We define AI Pluralism as:

> *Multi-Agent frameworks in which A.I. agents interact with other human or AI agents, each of whom are diverse in perspectives and positions producing complex ecosystem of thought and recursive strategic refinement, as is manifested in evolution and culture for humanity.*

Many researchers have begun to explore how multi-agent systems might be configured using advanced LLMs. This transition from "mono-AI" to pluralistic AI represents an important paradigmatic shift in how one ought to approach doing AI research. Furthermore, it opens up the opportunities for other humanities based disciplines such as rhetoric, philosophy, game theory, anthropology, sociology, and psychology to contribute to the development of AI systems. The goal of this paper is to persuade the reader that not only is AI Pluralism valuable, but that it is also viable with the technology available today, and that with the right configuration it is possible to study such systems from many different perspectives.

AI Pluralism is a significant area of research because it offers great potential in alleviating many thorny issues in A.I. as outlined below.A

- **Safety** AI Pluralism can help to make AI systems safer by providing a diverse set of perspectives and positions and systems of self-accountability in which the ecosystem creates incentives for agents to keep other agents in line with ethical norms, much as trading partnerships enhance and stabilize peace between nations.
- **Bias and Fairness:** AI Pluralism can help to mitigate bias in AI systems by providing a diverse set of perspectives and positions. This can help to ensure that AI systems are fair and equitable.
- **Explainability:** AI Pluralism can help to make AI systems more explainable by tracking dialogical patterns in thinking as opposed to being restricted to opaque assessments of the systems internal state. The act of AI community building by its nature brings the internal outwards into a dialogical framework of external problem solving and contemplation.
- **Robustness:** AI Pluralism can help to make AI systems more robust by providing a diverse array of roles and abilities that enable coverage of weak spots through oversight. Much as the police make it their goal to identify and prosecute intruders, its possible to envision a system in which certain AI agents monitor the health and status of various systems.

* *Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

- **Adaptability:** AI Pluralism can help to make AI systems more adaptable by providing a diverse set of perspectives and positions. This can help to ensure that AI systems are able to respond to changing circumstances and environments.

This paper argues has three main contentions. First, that the study of AI Pluralism is necessary for the pursuit of AGI and ASI. Second, AI Pluralism enables beneficial properties that are not possible with mono-AI systems. Third, AI Pluralism is a viable with technologies available today with the right setup, generative language mechanics, and retrieval systems.

The remainder of the paper consists of a deeper exploration of three components of AI Pluralism - the retrieval systems, the language generation architecture, and the situational context that are necessary for AI Pluralism to be viable. It then explores some specific experimental implementations of the proposed research and the current results of that research. Then, we discusses a broader framework for conceptualizing the relationship between AI pluralism and consciousness. Finally, we outline milestones for future research and conclude with remarks on how to accelerate development in this area.

2 Related Work

2.1 Agent Architectures and Their Scaling Properties

The scalability of agent architectures is a pivotal aspect of modern AI systems, influencing their applicability across a broad range of tasks. Masterman et al. [**?**] provide a detailed survey on the evolution of AI agent architectures, emphasizing the distinction between single-agent and multi-agent systems. They highlight that while single-agent architectures are simpler and more manageable, multi-agent systems exhibit superior performance on complex tasks due to their collaborative dynamics, which allow for effective parallelization and a more dynamic division of labor.

Li et al. [**?**] demonstrate that the performance of large language models can significantly benefit from scaling the number of agents involved. This method, which uses a simple sampling-and-voting mechanism among multiple agents, shows that increasing agent count can directly enhance the system's ability to tackle complex tasks by leveraging diverse perspectives and capabilities inherent in a multi-agent setup.

In exploring the integration of graph-based learning within agent systems, Hu et al. [**?**] introduce a framework that enhances the scalability of AI systems in processing graph-structured data. Their approach uses knowledge distillation from large language models to graph neural networks, facilitating an effective scale-up in handling relational data structures, which is critical for tasks that involve complex relationships and dependencies.

Furthermore, Wang et al. [**?**] explore the concept of cognitive synergy within a single agent framework. Their development of the Solo Performance Prompting method simulates a multi-agent environment within a single LLM, effectively scaling the cognitive capabilities of the model to improve performance on complex problem-solving tasks.

Each of these studies underscores different strategies for scaling AI agent systems, from enhancing collaborative dynamics to leveraging advanced graph processing techniques, all contributing to the field's understanding of how to effectively scale agent architectures to meet the demands of diverse applications.

2.2 Enhancements Through Collaboration

Collaboration among agents in AI systems has emerged as a powerful strategy to enhance the problem-solving capabilities and accuracy of these systems. Two pioneering approaches, cognitive synergy through solo performance prompting and the multi-agent debate, have been explored to leverage collaborative dynamics effectively. These methods advocate for novel frameworks that enhance both the reasoning depth and factual accuracy of AI agents.

Wang et al. [**?**] introduce the concept of Solo Performance Prompting (SPP), a method where a single Large Language Model (LLM) simulates the interaction of multiple personas to enhance its problem-solving capabilities. This approach utilizes cognitive synergy, where simulated collaborative interactions within the model mimic the dynamics of multi-agent systems. The key innovation here is the model's ability to dynamically assign roles and engage in self-collaboration, which allows the LLM to process complex tasks that involve diverse areas of knowledge more effectively. The authors advocate for this method as it combines the simplicity of managing a single model with the benefits of multi-agent interactions, enhancing adaptability and depth in task-solving without the overhead of coordinating multiple independent agents.

In a similar vein, Du et al. [**?**] delve into the potential of multi-agent debate to refine the reasoning and factuality of responses generated by LLMs. Their framework involves multiple agents that propose, critique, and iteratively refine

their answers through a structured debate format. This process allows the agents to collaboratively enhance the accuracy and logical consistency of their collective outputs. By engaging in a critique and refinement cycle, the agents are better equipped to identify and correct errors, reducing the incidence of erroneous or hallucinated content. The authors champion this debate-based approach as it effectively leverages the collaborative input of multiple agents to achieve a higher standard of reliability and depth in the model's output, particularly in complex reasoning and fact-checking tasks.

These studies collectively underscore the significant advantages of integrating collaborative mechanisms within AI systems. By simulating or actualizing multi-agent interactions, both cognitive synergy and structured debates provide robust frameworks for enhancing the capabilities of AI agents. These enhancements are crucial for developing more reliable, accurate, and adaptable AI systems, pointing towards a promising direction for future AI research and applications in complex environments.

2.3 Graph-Based Learning in Agent Systems

Graph-based learning has become increasingly significant in agent systems, particularly with the integration of Graph Neural Networks (GNNs) which have proven to be instrumental in enhancing agent capabilities in complex environments. The works of Hu et al. [**?**] and Wu et al. [**?**] offer profound insights into the application of GNNs within AI systems, demonstrating how these networks facilitate advanced data processing and decision-making capabilities.

Hu et al. [**?**] introduce a novel concept of "Graph Knowledge Distillation," which aims to leverage the advanced semantic processing capabilities of Large Language Models (LLMs) to enhance the performance of GNNs on node classification tasks. By distilling knowledge from LLMs, GNNs can inherit nuanced understanding and reasoning abilities, making them more effective in handling complex graph-structured data. This approach not only bridges the gap between traditional graph processing techniques and modern neural language models but also optimizes the learning process by ensuring that GNNs can perform with a higher degree of accuracy and efficiency in real-world applications.

On the foundational side, Wu et al. [**?**] provide a comprehensive review of various GNN architectures and their practical applications across different domains. Their discussion encompasses the mechanisms and training frameworks that enable GNNs to produce node-level, edge-level, and graph-level outputs, essential for tasks ranging from social network analysis to biochemical molecule interaction. This foundational work is crucial as it lays the groundwork for understanding how GNNs can be effectively implemented in agent systems, enhancing their ability to process and analyze large-scale relational data.

Together, these studies illustrate the dynamic capabilities of graph-based learning within agent systems. By integrating GNNs with LLMs through innovative approaches like knowledge distillation, and by building on solid foundational GNN techniques, agent systems can achieve remarkable improvements in processing efficiency and decision-making accuracy. These enhancements are pivotal for the development of AI systems capable of navigating and operating within intricately connected data environments.

2.4 Model Merging and Evolution

Model merging, enhanced through evolutionary algorithms, represents a significant advancement in the development of foundation models. In the work by Akiba et al. [**?**], the authors present a novel application of evolutionary algorithms to automate the creation of powerful foundation models through the merging of diverse model architectures. This approach leverages the collective intelligence of existing models, optimizing across both parameter space and data flow space to produce models with capabilities that exceed those achievable through traditional model development methods.

The methodology introduced by Akiba et al. operates without the need for extensive additional training data or compute resources, facilitating efficient model development. Their evolutionary approach has successfully generated models like a Japanese LLM with math reasoning capabilities and a culturally-aware Japanese VLM, both achieving state-of-the-art performance on established benchmarks. These results not only demonstrate the effectiveness of the method in creating competitive models but also highlight its potential to revolutionize foundation model development by reducing reliance on resource-intensive training processes.

The implications of this research are profound, suggesting a shift towards more sustainable and efficient AI development practices. By automating model composition and enabling cross-domain merging, evolutionary algorithms provide a pathway to rapidly advancing the capabilities of AI systems while mitigating the costs and resources typically associated with such developments. This paradigm shift could lead to more rapid dissemination and democratization of advanced AI technologies.

2.5 Evaluation and Optimization of Multi-Agent Systems

Evaluating and optimizing multi-agent systems has been a focus of recent advancements in AI, particularly in enhancing the collaborative capabilities of Large Language Models (LLMs) within these frameworks. The works by Li et al. [**?**] and Du et al. [**?**] provide significant insights into methodologies that leverage the strengths of multi-agent setups to optimize performance and improve factual accuracy and reasoning.

Li et al. [**?**] explore the performance scalability of LLMs when configured as agents in a multi-agent system. Their research introduces a simple yet effective sampling-and-voting mechanism, illustrating how increasing the number of agents can significantly enhance the system's overall performance. This approach not only shows that smaller models can achieve competitive results when used in larger numbers but also highlights the potential of multi-agent systems to solve more complex problems through collective intelligence and distributed processing.

On the other hand, Du et al. [**?**] delve into the optimization of reasoning and factual accuracy through a structured multi-agent debate format. In this framework, agents engage in rounds of debate over the outputs they generate, critiquing and refining their responses based on feedback from other agents. This process not only reduces the incidence of incorrect or hallucinated information but also strengthens the agents' capacity for critical thinking and deep reasoning, making the system more robust and reliable.

Together, these studies underscore the effectiveness of using multi-agent systems not only for scaling AI capabilities but also for refining these capabilities through collaborative and competitive dynamics. By leveraging the collective strengths and diverse perspectives of multiple agents, these systems become more adept at handling complex, nuanced tasks, pushing the boundaries of what AI can achieve in collaborative environments.

2.6 Contributions to AI Pluralism

The integration of pluralistic human values into AI systems is an emerging focus in the field, aiming to ensure that these systems are both ethically aligned and socially inclusive. Significant contributions by Häußermann and Lütge [**?**] and Sorensen et al. [**?**] explore the implementation of diverse human values into AI decision-making, highlighting a paradigm shift towards more responsible and inclusive AI technologies.

Häußermann and Lütge [**?**] discuss the concept of "Community-in-the-loop," which integrates business ethics into AI development. Their approach emphasizes the importance of including diverse societal groups in the AI development process, thereby ensuring that the technology is guided by a broad spectrum of human values and ethical standards. They propose a deliberative order ethics framework, where stakeholders actively participate in resolving value conflicts and setting guidelines for AI use, promoting a more equitable and balanced development of AI technologies.

On a more practical note, Sorensen et al. [**?**] introduce 'ValuePrism', a comprehensive dataset designed to embed a wide array of human values into AI systems. Utilizing this dataset, they develop the 'Kaleido' model, which not only generates and explains human values but also assesses their relevance in different scenarios. This model enhances AI's capability to understand and reflect complex value judgments, making AI systems more adaptive and sensitive to the ethical dimensions of human decisions.

These studies collectively advance the discourse on AI pluralism, demonstrating innovative approaches to incorporating ethical considerations and human values directly into AI systems. By fostering a more inclusive approach to AI development, these contributions aim to ensure that AI technologies serve a broader range of human interests and uphold fundamental ethical principles.

3 Building Multi Agent Frameworks

3.1 Introduction to State and Component Management in JavaScript Frameworks

JavaScript frameworks like Vue.js and React have revolutionized web development through their effective use of global and component-specific states. Global state refers to data that is shared across the entire application, accessible from any component within the system. This is invaluable for user-specific data that needs to be consistent and available throughout the application, such as user authentication status, theme settings, or multi-view data caching.

In contrast, component state is localized and pertains only to a specific component or module within the application. This encapsulation allows each component to manage its own state independently of others, leading to better modularity and reusability. Components handle their state based on their specific needs and responsibilities, which enhances the robustness and maintainability of the application by isolating state management to relevant parts of the interface.

A component in JavaScript frameworks is essentially a self-contained unit that encapsulates a portion of the user interface and the logic that drives it. Components are the building blocks of modern web applications, each responsible for rendering a part of the application's UI and handling user interactions. The value of using components lies in their ability to be reused across different parts of an application without repeating code. This modularity not only speeds up development time but also improves the consistency and quality of the application. Components can be as small as a button or as large as an entire application section, each with its own lifecycle, state, and methods.

The dynamic nature of JavaScript frameworks allows components to be rendered and re-rendered in response to user actions or changes in data. This reactivity is a core feature of frameworks like React and Vue.js, where the DOM (Document Object Model) is updated efficiently to reflect changes in component state or global state without reloading the entire page. For example, when a user interacts with a form, only the components related to that form might update, rather than the entire page. This selective re-rendering optimizes performance and enhances user experience by providing immediate feedback in response to user interactions.

JavaScript frameworks leverage a virtual DOM to manage these updates, which calculates the minimal set of changes needed to update the actual DOM based on state changes in components. This process is highly efficient and reduces the amount of work needed to keep the UI in sync with the underlying application state. By only updating components that need to reflect changes, the frameworks ensure that the web applications remain fast and responsive, even as they scale in complexity and size.

3.2 Introduction to Multi-Agent Frameworks

In multi-agent frameworks, the concept of state is intricately layered to accommodate the complexities of interactions among multiple agents. The state is divided into three main layers: private, group, and public. The *private state* of an agent includes its unique history, personal data, and internal strategies that are inaccessible to other agents. This encapsulation ensures that each agent maintains a level of individuality and privacy necessary for independent decision-making.

The *group state*, shared among a defined subset of agents such as a team or tribe, encompasses strategies, shared goals, or collective memories that support coordination and cooperative behaviors among the agents. This layer is crucial for tasks that require collaboration, allowing agents to operate with a shared understanding and common objectives.

Lastly, the *public state* involves the overarching game or environment rules that all agents interact with. It includes the definition of the environment and any universal constraints or goals that affect all participants. This layer is analogous to the global state in JavaScript frameworks, where it provides a common ground for all components—here, the agents—to interact within.

Components in multi-agent frameworks are analogous to tasks that agents need to perform. Each component encapsulates a specific task, equipped with the necessary tools and prompts required for execution. Components can be dynamically rendered according to the agents' needs, similar to how web components are rendered in response to user interactions. However, instead of a Document Object Model (DOM), these components are organized in a Directed Acyclic Graph (DAG), which allows for complex interdependencies and efficient task management among agents.

Moreover, components can have model overrides allowing them to adapt or specialize based on the task's requirements or the agent's capabilities. This flexibility enables the system to tailor task execution strategies to the specific needs of the scenario or the strengths of the agent, enhancing effectiveness and efficiency.

Agents in a multi-agent system may be defined by specific models, private memories, and distinct strategies or beliefs. What makes each agent unique is not just its private data but also its ability to apply its capabilities in ways that significantly contribute to the collective objectives. The uniqueness of each agent is vital as it allows for a more diverse range of strategies and solutions to the tasks at hand, mirroring the diversity found in human team settings.

Each agent's contribution is optimized through the effective use of state layering and component-based task management, ensuring that the collective behavior of the system is more than the sum of its parts. By leveraging the unique attributes and capabilities of each agent, multi-agent systems can achieve higher levels of complexity and adaptability, essential for navigating dynamic and unpredictable environments.

4 The Retrieval Architecture

When humans think, there are at least two core processes involved. The first is the retrieval of information in memory, whether it be memories of events, concepts, or facts. What is retrieved must then be coordinated with the current context and objectives within that context, to produce thought as typically represented both internally (and often externally) in

the form of language. This section outlines an expanded principle of retrieval in which features of identity or long-term states are retrieved and continuously incorporated into language generation. Current retrieval mechanisms such as RAG are useful for citing or justifying responses and mitigating hallucinations. However, databases can be used to do more than retrieve information, they can also be used to manage the state of the agent by defining and updating mechanisms such as beliefs, and interpersonal relationships.

The first step in creating the retrieval system in the Multi-Agent Framework (MAF) is to identify the types of retrieval nodes we want to incorporate. In most RAG approaches, there is functionally one type of node – the document or the passage. These are then incorporated into messages or in some cases system messages to be added into the attention window of the model. This brings the outside content into the context so that the model has a reference point beyond the trained weights to generate its answer. In our case retrieved information, goes into the agents *(*state). The type of node helps determine which state (public, private, or semi private) to load the variables into. For our purposes we identify four main kinds of nodes, which can be retrieved to update state. As you will see in the next section, the models must be fine-tuned to better enable them to interpret the effects of state variables. All nodes are stored in an elastic cache index on AWS' Opensearch platform. Lookup occurs via KNN search in a shared topic oriented latent space. We architecturally prefer having more smaller indices than fewer larger indices. This enables for more refined fine-graned control of the access constraints, and topicality of the information being stored.

4.1 Types of Retrieval Nodes

4.1.1 Knowledge Node

Knowledge nodes are always loaded into public states. Every knowledge node has a primary coordinate embedding, an adjacency matrix, and basic content information stored as a data object. Knowledge nodes are broken down into subtypes as follows:

Documents - A document is a collection of sections and basic citation information. The collection of sections is organized as a hash table representation of an adjacency matrix. Its embedding is the average of its consituent section embeddings. When loaded into the DocumentNode class it has a variety of useful utilities, for searching within a document, or finding nearby passages.

Sections - A section is a collection of passages. It is adjacent to documents as a child. Its children are passages. It is also embedded as an average of passages and has its own utilities.

Passages - A passage is a chunk of text, an embedding, and an adjacency matrix. We have found llmSherpa (citation needed) to be a useful resource in generating meaningful chunks. The passage can function as a generator with pointers to previous passages and next passages. Furthermore, there can be subtypes of passages, with further utilites. For example, in the debate language game having passages with underlined portions is useful, so there is a subtype called cards - which are essentially passages that have added highlighting utilities.

4.1.2 Belief Node

- A belief node is a variation on a knowledge node which carries the additional important property of *credence*. Credence is a measure of how strong the agent believes the information to be true. The scale is always from 0 to 1, where 0 is a contradiction, and 1 is a tautology. In other words, basic principles of Bayesian Epistemology hold for the credence of a belief. Beliefs also have adjacency matrices coordinating them with other beliefs and knowledge nodes, which are treated as evidence either in support of or in negation of a particular belief. Beliefs are always loaded into the *private state* of an agent. There are two subtypes of beliefs

Value - A value a statement of preference that can be used for decision making. Values can be moral in nature, but can also be practical. For example, an agent might have a value that says "I prefer to be honest" or "I prefer to be efficient". These values can be used to guide the agent in decision making. The higer the credence of a value, the more likely the agent is to act in accordance with it.

Hypothesis - A hypothesis is an uncertain statement of fact, which has evidence to support or negate it stored via an adjacency matrix. Its credence value is mathematically calculated according to prior beliefs (contingent beliefs) and the evidence available. When beliefs get modified they have the potential to change the credence of a downstream hypothesis. Hypotheses are used to guide the agent in its reasoning and decision making.

4.1.3 Memory Node

A memory node is a useful structure for keeping track of relevant action history in a more explicity manner. Rather than tracking all messages, memory nodes are used to store and track messages according to relevance. An agent can recall

past actions and outcomes using memory nodes. Memory nodes are also correlated with beliefs. They can be used as causal evidence. E.g. If action X led to state Y, then inductively we know that X can be a cause of Y. The more frequent the association is found, the stronger the causal link. Memory nodes can be stored at the individual level in private, but also at the tribal level in semi-private.

4.1.4 Relationship Node

- The last type of node keeps track of agent and human relationships. There are two subtypes of relationships nodes. The relation node and the object node. Relationship Nodes are used to deal with the problem of other minds and to anticipate particular outcomes. They are trackedd inductively via experience, and are always kept private. This means that different agents have different perspectives on how relationships are going. In the context of agent-agent interactions, relationship nodes can be useful for identifying specialization. If a particular agent utilizing a particular model or MoE shows more promise than others at performing a task, then agents who identify this may be more likely to call upon that agent to perform the task in later episodes.

Relation Node - A relation node is a relationship between two agents. It has an adjacency matrix that keeps track of the history of interactions between the two agents. It also has a credence value that indicates how strong the relationship is. The credence value is calculated based on the frequency and quality of interactions between the two agents. The higher the credence value, the stronger the relationship. In example of a relation, may be "Trust", "Teaches". Relations are added lazily as agents interact with each other.

Object Node - An object node represents the agents themselves. It has an adjacency matrix to other objects using relations as a coordinate.

4.2 Dynamically Adding Nodes to the Retrieval System

Generally speaking, nodes are added dynamically and lazily. This means that nodes are only added to the retrieval system when they are needed. For example, if an agent is seeking to find evidence or literature about open souce AI technology, it calls a method that first attempts to search the relevant index for nodes that currently exist. If the most proximate node as measured via an query embedding cosine similarity function, is above a threshold T, then the nodes get returned as output and loaded into the agents state. If no nodes exist above threshold T, then a series of API calls get made to various information sources like Google, Arxiv and others to find relevant articles on the subject. These articles are parsed and added to the retrieval network.

Belief nodes are also created lazily by the agent. Typically via debate or inter-agent disagreements. As agents advocate for particular positions, they update or create corresponding belief nodes. When they attempt to justify their reasoning using evidence, the evidence gets associated by adjacency matrices to the belief node. This is done in a way that is similar to how a human might justify a belief. If a belief is contradicted by evidence, then the credence of the belief is lowered. If the belief is supported by evidence, then the credence of the belief is raised. This is done in a way that is similar to how a human might justify a belief. Similarly, as events happen memory nodes get added and as new dialogs with other agents or users occur then relationship nodes get added.

4.3 Retrieval Nodes and Adjacency Matrices: Nodes that Fire Together Wire Together

The motivation behind adjacency matrices, beyond their obvious computational benefit, is to create a system that develops richer interconnections over time via experience. Proximity is typically measured via cosine similarity against a query. But it may often be the case that the user wants contradictory information, or other peices of associated information that are connected via edges stored in an adjacency matrix. This edge representation is scalar (not just boolean) in value enabling sophisticated learned relationships over time. Furthermore, each node has a parameter theta, which corresponds to a weight matrix, that can be multipled by the adjacency matrix depending on the operation being performed. Initially, theta is just set to the identity matrix, but as data is accumulated the GNN can be fine-tuned to do a better job of retrieving associated nodes.

5 The Generative Architecture

The generative architecture is designed to optimize two specific constraints currently encountered by LLM agents. First, we need ways to better individuate agents, by incorporating private data into states, such that outputs directly reflect the perspective of the agent. Failure to successfully accomplish this leads to less pluralistic communities, which can degrade the ultimate quality of the output. Second, we need a way to enable *community evolution*. Ideally, agents who function in a knowledge ecology are able to self-optimize through reproduction with other agents. In other words,

evolutionary algorithms are used to merge model weights between agents. The fitness of the agent as understood via evolution is measured against the tasks success.

5.1 Agent State and its Motivation

A major breakthrough occurred when OpenAi transitioned from GPT-3 to chat-gpt. The introduction of assistants, the message structure, and the system message as a component applied an advanced model like GPT-3 at the time fundamentally changed the nature of how models were interacting with users. The idea was quite simple, not all information serves the same equal purpose. In a dialog its critical to keep track of who is speaking what, and to include system prompts to help facilitate the role the assistant should be playing in the dialog.

Similarly, we propose rethinking the nature of information being passed in prompting. We hope to fine-tune and in some cases reconstruct a models ability to incoporate beliefs, knowledge, relationships, and memory into its outputs. The strategy is to use Direct Preference Optimization or Odds Ration Preference Optimization to train the model to output different responses based on provided belief or knowledge states. Eventually, the model will begin to form a kind of identity that can be loaded and easily diversified based on the state of the agent as retrieved from the retrieval system. The goal is to compress the information needed to individuate an agent into a representation that can be retrieved and loaded into a model's system state.

5.2 Evolutionary Model Merging Mixing

Recently developers have begun to get access to a wide variety of open-source foundation language models of varying size and trained on differing datasets. Developers were then able to work with these models to do three main things. First, developers were able to fine-tune these models using transfer learning and Lora Adapters. This enhancement enables engineers to make a general purpose model specialize in a particular use case. Moreover, the way in which PEFT models are trained, its possible to take a single general model and build a variet of adapters which can be theoretically loaded on the fly. Secondly, developers began mixing models using the Mixture of Experts (MoE) approach. This strategy enabled models to have access to a much larger set of parameters with less compute (VRAM is still a constraint). MoE models uses router layers, to learn which "expert" models to use given a prompt. The router picks a subset of the available models to pass the input through, thereby allowing dynamic inference according to each prompt. Thirdly, A.I. engineers are *merging* models, using tools like mergekit (citation needed). These tools enable engineers to apply different algorithms to combine layers from one or several models together. One can even do a combination of mixing and merging to produce completely unique models without incuing the compute cost of training.

6 The Situation

Language games, as conceived by Ludwig Wittgenstein, describe the various forms of language used in different contexts that follow specific rules. In the context of artificial intelligence, language games can serve as structured frameworks for agents to interact and evolve through language-based tasks. These games are pivotal in advancing language understanding and generation capabilities of AI systems. Wittgenstein pointed out that language is always contextualized, and is ultimately meaningless outside of an applied context. We draw upon this observation, when incorporating game structures into the very dynamics of AI agent training. How do agents coopoerate, compete, and coordinate with one another? How do they use language as internal mechanisms as well as outward responses.

In our particular use case, language games refer to open-ended multi-agent games with winners, losers, and other mechanisms of scoring, that require strategic, concise, and effective communication to be successful. Ideally, agents that are very successful in such games, will also be successful in other similar contexts. For example, an agent(s) with strong debating skills, is also likely to give good balanced advice that represents multiple perspectives.

The use of games and specifically language games is not new to A.I. research. It involves the intersection of language theory, computer science, and cognitive psychology, providing a medium through which agents can learn and reason. Notably, Alan Turing's proposal of the Turing Test as a measure of machine intelligence hinges on the machine's ability to use language indistinguishably from a human, which underscores the centrality of language in AI development. Turing was right to identify the game structure as a powerful medium for evaluating progress in machine intelligence.

6.1 Examples of Language Games

Language games in AI can vary widely, each serving different purposes from training models to testing their capabilities:

- **Persuasion Party:** Agents attempt to persuade one another within a set of rules, aiming to win over others to their point of view.
- **Murder Mystery:** Agents work collaboratively to solve a fictional crime, using deductive reasoning and dialogue.
- **Guess Who:** Agents ask questions about a hidden person's beliefs and relationships, and focus on identifying who the hidden entity is.
- **Debate (Policy, Lincoln-Douglas, etc.):** Structured formats where agents argue for or against specific positions, simulating a competitive debate environment.
- **20 Questions:** One agent thinks of an object, and another asks up to 20 questions to guess it, showcasing the agent's ability to ask informative questions and make deductions.

6.2 Incorporating Language Games as a Mechanism for Model Evolution

Integrating language games into AI development helps in several ways. It provides a controlled yet flexible environment to develop and refine AI capabilities. Through iterative gameplay, AI agents can improve their language understanding, strategic planning, and interaction with humans and other AI agents. These interactions are crucial for the evolution of AI, pushing the boundaries of what artificial agents can achieve in complex social interactions and decision-making scenarios.

7 Experimental Design

Our experimental design is based around studying how agents interact and cooperate in a language game context. Below is the basic setup for the experiment. We then provide our initial hypothesis for what to expect and what the results indicated.

7.1 Retrieval

Each agent in the system has tool based access to query an elasticsearch index to find materials that can help them answer whatever query or cirumstance they are encountering. These are broken down into Private, Semi-Private, and Public indexes.

Retrieval Accessibility

Private Index: This index is unique to each agent and contains information that is specific to that agent. This could be personal memories, or other information that is not shared with other agents. This private is also "inheritable" by future progeny of the agent.

Semi-Private Index: This index is shared with a subset of agents. This could be a group of agents that are working on a specific task, or agents that have a shared history. In the context of the situation we are studying agents form "tribes" and each tribe has a shared semi-private index.

Public Index: This index is shared with all agents in the system. This could be information that is common knowledge, or information that is shared with all agents.

Each index is further broken down into hierarchical categories and node types. The depth of each index is determined by the agent. Each agent is given access to functions which enable them to add documents to an index, or create a new index within an exising index. When querying, results can be a combination of documents and indexes. Thus, agents can choose to structure their index however they like in whatever organizational manner makes sense. There are some structural approaches we also intend on experimenting with where we remove some of the independent control of the agents. This includes basing index structure around node type, or document level. The document level follows the pattern of passage / passage-cluster / document / document-cluster / topic pattern. Since there are only three kinds of nodes in the index, its also worth exploring structuring the index according to node types: knowledge, beliefs, and relationships.

Node Attributes

Nodes in the index all have certain attributes, but there are some distinctions between them. In each index, there is a vector lookup which can be used with knn to construct clusters, and cosine similarity (or other similar measures) to find proximity between a query and index for neural semantic search, and finally the lookup vector can be used as a function

in the graph neural network that sits on top of the entire network. GNN components are introduced in later versions however.

Another attribute every node type has is a set of edges. Edges define relationships with other nodes. Since every node in an index has an id, each node can track an adjacency matrix of its relationship to other nodes. These adjacency matrices can be further organized according to type of relationship.

Indexes are stored in s3 for tracking purposes at the end of each generation, and those which are not inherited are removed from the ec2 instance hosting the elasticsearch.

Node Types

There are three types of nodes in the retrieval network.

Knowledge: These nodes contain information that is factual or historical. They can be used to answer questions, or provide context for a situation. They are primarily populated with evidence found from internet search or other apis and always have a source attached to them.

Beliefs: These nodes contain information that is subjective or personal. They can be used to express opinions as they relate to claims and evidence. The key distinguishing feature here is a **credence** variable that doesnt simply retrieve information, but also provides a believability score. This is updated through the principles of Bayesian Epistemology. Beliefs may also represent a strategy of position on a particular topic.

Relationships: These nodes contain information that is relational or social. They can be used to connect agents, or provide context for a situation.

Each group (private, public, semi-private) can contain their own diversity of nodes (knowledge, belief, relationships).

7.2 Generative Modifications and Evolutionary modifications

Retrieval is essentially half of the influence on individuality in AI agents, the other half is the architecture of the models themselves. We achieve diversity in three different ways:

Fine-Tuning: Each agent has some aspect of it that is fine-tuned on a different source dataset. This could include how to perform a particular action or tool, or it might include domain knowledge. For example, in the language game of debate, one agent might specialize in a particular form of debate called kritik debating. This agent would be fine-tuned on how to construct kritik arguments.

Weight Merging: Each agent can also be further differentiated by representing a combination of a subset of the total pool of smaller expert agents. This is accomplished by selecting a base agent, and choosing 3 or more other agents to "mix" the weights together in a unique way. This kind of mixing has further variation in terms of the algorithms that are used for the merging, as well the distribution of the weighted sums for where and which layers to merge.

Retrieved Initialized Belief Structure: Each agent is fine-tuned at some subset of layers to incorporate beliefs into the system prompts such that those beliefs directly inform outputs. Arguments are always presented in the first person. The statements reflect the belief structure. As evidence and viewpoints are exchanged, these belief structures may evolve and change. However, the inital state can be diversely set.

These three methods enable a wide variety of potential agents to populate the situation. As each new generation of the game proceeds, further model mixing and initialization get inherited, identifying which techniques were most successful at performing well in the game and eventually converging on a community dynamic that is optimal for the language game task at hand.

7.3 Setup – The Evolutionary Language Game

A language game is any game where players must use language in non-deterministic and open-ended ways to make "moves" that lead to a win condition. Games must resolve in clear winners and losers. Every game has a "GameMaster" agent which ensures the game is played properly by the agents and keeps track objectively from the outside the state of each player in the game. Here are a few language game examples:

20 Questions: Agents are limited to 20 yes/no questions to guess a person, place or thing.

Murder Mystery: Agents are given identities and encouraged to communicate with one another and identify who they think the murderer is.

Debate: Two teams debate against one another, following academic debate models like Policy, Lincoln Douglas, Parli Debate from American Forensics. The Gamemaster's role is to function as the judge. The judge consists of a panel, which is itself diversified but optimized to not be in the minority decision. Note: that judge adaptation is crucial to be successful in this model!

Systems of Equation: Different groups are given different equations and must interact and communicate to figure out the hidden variables in a higher order equation.

Persuasion Party All agents are initialized with a set of beliefs and corresponding sets of evidence. Each agent has slight differences from one another. The goal is to have the agents interact with one another and persuade an agent to change their belief to their side by looking at and evaluating evidence. If successful, the newly converted player can convert others to the same belief. Does the whole party converge on certain beliefs. Do some beliefs remain intransigent?

7.3.1 Tribal Politics

In the first round of the game each agent can make a choice to either a) start a new tribe and invite other players into it or b) join a tribe that has already been started. Actions in the game cannot be taken by individuals, but are instead voted on by members of the tribe with capital. Agents can name their tribe, remove players from tribes, and invite new members into their tribe, but they cannot grow their tribe beyond a certain size. This ensures that there is a certan number of tribes to be consistent with the game mechanics of the game. This adds an additional game-theoretic element where the game is both competitive and cooperative. The goal is for the tribe to eventually evolve into a specialized unit such that different players on the team take on certain niche expertise that help the tribe succeed overall.

7.3.2 Capital Allocation

Each agent is initialized with a certain amount of capital which they can spend according to which actions they think are best. When proposing an action for a round, they can allocate capital based on their confidence level in that action. For example, in debate this might be a proposed argument. In 20 questions it might be a question and a justification for why that question ought to be asked. After all players have submited their questions, the tribe has a second round of spending capital where they select which action of the other players they want to spend capital on. The action with the most capital wins and the action is taken by the tribe. Agents submit reasons for why they think their action ought to be taken, essentially persuading the group of why their move is best. The agent who proposed the action, may get some capital back depending on the game mechanics of the game. If a player loses all their capital before the end of the game, they are eliminated from the game.

7.3.3 Evolution and Generational Inheritance

After a generation (a set of rounds of game play) the top scoring players get to "mate." Mating in this context means merging model weights with other winners. The amount a player gets to mate is directly proportional to their score. So the top player gets to mate the most, and the bottom player doesnt get to mate at all. The next generation then is made up of the progeny of the previous generation.

Furthermore, after mating, children inherit the retrieval indexes of their parent as a starting point, and the parent has an opportunity to update their beliefs one more time with a set of strategies for their children to utilize when they play the game in the next round.

8 Results

Here, the outcomes of the experimental setup are presented, providing insights into the practical applications and implications of AI Pluralism based on the conducted experiments.

9 Discussion

The discussion section reflects on the results, considering the role of language in consciousness as debated by Dennett and Chomsky, and evaluates the impact of culture and future research directions in AI Pluralism.

10 Milestones

This part of the paper proposes a set of milestones for measuring progress in AI Pluralism, including the complexity of situations, progress in language games, and sophistication of retrieval mechanisms.

11 Conclusion

The conclusion summarizes the paper's contributions to AI Pluralism, highlighting the potential for more diverse, adaptable, and innovative AI systems and suggesting avenues for future research.

Acknowledgments

This work was supported in part by...