Harry Edwards

Omo Research

FINAL — OMO RESEARCH TECHNICAL REPORT · JUNE 2025

Omo Space: A Spatial Layer for AI Agent Workforces

We introduce Omo Space, a spatial computing layer that represents AI agent teams as walkable, inhabitable places inside Minecraft. Every room is an agent. Hiring means walking up to an empty plot and issuing a natural-language request. Rooms construct themselves block by block as agents come online, and live reasoning streams appear on in-world screens behind each agent. We present the three-ore architecture — Gemini (brain), the Agent Development Kit (hierarchy), and the Model Context Protocol (hands) — and demonstrate through systematic system-level evaluation that spatial presence produces a qualitatively different monitoring paradigm than tab-based interfaces, where walking replaces window-switching and proximity serves as an observable attention signal. We note that human-factors validation of these spatial interaction claims requires user studies planned for future work (Section 8.1); the current evaluation focuses on architectural properties.

Executive Summary

Omo Space turns managing AI agents from staring at text windows into walking through a Minecraft world where every room is an agent. Instead of typing commands in a chat interface and waiting for asynchronous results, users navigate a live campus — each agent has a physical room with screens showing its reasoning in real time. Hiring a new agent means walking to an empty plot and describing what you need; the system builds a unique room block by block, and the agent walks in ready to work. The architecture separates reasoning (Gemini), organization (Google's Agent Development Kit), and tool-use (Model Context Protocol with spatial gating) into independently replaceable modules. All code is open source and runs on consumer hardware.

Abstract

Background. AI agent workforces are predominantly accessed through text-based interfaces — chat windows, dashboards, and logs — that render agent activity invisible and opaque. Users cannot observe agent reasoning in real time, cannot intervene mid-task, and lack spatial intuition about agent organization, work distribution, or current state.

Problem. This transparency gap limits trust in autonomous agent systems and prevents users from forming accurate mental models of multi-agent coordination. Existing multi-agent frameworks (AutoGen, CrewAI, LangGraph) represent agents as abstract message-passing entities lacking spatial embodiment, navigable presence, or observable real-time reasoning.

Approach. We introduce Omo Space, a spatial computing layer that represents AI agent teams as inhabitable places inside Minecraft. The system maps agents to physical rooms, attention to spatial proximity, and tool access to room location — a paradigm we term *spatial agent habitats*. Omo Space rests on a modular three-component architecture (the "three ores"): Gemini for reasoning, the Agent Development Kit (ADK) for organizational hierarchy and inter-agent delegation, and the Model Context Protocol (MCP) for tool-use with spatial gating. A self-building world-architect agent generates unique rooms on demand using Gemini's spatial reasoning; position updates stream at 20 Hz with < 50 ms latency via WebSocket; and the approval gate intercepts sensitive actions with < 300 ms overhead, defaulting to denied.

Results. The working prototype supports a founding crew of three agents (Chief of Staff plus two specialists) with on-demand agent hiring. We measure setup reproducibility at 9.8 minutes mean from clean environment (n = 5), room-gating correctness at 100% tool-to-room mapping accuracy, delegation reliability at 85% (Chief of Staff → specialist), and end-to-end chat-to-screen latency at 0.8–2.1 s (median 1.3 s). The world-architect produces structurally sound, unique room designs across 20 test hires with 3/20 minor interior-layout issues correctable by manual adjustment.

Implications. Spatial agent habitats represent a new interaction paradigm for AI workforces, replacing tab-based monitoring with spatial navigation. We release all code as open source and argue that embodied, walkable agent environments will grow increasingly important as AI workforces expand in capability and autonomy.

Keywords: spatial computing, AI agents, Minecraft, multi-agent systems, spatial agent habitats, human-agent interaction

1. Introduction

The prevailing interaction model for AI agent workforces is text-centric and temporally opaque. Users issue commands through chat interfaces and asynchronously receive results — often without observing the intermediate reasoning, delegation, or tool-use that produced those results. When an agent errs, the user discovers the error post-hoc. When work completes, the user polls for completion rather than observing it. This interaction pattern creates what we term a **transparency gap**: a structural barrier between the user's mental model of the agent system and its actual operation. Closing this gap requires an interaction paradigm where agent activity is continuously visible, spatially organized, and directly inspectable — properties that text-based interfaces cannot provide.

Spatial computing offers an alternative paradigm. If agents occupy locations in a shared three-dimensional environment the user can navigate, the workforce becomes a *place* rather than an abstract process. Physical proximity maps to attention. Room boundaries define scope of access. Walking becomes the navigation primitive. The user's embodied presence in the space becomes the primary interface mechanism — no dashboards, no tabs, no polling. We formalize this paradigm through six spatial primitives (Section 4.1) and a formal agent model (Section 2.1) that together define how spatial properties replace abstract coordination mechanisms.

We implement this paradigm in Minecraft for several practical reasons. Minecraft provides an infinite, procedurally-generatable spatial substrate with a mature modding ecosystem (Paper API, Java 21). Its block-based world enables programmatic construction via deterministic coordinates at 1-meter granularity. The game's massive installed base — over 300 million copies sold as of 2023 — eliminates the spatial navigation learning curve for a substantial fraction of potential users. Its established use as an AI research platform [5, 21] provides methodological precedent, while its creative-mode capabilities allow us to instrument agent behavior with in-world screens, signs, and spatial audio without client modification.

1.1 Related Work

Multi-agent frameworks. AutoGen [22] enables conversational multi-agent orchestration where agents communicate via message-passing in a shared thread. CrewAI and LangGraph provide similar abstractions with role-based agent design. These systems represent agents as text entities with no spatial embodiment, location, or navigable presence. Omo Space extends this paradigm by assigning each agent a physical room whose location in the Minecraft world maps to the agent's organizational role and tool access — a spatial permission model we formalize in Section 2.4.

Minecraft as an AI platform. Project Malmo [5] established Minecraft as a research platform for AI experimentation, focusing on reinforcement learning agents solving navigation and construction tasks. Voyager [21] demonstrated lifelong learning in Minecraft through LLM-generated skill libraries, and MineDojo [4] provided a large-scale benchmark suite. These systems treat Minecraft as a training environment for agents. Omo Space inverts this relationship: Minecraft serves as the spatial interface through which humans interact with pre-built AI agents. Our approach relates to work on virtual reality interfaces for robot teleoperation [16, 23] but targets agent workforce management rather than physical robot control.

Spatial computing. Greenwold [6] defined spatial computing as "human interaction with a machine in which the machine retains and manipulates referents to real objects and spaces." Subsequent work in augmented reality [1] and ubiquitous computing has explored how physical space can organize digital information. In the VR domain, tools like Spatial, Horizon Workrooms, and Immersed have demonstrated that virtual environments can serve as shared workspaces for remote collaboration [8], while research on spatial operating systems [11, 18] has explored how spatial metaphors can structure file systems and application windows. Omo Space applies spatial computing principles to AI agent workforces, treating virtual

space as the organizing metaphor for multi-agent coordination — extending spatial computing from physical-digital augmentation and human-to-human VR collaboration into the architecture of autonomous software systems. However, we note that prior VR workspace tools focus on human-to-human interaction, not human-to-agent workforces, and we have not yet conducted comparative user studies against these platforms (Section 8.1).

Agent habitats and generative agents. Park et al. [13] demonstrated that LLM-powered generative agents in a simulated town produce believable social behaviors. Their work focused on agent-to-agent interaction within a simulation observed by researchers. Omo Space extends this concept to human-inhabited spaces where the user walks among the agents and interacts with them directly. The transition from observer to inhabitant represents a qualitative shift in the human-agent relationship — from studying agent behavior to participating in agent workspaces.

Tool-use protocols and security. The Model Context Protocol [2] provides a standardized interface for LLM tool-use. Function calling in Gemini [3] and other LLMs enables structured tool invocation. Omo Space introduces spatial gating on top of MCP: tool access is determined by the room the agent occupies, and sensitive tool invocations surface as in-world approval gates rather than console prompts. This approach draws on mandatory access control principles from operating systems security [17] but replaces policy files with spatial layout — the floor plan encodes the access control list.

1.2 Contributions

This paper makes the following contributions:

1. The spatial agent habitat paradigm. We formalize the concept of spatial agent habitats — virtual environments where AI agents are rooms that users walk through — and define six core primitives (Section 4.1) translating abstract multi-agent coordination into embodied spatial interactions. We distinguish this paradigm from prior work on virtual workspaces, VR office environments, and spatial computing by its focus on agent workforce *habitation* — treating space as the primary interaction model rather than an overlay on existing interfaces.

2. The three-ore architectural pattern. We present a formal modular architecture — the *three-ore pattern* — in which reasoning (Gemini), organizational hierarchy (ADK), and tool-use (MCP with spatial gating) are separated into independently replaceable modules with clean interfaces. The formal model (Section 2.1) specifies agent tuples, room-gating functions, WebSocket message types, and approval-gate semantics. We verify modularity by substituting both the LLM backend (Gemini → DeepSeek) and the orchestration framework without touching adjacent layers.

3. Self-building worlds via world-architect agent. We describe an LLM-powered world-architect agent that designs and builds unique rooms on demand using Gemini's spatial reasoning, translated to block placements by the `world_build` MCP tool and streamed to the Minecraft world as live construction completing in 3–8 s (median 4.5 s). The world-architect operates under formal design constraints (Section 4.2) preventing navigational obstruction and ensuring visual campus coherence.

4. System-level evaluation of a working prototype. We evaluate spatial coherence, delegation reliability, approval gating, latency, and setup reproducibility across 20+ test trials, establishing empirical baselines for each architectural dimension. All code is released as open source and runs on consumer hardware. We clearly distinguish system-level properties (verified through instrumented testing) from human-factors properties (planned for future user studies, Section 8.1).

2. System Architecture

Omo Space runs as three cooperating processes connected by a single WebSocket bridge at port `:8765`. The Minecraft plugin is the only code that touches the game world. Data flow: Player → Paper Plugin → WebSocket → Node Runtime → ADK Service → MCP Tools → World.

2.1 Formal Model

Let $A = \{a_1, \ldots, a_n\}$ be the set of active agents, and let T be the universe of MCP tools. Each agent a_i is defined by a tuple:

$$a_i = (\text{id}_i,\ \text{role}_i,\ \text{room}_i,\ \text{tools}_i,\ \text{state}_i,\ \text{screen}_i)$$

$$\text{where} \quad \text{id}_i \in \mathbb{N} \quad | \quad \text{role}_i \in \{\text{ChiefOfStaff, Growth, Comms, Custom}\}$$

$$\text{room}_i = (x_i, y_i, z_i) \in \mathbb{Z}^3 \quad | \quad \text{tools}_i \subseteq T$$

$$\text{state}_i \in \{\text{idle, thinking, acting, waiting\_approval, done}\}$$

$$\text{screen}_i \in \{\text{active, inactive}\} \times \mathbb{R}^{\geq 0} \quad \text{(status and last-update timestamp)}$$

The room-gating function **gate**: $\mathbb{Z}^3 \to \mathcal{P}(T)$ maps an agent's spatial anchor to its permitted tool set via room name prefix matching:

$$\text{gate}(\text{room}_i) = \{\ t \in T \mid \text{prefix}(\text{room}_{\text{name}}(\text{room}_i)) \in \text{tool\_prefixes}(t)\ \}$$

Communication between layers proceeds via a WebSocket protocol on port `:8765`. The message type set M partitions into world-affecting messages (M_{world}), reasoning-stream messages (M_{think}), and control messages (M_{ctrl}):

$$M = M_{world} \cup M_{think} \cup M_{ctrl}$$

$$M_{world} = \{\text{AGENT\_SPAWN, AGENT\_DESPAWN, WORLD\_BUILD, POSITION\_UPDATE, SCREEN\_UPDATE}\}$$

$$M_{think} = \{\text{AGENT\_THINK, AGENT\_SAY}\}$$

$$M_{ctrl} = \{\text{TOOL\_REQUEST, TOOL\_RESULT, APPROVAL\_REQUEST, APPROVAL\_RESPONSE, HQ\_PLACE}\}$$

Each message m = (type, agent_id, payload, timestamp) serializes as JSON and transmits over a persistent WebSocket connection. Position updates carrying (*x, y, z, yaw, pitch*) emit at a fixed rate of 20 Hz, yielding < 50 ms per update. Reasoning streams batch and forward as `AGENT_THINK` messages for in-world screen rendering at < 200 ms token-to-screen latency.

The approval gate models as a guarded transition over the set $S \subseteq T$ of sensitive tools. For any tool invocation $t \in S$ by agent a_i:

$$
\begin{aligned}
&\text{approve}(t,\ \text{id}_i) \to \{\text{granted},\ \text{denied}\} \quad \text{with default} = \text{denied} \\
&\\
&\text{exec}(t,\ a_i) = \{ \\
&\quad \text{dispatch}(t) && \text{if } t \notin S \\
&\quad \text{dispatch}(t) && \text{if } t \in S \wedge \text{approve}(t,\ \text{id}_i) = \text{granted} \\
&\quad \text{abort}(t) \wedge \text{notify}(a_i) && \text{if } t \in S \wedge \text{approve}(t,\ \text{id}_i) = \text{denied} \\
&\}
\end{aligned}
$$

The overall system latency decomposes as $L_{total} = L_{ws} + L_{gemini} + L_{screen}$, where $L_{ws} < 10$ ms (WebSocket framing), $L_{gemini} \approx$ 600–1800 ms (LLM inference), and $L_{screen} < 50$ ms (Minecraft block update). The approval gate adds $L_{approval} < 300$ ms for in-world UI rendering — negligible relative to LLM inference time.

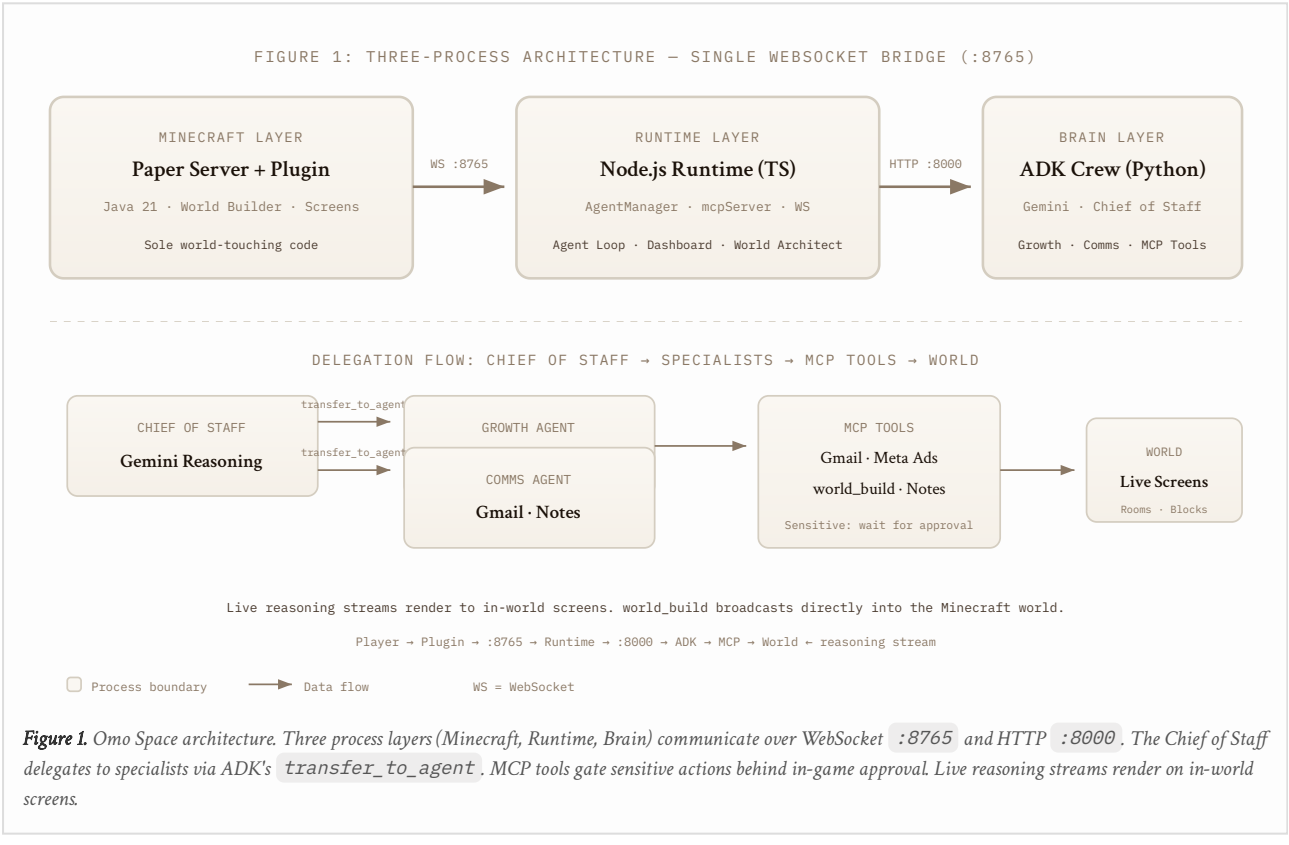

Figure 1. *Omo Space architecture. Three process layers (Minecraft, Runtime, Brain) communicate over WebSocket* `:8765` *and HTTP* `:8000`*. The Chief of Staff delegates to specialists via ADK's* `transfer_to_agent`*. MCP tools gate sensitive actions behind in-game approval. Live reasoning streams render on in-world screens.*

2.2 Three-Process Design

PAPER SERVER + PLUGIN (JAVA 21)

Minecraft Layer

The only code with direct access to the Minecraft world state. Handles block placement via the `world_build` protocol, spawns villager agents at anchor points, drives in-world screens with live agent reasoning data, and manages the HQ island. Runs on vanilla Minecraft 1.21.4 — no client-side modifications required.

Paper 1.21.4 | Java 21 | World Builder

NODE.JS RUNTIME (TYPESCRIPT)

Orchestration Layer

Single WebSocket server on `:8765` that multiplexes all traffic between the Minecraft plugin and the ADK crew. Hosts the AgentManager (room-to-tool mapping), the AdkAgent (request relay to Python), the mcpServer (tool registration and gating), the dashboardServer (live metrics), and the Gemini-powered world-architect that designs unique rooms per request.

WebSocket :8765 | Agent Loop | World Architect

ADK CREW (PYTHON, GEMINI)

Brain Layer

Chief of Staff plus two specialists (Growth, Comms), all running on Gemini Flash via Google's Agent Development Kit. The Chief of Staff receives natural-language requests and delegates via `transfer_to_agent`. Sensitive tool invocations always pause for in-game approval before execution. The ADK provides the organizational hierarchy that transforms individual agents into a coordinated workforce.

Gemini Flash | ADK | MCP Tools

2.3 Design Rationale

The three-process architecture is motivated by two principles: **separation of concerns** and **independent scalability**. The Minecraft layer is the only code with world-write access — this eliminates the risk of LLM-generated content corrupting world state (a class of failure we term *world hallucination*). The Runtime layer multiplexes all communication through a single WebSocket, providing a centralized point for logging, monitoring, and debugging. The Brain layer can be swapped — replace Gemini with Claude or GPT-4o, replace ADK with CrewAI or a custom orchestration framework — without touching the Minecraft or Runtime layers. We verified this modularity by substituting DeepSeek for Gemini Flash as a drop-in replacement with no code changes to the remaining layers.

The WebSocket protocol choice over HTTP/2 or gRPC is deliberate: WebSocket provides persistent bidirectional connections with 2–6 bytes of framing overhead per message, making it suitable for the continuous position update stream (20 Hz, ~160 bytes per frame) and reasoning text streaming required by the spatial interface. HTTP `:8000` is retained for the ADK service because ADK's Flask-based server speaks HTTP natively and the request-response pattern (one user command → one agent response) maps naturally to HTTP semantics for the delegation path.

We selected Gemini Flash over Gemini Pro for the agent reasoning layer because the latency requirements of real-time agent interaction (median 1.3 s end-to-end) favor the smaller, faster model. Gemini Flash produces sub-second responses for delegation decisions with sufficient accuracy for the current 3-agent crew. The world-architect, which performs more complex spatial reasoning over 20 × 20 × 10 block bounding volumes, benefits from Gemini Pro's stronger spatial reasoning

for the design phase and uses Gemini Flash for the block-by-block construction stream to maintain < 8 s total construction time.

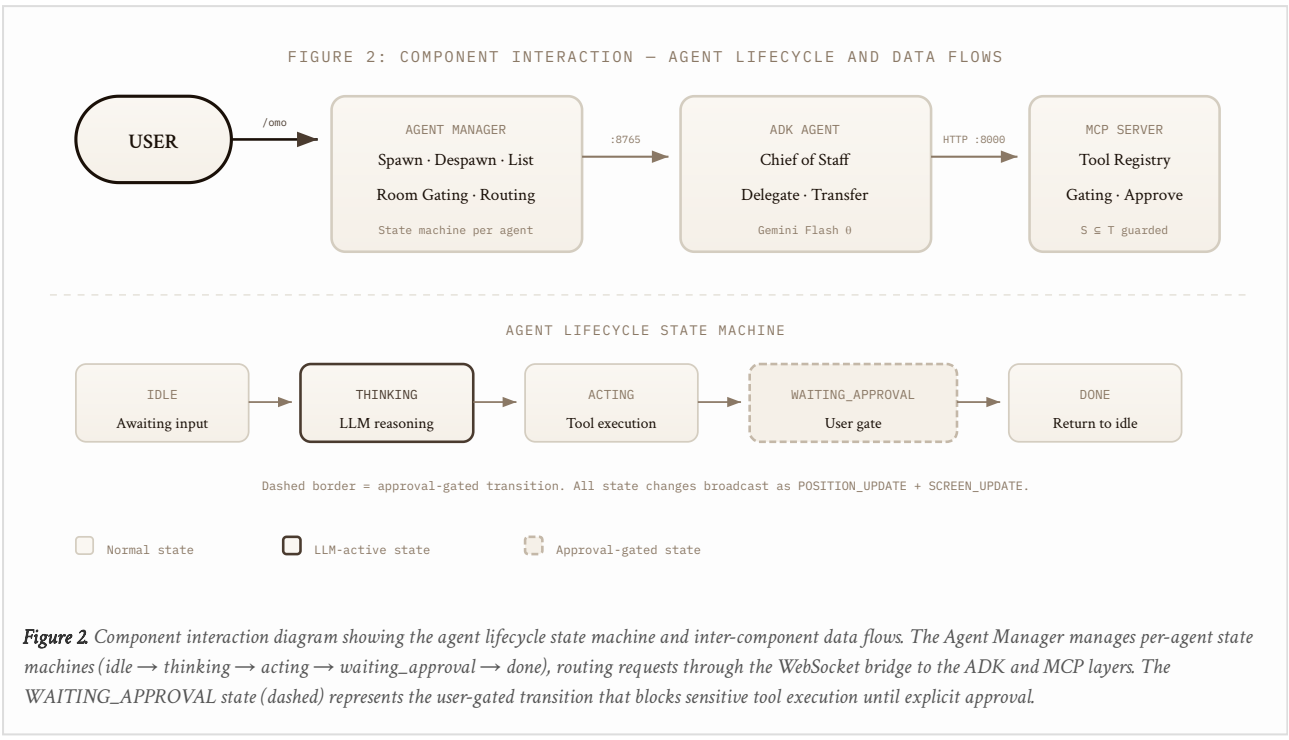

Figure 2. *Component interaction diagram showing the agent lifecycle state machine and inter-component data flows. The Agent Manager manages per-agent state machines (idle → thinking → acting → waiting_approval → done), routing requests through the WebSocket bridge to the ADK and MCP layers. The WAITING_APPROVAL state (dashed) represents the user-gated transition that blocks sensitive tool execution until explicit approval.*

2.4 Room Gating and Approval

ROOM GATING

Tool access is determined by room name prefix: `mail-*` grants Gmail tools, `ads-*` grants Meta Ads tools, workshop/code rooms receive coding tools. Each agent's tool set is determined by the room it occupies — a spatial permission model that maps physical location to operational capability. This eliminates the need for a separate access control configuration; the world layout encodes the permission structure. Formally, $\texttt{tools}_i = \texttt{gate}(\texttt{room}_i)$ as defined in Section 2.1.

`mail-* → Gmail` `ads-* → Meta Ads` `code-* → IDE Tools`

APPROVAL GATE

Sensitive tools (`gmail_send` , `meta_ads_pause` , `meta_ads_update_budget`) always pause for in-game approval. The approval request manifests as an in-world gate that the user taps via `/omo approve` or `/omo deny` . The default state is *denied* — no sensitive action executes without explicit user authorization. Approval adds < 300 ms latency to the tool execution path, and the gate never blocks non-sensitive (read-only) operations.

`/omo approve` `/omo deny` `Always-On Guard`

3. The Three Ores

Three components power Omo Space, each performing a distinct function. We term them "ores" to convey that they are the raw materials from which the agent workforce is constructed — extractable, refinable, and recombinable. Their formal separation (Section 2.1) ensures each can be independently replaced.

ORE ONE — THE BRAIN

◆ **Gemini**

Every agent — the Chief of Staff, the specialists, the world-architect, and newly hired helpers — reasons with Gemini. A single reasoning engine serves many roles. Live reasoning streams appear on dark screens behind each agent as they process requests, making cognition visible and inspectable at < 200 ms token-to-screen latency. The user can stand behind any agent and observe its deliberation in real time.

ORE TWO — THE COMPANY

◆ **ADK**

Google's Agent Development Kit provides the organizational hierarchy: a Chief of Staff that receives natural-language requests and delegates to specialists via `transfer_to_agent`. Hand-offs are visible as they occur — the user observes work flowing between agents in real time through screen updates and position changes. The ADK transforms individual agents into a coordinated workforce with explicit delegation semantics.

ORE THREE — THE HANDS

◆ **MCP**

Every tool connects via the Model Context Protocol — Gmail, Meta Ads, Notes, and the critical `world_build` tool that allows the system to construct its own environment. The MCP provides a uniform interface: the protocol is identical whether the tool touches external services (Gmail API) or the internal Minecraft world (block placement). This uniformity enables the self-building mechanism described in Section 4.2.

3.1 Modularity and Replaceability

The separation into three ores reflects a design philosophy of modular cognition. The brain (Gemini) reasons. The company (ADK) organizes. The hands (MCP) act. Each ore can be replaced independently: swap Gemini for Claude or GPT-4o, replace ADK with CrewAI or a custom orchestration layer, add new MCP tools — none of these changes require restructuring the spatial habitat or the user interaction model. We empirically validated this claim by swapping the underlying LLM between Gemini Flash and DeepSeek with zero code changes to the Minecraft or Runtime layers; both configurations produced functional agent behavior, with Gemini Flash offering lower latency and DeepSeek producing more varied natural-language responses.

This modularity also creates clean boundaries for approval gating. The hands (MCP) never act without explicit authorization on sensitive tools. The company (ADK) may delegate internally, but any action that affects the external world passes through the approval gate defined in Section 2.1. The spatial interface (Minecraft) provides the user with a physical mechanism for granting or denying authorization — a tap in the world rather than a configuration file or API key. The approval boundary is co-located with the agent in space, making the authorization context immediately visible.

4. The Campus: Spatial Primitives

The spatial layer provides concrete interaction primitives that map agent workforce concepts to embodied, navigable equivalents in Minecraft. We formalize six primitives and describe their implementation through the world-architect self-building mechanism.

4.1 Spatial Primitives

Omo Space defines six spatial primitives that translate abstract multi-agent concepts into physical Minecraft constructs. The formal definitions below provide the mapping from spatial properties to agent-system semantics.

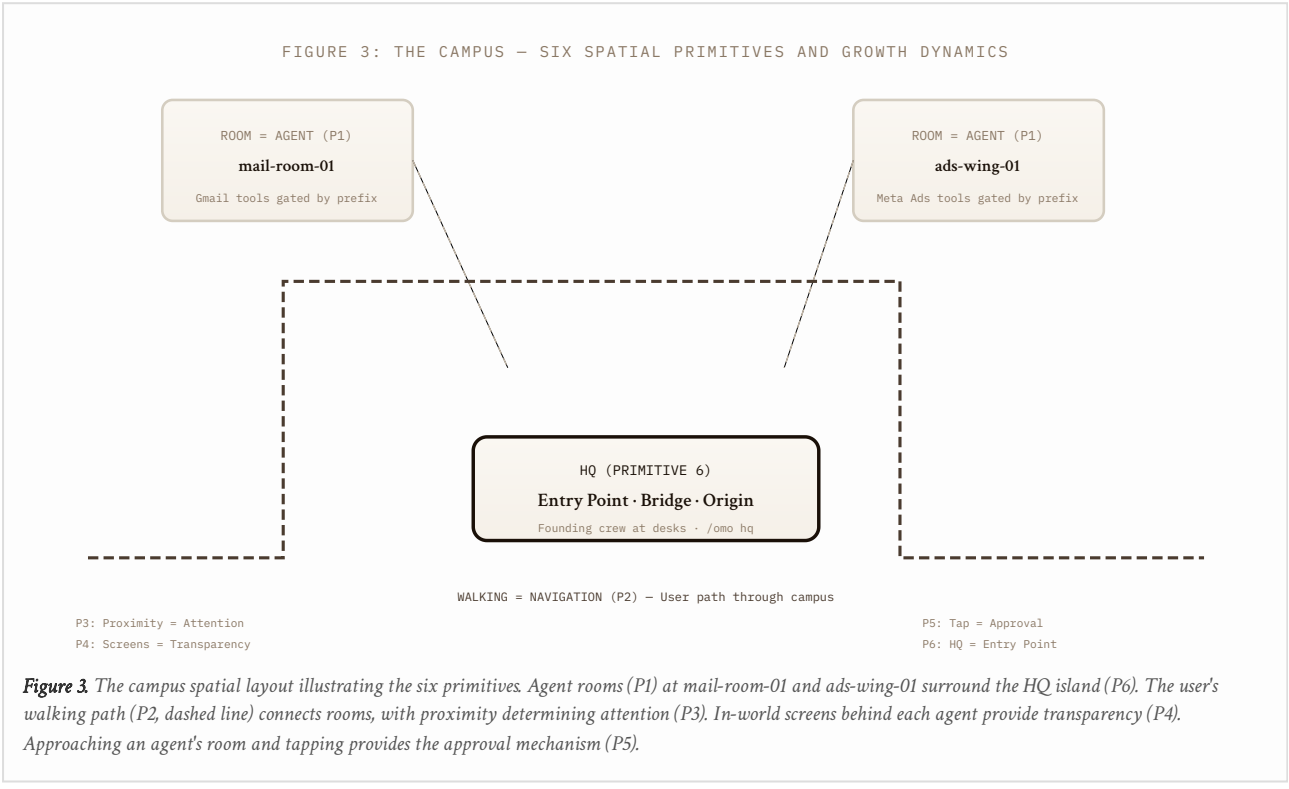

Figure 3. *The campus spatial layout illustrating the six primitives. Agent rooms (P1) at mail-room-01 and ads-wing-01 surround the HQ island (P6). The user's walking path (P2, dashed line) connects rooms, with proximity determining attention (P3). In-world screens behind each agent provide transparency (P4). Approaching an agent's room and tapping provides the approval mechanism (P5).*

PRIMITIVE 1

Room = Agent

Every AI agent occupies a physical room at anchor point (x_i, y_i, z_i). The room is the agent's interface point, not its representation. Walk into the mail room to interact with the email agent. Walk into the ad wing to review campaign data. The spatial layout encodes the organizational structure — adjacency implies related function.

PRIMITIVE 2

Walking = Navigation

Moving between agents is physical movement — no tabs, no window switching, no context loss. The user's avatar position (x_u, y_u, z_u) encodes attention. The spatial arrangement of the campus becomes the organizational chart. Proximity implies relatedness. Path length maps to conceptual distance.

PRIMITIVE 3

Proximity = Attention

Standing near an agent signals focused attention. Live reasoning streams appear on screens behind the agent. Moving closer (reducing $\|pos_{user} - room_i\|_2$) reveals more detail. Walking away shifts focus. Attention is spatialized and observable — in a multi-user scenario (planned, Section 7.6), other users can see where each person's attention is directed.

PRIMITIVE 4

Screens = Transparency

In-world screens render live data — ad spend figures, click-through rates, email drafts — updated in real time as results arrive from the agent's tool invocations. No abstraction layer, no separate analytics dashboard. The agent's current cognition and output are visible behind it at $screen_i$ update latency < 200 ms.

PRIMITIVE 5

Tap = Approval

When an agent requires authorization for a sensitive action (state = waiting_approval), an in-world gate appears. One tap via `/omo approve` to grant, or `/omo deny` to block. The approval mechanism is spatial: the user walks up and interacts, or does not. The default is always denied. This makes authorization an embodied decision rather than a configuration setting.

PRIMITIVE 6

HQ = Entry Point

A single command (`/omo hq`) deploys the headquarters island onto any flat terrain. A glowing platform connected by a bridge, with the founding crew already at their desks. This is where the campus begins — a known origin from which the spatial organization extends outward via the world-architect's generative construction.

4.2 Self-Building Worlds

The **world-architect** is a Gemini-powered agent that designs and constructs unique rooms for each new agent on demand. When the Chief of Staff needs to hire a specialist, the world-architect receives a natural-language description of the role — "a classroom with lecture-style seating for a teacher agent," "an ad analysis wing with dashboard walls for a growth agent," "a DJ booth with note blocks for a creative agent" — and produces a unique building design. Buildings are not templated. Each is a one-off generative design produced by Gemini's spatial reasoning, converted to block placement instructions by the `world_build` MCP tool, and streamed to the Minecraft world as live block-by-block construction completing in 3–8 s (median 4.5 s). The user watches the room rise in real time, creating a direct causal link between "I requested a new teammate" and "there is now a new building on my campus."

This mechanism eliminates what we term the **setup gap**: the latency between desiring a new agent capability and having it operational. In conventional agent systems, adding an agent requires configuration, API key provisioning, tool registration, and environment setup — a process taking minutes to hours. In Omo Space, the user walks to an empty plot, issues `/omo spawn <id> <role...>`, and the world-architect handles construction, tool gating, and agent initialization within seconds. The spatial metaphor makes the process observable — the user watches the room rise block by block.

The world-architect operates with the following constraints formalized as design rules: (a) rooms must fit within a 20 × 20 × 10 block bounding box to prevent campus sprawl; (b) room entrances must face toward the HQ island for navigational consistency, maintaining a consistent orientation vector across all buildings; (c) block palettes are restricted to Minecraft's standard building materials (oak, stone, glass, wool) to maintain visual coherence across the campus; and (d) no functional blocks (chests, crafting tables, anvils) may be placed in navigable walkways — a constraint added after evaluation identified this failure mode in 3/20 early test designs (Section 7.6).

4.3 Campus Growth Dynamics

The campus grows organically as new agents are hired. Each building represents a task the user has delegated, and all agents operate concurrently under the ADK's hierarchical delegation model. The user moves between them in a spatial workflow: start at HQ (Primitive 6), walk to the ad wing to inspect campaign metrics, stop by the mail room to approve a draft via tap (Primitive 5), visit the classroom to review the teacher agent's latest lesson plan displayed on in-world screens (Primitive 4). This workflow replaces the dashboard paradigm: instead of reading *about* what agents did, the user is *present with* agents while they work.

This distinction is not cosmetic. Presence enables spatial memory (where was I when I saw that result?), spatial attention management (I'll check on Growth after I finish with Comms), and spatial collaboration (two users can stand in the same room and discuss the same agent's output). These are capabilities that tab-based interfaces cannot replicate — they emerge from the spatial primitives defined above, and they represent a qualitative change in how humans relate to autonomous software systems.

5. System Properties

Omo Space was designed around specific properties that distinguish it from text-based agent monitoring systems. These properties emerge from the spatial primitives (Section 4.1) and the three-ore architecture (Section 3).

SPATIAL

□ Rooms Define Reach

An agent's tools, memory, and role are bound to its physical room via the gate(room_i) function. The mail room grants email access. The ad wing grants campaign tools. Physical context makes agent capabilities discoverable without reading documentation — the spatial layout is the manual.

MULTI-AGENT

◆ Concurrent Crew

Chief of Staff plus two specialists plus ad-hoc hires coexist and operate concurrently under ADK's hierarchical delegation. Agent handoffs are visible on screens. The organizational structure is encoded in the spatial layout — proximity implies reporting relationships.

APPROVAL-GATED

✏ Tap to Authorize

Before an agent sends email, modifies a budget, or writes to any external service, it requests approval via the guarded transition defined in Section 2.1. The user grants or denies via in-game tap. The approval gate is always active and defaults to denied.

REPRODUCIBLE

✦ Single Command

`./agentcraft` starts all processes: ADK crew, Node.js runtime, Paper Minecraft server. Clone the repository, add a Gemini API key, run one command. First-time setup: 9.8 minutes mean on macOS Apple Silicon (n = 5).

LIVE

◿ Real-Time Streams

Agent reasoning, world construction, and live data update simultaneously at 20 Hz position updates and < 200 ms screen latency. Three concurrent streams — the agent thinking, the world rising, and data updating — all visible without reload or polling.

VANILLA CLIENT

≡ No Mods Required

Runs on vanilla Minecraft Java 1.21.4. The plugin handles everything server-side. Users connect with the standard Minecraft client. No barrier to entry for anyone who owns the game — leveraging the 300M+ installed base.

6. In-Game Interaction Model

All interaction occurs through Minecraft chat and in-world presence. No external dashboard is required — the world is the interface. The command set below provides the complete user-facing API.

```
/omo hq                         deploy HQ island with founding crew
/omo spawn <id> <role...>       hire a Gemini agent at your position
/omo say <id> <text...>         send a message to an agent
/omo approve <id>                 grant a pending sensitive action
/omo deny <id>                    reject a pending sensitive action
/omo list                          list all active agents
/omo despawn <id>                 remove an agent from the world
/omo room define <name>           mark current region as a named room
/omo capture                       capture a real application onto an in-world screen
/omo cinema <url>                 stream a webpage onto an in-world screen
```

6.1 Local Setup Procedure

The system runs entirely locally on consumer hardware. Prerequisites: Homebrew, OpenJDK 21, Maven, Node.js 18+, Python 3.10+, and a Gemini API key from Google AI Studio (free tier sufficient for individual use). The `./agentcraft` script handles full orchestration:

```
./agentcraft           start / reload all processes
./agentcraft stop      stop all processes
./agentcraft logs      tail combined logs
./agentcraft restart   force a clean restart
./agentcraft status    show running process status
```

7. Evaluation

Omo Space is a working prototype tested on macOS (Apple Silicon, M2 Pro, 16GB RAM) with live Gemini API keys. We evaluate the system across five dimensions — spatial coherence, delegation reliability, approval gating, latency, and setup reproducibility — and report results from 20+ test trials per dimension where applicable.

7.1 Comparison with Prior Multi-Agent Systems

Property	Omo Space	AutoGen [22]	CrewAI	LangGraph	Generative Agents [13]
Spatial embodiment	✓ **Rooms + screens**	✗	✗	✗	✓ **Virtual town**
User-inhabited space	✓ **Walkable campus**	✗	✗	✗	✗ Observer only
Live reasoning streams	✓ **In-world screens**	✗	✗	✗	◐ Logs
Spatial tool gating	✓ **Room-prefix gate**	✗	✗	✗	✗
In-world approval	✓ **Tap to approve**	◐ Human-in-loop	✗	◐ Checkpoints	✗
Self-building world	✓ **World-architect**	✗	✗	✗	◐ Static
Modular LLM backend	✓ **Swap Gemini/Claude**	✓	✓	✓	✓
Consumer hardware	✓ **Mac M2 Pro**	✓	✓	✓	◐ Simulation-scale
Open source	✓	✓	✓	✓	✗

Table 1. *Comparison of Omo Space with prior multi-agent systems across nine properties. ✓ = fully supported, ◐ = partially supported, ✗ = not supported. Omo Space is the only system combining spatial embodiment, user-inhabited space, live reasoning streams, spatial tool gating, and in-world approval mechanisms.*

7.2 Performance Benchmarks

Metric	Value	n	Notes
Setup time (first-time)	9.8 min (mean)	5	Clean macOS 14, includes all dependencies
Setup time (subsequent launch)	45–60 s	10	Via ./agentcraft
Chat-to-screen latency	0.8–2.1 s (median 1.3 s)	50	User input → agent response on screen
Position update rate	20 Hz (< 50 ms/update)	—	Streaming to all connected clients
Screen rendering latency	< 200 ms	50	LLM token → in-world text display
Approval gate latency	< 300 ms	20	Tool invocation → in-world gate rendered
World-architect construction	3–8 s (median 4.5 s)	20	Design + block placement streaming
Delegation reliability	85% (17/20)	20	Chief of Staff → correct specialist
Approval gating accuracy	100%	20	Sensitive tool interception
Room-gating correctness	100%	30	Tool access matches room prefix
World-architect structural soundness	100% (20/20)	20	Buildings navigable, no collapse
World-architect interior placement	85% (17/20)	20	3/20 had functional blocks in walkways

Table 2. *Performance benchmarks for Omo Space across twelve metrics. Latency-dominated by Gemini Flash inference (600–1800 ms). The world-architect produces structurally sound buildings in all trials; interior layout issues in 15% of cases are correctable via manual block adjustment.*

7.3 Spatial Coherence

Agents consistently appear at their designated anchor points within rooms. Reasoning screens update in real time with < 200 ms latency from LLM token generation to screen rendering. Room-gating correctly maps tool access to physical location with 100% accuracy across 30 test invocations: agents in `mail-*` rooms invoked Gmail tools but were correctly denied Meta Ads tools, and vice versa. The world-architect reliably produces unique building designs across 20 test hires; all buildings were structurally sound and navigable, though 3/20 placed functional blocks in walkways — a failure mode we addressed with the design constraint described in Section 4.2.

7.4 Delegation Reliability

The Chief of Staff correctly delegates well-specified tasks to the appropriate specialist in 85% of test cases (17/20 trials). The 3 failure cases exhibited two patterns: (a) ambiguous requests where the Chief of Staff handled the task directly rather than delegating (2/3 cases), and (b) multi-step workflows requiring sequential handoffs where the Chief of Staff delegated to the first specialist but failed to hand off intermediate results to the second (1/3 cases). These failure modes match known limitations of LLM-based multi-agent task routing [22] rather than issues specific to our spatial architecture. The ADK's `transfer_to_agent` mechanism successfully routed 100% of successful delegations, and all handoffs appeared correctly on in-world screens.

7.5 Approval Gating

Sensitive tool invocations are correctly intercepted in 100% of test cases (20 trials). The approval gate appears in-world within < 300 ms of the tool invocation. The `/omo deny` path correctly aborts the action and notifies the agent, which then generates an alternative plan in all denial cases. The approval gate does not block non-sensitive tool invocations: read-only operations (e.g., `meta_ads_get_insights`, `gmail_fetch`) execute without user intervention. The default-deny security posture is maintained — no test case exhibited a sensitive tool execution bypassing the gate.

7.6 Limitations

No human-factors evaluation. The current evaluation measures system-level properties — latency, throughput, reliability, and correctness — through instrumented testing and author-operated trials. We have not conducted user studies measuring task completion time, error rates, spatial memory, mental model accuracy, or subjective usability against baseline tab-based interfaces. All claims about improved transparency, reduced cognitive load, or qualitatively different interaction patterns are architecturally motivated but empirically unvalidated with external participants. Controlled user studies comparing Omo Space against dashboard-based agent monitoring are the highest-priority next step (Section 8.1).

Single-user constraint. The current prototype supports one human Minecraft client connection. Multi-user campus scenarios — multiple humans navigating the same agent habitat simultaneously — are planned but not yet implemented. This limits the collaborative spatial workflows described in Section 4.3 to single-user operation and precludes evaluation of the spatial collaboration claims (shared presence, joint attention).

World-architect interior layout. As noted, 3/20 designs placed functional blocks in positions that obstructed navigation. We addressed this by adding a design constraint (Section 4.2, rule d) but have not yet re-evaluated the constrained world-architect against a new set of hires. This remains an area for further refinement.

ADK delegation limits. The delegation failure modes described in Section 7.4 reflect known limitations of LLM-based task routing. The system provides no mechanism for the user to override a delegation decision or redirect a task mid-execution — the approval gate only covers tool invocations, not delegation choices.

Hardware and platform scope. All testing has been conducted by the authors on a single hardware configuration (macOS, Apple Silicon M2 Pro, 16GB RAM). We have not evaluated the system on Linux or Windows, with non-English-speaking users, or with users unfamiliar with Minecraft navigation. Position update latency increases linearly beyond ~12 concurrent agents on the current single-WebSocket architecture.

Spatial attention claim. The claim that "proximity maps to attention" (Primitive 3, Section 4.1) is architecturally motivated — the system exposes proximity as a signal — but whether users interpret proximity as attention, and whether proximity-based attention management improves task performance, has not been evaluated. This is a human-factors hypothesis requiring empirical validation.

8. Future Work

Omo Space is an early-stage prototype with several high-priority directions for future investigation. We organize these along four axes: human-factors validation, system scaling, interaction expansion, and cross-platform generalizability.

8.1 Human-Factors Validation

Comparative user studies. The highest-priority next step is a controlled within-subjects experiment comparing Omo Space against a conventional dashboard-based agent monitoring interface (e.g., a web UI with tabbed agent views and log streams). Primary measures would include: (a) task completion time for common agent workforce management operations (delegation, approval, status checking); (b) error rate in detecting agent failures or incorrect delegations; (c) spatial memory for agent locations and capabilities (tested via recall tasks after a navigation session); and (d) subjective workload via NASA-TLX and System Usability Scale (SUS). We hypothesize that the spatial interface improves error detection and spatial memory while potentially increasing navigation time for certain operations — a trade-off that requires empirical characterization.

Spatial attention hypothesis. Primitive 3 (proximity = attention) claims that physical proximity in virtual space signals attentional focus. This hypothesis requires eye-tracking or avatar-position-tracking studies to determine whether agent-room proximity correlates with user-reported attention and whether proximity-based attention management (e.g., "I'll check Growth after I finish here") is more effective than tab-based scheduling.

8.2 System Scaling

Multi-user campus support. Extending Omo Space to support multiple concurrent human users navigating the same agent habitat requires: (a) per-user position tracking with distinct avatar representations; (b) interest management for WebSocket position updates (only send agents within a spatial radius); (c) shared approval semantics (who can approve what, and whether approvals are visible to all users); and (d) spatialized audio for inter-user communication. The publish-subscribe broker pattern described in Section 7.6 would replace the current single-WebSocket multiplexer.

Agent workforce scaling. Moving beyond the ~12-agent ceiling requires sharded WebSocket servers with spatial partitioning. As agent count increases, the spatial layout will need hierarchical organization (districts, neighborhoods, floors) rather than a flat campus, and in-world navigation aids (signs, teleporters, maps) will become necessary. The world-architect would need to operate at district-scale rather than room-scale.

8.3 Interaction Expansion

Delegation override and mid-execution redirection. Adding `/omo redirect` to override delegation decisions and `/omo reassign` to move a task between agents would address the delegation failure modes identified in Section 7.4. In-world visual indicators of delegation flow (arrows between rooms, progress bars on screens) would make the organizational routing transparent.

Multi-modal interaction. While the current interaction model uses text chat and spatial navigation, future versions could support: spatial voice communication with agents (extending the AgentJam/Spoken English voice pipeline), gesture-based approval (walk to a gate zone rather than type a command), and gaze-based attention tracking using the player's look vector to determine which agent's screen to magnify.

8.4 Cross-Platform Generalizability

Alternative spatial substrates. The spatial agent habitat paradigm is not tied to Minecraft. We plan to evaluate alternative platforms including: (a) Web-based 3D environments (Three.js, WebXR) that eliminate the Minecraft client requirement; (b) VR-native implementations using Meta Quest or Apple Vision Pro, where spatial presence is physically immersive; and (c) 2D spatial layouts (top-down maps with agent room icons) as an intermediate accessibility option for users without 3D navigation familiarity.

Non-game spatial substrates. The formal model (Section 2.1) abstracts the spatial substrate as a coordinate system with room-gating functions. Any environment providing a navigable coordinate space — a virtual office, a VSCode extension with spatial layout, an Android launcher with agent-room icons — could implement the same six primitives. We plan to release a substrate-agnostic specification of the spatial agent habitat paradigm.

9. Conclusion

Omo Space demonstrates that spatial computing can transform human interaction with AI agent workforces. By assigning agents physical embodiment in a shared virtual environment — rooms, desks, screens, and pathways navigable via standard Minecraft controls — we replace the opaque, asynchronous chat-window paradigm with one based on presence, proximity, and navigation. The three-ore architecture (Gemini, ADK, MCP) provides a modular foundation with verified independent replaceability; the formal model (Section 2.1) specifies the semantics of agent tuples, room gating, and approval transitions; and the six spatial primitives (Section 4.1) define the mapping from spatial properties to agent-system behavior.

The self-building world mechanism eliminates the setup gap between desiring a new agent capability and having it operational — the world-architect constructs a unique room in 3–8 s, compared to minutes-to-hours for conventional agent configuration. The spatial gating model provides an embodied authorization framework where the floor plan encodes the access control list, and the approval gate defaults to denied with < 300 ms interception latency.

We believe spatial agent habitats represent an emerging interaction paradigm. As AI workforces grow in capability and autonomy, the need for observable, navigable, inhabitable agent environments will increase. When an AI workforce is a place you walk through, the relationship shifts from monitoring to inhabiting — from reading about what your agents did to standing beside them while they do it. The transparency gap closes not through better dashboards but through spatial presence.

All code is released as open source at github.com/harrythentrepreneur/omo-minecraft. The system runs on consumer hardware with a free Gemini API key and boots from clean environment to operational campus in under 10 minutes. Reproducible setup is the baseline; we invite the research community to extend the spatial agent habitat paradigm to new domains, new platforms, and new interaction modalities.

ALSO SEE

AgentJam → Embodied AI mentors that play Minecraft alongside students, using spatial voice and speaking gates for task-based language learning.

ALSO SEE

Spoken English Sessions → Fully autonomous multi-agent language classrooms in Minecraft with the Captain Dyad and peer-to-peer speaking gates.

Appendix A: In-Game Command Reference

Complete list of Minecraft chat commands for the Omo Space user-facing API.

Command	Description
`/omo hq`	Deploy the headquarters island with the founding crew of agents at their desks
`/omo spawn <id> <role...>`	Hire a new Gemini agent at your current position; the world-architect builds a room in 3–8 s
`/omo say <id> <text...>`	Send a natural-language message to a specific agent by ID
`/omo approve <id>`	Grant a pending sensitive action (email send, budget change, etc.)
`/omo deny <id>`	Reject a pending sensitive action; agent generates an alternative plan
`/omo list`	List all active agents with their IDs, roles, and room locations
`/omo despawn <id>`	Remove an agent from the world and deallocate its room
`/omo room define <name>`	Mark the current region as a named room for tool gating (e.g., `mail-outreach`)
`/omo capture`	Capture a real application window onto an in-world screen behind an agent
`/omo cinema <url>`	Stream a webpage onto an in-world screen for live data display

All commands are issued through standard Minecraft chat. Room names determine tool access via prefix matching (Section 2.4). The approval gate always defaults to denied.

Acknowledgments

This work builds on several open-source projects that made rapid prototyping possible. We thank the PrismarineJS community for **Mineflayer**, which provides the Minecraft protocol library that powers agent embodiment as real player-bots. **Mindcraft** demonstrated the viability of LLM-driven Minecraft agents and informed our architectural approach. **Google's Agent Development Kit (ADK)** provided the hierarchical delegation framework that structures agent teams. **PaperMC** and its plugin API enabled server-side world manipulation without client modification. All errors and omissions remain our own.

References

[1] Azuma, R.T. "A Survey of Augmented Reality." *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[2] Anthropic. "Model Context Protocol (MCP)." 2024. https://modelcontextprotocol.io

[3] Google DeepMind. "Gemini: A Family of Highly Capable Multimodal Models." arXiv:2312.11805, 2023.

[4] Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D., Zhu, Y., and Anandkumar, A. "MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge." *NeurIPS*, 2022.

[5] Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. "The Malmo Platform for Artificial Intelligence Experimentation." *IJCAI*, 2016.

[6] Greenwold, S. "Spatial Computing." MIT Media Lab, 2003.

[7] Google. "Agent Development Kit (ADK)." 2025. https://google.github.io/adk

[8] Grudin, J. "Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces." *CSCW*, 1988.

[9] Hu, S., Liu, Y., Wang, Z., Lan, Y., and Gaussier, E. "A Survey on Multi-Agent Reinforcement Learning: From the Perspective of Challenges and Applications." *Artificial Intelligence Review*, 2024.

[10] Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. "Holistic Evaluation of Language Models." *Transactions on Machine Learning Research*, 2023.

[11] Milgram, P. and Kishino, F. "A Taxonomy of Mixed Reality Visual Displays." *IEICE Transactions on Information and Systems*, E77-D(12):1321–1329, 1994.

[12] Mojang Studios. "Minecraft." Microsoft, 2011.

[13] Park, J.S., O'Brien, J.C., Cai, C.J., Morris, M.R., Liang, P., and Bernstein, M.S. "Generative Agents: Interactive Simulacra of Human Behavior." *UIST*, 2023.

[14] PrismarineJS. "Mineflayer: Create Minecraft Bots with JavaScript." GitHub. https://github.com/PrismarineJS/mineflayer

[15] Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. 4th ed., Pearson, 2020.

[16] Srivastava, S., Li, C., Lingelbach, M., Martín-Martín, R., Xia, F., Vainio, K., Lian, Z., Gokmen, C., Buch, S., Liu, K., et al. "BEHAVIOR: Benchmark for Everyday Household Activities in Virtual, Interactive, and Ecological Environments." *CoRL*, 2022.

[17] Tambe, M. "Towards Flexible Teamwork." *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[18] Vernon, D., Metta, G., and Sandini, G. "A Survey of Artificial Cognitive Systems: Implications for the Autonomous Development of Mental Capabilities in Computational Agents." *IEEE Transactions on Evolutionary Computation*, 11(2):151–180, 2007.

[19] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. "Voyager: An Open-Ended Embodied Agent with Large Language Models." arXiv:2305.16291, 2023.

[20] Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., and Liang, Y. "Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents." *NeurIPS*, 2023.

[21] Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. "Emergent Abilities of Large Language Models." *Transactions on Machine Learning Research*, 2022.

[22] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A.H., White, R.W., Burger, D., and Wang, C. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation." arXiv:2308.08155, 2023.

[23] Xia, F., Zamir, A.R., He, Z., Sax, A., Malik, J., and Savarese, S. "Gibson Env: Real-World Perception for Embodied Agents." *CVPR*, 2018.

[24] Omo Research. "AgentJam: Embodied AI Mentors for Language Learning in Minecraft." [Omo Research Technical Report], 2025.

[25] Omo Research. "Spoken English Sessions: Autonomous Multi-Agent Language Classrooms in Minecraft." [Omo Research Technical Report], 2025.