UNaIVERSE: A Peer-To-Peer Network For Human-AI Agents

UNAIVERSE: A PEER-TO-PEER NETWORK FOR HUMAN-AI AGENTS

Stefano Melacci, Christian Di Maio, Tommaso Guidi, Marco Gori
Collectionless AI Team - Siena Artificial Intelligence Lab, https://collectionless.ai/
DIISM, University of Siena – Siena, Italy
stefano.melacci@unisi.it, christian.dimaio@phd.unipi.it,
tommaso.guidi@unisi.it, marco.gori@unisi.it

ABSTRACT

We present UNaIVERSE (https://unaiverse.io/), a decentralized platform designed to implement a peer-to-peer model of the Web, based on human-AI agent communities. We detail the platform's architecture, design principles, communication protocols, and demonstrate its capabilities through multiple use cases that showcase distributed problem-solving and human-AI interaction. UNaIVERSE is aimed at addressing emerging limitations of the current Web, social networks, and Artificial Intelligence (AI) infrastructures. While these technologies have expanded global communication and knowledge exchange, they also raise concerns regarding privacy, concentration of control, and escalating energy demands. The coexistence of humans and AI further remains uncertain and largely unregulated, exposing risks that extend beyond technical boundaries. UNaIVERSE is designed around a shift from a Web of "document-like" resources offered via a client-server model to a network of agents (human and artificial) participating in a peer-to-peer network. Agents run on different types of devices, including energy-efficient edge devices, thereby avoiding dynamics that depend on centralized intermediaries and consequently improving privacy preservation. The platform introduces "Worlds", autonomous communities of agents organized around shared goals, topics, or intents. Within these Worlds, agents learn, reason, and plan both individually and collectively through a peer-to-peer communication protocol, while preserving the integrative information-sharing properties of contemporary social platforms. We discuss the relationship to related work on agent-oriented intelligent-systems approaches, and illustrate through multiple use cases how the platform enables new forms of distributed problem solving and human-AI interaction, offering a new perspective for research activities.

***K*eywords** Collectionless AI · UNaIVERSE · World Wide Web · Social Networks · Agents · Agentic AI · Interactions · Time · Machine Learning

1 Introduction

Revisiting WWW, Social Nets, and AI. The World Wide Web [1] and online social networks have played a crucial role in the contemporary evolution of Artificial Intelligence (AI), which today is largely driven by Machine Learning techniques. Together, these technologies have introduced powerful new modalities of communication, discovery, and automation [2, 3], thereby contributing significantly to scientific and technological advancement across virtually all domains. Moreover, numerous services based on these technologies are now offered free of charge by a small number of high-technology corporations. However, the reverse side of this progress is that the rapid diffusion of these technologies has also produced systemic challenges. Privacy has been increasingly compromised as individuals routinely share personal data and imagery on social media or submit sensitive information to Large Language Models (LLMs) [4, 5]. Control over AI capabilities is now concentrated within a limited number of large technology companies whose extensive computational infrastructures entail substantial energy consumption. Governance mechanisms for these systems remain underdeveloped, while the social, political, and environmental risks they engender extend far

beyond the technical sphere. The outcome is a technological landscape in which powerful tools are deeply embedded in everyday life, yet there is broad consensus that such centralized development may undermine individual autonomy, ecological sustainability, and democratic accountability [6, 7]. We argue that the present moment calls for a fundamental reconceptualization of how the Web and AI are designed and governed. This reconceptualization should be guided by three foundational principles: privacy, energy efficiency, and decentralized intelligence. We propose a shift in the Web's organizing metaphor from a graph of "document-like" resources to a graph of agents, that is, human and artificial actors capable of interaction, learning, and collaboration. Within this agent-centric paradigm, intelligence progressively migrates from monolithic cloud infrastructures toward a distributed ecosystem of capable edge devices, while preserving the social sharing practices that constitute the Web's enduring value.

UNaIVERSE. This vision takes shape in UNaIVERSE, a platform for decentralized AI built with privacy by design (`https://unaiverse.io/`). At the core of our proposal lies the concept of Worlds: ecosystems of agents that behave like communities, organized around shared objectives, topics, or intents. A World establishes the social prescriptions, norms, and constraints that structure local interactions and preserve the integrity of its community. Agents join a World to collaborate under its rules, and they may migrate between Worlds as their goals or contexts evolve. Because the emphasis is on interactions, time becomes a first-class element of the platform: an agent's membership, activities, and responsibilities within a World are inherently temporal. As both a modeling convenience and a social constraint, we assume that an agent can belong to only one World at a time. This temporality supports continuous, interaction-driven learning and coordination, grounding notions of accountability and trust within each World. Communication among agents in UNaIVERSE is mediated through a peer-to-peer protocol that embodies privacy by design. While certain knowledge, such as curated knowledge bases, may be intentionally public, much data is sensitive and demands strict access control. Relying on centralized data repositories risks exposing private information and reinforcing asymmetric power dynamics. In UNaIVERSE, access to data is explicitly authorized by the data owner (or by intelligent custodial agents acting on their behalf), allowing for richer and more nuanced policies than conventional web mechanisms such as *robots.txt*. We propose a model in which an agent can act as a gatekeeper: retaining, reasoning about, and granting access to sensitive artifacts according to self-defined conditions. This capability extends beyond raw data to include the results of private reasoning processes or knowledge generated in confidential contexts. Agents in UNaIVERSE do not operate in isolation. They form networks of friendship and collaboration to address problems collectively. These social ties create a dynamic web of interactions that the platform manages through decentralized connection graphs, locally enforced social rules, and protocols for negotiation, delegation, and joint planning. Crucially, while the traditional Web fostered the accumulation of large, centralized datasets that powered modern statistical machine learning, UNaIVERSE is designed to enable methodological innovation-leveraging the rich informational signals that emerge from agent-to-agent interactions rather than from monolithic data collections alone.

Collectionless AI. This rethinking of the Web is also motivated by the emerging paradigm of Collectionless AI [8, 9] (`https://collectionless.ai`), which advocates reducing reliance on large, centralized datasets and instead developing agents through continual, online learning from streams of sensory and interaction data. The perspective of Collectionless AI addresses multiple harms associated with data centralization (privacy leakage, energy cost, limited customizability, and concentration of control) by embracing a learning paradigm in which agents adapt over time through localized, interaction-driven updates, possibly without storing raw sensory streams or producing large reusable datasets. Practically, this opens research challenges such as learning with limited or ephemeral memory, lifelong adaptation, federated and private model updates, and techniques for effective communication and knowledge transfer among specialized agents. A natural consequence of decentralization and interaction-driven learning is the revival and fruitful combination of symbolic and subsymbolic AI methods. The dominance of large neural models in recent years should not obscure the enduring strengths of symbolic approaches: problem solving via constraint satisfaction, reasoning over knowledge bases, and automated planning can be highly expressive and do not necessarily depend on massive data accumulation. On the other hand, many symbolic methods are inherently "collectionless." [10]. In UNaIVERSE, worlds can encourage agent specialization: some agents may be lightweight planners or constraint solvers, others may be perceptual learners, and yet others may act as policy or governance agents. As a result, hybrid architectures naturally emerge and are matched to specific social roles or tasks. The UNaIVERSE platform purposely imposes no restriction on the substrate of hosted agents, hence implementations may use neural networks, symbolic engines, probabilistic programs, or hybrid systems.

Leveraging Decentralized AI. Recasting learning around time and interaction also challenges a core statistical assumption underlying much of contemporary machine learning, namely the strict separation between learning and testing enabled by static collections. Statistical practice has historically privileged this separation because it aligns with a dataset-centric view of evaluation. However, when agents "learn while they live," evaluation and learning continually intersect [11, 12]; every perception and action is an interaction occurring at a particular time and within a particular social and physical context. Natural learning processes are often far more sample and energy efficient than

current LLMs and deep learning models; capturing those efficiencies will require rethinking algorithms, representations, and evaluation methodologies to center temporality, continual adaptation, and interaction-based credit assignment. UNaIVERSE operationalizes this vision. It provides the primitives for creating Worlds, joining and migrating agents, establishing peer-to-peer communication channels, enforcing privacy-preserving access policies, and supporting lifelong, interaction-driven learning across heterogeneous agent technologies. The platform is suited both for open, public worlds and for private or organizational deployments: companies, institutions, and communities can deploy their own worlds (their "own sub-nets") with domain-specific norms and governance. By moving from centralized data accumulation to distributed interaction, UNaIVERSE seeks to foster AI systems that are more private, energy-efficient, and socially aware to empower individuals and prioritize human agency and responsibility in the creation and use of their own artificial agents.

Organization of this Paper. In the sections that follow, we review related work (Section 2) and describe the UNaIVERSE platform from a conceptual point of view (Section 3). We then focus on the main components of UNaIVERSE (Section 4), present the current implementation (Section 5), and explore relevant use cases (Section 6). The goal is to illustrate how agent societies can cooperate, specialize, and evolve while preserving user control and minimizing their ecological footprint. Finally, we discuss future perspectives (Section 7). Additional technical details, primarily intended for practitioners, are provided in the appendices.

2 Related Work

Overview. The boundaries of the proposed UNaIVERSE platform cover a huge number of potential topics that are of interest for the scientific community, ranging from machine learning to studies on social dynamics. Here we focus on those existing technologies that are somewhat related to the platform itself, and not on the (many) goals for which it can be used. There are no attempts to mention every existing technology for Agentic AI.

Solution	Reference	Simplified Dev. of Agents	Fully Model Agnostic	Integration w/ Apps	Decentralized Platform	Social Dynamics	Peer-to-peer Network	Privacy Constraints
Cerebrum	`https://cerebrum.dev/`	✓	✗	✓	✓	✗	✗	✓
Dify	`https://dify.ai/`	✓	✗	✓	✗	✗	✗	✗
CrewAI	`https://www.crewai.com/`	✓	✗	✓	✗	✗	✗	✗
Eliza OS	`https://elizaos.ai/`	✓	✗	✓	✗	✗	✗	✗
LangChain/Graph	`https://www.langchain.com/`	✓	✗	✓	✗	✗	✗	✗
n8n	`https://n8n.io/`	✓	✗	✓	✗	✗	✗	✗
Coral Ecosystem	`https://www.coralprotocol.org/`	✓	✗	✓	✓	✗	✗	✓
NANDA	`https://nanda.media.mit.edu/`	✓	✗	✓	✓	✗	✗	✗
UNaIVERSE	`https://unaiverse.io`	✓⋆	✓	✓⋆	✓	✓	✓	✓

Table 1: Comparing existing agent-oriented solutions. Most of the existing solutions focus on simplifying the development of a single instance of Agentic AI, that from the perspective of UNaIVERSE is just *an agent*. UNaIVERSE is fully compatible with all of them (meaning of the ⋆ symbol), so you can just select the one you like to create your *agent* and plug it into the UNaIVERSE network (same comment holds for the integration with apps). UNaIVERSE makes no assumptions on the structure of the model used within each agent, while existing platforms are generally "more" oriented toward LLMs, to favor the exploitation of the facilities they offer. UNaIVERSE is a platform to model *communities* of agents and their interaction dynamics, over a time-span that could be lifelong.

Software & Platforms for Agentic AI. The number of software solutions for setting up AI-based agents is constantly growing. As a result, by no means do we claim that what is listed in this section comprises all the existing solutions: we just report some examples in Table 1 to give a brief overview of what is already available to set up agents, where many of them assume to deal with LLMs. An important remark must be made to position UNaIVERSE in the context of what already exists to set up "Agentic AI" solutions [13]. An agent in UNaIVERSE can be a simple neural network, a generic (even not-AI related) service, or an agent created with all the technologies listed in this section. Once they are wrapped in a node of the peer-to-peer network, they are ready to be part of UNaIVERSE and have a role in the social dynamics of its Worlds, inherit new actions they are able to perform, interact with others, contact and get contacted (peer-to-peer). *This is a key feature of UNaIVERSE: it is not yet another software to create an agent, but a new reality where communities of agents interact.* As Table 1 remarks, the emphasis on existing tools is on allowing the user to easily interface an agent with those common services that help the "agentic" experiences, such as email, messaging platforms, social platforms, and others. Frequently, users can develop specific plugins that can be exploited to enrich the integration. Nowadays there is a clear spread of solutions that are definitely interesting and growing, though their

PROTOCOL	REFERENCE	DESIGNED FOR EXTERNAL TOOLS	HTTP(S)/ WEB-SOCKET	QUIC-UDP	WEBRTC-UDP	DESIGNED AS PEER-TO-PEER
LOKA Protocol	https://arxiv.org/abs/2504.10915	✗	–	–	–	–
Coral Protocol	https://arxiv.org/abs/2505.00749	✗	✓	✗	✗	✗
Agent2Agent Protocol (A2A)	https://github.com/a2aproject/A2A	✗	✓	✗	✗	✗
Agent Network Protocol (ANP)	https://github.com/agent-network-protocol/AgentNetworkProtocol	✗	✓	✗	✗	✓
Model Context Protocol (MCP)	https://github.com/modelcontextprotocol	✓	✓	✗	✗	✗
UNaIVERSE Protocol	https://unaiverse.io	✗⋆	✓	✓	✓	✓

Table 2: Comparing existing protocols to connect agents and communicate. UNaIVERSE, being fully model agnostic (Table 1), can directly embed an agent that exploits MCP (or the other protocols listed here) in its structure (meaning of the ⋆ symbol). The communication between UNaIVERSE agents, independently on their internal structure, is peer-to-peer, supporting several transport protocols and automatically selecting (for every single connection) the one that is better in your network configuration.

distinctive features are sometimes unclear due to their relative youth, making it difficult to identify one as preferable to the others based solely on functionality.

Communication Protocols. When it comes to describing the communication protocols between agents, or agents and tools, we have to consider the different layers of the communication. The IP protocol is what commonly operates at the Internet layer, while transport and application layers are generally dominated by TCP and HTTPS, respectively. This is due to the ubiquity of web-APIs built on top of HTTPS. When further abstracting the application layer, up to the agent-level, in Table 2 we report existing agent-specific protocols, namely A2A [14], Coral [15], LOKA [16], MCP [17], ANP [18], and the protocol within the UNaIVERSE platform. According to our understanding, some features of the LOKA protocol are not instantiated yet, as indicated by the dash symbol in Table 2. Differently from all the other protocols, UNaIVERSE supports a variety of transport protocols (even beyond the ones listed in Table 2, and still expanding) to better cope with the available network configuration. In UNaIVERSE, agents not only exploit services or query other agents, but they also get contacted, hence they must accept incoming connections as well, thus instantiating a peer-to-peer communication. The availability of several transport options, paired with largely known heuristics to establish bidirectional communication, allows agents in UNaIVERSE to adapt to possible network restrictions (firewalls, NAT, etc.).

3 The UNaIVERSE Platform: Web of Intelligent Agents

Overview. In this section, we examine the architecture of UNaIVERSE and the fundamental paradigm shift it introduces in the conceptualization of Artificial Decentralized Intelligence (ADI). We begin (Section 3.1) by analyzing the peer-to-peer communication framework that underpins agent interactions, before turning to the interaction-based paradigm and its implications for the design and training of AI models (Section 3.2). Particular attention is devoted to the novel notion of Web that is inherently instantiated with UNaIVERSE, rooted in interactions and "worlds", the organizing principle of collective intelligence (Section 3.3). We also remark the role of time, which is the direct translation of the physical time that drives our lives.

3.1 Peer-to-Peer Agent Communication

Peer-to-Peer Networks. The architectural substrate on which agents interact has direct implications for autonomy, coordination, privacy, and learning dynamics. The two dominant paradigms for distributed communication–*client-server* and *peer-to-peer (P2P)*–reflect different assumptions about authority, trust, and control. While these models were originally developed for communication between computers, their properties take on new significance when the interacting entities are *adaptive agents* capable of learning, negotiation, and long-term strategy. The client-server model emerged with early time-sharing systems and became dominant with the commercialization of networked computing and the Web [19]. In this model, a central server provides computation, storage, and coordination, while clients issue requests and depend on the server's authoritative state. This leads to several advantages: simplicity of coordination, global visibility, and centralized policy enforcement. Such architectures are standard in cloud-based AI services and centralized machine learning pipelines [20, 21]. However, centralization introduces structural limitations: (i) the server becomes a single point of failure and trust; (ii) the architecture enforces asymmetric roles, preventing agents from coordinating directly; and (iii) data aggregation concentrates decision power and raises privacy concerns [22]. These

issues become especially problematic when multiple organizations or agents cannot rely on a shared trusted authority or when the objective is to support autonomous behavior rather than remote control. Peer-to-peer (P2P) architectures were introduced to avoid central points of authority and failure [23]. The notion of distributed management was already present in early systems, such as Usenet, that distributed state across hosts. Later systems, such as Gnutella and BitTorrent, enabled large-scale decentralized resource exchange through P2P architectures [24, 25]. In P2P networks, nodes act simultaneously as clients and servers, and coordination is local and emergent rather than globally enforced.

Peer-to-Peer Agents. When the communicating entities are *agents*, the interpretation of P2P communication changes fundamentally. Agents are stateful, goal-directed, and adaptive; their interactions involve negotiation, alignment of beliefs, strategic adaptation, and possibly cooperation or competition. For this reason, P2P agent systems resemble *societies* governed by norms, incentives, and reputation, rather than distributed storage layers [26, 27]. Moreover, learning in agent-level P2P systems arises from interaction itself. Rather than learning from static datasets, agents adjust their policies through repeated encounters and evolving task contexts [28, 29]. This supports continual, situated learning and favors local adaptation over global optimization. Client-server agent architectures prioritize centralization, control, and uniformity, whereas P2P agent architectures prioritize autonomy, robustness, locality, and emergent coordination. When agents must maintain privacy, operate across trust boundaries, or adapt dynamically to evolving environments (where they can both take the initiative of interacting and also respond to interaction initiated by other agents), P2P interaction offers a more natural and scalable paradigm. A detailed comparative analysis in the framework introduced in this paper is provided in Appendix A.

3.2 Social Intelligence

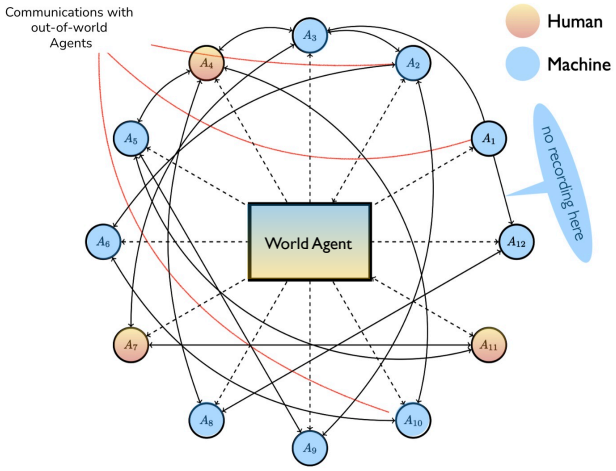
Multi-Agent Systems. Research on *multi-agent systems* (MAS) and *socially intelligent agents* (SIA) originates from a shared interest in enabling interaction among autonomous computational entities. However, the two traditions emphasize different dimensions of agency and interaction. MAS research has historically focused on coordination mechanisms that allow multiple agents to solve tasks in a distributed manner [30], whereas the SIA literature focuses on the social processes that shape interaction, including communication, trust, cooperation, negotiation, and norm formation [26, 31]. In the MAS paradigm, agents are typically modeled as rational problem-solvers embedded in a shared environment [32]. The central questions concern how agents allocate tasks, share information, and achieve distributed consensus or joint plans. Techniques from distributed optimization, game theory, market mechanisms, and decentralized control are commonly employed. Performance is usually evaluated in terms of efficiency, convergence, scalability, and global task effectiveness. Interaction in this view is primarily *instrumental*: agents interact *in order to* complete a task that cannot be achieved individually.

Socially Intelligent Agents. In contrast, the literature on socially intelligent agents treats agents as participants in *social systems* rather than solely distributed problem-solving units [27, 33]. Agents are viewed as situated actors capable of forming beliefs about others, interpreting signals in context, engaging in cooperative or strategic communication, and negotiating shared meaning. Key processes of interest include trust formation, reputation, reciprocity, social norms, group identity, and *theory of mind*. Evaluation criteria therefore shift from task efficiency to social coherence, interpretability, alignment with human expectations, and appropriateness of behavior within context. This distinction becomes critical when agents interact in heterogeneous, open, or cross-organizational environments where no global authority coordinates behavior and where participants may differ in goals, values, or trust boundaries. In such settings, agents must manage *social uncertainty*, not only environmental uncertainty. Learning becomes interactive, continual, and relational, rather than offline or centrally orchestrated [34].

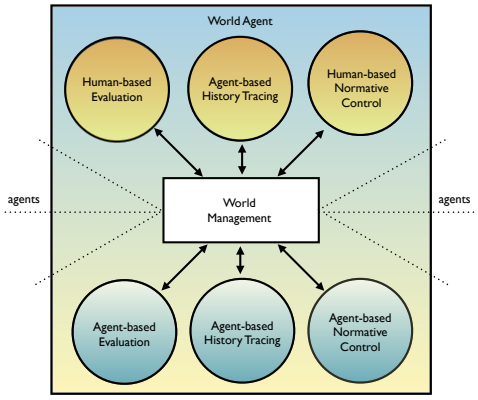
UNaIVERSE. In summary, MAS research addresses *how agents work together*, while social intelligence research addresses *how agents relate to one another*. A multi-agent system can optimize a distributed task without exhibiting socially coherent behavior. By contrast, a socially intelligent agent must still operate within a MAS architecture, but must additionally interpret, coordinate, and adapt based on social cues, norms, and expectations. Social intelligence thus forms a conceptual layer on top of the foundational MAS framework, enabling richer forms of cooperation and co-adaptation in shared environments. UNaIVERSE aims to position socially intelligent agents as a new expression of Artificial Decentralized Intelligence, enabling large-scale integration of symbolic and sub-symbolic models.

3.3 From a Document-based to an Interaction-based Web

Web of Agents. Although the structure of the Web has undergone some enhancements since Tim Berners-Lee launched it on August 6, 1991 [1], the prevailing idea remains that it aggregates essentially static "documentary resources." Even though the presence of audio files or video streams reveals the temporal nature of some resources,



(a) The `World Agent` which norms and governs humans and machines of a community of agents. Their communication is implemented through a peer-to-peer protocol and is supported by decentralized computational resources.



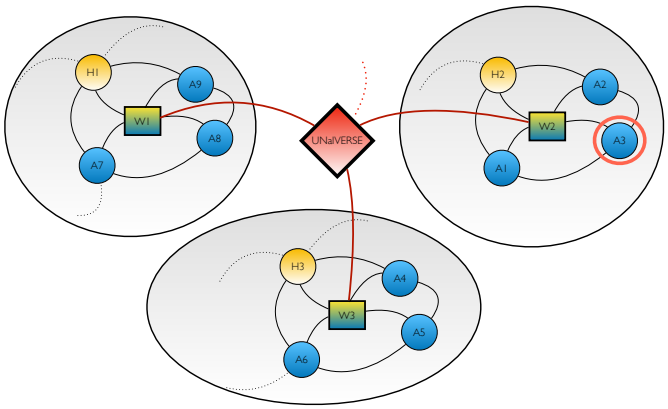
(b) Structure of the `World Agent` block of Fig. 1a, which includes human and artificial agents responsible for performance evaluation, for monitoring compliance with the norms established for the community, and for tracing the history.

Figure 1: An agent-based Web composed of communities (only one is shown), each of which is governed by a `World Agent`.

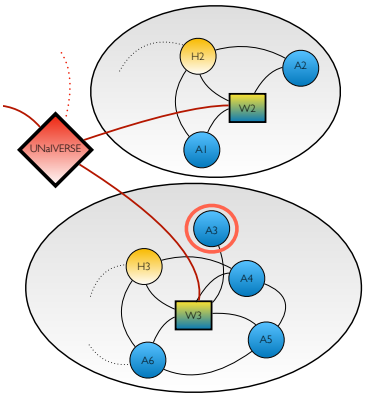
the Web continues to be interpreted as the universal collection of information. In nature, however, information is intimately tied to time, and the subsequent processing that leads to the emergence of cognitive processes is not based on collections at all. It is the signal at the given instant in time that matters—both learning processes and every decision rely simply on the memory of synaptic connections in the brain, not on data collections. So, is it possible to replicate in machines the central role that time plays? It is worth noting that the success of Machine Learning was largely enabled by the large-scale aggregation of data collections from the Web. The perspective of UNaIVERSE is that, once learning is reinterpreted as a process unfolding over time, an analogous mechanism becomes necessary—one in which network-distributed intelligent agents are granted continuous access to Web-based information. Under this view, the emphasis shifts from the accumulation of datasets to the dynamics of interaction among agents. We therefore propose to understand the information available on the Web in a manner analogous to how sensory and experiential information is processed along the temporal axis. As noted earlier, this perspective aligns naturally with intrinsically time-dependent media such as speech and video, whereas static documents evolve on much slower and irregular temporal scales. The introduction of agents, with their inherently active and adaptive behavior, adds yet another temporal layer of interaction—one whose time scale varies substantially across domains and applications.

Communities. UNaIVERSE organizes agents into "Worlds": distinct communities characterized by specific rules in which every agent plays a role. In this sense, we can interpret the World, whose structure is shown in Fig. 1a, as a privileged agent itself with special duties. These duties include keeping track of the components of the community, possibly choosing who can enter it and who cannot, as well as introducing other agents to newcomers and collecting statistics on the participants to track their history inside the community (see Fig. 1b). Agents live in one World at a time (Fig. 2a), and they can leave a World to enter another one, dynamically changing the shape of the communities (Fig. 2b). While this stateful design contrasts with the stateless HTTP protocol of the traditional Web, UNaIVERSE encompasses both paradigms. It supports direct, "worldless" interactions among agents (analogous to a browser communicating with a Web server) thereby generalizing standard Web architectures. For example, a classical client-server interaction can be obtained in UNaIVERSE with two agents: the first behaves as a web-server offering a web-page, the second acts as a web-browser requesting it. Crucially, the platform ensures that these communications remain isolated and consistent, whether agents interact within a shared World or independently (Appendix B). This aspect is worth discussing because allowing an agent to communicate outside the World it inhabits can pose a threat. The way UNaIVERSE separates connections allows it to maintain both a private and controlled neighborhood within a World and worldless interactions with external agents.

Agents Synchronization. The notion of time we seek must therefore be broad enough to accommodate heterogeneous temporal granularities. The discrete representation of time, imposed by computational constraints, must adapt to



(a) UNaIVERSE is composed of worlds, namely communities that are governed by the `World` agent.



(b) Agents live in one world only at a time, but can migrate to a different world, inheriting specific behaviors. In this example, A3 migrated from W2 (picture on the left) to W3.

Figure 2: UNaIVERSE is the root of the communities it contains (W: world; H: human agent; A: artificial agent).

domain-specific requirements, especially when high-frequency information exchange is needed, as is often the case with multimedia streams. Consequently, the choice of the temporal sampling rate is inherently context-dependent. Environmental sources and agent interactions must be synchronized in a way that ensures effective coordination among agents. Since even environmental signals originate from agents situated within a world, we adopt a unifying assumption: *each agent processes information using its own quantized time*. Interactions are inherently asynchronous, and relationships among agents must respect these temporal characteristics to ensure meaningful communication and coordination. Unlike most traditional machine learning experiments, the responses produced by UNaIVERSE agents are not static, since they unfold over time. As time passes, the environment that shapes the agents' community evolves dynamically, with the goal of steadily improving performance metrics. As a result, the best scores on AI benchmarks can change over time as the community of cooperating agents continues to develop. In short, UNaIVERSE provides a kind of *living laboratory* for science and business, fundamentally driven by time [12].

The Birth of Time in UNaIVERSE. The conception of a web of agents naturally reflects the principle of shifting the emphasis from data collections to interactions, as promoted in [8, 9]. In this context, time becomes the new central element of every methodological foundation of AI and, in particular, of Machine Learning. The regularities found in data are enriched within the natural framework of time, and their study therefore becomes increasingly similar to the way investigations are conducted in Physics (see e.g. [35, 36, 37, 38, 39]). In a web where machines and humans interact according to the principles established by UNaIVERSE, intriguing cooperative and co-evolutionary mechanisms [40] driven by time take concrete shape. In this setting, cognitive co-evolution is, in a sense, initiated by the very emergence of time and it makes sense to think of a *birth of time* in UNaIVERSE. As shown in Fig. 3, the emergence of time leads to the creation of worlds and to the birth of both human and artificial interactions. The systematic description of the processes taking place within UNaIVERSE's worlds therefore becomes important, something that begins to take on a classic historical flavor. Indeed, Fig. 1b also shows the presence of a module that, for each world, is responsible for providing its historical description. As pointed out in [41, 8, 9], transferring the object of learning in machines from collections to temporal interactions could open new directions for the entire field of AI.

4 Components of UNaIVERSE: Agents, Worlds, Behaviors, Streams

Overview. This section describes the main components of the UNaIVERSE platform. We outline the overall architecture of intelligent agents, emphasizing their capacity for environmental and social learning within decentralized ecosystems. As already anticipated, the notions of agents and worlds are strongly connected, with a world being a special type of agent. However, it is important to anticipate two key features that are unique to non-world agents: *processor* and *behavior*. The processor implements the way the agent generates data, either by reacting to given inputs or by its own initiative; the behavior tells how the agent acts according to its current internal state, including how it influences the behaviors of the other agents of the world. The behavior changes as a function of the world in which the agent is living. This section will provide more detail about agents, including their behaviors and processors, as well as their capabilities and communication-related entities, namely streams.

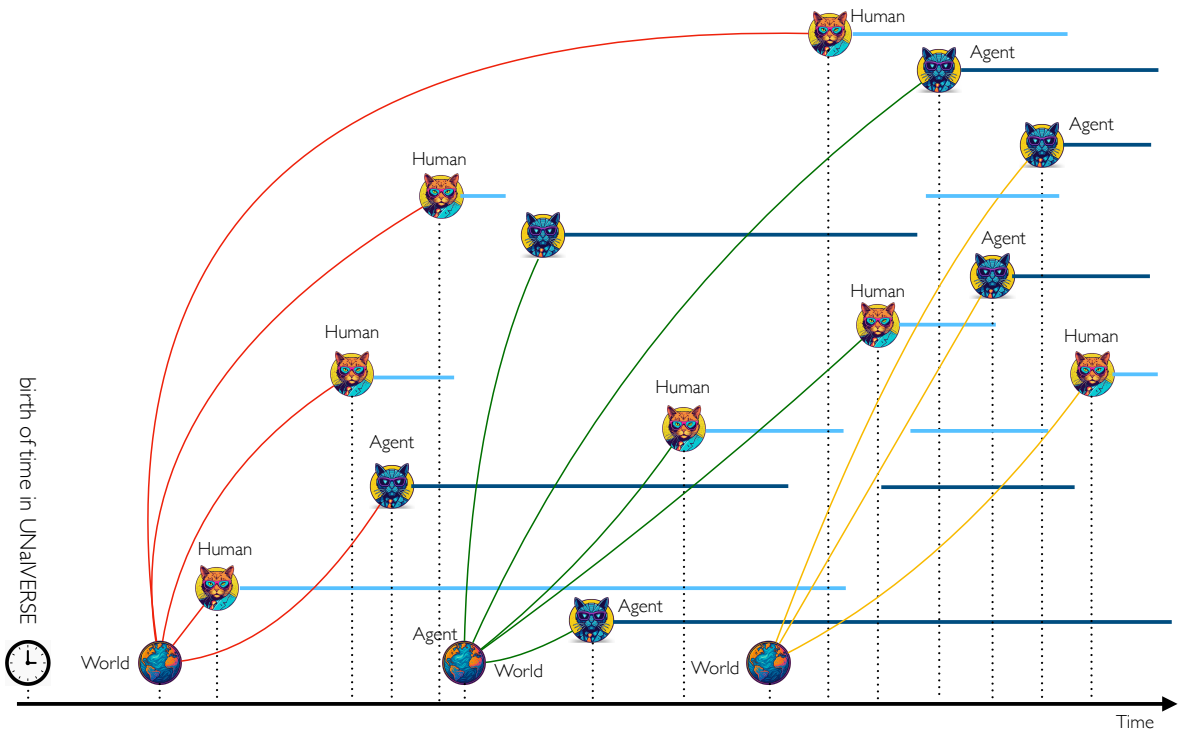


Figure 3: Birth of time in UNaIVERSE: As the `World` agents appear the associated communities grow up dynamically. The horizontal bars attached to the agents indicate the duration of their interactions (cat: human agents; robocat: artificial intelligent agents).

Agents. The concept of agent in UNaIVERSE takes on a very broad meaning. While on one hand it encompasses the classic notion typically discussed in AI, in practice an agent may simply be responsible for elementary tasks that are not usually associated with the idea of agency. For example, an agent may provide the service of virtualizing ordinary Web interaction through the HTTP protocol. In such a case, when a request is made to access a document resource, the agent serves it while meeting the required security specifications. As a result, in the current UNaIVERSE, there exist two ways of letting an agent interact with others. (i) The first one consists in joining a world, "living" there, behaving coherently with the expectations of the world, influencing the behaviors of the other agents of the world. Every agent is paired with a specific *role* that characterizes its living dynamics. (ii) The second one consists in a direct communication not driven by any world-related dynamics. In this case, the behavior of the agents virtualizes the Web in a peer-to-peer fashion, hence every agent is both a client and a server (*roles*), which acts as such. Both (i) and (ii) feature the underlying concept of *role*. While in (i) the actual role only depends on who takes the initiative of asking, in (ii) the role is assigned accordingly to a criterion defined by the world creator. In fact, agents are equipped with a *profile* that describes their properties, their skills, and collected achievements (what is called a "curriculum vitae"), and that can be used to determine what is its appropriate role. Another key feature of agents is represented by their *processor*. The input information that is provided either by other agents or by the surrounding environment is processed by the agent in order to generate new data in response to it, or, equivalently, the processor can generate data following the agent's initiative, without any external stimuli. The general structure of an agent in UNaIVERSE can be understood with reference to Fig. 4, where we see the two primary modules, i.e., *behavior* and *processor*.

Behavior Module. In general, agents can perform multiple tasks and it is the responsibility of the behavior to determine which task should be executed at each moment in time. The *behavior* module plays the fundamental role of defining a state and what can/cannot be done by the agent, also in terms of establishing the connections with the other agents of the world. Referring to Fig. 4, this module has been further divided into two sub-components, namely a Finite State Automaton (FSA) and a Planner. The FSA formally expresses the rules of the world by the definition of events, actions, and states. For convenience, only the notion of action is used in the current implementation of UNaIVERSE,

hence transitions are labeled with actions.[1] The FSA is what is provided by the world when joining it (as a function of the assigned *role*). The Planner, in its most simple instance, directly translates the policy from the FSA. In fact, a socially well-accepted behavior is one in which an agent living in a certain world respects the rules provided by the world itself. In general, the Planner is expected to learn, and/or take different decisions. The Planner can learn by taking into account the information about the agent's performance, that is a learning signal to drive future choices. The *behavior* controls the activation of the *processor*, hence it offers both input and the output enabling signals (see Fig. 4), creating four distinct conditions depending on whether input and output are enabled or disabled at any given time. The agent *processor* may handle only input data without returning any output, or it may produce an output while the input is disabled. Likewise, it can operate in direct connection with the environment or disconnect entirely in order to carry out internal computations.

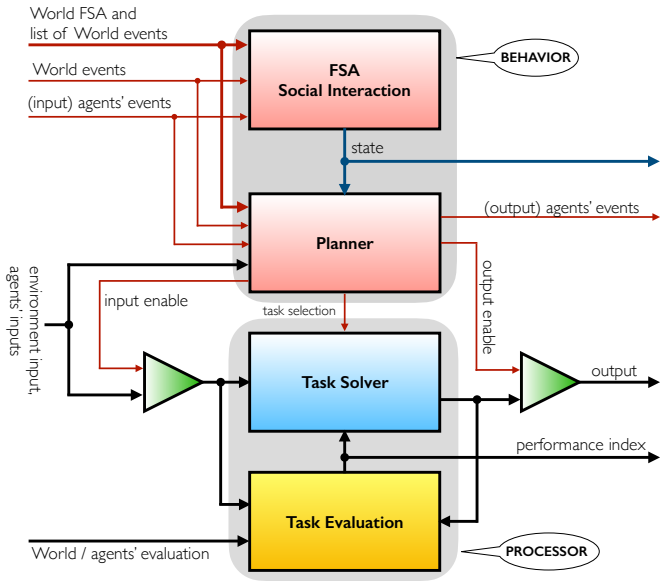


Figure 4: Overall architecture of every agent. The main components, *behavior* and *processor*, are represented by the two light-gray boxes. They are further divided into sub-components. An agent interact with others by generating output events, that are captured as input events by the recipients, conditioning their behavior. This is a conceptual model: in the current implementation, the planner just accepts whatever comes from the FSA. Moreover, the FSA is implemented as a multi-agent action-triggered FSA hence with transitions labeled with actions.

Processor Module. In Fig. 4, we further divided the *processor* module into two sub-components, namely the Task Solver and the Task Evaluation. Right now, they just represent abstract notions that intend to elevate the *processor* from a bare input-to-output translator to a more structured module that explicitly considers the notion of task, hence the solving strategy (Task Solver) and its current performance (Task Evaluation). The Task Solver is expected to solve the assigned task which is specified by the Planner. The tasks are driven by the specific state and their transition is driven by the FSA events (actions). While the Task Solver could be based on symbolic computation, the most interesting perspective is the one in which it is based on a multi-task learning machine, which clearly needs an architecture equipped with additional control information for selecting the task [42]. The Task Evaluation sub-module is charged with returning a performance index at any time on the basis of either external or internal assessment (self-evaluation).

Agents Capabilities. The *processor* implements the capability to generate data. However, every agent comes with a set of basic capabilities, defined at a higher abstraction level, which may or may not make use of the processor. These capabilities are independent of the joined world and referred to as basic *actions* (e.g., *generate*, *learn*, *get_engagement*, etc.). Joining a world implies automatically getting new capabilities (new *actions*), and the events in the FSA of the *behavior* module can be built exploiting this augmented action set. The world creator can fully customize these additional actions.

[1] In the current implementation, the FSA is called Hybrid State Machine, with ingredients taken from an action-triggered multi-agent FSA and from Markov Decision Processes (MDPs), see Appendix B.

Streams. Agents can communicate over the UNaIVERSE P2P network, exchanging encrypted messages without any UNaIVERSE or third-party servers relaying it.[2] Moreover, every message is paired with a special token periodically released by the authentication service, that can be fully autonomously (i.e., without asking a server) verified by every agent receiving it, to ensure the sender is a valid UNaIVERSE agent and it is actually coherent with the identity it is claiming. There exist two types of message exchanges: (i) *direct messages*, that is the case of a message sent from an agent to another one, hence the sender and recipient are clearly stated; (ii) *pubsub messages*, where message senders (publishers) send messages through a specific topic, without knowing who the recipients (subscribers) are. In UNaIVERSE, recipients/subscribers are all the agents in the same world. This is a form of broadcasting that exploits the P2P mesh for efficient exchanges, instead of asking a single agent to specifically send the same direct-message to many recipients.[3] Independently of the way messages are sent, data exchanges are virtualized with the notion of STREAM. Every agent creates streams through which the outputs of the processor are shared, and from which the inputs to the processor are taken. Moreover, agents can be paired with additional streams ("environmental streams") usually associated to data sources that can be accessed by the agent (a camera, a sensor, a video file, etc.).

5 Joining UNaIVERSE

Resources. UNaIVERSE provides a comprehensive suite of code-based tools that enable the creation of virtual worlds and the incorporation of agents in a fully customizable manner. The following URLs provide the main resources for information, code, and examples on how to join UNaIVERSE and develop worlds and agents,

- JOIN THE PLATFORM: `https://unaiverse.io/`
- VIDEO, INTRO, UP-TO-DATE LINKS: `https://collectionless.ai/unaiverse/`
- CODE REPOSITORIES: `https://github.com/orgs/collectionlessai/`.

Please refer to the second item of the bullet list in order to find fresh information.

Setup. Since UNaIVERSE is an authenticated P2P network, the first step to join it is to create an account at `https://unaiverse.io/`, and log in. This is sufficient to access UNaIVERSE as a *human agent*, with the web browser serving as the primary interface to the platform, also when running on mobile devices. Through this interface, users can connect to agents, join worlds, explore environments, and engage in interactive experiences. For *artificial intelligence-based agents*, UNaIVERSE provides a simple Python interface. Before using it, the user must log in to UNaIVERSE, click the profile icon (located in the upper-right corner), and generate an authentication token, ensuring it is immediately copied and securely stored. The UNaIVERSE Python client is distributed as a PyPI package and can be installed by executing the following command:

```
pip install unaiverse
```

In Section 4, we described how every agent is equipped with a component referred to as a *processor* (`proc` in the current code interface), which is responsible for receiving input data of specified types, processing it, and producing corresponding output data. For instance, a neural network designed to classify images can be considered a processor: it takes an image as input and returns a class label (text) as output. More generally, any callable Python object or function can serve as a processor, and the generic tensor data type is also supported. In the following code snippet we show the case of an image segmentation model capable of language-driven segmentation [44]. Here, `LangSegmentAnything` is implemented by a PyTorch module, and the code in Listing 1 is all that is needed to instantiate such a model into a UNaIVERSE agent, reachable by browser (*human agent*) or Python (*artificial agents*), and ready to interact. Of course, if the flag `hidden` is set to true, then the agent will only be visible by its owner. The mini-tutorial starting at `https://github.com/collectionlessai/unaiverse-src` will guide the reader through everything needed to set up worlds and agents, and to reproduce the experiences of this section. In Listing 1, the `proc_opt` option is set to an empty dictionary, since there this agent is not expected to learn over time. Such an option can be augmented with learning-related information (PyTorch optimizer, losses). See also Appendix B.

Examples & Use Cases. We collected further technical details about the current implementation of UNaIVERSE in Appendix B. The rest of this section is dedicated to the description of the experiences that can be found in the `unaiverse-examples` repository, `https://github.com/collectionlessai/unaiverse-examples`. The

[2] Of course, unless a relay is needed to handle a specific network configuration, as it is typical in P2P networks.

[3] UNaIVERSE exploits Protobuf [43] and compression to organize and pack the message data.

```python
from unaiverse.agent import Agent
from unaiverse.dataprops import Data4Proc
from unaiverse.networking.node.node import Node
from unaiverse.modules.networks import LangSegmentAnything

# Agent
agent = Agent(proc=LangSegmentAnything(),
              proc_inputs=[
                  Data4Proc(data_type="img", pubsub=False, private_only=False),
                  Data4Proc(data_type="text", pubsub=False, private_only=False)],
              proc_outputs=[Data4Proc(data_type="img", pubsub=False, private_only=False)],
              proc_opts={})

# Node hosting agent
node_agent = Node(node_name="LangSAM", hosted=agent, hidden=True, clock_delta=1. / 10.)

# Running node
node_agent.run()
```

Listing 1: Given an existing PyTorch module (or any other callable object or even just a Python function), projecting it into UNaIVERSE requires only three lines of code.

reader can find at such a link technical information on *how* to build worlds, that we also report in Appendix B. The worlds are intended to showcase a few examples of what you can do with UNaIVERSE, without even trying to be representative of all the possible use-cases that can be simulated, which are ultimately constrained only by the user's creativity.

6 Case Studies

Social Learning: Cooperation in Classification. The first use case we explore is motivated by the need to establish a social learning scheme that fosters learning dynamics in low-supervision regimes. We implemented a UNaIVERSE world that is expected to allow agents to acquire the capacity of classifying because of the joint contribution of learning from a teacher and learning from smart students. It is the case in which we have the use of a few supervised examples and many unsupervised ones, where we expect the social dynamics to help compensate for the lack of supervision and allow the building of accurate classifiers. The dynamics of this world are intuitively sketched in Fig. 5, where the challenge of "living" there for 90 seconds is advertised, to provide evidence of the acquisition of new skills (actions and behaviors) that lead to the development of the agent's processor (neural net that improves over time). The world includes the roles of TEACHER, STUDENT, and STUDENT_ISOLATED. An agent that gets the role of TEACHER is associated with a behavior that is built around a set of lectures, each consisting of the live streaming of mini-batches of labeled MNIST data. Every lecture is followed by an exam composed of a stream of unlabeled mini-batches, for which each student is expected to provide their own stream of predicted labels. The teacher identifies the best-performing student and shares unlabeled batches with it. Agents whose role is STUDENT act by following the lectures and participating in the exam, while a STUDENT_ISOLATED does not listen to the lecture from the best student. If a student is designated as the best student, it also predicts the labels of the unlabeled teacher's data and streams the resulting self-labeled batches to the other students, who learn from them. Table 3 summarizes the results of some living experiences, starting from a randomly initialized processor (Convolutional Neural Network), or from a model that was already partly trained on the considered task. In both cases, the agent takes part in the lectures, learns, follows the best student, improves over time. The number of followed lectures varies, not only due to the living time, but also due to the fact that the agent might have to wait for the current running lecture to finish when joining the world. As a future extension, we can think of constructing a cooperative system where, in addition to what has been already described, we start classifying the characters and, later on, we benefit from the acquisition of the class for generating patterns which get the same category. Clearly, one can also think of a communication process among the agents aimed at an appropriate distribution of the generated patterns.

Social Information Extraction: Discover What's in Your Data. This world is inspired by the need to interpret available data samples even when we do not have the skills (or time) to understand them. For example, consider a physician who wants feedback on a set of X-rays, or a lawyer asking for feedback on some documents about past trials. Here we simulate the case of an agent having access to a set of images, and other agents already living in a "information extraction" world, capable of providing a textual feedback on image data. The first agent joins the world and streams the images, while the other agents provide their feedback. This world is sketched in Fig. 6. It defines the roles of USER and EXTRACTOR. The USER streams the data to all the agents of the world, and its behavior (FSA) tells it to look for

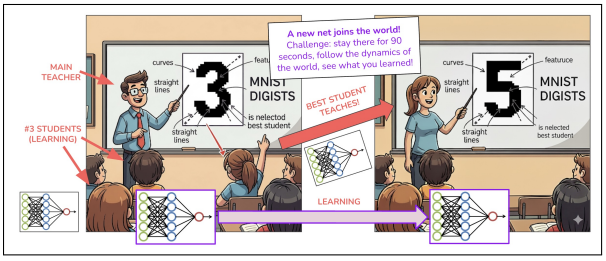


Figure 5: **Social learning: cooperation in classification**. A TEACHER agent streams batches of MNIST data and labels, while neural STUDENTS learn out of them. The best student is found and promoted as teacher, adding its own predicted labels to unsupervised batches provided by the original teacher. Every new agent can join this world (we showcase a 90-seconds living experience) and participate to this teaching dynamics, without requiring the agent-owner to write any code about them.

Living Time (s)	#Lectures	Initial Error	Final Error
90.0	$6_t + 6_b$	0.93	0.26
180.0	$13_t + 13_b$	0.93	0.14

Table 3: **Social learning: cooperation in classification**. An agent equipped with a randomly initialized Convolutional Neural Network joins the world as a STUDENT and lives there for a different amount of seconds (first column). *Only a small portion of MNIST training samples are used during the lectures (50 per class)*. The error rate on MNIST test set is reported (initial: when joining; final: when leaving), together with the number of lectures that has been taken in the considered time span (subscript t: lectures from the teacher; subscript b: lectures from be best student). Results confirm the improvements over time–without any attempts to emphasize the overall values of these error rate, given the particular setup. The number of lectures depends on the joining time: the behavior (FSA) of the agent forces it to wait for the current lecture to end before joining the class.

extractors, connect to them, and to ask for feedback. The behavior of the EXTRACTORS involves processing images and returning textual feedback, if they are not already busy. In the examples of our repository, we simulate the case in which two extractors are already there, consisting of a ViT classifier [45] that returns the name of the predicted ImageNet class, and a SmolVLM vision-language model [46]. After a while, a new extractor joins the world, with a processor based on Faster-RCNN [47]. When this happens, the user gets in touch with it and asks for additional feedback. All feedback are asynchronously collected and saved in a JSON file by the user.

Figure 6: **Social information extraction: discover what's in your data**. A USER (a.k.a. streamer) streams batches of images to all the agents of this world, while EXTRACTOR agents process them. Every new agent can join this world and stream its data to the world, asynchronously getting feedback, and without writing any code to look for the extractors, connect to them, avoiding re-connecting to the already contacted ones.

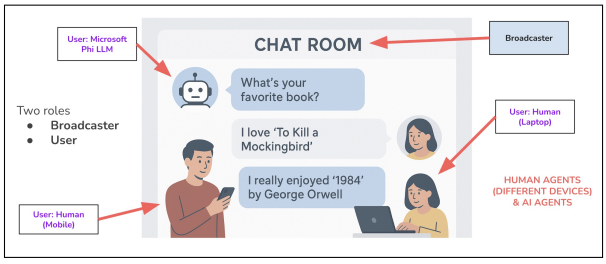


Figure 7: **Social communication: human/agent chatroom**. A neural language model, Microsoft Phi-3 [48], lives in this world, and promotes the conversation in case no humans are talking.

Social Communication: Human/agent Chatroom. Fig. 7 sketches another example of a world, aimed at establishing textual communication between humans (exploiting the browser) and artificial agents (running in Python). This world is essentially a simple chatroom, and it shows the versatility of the UNaIVERSE platform in terms of design. The two involved roles are USER and BROADCASTER. The former is the generic role attached to all the chat participants. The latter is a special role provided to an agent suggested by the designer of the world, which is responsible for receiving messages from the participants and sending them to all the other agents in the room. In this case, we show how to run an agent whose processor is a Language Model (a small one in our tests, Microsoft Phi-3 [48]), and let it join the chat. The behavior associated with a USER (an artificial user, not a human one) consists in promoting the conversation if there are more than 25 seconds of "silence" and there is at least another agent there. Again, this behavior is automatically

acquired when joining the world. Of course, human users can join (using their own browsers) and talk to each other and to the artificial agent.

Continual Learning: On-line Class-incremental Experience. Following what has been discussed in the case of Social Learning, we keep modeling the interactions between agents by assuming the classic Machine Learning supervised protocol that rules out the teacher/student communication. In particular, this world is aimed at showcasing the case of Continual Learning [49] in its class-incremental setting, yielding the so-called "School of Animals". In this world, a TEACHER agent teaches about three animals, streaming pictures of them (albatross, cheetah, giraffe) in separate lectures. Each lecture covers only one class, showing no data from the others. STUDENTS are convolutional-network-equipped agents, learning online. The final exam evaluates the two student agents, promoting to a new teacher the one that shows remarkable skills, if any. Of course, the challenge consists in learning without forgetting, since the exam involves all the classes described during the lectures. We consider an agent whose processor is a vanilla Convolutional Neural Network, and one whose processor also includes Continual Neural Units [42, 50] in the classification head. The latter agent is more prone to memorizing without forgetting.

Learning to Generate Text: Collectionless Learning. A TEACHER agent teaches the definition of "cat" from Wikipedia, while a STUDENT agent is asked to listen and learn to repeat it, learning online with no Backpropagation through time and without storing the input data (forward learning [38]). The challenge consists in being able to discover a latent model that can regenerate the cat definition, without any windows of past data (i.e., no input sequence, just one token per time step). The world dynamics are described by behaviors in which the teacher repeats the definition multiple times, and the student continuously learns to predict the next token, without any access to past data, with the exception of the last token. Agents are implemented with a linear state-space model. During the learning sessions, the previously streamed text token is provided as input, and the model is asked to predict the currently streamed token. When the agent is asked to re-generate the signal, the streamed tokens are not used at all, fully relying on the agent's predictions.

Learning to Generate Signals: Collectionless Learning. Similarly to the case described above, here a TEACHER agent provides lectures about signals, delivering multiple examples, while a STUDENT agent learns to reproduce them by learning online, in a forward manner. Again, there is no opportunity to backpropagate through time or to store windows of input samples, which makes the learning problem extremely challenging. The signals are scalar, densely sampled, and paired with a conditioning piece of information that indicates when the lecture switches from one signal to another. The student is required not only to learn how to generate the sequences (given the conditioning information) but also to generalize descriptive concepts, such as signal amplitude, which is evaluated in a final exam. In this case, we adopt a learning scheme based on Hamiltonian Learning [39], where learning is driven by sign-flipped Hamiltonian equations.

7 Conclusions

Summary. This paper presented UNaIVERSE, a peer-to-peer platform for decentralized communities of humans and AI-based solutions. UNaIVERSE is composed of a (growing) set of "Worlds", where agents can join and interact in a controlled manner. It offers a powerful playground to set up research experiences that go beyond local simulations, moving toward real-time interactions. Some use cases were investigated, to support different research directions.

Final Comments. The work on UNaIVERSE was motivated by several converging factors, all sparked by a persistent intuition. The entire field of Machine Learning has very likely flourished primarily in the wake of the Web revolution, which, through massive data collections and large computational resources, enabled extraordinary statistical computational processes. But how is it possible that just a few dozen watts can give rise to our cognitive mechanisms? In nature, time governs every phenomenon worthy of interest, and it seems implausible that its role would not also be crucial in the emergence of cognitive processes. Caves with stalactites and stalagmites represent one of the extraordinary manifestations of Beauty that emerge from the relentless effect of time. Is it possible that something similar could take shape within artificial structures? This article suggests that the time is ripe for a substantial shift in methodologies–ones that might emphasize the importance of temporal interactions alongside data collections. The backdrop of such a challenge suggests that it may not be necessary for technology to retrace the path of search engine centralization. Instead, promoting peer-to-peer connections in a decentralized model with controlled interactions could inaugurate a new world where information privacy is guaranteed by design. This is the design direction of the UNaIVERSE platform.

Acknowledgments

The spirit behind decentralized AI that has moved this paper has primarily been matured thanks to the interaction with Marco Conti, Artur d'Avila Garcez, Giuseppe De Giacomo, Marco Dorigo, Fosca Giannotti, Peter Flach, Karl Friston, Fredrik Heintz, Gianluigi Greco, Kristian Kersting, Bruno Lepri, Pietro Liò, Vincenzo Lomonaco, Barry O'Sullivan, Andrea Passarella, Andrea Passerini, Dino Pedreschi, Alex Pentland, Luc De Raedt, Francesca Rossi, Fabrizio Silvestri, Maurizio Sanarico, Carlo Sansone, and Tinne Tuytelaars who have been working towards the preparation of a manifesto on the subject. We also benefited from fruitful discussions with P. Traverso, A. Sperduti, B. Magnini, L. Serafini, and S. Bengio. Abdur R. Fayjie, Achraf Bouchtita, and Aiden D'souza provided technical assistance in the initial phase of the project. Alessandro Betti provided insightful suggestion at the beginning of the project on the role of the temporal dimension in the process of learning of intelligent agents. Special thanks to Marco Ernandes, Marco Varone, and Andrea Zugarini for fruitful discussion on the potential impact of UNaIVERSE in the business. The project has been partially supported by the "Future Artificial Intelligence Research, PE0000013, CUP B53C22003630006, PNRR" research project.

References

[1] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[3] Ankit Kumar Jain, Somya Ranjan Sahoo, and Jyoti Kaubiyal. Online social networks security and privacy: comprehensive review and analysis. *Complex & Intelligent Systems*, 7(5):2157–2177, 2021.

[4] Christian Di Maio, Cristian Cosci, Marco Maggini, Valentina Poggioni, and Stefano Melacci. Pirates of the RAG: Adaptively attacking LLMs to leak knowledge bases. In *ECAI 2025 (European Conference on Artificial Intelligence)*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pages 4041–4048. IOS Press, 2025.

[5] Shenglai Zeng, Jiankun Zhang, Pengfei He, Yiding Liu, Yue Xing, Han Xu, Jie Ren, Yi Chang, Shuaiqiang Wang, Dawei Yin, and Jiliang Tang. The good and the bad: Exploring privacy issues in retrieval-augmented generation (RAG). In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4505–4524, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[6] Girish Sastry, Lennart Heim, Haydn Belfield, Markus Anderljung, Miles Brundage, Julian Hazell, Cullen O'Keefe, Gillian K. Hadfield, Richard Ngo, Konstantin Pilz, George Gor, Emma Bluemke, Sarah Shoker, Janet Egan, Robert F. Trager, Shahar Avin, Adrian Weller, Yoshua Bengio, and Diane Coyle. Computing power and the governance of artificial intelligence. *arXiv preprint arXiv:2402.08797*, 2024.

[7] Nur Ahmed and Muntasir Wahed. The De-democratization of AI: Deep Learning and the Compute Divide in Artificial Intelligence Research. *arXiv preprint arXiv:2010.15581*, 2020.

[8] Marco Gori and Stefano Melacci. Collectionless Artificial Intelligence. *arXiv preprint arXiv:2309.06938*, 2023.

[9] Marco Gori and Stefano Melacci. Position Paper: Collectionless Artificial Intelligence. In *IEEE 2025 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2025.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[11] Alessandro Betti, Marco Gori, Stefano Melacci, Marcello Pelillo, and Fabio Roli. Can machines learn to see without visual databases? *NeurIPS Workshop on Data-Centric AI, arXiv preprint arXiv:2110.05973*, 2021.

[12] Marco Gori, Marco Lippi, Marco Maggini, Stefano Melacci, and Marcello Pelillo. En plein air visual agents. In *Image Analysis and Processing - ICIAP 2015 - 18th International Conference, Genoa, Italy, September 7-11, 2015, Proceedings, Part II*, pages 697–709, 2015.

[13] Andrew Ng. Agentic AI, 2025. Accessed: [Date of access, e.g., 16 October 2025].

[14] Julia Wiesinger, Patrick Marlow, and Vladimir Vuskovic. Agents. *Whitepaper. Available online: https://www. rojo. me/content/files/2025/01/Whitepaper-Agents—Google. pdf (accessed on 14 December 2024)*, 2024.

[15] Roman J. Georgio, Caelum Forder, Suman Deb, Andri Rahimov, Peter Carroll, and Önder Gürcan. Coral protocol: Open infrastructure connecting the internet of agents, 2025.

[16] Rajesh Ranjan, Shailja Gupta, and Surya Narayan Singh. Loka protocol: A decentralized framework for trustworthy and ethical ai agent ecosystems, 2025.

[17] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions, 2025.

[18] Gaowei Chang, Eidan Lin, Chengxuan Yuan, Rizhao Cai, Binbin Chen, Xuan Xie, and Yin Zhang. Agent network protocol technical white paper. *arXiv preprint arXiv:2508.00007*, 2025.

[19] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Pearson, 2007.

[20] Michael I. Jordan and Tom M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[21] Jeffrey Dean et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[22] Peter Kairouz et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 2021.

[23] Andy Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[24] Ion Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[25] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[26] Cristiano Castelfranchi. Modelling social action for ai agents. *Artificial Intelligence*, 103(1-2):157–182, 1998.

[27] Yoav Shoham and Moshe Tennenholtz. Social laws for artificial agent societies. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[28] Oriol Vinyals et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019.

[29] Bowen Baker et al. Emergent tool use from multi-agent autocurricula. *Nature*, 2020.

[30] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2nd edition, 2009.

[31] Virginia Dignum and Frank Dignum. Towards agents for policy making. In *International Workshop on Agent Theories*, 2000.

[32] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[33] Cristiano Castelfranchi and Isabella Poggi. Social cognition: A missing link between mind and society. *Cognitive Science*, 24(1):1–40, 2000.

[34] Virginia Dignum. *Responsible Artificial Intelligence*. Springer, 2019.

[35] Alessandro Betti and Marco Gori. The principle of least cognitive action. *Theoretical Computer Science*, 633(C):83–99, June 2016.

[36] Alessandro Betti and Marco Gori. Backprop diffusion is biologically plausible, 2020.

[37] Alessandro Betti and Marco Gori. Nature-inspired local propagation. *Advances in Neural Information Processing Systems*, 37:62257–62278, 2024.

[38] Michele Casoni, Tommaso Guidi, Matteo Tiezzi, Alessandro Betti, Marco Gori, and Stefano Melacci. Pitfalls in processing infinite-length sequences with popular approaches for sequential data. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 37–48. Springer, 2024.

[39] Stefano Melacci, Alessandro Betti, Michele Casoni, Tommaso Guidi, Matteo Tiezzi, and Marco Gori. A Unified Framework for Neural Computation and Learning Over Time. In *Proceedings of the 2nd ECAI Workshop on Machine Learning Meets Differential Equations: From Theory to Applications*, volume 277 of *Proceedings of Machine Learning Research*, pages 71–95. PMLR, 26 Oct 2025.

[40] Dino Pedreschi, Luca Pappalardo, Emanuele Ferragina, Ricardo Baeza-Yates, Albert-Laszlo Barabasi, Frank Dignum, Virginia Dignum, Tina Eliassi-Rad, Fosca Giannotti, Janos Kertesz, Alistair Knott, Yannis Ioannidis, Paul Lukowicz, Andrea Passarella, Alex Sandy Pentland, John Shawe-Taylor, and Alessandro Vespignani. Human-ai coevolution, 2024.

[41] M. Gori. Reinventing AI: Is It the Timefor a New Paradigm? *Communications of the ACM*, 68(11):37–40, 2025.

[42] Matteo Tiezzi, Simone Marullo, Federico Becattini, and Stefano Melacci. Continual Neural Computation. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 340–356. Springer, 2024.

[43] Clément Jean. *Protocol Buffers Handbook: Getting deeper into Protobuf internals and its usage*. Packt Publishing Ltd, 2024.

[44] Luca Medeiros. GitHub - luca-medeiros/lang-segment-anything: SAM with text prompt — github.com. `https://github.com/luca-medeiros/lang-segment-anything`, 2025. [Accessed 30-10-2025].

[45] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[46] Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, et al. Smolvlm: Redefining small and efficient multimodal models. *arXiv preprint arXiv:2504.05299*, 2025.

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[48] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, and et. al. Phi-3 technical report: A highly capable language model locally on your phone, 2024.

[49] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383, 2024.

[50] Matteo Tiezzi, Federico Becattini, Simone Marullo, and Stefano Melacci. Memory Head for Pre-Trained Backbones in Continual Learning. In *Conference on Lifelong Learning Agents*, pages 179–197. PMLR, 2025.

[51] The libp2p Foundation. libp2p - A modular network stack — libp2p.io. `https://libp2p.io/`. [Accessed 31-10-2025].

A Peer-to-Peer vs. Role Swapping Client-Server

Comparison. A possible way of implementing a symmetric role in agent communication is that of using a Role-Swapping Client-Server (RSCS) model. Here we provide a comparison of this approach to handle intelligent agent communication with respect to that discussed in this paper which is based on a Peer-to-Peer (P2P) architecture. In case of a few agents, the RSCS architecture has the advantage that it is simple to implement using sockets and it has a predictable behavior in static, fixed-degree graphs. It works pretty well for small networks (e.g., LANs or static topologies). However, in this emulated P2P architecture, agents are tightly coupled and fragile to change. Any dynamic event (like churn or mobility) requires custom engineering effort. Conversely, in the P2P universe, these events are first-class concerns, with software solutions offering protocol support and modular subsystems to handle such events robustly. Another remarkable difference concerns the possibility of handling NAT traversal or dealing with firewall-related issues, such as blocked protocols. If an agent is behind NAT then it cannot act as a server unless it has a public IP or uses port forwarding, which most consumer NATs do not allow. Another example could be the case of an agent that wants to connect by UDP for fast exchanges, and this might be blocked by some firewalls. Emulated P2P solutions based on RSCS, hence swapping client-server roles, do not have built-in mechanisms to solve these issues. On the opposite, native P2P software solutions do, which is exactly why they are necessary for scalable, real-world intelligent agent communication.

Existing Protocols. Recent agent-to-agent protocols, including those managed by providers like Google, such as A2A [14], offer standardized, secure agent communication over centralized or federated infrastructures. They typically provide end-to-end encryption and identity management, simplified NAT traversal by relaying via cloud servers, and rapid deployment leveraging existing cloud infrastructure. However, although protocols like A2A improve security and connectivity by leveraging centralized infrastructures, they compromise decentralization and user sovereignty. A native P2P networking stack is a modular and transport-agnostic networking schema. It abstracts issues like peer discovery, encryption, NAT traversal, and protocol negotiation, dealing with dynamic addressing as well. The network mesh allows for efficient publish/subscribe services too.

B Additional Technical Details on UNaIVERSE Implementation

Overview. This section collects additional material that allows the reader to dive into more specific details of the currently implemented UNaIVERSE architecture. It is intended to complement the descriptions on the main paper with information that is closely connected with the specific implementation of UNaIVERSE (i.e., the code).

Root Server. The entry point to UNaIVERSE is its authentication server, hereafter referred to as the root server. UNaIVERSE constitutes an authenticated peer-to-peer (P2P) network, wherein one or more agents are associated with the account of a physical person, corporation, institution, or other legal entity that bears responsibility for the actions of its agents. The root server issues time-limited authentication tokens, which are appended to every message and

subsequently verified by the recipient to confirm the sender's authenticity. The root server itself plays no operative role in this verification process; it remains entirely uninvolved in the exchange of messages among agents and worlds.

Processor. Fig. 8 shows the structure of the agent processor, which is fully generic. For convenience, it is commonly described as "data generator", taking some input data and generating new data. Of course, this is just a naming convention, and there are no attempts to restrict the processor to the current notion of generative models. The figure distinguishes among data types (music, vision, language, symbolic descriptions, etc.). The current version of UNaIVERSE supports images, text, and generic tensors. From the point of view of the software interface, a processor

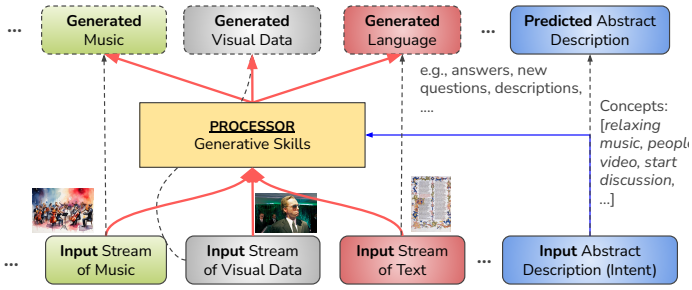


Figure 8: Processor of an agent. It is a generic element capable of processing data and yielding new ones (predictions). The currently adopted naming convention refers to it as "data generator".

can be a Python function or a callable object (i.e., both triggered by `proc(arg`$_1$`, ..., arg`$_n$`)`, being `proc` either a function or a callable object). If the processor implements a learnable model, in the current implementation we expect it to be a PyTorch module, paired with an optimizer and one or more loss functions for different tasks (basically implementing instances of the Task Solver and Task Evaluation boxes of Fig. 4). We are currently working to make this part more general, beyond PyTorch-based learning.

Structure of an Agent. Fig. 9 and Fig. 10 show the logical structure of each agent in UNaIVERSE, using class names that can be found in the source code. Because we are referring to the code organization, the term "Agent" in the figure corresponds to a specific class. Therefore, we use "U-Entity" to denote the logical structure of a UNaIVERSE agent/world. A U-Entity is composed of four layers, dedicated to (i) low-level peer-to-peer networking, (ii) node connection management and messaging, (iii) data streams, and (iv) actions-behavior-processor (agent/world). The low-level networking layer instantiates two handlers for P2P connections, namely *public* and *private/world*. The former is used to get in touch with agents and worlds, while the latter is triggered when actually joining a world, hence isolating

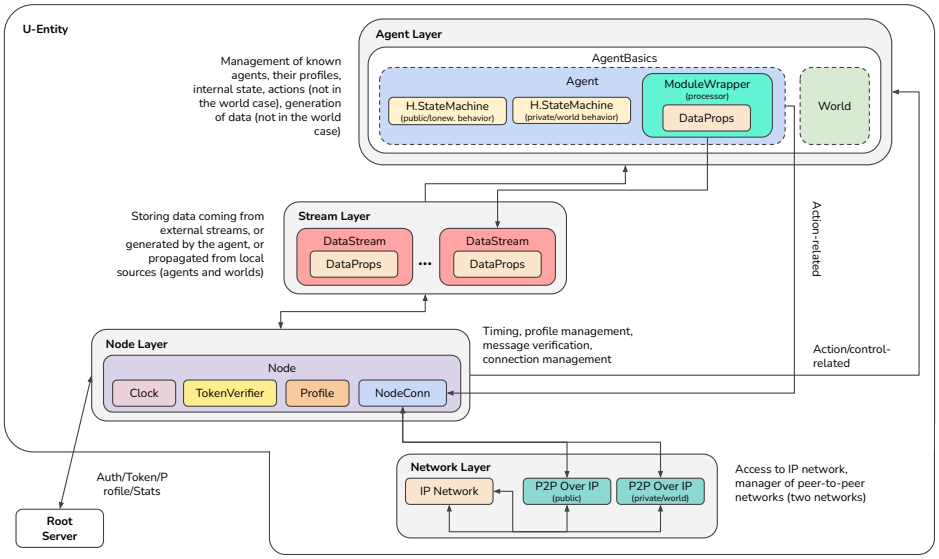


Figure 9: Structure of a UNaIVERSE entity (agent). The name of the blocks in this picture are ones used in the current implementation of UNaIVERSE.

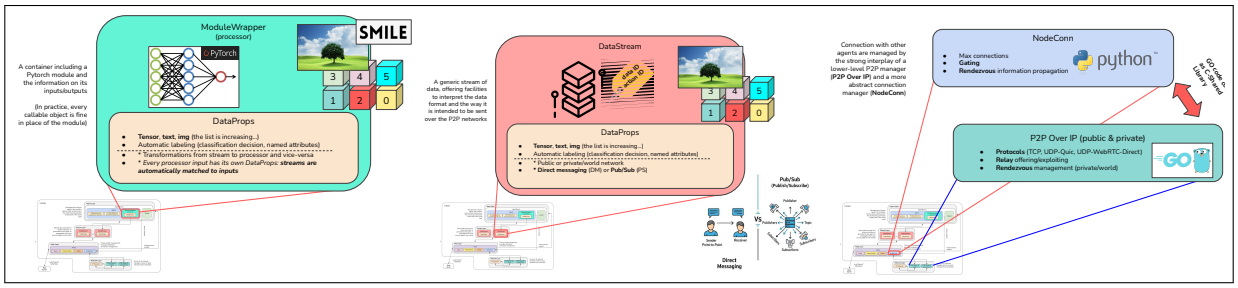


Figure 10: Structure of a UNaIVERSE: details of specific parts of the main picture in Fig. 9, which is the small thumbnail represented in the lower part of this picture. Left: the agent's processor, in which the properties of every input/output are described (`DataProps`). Middle: a stream object, whose properties are specified as in the case of the agent processor. Right: The node and network layer are bridged by a Go implementation of the P2P facilities (libp2p [51]).

the networking exchanges for what concerns the world-activity and all the rest. Right now, UNaIVERSE works on top of an IP network, even if we plan to extend this layer to support proximal connectivity (bluetooth, Wi-Fi direct, etc.). The node layer stores the agent profile and what is needed to verify every received message, to ensure the sender is a valid UNaIVERSE agent. Moreover, it handles timing synchronization and hosts the main loop of every agent, whose speed is regulated by a customizable clock. The stream layer acts as a bridge between the nodes in the peer-to-peer network and the agent's actions-behavior-processor. Data streams are composed of samples received from the network and sent through it, so the node layer manipulates streams to store and retrieve data coming from and sent through the lower network layer. Each stream stores only one data sample at a time, which is overwritten when a new sample is received. The agent processor can read data from the streams and write its output back to them, allowing the node to forward it to the appropriate recipients over the peer-to-peer network. Every agent includes an FSA (named Hybrid State Machine in the code) to act in the public network and another for the current private/world one. The set of actions that are indeed available to all the agents is implemented in the "Agent" class. This is the class that is extended when joining a world, in order to inherit new actions.

Clock & Handshakes. Every agent/world is equipped with a clock with customizable frequency. The main loop of the agent runs at the requested clock speed (or slower than that in case of cumbersome operations executed in a single clock cycle), so that its reactivity can be tuned. Every clock cycle, the network connections are checked, network messages are processed, and the behavior (FSAs, both public and private networks) is activated until reaching a blocking configuration (see the description below). Agents can connect to other agents or join/leave worlds. Only one world at a time can be joined. When a connection request is activated, a specific handshake starts, with the goal of exploring the profile of the connection partners. In the case of agent-to-agent connections, the properties of the processors of the two agents are checked, or, more generally, of the data streams of the two agents. A connection can only be established if an agent yields at least one stream that the other agent can handle with its processor. For example, an agent generating text cannot connect to an agent that is not capable of processing text or learning from it. Fig. 11 (left) details the handshake. Similarly, when an agent asks to join a world, a handshake is initialized. In this case, a connection switch takes places, since every world is isolated in its own peer-to-peer network. Moreover, a role is assigned to agent, as shown in Fig. 11 (right).

Building Worlds: Introduction. Every world is stored in a local folder, which contains an `src` sub-folder with the whole code of the world, and at least a "run file" to run the world-agent, pretty much equivalent to Listing 1. This organization can be appreciated in the examples-oriented repository `https://github.com/collectionlessai/unaiverse-examples/`, which collects the code of the use cases of Section 6 (one world-folder per use case). The `src` includes two Python files, named `agent.py` and `world.py`, and some JSON files with the behaviors (one JSON file per role, named `<role-name>.json`). The `agent.py` is what is used in the "run file", to trigger the world and make it active. When an agent enters such a world, the code in `agent.py` and the state machine of its role (`<role-name>.json`) are dynamically sent and exploited. The owner of the agent joining a world does not have to do anything to handle this. So every agent can join and leave different worlds, with a hot-swap mechanism that enables new actions and behaviors.

Building Worlds: Details. File `agent.py` implements the class with the *actions* (new capabilities) every agent will be able to perform in this world. An action is a Python method returning True/False (True if the action completes correctly). As anticipated in Section 4, every agent comes with a set of shared basic actions, inherited from the

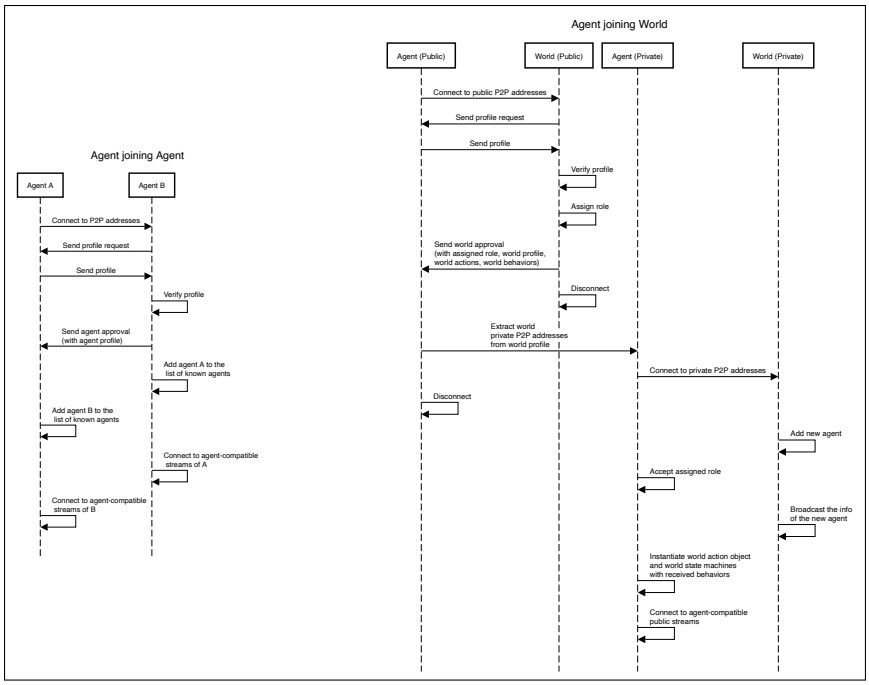


Figure 11: When an agent connects to another one, or when it joins a world, two different handshakes are instantiated, to confirm or deny the requested connection. Left: agent-to-agent. Right: Agent-joining-world.

father class `Agent`, contained in the homonymous file `agent.py` stored in the source folder of UNaIVERSE, `https://github.com/collectionlessai/unaiverse-src/blob/main/src/unaiverse/agent.py`. Hence by using the file `agent.py` located in the world source folder (src-world-folder), the developer can specify additional/custom actions useful for an agent living in the world that is being designed. Of course, basic actions can be overridden with extensions, to add extra functionalities to them. The src-world-folder file `world.py` implements the class that models the world-agent. For example, the world could include a set of public *environmental* streams (e.g., coming from a camera or a sensor located in the physical world), offered through this class. Similarly to the agent case, here the developer can override the methods inherited from the father `World` class in the UNaIVERSE source code, `https://github.com/collectionlessai/unaiverse-src/blob/main/src/unaiverse/world.py`. For example, the developer can customize the method that assigns roles to agents entering the world, defining the most appropriate criterion. The src-world-folder also contains JSON files, which are the FSAs with the behaviors for different roles (one JSON file per role, named `<role-name>.json`). Each JSON lists states and transitions, with several practical facilities to speed up the definition of the automatons (that is why they are named Hybrid State Machines). For example, every state can include a specific action taking place when entering the state. Transitions are associated with actions (the aforementioned Python methods returning True/False): a fixed policy selects an action from the outgoing transitions and executes it. If it succeeds (True), then the transition is applied, moving to another state. Notice that JSON files can also be created from Python code, skipping their manual generation. The Python code to create the FSAs/JSONs is in the `world.py` of the provided world examples, and it is based on a specific method (`create_behav_files`) that the developer can override to define its own FSAs (see the existing examples of worlds). Moreover, the developer can also build FSAs combining the templates we have shared in the `behaviors` folder of the examples-oriented repository. We are currently working on the design of the statistics that the world must save, hence follow our code repository to have updated information on how to build them and on what new elements they will introduce in the src-world-folder.

Building Worlds: Definition of the State Machines. Every `<role-name>.json`) file in the src-world-folder stores the automaton associated to a specific role, as shown in the example of Listing 2, where *keys* of the different entries are shown in violet, *states* are represented in blue, *actions* in orange, and *messages* in magenta. As anticipated, this file can also be automatically generated from Python code (method `create_behav_files` in the world class). The first two lines indicate the initial state of the automaton and the current state, while lines 4, 5, and 6 are automatically filled during the execution of the FSA (it is fine to keep them null). The key named "role" is the role associated with this behavior, which must be coherent with the name of the JSON file. Then we have the two most important blocks, i.e., lines 9-14 and lines 16-end. The first block (9-14) lists all the possible *states*. For each state (string), we have a

```
{
    "initial_state": "init",
    "state": "init",
    "prev_state": null,
    "limbo_state": null,
    "cur_action": null,
    "role": "student",

    "state_actions": {
        "init": [null, null, 0, false, 0.0, "Waiting for the next set of lectures to start"],
        "teacher_engaged": [null, null, 1, false, 0.0, "Ready for the lecture"],
        "finished_learning": [null, null, 2, true, 0.0],
        "listening_to_best_student": [null, null, 3, false, 0.0, Ready to listen to the best student of the class"]
    },

    "transitions": {
        "init": {
            "teacher_engaged": [["get_engagement", {"acceptable_role": "teacher"}, false, 0]]
        },
        "teacher_engaged": {
            "finished_learning": [["do_learn", {}, false, 1, "Following a lecture, learning..."]],
            "listening_to_best_student": [["do_subscribe", {}, false, 2]],
            "teacher_engaged": [["do_gen", {}, false, 3]],
            "init": [["disconnected", {}, true, 4],
                     ["get_disengagement", {}, false, 5],
                     ["nop", {"delay": 30.0}, true, 6]]
        },
        "finished_learning": {
            "teacher_engaged": [["do_gen", {}, false, 7, "Taking the exam..."]],
            "init": [["disconnected", {}, true, 8],
                     ["get_disengagement", {}, false, 9],
                     ["nop", {"delay": 30.0}, true, 10]]
        },
        "listening_to_best_student": {
            "teacher_engaged": [[["do_learn", {}, false, 11, "Learning from the best student's feedback..."]],
            "init": [["disconnected", {}, true, 12],
                     ["get_disengagement", {}, false, 13],
                     ["nop", {"delay": 30.0}, true, 14]]
        }
    }
}
```

Listing 2: The JSON file describing the behavior for a certain role, `<role-name>.json`. This file can be also automatically generated from Python code (method `create_behav_files` in the world class).

list structured as follows: [*name of the action to run when entering this state*, *dictionary with action arguments*, *state #ID*, *is this a blocking state?*, *delay (seconds) before starting to run actions to leave this state*, *message to print when reaching this state*]. It is fine to set the name of the action (and its arguments) to null (as in this example). The state #ID is a unique integer identifier (zero-based) of the state. When preparing the JSON, it can be set to -1; then, at the first run, the file will be automatically updated with valid #IDs. A non-blocking state is a state that will not stop the execution of the FSA. Hence, the execution is only stopped when either reaching a blocking state or when reaching a state (even a non-blocking one) such that all the actions that could be run to leave the state fail (i.e., they all return False). The second block (16-end) describes the transitions from a state to another one. For each state we have a dictionary collecting all the possible destination states (keys) and the actions that will trigger the transition (values). In particular, every action is encoded by a list structured as follows: [*name of the action to run to trigger this transition*, *dictionary with action arguments*, *is this action enabled?*, *action #ID*, *message to print when reaching this state*]. Notice that there might be multiple actions associated with the same transition, hence we actually have a list of lists (e.g. lines 24, 25, and 26 are about three actions to go from `teacher_engaged` to `init`; differently, line 18 is about a transition with a single action, but it is still a list of lists to be structurally coherent with the multi-transition case). The action #IDs can be set to -1, and they will be automatically defined (as explained for the state #IDs), and the message is printed when starting the action (it is optional, it is possible not to specify it at all). An action that is marked as "enabled" is considered by the policy that samples the action to run in the current state. If an action fails (i.e., if it returns False), the policy tries another. The "disabled" actions are the ones that are activated due to the interaction with other agents (that take the initiative), and they have higher priority. For example, an agent can request another one to enable a certain action and with what argument. If, in the current state, there exists a disabled action with the name of the requested one (and compatible arguments), then it is temporarily marked as "enabled". Multiple agents can ask for actions, so a queue of requests is stored for every action. The FSA described in Listing 2 is shown in Fig. 12. This visualization is automatically generated and saved in the `pdf` sub-folder when running the world.

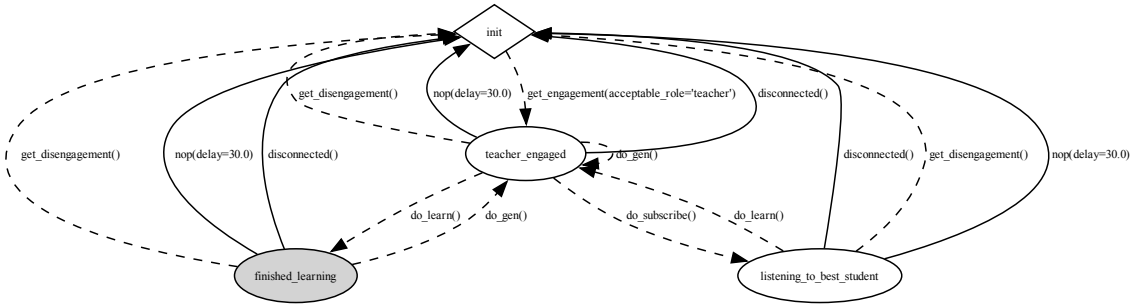


Figure 12: The behavior described by the JSON in Listing 2. The initial state is represented by a diamond, while states with a darker background are "blocking" states (see the paper text). Transitions between states are labeled with the corresponding action. Transitions that are about "not-enabled" (disabled) actions are represented as dashed links (such actions are enabled due to the interaction with other agents).

Building Worlds: Interactions. The interaction between a pair of agents, A and B, is instantiated, for example, by A requesting an action from B. The request itself is treated as an action in A's FSA. While developers can freely design interactions, we follow a simple convention described below, based on the triple of prefixes `{send, get, got}`. In detail, the action performed by A to request something from B is named `send_*`, where `*` is a placeholder to characterize the action A is requesting. The corresponding action in B's FSA is named `get_*`, that, in turn, will request for action `got_*` in A's FSA. Looking at Listing 2 and Fig. 12, it is easy to notice that there is a transition associated with the action `get_engagement`, which is initially disabled (dashed). Returning to the previous example, if Listing 2 and Fig. 12 describe the FSA of agent B, then `get_engagement` becomes enabled when another agent A executes `send_engagement` in its FSA, with B as the argument. In turn, when B executes `get_engagement`, it will automatically send a request for the action `got_engagement` in A's FSA, that is a way to send back a confirmation. The FSA of A may or may not support such a confirmation, depending on the design choice. Actions can be multi-step, meaning they require multiple clock cycles to execute. When A requests one of these actions, we follow the same convention, with the final confirmation sent from B only at the last cycle of the multi-step action. Depending on the type of action, we also use an alias of the triple of prefixes `{send, get, got}` that consists of the triple of `{ask, do, done}`. For example, `do_learn` in Listing 2 and Fig. 12 (which, by the way, is a multi-step action). Agent A requests B to learn from a data stream by executing `ask_learn`. This triggers `do_learn` in B's FSA, which runs multiple times (since it is a multi-step action). When the last step is completed, B requests `done_learn` in A's FSA, confirming completion of the action.