Minecraft AI Whitepaper

https://github.com/aeromechanic000/minecraft-ai-whitepaper

Minecraft AI: Toward Embodied Turing Test Through AI Characters

Minecraft AI

June 5, 2025

Preface

The Minecraft AI project was born from a simple but powerful realization: with the emergence of advanced large language models (LLMs), virtual agents are no longer confined to pre-scripted behaviors or fixed dialogue trees. For the first time, we can build characters in games who are not only reactive but reflective—agents who learn, adapt, collaborate, and grow alongside human players. These are not just NPCs, but potential companions, mentors, builders, and storytellers.

We believe that Minecraft, with its open-ended, persistent, and modifiable world, offers a uniquely fertile ground for advancing this vision. It is more than a game—it is a universal sandbox that bridges technical creativity and social interaction. By designing intelligent AI characters within this space, we are not only making play more immersive, but also prototyping how humans and machines may co-exist in richer digital societies.

At the core of the Minecraft AI project is the conviction that smart agents in virtual environments can become meaningful contributors to future communication platforms, online collaboration systems, and real-world productivity pipelines. Through memory, reasoning, dialogue, and embodiment, these agents are being equipped not just to simulate intelligence, but to participate in it. We view this work as a crucial step toward more natural, more capable, and more trustworthy AI systems that can assist in education, design, research, and everyday tasks.

To support this long-term goal, Minecraft AI aims to provide a robust ecosystem of tools, libraries, and modular frameworks that empower developers, modders, students, and researchers alike. Our platforms are designed to make it easy to create expressive, personalized AI characters; to explore new agent architectures; and to test cutting-edge theories of cognition, memory, planning, and social reasoning—grounded in the tangible affordances of a real game world. We are committed to a few core principles:

- **Openness:** Our primary platforms are open-source and community-driven, enabling shared innovation and transparent experimentation.
- **Stability:** We prioritize long-term stability to support real production deployments—whether you're building a classroom companion, a game mod, or a scientific experiment.
- **Accessibility:** We aim to make powerful AI tools available to all, and will continue to release key AI character capabilities as free Minecraft mods.

Minecraft AI is not just a research project or software toolkit—it is a collective effort to imagine and build a future where AI characters are more than background figures. They are active participants in digital worlds, shaping the way we learn, create, and play. We invite you to join us in this journey.

Minecraft AI
2025

Contents

Preface **1**

1 Introduction: Embodied AI and Minecraft **6**
1.1 Overview . . . 6
1.2 History of Embodied AI . . . 8
1.2.1 From Behavior-Based Robotics to Embodied Cognition . . . 8
1.2.2 Foundations Beyond Symbolic and Statistical Learning . . . 10
1.2.3 Recent Progress in Virtual Embodied AI . . . 11
1.3 Why Minecraft . . . 12
1.4 What We Do in Minecraft AI . . . 13
1.4.1 Open Source Projects for Agents in Minecraft . . . 13
1.4.2 The Minecraft AI Community: Research, Collaboration, and Outreach . . . 13
1.5 Primary Projects: Minecraft AI and Minecraft AI-Python . . . 14
1.6 The Whitepaper Structure . . . 15

2 Open Source Project: Minecraft AI **16**
2.1 Overview . . . 16
2.2 AI Character Architecture . . . 17
2.2.1 Lifecycle Management . . . 18
2.2.2 Behavior Management . . . 18
2.2.3 Memory Management . . . 20
2.3 Action Triggers . . . 21
2.4 Enhanced Interactions . . . 22
2.5 Extension as Plugin Systems . . . 23
2.5.1 Typical Plugin Usage Patterns . . . 23
2.5.2 Unified Capabilities Across Plugins . . . 24
2.6 Integrated Cognition . . . 25
2.6.1 System Architecture . . . 25
2.6.2 Key Capabilities of Integrated Cognition . . . 26
2.7 Discussion . . . 27

3 The MARC Architecture **28**
3.1 Ideas Behind MARC . . . 28
3.2 Memory: Enabling Behavioral Coherence and Consistency . . . 29
3.2.1 Decision Coherence Through Contextual Awareness . . . 29
3.2.2 Behavioral Consistency Through Social Context . . . 29
3.2.3 Memory Organization and Representation . . . 30
3.2.4 Memory Implementation in Minecraft AI . . . 31
3.3 Action: Adaptive Execution in a Dynamic World . . . 31

3.3.1 Context-Aware Action Selection . . . 31
3.3.2 Abstract and Programmable Actions . . . 32
3.3.3 Action Selection via MCP . . . 32
3.4 Reflection: Sustaining Autonomy through Self-Initiated Behavior . . . 33
3.4.1 Action Outcome Evaluation and Retry Logic . . . 33
3.4.2 Long-Term Task Reinvocation via Contextual Reflection . . . 34
3.5 Coding: Introduce the Infinite Possibilities . . . 34
3.5.1 Coding Capability and Output Variability . . . 34
3.5.2 Isolation and Context Maintenance . . . 35
3.6 Summary . . . 36

4 Open Source Project: Minecraft AI-Python 37
4.1 Overview . . . 37
4.2 Implementation of MARC Architecture . . . 38
4.2.1 Memory and Action Modules: Inheritance and Optimization . . . 38
4.2.2 Reflection Module: Core for Multi-Step Reasoning . . . 39
4.2.3 Coding Component: On-Demand Tool Creation . . . 40
4.3 Messages as The Only Action Triggers . . . 40
4.4 Reflection as The Core Cognition . . . 41
4.5 Advantages and Limitations . . . 43

5 The Embodied Turing Test 44
5.1 Background: From Dialogue to Embodiment . . . 44
5.2 Criteria for Evaluation . . . 45
5.2.1 Benchmark Task Types . . . 46
5.2.2 Example Scenarios . . . 47
5.2.3 Toward Standardized Evaluation Frameworks . . . 47
5.3 Challenges for Virtual AI Characters . . . 48
5.3.1 Coherence Within a Single AIC . . . 49
5.3.2 Collaboration Across Multiple AICs . . . 49
5.3.3 Maintaining Accurate and Adaptive Memory . . . 50
5.4 Discussions . . . 51

6 Coherence within Single AI Character 53
6.1 Background: What is Coherence and Why It Matters . . . 53
6.2 Persistent Memory Architectures for Coherent Agents . . . 55
6.2.1 Neural–Symbolic Precursors . . . 55
6.2.2 Hierarchical Stores and Reflective Summarization . . . 56
6.2.3 Retrieval-Augmented Generation as a Coherence Layer . . . 56
6.3 Persona and Goal Stabilization . . . 57
6.3.1 Identity Kernels: Freezing the Self . . . 57
6.3.2 Self-Reflection and Metacognitive Loops . . . 57
6.3.3 Plan Stacks: Durable Intentions . . . 58
6.3.4 Implementations in Original Character Games . . . 59
6.4 Cross-Modal Grounding . . . 59
6.4.1 Unified Multimodal Embeddings and Models . . . 59
6.4.2 Industry Toolkits . . . 60
6.5 Long-Horizon Evaluation for Coherence . . . 60
6.5.1 Academic Benchmarks and Methodologies . . . 60
6.5.2 Deployment Challenges . . . 61
6.6 Discussions . . . 61

7 Collaboration of Multiple AI Characters **63**
7.1 Background: The Motivations behind Collaborations . . . 63
7.2 Centralized Collaboration Methods . . . 64
7.2.1 Task Decomposition and Distribution . . . 66
7.2.2 Communication and Dynamic Role Allocation . . . 66
7.2.3 Strenghts and Limitations . . . 67
7.3 Decentralized Coordination Methods . . . 67
7.3.1 Peer-to-Peer Negotiation and Role Emergence . . . 68
7.3.2 Emergent Societies and Norm Formation . . . 68
7.3.3 Coordination via Shared Memory and Local Perception . . . 69
7.3.4 Strengths and Limitations . . . 70
7.4 Hybrid Coordination Methods . . . 70
7.5 Discussions . . . 71

8 Memory Mechanisms for AI Characters **73**
8.1 Background: The Central Role of Memory . . . 73
8.1.1 Biological Inspiration for Artificial Memory . . . 73
8.1.2 Practical Expectations for Stable Memory Mechanisms . . . 74
8.1.3 Theoretical Definition and Constraints . . . 75
8.2 Memory in LLM-Based Agents . . . 76
8.2.1 Contextual Window Memory . . . 76
8.2.2 Retrieval-Augmented Memory . . . 77
8.2.3 Structured Long-Term Memory . . . 78
8.2.4 Reflective and Self-Modeling Memory . . . 79
8.2.5 Summary and Outlook . . . 80
8.3 Memory in Embodied Game Agents . . . 81
8.3.1 Implemented Architectures and Patterns . . . 81
8.3.2 Challenges and Trade-offs . . . 82
8.4 Toolkits and Productions . . . 83
8.5 Evaluations and Benchmarks . . . 84
8.6 Discussion . . . 85

9 LLM-Based Embodied AI **87**
9.1 LLMs as the Backend of Embodied Agents . . . 87
9.2 Memory: Generating Contextual and Long-Term Narratives with LLMs . . . 88
9.3 Action Selection: Contextual Decision-Making with LLMs . . . 90
9.4 Reflection: Task Reassessment and Long-Term Intent Continuation . . . 90
9.5 Coding: Extending Behavior Through LLM-Generated Functions . . . 93
9.6 Leveraging LLMs for Structured and Creative Task . . . 94
9.6.1 LLM-Assisted Planning for Structured, Multi-Step Tasks . . . 94
9.6.2 Use of Contextual References . . . 95
9.7 Discussions . . . 95

10 The Future with Multimodal LLMs **97**
10.1 Visual Perception for Human-Like Understanding . . . 97
10.1.1 Incorporating Visual Input into VLM-Powered Perception . . . 98
10.1.2 Challenges in Integrating VLMs into AIC Logic . . . 98
10.2 Video and Audio Perception . . . 99
10.3 Understanding Structural Data in Virtual Worlds . . . 100
10.3.1 Limitations of General-Purpose Perception . . . 100
10.3.2 Toward Structure-Aware Multimodal LLMs: Needs and Challenges . . . 100

10.4 Discussions 102

11 The Roadmap of Minecraft AI **103**
11.1 Maintaining and Expanding the Core Platforms 103
11.2 Establishing Benchmarks and Shared Databases 104
11.3 Fostering a Vibrant and Inclusive Community 105
11.4 Summary 106

Bibliography **106**

Chapter 1

Introduction: Embodied AI and Minecraft

1.1 Overview

As a subfield of artificial intelligence, Embodied Artificial Intelligence (Embodied AI) focuses on creating agents capable of physical or simulated interaction with the environment [Paolo et al., 2024]. Unlike traditional AI systems that passively process curated data, embodied AI systems operate in real or virtual environments, sensing and responding to changes as they occur. This approach draws inspiration from how humans and animals develop intelligence—through embodied experience and constant feedback loops [Kirchhoff et al., 2018]. At its core, an Embodied AI system consists of three major components:

- **Perception:** The agent continuously gathers information from the environment using sensors such as cameras, microphones, tactile sensors, or simulated equivalents in virtual worlds. This data provides context for decision-making.
- **Cognition:** Based on perceived data, the agent performs reasoning, planning, and decision-making. Modern systems often utilize large language models (LLMs) and visual-language models (VLMs) to enhance situational understanding, multimodal reasoning, and long-term goal alignment.
- **Action:** The agent executes actions using actuators (e.g., motors, robotic arms, or digital avatars in simulated environments), modifying the environment and triggering new perceptual feedback.

These core components operate within a continuous feedback loop: perception informs cognition, cognition guides action, and actions in turn alter the environment, generating new sensory input. The strength of Embodied AI lies in its seamless integration of perception and action within a goal-oriented framework. This tight coupling enables the development of intelligence that is more adaptive, generalizable, and aligned with human reasoning. By engaging with the world through a physical or simulated body, embodied agents acquire a grounded understanding of abstract concepts such as spatial relationships, causal dynamics, and environmental affordances.

A defining characteristic of modern Embodied AI is its departure from static, task-bounded execution models. Instead of performing isolated tasks with clearly defined start and end points, Embodied AI systems are typically designed to handle multiple tasks simultaneously, each with variable duration and shifting priorities. For example, a mobile robot may need to navigate toward a designated location while concurrently scanning the surrounding environment for a specific object, requiring it to balance locomotion, object recognition, and spatial awareness in real time.

In such scenarios, perception, cognition, and action modules frequently operate asynchronously across distributed computing environments [Brooks, 1991]. This modular, concurrent architecture supports more

flexible and scalable agent behavior, but also introduces significant engineering challenges. Chief among these is the need to maintain cognitive coherence—ensuring that the agent's beliefs, goals, and perceptions remain aligned—and to synchronize internal states across subsystems in the presence of constant environmental change and parallel processing.

This growing complexity has fueled interest in virtual environments as practical and powerful alternatives for Embodied AI research. Although virtual environments offer a simplified approximation of physical reality, they abstract away low-level noise and hardware constraints while preserving essential components of embodied interaction—namely, perception, agency, and temporal decision-making. This makes them ideal for rapid prototyping and scalable evaluation of general intelligence capabilities.

For example, platforms such as Minecraft provide rich and open-ended 3D worlds where agents can concurrently pursue navigation, construction, resource management, and social interaction. These environments allow researchers to implement and study mechanisms of memory, goal planning, and multi-agent collaboration in settings that are both controlled and expressive. The virtual domain's speed, safety, reproducibility, and support for structured data collection make it a compelling frontier for advancing Embodied AI toward more robust and generalizable models of intelligent behavior.

More specifically, the implementation of Embodied AI systems (or agents) in virtual game environments typically includes structured support for the three primary components:

- **Perception:** In virtual environments, acquiring relevant environmental information is generally straightforward and efficient. This is because the underlying game engine maintains a complete and consistent system state, allowing agents to access accurate values for any observable parameter—unless certain information is intentionally hidden by the system designer. However, this convenience does not eliminate the need for a perception module. Instead, it allows researchers to simulate different levels of sensory fidelity and noise. The key challenge remains: extracting salient features and translating raw environmental data into a meaningful internal representation for the cognition module, all within limited computational time.

- **Cognition:** In virtual settings, designing and testing the cognition module often becomes the primary focus of research and development. These environments can offer large, persistent worlds and extended temporal scales, enabling agents to simultaneously pursue long-term objectives while responding to short-term demands. This facilitates the study of complex cognitive processes such as hierarchical planning, adaptive behavior, skill acquisition, and even the emergence of novel strategies or abilities—beyond those explicitly programmed.

- **Action:** On one hand, action interfaces in virtual environments are typically simple, well-defined, and produce consistent outcomes. On the other hand, modern AI models often have access to low-level scripting or APIs that expose much of the game engine's internal logic. This opens up the possibility for agents to execute a wide range of actions—so long as they remain consistent with the fundamental rules of the simulated world. This high degree of flexibility is arguably one of the most significant advantages of virtual environments as testbeds for Embodied AI, compared to the constraints and unpredictability of physical systems.

Virtual environments are increasingly being treated not only as testbeds or the training simulators, but as intrinsic domains of embodied cognition. Current applications of Embodied AI in virtual environments include:

- **Lifelong Learning Agents**: Agents continuously explore and adapt to new tasks and environments. Lifelong learning in virtual settings supports curriculum learning, skill reuse, and hierarchical task decomposition.

- **Interactive NPCs and Game Characters**: Embodied AI is used to create non-playable characters (NPCs) that exhibit realistic dialogue, memory, and behaviors consistent with long-term narrative arcs.

This enables more immersive and dynamic player experiences in both single-player and multiplayer game contexts.

- **Science Simulation and Social Psychology**: Virtual environments serve as platforms to simulate human-like reasoning and interaction patterns, including emotion, motivation, and collective decision-making [Park et al., 2023].

Looking ahead, as we gradually approach the Metaverse—often envisioned as the most powerful and flexible form of virtual environments—simulation fidelity continues to increase, and multimodal interfaces become more sophisticated. As a result, the boundary between physical and virtual embodiment is expected to become increasingly fluid. The virtual world not only serves as a safe and scalable sandbox for pretraining and strategy development, but also as a standalone environment for knowledge-intensive and socially interactive tasks.

In the near future, it is likely that individuals will spend a significant portion of their time in virtual environments, acquiring information, interacting with others, and seamlessly transferring knowledge between virtual and physical domains. Moreover, elements within the virtual world are being designed to interact with physical systems, enabling bidirectional influence. In this context, the capabilities of Embodied AI agents will no longer be limited to virtual interactions alone, but will increasingly extend to meaningful engagement with the real world.

When that day comes, Embodied AI may no longer function merely as intelligent tools, but instead take on roles more akin to player-controlled characters—serving as autonomous actors within complex systems and contributing directly as a new form of artificial productivity.

1.2 History of Embodied AI

1.2.1 From Behavior-Based Robotics to Embodied Cognition

The roots of Embodied AI extend far earlier than the formalization of the concept in the 1990s. Eearly researchers were already exploring how intelligence might emerge from agents physically embedded in the world. These pioneering efforts laid the groundwork for what would eventually become the field of embodied artificial intelligence.

One of the earliest and most influential figures in this area was William Grey Walter [Wikipedia contributors, 2024b, Bladin, 2006], a British neuroscientist who, in the 1950s, created a series of autonomous robots known as the *cybernetic tortoises*. These simple machines used analog circuitry to exhibit basic reactive behaviors such as phototaxis—moving toward light sources. Despite their simplicity, the tortoises demonstrated that relatively minimal sensory-motor coupling could produce lifelike, adaptive behavior [Wikipedia contributors, 2024a]. Walter's work was among the first to propose that intelligence could emerge from the direct interaction between body, environment, and sensory input.

In the 1960s, the Stanford Research Institute (SRI) developed one of the most iconic robots of the early AI era: *Shakey the Robot*. Shakey integrated sensors (such as cameras and bump detectors), a suite of planning algorithms, and actuators to navigate structured environments like hallways. Unlike Walter's tortoises, Shakey combined symbolic reasoning with real-world perception, becoming the *first robot capable of reasoning about its actions* in a world it could perceive [Wikipedia contributors, 2025d]. Though reliant on a deliberative architecture, Shakey marked a shift in AI research from pure computation to physical instantiations of intelligence.

These early systems—ranging from reactive cybernetic machines to symbolically guided planners—were foundational in shaping the trajectory of AI. While they predated the term "Embodied AI," they addressed its core concerns: how physical presence, perception, and situated action contribute to intelligent behavior.

By the late 1980s, however, frustration with the brittleness of symbolic robotics led to a paradigm shift. Robots like Shakey were intelligent in principle but inflexible in practice, struggling in unstructured environments. This dissatisfaction culminated in Rodney Brooks's behavior-based robotics paradigm, which

argued that intelligence emerges from dynamic sensorimotor interaction with the world rather than from abstract representations [Brooks, 1991].

Rodney Brooks's seminal paper *Intelligence Without Representation* [Brooks, 1991], which was published in 1991, is a landmark in Embodied AI research. In stark contrast to traditional AI systems reliant on complex symbolic representations and central planning, Brooks proposed that intelligent behavior could arise directly from real-time interactions between the agent and its environment. His subsumption architecture emphasized decentralized control, layered behaviors, and real-world grounding through sensors and actuators, laying the foundation for behavior-based robotics.

Building on this departure from symbolic AI, Rolf Pfeifer and Christian Scheier further formalized the principles of Embodied AI in their book *Understanding Intelligence* [Pfeifer and Scheier, 2001]. They proposed that intelligence does not reside solely in the brain or in algorithmic processes, but emerges as a result of the full interaction between the physical body, the nervous system, and the environment. This perspective, often referred to as "embodied cognition", underscores the importance of an agent's morphology (body shape and capabilities) in shaping its cognitive processes. They argued that the agent's sensory-motor coordination, bodily constraints, and affordances are not limitations to be overcome, but rather enablers of intelligent behavior.

In 2005, Linda Smith contributed to this trajectory from the domain of developmental cognitive science by proposing the "embodiment hypothesis" [Smith, 2005], which postulates that cognition arises from the dynamic coupling between brain, body, and world. This framework implies that perception, learning, memory, and even abstract reasoning are deeply grounded in sensorimotor experience. Her research in child development demonstrated that early physical interactions, such as reaching and grasping, play a foundational role in the development of spatial awareness, categorization, and causal reasoning.

These early works collectively established several enduring principles that continue to shape the development of Embodied AI:

- **No reliance on predefined logic:** Rather than relying on rigid, symbolic reasoning systems or handcrafted rules, early embodied systems demonstrated that intelligent behavior can emerge from real-time sensory-motor interactions with the environment. This principle shifted focus from deductive logic to situated responsiveness, laying the groundwork for reactive and behavior-based control architectures.

- **Learning is evolutionary and adaptive:** Intelligence in embodied systems is not seen as something that is wholly engineered upfront, but rather something that evolves and adapts over time through interaction. Feedback from the physical world plays a crucial role in shaping behavior, enabling agents to refine their strategies and develop new capabilities through experience. This inspired the adoption of learning paradigms such as reinforcement learning and evolutionary algorithms in robotics and AI.

- **The environment is not a backdrop but a partner:** In contrast to earlier views of the world as a passive context in which cognition happens, Embodied AI regards the environment as an active component in the cognitive loop. The structure and affordances of the environment influence the development of perception, action, and decision-making. Early experiments with simple robots highlighted how even minimal embodiment in a structured world could give rise to seemingly complex behavior, underscoring the co-dependence between agent and world.

- **Intelligence is grounded in physical interaction:** Cognition is not divorced from the body—it emerges from sensorimotor contingencies and physical experiences. Early embodied agents, even with limited sensing and actuation, revealed that meaningful behavior is grounded in how an agent moves, senses, and is constrained by its physical form. This concept continues to influence the design of modern AI agents that aim to learn through embodied simulation or physical embodiment.

- **Modularity and decentralization are effective strategies:** Inspired by biological systems, early embodied architectures often employed decentralized control mechanisms instead of a single centralized

planner. This approach proved to be more robust, scalable, and adaptable in dynamic or uncertain environments. It also enabled parallel development of behaviors, each responsible for specific functions, contributing to the overall adaptability of the agent.

1.2.2 Foundations Beyond Symbolic and Statistical Learning

The early 2000s saw renewed interest in the theory of embodied cognition, which posits that intelligence is not merely a computational process over abstract symbols but emerges through active engagement with the world via a body. This framework inspired the design of AI systems that leverage sensory-motor coupling, situated reasoning, and reactive control dynamics [Pfeifer and Scheier, 2001].

Robotics was still the primary platform for Embodied AI experimentation during this period. Numerous efforts focused on locomotion, manipulation, and adaptive behavior through reinforcement and developmental learning mechanisms [Lungarella et al., 2003]. Inspired by natural organisms, embodied robots demonstrated how physical embodiment could simplify control, offload computation to morphology, and enhance generalization.

During the early years of 2000s, there is also a rise of *evolutionary robotics* and *developmental approaches* [Gomi, 2001, Zlatev and Balkenius, 2001], which applied biologically inspired algorithms to optimize robot behavior and body design. Evolutionary methods searched through large design spaces to discover robust controllers, while developmental robotics modeled learning processes akin to those of infants. Both lines of research emphasized self-organization, robustness, and adaptation over prescriptive programming.

Another prominent theme in this period was *morphological computation* [Pfeifer et al., 2006, Paul, 2006], the idea that a robot's physical form and material properties could contribute to computation and control. By exploiting passive dynamics, compliance, and structural constraints, researchers demonstrated that the body itself could offload cognitive burden from central controllers. This concept, inspired by biological systems such as insect locomotion, emphasized how morphology can simplify control and enhance adaptability in unstructured environments.

Although still in its infancy, soft robotics emerged during this period as a complementary approach to classical robotics. These systems used flexible, deformable materials to better accommodate interaction with humans and unstructured environments. Biohybrid systems also explored integrating living tissue or biomimetic structures into robotic designs, reinforcing the embodied perspective that cognition arises from dynamic interactions between body and world [Feinberg et al., 2007, Trivedi et al., 2008].

Throughout the first decade of 2000s, the theoretical underpinnings of embodied cognition were further developed. Influences from ecological psychology, enactivism, and systems theory highlighted that perception, action, and cognition co-emerge from real-world engagement. As a result, Embodied AI research increasingly became a multidisciplinary field, blending robotics, neuroscience, artificial life, and philosophy. Without large datasets or neural networks, control architectures emphasized tight coupling between sensing and action. Embodied agents were equipped with low-level controllers designed to respond in real time to their surroundings. Obstacle avoidance, object tracking, and adaptive locomotion were achieved through direct sensorimotor mappings rather than internal representations or planning. These systems often relied on layered or subsumption architectures, continuing the trajectory set by Brooks, but with increasing physical realism and complexity Communications [2023].

The rise of deep learning, particularly deep reinforcement learning (deep RL), has marked a turning point in the evolution of Embodied AI [Hartono and Kakita, 2009, Hester et al., 2010]. Unlike earlier architectures that relied on handcrafted controllers and symbolic reasoning, deep RL enables agents to learn complex behaviors directly from raw sensory data through trial-and-error interaction with the environment [Gupta et al., 2021]. This end-to-end learning paradigm has significantly expanded the range and sophistication of tasks that embodied agents can perform, from fine-grained manipulation to long-horizon planning [Lillicrap et al., 2015].

Simulated photorealistic environments now serve as scalable training grounds, where agents can undergo millions of safe interactions before real-world deployment. Several high-fidelity simulators emerged to accelerate Embodied AI research by enabling scalable, safe, and efficient training of agents in virtual

settings. While there are comprehensive surveys like Liu et al. [2024b] which have outlined the critical role of simulation in Embodied AI research, the representative works included AI2-THOR [Kolve et al., 2017], Gibson [Xia et al., 2018], and Habitat [Savva et al., 2019].

These platforms allowed researchers to gathering training data for the deep learning-based study, and they provide more flexible environments for studying long-horizon planning, multimodal perception, and embodied instruction following, laying the groundwork for today's simulation-to-reality transfer pipelines and large-scale embodied learning.

1.2.3 Recent Progress in Virtual Embodied AI

As simulators, open platforms, and interactive games become increasingly integral to Embodied AI research, recent years have witnessed a major shift toward studying agents within high-fidelity virtual environments [Duan et al., 2022, Zhang et al., 2024]. This evolution is driven by the scalability, safety, and flexibility that simulation offers for developing and evaluating complex, adaptive behaviors. By supporting multimodal perception and diverse task domains—including vision, navigation, manipulation, and language—virtual agents are now capable of performing sophisticated behaviors such as household assistance, tool use, and multi-step reasoning. Unlike symbolic AI systems, these agents develop cognitive and motor skills through direct sensorimotor interaction with their environments, learning from raw multimodal inputs to build spatial awareness, memory, and control policies. Standardized research frameworks such as AI2-THOR, Gibson, and Habitat have become widely adopted for benchmarking tasks like point-goal navigation, semantic exploration, and embodied question answering.

The emergence of embodied agents powered by large language models (LLMs) has been shaped by several pioneering efforts across academia and industry. One of the most influential contributions is the *Generative Agents* framework introduced by [Park et al., 2023], which simulates believable human behavior within a sandbox world. These agents utilize memory, reflection, and planning modules to dynamically engage in lifelike social interactions, forming a milestone in the simulation of autonomous characters without predefined behavior trees.

Concurrently, *Voyager* by [Wang et al., 2023a] reimagines the concept of lifelong learning in Minecraft by allowing agents to generate and iteratively refine code using GPT-4 [OpenAI, 2023]. Voyager's automatic curriculum and skill library enable self-improving behaviors, facilitating generalization across environments and surpassing earlier hard-coded or reward-driven agents in both exploration and skill diversity.

Expanding the scope of agent populations, *Project Sid* [AL et al., 2024] introduces the PIANO architecture to support large-scale, real-time coordination of 1,000 agents in Minecraft. These agents spontaneously developed economic systems, governance structures, and even religious beliefs—demonstrating the capacity for emergent social complexity in simulated worlds. This large-scale multi-agent simulation suggests new pathways for studying social behaviors, coordination, and digital societies using LLM-based agents. Moreover, Wu et al. investigate generative multi-agent collaboration in Embodied AI [Wu et al., 2025]. Their findings highlight how LLMs improve inter-agent communication, shared planning, and coordination, particularly in dynamic, partially observable environments. This aligns with the goals of creating believable multi-agent ecosystems within virtual worlds.

Beyond Generative Agents and Voyager, the MINDcraft [1] framework represents a significant contribution to the field [White et al., 2025]. Introduced by White et al., MINDcraft is an extensible multi-agent platform built atop the Mineflayer API [PrismarineJS Contributors, 2017] for Minecraft, enabling large language models (LLMs) to control agents through high-level commands in complex, partially observable worlds. It facilitates advanced embodied reasoning and coordination tasks such as cooking, crafting, and construction, encapsulated in the benchmark MineCollab. These tasks are procedurally generated to test agents' abilities in task delegation, resource sharing, and goal completion. A key finding of the work is that communication efficiency remains a major bottleneck in multi-agent embodied settings, with task performance dropping notably when agents must explain or negotiate complex plans.

[1] https://github.com/kolbytn/mindcraft

The authors also provide a supervised fine-tuning dataset from over 2,000 trials and show that lightweight models can achieve near-parity with larger models on collaborative tasks when trained on this curated data. MINDcraft thereby sets a new standard for Embodied AI research in open-ended, language-controllable environments.

1.3 Why Minecraft

Minecraft [Wikipedia contributors, 2025b], developed by Mojang Studios and initially released in 2009, is a sandbox video game that allows players to explore, create, and interact with a procedurally generated 3D world composed of blocks. It has become one of the most popular games of all time, with over 300 million copies sold globally. Designed with an emphasis on creativity and open-ended play, Minecraft supports both survival-based challenges and free-form construction, offering a virtual environment where players—and increasingly AI systems—can learn through interaction.

Beyond entertainment, Minecraft has found widespread applications in education [Ekaputra et al., 2013], simulation, and research. Educational platforms such as Minecraft: Education Edition [Kuhn, 2018] are used to teach subjects ranging from chemistry and programming to history and architecture. Its programmable and modifiable architecture has also made it a widely adopted environment in AI and robotics research, where it serves as a sandbox for experimentation with intelligent agents, human-AI interaction, and collaborative problem-solving.

In recent years, Minecraft has emerged as one of the most promising virtual environments for advancing Embodied AI. Its unique blend of structured physicality, open-ended gameplay, and accessible modding interfaces makes it an ideal platform for studying perception, reasoning, and behavior in AI systems—especially those driven by large language models (LLMs).

First and foremost, Minecraft offers a richly interactive and persistent 3D environment. Agents must navigate and manipulate their surroundings in real time, encountering dynamic elements such as terrain, weather, and mobs. This interaction mirrors the perception-action loops found in physical embodied systems, allowing AI models to learn from environmental feedback rather than static datasets.

Second, the game's block-based representation provides an elegant balance between *expressiveness and simplicity*. Every object and interaction is encoded in a discrete, spatially consistent format, enabling precise control and observation while still allowing for creative and complex emergent phenomena. This granularity supports both fine-grained skill learning (e.g., tool use, crafting) and high-level planning (e.g., exploration, architecture, multi-step missions).

Third, Minecraft is inherently a domain of *cognitive challenge*. Even its basic survival gameplay involves navigation, spatial memory, goal prioritization, and resource management—traits that are critical to embodied reasoning. Tasks such as crafting, farming, or building shelters require long-horizon planning and adaptation to uncertainty, making it a natural testbed for LLMs with advanced planning capabilities.

Moreover, Minecraft's modding ecosystem and APIs (e.g., Mineflayer [PrismarineJS Contributors, 2017]) offer fine-grained control over the environment, agents, and interaction loops. Tools like MineDojo initiative [Fan et al., 2022], Voyager [Wang et al., 2023a] and MINDcraft [White et al., 2025] have already demonstrated the potential of combining LLMs with Minecraft for open-ended task solving and skill acquisition.

Finally, the platform is highly conducive to multi-agent scenarios. From cooperative building to social simulations, Minecraft enables the study of communication, coordination, and emergent behavior among AI agents—further aligning it with the goals of embodied multi-agent intelligence research.

In summary, Minecraft offers an ideal synthesis of physicality, abstraction, and interactivity. It enables AI researchers to simulate real-world cognition and behavior in a controlled, flexible, and richly instrumented virtual world. For embodied agents driven by large language models, it represents a powerful proving ground where perception, memory, planning, and action converge in a lifelike digital setting.

1.4 What We Do in Minecraft AI

Minecraft AI is an open-ended research and development initiative dedicated to exploring how large language models (LLMs), symbolic reasoning, and embodied learning mechanisms can be combined to create intelligent AI characters (AICs) within interactive 3D environments. At the core of this initiative is the belief that AI agents in virtual worlds like Minecraft should not only solve tasks—they should evolve into believable, collaborative, and socially integrated companions that can play alongside humans. The Minecraft AI effort is twofold: an open-source project ecosystem for embodied agent development, and a vibrant community hub for advancing in gaming.

1.4.1 Open Source Projects for Agents in Minecraft

The primary technical goal of the Minecraft AI project is to develop a unified agent framework capable of supporting diverse, lifelike AI characters in Minecraft, which are envisioned as autonomous companions, collaborators, and explorers.

At the architectural level, Minecraft AI integrates LLMs, symbolic planners, memory systems, and environmental interfaces to simulate embodied intelligence. The framework supports both JavaScript and Python implementations, leveraging APIs such as Mineflayer [PrismarineJS Contributors, 2017] for real-time agent control.

A key aspiration of the project is to track progress toward an *Embodied Turing Test*—a long-term benchmark in which AI characters in Minecraft are indistinguishable from human players based on their behavior, interactions, and problem-solving capabilities. This goal shapes the design of agents that not only complete tasks efficiently but also demonstrate personality, social awareness, and long-term adaptation.

To support experimentation and reproducibility, Minecraft AI is planning to provide modular libraries and development tools that allow researchers and developers to build, customize, and evaluate AI characters with varied cognitive and behavioral configurations.

1.4.2 The Minecraft AI Community: Research, Collaboration, and Outreach

In addition to the codebase, Minecraft AI fosters a growing open community focused on research, education, and public engagement around intelligent agents in gaming environments. This community brings together AI researchers, educators, indie developers, and hobbyists who are passionate about the future of AI in interactive worlds.

- **Research Collaboration:** Members contribute to cutting-edge topics such as agent memory design, planning in partially observable environments, and generative behavior modeling.
- **Workshops and Forums:** The Minecraft AI community organizes online and offline meetups, hackathons, and workshops focused on Embodied AI in games. These events provide venues to share findings, demonstrate prototypes, and host competitions (e.g., cooperative build challenges or survival trials for AI agents).
- **Education and Outreach:** With the Python-based agent implementation, Minecraft AI has become a valuable educational platform for high school and university students learning about AI, human-computer interaction, and agent-based simulation.

Through its commitment to open collaboration, the Minecraft AI community aims to democratize the development of AI characters that can one day serve as teammates, mentors, and friends—not just in Minecraft, but in the broader landscape of embodied interactive systems.

1.5 Primary Projects: Minecraft AI and Minecraft AI-Python

To support diverse use cases in research, prototyping, and education, the Minecraft AI initiative offers two parallel open-source implementations. While both adhere to the same cognitive architecture and design principles, they are optimized for different technical ecosystems and application needs.

Minecraft AI (JavaScript version). [2] This implementation is extended from the MINDcraft framework [White et al., 2025] and is built on Node.js, utilizing the Mineflayer API [PrismarineJS Contributors, 2017]. It provides fast deployment, real-time agent control, and flexible plugin support—making it especially suitable for live demos, interaction-heavy showcases, and rapid experimentation. Key features include:

- **Action-call prompting framework:** Integrates tightly with the Mineflayer bot interface through a prompt-based action-calling scheme inspired by Anthropic's Model Context Protocol (MCP) [Anthropic, 2023]. This mechanism facilitates clear and extensible action decision workflows. It inherits this structure from MINDcraft with improved prompt templates for easier customization.
- **Modular plugin system:** Minecraft AI introduces a flexible plugin architecture allowing dynamic modification of agent behaviors. While some compatibility with MINDcraft plugins remains, several core components in Minecraft AI have been modularized or restructured for greater extensibility, requiring minor adaptation of plugins when migrating between platforms.
- **Integrated Cognition:** A novel feature that offloads computationally intensive tasks—such as memory management and reflection—to independently hosted services, thereby reducing the architecture complexity and enabling broader accessibility.

Minecraft AI-Python. [3] This Python-based implementation reimagines the same agent logic in a more research-focused environment. It prioritizes system robustness, fine-grained control, and compatibility with the Python scientific and AI tooling ecosystem. It is particularly well-suited for educational use, long-term behavioral analysis, and the development of memory-rich agents. Key features include:

- **Native Python architecture:** Planning, action selection, and memory systems are all implemented in Python, simplifying code extension, experimentation, and integration with other Python-based AI tools and models.
- **Message-loop processing design:** Employs a single-threaded message loop architecture to ensure predictable and stable task execution. While this design sacrifices concurrency, it enhances reliability and reproducibility—both critical for controlled research and teaching contexts.
- **Full support for the MARC architecture:** Implements the full **Memory–Action–Reflection–Coding (MARC)** framework, designed to cleanly separate agent control logic from language model capabilities. This facilitates more modular, inspectable, and extensible embodied agent systems.

Shared Vision and Synergy. Despite their technical distinctions, both implementations are united by a common objective: building Embodied AI agents that can perceive, reason, and act with autonomy and coherence in complex virtual environments. They serve as parallel instantiations of a shared cognitive blueprint, and both contribute to a larger research agenda—namely, the pursuit of the **Embodied Turing Test**, where virtual agents become indistinguishable from human players in both behavior and social presence.

Together, Minecraft AI and Minecraft AI-Python form a comprehensive ecosystem for Embodied AI research and deployment in Minecraft. Developers and researchers can choose the toolchain that best

[2] https://github.com/aeromechanic000/minecraft-ai
[3] https://github.com/aeromechanic000/minecraft-ai-python

aligns with their specific goals—whether that be real-time interaction, educational engagement, or deep investigation into AI character cognition—without departing from the foundational principles of intelligent, persistent, and adaptive agents.

1.6 The Whitepaper Structure

This whitepaper presents a comprehensive overview of the architecture, capabilities, and future directions of the Minecraft AI initiative. The content is organized into three major parts:

- **Part 1: From MINDcraft to MARC.** This section introduces the two core open-source projects—Minecraft AI (JavaScript-based, Chapter 2) and Minecraft AI-Python (Chapter 4)—and elaborates on the underlying **MARC** architecture (Chapter 3), which defines the cognitive control structure for AI characters in virtual environments.
- **Part 2: Toward the Embodied Turing Test.** We provide the conceptual background of the Embodied Turing Test and propose application scenarios and evaluation benchmarks for it (Chapter 5). This part also explores key research challenges inspired by the test, including: maintaining internal coherence of an AI character over time (Chapter 6); enabling collaborative behavior among multiple agents for shared long-term goals (Chapter 7); and designing memory mechanisms to support human-level continuity and planning (Chapter 8).
- **Part 3: From Language to Action.** We investigate the role of large language models (LLMs) in driving human-like behavior in embodied agents and discuss current advancements in applying LLMs within interactive virtual environments (Chapter 9). In addition, we explore emerging trends, particularly the integration of multimodal models such as visual-language models (VLMs), which are poised to greatly enhance perception, reasoning, and natural interaction (Chapter 10).

The whitepaper concludes with a development roadmap (Chapter 11) and a final summary of the initiative vision. Our goal is to provide a foundational reference for researchers, developers, and educators interested in building next-generation embodied AI systems—starting with Minecraft, and extending to broader virtual and physical domains.

Chapter 2

Open Source Project: Minecraft AI

2.1 Overview

The Minecraft AI [1]project is built upon the foundational architecture of MINDcraft [White et al., 2025], preserving its core logic while enhancing modularity, flexibility, and scalability for broader research and real-time applications. MINDcraft introduced a novel agent framework that effectively combines large language models (LLMs) with the behavioral logic of AI characters (AICs) in Minecraft. Minecraft AI continues this line of development while modularizing its components and supporting extensible plugin-based design. The core logic inherited from MINDcraft includes:

- **LLM-Driven Action Selection via Chat Understanding.** The agent processes messages captured from the in-game chat and leverages an LLM to select the most appropriate action. Actions are defined following the Model Context Protocol (MCP) introduced by Anthropic [Anthropic, 2023], and are executed via structured function calls. Additionally, the framework supports a special command `!newAction`, which allows the agent to generate and execute new behaviors through real-time code generation—greatly enhancing behavioral extensibility.
- **Contextual Memory Integration.** Minecraft AI utilizes LLMs to summarize the interaction history, forming a lightweight but effective long-term memory system. This memory is dynamically composed with the agent's current status and short-term goals, providing the LLM with a rich and coherent decision-making context. This allows AICs to maintain memory and goal continuity over extended interactions.
- **Hierarchical Planning through Goal Abstraction.** For handling more complex tasks, the system supports a higher-level planning directive `!goal`, enabling agents to break down and pursue multi-step objectives. Combined with the flexible `!newAction` functionality, agents can engage in chain-of-thought (CoT) [Wei et al., 2022] planning and dynamically expand their action repertoire. This hybrid strategy significantly improves the adaptability and autonomy of agents in open-ended environments.

Building on the core agent design and reasoning logic introduced by MINDcraft, the Minecraft AI project introduces two major architectural innovations aimed at improving scalability, flexibility, and real-time responsiveness.

- **Plugin System for Modular Action Extension.** One major challenge faced by LLM-driven action call is the context limitation when encoding large action spaces into the prompt. Although MINDcraft leverages retrieval-augmented generation (RAG) [Lewis et al., 2020] to select a subset of candidate

[1] https://github.com/aeromechanic000/minecraft-ai

actions to include in context, this approach is sensitive to the quality of retrieval and the clarity of action descriptions. To address this, Minecraft AI introduces a plugin system that separates agent capabilities into core (built-in) and extended (plugin-based) modules.

This system enables users to explicitly enable or disable action modules, thereby reducing unnecessary context load and making it easier to manage the agent's behavioral repertoire. More importantly, it opens the door to hierarchical action composition—allowing RAG to operate at the level of *plugin sets* rather than individual actions. This facilitates greater scalability and flexibility in agent behavior design while maintaining prompt length within operational limits.

- **Integrated Cognition via LLM Agent APIs.** A second key improvement focuses on offloading expensive context management operations such as memory summarization and code sampling for dynamic code generation. In MINDcraft, these operations are managed locally by the AIC host process, which can result in high computational demand and occasional latency during action planning.

  Minecraft AI proposes a new abstraction called *Integrated Cognition*, where a set of specialized LLM agents—deployed locally or on the cloud—provide external APIs to handle memory summarization, long-term context management, and even code database for new actions. Rather than building full prompts locally, the agent interacts with a remote cognition service that encapsulates both historical memory and goal context into a unified prompt for LLM queries.

  This approach transforms complex multi-step procedures (e.g., context aggregation → LLM call) into single API requests, reducing local complexity while enabling centralized management of memory and reflective state. In the long term, this system is envisioned to evolve into a full backend agent orchestration layer, supporting shared memory, distributed planning, and fine-grained role delegation across AICs.

Together, these improvements aim to extend the capabilities of Minecraft AI agents while minimizing developer burden and runtime overhead—paving the way for scalable, persistent, and extensible embodied intelligence in virtual environments.

2.2 AI Character Architecture

In the Minecraft AI project, we inherit the core agent architecture from the original MINDcraft project, while further refining the underlying functionality and modularizing all non-essential features into separate plugins. The result is a cleaner, more extensible framework for AI Characters (AICs), designed to support rich in-game interaction and adaptive behavior.The architecture of each AIC consists of the following key components:

- **Lifecycle Management:** Governs the AIC's existence in the world, including how it joins the game, responds to events, processes messages, and exits. This ensures the AIC behaves consistently across different stages of interaction.
- **Behavior Management:** Responsible for interpreting incoming messages and selecting the appropriate action using an LLM-based reasoning system. This layer includes action registration, decision-making, and execution monitoring.
- **Memory Management:** Separates short-term and long-term information through an extensible memory system. Recent events inform immediate decisions, while summarized memories support continuity, learning, and role consistency over time.

2.2.1 Lifecycle Management

Each AIC operates within a structured lifecycle, built on the event-driven architecture of Mineflayer [PrismarineJS Contributors, 2017]. This lifecycle defines how the AIC enters, acts within, and exits the game world. The primary events include:

- **Join Game:** Upon entering the Minecraft environment, the AIC is initialized and brought online. This marks the beginning of its active state, where it becomes capable of responding to player input or initiating autonomous behaviors. During this phase, the AIC typically receives an `init_message`—a predefined trigger to verify readiness and initiate any startup routines. Otherwise, the AIC will just say "hello world".
- **Process Message:** This is the core operational phase in which the AIC interprets and reacts to incoming messages—whether from players, the environment, or internal processes. The AIC queries a large language model (LLM) to determine the most appropriate action based on a combination of message content, current status, agent profile, memory context, and available action documentation.
- **Leave Game:** When the AIC disconnects—either manually or due to failure—it triggers the cleanup process. A process monitor observes the disconnection behavior to determine whether the exit was expected. If the AIC exited unexpectedly and recovery is possible, the system can automatically relaunch the instance and resume from the last known state, ensuring continuity and resilience.

Among these, *Process Message* serves as the operational heart of the AI Character. Every behavior—whether building, crafting, navigating, or engaging in combat—ultimately traces back to one or more message-processing steps. This message-driven paradigm ensures modularity, consistency, and adaptability, allowing new capabilities to be integrated seamlessly as plugins or specialized behaviors. It is this structure that enables AICs to maintain coherent, human-like behavior across diverse tasks and long-term interactions.

2.2.2 Behavior Management

In Minecraft AI, behavior management is responsible for selecting and executing the most appropriate action in response to incoming messages. At the core of this process lies a simple yet powerful question: *Given a message, which action should the AI Character take?* This decision-making process defines the intelligence and autonomy of each AIC.

Our system builds upon the action selection framework established in MINDcraft, which adopts the Model Context Control Protocol (MCP) proposed by Anthropic [Anthropic, 2023]. Under this protocol, developers can define new actions by following a standardized schema. Once properly registered, these actions become eligible for selection by the LLM-powered planner, without requiring any additional integration. A standard MCP-compliant action is defined with the following elements:

- **Action Name:** A unique identifier that captures the intent or outcome of the action (e.g., `!placeBlock`, `!craftRecipe`).
- **Purpose Description:** A natural language explanation that describes the scenarios in which the action is applicable. This helps the LLM understand when and why the action should be used.
- **Parameter Specification:** Each action defines required or optional parameters, including type annotations and role descriptions. These are used to extract structured arguments from LLM-generated responses.
- **Perform Function:** The core of the action is a programmatic function (usually JavaScript or Python) that executes the behavior when the action is invoked. Parameters extracted from the LLM are passed to this function at runtime.

Move the AIC toward a target player.

```
{
    name: '!goToPlayer',
    description: 'Go to the given player.',
    params: {
        'player_name': {
            type: 'string',
            description: 'The name of the player to go to.'
        },
        'closeness': {
            type: 'float',
            description: 'How close to get to the player.',
            domain: [0, Infinity]
        }
    },
    perform: runAsAction(async (agent, player_name, closeness) => {
        await skills.goToPlayer(agent.bot, player_name, closeness);
    })
}
```

Acion 2.1: Move the AIC toward a target player.

In Action 2.1, it shows a concrete example of an action implemented using the MCP structure. The `!goToPlayer` action allows an AIC to move to the player's position.

When an action is selected and executed, the system monitors for success, failure, or unexpected feedback. Errors may arise due to invalid parameters, unavailable items, blocked paths, or timing issues. To handle such scenarios gracefully, the behavior system includes a retry mechanism with configurable retry limits. On each failure, the system can either reattempt the action with updated parameters or fall back to alternative strategies. All outcomes are logged and may contribute to long-term memory or learning.

The behavior system includes two powerful meta-actions that expand the expressive capacity of AICs beyond predefined routines:

- **`!goal`**: This action handles complex, high-level goals that cannot be achieved in a single step. It triggers a recursive decomposition process, in which the LLM breaks the overarching task into subgoals, each of which can be addressed via individual atomic actions. This mechanism enables AICs to handle tasks like "build a house" or "prepare for battle" with step-by-step reasoning.
- **`!newAction`**: This is a dynamic code-generation mechanism that allows AICs to extend their own behavior repertoire. Leveraging the LLM's ability to generate executable code, the AIC can synthesize new actions on-the-fly to meet novel needs. These generated actions can interface directly with map APIs, utility functions, or plugins. This creates a pathway for emergent generalization and tool-use capabilities in real time.

To ensure that only the most relevant actions are considered during decision-making, RAG techniques are applied. When a message is received, a semantic search is first performed over all registered actions using the message content. The top-k most relevant actions are then included in the prompt context, along with their documentation, for the LLM to choose from. This not only reduces hallucination but also improves planning efficiency and task grounding.

The behavior management module forms the reasoning and execution engine of each AI Character. It transforms unstructured input into structured, purposeful behavior through LLM planning, protocol-compliant action design, and adaptive execution strategies. By combining standardized action schemas, hierarchical planning, and real-time code generation, Minecraft AI delivers a highly extensible and resilient architecture for autonomous agents.

2.2.3 Memory Management

The memory management system in Minecraft AI plays a foundational role in maintaining context continuity, enabling AI characters to exhibit coherent long-term behavior. Inspired by the mainstream LLM-driven agent frameworks, the system is designed to persistently record and organize interactions between the AI character, the large language model, and the Minecraft environment. These records not only serve as historical references to avoid redundant queries but also support the construction of lifelike memory continuity and experience-based behavior evolution.

Minecraft AI adopts a hierarchical and modular memory architecture composed of three functional layers: personalized memory configuration, short- and long-term memory separation, and a pluggable memory processing engine.

- **Personalized Memory Configuration.** Each AI character can be initialized with a *personal description*, a set of configurable traits that guide its decision-making tendencies and behavioral style. These settings affect how the character interprets and prioritizes goals based on contextual data. For example, an exploration-oriented agent may prioritize uncharted terrain and prefer collecting rare resources such as diamonds; A collaborative engineer-type character may favor stable structures and frequently request assistance from teammates.

  This module serves as an input to the LLM prompting process and influences planning logic, thereby enabling behavioral differentiation across characters.

- **Short-Term Memory.** captures real-time contextual data essential for immediate decision-making. It includes the information like recent interaction records, current environment status (coordinates, terrain, weather), and inventory and item usage. Short-term memory is stored in temporal order and supplied as part of the LLM prompt for each inference cycle.

- **Long-Term Memory.** retains a historical trace of interactions and task executions, supporting experience-based planning and behavioral continuity across sessions. Inherited from MINDcraft, by default Minecraft AI employs LLM to generate a condensed narrative that preserves key actions and logic. For instance:

  > "Successfully defended village against skeleton and zombie raid using watchtowers and reinforced walls; resources consumed: 240 stone, 45 arrows."

  While concise, this strategy may lose granular details for complex interactions. An improvement approach is splitting the long-term memory by themes such as *construction*, *resource collection*, or *social interaction*. Each segment is indexed and stored separately. A retrieval engine dynamically selects and assembles relevant segments based on current context using techniques like Transformer-based similarity search or keyword matching. This hybrid method balances memory precision with prompt size constraints.

To ensure extensibility and decoupling, Minecraft AI implements a standardized memory processing API, enabling integration of diverse backends and algorithms.

By fusing behavior configuration, contextual layering, and modular strategy support, the memory management system in Minecraft AI enables each agent to develop a persistent internal model of its experiences. This architecture not only supports sophisticated task execution but also lays the groundwork for simulating long-term identity, skill accumulation, and collaborative memory among agents.

2.3 Action Triggers

In the Minecraft AI framework, all agent actions are ultimately grounded in the event-driven system provided by Mineflayer, the underlying bot control library. Every physical behavior exhibited by an AIC — from mining and crafting to building and fighting — is realized through invoking concrete Mineflayer interfaces. The role of Minecraft AI, therefore, is to serve as a cognitive layer that interprets context, plans behavior, and maps abstract intentions to executable actions via these interfaces. At the core of this design lies a simple yet powerful mechanism:

Message → Action Trigger → Bot Execution

The generation and interpretation of *messages* serve as the primary driver for action triggering, functioning as the cognitive "impulses" of AICs. These messages can originate from external or internal sources, and their handling determines how and when an action is executed. We categorize message sources and trigger types into three primary classes:

- **Player and Agent Interaction Signals (Chat-Based Triggers).** This is the most direct and interactive form of behavior activation. Messages are captured from the Minecraft chat channel, including those sent by players, other agents, or system scripts. The workflow includes:
  - **Source Filtering**: The framework distinguishes between message origins (e.g., self, player, other bots) and applies selective processing rules. For example, self-generated messages are ignored to prevent echo loops, while player-issued commands receive high priority.
  - **Content Interpretation**: Natural language processing is delegated to a large language model (LLM), which parses the message, determines intent, and optionally invokes function calls to trigger appropriate actions. This mechanism supports commands such as 'Lucy, build a house near the lake" or "Max, follow me and help fight monsters."
  - **Response Feedback**: After parsing and triggering, AICs respond via chat, acknowledging commands or reporting failure, thereby maintaining an interactive loop.
- **Internal Signals from Ongoing Tasks (Self-Generated Triggers).** While executing actions, the bot autonomously emits messages that reflect its progress, status, or encountered problems. These messages feed back into the message-processing engine to refine current or future plans:
  - **Progress Reports**: Messages such as "Collecting wood, 5 logs remaining" provide real-time status updates to the user and assist the agent in short-term memory consolidation.
  - **Error Feedback**: If execution fails (e.g., "No axe available to chop wood"), the agent can autonomously trigger fallback strategies or request user assistance.
  - **Task-Level Milestones**: For multi-stage tasks like construction, AICs generate phase-level summaries (e.g., "Foundation completed, starting wall layout"), facilitating modular planning and error recovery.

  This layer of messaging introduces a form of *self-awareness*, enabling agents to recognize execution bottlenecks and adapt behavior without user intervention.
- **Environmental Events and Passive Triggers.** Beyond human instruction and internal reasoning, Minecraft AI agents passively monitor game events to autonomously initiate actions in response to environmental changes:
  - **Periodic Checks**: Using in-game time (e.g., Minecraft ticks), AICs can schedule recurring behaviors such as "Harvest farmland every morning at dawn", enabled through event hooks on game time updates.

  - **Environmental Conditions**: Plugins can monitor changes like weather, nearby mob presence, or terrain alterations. For instance: "If rain starts, seek shelter and place torches".

  Developers can extend passive triggers via the plugin system, subscribing to native Mineflayer events and linking them to message generators or direct action invocations. This class of triggers enables context-aware autonomy and resilience in survival and exploration scenarios, even without explicit user input.

The action trigger mechanism in Minecraft AI combines human-interactive, self-reflective, and environment-reactive messaging into a unified behavioral pipeline. This design maintains full compatibility with the native Mineflayer execution model while enabling LLM-based decision-making and abstraction. It forms the behavioral backbone for building higher-level features such as long-term planning, emergent collaboration, and autonomous survival strategies in complex multi-agent environments.

2.4 Enhanced Interactions

A advancement of the Minecraft AI framework over its predecessor, MINDcraft, lies in its support for enhanced multimodal interactions. While retaining compatibility with the original chat-based interface, Minecraft AI introduces extended capabilities in three major dimensions: multimodal message input, vision plugin architecture, and rich speech feedback. These features enable AICs to engage in more natural, expressive, and context-aware interactions.

- **Multimodal Messaging Support.** Minecraft AI maintains MINDcraft's independent communication server, which allows external messages to be delivered to AICs outside the Minecraft client. This server, when coupled with a browser-based frontend, facilitates interaction beyond the default in-game chat, extending the interface to include multimodal inputs.

  - **Text**: Standard command or dialog messages remain supported as the default input format.
  - **Voice**: Real-time speech-to-text (STT) modules allow users to speak directly to AICs via microphone input. These are converted into actionable messages using natural language parsing.
  - **Image**: Users can send images to AICs via the web interface. With the help of external vision models (e.g., CLIP, BLIP, GPT-4V), these images are analyzed and converted into descriptive messages or direct action triggers.
  - **Video**: Experimental support exists for video snippets as contextual inputs, which can be interpreted via multimodal video analysis APIs.

  This architecture allows players to interact with AICs through intuitive, human-like channels and offers possibilities for scenario-based input (e.g., showing a blueprint or an object of interest) that are otherwise impossible within Minecraft's default GUI.

- **Vision Plugin Architecture.** To enable visual perception, Minecraft AI refactors the original MINDcraft vision module into a "VisionInterpreter" plugin. This design allows developers to selectively activate and customize visual analysis capabilities depending on task requirements.

  - **Plugin Flexibility**: Developers can register custom similar plugins that capture images from the game world and process them using external or local vision models.
  - **Contextual Activation**: Vision plugins can be triggered by specific conditions (e.g., entering a new area, facing an object) to enable responsive perception.
  - **Semantic Parsing**: Extracted image content is parsed into structured messages (e.g., object detection, spatial layout) and integrated into the AIC's message-processing pipeline.

The plugin system also lays the foundation for supporting additional perception modes such as depth estimation, gesture recognition, and semantic mapping, pushing Minecraft AI closer toward embodied AI applications.

- **Rich Feedback and Speech Output.** Minecraft AI extends the original speak module into a full-featured "Speak" plugin that supports both local and cloud-based Text-to-Speech (TTS) generation. This provides greater flexibility and realism in AIC feedback.
  - **Local TTS**: Basic speech synthesis can be performed using system-native TTS tools, such as `say` (macOS) or Windows SAPI. For Linux users, local TTS requires the installation of TTS tools such as `espeak`.
  - **Cloud TTS**: Integration with external services (e.g., Google Cloud, Azure, ElevenLabs) enables support for multilingual, customizable, and expressive speech synthesis.
  - **Voice Profiles**: Each AIC can be configured with a unique voice identity, enhancing personality consistency and enabling diverse interaction styles.

  This output mechanism makes it possible for users to hear AICs speak in natural voices, turning them from text-only bots into lifelike characters capable of vocal communication and narrative expression.

By integrating multimodal inputs, perceptual plugins, and expressive outputs, Minecraft AI transitions from static command-based control toward a dynamic, human-centered interaction model. As such, Minecraft AI is not only a toolkit for building game agents—it is a stepping stone toward rich, Embodied AI ecosystems that understand, respond to, and collaborate with humans in complex, grounded environments.

2.5 Extension as Plugin Systems

To support modularity, flexibility, and collaborative development, Minecraft AI introduces a robust plugin system architecture. This system enables developers to extend or modify the behavior of AICs without altering the core runtime. Inspired by software engineering best practices, the plugin architecture serves two core purposes: *capability separation* and *safe extensibility*.

- **Capability Separation**: The core capabilities of AICs—such as navigation, observation, and planning—are encapsulated and isolated from extended functionalities like multimodal perception or specialized task routines. This design minimizes coupling and simplifies debugging.
- **Safe Extensibility**: The plugin system ensures safe extensibility by allowing developers to keep the core logic minimal and robust, while enabling new features to be added independently without altering internal state management or risking unintended side effects in core behavior.

This system allows Minecraft AI to remain lightweight at its core, while offering extensive flexibility to scale to diverse use cases, from research prototypes to complex multi-agent simulations.

2.5.1 Typical Plugin Usage Patterns

One of the primary use cases of the plugin system is to define custom actions that AICs can perform. This addresses a common challenge in LLM-based planning: excessive candidate actions can burden the context window, causing longer inference times and degraded decision quality.

Beyond extending the action space, plugins can register event listeners to Mineflayer bot hooks and respond dynamically to in-game events.

- **Trigger-Based Behavior**: Events such as block updates, health changes, tool durability drops, or nearby entity detection can invoke specific routines. For instance, a plugin could define a rule like "if night falls and the agent is not near a shelter, seek cover automatically".

- **Routine Automation**: Common and repetitive tasks like periodic health checks, idle-time patrols, inventory audits, or navigation to waypoints can be modularized into reusable plugins. This reduces the need to encode such routines repeatedly in the LLM prompt.
- **Dynamic Response**: Reactions to environmental changes become easier to configure without modifying the core AI loop, enabling flexible behavior composition. For example, rain could trigger a "light-torch" action and bad weather could temporarily disable long-distance exploration.

On the output side, plugins can influence both the AIC and the game environment through multiple channels:

- **Memory Integration**: Plugins are capable of reading from and writing to the AIC's internal memory. This allows them to shape both short-term reactions and long-term strategic behavior. Typical use cases include tracking ongoing objectives, encoding environment-specific heuristics, and maintaining agent-specific knowledge for context-aware reasoning.
- **Inter-Agent Communication**: Plugins can dispatch structured messages either to the AIC itself or to other in-world entities, including players and fellow AICs. This enables dynamic workflows driven by internal or external stimuli, facilitates multi-agent coordination without central orchestration, and supports the emergence of complex social behaviors among agents.

These flexible usage patterns make the plugin system not just an extension mechanism, but a core enabler of behavior modularity, experimentation, and scalability in the Minecraft AI ecosystem.

2.5.2 Unified Capabilities Across Plugins

The Minecraft AI plugin system is designed not only for modularity and extensibility, but also to ensure a unified, powerful runtime interface across all plugin types. Whether a plugin functions as an action injector, a sensor module, or a behavioral extension, it operates under a common execution model that grants it deep integration capabilities and autonomy.

At its core, each plugin has direct access to the Mineflayer bot API, enabling it to invoke actions in response to real-time environmental changes, implement new routines, or override default behaviors under specific conditions. This allows plugins to operate with procedural logic, react adaptively to gameplay contexts, and modify the agent's behavior dynamically—such as switching into exploration or combat modes depending on situational triggers.

In addition to action control, plugins are tightly integrated with two fundamental mechanisms of the AIC architecture: memory and messaging. Plugins can read from and write to the AIC's internal memory module, allowing them to influence both short-term decision making and long-term behavioral tendencies. This facilitates the simulation of learning, situational awareness, and context preservation.

Together, these unified capabilities transform plugins from simple extensions into intelligent, autonomous modules. They are capable of perceiving, reasoning, and acting independently, all while remaining composable within the broader AI framework.

The impact of this system is far-reaching. It enables researchers and developers to construct sophisticated AIC behaviors by assembling and configuring plugins rather than modifying core logic. This supports iterative experimentation, collaborative development, and rapid prototyping of new modalities—from visual or auditory sensors to high-level cognitive skills such as diplomacy, strategy, or navigation.

Looking forward, the Minecraft AI plugin system will evolve toward full modularity and runtime flexibility. Planned features include dynamic plugin loading and unloading, dependency resolution, and persistent plugin state management. These advancements will further solidify Minecraft AI as a robust platform for embodied AI research, education, and application development.

2.6 Integrated Cognition

Integrated Cognition is one of the core mechanisms introduced by Minecraft AI, designed to enable embodied AICs to perceive, remember, reason, and act in a coherent and consistent manner. It achieves this by integrating information management, context-aware reasoning, and LLM capabilities into a unified cognitive interface. This approach turns LLMs from passive response generators into active reasoning agents, capable of maintaining internal states and making informed decisions based on historical context and current tasks.

Traditional use of LLMs in applications often follows a stateless question-answer pattern, which limits their ability to model persistent behavior or complex task progression. *Integrated Cognition* addresses this limitation by introducing a cognition pipeline that wraps around the LLM and provides structured memory access, context construction, and high-level reasoning.

This pipeline is abstracted into a standardized cognition API, which can be invoked by AICs without exposing the underlying complexity of prompt engineering or context management. From the AIC's perspective, making a cognition request is as simple as calling a RESTful or internal function interface, passing structured task-related data. From the system's perspective, this triggers a deeply contextualized reasoning cycle involving memory retrieval, context construction, and behavior generation. In this sense, *Integrated Cognition* can be regarded as the API interface to LLM-based agents, providing a structured and consistent way to invoke their reasoning capabilities.

The goal is to establish a "cognitive feedback loop", where the AIC does not merely respond to its environment, but interprets it in light of prior experience, ongoing goals, and internal state — thus behaving more like a cognitive entity than a reactive script.

2.6.1 System Architecture

Technically, *Integrated Cognition* is realized through a modular LLM-based agent that exposes cognitive capabilities to plugins, behavior trees, and high-level planning modules. This system forms the backbone of intelligent decision-making for each AIC, bridging perception, memory, and action through unified interfaces. The architecture is composed of the following core components:

- **Memory Manager**: Maintains structured and queryable records of the AIC's internal state, observations, long- and short-term goals, past actions, and interactions. This module supports bidirectional access—allowing cognitive routines to update or retrieve memories as part of each reasoning cycle.
- **Context Builder**: Dynamically assembles an optimal prompt for LLM queries by selecting and formatting relevant context elements, including agent status, environmental cues, task metadata, and memory snapshots. This automation ensures prompt efficiency and relevance, and removes the need for manual context engineering.
- **Reasoning Executor**: Interfaces with the LLM (or a hybrid model) to perform high-level cognitive tasks—such as decision making, behavior explanation, task decomposition, or inner monologue generation—based on the constructed context window.

All cognition requests—whether initiated by a behavior plugin, a sensory event, or a player-issued command—are funneled through this stack, ensuring consistent cognitive processing, internal state alignment, and coherent decision outputs across modules.

On the other hand, the Integrated Cognition system is designed for flexible deployment to accommodate diverse use cases and scalability requirements. It can operate in two modes:

- **Embedded Mode**: The cognition stack is embedded directly within the runtime of the Minecraft AI instance. This approach ensures low latency and tight coupling with in-game actions, ideal for local experiments and offline simulation.

- **Service Mode**: The cognition module can also be deployed as a standalone service, communicating with AIC instances via HTTP or WebSocket interfaces. This configuration decouples cognitive load from game logic and enables several advanced capabilities:
  - **Dedicated Resources**: Developers can allocate separate compute resources (e.g., GPUs, LLM API quotas) specifically for cognition tasks, improving performance under load.
  - **Cloud-Based Integration**: Deploying cognition modules to the cloud allows agents across multiple Minecraft servers or instances to share cognitive services, synchronize memory, and persist long-term knowledge across sessions.
  - **Scalability and Fault Tolerance**: Service-based deployment supports containerization, autoscaling, and fault recovery, facilitating large-scale experiments or distributed multi-agent simulations.

This modular and service-oriented architecture makes *Integrated Cognition* not just a runtime component, but a composable infrastructure layer for Embodied AI development. Whether used for educational demos, agent-based research, or cloud-deployed simulations, it enables agents to reason coherently, remember persistently, and act intelligently.

2.6.2 Key Capabilities of Integrated Cognition

The *Integrated Cognition* system is the foundation of AIC's decision-making, enabling context-rich reasoning, high-quality behavior generation, and persistent intelligence across sessions. It delivers critical advantages that fundamentally enhance how AI Characters (AICs) perceive, act, and evolve. These capabilities center on three pillars:

- **Centralized Context for Efficient and Informed Decision-Making.** Through *Integrated Cognition*, AICs access a unified cognitive interface that consolidates environmental observations, internal states, task metadata, and relevant memory snapshots. Instead of requiring handcrafted prompts or localized information assembly, this centralized model ensures that every decision is based on a comprehensive, pre-processed cognitive context. This design yields two key benefits:
  - **Computational Efficiency**: By abstracting context construction into a dedicated backend, the system ensures optimal prompt length, reduces redundancy, and avoids LLM token waste—resulting in faster and more cost-effective inference.
  - **Action Coherence**: Decisions made by the LLM agent are informed by the full situational background, improving alignment with ongoing tasks and increasing behavioral consistency across time.
- **Backend Fusion of State, Memory, and Task Awareness.** *Integrated Cognition* introduces a powerful separation of concerns: while local AIC logic handles real-time interaction and responsiveness, the cognition backend conducts deeper analysis of agent status, short- and long-term goals, memory history, and external context. This backend-level fusion provides:
  - **Robust Situation Awareness**: By aggregating and interpreting high-level AIC state data—including past missions, dialogue history, environmental changes, and inter-agent dynamics—the system maintains a consistent mental model of each agent's journey and surroundings.
  - **Uninterrupted Behavior**: These cognitive operations occur asynchronously or on-demand, meaning they do not interrupt or delay the local behavior execution loop. This allows real-time interaction and strategic reasoning to coexist fluidly.

- **Enhanced LLM-Driven Action Generation with Knowledge Integration.** With full situational context and memory continuity, the cognition system enables AICs to generate high-quality code and behavior plans using LLMs. This is especially impactful when creating novel actions, extending the action space, or adapting to unexpected situations. Key capabilities include:
  - **Code Quality and Safety**: By grounding LLM-generated code in the agent's current goals, environmental constraints, and memory trace, Integrated Cognition reduces hallucinations and ensures that generated code aligns with both explicit instructions and implicit gameplay limitations.
  - **Knowledge-Infused Planning**: The cognition system can integrate with external knowledge bases or game-specific APIs, enriching the LLM's decision space with domain expertise. This enables advanced planning strategies, more realistic behavior trees, and a deeper connection between perception and action.

By combining efficient context assembly, backend cognition, and knowledge-grounded code generation, *Integrated Cognition* transforms AICs from reactive rule-followers into proactive, self-consistent digital characters. This architecture also lays the groundwork for large-scale, multi-agent systems where each AIC can adapt, collaborate, and evolve across sessions and environments.

2.7 Discussion

The Minecraft AI project builds upon the foundation laid by MINDcraft, expanding it into a more modular, flexible, and developer-friendly framework for embodied AI experimentation. One of the core motivations behind this redesign is to lower the barrier for both researchers and hobbyists to create, customize, and scale AICs across a wide variety of testbeds—from academic simulations to rich entertainment scenarios.

The system introduces a clean modular control framework, where each AIC is managed as a container of cognitive and behavioral modules that can be swapped, extended, or tuned independently. This promotes a plug-and-play development philosophy, allowing users to quickly try new planning strategies, integrate custom sensors or tools, or run agents in specialized tasks without disrupting the entire architecture.

Moreover, by exposing cognition and behavior generation through well-defined interfaces, Minecraft AI encourages experimentation with different reasoning strategies, including prompt tuning, fine-tuned models, or structured memory modules. The goal is not just to control agents efficiently, but to enable continual, adaptable, and expressive intelligence—AICs that can reflect, remember, and evolve across play sessions and contexts.

However, through practical implementation and community feedback, a recurring observation has emerged: while the framework supports extensibility and advanced integration, its architectural complexity may be overwhelming for newcomers or lightweight applications. Researchers focused on learning dynamics or LLM behavior analysis may not always require full agent stacks, while developers building interactive games may prioritize simplicity and rapid iteration over full cognition fidelity.

This tension between "architecture complexity" and "cognitive clarity" led us to reflect on a more minimal yet powerful abstraction to organize the key components of AIC intelligence. As a result, we propose the **MARC** architecture—a conceptual interface that separates memory, action, reflection, and code generation, offering a more interpretable and adaptable bridge between AIC behavior and large model capabilities.

MARC is not a replacement for the Minecraft AI framework, but rather a lens through which developers and researchers can better understand and experiment with the decision-making core of an intelligent agent. It enables clearer reasoning about when and how large models should be invoked, what memory structures are needed, and how action generation can be aligned with context and goals.

In the next section, we introduce the MARC architecture in detail, highlighting how it can be layered on top of existing AIC implementations to provide better modularity, traceability, and adaptability—paving the way toward more scalable and explainable agent intelligence.

Chapter 3

The MARC Architecture

3.1 Ideas Behind MARC

Memory-Action-Reflection-Coding (MARC) is a modular architecture designed to define the control logic of AI Characters (AICs) and to standardize their interaction with large language models (LLMs). By establishing a clear communication protocol, MARC improves the efficiency of information processing and transmission between decision-making agents and predictive models.

As a protocol layer between AI agents and LLMs, MARC decouples the responsibilities of memory management, action selection, reflective evaluation, and adaptive coding, allowing both ends (AIC and LLM) to maintain modular independence while working in coherent unison. The architecture ensures that contextual information is organized, passed, and acted upon systematically across different agent operations.

In the process of *Integrated Cognition*, MARC provides a structured scaffold to guide LLM-based agents. By defining the core components that LLMs should consider, the architecture helps maintain relevant context boundaries, ensuring the inclusion of essential information while filtering out noise. This improves the predictive accuracy and consistency of the LLM across multi-step decision-making tasks.

From the perspective of the AI Character, MARC guides the logic of when and how to interact with the LLM, emphasizing clarity and coherence at each communication node. This allows AICs to make consistent, well-informed decisions by managing the sequence of information usage effectively. It also ensures the seamless integration of LLM-generated outputs into real-time agent behavior, reducing errors and improving responsiveness in dynamic environments.

The core philosophy of MARC is to embed structured cognition within the AI agent's behavior pipeline. Rather than treating the LLM as a black-box oracle, MARC provides well-defined modules and roles that reflect human-like cognitive stages—memory recall, action deliberation, reflective adjustment, and code-based problem-solving. This structured guidance reduces ambiguity and enhances the interpretability and reliability of complex agent systems. As the name suggested, the MARC architecture is composed of four fundamental components: **Memory** (manages layered memory systems for short-term and long-term knowledge), **Action** (defines the interface for decision-making and feedback handling), **Reflection** (enables meta-cognitive assessment, error handling, and goal alignment), and **Coding** (dynamically generates new behaviors or routines through programmatic reasoning).

Among them, Memory and Action are considered essential and must be implemented in every AIC. Reflection and Coding serve as optional enhancements depending on the complexity and adaptability required by AICs. MARC thus offers a unified yet flexible design framework, empowering developers and researchers to build LLM-driven agents that are both interpretable and capable of long-term learning and adaptation.

3.2 Memory: Enabling Behavioral Coherence and Consistency

The **Memory** component is a cornerstone of the MARC architecture, designed to support the development of intelligent, believable, and context-sensitive AICs. In contrast to traditional rule-based or stateless agent-frameworks, the memory module in MARC enables AICs to retain, organize, and utilize past experiences in order to sustain both behavioral continuity and social consistency across time.

Rather than acting as a passive record of prior events, memory in embodied agents—particularly those powered by LLMs—functions as a dynamic and structured knowledge substrate. It allows AICs to reflect on past actions, recall relevant context, maintain long-term objectives, and adhere to commitments made in dialogue or collaboration. In doing so, it bridges short-term reactive behavior with long-term narrative development, laying the groundwork for agents that are not only functional but also believable and consistent across extended interaction windows.

3.2.1 Decision Coherence Through Contextual Awareness

One of the core responsibilities of the Memory Component is to maintain *decision coherence*—the ability for an AIC to make decisions that are contextually grounded and narratively consistent over time. In the MARC architecture, this is achieved by continuously capturing, organizing, and selectively retrieving information from the AIC's past experiences to inform future actions. Specifically, the memory component retains structured traces of the following:

- **Past actions and their outcomes:** Including both successful and failed attempts, enabling the agent to learn from experience and avoid redundant or contradictory behaviors.
- **Short-term and long-term goals:** Maintaining a hierarchy of objectives allows the agent to align immediate actions with overarching intentions.
- **Perceptual changes:** Tracking how the external environment has evolved due to the agent's (or others') actions—crucial for dynamic adaptation.
- **Internal reasoning and reflections:** Preserving intermediate thoughts, hypotheses, and planning stages to support continuity in thought processes.

By integrating these memory traces during decision making, an AIC can anchor its next action not just to the current stimulus, but to a temporally extended context. This enables coherent multi-step behavior and significantly reduces the risk of contradiction or inconsistency in long-term interaction.

Consider a Minecraft AIC engaged in building a stone house. Rather than treating each instruction—such as "mine stone", "place block", or "build wall"—as an isolated command, the AIC continuously recalls: the initial goal to construct a house with specific materials; previously collected stone quantity; incomplete parts of the structure; any constraints or preferences expressed by the user (e.g., "make it symmetric").

As a result, the AIC can resume work after interruptions, adjust the plan if stone supply runs low, and explain its actions when asked—all hallmarks of coherent, context-aware behavior.

This coherence is not just an aesthetic improvement; it is essential for agents operating in persistent, multi-task environments. Without contextual memory integration, agents risk regressing into disjointed or repetitive behavior, breaking the illusion of intelligence and frustrating human collaborators.

3.2.2 Behavioral Consistency Through Social Context

While internal planning coherence is essential, intelligent agent in a shared virtual world must also demonstrate social awareness and behavioral consistency in multi-agent interactions. The Memory Component plays a critical role in this dimension by recording and retrieving socially relevant context, ensuring that an AIC's actions remain aligned not only with its own history but also with the expectations and commitments formed through interaction with others.

Specifically, the memory system encodes a range of socially relevant information, including dialogue history with players and other agents—such as promises, requests, and unresolved queries—as well as explicit agreements and coordination plans, like task division or turn-taking protocols in collaborative scenarios. It also captures inferred social cues, including roles, emotional states, and personal preferences derived from prior interactions. These memory traces enable AICs to honor social commitments, sustain shared situational awareness, and modulate behavior to remain aligned with collaborative objectives. For example, if an AIC agrees to help another character gather wood before sunset, its memory ensures that this commitment is remembered and prioritized, even when new tasks arise. In group construction tasks, memory allows each AIC to track overall progress, synchronize material usage, and coordinate actions with teammates—thereby reducing redundancy and enabling fluid cooperation.

Moreover, this socially grounded memory enables long-term interaction patterns. AICs can recognize and adapt to the habits or personalities of familiar users and teammates, improving perceived intelligence, trustworthiness, and emotional believability.

In sum, behavioral consistency in social settings is not only a technical necessity for multi-agent systems—it is also a cornerstone of the Embodied Turing Test vision.

3.2.3 Memory Organization and Representation

To support coherent, scalable, and contextually relevant decision-making over long-term interactions, the memory system may involve information in several hierarchical layers. This structure reflects human-like cognition by separating memory into layers that capture different temporal and conceptual scopes of experience. The three primary layers are:

- **Episodic memory**: This component stores temporally ordered records of specific events, actions, observations, and dialogues. Each memory trace is typically time-stamped and associated with contextual metadata (e.g., location, agent involved, emotional tone). This layer enables agents to reconstruct the sequence of past interactions and respond to time-dependent queries such as "What was I doing yesterday?" or "Who gave me this task?"

- **Semantic memory**: Over time, agents abstract patterns and generalizations from their episodic memories. These distilled insights are stored as semantic knowledge—facts, rules, or common strategies learned from repeated experiences. For example, an agent may learn that "wood and stone are typical materials for building towers," or that "players usually prefer to gather resources before sunset." Semantic memory supports more efficient reasoning and enables transfer of knowledge across contexts.

- **Reflective memory**: The highest layer encodes meta-cognitive representations—insights gained from self-reflection, failures, or strategic evaluations. This includes lessons learned from mistakes, personal preferences, strategy assessments, or summaries of social experiences. For example, an agent might reflect that "I tend to succeed more when collaborating with Max on mining tasks" or "My last attempt at group coordination failed due to unclear goal division." Reflective memory allows for self-improvement and behavioral refinement over time.

This layered organization enables AICs to operate across both immediate context and long-term knowledge, enhancing their ability to plan, generalize, and adapt. It also facilitates memory querying and prioritization. For instance, when asked to explain a decision, an agent might pull relevant facts from semantic memory, reference a specific episode to justify its context, and cite a prior reflection to support its judgment. This multi-level memory structure thus underpins goal continuity, self-awareness, and explainable behavior—all critical for human-aligned Embodied AI.

3.2.4 Memory Implementation in Minecraft AI

The memory component in Minecraft AI inherits design from the MINDcraft architecture, where memory is not treated as a simple chronological log but rather a set of semantically meaningful and dynamically updated summaries. This approach focuses on both general awareness and goal-specific relevance, making it well-suited for LLM-driven embodied agents operating in complex, persistent environments. By default, each AIC maintains two key types of summarized memory traces:

- A **general summary piece**, which condenses the AIC's overall interaction history, dialogue context, and episodic milestones. This serves as a persistent background context for most reasoning and planning activities.
- A **goal-oriented summary piece**, which captures the current long-term objective, including progress, setbacks, and past decisions related to the goal. This allows the agent to align short-term behavior with ongoing strategic intent.

Both summaries are personalized according to the AIC's individual profile—a lightweight metadata structure that defines personality, behavioral style, memory capacity, and role preferences. This ensures that memory updates are not only based on event content but also shaped by the AIC's cognitive traits, yielding more coherent and consistent decision-making across time.

Building upon this structure, Minecraft AI introduces an **extended memory model** that supports multiple topic-based memory summaries. Instead of relying on only one general and one goal-specific summary, this design enables agents to organize their experiences around various themes or topics.

At retrieval time, the memory system employs RAG techniques to dynamically select the most contextually relevant memory segments and inject them into the prompt presented to the LLM. This enables the agent to reason and act based on a compact yet meaningful subset of its accumulated experience.

3.3 Action: Adaptive Execution in a Dynamic World

The **Action** component in the MARC architecture serves as the decision-making core responsible for selecting the next operation based on the agent's current status and task demands. Actions may include sending structured messages to other AI AICs, or executing compound physical operations via Mineflayer bot APIs.

By offloading certain planning responsibilities to the LLM and retaining local control over safety, feedback, and iteration, the Action module becomes a powerful bridge between deterministic systems and creative reasoning engines. This approach supports the execution of open-ended, non-scripted tasks, allowing AICs to act as truly adaptive agents in complex simulated worlds.

3.3.1 Context-Aware Action Selection

At the heart of the Action component lies the principle of situational awareness. Instead of blindly following high-level instructions, it should at first interprets the environment and internal status of AIC to determine the most appropriate immediate action. For example, if the AIC receives a command to gather wood but is currently submerged underwater, it must first prioritize surfacing to ensure survival before beginning the wood-collecting task. This context-sensitive filtering capability ensures safe and coherent execution of complex objectives.

After executing an action, AIC should update its internal status and environmental understanding, in order to enable the **Reflection** component to assess whether the action had the intended effect, and it can also involve retry mechanisms or adaptive planning, ensuring the next action aligns with the evolving context.

This cycle ensures continuity and robustness, particularly in dynamic environments like Minecraft where the world may change unpredictably.

3.3.2 Abstract and Programmable Actions

When tasks become more complex or less deterministic, the Action module can invoke two powerful abstractions designed to extend the agent's capabilities beyond rigid predefined behaviors:

- `!goal`: This abstract action is used to define a high-level objective that may require multiple steps to complete. Once triggered, the agent enters a subroutine where it repeatedly analyzes the current state and selects appropriate actions over multiple iterations. This mechanism is particularly effective when the objective must be decomposed into a sequence of interdependent subtasks—such as "build a shelter" or "gather materials for a farm". In these cases, the agent continuously re-evaluates progress toward the goal, adjusting its behavior accordingly until completion or failure is reached.
- `!newAction`: In contrast, this meta-action allows the agent to programmatically create new composite actions on the fly by calling a LLM. These generated actions often involve chaining together existing bot APIs or invoking a set of predefined commands in a novel sequence. For example, a `!newAction` might encapsulate a behavior like "scout a nearby area, build a lookout post, and return" by stitching together lower-level actions such as movement, block placement, and orientation control.

The `!goal` abstraction excels in long-term planning and decomposition of an abstract task into an adaptive action sequence, making it ideal for open-ended objectives that evolve based on environmental feedback. On the other hand, `!newAction` is well-suited for scenarios where the task can be expressed as a finite, programmable sequence leveraging existing APIs and routines—essentially enabling just-in-time behavior composition.

In practice, the line between the two can be blurry. Some goals may appear abstract but are better addressed through explicit programming, while some composite actions may unfold into long-term strategies. As such, developers must make context-sensitive choices when prioritizing between `!goal` and `!newAction` in different applications. MARC's architecture accommodates both modes, offering flexible pathways to bridge high-level intent with low-level execution in ever-changing virtual environments.

3.3.3 Action Selection via MCP

To enable robust, interpretable decision-making within the Action module, it is convenient to adopt a context-driven action selection approach inspired by Anthropic's *Model Context Protocol* (MCP) [Anthropic, 2023]. This protocol emphasizes explicit, structured message passing between the agent's internal components and the LLM planner, ensuring that decisions remain grounded in both world state and agent intent. In a typical implementation, each action selection cycle begins by constructing a serialized context package that includes:

- **Current Status:** A concise but information-rich summary of the agent's present state, including location, inventory, nearby entities, active tasks, and environmental conditions.
- **Goal or Intent:** The agent's current short-term objective, derived from user commands or higher-level planning components.
- **Action History:** A brief log of recent actions and their outcomes, including failures or retries, to help the LLM avoid redundant or ineffective decisions.
- **Environment Snapshot:** A distilled representation of the surrounding world state (e.g., key block types, obstacles, time of day) to provide spatial and situational grounding.

This structured context is serialized into a prompt-like input and sent to the LLM, which is queried with a specific instruction: select the next best action from a known action set, or propose a new one (e.g., `!newAction`) as needed. The output is parsed by a dedicated executor that validates feasibility, dispatches the action, and manages post-action reflection or retries. This MCP-inspired approach empowers the Action

componnet to reason not in isolation, but as a grounded, context-aware AIC embedded in a dynamic world—laying the foundation for embodied intelligence that evolves through interaction.

3.4 Reflection: Sustaining Autonomy through Self-Initiated Behavior

The Reflection module plays a critical role in enabling MARC agents to exhibit self-correcting behavior and long-term intentionality. It operates at two complementary levels: post-action evaluation and goal continuity through proactive reflection.

At the first level, Reflection acts as an internal reviewer of actions that have just been executed. By analyzing the outcome of these actions in relation to the environment and the current goal, the agent determines whether an action should be retried, modified, or abandoned. This is particularly essential in dynamic, uncertain environments like Minecraft, where unexpected events—such as environmental hazards or changes in terrain—can disrupt execution even for well-formed plans.

At the second level, Reflection serves as a cognitive checkpoint that helps the agent maintain alignment with broader goals, especially when short-term tasks have been completed or interrupted. Through periodic or event-triggered introspection, agents are able to recall open-ended objectives or deferred long-term plans, and take initiative to act on them when contextually appropriate. This enables the AICs to not only react adaptively, but also drive their own behavior forward, even in the absence of external commands—demonstrating early forms of autonomous, goal-consistent reasoning.

3.4.1 Action Outcome Evaluation and Retry Logic

To ensure agents maintain responsiveness and reliability during task execution, the Reflection component operates in close coordination with the Action component, forming a tight feedback loop. Rather than functioning as an isolated review system, reflection is woven into the action execution process itself—monitoring each step, interpreting contextual signals, and determining appropriate follow-ups based on real-world outcomes.

After every executed action, the Reflection module assesses its effectiveness by observing the resulting changes in the environment and comparing them against the expected effects. This continuous evaluation enables two primary forms of error recovery:

- **Direct Retries**: For predefined atomic actions (e.g., movement, mining, or block placement), failures—such as path obstructions, target unavailability, or timeouts—are often transient. In these cases, the Reflection module can trigger immediate retries, optionally incorporating minor adjustments (e.g., reorienting the agent or adding a short delay) to improve success likelihood. For example, if an AIC tries to place a block and the placement fails due to the target being out of range, the agent may say:

  > "Placement failed—adjusting position slightly and retrying."

  Or, if it attempts to walk forward but hits a wall:

  > "Path blocked. Let me try a different angle."

- **Strategic Rethinking**: For abstract or multi-step actions like `!goal`, which rely on iterative planning and long-horizon execution, reflection becomes more interpretive. It inspects intermediate state transitions, evaluates progress toward the broader objective, and decides whether to continue with the current subtask, adjust the plan to accommodate new circumstances, or abort and replan entirely.

  For instance, if the goal is to "gather wood," but the agent finds itself in a desert biome, it might reflect:

  > "No trees nearby. Current path inefficient—replanning route toward forested area."

Or, if it's building a structure and a key resource is missing:

> "Missing cobblestone for next layer. Switching to resource collection before resuming build."

This tight reflection-action integration forms a resilient behavior loop that empowers AICs to dynamically respond to unexpected conditions. It mitigates the risk of cascading errors, prevents rigid adherence to invalidated plans, and promotes situational adaptability—a critical capability for agents operating in open-ended, partially observable environments like Minecraft.

3.4.2 Long-Term Task Reinvocation via Contextual Reflection

Beyond immediate correction, the Reflection component also serves as a bridge to long-term continuity. When short-term tasks are completed or the AIC enters an idle state, Reflection can be triggered—either periodically or by contextual cues—to revisit persistent goals or high-level missions.

These reflective prompts often result in self-initiated behaviors that align with the AIC's broader narrative or strategic agenda, such as: resuming construction on an unfinished structure; exploring new areas to gather resources for an overarching project; re-engaging in collaborative efforts with other AICs after task divergence.

To avoid task collisions, the Reflection component should ensure that such autonomous behaviors are only launched when environmental conditions are safe and no immediate goals are in conflict. This mechanism empowers AICs to act not just reactively, but proactively, maintaining coherence between moment-to-moment actions and longer-term intentions.

In essence, the Reflection component allows agents to learn from failure, adapt in real time, and remain aligned with their broader objectives—hallmarks of intelligent, autonomous behavior in a dynamic world.

3.5 Coding: Introduce the Infinite Possibilities

The **Coding** component enables AIC agents to generate and execute code at runtime in response to complex or previously unseen tasks. Rather than relying solely on predefined routines, agents can dynamically create new behaviors through the code-generation capabilities of LLMs.

For example, in Minecraft AI, such code is expressed as JavaScript functions compatible with the Mineflayer Bot API, and executed in a sandboxed environment to safeguard runtime stability and security.

3.5.1 Coding Capability and Output Variability

The quality, safety, and adaptability of runtime code generated by the Coding component are closely tied to the underlying LLM's coding capability. This capability goes beyond merely producing syntactically correct code—it involves context-aware reasoning, robust use of APIs, and the anticipation of potential failure modes. We classify this capability along three dimensions:

- **Correct API Usage:** A competent model must understand and properly invoke APIs exposed by the Mineflayer's bot module. Failure to do so can lead to errors, undefined behaviors, or silent failures. For example, a capable model might correctly generate: `bot.pathfinder.setGoal(new GoalNear(x, y, z, 2));`. But a weaker model might return a non-existent method such as: `bot.walkTo(x, y, z);`
- **Runtime Context Awareness and Error Handling:** Beyond correct syntax, high-quality code must account for the validity and availability of runtime objects and services. A robust implementation might verify the presence of critical modules (e.g., `bot.pathfinder`) before invoking their methods; detect if a module is temporarily unavailable and employ retry mechanisms or report back to the Coding component; and gracefully handle asynchronous errors through promises or event listeners, rather than relying on blocking calls or risky assumptions.

For instance, when invoking `bot.pathfinder.setGoal()`, a resilient model should first check:

```
if (bot.pathfinder && bot.pathfinder.setGoal) { ... }
```

This ensures that transient failures do not escalate into larger execution breakdowns.

- **Corner Case Sensitivity and Spatial Reasoning:** Many tasks require awareness of edge conditions or environmental constraints. For example, when placing a block at a target position, a smart model should check whether there is a solid adjacent block for attachment, or whether scaffolding is required. In another case, if no built-in helper is available, it should fall back to generating scanning logic based on block types and adjacency.

  This requires not only code generation but also spatial planning and environment querying, which tests the model's integrated understanding of game mechanics.

These capabilities—and their limitations—vary significantly across LLMs, and are further influenced by context design, token length, few-shot prompting, and temperature settings. By monitoring these variables and isolating weak behaviors through sandbox testing, the MARC system can selectively filter, revise, or reinforce agent behaviors, improving both execution safety and mission success rates.

3.5.2 Isolation and Context Maintenance

Coding tasks are treated as functionally independent units, even when embedded within higher-level composite instructions such as `!newAction`. Each task is completed through one or more rounds of code generation, orchestrated by the Coding component, which is responsible for preparing and maintaining a precise execution context.

The coding context typically includes several structured components designed to ground the model's code generation process in both semantic clarity and system-level constraints. These components work together to simulate a lightweight development environment tailored to the current task:

- **A concise description of the agent's environment and task goals:** This description provides situational awareness, enabling the LLM to align its output with the physical constraints and intent of the task. For example, the prompt might state: "The agent is in a cave with limited lighting, attempting to place torches every 5 blocks while avoiding water tiles." Such context helps ensure that the generated code is not only functionally correct but also situationally appropriate.

- **A curated list of accessible API functions and recommended usage patterns:** To reduce ambiguity and improve alignment with the agent's execution engine, the Coding module presents a narrowed set of relevant functions (e.g., `bot.placeBlock()`, `bot.findBlock()`, `bot.setControlState()`) along with best practice usage. These may include inline examples, warnings (e.g., "avoid direct teleportation without updating state"), or preferred alternatives (e.g., "use `bot.pathfinder.setGoal()` instead of manually walking to a position"). This encourages adherence to tested patterns and reduces misuse of the API surface.

- **Implementation guidelines and constraints:** These inform the model of system-level expectations to ensure generated code is efficient, stable, and aligned with architectural principles. Key requirements typically include adopting an asynchronous design—returning promises or using `async/await` to prevent blocking behavior—and respecting time or resource limits, such as avoiding full-world scans in favor of localized operations or capped entity lists. The model is also expected to handle failure gracefully: for instance, if a required block is not found, the function should return `null` or trigger an appropriate error handler, rather than causing an unhandled exception. Additionally, the output should include basic runtime status reporting through mechanisms like `bot.chat()` or logging, to facilitate feedback and debugging during task execution.

This context-aware design ensures that code is generated within safe, verifiable boundaries, reducing the risk of side effects or environment disruption. It also allows fine-grained control over behavioral complexity, enabling modular testing and debugging.

The Coding component exemplifies a broader principle of AIC's design philosophy: effective LLM usage requires more than generic language understanding or basic code generation. Instead, it imposes a dual-layered competency expectation on the model:

- **General-purpose programming ability**, such as generating syntactically valid and logically sound JavaScript code, reasoning about control flow, handling asynchronous logic, and understanding modular design principles.
- **Domain-specific framework expertise** is also essential, particularly regarding best practices within the Mineflayer ecosystem. The model must understand not only the available APIs but also how to use them effectively in a real-time, event-driven environment.

Failure to meet either criterion can lead to fragile, unpredictable, or unsafe behaviors. Conversely, a well-configured LLM equipped with both forms of competency can serve as a powerful, context-sensitive micro-planner capable of generating efficient and reliable action-level implementations on the fly.

In this sense, the Coding component is not merely a code execution tool—it is an architectural bridge between natural language reasoning and programmatic embodiment, and a stress test for the alignment between large model generalization capabilities and real-world implementation domains.

3.6 Summary

The MARC architecture provides a structured and modular framework that clarifies how AICs interact with LLMs across four specialized components—**Message**, **Action**, **Reflection**, and **Coding**. Each component defines a distinct interaction pattern and functional boundary, effectively mapping the application scenarios and capability expectations for LLMs in an embodied agent setting.

This architectural decomposition serves two critical purposes. First, it implicitly highlights the need for *Integrated Cognition*, where the selection and invocation of LLMs must consider differences in reasoning ability, coding accuracy, and cost-efficiency. By assigning different roles to LLMs across modules, MARC encourages the orchestration of heterogeneous model capabilities—leveraging stronger models where reasoning depth is needed and more efficient ones for routine or context-aware processing. This not only optimizes system performance but also surfaces concrete directions for improving underlying models in areas such as error recovery, interface reasoning, or domain-specific coding proficiency.

Second, by centralizing agent logic into these four cognitive components, MARC enhances the transparency, composability, and maintainability of AIC behavior design. It provides a clear blueprint for balancing the intelligence benefits of model-based decision-making against the uncertainties and overheads introduced by model invocation. This balance is essential in crafting agents that are not only capable of intelligent, autonomous behavior but also robust and predictable in dynamic, open-ended environments.

Through the MARC framework, we aim to bridge the gap between general-purpose LLM intelligence and embodied, context-sensitive action—bringing structured cognition to life in interactive game worlds and beyond.

Chapter 4

Open Source Project: Minecraft AI-Python

4.1 Overview

Minecraft AI-Python [1] is a research-oriented reimplementation of the original Minecraft AI system, inspired by the modular architecture and cognitive logic separation introduced in the MARC framework. This version retains the core principles of embodied intelligence while adapting the infrastructure to better align with the needs of education, experimentation, and long-term behavioral research.

From a design standpoint, Minecraft AI-Python simplifies many of the operational complexities present in the JavaScript-based implementation. This simplification is intentional and reflects the typical nature of educational and experimental scenarios, where agents are often evaluated in structured environments with clearly defined tasks. In these settings, the trade-off of reduced execution efficiency is more than offset by the benefits of enhanced transparency, debuggability, and traceability. By enforcing a cleaner, sequential logic of perception, action, and reflection, the Python version enables easier analysis of agent behavior and facilitates the development of more interpretable AI systems.

Moreover, beyond its pedagogical value, this implementation also aims to provide a controlled framework for conducting Embodied Turing Test experiments. In the JavaScript version, AICs are capable of concurrent multi-task execution and can access highly detailed system-level information. While powerful, this setup introduces significant challenges for maintaining internal coherence and coordinating collaboration among multiple agents. It also increases decision-making complexity, especially when combined with multimodal inputs or high-frequency context updates.

In contrast, Minecraft AI-Python adopts a deliberately simplified, message-driven execution model. All agent behaviors are triggered and managed through a single-threaded message loop, which sacrifices concurrency in favor of predictability and reproducibility. This makes it particularly suitable for evaluating how agents perform under constrained, well-scoped tasks—providing valuable insight into their planning efficiency, memory usage, and general reasoning capabilities. Such controlled scenarios serve as effective testbeds for incrementally improving AICs and isolating the sources of behavioral bottlenecks.

The architecture of Minecraft AI-Python is centered on three key design principles that make it especially well-suited for educational and research-oriented applications in embodied AI. First, the agent's internal logic is cleanly modularized following the MARC (Memory–Action–Reflection–Coding) architecture. This modularization separates memory management, action execution, reflective reasoning, and dynamic code generation into distinct, inspectable components—each with its own responsibilities and interfaces. This design not only enhances transparency and extensibility but also allows developers to trace, debug, or augment specific cognitive behaviors with precision.

[1] https://github.com/aeromechanic000/minecraft-ai-python

Second, agent execution is entirely message-driven, relying on a single-threaded loop that processes structured messages as the sole trigger for planning and action. This approach simplifies task coordination, removes the complexity of concurrent event handling, and ensures deterministic behavior—crucial for controlled experiments and reproducible classroom scenarios.

Finally, the agent's reflective capacity plays a central role in its decision-making loop, allowing it to revise goals, reevaluate actions, and adapt to evolving contexts.

The implementation of Minecraft AI-Python realizes a transparent cognitive loop where every behavior can be traced back to a message, every action is selected based on updated memory, and every long-term task is driven by reflective metacognition. This design encourages modular experimentation, supports reproducible research workflows, and provides a highly controlled framework for studying embodied AI behavior, particularly in educational and scientific contexts.

4.2 Implementation of MARC Architecture

Minecraft AI-Python fully adopts the MARC architecture to establish a modular, interpretable, and extensible cognitive framework for embodied agents. This implementation redefines the internal control logic of AICs with a focus on educational and research applications, while aligning with the two core design goals of the MARC framework:

- **Logical Transparency:** The standardized control loop and clear separation of responsibilities across cognitive modules make the AIC's reasoning process easy to trace, debug, and analyze—ideal for classroom demonstration, algorithm introspection, and controlled experiments.
- **Scenario Extensibility:** The modular architecture allows each cognitive component to be independently examined, extended, or replaced, facilitating rapid iteration for specific tasks, collaborative research, and hypothesis-driven development.

4.2.1 Memory and Action Modules: Inheritance and Optimization

Minecraft AI-Python inherits core abstractions for Memory and Action from the original JavaScript-based Minecraft AI framework, preserving proven patterns for storing AIC's knowledge and triggering behaviors. However, the internal linkage between message processing, memory updates, and action selection has been redesigned for greater determinism and minimalism.

- **Message-Driven Memory Update:** All agent inputs, whether from the environment, user, or internal components, are standardized as message objects. Every message is either filtered or stored and used to update memory content in a structured manner.
- **Synchronized Action Dispatch:** An action may only be executed after memory is fully synchronized with the latest messages. The memory module identifies unresolved goals or opportunities, which are explicitly marked and passed to the action selector.
- **Execution Chain Abstraction:** The entire cognitive pipeline is reduced to a clean loop:

  $$\textbf{Message} \rightarrow \textbf{Memory Update} \rightarrow \textbf{Action Selection} \rightarrow \textbf{Execution}$$

  All external triggers (e.g., timeouts, sensory input) must generate new messages to participate in the loop, ensuring full traceability and repeatability.

Minecraft AI-Python provides strong research utility by enforcing a unified knowledge representation through standardized memory summarization and fusion, ensuring consistent internal states and coherent reasoning across tasks. Its single-step execution model, where each cycle triggers only one action, enables

fine-grained observation of agent decision-making—ideal for educational use, automated testing, and empirical research on goal-directed behavior. While this strict sequential design may limit the agent's ability to handle time-sensitive or parallel tasks, it offers valuable trade-offs in interpretability, control, and cognitive debugging within focused experimental scenarios.

4.2.2 Reflection Module: Core for Multi-Step Reasoning

The Reflection module plays a central role in enabling intelligent, multi-step reasoning without compromising the simplicity of single-action execution. Rather than encoding task-specific logic or predefining multi-step plans, the system delegates all such complexity to Reflection, which functions as the agent's metacognitive engine. It examines memory, interprets context, and formulates new actions as structured messages—thereby maintaining the clean "Message → Memory → Action" cycle.

- **Event-Triggered Reflection:** Each time an action is completed and its outcome is committed to memory, Reflection is triggered to analyze results, update progress, and determine what should happen next. This ensures that agent cognition is continuously informed by its environment and past behavior.
- **Timed Reflection:** In addition to being event-driven, Reflection can also be scheduled to run periodically. This allows the agent to reassess goals and replan even in the absence of new external input, preventing cognitive stagnation and supporting long-term autonomy.

Unlike systems that rely on rigid multi-step plans or hardcoded task workflows, the Minecraft AI-Python framework enforces a strict single-action execution rule: if a task must be decomposed into atomic steps, then each step must be explicitly proposed by Reflection through a new message. This design simplifies control logic and makes agent behavior easier to analyze, but it also pushes more responsibility onto the Reflection process—especially when dealing with long-horizon, ambiguous, or emergent goals. Particularly, this design introduces the following features in native:

- **Single-Action Enforcement:** Complex tasks such as building structures, navigating around obstacles, or managing resources are never hardcoded as multi-step routines. Instead, Reflection breaks them down into discrete, explainable actions, preserving traceability and thus in native it allows interruption or revision at any time.
- **Adaptive Recovery:** When plans fail, obstacles appear, or assumptions break down, Reflection dynamically generates recovery strategies in the form of new messages—realigning agent behavior with its long-term objectives without external reprogramming.
- **Module Collaboration:** By serving as the bridge between Memory and Action, Reflection maintains the temporal coherence of behavior by generating "reminder messages, enabling smooth adaptation in dynamic environments and fostering long-term consistency in cognitive flow.

This approach increases the cognitive load placed on the underlying language model, since Reflection must infer not only what to do next, but also why, under uncertain or evolving conditions. Rather than relying on predefined rule sets or handcrafted task trees, this architecture leverages the LLM's general scene understanding and fuzzy reasoning capabilities to adaptively respond to context. While this may lead to less efficient execution in narrowly-defined tasks, it offers a powerful advantage in flexible, open-ended scenarios where agent behavior should appear human-like.

By emphasizing Reflection as the exclusive locus for planning, persistence, and correction, Minecraft AI-Python aligns closely with the goals of the Embodied Turing Test: we are not merely building agents that can complete tasks, but ones that behave believably—exhibiting autonomy, coherence, and spontaneity in ways that resemble human players more than deterministic scripts.

4.2.3 Coding Component: On-Demand Tool Creation

The Coding component introduces a dynamic and extensible layer of cognition, enabling AICs to generate and execute short Python scripts at runtime with the assistance of LLMs. This mechanism allows agents to move beyond their built-in capabilities by creating bespoke tools tailored to their immediate context, effectively functioning as "just-in-time" cognitive augmentations.

- **LLM-Powered Code Generation:** When a message indicates the need for computation, analysis, or environment querying beyond native capabilities, the system invokes a local or API-connected LLM to synthesize Python code. This process is guided by natural language task descriptions, interface constraints, and contextual cues from the agent's memory and current state.
- **Environment Compatibility:** To ensure robustness and safety, generated scripts conform to a standardized schema compatible with the Mineflayer's bot interface. This guarantees that the code can be executed in a controlled manner, and that results can be reliably passed back into the agent's memory and reflection cycle.
- **Sandboxed Execution and Context Injection:** All generated scripts are run inside a sandboxed Python environment with constrained resources and limited scope. This isolation protects the core agent system from runtime failures and ensures stability. In addition, contextual variables (e.g., recent memory entries, environmental state, or action results) are injected into the script environment to improve the relevance and correctness of LLM-generated code.

This architecture empowers AICs with a form of open-ended, task-specific reasoning. Rather than pre-encoding solutions for every possible situation, the AIC can dynamically generate scripts to process structured data, formulate hypotheses, query abstract relationships, or interact with external libraries and APIs—bridging symbolic logic with learned language understanding. This is especially valuable in research and educational settings, where agents may be tasked with solving unique, data-rich problems or integrating with domain-specific tools.

4.3 Messages as The Only Action Triggers

A core design choice in Minecraft AI-Python is the enforcement of a unified triggering mechanism for all actions: **only messages can initiate agent behavior**. This seemingly restrictive policy is introduced to address a key source of cognitive and architectural complexity—the proliferation of heterogeneous action triggers in traditional embodied agent systems.

In many Embodied AI frameworks, actions may be triggered in various ways: through external commands, internal goals, periodic loops, or reflective analysis. For example, a reflection process might directly trigger an action as a consequence of reasoning, or a periodic behavior (like "look around" or "patrol") might bypass cognitive processing entirely.

While such diversity in action triggers may appear flexible, it introduces significant challenges. It becomes difficult to trace the origin of an agent's behavior—whether it was triggered by reflection, periodic routines, or reactive inputs—making debugging and behavior analysis more complex. The lack of a unified trace mechanism leads to inconsistent logging and control, preventing clean pausing, revision, or reinterpretation of actions. Moreover, this heterogeneity breaks modularity by entangling planning, perception, and execution logic, ultimately reducing the system's interpretability and reusability.

To resolve this, Minecraft AI-Python adopts a **message-driven architecture**, where all decisions and downstream behaviors are derived from message processing. Each message serves as a declarative record of why the agent may choose to act, preserving both intention and context. In particular, the system defines four distinct types of messages, each corresponding to a different kind of cognitive or perceptual stimulus:

- **message** — These are sent from other players or entities to the AIC. They often reflect external communication or interactions, such as speech, gestures, or actions directed at the agent. The agent must interpret and respond appropriately.
- **reflection** — Generated by the agent itself, these messages represent self-driven cognition. They are typically triggered after action completion or at periodic intervals, prompting the agent to review memory, assess goal progress, and potentially generate a new plan or action. This type encapsulates autonomous behavior and metacognition.
- **status** — Issued by systems external and other players or entities to the AIC (e.g., environment monitors), status messages log situation changes not directed at the agent. Though not requiring a response, they are archived in memory to inform future reasoning and context-aware adaptation.
- **report** — Also self-generated (and sent to the chat), these messages document the outcome of prior actions or significant environmental observations. They function as structured updates to memory and assist in evaluating progress or triggering new reflections.

By consolidating all cognitive processing and behavioral execution into a unified message-driven pipeline, Minecraft AI-Python achieves a streamlined and principled control architecture that supports clarity, extensibility, and realism. This design enforces a single entry point for all AIC activity—regardless of source—ensuring consistent handling, easy integration, and traceability.

- **Unified Control Flow:** With all AIC decisions funneled through message handling, developers can easily trace the origin, processing, and outcome of any action. This sharply reduces the mental overhead required for debugging and facilitates step-by-step replay of agent behavior for educational or experimental analysis.
- **Interpretable Reasoning:** Every action taken by the agent is a reaction to a specific, loggable message—whether it originates from the environment, another entity, or the agent's own reflective thought. This message–response structure offers a natural audit trail for understanding and explaining agent decisions, which is particularly valuable in educational and research contexts.
- **Embodied Realism:** Modeling perception, self-reflection, and external communication all as messages closely resembles human cognitive behavior. Just as humans interpret the world through signals—whether visual cues, internal thoughts, or spoken words—the AIC operates through a unified semantic stream. This alignment strengthens the realism and coherence of the agent's behavior, supporting the overarching vision of the Embodied Turing Test, where believability and human-likeness of cognition are prioritized over brute task performance.
- **Cognitive Simplification at Scale:** In multi-agent scenarios or long-running deployments, enforcing message-only triggers eliminates hidden background processes and periodic logic checks, reducing system complexity. This makes it easier to simulate emergent behavior, coordinate inter-agent communication, or pause and resume tasks without breaking mental continuity.

In summary, Minecraft AI-Python's message-centric architecture promotes cognitive clarity, behavioral coherence, and system stability. By treating all triggers as messages—whether internally or externally generated—the framework achieves a high degree of modularity and interpretability, laying a strong foundation for scalable, research-ready agent development.

4.4 Reflection as The Core Cognition

In Minecraft AI-Python, cognition is not hardcoded into symbolic rules or procedural planning pipelines—instead, it emerges from the interplay between memory and reflection. At the heart of this system lies the principle that awareness and response must evolve through situated experience, not pre-defined logic.

The Memory module serves as the structural foundation of cognition. Every action, observation, message, and status change is recorded into memory, forming a rich temporal trace of the agent's lived experience. Over time, these records are not merely stored but synthesized into topic-based summaries, enabling the agent to retrieve relevant context and maintain thematic continuity. These summaries act as the data form of cognition—representing how thoughts and perceptions evolve based on history.

While memory anchors long- and short-term knowledge, its function is primarily passive: it reflects what has happened. The active interpretation of memory—what the agent understands about its current situation, what it should do next, and how it evaluates outcomes—is handled by the Reflection module.

In Minecraft AI-Python, the Reflection mechanism serves as the central cognitive engine that actively transforms memory into intelligent, context-aware behavior. Unlike passive memory storage, Reflection continuously interprets the evolving state of the world and the agent's internal context to drive purposeful decision-making. It embodies the AIC's metacognitive capacity—the ability to think about its own actions, adjust strategies, and maintain behavioral coherence over time.

Every action executed by an AIC is not the end of a process, but the beginning of reflection. The action's outcome—success, failure, or partial result—is logged into memory and immediately triggers the Reflection cycle. Through this process, the agent evaluates not just **what** happened, but **why** it happened, and **what to do next**. Depending on the evaluation, Reflection may:

- **Assess Success or Failure:** Reflection begins by evaluating whether the previous action achieved its intended outcome. This evaluation is not limited to simple success/failure flags—it considers contextual clues such as environmental changes, item state updates, or new observations in memory. The agent learns to recognize patterns of success and detect silent failures, such as misaligned block placements or incomplete pathfinding, enabling more nuanced awareness of interaction quality.
- **Guide Adaptation or Retrying:** When an action fails or leads to suboptimal results, Reflection plays a pivotal role in determining the appropriate course correction. It may suggest a direct retry (e.g., if the failure was due to temporary obstruction), adjust action parameters (e.g., choose a different tool or path), or invoke an alternative strategy. This ability to dynamically recover from partial or full failure is key to achieving robustness in unpredictable game environments.
- **Formulate Follow-Up Objectives:** Rather than treating each action as an isolated response, Reflection integrates short-term outcomes into a broader narrative context. It formulates follow-up objectives that logically extend the current plan, ensuring continuity of intent. For example, after successfully crafting a tool, the agent may automatically proceed to equip and use it, reflecting an understanding of both the task's substructure and its final goal.
- **Update Memory and Task State:** Reflection continuously injects refined observations, derived inferences, and updated task statuses into memory. This includes not just new facts (e.g., "furnace built at (x, y, z)"), but also synthesized summaries (e.g., "resource gathering completed") that contribute to the AIC's situational awareness. Such memory updates are essential for maintaining an accurate mental model of the world, and for enabling downstream reasoning across future Reflection cycles.

Through this unified cycle of evaluation and response, Reflection allows the agent to remain robust in dynamic environments, resilient to unexpected outcomes, and capable of improvising within the logic of its past experience. It is this mechanism that transforms isolated actions into a coherent, embodied stream of behavior—essential for creating agents that not only act, but understand, adapt, and evolve.

In the pursuit of the Embodied Turing Test, where agents must behave indistinguishably from human players, Reflection is not an auxiliary function—it is the essence of cognition itself.

Unlike traditional planners that construct rigid goal sequences, Reflection enables incremental, adaptive, and embodied cognition. It allows the agent to remain grounded in its environment, react to surprises, recover from errors, and generate continuous, human-like behaviors even in open-ended worlds. This process tightly couples situational awareness with reactive reasoning, forming a complete loop of perception, memory, and action.

4.5 Advantages and Limitations

The Minecraft AI-Python implementation adopts a streamlined and modular design centered around message-driven cognition and action, which brings several notable advantages, particularly in educational and research contexts:

- **Simplified Action Trigger Logic:** By unifying all AIC actions under a single message-processing mechanism, the system avoids the complexity of managing multiple asynchronous triggers. This makes the behavior pipeline highly transparent and easy to trace, which is ideal for debugging, comprehension, and controlled experimentation.
- **Clear Modularity and Separation of Concerns:** Each subsystem—perception, reflection, planning, and action—communicates through well-defined message interfaces. This clean separation encourages extensibility and supports focused exploration of individual cognitive components, such as testing alternative reflection strategies or memory management policies.
- **Designed for Educational Use and Cognitive Research:** The reduced complexity and high interpretability of the system make Minecraft AI-Python well-suited for classrooms, AI labs, and cognitive modeling experiments. It allows students and researchers to clearly observe how messages drive agent behavior, and to manipulate components without navigating low-level implementation noise.

Despite its clarity and educational value, the current design of Minecraft AI-Python has trade-offs that may limit its applicability in large-scale or high-performance scenarios:

- **Limited Parallelism and Temporal Granularity:** Unlike the JavaScript-based Minecraft AI implementation, which supports multiple asynchronous triggers (e.g., periodic timers, background routines, sensory polling), the Python version relies solely on discrete message events. As a result, handling time-sensitive or concurrent tasks (e.g., real-time combat reactions, multitasking during exploration) may be less efficient or responsive.
- **Potential Bottlenecks in Dynamic Environments:** In rapidly changing or heavily populated environments, where multiple events must be processed in short intervals, the single-threaded message handling model may struggle to keep pace, leading to delayed reactions or missed opportunities.
- **Lower Suitability for Production-Grade Automation:** While ideal for prototyping and research, Minecraft AI-Python's architecture may not be optimal for fully autonomous AICs in live servers or long-duration tasks without further optimization in scheduling, concurrency, and environmental awareness mechanisms.

In summary, Minecraft AI-Python prioritizes simplicity, interpretability, and modularity over execution complexity. It serves as a robust foundation for understanding and experimenting with Embodied AI behaviors, while acknowledging that more intricate timing and coordination features are better served by alternative or hybrid architectures in practical deployments.

Chapter 5

The Embodied Turing Test

5.1 Background: From Dialogue to Embodiment

The **Turing Test** [Wikipedia contributors, 2025e], originally proposed by Alan Turing in his work "*Computing Machinery and Intelligence*" [Turing, 1950], was intended as a practical criterion for evaluating machine intelligence. In its classic form, the test involves a human evaluator engaging in natural language conversations with both a human and a machine, typically through a text-only interface. If the evaluator cannot reliably distinguish the machine from the human, the machine is said to have passed the test and thus demonstrated intelligent behavior indistinguishable from that of a human.

Over the decades, the Turing Test has served as a philosophical and engineering benchmark for artificial intelligence. However, it has also been the subject of ongoing critique. One major limitation is its exclusive focus on "disembodied language use"—treating intelligence as purely linguistic and ignoring the broader, physically grounded aspects of cognition. However, more and more researchers and developers support that true human-level intelligence emerges not only from language use but from embodied experience [Pfeifer and Scheier, 2001]. Children learn concepts such as "soft", "heavy", or "dangerous" not by reading definitions, but by interacting with the physical world. They learn to plan, collaborate, and reason by engaging with objects, environments, and other agents over time.

On the other hand, the emergence of large language models (LLMs) like GPT-4, Claude, and Gemini has reignited debates around the Turing Test. These models can produce remarkably coherent, fluent, and even context-sensitive dialogue—often outperforming humans in benchmarks of factual recall, summarization, and reasoning. In some constrained domains, they can successfully "fool" human evaluators, raising the question: have LLMs passed the Turing Test?

Superficially, the answer may seem to be yes. While there is ongoing debate [Jones and Bergen, 2024a], it is undeniable that these models now outperform human participants in a wide range of linguistic tasks [Mittelstädt et al., 2024, Sejnowski, 2023, Jones and Bergen, 2024b, Shultz et al., 2024]. In structured benchmarks involving summarization, translation, factual recall, and even logical reasoning, LLMs frequently achieve or surpass human-level results. In many interactive settings, they are capable of sustaining long, coherent, and context-sensitive conversations that convincingly emulate human dialogue.

As a result, it has become increasingly evident that the original Turing Test—focused solely on text-based conversation—no longer provides a sufficient or discriminative benchmark for evaluating general intelligence. Rather than rendering the Turing Test obsolete, however, the remarkable capabilities of modern LLMs have inspired researchers to broaden its scope. There is a growing call for a more comprehensive and rigorous interpretation—one that incorporates physical embodiment, situational context, and behavioral consistency over time.

This shift has culminated in the formulation of the **Embodied Turing Test** [Ortiz Jr, 2016], which provides a more comprehensive and realistic benchmark for artificial intelligence [Jiang et al., 2025], one

grounded not only in what agents *say*, but in how they *perceive*, *plan*, and *act* within a shared environment, and thus it presents multiple interrelated research challenges:

- **Perception and Sensorimotor Skills.** Agents must develop robust sensorimotor control: interpreting vision, sound, and spatial layout; performing precise actions; and understanding affordances and constraints in the environment.
- **Adaptability and Embodied Learning.** The Embodied Turing Test assumes agents learn through experience. This means generalizing to new tasks, recovering from failure, and building cumulative knowledge from prior interaction—hallmarks of biological learning.
- **Social Intelligence and Collaboration.** Human-level intelligence is not just individual—it is also social. Embodied agents must model other agents' goals, form commitments, and collaborate effectively.
- **Trust and Transparency.** To succeed in human-agent collaboration, AI agents must be interpretable and trustworthy. The ability to explain decisions, act consistently with intent, and be perceived as reliable is increasingly viewed as part of passing the Embodied Turing Test.

A central challenge in operationalizing the Embodied Turing Test lies in defining what it means for an agent to "pass" the test. Unlike the classical Turing Test, which involves a discrete, turn-based dialogue judged by a human interrogator, the Embodied Turing Test evaluates a broader spectrum of behaviors unfolding continuously in time and space. There is no singular conversation to analyze; instead, assessment must be distributed across multiple modalities, including perception, reasoning, action, and social interaction.

In recent years, virtual platforms like Minecraft [Wikipedia contributors, 2025b] have emerged as promising testbeds for the Embodied Turing Test, as they offer a compelling balance of complexity and control, enabling researchers to study agents in rich, interactive environments without the logistical constraints of physical robotics. By bridging the gap between tightly constrained laboratory setups and the chaotic complexity of the real world, these sandbox environments make it feasible to evaluate embodied agents at scale.

5.2 Criteria for Evaluation

Evaluating whether an AI agent has "passed" the Embodied Turing Test is fundamentally more complex than its classical counterpart. Unlike the text-only Turing Test, the Embodied Turing Test requires agents to demonstrate integrated intelligence through perception, reasoning, action, and interaction over time within a shared environment. This section outlines the key dimensions and metrics for Embodied Turing Test evaluation, along with task types and scenario examples.

Successful embodiment requires competence in multiple interacting capacities. We propose the following dimensions as foundational for the assessment:

- **Task Performance and Adaptability.** A fundamental evaluation criterion for embodied agents is their ability to complete tasks effectively and efficiently across varying levels of complexity. Success is not only measured by task completion rate, but also by how efficiently the agent achieves its goals—such as minimizing the number of steps or optimizing time to completion. Equally important is the agent's adaptability: can it generalize to novel environments or handle task variations it has not seen during training? Robustness under noisy or unpredictable conditions, such as environmental changes or interruptions, also reflects real-world competence and is essential to meeting the Embodied Turing Test's demands.

- **Behavioral Realism and Believability.** Beyond task completion, the agent must behave in ways that are socially and contextually appropriate from a human observer's perspective. This includes plausibility of action selection within the current environment, spontaneous reactions that reflect situational awareness, and social behaviors such as politeness, cooperation, or deference when appropriate. Evaluators judge whether the agent appears lifelike—avoiding obviously scripted behavior and instead showing emergent, naturalistic responses to complex stimuli. Believability is especially critical in scenarios involving human-agent interaction.
- **Consistency and Memory Coherence.** True intelligence includes the ability to maintain coherent behavior over time. This requires a functioning memory system that allows the agent to recall prior goals, conversations, or decisions across long temporal spans. Agents should demonstrate continuity in their long-term objectives, consistently follow through on promises or plans, and preserve their personality traits or social roles throughout interactions. For example, if an agent commits to helping with a construction project on one day, it should remember and act on that commitment later—demonstrating memory-informed behavior.
- **Collaboration and Social Awareness.** In both cooperative tasks and social environments, agents must demonstrate awareness of others and the ability to collaborate effectively. This includes aligning goals with teammates, negotiating roles or strategies, and coordinating action without redundancy. Agents should exhibit turn-taking behavior, assist or lead as needed based on social context, and adapt to group dynamics. Furthermore, emotional and role-based understanding—such as recognizing another agent's frustration or taking initiative in group settings—contributes to the perception of intelligence and social competence, and is a core marker of success in the Embodied Turing Test.

5.2.1 Benchmark Task Types

To rigorously assess whether an AI agent can pass the Embodied Turing Test, it must be evaluated across a diverse and representative set of embodied tasks. These tasks should stress different dimensions of intelligence, including perception, planning, memory, social interaction, and physical coordination. Importantly, the evaluation framework should apply to both virtual and physical embodiments, as well as hybrid environments (e.g., digital twins or augmented reality). Representative task categories include:

- **Conversational Competence in Contextual Interaction:** Unlike the classical Turing Test, which evaluates conversation in isolation, the Embodied Turing Test requires agents to integrate language use with situated behavior. This involves not just generating fluent responses, but selecting when to speak, whom to address, and what kind of dialogue act is contextually appropriate (e.g., asking fo
- **Survival and self-maintenance tasks:** Agents are required to autonomously manage resources, adapt to environmental changes, and preserve their functional status. These tasks may include maintaining energy levels, avoiding hazards, and responding to emergent threats. In robotics settings, this could involve recharging and obstacle avoidance; in virtual settings, it may involve foraging and health tracking.
- **Navigation and mobility tasks:** Embodied agents must demonstrate the ability to move purposefully through space, dealing with obstacles, landmarks, and changing terrain. Evaluation may involve target reaching, path optimization, or context-sensitive navigation. In physical systems, real-world noise and sensor errors further challenge the robustness of such skills.
- **Interaction and Environmental Manipulation Tasks:** These tasks evaluate an agent's ability to perceive, affect, and adapt to its surroundings in goal-directed ways. Depending on the embodiment, this may involve physically rearranging objects, issuing commands to simulated environments, or triggering in-world dynamics through indirect means. Success in such tasks reflects the agent's understanding of causal relationships, affordances, and action consequences in both structured and open-ended settings.

- **Social and interactive tasks:** These require agents to collaborate or communicate with humans or other agents to achieve shared objectives. Success depends on understanding social norms, modeling others' goals and knowledge, and adapting behavior in response to social feedback. These tasks are critical in both physical human-robot interaction (HRI) and virtual multi-agent simulations.
- **Memory-dependent and long-horizon tasks:** Agents must demonstrate continuity across time, such as resuming unfinished plans, recalling past dialogues or events, and adapting based on accumulated experience. This includes both episodic memory (what happened before) and semantic memory (what the agent has learned). These tasks are especially important for evaluating cognitive coherence, planning over delayed rewards, and agent identity persistence.
- **Goal generalization and transfer tasks:** Here, agents face novel variations of previously encountered tasks or environments, without explicit retraining. Success implies the ability to abstract learned skills, recompose known actions, and generalize to unseen but related challenges. This dimension is crucial for demonstrating robust intelligence beyond rote memorization or narrow optimization.

5.2.2 Example Scenarios

To meaningfully evaluate an agent's performance across the Embodied Turing Test dimensions, testing should involve dynamic, context-rich scenarios that activate multiple cognitive and behavioral faculties. Below are representative examples aligned with the task types described above:

- **Autonomous Survival Scenario.** In a resource-constrained open world (e.g., simulated or robotic), the agent must independently manage energy sources, avoid hazardous regions (like lava pits or stairs), and make trade-offs between foraging and safety. Environmental unpredictability requires on-the-fly adaptability.
- **Navigation in an Unknown Structure.** The agent is placed in a procedurally generated environment (or physical maze) and asked to find a target location. Tasks include identifying landmarks, re-routing after a blocked path, and explaining its strategy when asked. Success indicates spatial understanding and robustness to disruption.
- **Interactive Object Rearrangement.** Given a disordered room or workshop, the agent is tasked with organizing objects based on verbal or visual instructions. This tests its ability to interpret multimodal cues, predict object affordances, and manipulate elements within physical or simulated constraints.
- **Team-based Quest Execution.** In a multi-agent scenario, the agent joins other AIs and humans to complete a mission (e.g., defending a village or gathering rare materials). Evaluation focuses on negotiation, division of labor, coordination timing, and role adherence. Unexpected events like attack or betrayal can test flexibility.
- **Memory-Persistent Storyline.** Spread across several sessions, the agent participates in a narrative arc (e.g., repairing a broken town economy), where remembering past decisions and relationships is key. It must resume unfinished tasks, recall prior dialogues, and reference its own learning.

5.2.3 Toward Standardized Evaluation Frameworks

While the above approaches illustrate promising directions, the field of embodied intelligence still lacks a unified and widely accepted evaluation standard that balances task specificity with open-ended behavioral flexibility. This absence hinders progress in comparing different systems, measuring progress over time, and facilitating collaborative research. To address this, future work should prioritize the development of standardized benchmarking ecosystems that encompass the following dimensions:

- **Unified evaluation suites:** Benchmarks should combine both automatic metrics (e.g., task success rate, path efficiency, dialogue coherence scores) and human-in-the-loop evaluation (e.g., believability ratings, social appropriateness, trustworthiness). This dual-layered approach captures both objective and subjective qualities of intelligent behavior.
- **Cross-platform compatibility:** Evaluation frameworks must generalize across platforms such as Minecraft, Malmo, MineLand, and potentially physical robotics environments. Abstract task definitions, shared protocols, and standardized logging formats are needed to ensure replicability and fair cross-system comparison.
- **Replay and audit tools:** Effective benchmarking requires the ability to trace and analyze agent behavior over time. Replay systems should allow step-by-step inspection of perceptual inputs, memory accesses, and decision trajectories. This supports debugging, human assessment, and transparency in model behavior.
- **Agent-agnostic protocols:** Testing environments should avoid hardcoding assumptions about agent architecture (e.g., LLM-driven, symbolic planner, hybrid controller). Instead, they should define tasks, roles, and success criteria in abstract terms, enabling fair and inclusive evaluation across diverse models and modalities.

Establishing such standards is essential for scaling the Embodied Turing Test beyond one-off demos or narrowly scoped experiments. As we move toward rigorous, reproducible evaluation of embodied intelligence at scale, virtual environments like **Minecraft** offer an ideal balance between realism and controllability. Their open-ended design, modifiability, and wide adoption position them as a foundational platform for next-generation testing of AI agents that not only think and speak—but also perceive, plan, act, and socially interact like humans.

5.3 Challenges for Virtual AI Characters

Virtual platforms—most notably Minecraft—have emerged as powerful and practical testbeds for developing and evaluating embodied intelligence. Their flexibility, scalability, and modifiability offer distinct advantages over purely physical environments, particularly in the context of the Embodied Turing Test. Key strengths include:

- **A richly embodied and physics-aware 3D world.** Minecraft features spatially grounded, voxel-based mechanics that simulate physics-like constraints. AICs must navigate terrain, manipulate objects, craft tools, build structures, and respond to dynamic events—all of which require the integration of perception, planning, and action within a temporally extended sensorimotor loop.
- **Open-ended task domains and programmable environments.** Minecraft's scripting and modding capabilities allow researchers to design custom tasks and rulesets, ranging from structured quests (e.g., "build a bridge across a ravine") to open-ended objectives (e.g., "collaborate with villagers to improve the town"). This supports fine-grained probing of specific cognitive capabilities such as memory use, multi-step reasoning, or social alignment.
- **Scalable and reproducible experimentation.** Unlike physical robots, AICs can be instantiated at scale, reset with ease, and evaluated under identical conditions. Large-scale ablation studies, benchmarking across architectures, and longitudinal analysis over thousands of simulated hours become feasible, enabling data-rich, reproducible comparisons across agent designs.
- **Support for multi-agent ecosystems.** Minecraft inherently supports multiple AICs interacting in the same world, making it a fertile ground for studying emergent behavior, teamwork, negotiation, and role-based collaboration. The evaluation of Embodied AI thus extends naturally from single-agent

competence to socially intelligent group behavior—essential for passing more advanced versions of the Embodied Turing Test.

While virtual environments like Minecraft offer powerful infrastructure for evaluating Embodied AI, designing AICs capable of passing the Embodied Turing Test introduces three core challenges. Each reflects a fundamental aspect of intelligent behavior: self-consistency, social coordination, and temporal awareness. Addressing these challenges is essential for building agents that not only complete tasks but do so in a human-like, believable manner.

5.3.1 Coherence Within a Single AIC

To appear intelligent and human-like, an AIC must maintain internal coherence across all its actions, dialogues, and decisions. This is not merely about correctness, but about exhibiting believable continuity—where each decision appears informed by a coherent personality, memory of the past, and awareness of current context. Core aspects of such coherence include:

- **Behavioral consistency:** The AIC should manifest stable personality traits, values, and long-term preferences across time. For example, a friendly AIC should not suddenly become hostile without reason; similarly, an AIC committed to pacifism should not initiate combat. Behavioral consistency also enhances the user's trust and identification with the character, especially in multi-day or narrative-driven interactions.
- **Context-aware responses:** The AIC must interpret its surroundings—including environmental changes, social cues, and conversation history—and respond in a way that fits the current situation. For instance, if it is raining and the player says "Let's find shelter", the agent should not propose a picnic. This requires maintaining a window of recent context (short-term memory) and being able to resolve references, implicit cues, and task state.
- **Goal alignment:** AICs often operate under multiple concurrent goals and interruptions. A coherent AIC must juggle short-term tasks (e.g., gathering wood) while still tracking and prioritizing long-term objectives (e.g., building a tower). Even after being distracted—say, by helping another character—it should resume its previous plan smoothly, rather than forgetting or starting anew. This kind of goal alignment is a hallmark of intelligent behavior.

Achieving such coherence is technically challenging. It demands tight integration between *episodic memory*, which stores past experiences; *reflective memory*, which summarizes goals and conclusions; and a *planning module*, which tracks task structure and dependencies. In practice, these modules must coordinate under real-world constraints: limited context windows for LLMs, asynchronous environments, and noisy or incomplete perception. Effective AICs often rely on external memory stores, structured representations of goals, and periodic self-reflection routines to maintain internal coherence over time.

Ultimately, an AIC that cannot explain its own behavior, track its goals, or maintain a stable personality is unlikely to be judged as truly intelligent—no matter how fluent its language output may be. Coherence is thus a foundational criterion for passing the Embodied Turing Test.

5.3.2 Collaboration Across Multiple AICs

Human-like intelligence extends beyond individual competence—it encompasses the ability to understand, coordinate with, and adapt to others in shared environments. In multi-agent scenarios, an AIC is not merely solving its own task, but must function as part of a larger social and operational system. Successful collaboration hinges on a combination of perception, inference, memory alignment, and communication. Key requirements include:

- **Modeling other AICs' intentions and knowledge:** An AIC should be able to infer what its collaborators know, want, and plan to do. This requires maintaining a dynamic model of each teammate's goals, progress, and perspective—akin to theory of mind in human cognition. For example, if one AIC notices a teammate struggling with resource collection, it may autonomously assist without being explicitly asked.
- **Role negotiation and task decomposition:** Effective collaboration often involves assigning responsibilities and dividing complex goals into subtasks. AICs must be able to negotiate roles dynamically, especially in open-ended environments like Minecraft where task structure is emergent. They should recognize when to lead (e.g., initiating group strategies), follow (e.g., supporting a builder), or independently pursue complementary actions.
- **Adapting to group dynamics and social context:** Beyond rational planning, AICs should display socially aligned behavior—such as waiting their turn, coordinating movement in shared space, or modulating communication tone based on familiarity or hierarchy. These behaviors require fine-grained awareness of social context and implicit norms, especially when interacting with human players.

Implementing these capabilities presents multiple design challenges. First, collaboration implies *shared situational awareness*, which depends on synchronized state representations across agents. Without consistent world models and updates, miscoordination and contradictory actions arise. This motivates the design of **shared memory structures**, which allow agents to query collective knowledge, update plans collaboratively, and track dependencies between actions.

Second, scalable multi-agent systems require well-defined **communication protocols**, balancing explicit messaging (e.g., via chat) with implicit signaling (e.g., through movement or construction behavior). Coordination mechanisms—such as blackboard architectures, publish-subscribe message passing, or learned communication policies—are active research topics.

Finally, social reasoning in multi-agent systems opens new questions about fairness, empathy, and adaptation. For instance, AICs may need to adjust their tone when addressing new versus familiar players, or prioritize helping weaker AICs over optimizing their own success.

Overall, enabling believable and effective AIC collaboration remains a frontier of Embodied AI. Success will likely require the integration of multi-agent planning, natural language dialogue, distributed memory access, and emergent social modeling—making this one of the most complex and rewarding domains within the Embodied Turing Test paradigm.

5.3.3 Maintaining Accurate and Adaptive Memory

Among all capabilities required for passing the Embodied Turing Test, the ability to remember—both reliably and meaningfully—stands as one of the most foundational. Unlike stateless or reactive systems, AICs must maintain internal representations of the past to inform decisions in the present and plan for the future. Memory is not simply an auxiliary feature; it is essential for coherence, context-awareness, learning, and adaptability across time. A robust memory system must support multiple functions and timescales, including:

- **Short-term context tracking:** For immediate situational awareness, the AIC must track recent perceptions, conversation turns, and task steps. This includes maintaining dialog state, monitoring recently observed changes in the environment, and remembering its most recent intentions or interrupted subtasks. For example, if a user issues a multi-step instruction, the agent must remember the command's context even after being momentarily interrupted.
- **Long-term continuity:** Embodied agents are expected to behave as persistent characters across multiple interactions or sessions. This demands the recall of prior goals, user preferences, spatial landmarks, social relationships, and past experiences. An agent helping a player build a base, for

instance, should remember the location, architectural style, and unfinished elements across days or weeks of play.

- **Dynamic updates and relevance filtering:** As memory grows, not all experiences remain equally useful. AICs must learn to revise outdated beliefs, summarize redundant experiences, and prioritize information relevant to ongoing goals. This calls for adaptive forgetting, memory compression, and mechanisms for identifying salient updates—ensuring that memory remains useful without overwhelming the system with irrelevant detail.

In practice, this necessitates a hybrid memory architecture that combines:

- **Episodic memory:** Time-stamped logs of raw interactions and experiences, useful for reconstruction and accountability.
- **Semantic memory:** Generalized knowledge derived from repeated patterns or summarized content (e.g., "The player prefers stone over wood for building").
- **Reflective memory:** Meta-cognitive insights, such as lessons learned, failure modes, or evolving strategies—used to guide high-level planning and adaptation.

Many modern implementations rely on RAG pipelines, where relevant memory entries are retrieved and injected into the prompt of a LLM at inference time. These are often backed by external memory graphs, vector stores, or document caches that support semantic similarity retrieval. However, managing retrieval relevance, minimizing prompt bloat, and ensuring the interpretability of retrieved memory remain open technical challenges.

Moreover, collaborative settings pose additional requirements: memory must be partially *shared*, *referenced*, or even *negotiated* between agents. In such settings, designing memory synchronization protocols—whether through explicit communication, broadcasted updates, or shared memory services—is essential to support cooperative coherence.

In summary, memory in the Embodied Turing Test is not a passive record of the past—it is an active, structured, and evolving knowledge system that empowers agents to act with continuity, context, and foresight. Designing such systems at scale remains one of the most complex and central problems in building believable AI characters.

5.4 Discussions

The Embodied Turing Test marks a significant evolution in the way we evaluate artificial intelligence. It shifts the focus from mere linguistic imitation—central to the classic Turing Test—to a more comprehensive standard that encompasses perception, memory, real-time action, and adaptive social interaction. In this expanded framework, intelligence is no longer defined by conversation alone, but by an agent's ability to *exist, behave, and evolve* meaningfully within a dynamic environment.

Virtual environments, particularly richly interactive platforms like Minecraft, provide an unprecedented opportunity to realize and study this vision. Despite these advantages, passing the Embodied Turing Test remains a grand challenge. Designing AICs that behave believably and coherently across time and context requires tightly integrated cognitive systems—combining language, perception, memory, planning, and social awareness.

Looking ahead, we envision a new generation of AICs that exhibit truly human-like cognitive and social abilities. These AICs will engage in context-sensitive and emotionally aware dialogue, expressing and interpreting intentions, roles, and affect with nuance. They will demonstrate adaptive problem-solving grounded in situational awareness and capable of decomposing complex goals creatively. In multi-agent or collaborative settings, they will negotiate and coordinate effectively with both human users and other AICs, contributing meaningfully to shared tasks. Crucially, their behavior will be shaped by memory-informed

planning that spans across sessions—allowing for persistent identities, long-term objectives, and the capacity to evolve over time in response to new experiences.

As such, the Embodied Turing Test is not just a milestone in AI benchmarking—it is a design philosophy. It challenges us to move beyond narrow metrics of intelligence and toward the creation of AICs that are contextually grounded, socially interactive, and behaviorally coherent across time. Virtual environments like Minecraft, with their balance of open-ended interaction and engineered control, stand as the most promising arenas for prototyping these truly embodied minds.

Chapter 6

Coherence within Single AI Character

6.1 Background: What is Coherence and Why It Matters

In the design of AI Characters (AICs), "coherence" denotes the degree to which a single agent maintains continuity, stability, and self-consistency across both the **temporal** and **multi-modal** dimensions, e.g. the multi-modal loop comprises language, physical action, affective display, and perception–action coupling. Unlike traditional script-driven or finite-state non-player characters (NPCs), an LLM-based AIC improvises its language and behavior at every time step. Such improvisation frequently results in identity drift, motivational rupture, or behavior that contradicts prior commitments [Backlund and Petersson, 2025].

Humans construct mental models of artificial agents through repeated interaction [Dennett, 1987]. Persistent contradictions erode that model, diminishing immersion and trust. The *Generative Agents* study [Park et al., 2023] reported that human participants considered the simulated townsfolk "believable" only when the AIC pipeline combined dependable **memory retrieval** with periodic **reflection-driven summarisation**. Similarly, evaluations of the embodied Minecraft agents STEVE-1 [Lifshitz et al., 2023] and Voyager [Wang et al., 2023a] demonstrate that the absence of persistent task context or policy coherence significantly reduces task completion rates.

Drawing on cognitive science and multi-modal agent practice, coherence can be decomposed into four complementary dimensions:

- **Narrative Coherence:** Linguistic outputs and episodic memories must reinforce a stable backstory, personality profile, and worldview. Coherence at this level is analyzed in narratology as the construction of a narrative identity that links past, present, and anticipated future selves [Herman, 2004]. For AICs, this means that dialogue, internal monologue, and memory recall all reference the same canonical facts (e.g., birthplace, core values) and that new experiences are integrated as extensions rather than contradictions. Script theory further suggests that agents rely on stereotyped situation models to decide what happens next, providing a scaffold for consistent storytelling [Schank and Abelson, 2013].

- **Behavioral Coherence:** The agent's overt behaviour—locomotion, tool use, block placement, and timing—must flow logically from its stated goals and the affordances of the current environment. Observers should be able to explain each motor sequence in terms of an articulated intention. For example, if the agent declares "I will harvest oak logs to extend the farmhouse", it should promptly walk to the nearest oak tree, wield an axe, and collect logs [AL et al., 2024]. Incoherence becomes apparent when speech and action diverge: an agent that announces "I am building a watch-tower now" but then remains idle or wanders aimlessly offers no perceptible link between intention and execution. Event-Segmentation Theory frames such action streams as chains of goal-directed events whose boundaries must be respected for behaviour to remain intelligible [Zacks et al., 2007].

- **Temporal Coherence:** The AIC must weave the arrow of time into its cognition, displaying both episodic recall of what has already happened and prospective simulation of what might happen next. Tulving's notion of *chronesthesia* highlights that a subjective sense of past-and-future is foundational for organising goal hierarchies and maintaining a personal narrative [Tulving, 1985]. From a cognitive-architectural standpoint, temporal coherence is therefore inseparable from the agent's memory system:
  - **Past as stable context.** Episodic traces—"I traded sixteen emeralds with the villager yesterday"—must be compressed, indexed, and retrievable on demand; otherwise the agent will repeat or contradict earlier choices (e.g., attempting to buy the same map twice at full price). *Reflective Memory Management*(RMM) modules such as those proposed by [Tan et al., 2025] provide one practical implementation, periodically summarising long interaction logs while preserving cues for high-salience events.
  - **Future as simulated trajectory.** Forward-looking planners rely on stored causal models to project action consequences, e.g. "If I smelt the iron overnight, I can craft rails at dawn". Without such simulation, the agent acts myopically, treating each tick as a fresh world state.
  - **Bidirectional coupling.** Memory retrieval dynamically alters future projections, while anticipated outcomes determine which memories are worth retaining—mirroring theories of working–long-term memory interplay in classical models of cognition [Newell, 1994].

  A breakdown in this coupling is immediately perceptible: an agent that promises "I will deliver the iron ingots tomorrow" yet fails to mention or pursue that goal the next morning betrays temporal incoherence, eroding the illusion of a continuous personality.

- **Cognitive Coherence:** The internal belief network and reasoning procedures must minimise contradiction and update rationally in light of new evidence, maintaining what Newell terms a "unified system" [Newell, 1994]. This entails explicit belief-revision policies, truth-maintenance systems, or Bayesian message passing so that newly acquired facts propagate consistently, thereby preserving Bratman's notion of stable intentions over time [Bratman, 1987]. Suppose an agent concludes, "The village well is poisoned", and therefore schedules time to build a purification device. Later, direct water-quality sensing reveals the well is actually safe. A cognitively coherent agent will retract the original belief, cancel the purification plan, and perhaps redirect effort toward farming. If the agent instead clings to the outdated belief—or worse, simultaneously warns players about the "poison" while drinking from the well itself—it exposes a contradiction that breaks cognitive coherence and damages user trust.

Techniques such as *Reflective Memory Management* for periodic summarisation and consolidation [Tan et al., 2025], together with *Retrieval-Augmented Generation* (RAG) for on-demand fact injection [Lewis et al., 2020], have markedly improved an AIC's long-term consistency. Yet significant obstacles remain: large language models are typically accessed through stateless APIs that forget prior turns; system prompts drift and cannot indefinitely anchor persona or goals; language, action, and affect must be multi-modally aligned to prevent an agent "saying one thing and doing another"; and there is still no widely accepted long-horizon benchmark for measuring coherence over hours or days, although efforts such as Vending-Bench point the way [Backlund and Petersson, 2025]. These gaps motivate four intertwined research directions that the rest of this white-paper will explore in depth:

- **Persistent Memory Architectures:** How hierarchical storage, reflective summarisation, and RAG can give an LLM a usable autobiographical record;
- **Persona and Goal Stabilisation:** How identity kernels, self-reflection loops, and plan stacks can curb prompt drift;
- **Cross-Modal Grounding:** How shared representations connect dialogue, locomotion, and affect into a single behaviour stream; and

- **Long-Horizon Evaluation:** How to design tasks and metrics that reveal coherence (or its absence) across extended play.

Together, these research threads sketch a path toward AICs whose stories, actions, and beliefs remain intelligible long after the first conversation ends.

6.2 Persistent Memory Architectures for Coherent Agents

AIC coherence degrades rapidly when an LLM forgets why it acted, contradicts earlier claims, or repeats completed tasks. Persistent memory architectures tackle this fragility by endowing the AIC with a usable autobiographical record that survives context window resets, thereby aligning future reasoning with past commitments. In this survey we focus on systems that explicitly target behavioural or narrative coherence, and general knowledge-retrieval techniques are covered only insofar as they support that goal.

6.2.1 Neural–Symbolic Precursors

Early neural–symbolic hybrids—Memory Networks [Weston et al., 2014], Neural Turing Machines [Graves et al., 2014], and the Differentiable Neural Computer [Graves et al., 2016]— demonstrated that an external read–write buffer allows a model to solve tasks requiring multi-step reasoning and recall. For example, in the bAbI two-support-fact task [Weston et al., 2015], a question such as "where is the apple?" must be answered by combining information from the earlier statements:

> "Mary went to the garden. Mary picked up the apple. Mary went to the bedroom."

A memory network writes each sentence into a slot and learns an attention key that first retrieves the sentence mentioning "apple", then hops to the sentence that places Mary in the "bedroom", finally returning "bedroom" as the answer. Without the external buffer—and the learned keys that link slots across hops—the model would have to compress the entire story into its hidden state, a strategy that quickly breaks down for longer narratives and destroys behavioural coherence in more complex agents.

Although these systems were toy-scale, they established a design pattern that endures today: *(i)* keep a separate scratch-pad outside the core language model, and *(ii)* learn retrieval keys that bridge the two.

[Sukhbaatar et al., 2015] introduced End-to-End Memory Networks, showing that a single soft-addressable memory matrix can solve the 20 bAbI story-understanding tasks with near-perfect accuracy—evidence that explicit memory markedly boosts the **narrative coherence**. Key–Value Memory Networks [Miller et al., 2016] separated address keys from content values, reducing spurious overwrites and improving multi-hop QA, while Dynamic Memory Networks [Kumar et al., 2016] added a gated episodic module that updates only when the input story reaches an event boundary, mirroring the "chunking" of human event perception.

As language tasks grew, the transformer-based models adopted similar ideas. *Transformer-XL* [Dai et al., 2019] caches hidden states from previous segments to extend the effective context window beyond 1000 tokens, cutting perplexity by 18 $\sim$ 24% on WikiText-103 [Merity et al., 2016] and creating more **temporal coherence** in generated stories. Compressive Transformers [Rae et al., 2019] push this further by down-sampling older memories rather than discarding them outright, retaining global plot structure over 8000-token sequences. More recently, RETRO [Borgeaud et al., 2022] and RePlug [Shi et al., 2023] swap hidden-state caches for a vector-store, blending Transformer learning with RAG-style retrieval and achieving lower factual error rates on long-form generation.

These advances do more than boost benchmark scores; they illustrate how explicit memory slots translate to agent coherence. A Minecraft AIC equipped with a key–value store can pin the location of every discovered village (keys) to its trading history (values), retrieving both facts when asked to "show me the best place to buy maps". Transformer-XL-style caches ensure the agent remembers that it promised to "finish the tower after sunset", while a compressive tier prevents ancient but critical memories—like the coordinates of a stronghold—from being silently overwritten. In short, each architectural variant tackles a different

facet of the coherence problem: "slot granularity" reduces **contradictions**; "segment recurrence" preserves **temporal continuity**, and RAG-style retrieval adds **grounded factuality**. The next wave of work combines these ingredients with hierarchical summarization and reflection to craft agents that stay true to their words hours—or days—after the first utterance.

6.2.2 Hierarchical Stores and Reflective Summarization

Coherence is not merely a matter of "finding facts". An AIC must continuously decide which experiences deserve to survive beyond the current context window. Hierarchical memory architectures address this by arranging storage into fast, limited-capacity "working tiers" and slower, long-capacity "archive tiers", while reflective mechanisms periodically distil raw logs into compact summaries. The result is a self-curating autobiographical record that anchors future reasoning to past commitments.

LongMem [Wang et al., 2023b] augments GPT-style models with a dual-tier store and a learned promotion–decay policy, sustaining coherent responses across 65000-token dialogues and reducing contradiction errors on *LongMemEval* [Wang et al., 2023b]. *MemGPT* [Packer et al., 2023] treats an LLM as an operating system, paging chunks between RAM-like and disk-like tiers and inserting interrupts that let the model decide when to swap memory. *MemoryBank* [Zhong et al., 2024] adds graph-structured indices plus decay schedules, cutting redundant tool reuse in Minecraft agents by 17%.

Reflective Memory Management (RMM) [Tan et al., 2025] extends hierarchical storage with an explicit metacognitive loop: agents alternate "prospective" reflections ("What will I need for the next plan?") and "retrospective" reflections ("Which events from the last hour should I keep?"). On *LongMemEval*, RMM yields a 13.2% boost in long-horizon task success and a 0.34-point gain in human ratings of perceived consistency. Ablation shows that removing either reflection phase doubles the rate of broken commitments, such as forgetting to finish a promised tower.

Hierarchical stores prevent "prompt drift" by pinning identity traits and long-term goals in a slow, immutable tier, while allowing volatile details—like a villager's current trading price—to reside in fast scratch-pads. Reflective summarisation compresses repetitive event logs into causal nuggets ("I mined 64 cobblestone for the tower foundation"), cutting retrieval latency and making agent explanations more legible. Together, the two mechanisms reduce three failure modes central to incoherence: forgotten promises, repeated tasks, and self-contradictory claims.

6.2.3 Retrieval-Augmented Generation as a Coherence Layer

Retrieval-augmented generation (RAG) [Lewis et al., 2020] couples an LLM with an external vector store, injecting the k most relevant memories into each new prompt. Although first proposed for factual QA, RAG has quickly become a **coherence layer**: it anchors generation to concrete episodes, reduces hallucination, and prevents an agent from contradicting its own history.

[Borgeaud et al., 2022] and [Shi et al., 2023] show that even generic chat models halve factual error rates once a similarity search feeds them background passages. Those gains translate to **narrative coherence**: when personal profile snippets are stored as retrievable keys, the model stops inventing new birthdays or job titles midway through a conversation. Self-RAG variants learn a policy for when to retrieve, further reducing contradiction rates on the Multi-Session Consistency Benchmark [Shi et al., 2023].

In open-ended Minecraft, *Voyager* [Wang et al., 2023b] appends every tool-use episode to a vector store; similarity search reduces repeated subtasks by 24% and boosts long-horizon quest completion. *Generative Agents* [Park et al., 2023] retrieve episodic memories ("I talked to Sam about the harvest") before each decision cycle; ablating retrieval halves the number of human judges rating villagers as "believable". *MineDojo* [Fan et al., 2022] combines code-snippets, wiki pages, and YouTube captions in a shared index, letting agents ground plans ("Craft an observer block") in cross-modal evidence.

On the other hand, specialised RAG pipelines target particular coherence failures. For example, in *Voyager*, the "tool-use logs" prevent "looping" on already-solved subgoals; In *Generative Agents* the "social memory banks" stop NPCs from repeating the same greeting; and the "spatial keymaps" keep exploration

agents from re-entering cleared rooms (*MemoryBank* extension [Zhong et al., 2024]). Key design questions are therefore: *(i)* what schema to embed, *(ii)* which distance metric best matches agent queries, and *(iii)* how large k can grow before context dilution outweighs benefit.

RAG alone cannot guarantee global consistency—stale or conflicting memories still require belief revision—but it has proven to be a lightweight, plug-and-play layer that lifts local coherence.

6.3 Persona and Goal Stabilization

A robust autobiographical memory does not, by itself, guarantee that an AIC will remain recognizably the same entity. Coherent AICs also need mechanisms that *(i)* lock core personality traits, *(ii)* monitor drift, and *(iii)* keep long-range objectives visible even when short-term tool calls dominate the prompt window. Three families of techniques have emerged:

- **Identity kernels**—immutable persona descriptors that anchor tone, values, and preferences;
- **Self-reflection loops**—metacognitive passes that detect and repair drift in real time;
- **Plan stacks**—explicit goal hierarchies that survive multi-step action sequences.

The subsections below survey academic and commercial systems that integrate these motifs, highlighting quantitative evidence that each reduces prompt drift and improves user-perceived coherence.

6.3.1 Identity Kernels: Freezing the Self

Identity kernels embed a fixed set of profile sentences or preference triples in the agent's *slow-memory* tier. Early work on persona-grounded chat—e.g. *PersonaGPT*—cut contradictions on *PersonaChat* by 35% when such kernels were included [Zhang et al., 2018]. [Park et al., 2023] transplanted the same idea to Sims-like villagers: immutable traits ("strict vegetarian," "soft-spoken") are never overwritten during daily reflections, so a villager does not suddenly agree to hunt rabbits for food. Identity rehearsal—rereading the kernel once per episode—further trims stylistic drift in week-long simulations from 14% to 3%.

Focusing on the game-centric extensions, *Character-LLM* trains a persona embedding jointly with dialogue generation; in an RPG quest log it cuts first-person/third-person inconsistencies by 27% and halves "out-of-role" utterances [Shao et al., 2023]. *RoleCraft-GLM* introduces emotion-tagged kernel sentences so NPCs exhibit stable affect as well as stable facts; user surveys show a 0.41 gain in perceived believability across a 90-minute dungeon run [Tao et al., 2023]. [Liu et al., 2025] pair identity kernels with a memory-consolidation scheduler that moves transient chit-chat into abstract "trait vectors". The framework raises dialogue-consistency $F1$ from 0.72 to 0.87 in a JRPG testbed. In *Ghost in the Minecraft* each sub-agent carries a "motivation kernel" (builder, explorer, farmer) that gates retrieval and planning; ablation reduces obtain-diamond success from 68 % to 44 % when the kernel is removed [Zhu et al., 2023].

When dialogue, action, and world state coexist, even tiny persona slips are conspicuous: a pacifist NPC swinging a sword or a proud dwarf speaking Elvish shatters immersion. Identity kernels act as "type-checks": before the agent speaks or plans, the kernel filters candidate outputs that violate its immutable descriptors. Empirically, every system above reports fewer repeated quests, reduced contradiction counts, or higher believability ratings once a kernel is in place—confirming that a small, write-protected seed of self-knowledge is a powerful antidote to prompt drift in long-running game worlds.

6.3.2 Self-Reflection and Metacognitive Loops

Even with a fixed persona, an agent can drift when its chain-of-thought accumulates errors or untested assumptions. *Reflexion* injects a *think–act–reflect* cycle: after every step the agent critiques—and, if needed, rewrites—its own reasoning, cutting repeated wrong assertions on *HotPotQA* by 21% [Shi et al., 2023]. *Self-Refine* applies the same critique-and-revise pattern to code generation, reducing logical bugs by 18% [Madaan

et al., 2023]. *Tree-of-Thought* (ToT) evaluates multiple reasoning branches and prunes any that contradict earlier assumptions, nearly doubling success on 24-step word puzzles while preserving a coherent narrative of partial solutions [Yao et al., 2023a].

For the gaming scenario, *Voyager* augments its GPT-4 planner with an iterative self-verification loop: after each code execution, the agent checks environment feedback and edits its next prompt accordingly. Ablation raises repeated-task rate by 24% and slows milestone unlocks by 3× in Minecraft [Wang et al., 2023a]. *MindForge* layers an explicit perspective-taking module on top of *Voyager*; reflection over false beliefs boosts task success in collaborative Minecraft runs by up to 70% [Lică et al., 2024]. *R-MCTS* replaces the back-prop phase of Monte-Carlo Tree Search with contrastive reflection, comparing expected outcomes to realized states; on the *VisualWebArena* game benchmark it yields a $6 \sim 30\%$ relative improvement over vanilla MCTS [Yu et al., 2024b]. *MrSteve* equips a low-level Minecraft controller with place-event memory plus a reflection rule that rewinds to the last safe waypoint when progress stalls, raising long-horizon task completion by 19% [Park et al.]. *Optimus-2* uses a Goal–Observation–Action-Conditioned Policy alongside a reflection gate that invalidates sub-plans if the observed world diverges from expectation, halving build-failure rates in survival mode [Li et al., 2025].

Suppose a Minecraft agent declares "I will finish the watch-tower before nightfall". A reflection pass can detect that only 12 blocks of cobblestone remain in inventory, conclude the goal is unattainable, and enqueue a mining sub-plan first—avoiding the broken promise that would otherwise shatter behavioural coherence.

Across these studies, reflection lowers three failure modes central to incoherence: *(i)* persistent factual mistakes, *(ii)* forgotten pre-requisites, and *(iii)* redundant or looping behaviours. By letting an AIC critique and repair its own reasoning—or even its own gameplay traces—self-reflection turns memory and persona into live constraints on future actions, yielding dialogue and behaviour that stay aligned over multi-hour sessions.

6.3.3 Plan Stacks: Durable Intentions

A **plan stack** records a hierarchical goal structure $\langle \text{mission} \rightarrow \text{sub-goal}_1, \text{sub-goal}_2, \ldots \rangle$ that persists across tool calls and API round-trips. By keeping the current high-level intention visible to the agent, plan stacks stop it from chasing short-term affordances that contradict its overall objective. For general LLM agents, *ReAct* interleaves chain-of-thought with action tokens, implicitly building a stack and lowering forgotten sub-goals on *ALFWorld* by 32% [Yao et al., 2023b]. *PEARL* [Sun et al., 2023] prompts an LLM to *plan, then execute* actions over long documents; and LangChain's engineering blog later generalised this "plan-then-execute" loop into a reusable agent template for real-world RAG and browser-automation tasks [LangChain Inc., 2024].

Voyager caches every discovered tool trajectory as a ⟨goal, sub-goal, path⟩ triple; this explicit stack lets the agent resume interrupted builds and lifts long-horizon quest completion in Minecraft by 24% [Wang et al., 2023a]. Ghost in the Minecraft [Zhu et al., 2023] decomposes tasks into goals, actions and operations; without the hierarchical stack, its success on "obtain diamond" falls from 68% to 44%. [Köhn et al., 2020] uses hierarchical planners to output multi-level Minecraft building instructions, showing that abstraction control improves user ratings of instruction clarity. *ForgER* [Skrynnik et al., 2020] extracts a chain of sub-tasks from noisy expert demonstrations; the resulting hierarchical DQN was the first to mine a diamond in the MineRL challenge. *Cicero* [Meta Fundamental AI Research Diplomacy Team et al., 2022] fuses a large dialogue model with a strategic planner that rolls out hundreds of candidate move sequences—effectively a latent plan stack—to maintain consistent negotiation in the board-game *Diplomacy* [Wikipedia contributors, 2025a], achieving human-level performance in online leagues.

In open-world games a single utterance "Let's build a beacon" may require hours of intermediate steps. Plan stacks keep this top-level promise in focus, preventing: *(i)* Goal amnesia — dropping or repeating sub-goals already achieved; *(ii)* Contradictory shortcuts — e.g. tearing down the half-built tower for materials; *(iii)* Style drift — switching from "master builder" to "speed-runner" mid-session.

Across the studies above, hierarchical stacks raise long-horizon success rates, lower mission aborts (web automation), and increase human believability scores (Diplomacy)—evidence that durable intentions are a critical pillar of AIC coherence.

6.3.4 Implementations in Original Character Games

Original-Character (OC) [Wikipedia contributors, 2025c] games let players author a fully fledged persona—name, backstory, goals, speech style, even moral code—and then inhabit that avatar inside an LLM-driven world. Such consumer-facing character platforms handle "persona stabilization" primarily through **profile kernels** (short and write-protected text blocks) that the model rereads (or the server prepends) at every turn. Several ecosystems now expose rich metadata fields—backstory, motives, relationship graphs, even preferred sentence length—to let designers lock an original personality before chat begins.

In Character.AI [Character.ai, 2025], Each bot ships with a "Character Profile" which contains `name`, `greeting`, `short_description`, `long_description`, and other attributes [Character.AI, 2025]. The profile is prepended to every prompt, acting as a write-protected identity kernel that keeps style and agenda on track.

As a local front-end, SillyTavern [SillyTavern Contributors, 2025] cannot alter the model but relies on the "character permanant tokens": `name`, `personality`, `description` and `scenario` [SillyTavern Maintainers, 2025]. A rolling memory string stores user-approved facts that auto-inject two lines above the assistant's turn, functioning as a lightweight reflection loop.

6.4 Cross-Modal Grounding

Coherence in an AIC ultimately means more than "saying the right thing"; the agent must *say, do, and emote* in a way that feels unified. Cross-modal grounding tackles this requirement by learning shared representations that tie dialogue, locomotion, and affect into a single behaviour stream. We survey the field along four axes.

6.4.1 Unified Multimodal Embeddings and Models

Recent advances in multimodal embedding have enabled the alignment between multimodal signals, allowing AICs to `think` across modalities with fewer contradictions and more stable intentions. Early breakthroughs such as *CLIP* aligned image and text in a joint space, enabling zero-shot captioning and visual search [Radford et al., 2021].

Robotics-oriented work soon followed: *R3M* [Nair et al., 2022] pre-trains a visual encoder on millions of egocentric videos paired with language, yielding a representation that transfers to 28 manipulation tasks with 3× less fine-tuning data. These embeddings serve as the glue connecting what the agent `sees` to what it can `say` or `plan`, reducing contradictory references to unseen objects. *RT-1* [Brohan et al., 2022] treats low-level motor commands as a third modality, mapping camera frames and language goals to 6-DoF actions and solving 708 kitchen tasks at 97% success. *RT-2* [Brohan et al., 2023] transfers web-scale language knowledge into robotic control via a unified VLA token vocabulary, boosting novel-object manipulation by 35%. *PaLM-E* [Driess et al., 2023] injects proprioceptive and vision tokens into a PaLM-style decoder, reducing tool-confusion errors from 11% to 4% in multi-step pick-and-place tasks.

For open-world games, **MineDojo** links wiki text, code snippets, and YouTube frames in a retrieval index; grounding plans in these cross-modal keys cuts failed crafting attempts by 30% [Fan et al., 2022]. *STEVE-1* [Lifshitz et al., 2023] learns a text-to-behaviour transformer that emits several minutes of Minecraft actions; an internal memory cache of sub-plans prevents mid-sequence drift, raising build-task completion by 15%. Together, these systems show that planning and execution stay aligned when dialogue, vision, and action tokens live in the same representational space.

6.4.2 Industry Toolkits

Commercial platforms are rapidly incorporating cross-modal grounding into their toolkits to support coherent, multimodal AI characters. These systems focus not just on speech or gesture in isolation, but on maintaining a unified internal state that synchronizes dialogue, animation, audio, and embodiment. As a result, coherence is enforced *across modalities*, producing agents whose verbal intentions and physical expressions remain in sync.

The NVIDIA Avatar Cloud Engine (ACE) [NVIDIA, 2024] provides an integrated pipeline for real-time multimodal agent control. Its architecture chains: *Riva* - for speech-to-text and text-to-speech; *Audio2Face* - for neural facial animation; and *Omniverse PhysX* - for gesture generation, all governed by a shared conversational state vector. This vector acts as a multimodal control token—when the agent says "I'm surprised", its voice tone, eyebrow raise, and body recoil are all driven by the same latent intention. This tight coupling prevents the incoherence typical of modular systems where speech and motion desynchronize under latency or drift.

Roblox Avatar Chat [Roblox Corporation, 2023] fuses voice input, facial tracking, and avatar animation into a synchronous multimodal expression pipeline. Real-time voice is transcribed, mapped to mouth and head-pose via learned motion embeddings, and displayed as synchronized speech bubbles and avatar motion. Unlike earlier avatar systems where chat, voice, and gesture were independent, the use of shared embeddings here ensures that "What are you looking at?" `sounds` correct, `looks` correct, and is delivered to the right `agent`, reducing ambiguity and boosting social presence.

Altera's *ProjectSid* [AL et al., 2024] introduces agents with structured autobiographical memory, episodic embeddings, and goal-conditioned planning to support long-term coherence. By combining high-level reflection, identity persistence, and socially plausible behavior, *ProjectSid* enables AI characters to maintain narrative consistency, respond adaptively, and evolve realistically within open-ended interactive environments.

[Richard et al., 2021] introduced a multimodal animation framework where facial expressions are driven by audio and eye-tracking signals, enabling expressive social communication with only partial sensor input. Later, [Bai et al., 2024] extended this work with a self-supervised approach to animate avatars using only cameras on VR headsets, removing the need for dense face capture setups. These systems demonstrate robust cross-modal coherence: when an avatar says "I'm shocked", its voice tone, eye widening, and mouth shape are jointly inferred from a shared latent state. This synchronization of audio, vision, and expression improves perceived realism and avoids the disjointed behavior (e.g., speaking with a neutral face) common in modular pipelines.

6.5 Long-Horizon Evaluation for Coherence

While short-term coherence can be inferred from individual responses or actions, assessing "long-horizon coherence"—how consistently an AIC behaves over extended interactions—remains an open challenge. Standard benchmarks often fail to reveal gradual persona drift, memory degradation, or narrative inconsistency. This section reviews key academic efforts and emerging commercial strategies for evaluating coherence across time, dialogue, and gameplay trajectories.

6.5.1 Academic Benchmarks and Methodologies

For the general research and development, several metrics have emerged to quantify coherence over time:

- **Contradiction Rate:** Measures the frequency of statements that contradict prior outputs. [Dziri et al., 2019] utilize natural language inference techniques to detect such contradictions in dialogue systems.

- **Memory Recall Accuracy:** Assesses the agent's ability to retrieve relevant past events without hallucination. The HEMA architecture demonstrates improved recall accuracy in long-context conversations [Meta Fundamental AI Research Diplomacy Team et al., 2022].
- **Persona Consistency:** Evaluates the stability of an agent's personality traits over time. [Park et al., 2024] employ the Big Five Inventory to measure consistency in generated personas.
- **Narrative Continuity:** Examines the logical progression and coherence of narratives. The SCORE framework introduces metrics for character consistency and plot progression to assess narrative coherence [Yi et al., 2025].

Additionally, temporal segmentation and trajectory analysis techniques are employed to model long-form coherence, particularly in video data [Ding et al., 2023].

In practices, [Backlund and Petersson, 2025] introduced *VendingBench*, a benchmark designed to expose long-term coherence failures. In this environment, agents operate a vending stall over 8 real-time hours, interacting with NPC customers, managing inventory, and recalling past conversations. Metrics include: *Goal fidelity* - does the agent persist in its business objective; *Episodic reference accuracy* - can the agent refer back to earlier customers or pricing decisions?; and *Trait consistency* - does the agent maintain its original temperament (e.g., cheerful, frugal)? Models augmented with structured memory and reflective summarization scored 0.34 higher in human-rated coherence over 100 interactions.

[Tan et al., 2025] evaluated their Reflective Memory Management (RMM) framework using *LongMemEval*, a suite of open-world tasks that span $4 \sim 12$ hours. Agents are assessed on memory longevity, plan persistence, and intention drift. Reflective agents achieved a 13.2% increase in long-horizon task success and demonstrated fewer contradictions in retrospective interviews with human evaluators.

[Park et al., 2023] introduced memory probe tests in their Sims-like town. After $2 \sim 3$ days of simulated time, agents are queried about social relationships and personal routines. Ablation studies show that disabling memory retrieval or self-reflection halves recall accuracy and increases behavioral contradictions, such as agents denying friendships they previously formed.

6.5.2 Deployment Challenges

Despite recent advances in evaluating long-horizon coherence, several key limitations remain unresolved. First, most existing benchmarks are constrained to relatively short timeframes—typically minutes to a few hours—failing to capture the kinds of drift and degradation that emerge over days or weeks. This limits our ability to assess whether agents can maintain consistent goals, personalities, and memories across truly extended interactions.

Second, many current evaluation methods focus almost exclusively on linguistic coherence, neglecting contradictions that manifest through physical actions, spatial behavior, or motor routines. Moreover, measuring coherence in social dynamics remains particularly challenging: agents may form or abandon relationships over time, but detecting and evaluating the consistency of these social behaviors is still poorly supported by automated tools. Addressing these challenges will require richer, longer simulations and new multimodal annotation frameworks that capture both verbal and nonverbal dimensions of agent behavior.

6.6 Discussions

The pursuit of coherent AICs has been fundamentally reshaped by the advent of LLMs. Unlike earlier rule-based or behavior-tree driven systems, which required exhaustive scripting and domain-specific engineering, LLMs offer a fluid substrate capable of synthesizing personality, memory, and intention at runtime. This shift has unlocked a new paradigm of character creation and maintenance.

First, LLMs dramatically lower the barrier for designing richly textured, narratively believable agents. Their pretrained corpora encode a vast base of cultural knowledge, social roles, and narrative archetypes—enabling AICs to adopt personas and backstories that rival, and in many cases exceed, those created by

professional character designers. Where traditional game development pipelines required weeks of manual effort to specify consistent background lore or behavioral rules, an LLM-guided AIC can improvise plausible autobiographical events, social norms, and moral stances in real time. This makes them particularly well-suited for open-ended environments and emergent storytelling.

Second, the analytical flexibility of LLMs affords a unique strength in maintaining coherence amid ambiguity and change. Human behavior is not deterministic, and coherence does not imply mechanical regularity. Rather, it reflects internal logic, continuity of beliefs, and plausible adaptation. LLMs can reason over fuzzy inputs, incomplete memory, and context shifts—preserving key informational threads (e.g., personal values, goals, relationships) while flexibly updating details. This capacity to tolerate noise while retaining identity mirrors human cognition, and stands in stark contrast to brittle symbolic systems of the past.

Finally, benchmarking long-horizon coherence remains a fundamental challenge. While scripted probes and contradiction detection are useful at small scale, they fall short in evaluating agent coherence over extended gameplay. A promising trend is to embed AICs as full participants in human-calibrated game scenarios—assigning them long-term objectives and comparing their performance, decision stability, and narrative plausibility against human players. This setup not only surfaces drift and inconsistency, but also provides feedback loops for model fine-tuning and memory optimization. In effect, coherence becomes a property emergent from task success, social participation, and developmental trajectory—moving evaluation closer to Turing-style interaction tests.

In sum, LLMs offer a generative and evaluative engine that aligns well with the multifaceted nature of coherence. They enable the construction of believable, adaptable characters and open the door to richer, long-term agent modeling. However, realizing their full potential will require continued innovation in memory architectures, interpretability tooling, and scenario-based benchmarking. Coherence is no longer a static checklist—it is an evolving property of agents embedded in complex, shared worlds.

Chapter 7

Collaboration of Multiple AI Characters

7.1 Background: The Motivations behind Collaborations

The promise of embodied inecraft agents extends beyond individual competence: it lies in the emergent capabilities that arise when *multiple* AI Characters (AICs) coordinate toward shared goals. Large-scale benchmarks such as *MineDojo* [Fan et al., 2022], open-ended agent frameworks like *Voyager* [Wang et al., 2023a], and sandbox social simulations of *Generative Agents* [Park et al., 2023] show how language models can endow agents with flexible reasoning, rich personas, and persistent memory. Building on these foundations, multi-agent collaboration unlocks several motivations that are difficult—or impossible—to realize with a single agent alone:

- **Task breadth and complexity.** Canonical Minecraft undertakings (e.g. mass-scale settlement building, long-distance exploration parties, or nether-to-overworld supply-chain automation) require division of labour, temporal synchronisation, and dynamic re-planning that exceed the cognitive envelope of a single agent. Collaborative experiments with more than 1000 agents report super-linear speed-ups for build-and-craft tasks when roles are specialised and plans are jointly optimised [Qian et al., 2024].
- **Collaborative scaling and emergent intelligence.** Recent work observes a *collaborative scaling law*: solution quality follows a logistic growth curve as the number of cooperating agents rises, with emergent abilities appearing long before a single model would exhibit them on its own [Qian et al., 2024]. Understanding and exploiting this law is a research driver in its own right.
- **Social believability and narrative richness.** To pass an Embodied Turing Test, AICs must display negotiation, humour, and shared rituals. World-scale agent simulations (*MineLand* [Yu et al., 2024a]) show that believable societies emerge only when many agents can interact, gossip, and form norms.
- **Human–AI co-creation and safety.** Frameworks such as *AutoGen* [Wu et al., 2023] expose multi-agent teams as a programmable interface, enabling players or modders to join the loop, audit plans, and over-ride unsafe actions. Collaboration therefore becomes both a creative catalyst and a fine-grained safety layer.
- **A research lens on LLM limitations.** Position papers criticise current LLM-centric systems for neglecting classic multi-agent principles [La Malfa et al., 2025]. Studying collaboration in Minecraft thus offers an experimental playground for aligning Multi-Agent System (MAS) theory with modern LLM practice.

To transform the motivations into practice, we organise review of related works along two orthogonal axes: `collaboration paradigm` (i.e. centralized orchestration, decentralized emergence, or hybrid mediation) and `research focus` (the eight systemic issues that cut across all paradigms). Table 7.1 maps each research focus to concrete questions that arise under each paradigm and cites representative Minecraft AIC studies, where:

- **Centralised orchestration.** While a planner simplifies global optimisation and debugging, it risks becoming the "single point of failure". Research should quantify how planner capacity, latency, and reliability bound team performance—especially in GPU-accelerated LLM loops [Qian et al., 2024, Gong et al., 2023].
- **Decentralised emergence.** Purely peer-to-peer systems scale elegantly but face coordination pathologies: split-brain world models, gossip overload, and fragile conventions (*MineLand* [Yu et al., 2024a], *CAMEL* [Li et al., 2023]). Key open problems include on-the-fly protocol repair and conflict-aware memory garbage collection.
- **Hybrid mediation.** Hybrids promise the best of both worlds but risk "shadow centralization": a mediator that quietly dominates by suggestion. Future work must formalize *influence budgets* and design experiments where mediator bandwidth—and thus power—is capped, echoing work in human deliberative democracy [OpenAI, 2024b].

The existing benchmark suites such as *MineRL* [Guss et al., 2019], *VillagerAgent*, and *TeamCraft* [Long et al., 2024] have demonstrated the value of empirical evaluation in Minecraft-based environments. *MineRL*, in particular, pioneered the use of human demonstrations and sparse rewards for hierarchical skill acquisition, establishing Minecraft as a credible testbed for embodied learning. However, these benchmarks remain limited in collaborative scope: they either focus on individual agents or support only tightly scoped multi-agent tasks with minimal coordination dynamics. A truly next-generation benchmark must widen the aperture. It should support toggling of collaboration paradigms (full planner, soft mediator, or pure decentralization), feature long-horizon construction missions punctuated by human-verified checkpoints, log fine-grained provenance for explainability audits, and chart "collaborative scaling-law" curves that extend well beyond the 1000-agent frontier.

Taken together, these requirements underscore a broader lesson. Paradigm choice shapes—but does not supplant—the eight systemic research directions outlined above. Meaningful progress will hinge on paired advances: for example, a collective-reflection mechanism that is stress-tested both with and without a central planner, or an emergent communication protocol evaluated under strict mediator bandwidth caps. Because Minecraft uniquely combines open-ended gameplay, rich visual grounding, and an extensive modding ecosystem, it stands out as an ideal crucible in which to pursue—and compare—these intertwined lines of inquiry.

7.2 Centralized Collaboration Methods

Centralized methods introduce a "single point of coordination"—commonly referred to as the `coordinator`, `mediator`, or `planner`—at the root of the agent hierarchy. This coordinator typically maintains an explicit world model, decomposes mission prompts into task graphs, allocates sub-goals to worker agents, and monitors execution for possible replanning or safety intervention.

Future works may extend this model by training a "shared reflector" that enhances each agent's self-critique loop, improving team-wide performance without modifying the underlying LLM-based actors. This approach decouples introspective reasoning from action policies, enabling more coherent and adaptive group behavior. However, several challenges remain open: notably, the problem of credit assignment in joint reflection processes, and the trade-off between reflection latency and responsiveness during time-critical scenarios such as survival raids.

Table 7.1: Paradigm–direction matrix for multi-AIC research. Grey cells indicate tensions that are uniquely challenging in that paradigm.

Research Direction	**Centralized**	**Decentralized**	**Hybrid**
Collective reflection & meta-cognition	Coordinator maintains a global reflector.	Agents reflect locally; need synchronized insights.	Mediator fuses distributed reflections.
Shared memory & knowledge	Central shard; key issue: write arbitration.	Peer-to-peer CRDT or vector-DB sync—conflict resolution is hard.	Mediator as gossip relay.
Emergent communication	Planner can enforce symbolic protocols (compressed).	Protocols must self-organize; observe language drift.	Mediator offers translation hints but cannot dictate.
Trust, explainability, reputation	Single source of truth enables causal tracing; planner must expose chain-of-thought.	Need local explainers and cross-checking reputations.	Mediator aggregates proofs and reputations for audit dashboards.
Dynamic role allocation & resources	Scheduler reallocates roles with global view.	Agents self-select; risk of oscillations.	Mediator proposes reallocations, agents voluntarily accept.
Safety, norms, governance	Central safety filters and vetoes.	Norms emerge; require distributed norm-violation signalling.	Mediator hosts deliberative court; cannot directly stop actions.
Human–AI co-creation	Planner presents unified interface.	Humans join as another peer; UI must visualise many voices.	Mediator bridges human chat to agent swarm, enforcing rate limits.
Benchmarking & scaling laws	Easy to count planner bottlenecks; explore planner depth vs. team size.	Measure convergence rates, social entropy.	Study mediator bandwidth vs. emergent efficiency frontiers.

7.2.1 Task Decomposition and Distribution

The coordination usually begins with a *task decomposer*—a planner that turns an open-ended mission prompt (e.g. "build an automated iron farm") into an executable set of sub-goals $\mathcal{G} = \{g_1, \ldots, g_n\}$. The existing works now converges on three technical strata:

- **Graph–based decomposers.** *VillagerAgent* constructs a directed acyclic graph (DAG) whose nodes encode spatial, causal, and temporal dependencies; an `AgentController` then assigns each leaf to the worker best matching its skill profile. *MACNet* generalises this idea: a root agent samples a DAG topology over heterogeneous specialists, discovers a Collaborative Scaling Law, and shows logistic performance growth up to 128 agents on MineDojo build-and-craft benchmarks [Qian et al., 2024].
- **Hierarchical planners.** Ghost-in-the-Minecraft (GITM) [Zhu et al., 2023] inserts a text-level goal–sub-goal–action–operation stack between GPT-4 and Minecraft API calls, enabling few-shot generalisation from "obtain obsidian" to multi-step lava-bucket workflows. Classic deep-RL work, such as H-DRLN [Tessler et al., 2017] pre-compiles reusable skills; modern planners simply invoke those skills as subtasks in the decomposition tree.
- **Language-model planners.** LLM-native frameworks embed the decomposer inside the model. *AutoGen*'s `Coordinator` prompts GPT-4 to output an ordered checklist of subtasks, then loops with executors until each item is verified. *MindAgent*'s GPT-4 `Scheduler` plays the same role in the *CuisineWorld* scenario and achieves +43% throughput over decentralized baselines. The prompting strategies such as *Tree-of-Thoughts* (ToT) [Yao et al., 2023a] generate a branching search over partial plans and prune with self-evaluation, producing deeper yet safer task graphs.

The task decomposition thus offers global optimality and clear explainability trails—but at the cost of a single failure point and potential creativity loss. Once $\mathcal{G}$ is built, coordinators invoke one (or a mix) of the following policies: *Skill-matched scheduling*—map each g_i to the agent with the highest learned competence vector ($VillagerAgent$); *Token auctioning*—workers bid natural-language rationales; the planner allocates tokens to the most confident proposal (*AutoGen*). Adaptive re-sharding hierarchical systems such as *HAS* [Zhao et al., 2024] dynamically split and merge sub-graphs as world conditions change.

Gabriel et al. [Jeyakumar et al., 2024] further advance centralized architectures by introducing an agentic framework that supports dynamic task decomposition, real-time tool integration, and adaptive execution. Their system constructs evolving task graphs from natural-language prompts and uses a reasoning loop to choose tools and dispatch subtasks during runtime. The accompanying AsyncHow dataset provides a controlled environment to evaluate coordination fidelity and structural correctness in decomposed plans, offering a valuable benchmark for future multi-agent Minecraft systems.

7.2.2 Communication and Dynamic Role Allocation

Effective communication between a central coordinator and distributed agents hinges on maintaining a consistent, queryable representation of shared state. Most systems adopt a centralized memory store—typically implemented as a vector index, key–value cache, or symbolic scene graph—that agents can read from and sometimes write to (*AutoGen*, *MindAgent*). This memory typically encodes current objectives, task dependencies, available resources, and the status of peer agents. Critical design concerns include arbitration of concurrent writes, detection of stale updates, and controlled memory scoping, particularly in settings where human players co-exist with autonomous agents. To address data isolation and access control, many systems now incorporate techniques such as role-based access control (RBAC), permission tokens, or namespace partitioning [Chen et al., 2023].

Beyond communication and memory, centralized systems must also manage the distribution of labor—deciding "who" should do "what" at any given time. This introduces the problem of "dynamic role allocation", where the coordinator not only decomposes the task into subtasks but also continuously reassigns

roles based on agent availability, expertise, or changing world conditions. *VillagerAgent* tackles this problem by mapping sub-goals from a task graph to agents based on pre-defined skill profiles and context-sensitive scoring. *MACNet* extends this further by dynamically adapting agent clusters and communication topology as the task progresses, showing that role diversity and adaptive delegation significantly improve build efficiency in large teams. The *HAS* framework [Zhao et al., 2024] introduces a hierarchical control layer where planners can reassign roles mid-task by monitoring execution traces and dynamically reorganizing subteams. These systems highlight a key advantage of centralized control: the ability to globally reason about team structure, balance workload, and intervene when bottlenecks, conflicts, or idle agents are detected.

To communicate effectively with multiple agents at scale, coordinators often rely on a hybrid of symbolic and natural language protocols. Symbolic formats—such as JSON stubs for positional data, inventory contents, or crafting plans—offer low-latency transmission and easy programmatic parsing. Meanwhile, natural language remains the medium of choice for negotiation, error recovery, and subtask clarification. Hybrid protocols that combine both symbolic structure and open-ended language have been shown to outperform pure language-based interaction in multi-agent settings—for example, by improving grounding accuracy, reducing ambiguity, and accelerating convergence in cooperative tasks [Gong et al., 2023]. Nonetheless, they introduce trade-offs in transparency and traceability, particularly when symbolic interactions bypass human-readable logging or break down during unexpected state transitions.

7.2.3 Strenghts and Limitations

Centralized coordination offers several strong benefits. First, it enables global optimization: a planner with access to the full world graph can run task schedulers, routing algorithms (e.g., A^*), or constraint solvers to produce efficient, near-optimal action plans and resource pipelines. Second, it provides safety and controllability: a central veto point can block unsafe behaviors, enforce permissions, or ensure norm compliance, which is particularly important in human–AI co-play scenarios. Third, centralization enhances explainability: the planner's decision trace can be logged and audited, making it easier to reconstruct why agents performed particular actions—something difficult to achieve in more emergent systems.

However, these benefits come at notable costs. Most obviously, the planner becomes a single point of failure: if the coordinator fails (e.g., due to latency, GPU unavailability, or logic bugs), all downstream agents may stall. Second, such systems often hit a scalability bottleneck, since the coordinator must process $\mathcal{O}(N)$ agent messages and maintain a coherent view of the global state—unless expensive sharding or planner-as-a-service techniques are employed. Finally, centralized control can stifle emergent creativity: by enforcing top-down constraints, it may prevent agents from discovering novel coordination strategies that arise more freely in decentralized or hybrid models.

Key open challenges remain in designing robust and adaptive centralized coordination systems. One major issue is planner resilience: current architectures often lack mechanisms for fault tolerance, such as redundancy, hot failover, or graceful planner handoff, leaving them vulnerable to single-point failures. Another challenge is bounded influence—that is, how to formally constrain the planner's authority so that hybrid systems preserve agent autonomy and do not regress into de facto centralization. Additionally, planner–agent alignment remains an open frontier: agents and coordinators must be co-trained or co-prompted to anticipate each other's strategies and intentions, reducing miscommunication and dialogue overhead. Finally, maintaining task stability across agents is non-trivial—especially when sub-goals must be reallocated dynamically due to agent dropout or unexpected world events. Together, these concerns define the limits of centralized designs and motivate exploration of more decentralized and hybrid approaches.

7.3 Decentralized Coordination Methods

In decentralized coordination, no single agent possesses global authority. Instead, collaboration emerges through peer-to-peer communication, local perception, and implicit social norms. Each AIC operates semi-autonomously, observing its local environment, querying shared memory (if available), and exchanging

messages or actions with peers to align toward common goals. While this removes the scalability bottleneck and single point of failure inherent to centralised systems, it also introduces new challenges in consistency, convergence, and emergent safety.

7.3.1 Peer-to-Peer Negotiation and Role Emergence

Frameworks like *CAMEL* [Li et al., 2023] and *AgentVerse* [Chen et al., 2023] demonstrate that LLM-powered agents can coordinate effectively through natural-language dialogue without centralized control. In these systems, agents are instantiated with persona profiles (e.g., builder, farmer, explorer) and operate by proposing plans, raising objections, voting, and offering alternatives in open-ended message-based dialogue. Task allocation is not explicitly assigned; instead, roles emerge organically as agents gain confidence or are acknowledged by peers as the most competent for a given subtask. Role emergence thus becomes a function of both self-assessment and peer validation, reflecting a decentralized analogue of trust-weighted delegation.

Building upon these principles, *ChatDev* introduces a collaborative software engineering framework where specialized LLM-driven agents simulate distinct roles (e.g., CEO, CTO, developer, tester) and work together through structured multi-turn dialogues to co-design, implement, and test software systems [Qian et al., 2023]. Coordination arises solely through language, using predefined interaction templates and conversation chains, without a central planner or global task scheduler. This architecture enables robust modularity and emergent division of labor in a completely decentralized setting.

ChatCollab further extends this direction by enabling open-ended team-based collaboration between human and AI agents [Klieger et al., 2024]. Unlike fixed-role systems, it allows agents—whether human or AI—to join or leave tasks dynamically, engage in peer-to-peer communication, and adapt roles mid-dialogue. This agent-agnostic design makes it possible to support hybrid teams in which autonomy, communication policy, and task execution are distributed without privileged control. Such designs are particularly relevant in Minecraft-like environments, where flexible, social collaboration is required across varied tasks and agent types.

These systems highlight that effective division of labor and consistent goal progress can arise even in the absence of global task graphs or predefined hierarchies. Decentralized negotiation has also been explored in game and simulation contexts, showing the emergence of behaviors such as persuasion, turn-taking, leadership negotiation, and even deception in purely peer-to-peer settings. While rich in expressiveness and scalability, these systems still face open challenges in convergence time, coordination stability, and maintaining coherence under partial observability.

7.3.2 Emergent Societies and Norm Formation

Decentralization supports large-scale simulation of social dynamics by allowing agents to independently develop conventions, behaviors, and interaction patterns over time. In *MineLand* [Yu et al., 2024a], hundreds to thousands of Minecraft agents interact within an open-ended environment featuring partial observability, stochastic events, and limited shared resources. Without explicit instruction, agents spontaneously engage in foraging, bartering, shelter construction, and task delegation. Remarkably, informal social conventions such as queueing behavior, ad hoc leadership, and item-exchange rates emerge purely through repeated local interactions.

The *Generative Agents* framework further demonstrates that even small-scale agent societies—25 agents in a sandbox town—can exhibit rich emergent routines. Agents in this simulation developed coherent social behaviors such as daily schedules, gossip propagation, party planning, and reactive memory updating. Despite lacking global state or external enforcement, the system generated plausible community-level patterns like town-wide events and the spread of shared beliefs. These behaviors arose from lightweight memory summarization and tree-of-thought planning loops, underscoring the expressive potential of LLM-guided, decentralized social agents.

Recent studies push this direction further. Altera AI's large-scale Minecraft simulation [AL et al., 2024] involved over 1000 agents interacting in an open world where localized rituals, emergent religions, and

territorial norms appeared over the course of days. Agents invented rituals such as fire-watching, artifact curation, and communal construction, driven by persistent memory, local consensus, and social reinforcement mechanisms. Similarly, *Voyager*-based multi-agent extensions have demonstrated the emergence of collective behavior patterns such as cooperative mining or tool-sharing heuristics when agents are provided only loose alignment incentives and shared world feedback.

In parallel, theoretical work has begun to formalize how LLM agents develop and propagate social conventions. Ashery and Baronchelli [Ashery et al., 2025] show that populations of LLMs interacting under turn-based constraints and memory can converge to stable norms—including cooperative biases and culturally biased language—without external control. These findings suggest that scalable norm formation may be a natural property of sufficiently expressive decentralized language agents, especially when paired with long-term memory and identity persistence.

Altogether, these systems demonstrate that decentralized coordination enables the study—and synthesis—of social intelligence at scale. Rather than simply solving tasks, agents begin to act as culture-bearers: developing local customs, shared rituals, and emergent institutions that reflect the unique pressures of their simulated environments. For Minecraft AICs, this opens the door to simulating lifelike, evolving villages, factions, or social contracts—without ever requiring top-down scripting or enforced norms.

7.3.3 Coordination via Shared Memory and Local Perception

While lacking a central planner, many decentralized systems still support indirect coordination through shared memory structures—such as vector stores, key–value memories, or text-based scratchpads—that agents can query, append to, or reason over. In these cases, memory acts as a public blackboard where agents leave plans, facts, observations, or intermediate results for others to interpret. For example, *Voyager* uses a persistent trajectory log and episodic reflection buffer that allows agents to revise strategies based on past performance and perceived intent of co-located agents. Similarly, systems like *MemGPT* [Packer et al., 2023] augment LLM agents with long-term external memory that is dynamically managed and structured for context-aware queries—making it possible for agents to build a shared situational picture without centralized coordination.

However, shared memory alone does not guarantee stable behavior. Without locking protocols or arbitration logic, concurrent writes may introduce inconsistency, duplication, or race conditions. In response, several frameworks (e.g., *AgentVerse*) restrict writing to scoped namespaces or introduce probabilistic write thresholds based on agent confidence or social agreement. Others, like decentralized *AutoGen* variants, use memory summaries (e.g., "most agreed-upon fact") or shared dialogue consensus to guide future decisions without granting hard authority.

An alternative coordination strategy relies solely on local perception and behavioral inference. In this view, each agent observes the actions and movement patterns of others in the environment and infers latent goals or beliefs via embedded heuristics or prompted LLM reasoning. This pattern is prominent in agent-based simulations such as *Generative Agents*, where agents adjust plans not via shared memory but by visually recognizing peers' activities (e.g., walking toward a bakery implies hunger). Similarly, models inspired by the ReAct [Yao et al., 2023b] can integrate perceptual cues, self-questioning, and action history to infer when collaboration is needed—without any explicit messaging.

These techniques—perception-driven inference and indirect memory-based coordination—are particularly well-suited to Minecraft, where shared memory access may be intermittent (due to chunk loading or scope boundaries), and rich visual signals (block changes, avatar movement, item transfers) offer abundant cues for decentralized decision-making. Open research questions remain around memory consistency under scale, perception noise tolerance, and the emergence of shared beliefs in the absence of structured communication.

7.3.4 Strengths and Limitations

Decentralized coordination methods offer powerful advantages, particularly in scalability and emergent expressiveness. Because there is no central authority to manage state or dispatch tasks, coordination scales sub-linearly with the number of agents. Each agent reasons locally based on shared memory, peer communication, or perceptual cues, enabling large populations to operate in parallel without overwhelming any single coordination channel. This naturally aligns with the open-ended structure of Minecraft worlds, where tasks are not rigidly bounded and agents must often adapt to fluid, partially observable environments.

Perhaps more significantly, decentralized methods foster "emergent creativity". When agents interact without scripted plans or top-down constraints, novel coordination strategies can arise—ranging from informal division of labor to rituals, local norms, or complex social dynamics. These emergent phenomena are not only functionally useful but also socially meaningful, contributing to the believability and richness of agent societies. In simulations such as *MineLand* or *Generative Agents*, agents have demonstrated the ability to form group habits, institutions, and even cultural practices—without any central controller or planner.

At the same time, these benefits come with significant trade-offs. Without a planner, convergence to coordinated behavior can be slow, fragile, or highly context-dependent. Agents may duplicate work, fail to resolve bottlenecks, or cycle endlessly between conflicting plans. Safety becomes probabilistic: undesirable behaviors like hoarding, selfishness, or aggression may go unchecked unless mitigated by emergent social norms or hard-coded constraints. Social alignment is difficult to guarantee, particularly when agents operate with heterogeneous goals, skills, or memory scopes.

These limitations open a series of pressing research challenges. A central question is how agents can dynamically learn "communication protocols"—whether symbolic or natural-language—that are interpretable, efficient, and robust to ambiguity or noise. Another issue concerns "norm enforcement": in the absence of a central adjudicator, agents must self-monitor for rule violations and apply corrective pressure through trust scores, exclusion, or collective sanctioning. Similarly, "distributed plan repair" remains an unsolved problem: when coordination breaks down, how do agents collaboratively detect, localize, and recover from failure? Finally, as agents maintain their own memories, multi-agent belief reconciliation becomes critical. Can agents detect when their internal models of the world diverge, and resolve those inconsistencies purely through interaction?

In summary, decentralized approaches enable lifelike, bottom-up social dynamics and scale well to large agent populations, but they demand careful system design to ensure robustness, coherence, and safety. The following section turns to hybrid architectures, which attempt to recover some of the structure and efficiency of centralized systems while preserving the adaptability and creativity that emerge from decentralized interaction.

7.4 Hybrid Coordination Methods

Hybrid coordination methods occupy the middle ground between centralized planning and fully decentralized self-organization. These systems aim to combine the global structure and oversight of centralized controllers with the scalability, flexibility, and emergent behavior found in decentralized systems. Rather than enforcing hard top-down commands or relying solely on open-ended negotiation, hybrid approaches typically deploy lightweight coordinator, suggestion systems, or hierarchical role assignment to scaffold collaboration while preserving local autonomy.

In the Minecraft domain, hybrid coordination is especially attractive due to the platform's open-endedness, spatial complexity, and diverse task granularity. A single build operation may require macro-level task decomposition (e.g., material gathering, area clearing, structure placement) and micro-level adaptability (e.g., navigating obstacles, compensating for missing tools, reacting to other agents' progress). Hybrid systems allow for centralized planners or coordinator agents to assign broad goals, while leaving implementation details and temporal control to autonomous sub-agents.

One representative design pattern is the "soft coordinator", a central agent that does not issue direct commands but instead suggests high-level plans, synthesizes global memory, or arbitrates disputes. For instance, in hierarchical frameworks like *HAS* [Zhao et al., 2024] and *S-Agents* [Chen et al., 2024], the coordinator agent proposes task tokens or mission goals, which are interpreted, refined, or delegated by subordinate agents. These agents can dynamically reorganize themselves—splitting into subteams, escalating tasks, or reallocating roles—while remaining loosely aligned to the overarching objective.

Another variation is the use of coaching agents or non-binding supervisors, as seen in adaptive variants of *AutoGen* and *MindAgent*. These agents monitor progress, offer reflective suggestions, and maintain shared memory consistency, but they do not enforce execution. This allows teams of Minecraft AICs to operate semi-autonomously while still benefiting from guidance, cross-agent memory reconciliation, and coarse-grained coordination.

Hybrid methods can also involve temporal shifting of control. Some systems switch dynamically between centralized and decentralized modes depending on the task phase, risk level, or failure state. During planning or emergency recovery, a central planner may temporarily assume control to reroute objectives or resolve contention. Once stability is regained, control is handed back to distributed agents. This elasticity has been explored in collaborative benchmarks like *TeamCraft* [Long et al., 2024], where bandwidth constraints, task dependencies, and user intervention affect the optimal level of centralization at runtime.

The primary advantage of hybrid systems is their ability to scale while maintaining alignment. They retain global visibility and coordination potential without requiring a fully centralized bottleneck, and they preserve creative behavior and emergent dynamics without drifting into incoherence. However, they introduce new complexities: agents must learn to interpret suggestions, coordinators must calibrate influence without dominating, and failure recovery may require mixed-mode reasoning.

Hybrid coordination strategies thus represent a pragmatic and powerful design for multi-agent collaboration in Minecraft. They allow central oversight to guide behavior without constraining it, enabling coherent yet adaptive societies of AI characters. The next frontier involves formalizing hybrid influence protocols, training agents for fluid role switching, and developing benchmarks that reward mixed-initiative planning and execution under real-time constraints.

7.5 Discussions

Collaboration among multiple AICs hinges on effective communication—both in form and function. Protocol design determines how agents share goals, negotiate tasks, and reconcile their actions with peers. Traditional symbolic systems offer bandwidth efficiency, clarity, and programmatic structure, but often fall short in flexibility or expressive range. On the other hand, natural language communication allows for rich intent sharing, negotiation, and alignment with human teammates, but introduces ambiguity, interpretability challenges, and higher computational cost. Hybrid protocols, now increasingly adopted in frameworks, blend these paradigms—using symbolic channels for state synchronization and natural language for clarification, conflict resolution, or adaptive planning.

LLMs significantly elevate collaboration performance by serving not only as communicative interfaces but also as flexible reasoning engines. With prompt adaptation, few-shot task understanding, and dynamic memory integration, LLMs allow agents to interpret context, infer others' intentions, and revise plans based on social cues. They enable decentralized agents to discover roles, central planners to summarize complex world states, and hybrid mediators to provide just-in-time guidance. Moreover, LLMs support emergent behaviors—such as leadership shifts, social norm learning, or tool handoff protocols—by simulating intuitive human-like cognition at interaction scale. In dynamic worlds like Minecraft, this flexibility transforms scripted behavior into adaptive social intelligence.

As collaboration complexity increases, maintaining coherence becomes a defining challenge. AICs must not only coordinate outwardly but also align their internal states, goals, and beliefs to preserve intelligibility and consistency over time. This reveals a deep structural connection between collaboration and coherence: both rely on shared context, stable identity, and recursive model-of-mind reasoning. AICs that collaborate

well are often those that behave coherently—predictable enough to build trust, yet flexible enough to respond meaningfully to others.

Memory emerges as the critical substrate linking these capacities. Long-term memory allows agents to maintain identity, recall interactions, accumulate shared history, and refine expectations about teammates. Without memory, coordination becomes reactive and brittle; with it, agents can reflect, adapt, and generalize across time and social configurations. Whether centralized (as in planner logs), decentralized (via scratchpads or peer-to-peer beliefs), or hybrid (e.g., shared vector stores with scoped access), memory is the key to scaling multi-agent intelligence.

Ultimately, the path toward achieving the Embodied Turing Test—where AICs behave as believable, socially adept entities—requires both coherent solo behavior and collaborative group dynamics. These are not orthogonal objectives. Rather, they co-evolve through systems that integrate communication, reasoning, and memory. Collaboration, at its most human-like, is coherence made social—and memory is the bridge that makes it possible.

Chapter 8

Memory Mechanisms for AI Characters

8.1 Background: The Central Role of Memory

Memory is foundational to the coherence, continuity, and collaboration of AI characters in embodied environments. Just as human identity relies on the accumulation and interpretation of past experiences, AI characters must also construct and maintain a memory model that supports their goals, relationships, and evolving understanding of the world. In the Minecraft AI ecosystem, memory is not merely a storage device—it is an enabler of narrative consistency, situational awareness, and social credibility.

This becomes particularly salient in the context of the Embodied Turing Test, which posits that intelligent agents should not only act competently but also behave in ways that appear continuously human-like to external observers. Without memory, an AI character may excel at immediate tasks but fail to uphold beliefs, preferences, or relational commitments across time, thus breaking the illusion of personhood. Consequently, designing robust memory mechanisms is central to advancing agent-level intelligence in open-ended, multi-agent environments like Minecraft.

8.1.1 Biological Inspiration for Artificial Memory

Much of the conceptual groundwork for artificial memory mechanisms draws inspiration from biological systems—especially the human brain. In cognitive neuroscience, memory is classically divided into multiple systems, including *sensory memory*, *short-term (working) memory*, and *long-term memory*, with long-term memory further subdivided into *episodic*, *semantic*, and *procedural* memory [Tulving et al., 1972, Squire, 1992]. This layered architecture has influenced both symbolic cognitive models (e.g., ACT-R, Soar) and recent neural network-based systems seeking to emulate memory functions more dynamically [Anderson, 1996, Laird, 2019].

Sensory memory in biological systems provides a short-lived buffer for raw perceptual information—analogous to frame-level or observation-buffering mechanisms in embodied agents. *Working memory* enables real-time reasoning and manipulation of information over short durations, similar to the limited context window in LLMs, where only a recent segment of dialogue or perception is available to guide decision-making. *Long-term memory*, by contrast, involves durable, structured storage of experiences, facts, and skills; in AI, this corresponds to persistent databases or retrieval-augmented architectures capable of episodic recall.

Furthermore, memory in biological organisms is not a passive recording system but a reconstructive, interpretive process. Empirical findings indicate that human memory is shaped by factors such as attentional focus [Chun et al., 2011], emotional salience [McGaugh, 2004], goal relevance [Anderson et al., 2004], and prior knowledge structures (schemas) [Bartlett, 1995]. These mechanisms contribute to selective encoding

and retrieval, and they support flexible narrative construction rather than rigid playback.

Such selectivity and adaptability are increasingly mirrored in artificial agents. For instance, systems like *MemGPT* [Packer et al., 2023] and the *Reflection* agents of [Shinn et al., 2023] explicitly model salience-based memory filtering, prioritizing observations that are surprising, socially significant, or critical for task performance. Similarly, neural memory models often implement decay functions or update thresholds to manage bounded memory capacity in long-running agents.

Beyond structure, the function of biological memory highlights a key insight: memory is inseparable from agency. Memories are encoded not just because they are experienced, but because they are acted upon, revisited, and reinterpreted. For AI characters in Minecraft, this means memory must support dynamic interaction with the world—providing context for social commitments, recalling partially completed projects, and informing decision-making in a temporally extended narrative arc.

Thus, biologically inspired memory systems are not merely referential—they are integrative, reconstructive, and behaviorally grounded. Emulating these principles offers a pathway to more coherent, believable, and agentic artificial characters.

8.1.2 Practical Expectations for Stable Memory Mechanisms

A memory mechanism suitable for embodied AI agents—especially those embedded in rich, evolving worlds like Minecraft—must satisfy several functional and computational criteria. These criteria reflect both cognitive plausibility and system performance requirements, bridging the gap between biological inspiration and implementable design.

- **Persistence and Stability:** Memories must persist across time, tasks, and sessions to enable coherent longitudinal behavior. Agents should recall prior interactions, goals, and outcomes even after environmental resets or long durations of inactivity. This mirrors the human capacity for autobiographical recall [**?**], and is critical for supporting social continuity and task resumption in virtual worlds. Modern systems like *MemGPT* [Packer et al., 2023] and *Generative Agents* [Köhn et al., 2020] attempt to emulate this by maintaining explicit long-term memory stores across agent lifetimes.

- **Low Latency Access:** For memory to be operationally useful, access latency must be minimal. Retrieval operations should support real-time planning, dialogue, and interaction without introducing delays that disrupt immersion or responsiveness. Systems like *ReAct* [Yao et al., 2023b] and *Reflexion* [Shinn et al., 2023] have demonstrated lightweight mechanisms for dynamically retrieving contextually relevant memory chunks within token-constrained inference windows.

- **Semantic Relevance:** Retrieval should not rely solely on surface similarity but incorporate higher-order structure—temporal sequences, spatial proximity, causal links, and social relations. This aligns with findings in human cognition where semantic networks and spatial schemas influence recall [Tversky, 2003, Barsalou, 2008]. Recent memory-augmented systems increasingly leverage vector databases and embedding-based search to support such relevance-aware retrieval.

- **Selective Updating:** Memory systems must implement mechanisms to control what is stored and for how long. Human cognition naturally performs selective encoding and forgetting, favoring emotionally salient or goal-relevant information [Anderson and Green, 2001, McGaugh, 2004]. Artificial agents can benefit from analogous policies that summarize, cluster, or prune low-utility memories. Work on memory compression and decay policies in cognitive architectures (e.g., *ACT-R* [Anderson, 1996]) and scalable LLM agents *MemGPT* shows promise here.

- **Explainability:** The ability to inspect, interpret, and audit memory content is critical for ensuring trust in AI systems. Memory should be introspectively accessible to the agent and externally queryable by users or developers. This supports debugging, narrative transparency, and alignment with human expectations [Doshi-Velez and Kim, 2017, Chen et al., 2020]. Some agents use natural language

reflections or summaries to render internal states interpretable in real time [Shinn et al., 2023, Park et al., 2023].

These expectations define the operational boundaries of agent memory, balancing cognitive realism with engineering feasibility. In environments like Minecraft, where agents must navigate evolving goals, dynamic teams, and open-ended construction, the memory system is not merely an auxiliary module—it is the substrate of agency, social presence, and learning.

8.1.3 Theoretical Definition and Constraints

From a formal standpoint, we may define a memory mechanism $\mathcal{M}$ as a tuple (E, R, U, D), where:

- E is the **encoding function**, mapping raw experience—whether perceptual, linguistic, or internal—to a structured representation (e.g., vector embeddings, symbolic triples, or narrative frames). Effective encoding must preserve key semantics while discarding irrelevant noise. Techniques range from attention-weighted transformer states [Vaswani et al., 2017] to handcrafted production rules in cognitive architectures such as *ACT-R*.
- R is the **retrieval function**, which takes a query (explicit or implicit) and returns the most relevant past content. Retrieval mechanisms may use similarity search over vector embeddings [Lewis et al., 2020], structured queries over episodic graphs [Urbanek et al., 2019], or memory search policies conditioned on current goals [Shinn et al., 2023].
- U is the **update policy**, determining how new experience modifies existing memory. Updates may include reinforcement (e.g., salience-weighted promotion), consolidation (e.g., summarization), or contradiction resolution (e.g., belief revision). This aligns with theories of memory reconsolidation in neuroscience [Nadel and Moscovitch, 1997] and continual learning in AI [de Masson D'Autume et al., 2019].
- D is the **decay function**, modeling forgetting or down-prioritization over time. This is essential for bounding memory growth and mirroring human-like retention curves [Wixted, 2004]. Decay may be time-based, usage-based, or modulated by emotional or social salience.

A well-designed memory system must ensure that $\mathcal{M}$ satisfies several theoretical and operational constraints:

- **Consistency:** Retrieved memories should not contradict each other or the agent's current belief state. Cognitive dissonance is acceptable (and human-like), but systemic contradictions can erode agent reliability. This links to belief tracking and theory of mind in both cognitive science and multi-agent systems [Devin et al., 2017, Rabinowitz et al., 2018].
- **Boundedness:** Memory systems must respect computational limits. As argued in the bounded rationality framework [Simon, 1972], intelligent behavior arises not from perfect memory but from effective heuristics operating under resource constraints. Bounded memory encourages prioritization, abstraction, and the emergence of cognitive economy.
- **Contextualization:** Retrieval and interpretation must be conditioned on physical, temporal, and social context. For example, recalling who gave the agent a specific item should depend on whether that interaction occurred recently, during a cooperative task, or in conflict. Context-aware memory retrieval has been emphasized in models of grounded cognition [Barsalou, 2008] and situated agents [Brooks, 1991].

- **Compositionality:** Memory should support the synthesis of new insights, plans, or beliefs from existing components. This is crucial for generalization, creativity, and reflection. Recent LLM-based agents have explored memory composition in reflective prompting and self-dialogue mechanisms [Shinn et al., 2023, Packer et al., 2023].

These theoretical constraints reflect the integrative nature of memory: it is not merely an auxiliary module, but a structural backbone for perception, reasoning, learning, and social interaction. In the Minecraft AI context, $\mathcal{M}$ must enable agents to persist across quests, adapt to dynamic alliances, and construct personal histories—hallmarks of intelligent, believable characters.

8.2 Memory in LLM-Based Agents

In recent years, memory mechanisms for large language model (LLM) agents have evolved rapidly, driven by the need to manage long-horizon context, encode personalized behaviors, and simulate believable agents. Unlike static language models, agentic LLMs operate over time, often across multiple sessions, requiring memory to track experiences, update internal states, and maintain coherent behavior. This section outlines the dominant paradigms in memory system design for LLM agents.

8.2.1 Contextual Window Memory

The most basic memory mechanism in LLM agents is the use of a fixed-length *context window*, an inherent architectural feature of transformer-based models. In this approach, all necessary information—prior dialogue turns, environment observations, intermediate reasoning steps, or system-level instructions—is serialized into a textual prompt and fed into the model as a single input sequence. The model then generates a continuation conditioned on this flattened context.

This mechanism leverages the core strength of autoregressive LLMs: learning to predict the next token based on prior tokens. As such, it does not require any architectural modification or training beyond prompt engineering. However, it is constrained by a maximum token limit (e.g., 4K in GPT-3, up to 128K in Claude or Gemini), and thus scales poorly with long-horizon or multi-session tasks. Despite these limitations, contextual memory has proven surprisingly effective for agents operating in short episodes or well-scoped interactive loops. For example, the *ReAct* framework chains observations, thoughts, and actions in a structured prompt format:

```
User: What's the capital of France?
Thought: I should look this up in Wikipedia.
Action: search("capital of France")
Observation: Paris is the capital of France.
Final Answer: Paris
```

By explicitly writing intermediate reasoning steps into the context window, ReAct-style agents simulate deliberation and maintain coherence across a few interactions. This is especially useful for open-ended problem solving, planning, and tool use. A similar strategy is employed in agentic frameworks like *AutoGPT* [Gravitas, 2025] and *BabyAGI* [Nakajima, 2025], where memory is entirely prompt-based. These agents dynamically append task histories, objectives, and tool usage records into the prompt. For instance, in *AutoGPT*, the system prompt might include:

```
You are AutoGPT, an autonomous agent. Your goal is to build a website. Previous steps:
1. Thought: I should plan the page layout.
2. Action: use(design_tool).
3. Observation: Layout created successfully.
Next step:
```

In both cases, the context window acts as a working memory buffer, storing the current state of deliberation. However, as task complexity grows, this prompt grows linearly, eventually exceeding the model's token limit. At that point, older entries must be truncated or summarized—leading to memory loss unless additional external memory mechanisms are employed.

Recent efforts to extend context window size—such as transformer modifications (e.g., FlashAttention-2 [Dao, 2023]), memory-efficient position encodings (e.g., YaRN [Peng et al., 2023]), or sparse retrieval strategies [Liu et al., 2023]—have pushed these limits further, but the fundamental issue remains: the context window provides *volatile*, *session-bound* memory with no persistence between runs.

In sum, contextual window memory offers a lightweight, zero-shot-compatible strategy for modeling memory in LLM agents. It excels in reactive, tightly-scoped settings but requires augmentation for long-horizon coherence, lifelong learning, or socially embedded behavior—all essential for believable AI characters in embodied environments like Minecraft.

8.2.2 Retrieval-Augmented Memory

To overcome the limitations of fixed-size context windows, many LLM agents incorporate Retrieval-Augmented Memory (RAM) mechanisms. These systems decouple memory from the transformer's native input stream by maintaining an external memory store—often a vector database or semantic cache—that holds structured representations of prior experiences. At inference time, the agent issues a query to the memory store and retrieves the top-k most relevant entries, which are then injected into the model's context window.

Formally, retrieval-augmented memory consists of two core operations:*(i)* an `encoding function` that maps past experiences into fixed-dimensional vector embeddings, and *(ii)* a `retrieval function` that selects relevant entries given a query vector. Most implementations use pretrained or fine-tuned encoders (e.g., *Sentence-BERT* [Reimers and Gurevych, 2019], OpenAI's *Ada* [OpenAI, 2022]) and rely on cosine similarity for nearest neighbor search (e.g. RAG [Lewis et al., 2020]). High-performance systems often index memory with approximate nearest neighbor (ANN) libraries such as FAISS [Johnson et al., 2021], Weaviate [Weaviate Team, 2021], or Milvus [Milvus Team, 2020]. A typical RAM architecture for an LLM agent consists of:

- **Memory buffer:** A persistent store containing prior interactions, events, and experiences, often annotated with metadata (timestamps, tags, social context).
- **Embedding function:** A model that converts textual memory entries and queries into dense vectors (e.g., 768D or 1536D).
- **Retrieval logic:** A similarity metric (usually cosine) and an indexing algorithm (e.g., HNSW [Malkov and Yashunin, 2020], IVF [Jégou et al., 2011]) for selecting relevant memories.
- **Memory integration:** Retrieved memories are serialized into natural language (e.g., "Previously, you helped the player build a stone tower near the village") and prepended to the current prompt before calling the LLM.

One canonical application of RAM is illustrated in the work by [Zeng et al., 2024], where the authors investigate how different memory structures and retrieval strategies affect the performance of LLM-based agents. They construct agents that store structured memory traces—such as knowledge triples, summaries, or atomic facts—during task execution, and evaluate retrieval methods including single-step, reranked, and iterative retrieval. When an agent re-encounters a previously visited state in a virtual environment, these retrieval strategies allow it to recall relevant prior information (e.g., observed entities or attempted actions), even if those events occurred far outside the current context window. This design supports long-term coherence and enhances situational awareness in embodied or task-driven settings.

Another influential example is *LangGraph* [LangChain Team, 2024], a framework for constructing multi-agent workflows with persistent, modular memory. *LangGraph* allows agents to define per-function memory

stores (e.g., "search memory", "dialogue memory", "build memory") and provides built-in support for graph-based memory composition. In a Minecraft-like environment, this might allow one agent node to remember crafting recipes, while another recalls long-term alliances with NPCs.

Even mainstream LLM applications such as *AutoGPT* and *BabyAGI* adopt RAM by logging task chains and retrieving prior completions. When the task planner formulates a new goal, it queries the memory database for past sub-tasks of similar structure or semantics, improving efficiency and avoiding repetition.

A retrieval-augmented LLM agent in Minecraft might implement the following logic:

```
[Query]: What should I do next in the iron smelting project?
⇓
[Retrieve]: You previously placed 6 furnaces near the river. You planned to collect more
coal before starting the smelting process.
⇓
[Augmented Prompt]: Include memory retrieval in LLM context
⇒ [Response]: I will now gather coal to continue the smelting task.
```

This type of memory access simulates short-term episodic recall, akin to how humans remember incomplete tasks and contextualize ongoing plans based on past actions.

RAM systems allow agents to operate over much longer horizons without overloading the model's context window, and they enable personalized behavior by retaining structured user-specific histories. However, retrieval quality is highly sensitive to embedding fidelity and index structure. Moreover, RAM does not inherently model memory dynamics such as salience, decay, or contradiction resolution—it is primarily a storage-and-fetch paradigm unless augmented with higher-level reasoning modules.

Nonetheless, retrieval-augmented memory provides a strong foundation for scalable, interpretable, and persistent cognition in LLM agents. It forms the backbone of many state-of-the-art planning agents and serves as a bridge toward more structured cognitive architectures.

8.2.3 Structured Long-Term Memory

Beyond raw retrieval, advanced LLM agents increasingly adopt Structured Long-Term Memory (SLTM) architectures that categorize and manage memories based on their cognitive role. Inspired by models of human memory in cognitive science, these systems differentiate memory into `episodic` (events and experiences), `semantic` (facts and rules), and `reflective` (self-assessments and goals). This type of memory structuring originates in symbolic architectures such as *ACT-R* [Anderson, 1996] and *Soar* [Laird, 2019], where declarative and procedural memory are handled through distinct storage and retrieval subsystems. In practice, structured memory is implemented as a typed memory buffer, often backed by a database or key-value store. Each memory unit is associated with metadata fields such as: *memory type* - episodic, semantic, or reflective; *timestamp and context* - when and where the memory was encoded; *salience score* - relevance or emotional intensity, which influences decay; and *tags and entities* - named entities, task references, or goals mentioned.

Retrieval can then be constrained or weighted based on memory type and query intent. For example, a planning task may request only `(semantic + recent)` memories related to crafting skills, while a reflection routine may search for `(episodic + high-salience)` failures or surprises. The *MemGPT* system offers a representative example of SLTM in action. In *MemGPT*, the agent maintains multiple memory streams:

- **Episodic memory** stores structured logs of interactions, including who, what, where, and when, formatted for both embedding retrieval and natural-language replay.
- **Semantic memory** is built from extracted knowledge statements, often compressed into symbolic triples or paraphrased summaries (e.g., "coal is required for smelting").
- **Reflective memory** contains self-generated reflections and narrative updates, like "I forgot to bring torches, which made mining dangerous".

The system dynamically triggers *consolidation* and *summarization* tasks when context length is exceeded or after episodes complete. A memory management loop selects which entries to keep, compress, or forget, based on recency, salience, and task relevance. Consider a Minecraft AIC tasked with multi-day construction of a fortified village. Using SLTM, the agent might exhibit the following behavior:

- **Episodic:** "On day 3, I began building the outer wall but was interrupted by a zombie raid."
- **Semantic:** "Cobblestone is the optimal material for durability against mobs."
- **Reflective:** "I need to reinforce the perimeter earlier in future projects."

When re-engaging with the project after a session break, the agent can retrieve its architectural plans from semantic memory, recall interruption patterns from episodic logs, and incorporate lessons learned from its own reflection. This leads to more believable, narratively grounded behavior that reflects continuity over time. Recent work has also emphasized *memory compression*—for example, via semantic chunking and retrieval-based summarization as in Liu et al. [2024a]—and **usage-aware memory control**, such as segment-level recurrence and attention gating used in Recurrent Memory Transformer [Bulatov et al., 2022]. These techniques allow structured memory to remain bounded while preserving long-term contextual relevance and behavioral continuity.

Structured long-term memory is a crucial component for agents that operate in persistent, dynamic environments. It enables internal differentiation between task knowledge, personal history, and learned behavior. By mimicking the organization of human memory systems, SLTM enables more believable, adaptive, and self-aware AI characters—a key objective for embodied agents in systems like Minecraft AI.

8.2.4 Reflective and Self-Modeling Memory

Beyond static storage and retrieval, recent LLM-based agents increasingly adopt *reflective* and *self-modeling* memory mechanisms. These go beyond traditional memory paradigms by introducing meta-cognition: the agent's capacity to evaluate its own performance, form internal beliefs, revise intentions, and narratively reason about its identity or history. This shift is foundational for long-term coherence, character believability, and adaptive learning in interactive environments. Reflective memory systems typically implement three interlinked capabilities:

- **Self-observation:** Agents monitor and log their own actions, including failures, inefficiencies, or goal deviations.
- **Self-critique and revision:** Agents analyze their behavior post hoc to generate reflective summaries (e.g., "I should have gathered wood before nightfall.")
- **Self-model updates:** Agents form and revise internal traits or role-based beliefs, such as "I am cautious" or "I prioritize group safety."

These capabilities are implemented via natural language updates to an internal memory buffer, stored either as raw reflective entries or structured into a self-model schema.

The *Reflexion* framework [Shinn et al., 2023] provides a clear architecture for this approach. After executing a task, an agent reflects on its performance by generating a textual critique (e.g., "I failed to find the key because I ignored the earlier hint about the bookcase"). This critique is then used to adjust future behavior—by reformulating plans, reprioritizing goals, or setting conditional reminders (e.g., "Check hints carefully before moving on").

In *Reflexion*, this loop serves as a verbal reinforcement learning cycle: reflection is treated as an intermediate supervision step between attempts, leading to cumulative performance gains across episodes

without gradient updates. The system has shown improvements on complex reasoning tasks, including *HotPotQA* [Yang et al., 2018] and *ALFWorld* [Shridhar et al., 2021].

The *Generative Agents* framework [Park et al., 2023] takes self-modeling further by building a full autobiographical memory system. In this setting, agents autonomously generate thoughts, summarize experiences, and revise internal traits through repeated reflection. For instance, an agent might produce thoughts such as:

```
I enjoyed helping my neighbor build their house.
I've been getting better at mining without getting lost.
I feel more confident when I stick to a plan.
```

These thoughts are later retrieved and abstracted into personality summaries (e.g., "is helpful", "values planning"), which influence future actions and dialogue. When asked about their preferences or goals, agents respond in ways consistent with this emergent personality model. This creates rich, dynamic characters that evolve based on lived experience. A reflective Minecraft agent might exhibit the following behaviors:

- After being attacked while gathering wood at night, the agent records: "Next time I should build a shelter before dark".
- Upon successfully completing a large construction task, it reflects: "Working in stages helped me stay organized".
- Over time, it accumulates beliefs like: " prefer teamwork" or "I avoid risky caves unless I have armor".

Such reflections can be stored as explicit memory entries or abstracted into behavior-modulating rules that shape long-term planning. This enables Minecraft agents to exhibit growth, personality, and narrative arcs—key components for passing the Embodied Turing Test. The reflective/self-modeling paradigm connects to several broader areas:

- **Cognitive architectures:** Meta-cognitive modules have long been modeled in systems like *Meta-AQUA* [Cox, 2005], which use reflection to guide learning and self-repair.
- **Behavioral cloning and critique:** Recent LLM agents have integrated human feedback loops that simulate reflection, e.g., *ReAct* with critique.
- **Character-centric simulations:** Simulation environments increasingly require agents that can behave in psychologically plausible ways over long spans of time, e.g. *Generative Agents*.

Reflective and self-modeling memory allows agents to go beyond recall—to monitor, evaluate, and refine their behavior. These systems support narrative selfhood, emotional plausibility, and continual behavioral adaptation. As LLM agents grow more autonomous and persistent, such meta-cognitive layers will be essential for producing credible, expressive, and socially grounded AI characters.

8.2.5 Summary and Outlook

Together, these memory systems form a flexible design space for LLM-based agents. Contextual memory ensures real-time responsiveness; retrieval-augmented memory enables recall over time; structured memory supports narrative integrity; and reflective memory gives rise to self-coherence. However, most of these implementations remain constrained to text-only domains, with limited embodiment, spatial reasoning, or multi-agent social memory.

In the Minecraft AI ecosystem, adapting these paradigms to grounded, persistent, and socially situated contexts poses a rich challenge. The next sections will examine how these memory mechanisms can be extended to support the needs of embodied agents operating in dynamic virtual worlds.

8.3 Memory in Embodied Game Agents

In game-based settings like Minecraft, memory must extend beyond linguistic history to encode spatial, social, and task-oriented knowledge in a persistent and actionable way. AI characters must remember not only what was said or seen, but also what was built, where resources were found, which agents are allies, and what strategies have succeeded in the past. Within the Minecraft AI framework, we distinguish several layers of memory mechanisms:

- **Perceptual Memory:** Buffering recent observations of the environment—e.g., block states, entity positions, inventory changes. This forms the sensorimotor basis of memory.
- **Episodic Memory:** Recording and indexing sequences of actions and events, often anchored to timestamps, spatial locations, or collaborators. These memories allow the agent to re-trace past missions or explain prior choices.
- **Semantic and Task Memory:** Encoding abstract task goals, partial plans, and learned schemas for crafting, building, or survival. These memories evolve as agents generalize from experience.
- **Social Memory:** Tracking interactions with specific players or agents, including reputations, alliances, and shared commitments. This supports collaboration, trust modeling, and socially coherent dialogue.
- **Reflective Memory:** Capturing agent-generated reflections, such as goal evaluations, emotional states, or narrative summaries. This enables agents to appear introspective or purposeful over time.

Such memory systems must operate under constraints of real-time reasoning, partial observability, and ongoing world change. Moreover, unlike in static text environments, Minecraft agents may need to act even while memories are being updated, retrieved, or interpreted—making memory not only a representational challenge but also an operational one.

8.3.1 Implemented Architectures and Patterns

In practical terms, embodied memory in Minecraft AI agents is often implemented as a modular system consisting of structured databases, vector stores, and dynamic event logs, each mapped to specific layers of cognition. These memory subsystems must operate under constraints of latency, partial observability, and non-stationarity—making implementation choices critical for maintaining coherence, adaptability, and explainability in long-lived agents.

- **Perceptual Buffers** are frequently realized as ring buffers or sliding windows of frame-wise observations, often limited to the past N frames. This allows agents to track short-term environmental dynamics—such as moving entities, block changes, or fluid flow—needed for tactical behaviors like evasion, path-following, or reactive building. Projects like *MineDojo* [Fan et al., 2022] and *Voyager* [Wang et al., 2023a] maintain low-latency access to local block and entity states for goal-conditioned control.
- **Episodic Memory Systems** are typically implemented using timestamped logs, graph-structured event chains, or language-based episodic summaries. For instance, the *VillagerAgent* project [Dong et al., 2024] encodes multi-turn dialogues and inventory changes as event sequences tied to player identities and time markers, supporting behavior explanation and role persistence. Similarly, in *TeamCraft* [Long et al., 2024], agents use structured mission logs to coordinate asynchronously and reconstruct team-level decision trajectories.
- **Task Memory** commonly takes the form of hierarchical plan representations—often tree-like structures or symbolic key-value maps that associate goals with partial completions. In *MinePlanner* [Hill et al., 2024], temporally abstracted plans enable agents to track dependencies between tasks such

as mining, smelting, and crafting, allowing for re-planning upon failure, dynamic interruption, or environmental changes.

- **Spatial-Semantic Maps** serve to bind perceptual observations to grounded concepts. Implementations vary from voxel-centric world graphs to concept-tagged landmarks stored in a semantic database. For example, agents may remember that a crafting table at (x, y, z) is the team's shared workspace, or that a ravine near base camp contains high-value ores. These mappings enable context-aware referencing, spatial reasoning, and consistent localization across memory updates.
- **Social Memory Modules** track inter-agent relationships, past conversations, trust levels, and cooperative roles. These modules may include simple counters (e.g., "blocks mined together") or more complex embeddings derived from dialogue and joint action histories. Social memory is key for establishing alliances, attributing reputations, and sustaining believable long-term relationships in multi-agent simulations like TeamCraft [Long et al., 2024] or Showrunner [Park et al., 2023].
- **Reflective Logs and Self-Narratives** are often implemented via agent-generated textual summaries, modeled after human journaling or self-talk. These reflections can capture emotional valence, task evaluation, or narrative interpretation. Inspired by the *Reflexion* architecture [Shinn et al., 2023], several Minecraft agents now include self-review phases triggered by failure, goal completion, or user query. This enables richer forms of memory retrieval, post hoc explanation, and internal goal adjustment.

Ultimately, these memory mechanisms form the substrate for continuity, accountability, and autonomy in Minecraft AI characters. As agents move toward persistent identities and evolving social lives, memory architecture becomes a first-class design element—not a peripheral utility.

8.3.2 Challenges and Trade-offs

Designing memory systems in embodied virtual environments introduces a range of architectural and operational trade-offs, many of which reflect broader tensions between scalability, reactivity, interpretability, and continuity of experience. These challenges are especially pronounced in environments like Minecraft, where agents engage in temporally extended, physically grounded, and socially embedded tasks.

- **Memory Saturation:** Open-ended environments like Minecraft produce high-volume, high-entropy data streams. Without memory regulation, systems quickly exceed storage and retrieval limits. Agents must therefore implement salience filters, decay heuristics, or abstraction mechanisms. For example, *Voyager* periodically summarizes task trajectories into concise goal–result pairs, while *Reflexion* limits episodic memory to only unexpected or reflection-worthy events.
- **Interleaved Execution:** Embodied agents must often plan, act, and reflect concurrently, especially in multiplayer or real-time settings. Blocking memory access or update operations risks breaking the illusion of continuous intelligence. To address this, some systems decouple perception and memory subsystems via event queues and background summarization threads. Asynchronous memory architectures also allow agents to continue interacting with the world while memories are written, consolidated, or pruned.
- **Grounding and Generalization:** Agents must ground memory content in the physical world while generalizing across changing contexts. For instance, if a landmark is destroyed or a teammate changes roles, memories must update accordingly without becoming inconsistent or obsolete. *MineDojo* and *MinePlanner* highlight this challenge by requiring agents to reason about abstract task concepts like "resource depot" or "base camp," which may move or evolve over time. Grounding strategies include spatial anchors, co-referenced world models, and ontological linking of entities to task roles.

- **Agent-Accessible Recall:** For memory to influence behavior, agents must retrieve relevant content in a form compatible with planning and reasoning subsystems. This often necessitates a bridge between low-level perceptual logs and high-level symbolic summaries. Hybrid architectures like *ReAct* and MetaGPT [Hong et al., 2024] demonstrate the power of using natural language as an intermediate memory representation—supporting both human-aligned interpretability and LLM-based planning. However, these approaches also risk abstraction loss if underlying observations are over-summarized.
- **Integration Challenges:** Embodied memory must interoperate with perception, planning, and communication modules. This involves trade-offs between storage complexity, query latency, and cross-module alignment. Embedding-based retrieval (e.g., via FAISS [Johnson et al., 2021] or Qdrant [Qdrant Team, 2025]) enables fuzzy search and scalable access, but symbolic overlays (e.g., RDF graphs [World Wide Web Consortium (W3C), 2025], JSON-based schemas) are often needed for fine-grained querying, causality tracking, or logic-based reasoning. Systems like *CAMEL* [Li et al., 2023] and *Generative Agents* leverage such hybrid representations to combine fast retrieval with structured introspection.

These challenges underscore that memory is not merely a passive record of prior inputs but an active, adaptive structure that mediates learning, behavior, and identity. As AI characters in Minecraft move toward greater autonomy, sociality, and narrative continuity, memory will become a bottleneck or bottleneck-breaker—determining whether agents can meaningfully evolve over time or remain reactive short-term automatons.

8.4 Toolkits and Productions

In virtual environments like Minecraft, building AI characters that exhibit narrative coherence, social continuity, and long-term goal pursuit demands robust memory infrastructures. Recent advances in agent development toolkits have lowered the barrier for implementing such capabilities by embedding memory management—semantic indexing, summarization, and gated retrieval—into the agent's control loop. These toolkits serve as architectural blueprints for translating abstract memory models into concrete, operational pipelines.

The *LangChain* framework provides a modular memory API tailored for composable agents. Its `Memory` primitives—such as `BufferMemory`, `SummaryMemory`, and `KGMemory`—enable integration of short-term recall, long-term abstraction, and structured symbolic knowledge within a single agent process [LangChain Developers, 2024]. In the context of Minecraft, such layered memory architectures allow agents to remember recent sensory events (e.g., block placements), persist high-level mission goals, and track evolving social dynamics (e.g., team formation). LangChain's backend-agnostic support for vector stores (e.g., Pinecone, Weaviate) further enables scalable memory implementations across multiplayer and persistent worlds.

LlamaIndex [LlamaIndex Team, 2024] enhances the memory infrastructure by supporting persistent storage layers (`StorageContext`) and intelligent routing via `SelectiveContext`. These features are particularly beneficial for Minecraft agents engaged in extended planning or construction tasks, where the ability to distill episodic interactions into compact summaries helps maintain focus and reduce context overload. LlamaIndex's modular design allows agents to draw from multiple indexed sources (e.g., spatial logs, crafting schemas, social interactions) while staying within the context limits of transformer-based models.

The opt-in memory feature in OpenAI's ChatGPT [OpenAI, 2024a] demonstrates how a minimal memory policy—centered on user-consented storage, summarization, and gated retrieval—can improve consistency and user experience across sessions. Although this system is designed for dialogue agents, the principles carry over directly to virtual agents in Minecraft: persistent memory buffers prevent identity resets, gated retrieval allows context-aware action selection, and user-overridable memories foster transparency and trust.

Cloud-native vector stores such as **Pinecone** [Pinecone Systems, Inc., 2025], *Weaviate* [Weaviate Team, 2021], and *Qdrant* [Qdrant Team, 2025] provide robust backends for retrieval-augmented memory. Their key advantage for embodied agents lies in efficient high-dimensional search over multimodal observations—such as summaries of locations, items, or events. When integrated with agents in sandbox environments

like Minecraft, these systems support memory architectures that recall strategic sites (e.g., a hidden mine), previously seen players, or partially completed structures across episodes.

Microsoft's *Semantic Kernel* treats memory as a first-class concept. Its `SKMemory` system supports retrospective and prospective memory filters, allowing Minecraft agents to selectively retain milestones (e.g., "first nether portal built") while discarding ephemeral interactions [Microsoft, 2025]. This granularity is crucial for managing memory load in dynamic, open-ended environments, and mirrors human memory strategies like selective encoding and rehearsal.

These toolkits embody a growing consensus: intelligent behavior in virtual worlds depends on persistent, interpretable, and context-aware memory. For AI characters in Minecraft, memory modules are not optional add-ons—they are essential mechanisms for tracking spatial and temporal continuity across builds and missions; maintaining social relationships and role commitments; learning from prior successes and failures via reflective summaries; and preserving identity and narrative arcs over long horizons.

Toolkits like *LangChain*, *LlamaIndex*, and *Semantic Kernel* encapsulate these patterns as reusable, declarative interfaces—making it feasible to deploy multi-session, multi-agent, and goal-driven characters in open-ended environments. By integrating these capabilities, designers can move beyond stateless command bots toward agents with memory-enabled agency—capable of developing stories, strategies, and social personas that persist.

8.5 Evaluations and Benchmarks

While memory plays a critical role in the cognitive fidelity of AI characters, evaluating the performance of memory mechanisms in LLM-based agents remains an open and multidimensional challenge. Traditional language model benchmarks—such as those for QA, summarization, or reasoning—often assume a static, one-shot context and do not stress-test the continuity, adaptability, or social integration of memory that are crucial for long-lived, embodied agents.

Recent efforts have begun to fill this gap. *LoCoMo* [Maharana et al., 2024] introduces a benchmark for evaluating long-term conversational memory, testing agents' ability to maintain coherence and recall over multi-session dialogues. *MemBench* [He et al., 2024] systematically evaluates factual retention and retrieval in LLMs across extended interactions. Additionally, *HELM* [Liang et al., 2022] proposes a holistic evaluation framework that includes long-term consistency as a sub-metric. However, these evaluations remain primarily linguistic and fail to assess whether memory supports coherent, situated behavior in grounded and interactive environments.

Minecraft, with its persistent world state, rich spatial dynamics, and social multiplayer affordances, provides a uniquely well-suited environment for embodied memory evaluation. Yet, few benchmarks explicitly isolate memory competence. For example, MineDojo offers a broad set of tasks involving navigation, manipulation, and crafting, where episodic and semantic memory are implicitly required, but not directly measured. *MinePlanner* [Hill et al., 2024] presents long-horizon planning benchmarks, where agents must maintain consistency across multiple steps, though memory errors are not diagnostically separated from planning failures.

Recent efforts like *TeamCraft* and *VillagerAgent* introduce multi-agent collaboration and long-term role play, where social and reflective memory become central. These environments stress-test aspects of trust modeling, goal persistence, and inter-agent referencing over extended sessions. However, these benchmarks still lack a fine-grained memory audit protocol.

To explicitly benchmark memory mechanisms for Minecraft-based agents, we propose a more structured and diagnostic evaluation framework with the following dimensions:

- **Temporal Consistency Tests:** Does the agent maintain beliefs, plans, or affiliations across time—e.g., revisiting an abandoned project or continuing a multi-step collaboration from yesterday?
- **Episodic Recall Tasks:** Can the agent narrate prior sequences of events accurately, particularly those involving temporally distant causes and effects?

- **Cross-Episode Memory Transfer:** Are learned concepts or relationships retained across multiple game sessions or environment resets?
- **Salience and Forgetting Dynamics:** Can the agent prioritize relevant memories while gracefully forgetting outdated or irrelevant experiences?
- **Counterfactual and Reflective Reasoning:** Can the agent explain failed attempts, revise plans, or refer to alternate strategies based on past experience?

Beyond performance metrics, we argue that explainability and narrative coherence should be treated as primary evaluation goals. Memory competence is best demonstrated not merely through successful recall but via consistent, interpretable behavior situated in the world. For example, an agent should be able to justify its decision to abandon a tower-building mission by referencing a past resource shortage or team conflict. Memory audits that combine behavioral probes with explicit narrative reconstruction—e.g., "What happened last week?"—can yield stronger evidence of agent-level continuity than retrieval accuracy alone.

In designing such benchmarks, inspiration can be drawn from cognitive psychology, where episodic and semantic memory are tested not only by factual reproduction but by the ability to contextualize and act upon remembered information [Tulving, 1983, Squire, 1992]. Minecraft-based memory benchmarks have the potential to bridge the gap between raw information retention and real-time memory-driven agency in open-ended worlds.

8.6 Discussion

The emergence of LLMs has fundamentally reshaped how memory is conceptualized, implemented, and utilized in the design of intelligent agents. Traditional memory architectures in game AI—often rigid, rule-based, and statically partitioned—were limited by fixed schemas and handcrafted logic. In contrast, LLM-driven agents operate within a more fluid paradigm, where memory is both a substrate for recall and an interface for reasoning, abstraction, and narrative synthesis.

At a practical level, LLMs have made it feasible to treat memory not as an external database but as a generative, queryable structure embedded in the agent's own language. This enables forms of memory that are inherently more interpretable, more compressible, and more flexibly scoped. Systems like *Reflexion* and *Voyager* demonstrate how agents can use language to reflect on their actions, summarize episodic experiences, and plan in light of remembered outcomes—all using prompts, rather than rigid retrieval APIs. This language-native representation opens the door to hybrid memory mechanisms that blend structured stores (e.g., vector or graph databases) with self-summarizing narratives.

From a cognitive perspective, LLM-based agents shift the focus from "storage fidelity" to "functional coherence". Rather than requiring exact replay of past events, LLMs support the synthesis of generalized or abstracted memories that remain behaviorally relevant. This supports a kind of "flexible forgetting", akin to human episodic compression and schematic recall. In environments like Minecraft, this means an agent may not remember every detail of a building sequence but can still explain the high-level strategy or social rationale behind a decision.

Moreover, LLMs enable forms of introspective memory previously infeasible at scale. Agents can now generate internal reflections ("Why did I fail this mission?"), engage in counterfactuals ("What if I had chosen a different teammate?"), and update their own goals or values based on prior outcomes. These capabilities were once hallmarks of symbolic cognitive architectures like *Soar* or *ACT-R*, but are now emerging through prompt engineering and memory routing in LLM systems.

Critically, the integration of LLMs into memory pipelines also blurs the boundary between memory and reasoning. Retrieval is no longer just about finding the right record—it's about synthesizing a coherent belief state or narrative from fragments. This has significant implications for coherence, continuity, and explainability in AI characters. For instance, an agent like Lucy in the Minecraft AI ecosystem can use

reflective summaries not only to justify past decisions to a human player but also to align future actions with an evolving sense of identity and history.

That said, LLM-based memory systems still face open challenges. These include controlling hallucination in self-generated memories, aligning retrieved content with the agent's factual world model, and maintaining long-term memory persistence across environment resets or model updates. Nonetheless, the trajectory is clear: LLMs are transforming memory from a static record-keeping function into a dynamic, language-mediated process at the heart of agent cognition.

In summary, the introduction of LLMs into embodied AI agents marks a paradigm shift in memory design. By enabling flexible, interpretable, and self-referential memory mechanisms, LLMs bring us closer to the vision of agents that not only act intelligently, but also remember purposefully, explain coherently, and evolve meaningfully over time.

Chapter 9

LLM-Based Embodied AI

9.1 LLMs as the Backend of Embodied Agents

Large Language Models (LLMs) such as DeepSeek [DeepSeek AI, 2024], Llama [Meta AI, 2024], GPT[OpenAI, 2023], Claude [Anthropic, 2024], and Gemini [Google, 2024] form the backend of modern embodied AI systems in virtual environments like Minecraft. Within the Minecraft AI framework, LLMs play a central role across multiple core cognitive functions: memory management, action decision-making, reflective reasoning, and dynamic coding. These models allow AI Characters (AICs) to reason flexibly, adapt to novel situations, and interact naturally with humans and other agents—all within a dynamic and partially observable world.

Without LLMs, embodied agents are limited to predefined rules or handcrafted decision trees. With LLMs, however, these agents become capable of emergent behaviors and context-aware responses, closely mirroring human-like intelligence. In particular, the application of LLMs to four components of MARC architecture enables powerful capabilities:

- **Memory:** LLMs interpret, summarize, and retrieve episodic and semantic memories, allowing AICs to reflect on past experiences, recall prior goals, and maintain narrative continuity over time.
- **Action:** Given the current perception and internal state, LLMs evaluate possible actions and select the most appropriate one using prompt-based reasoning frameworks, such as those inspired by the Model Context Protocol (MCP). This allows agents to operate in complex, multi-step environments without rigid scripts.
- **Reflection:** AICs periodically engage in reflective reasoning, synthesizing past events into higher-level insights and intentions. LLMs enable this capability by generating structured reflections that can influence future plans or update internal beliefs.
- **Coding:** In some frameworks (e.g., Voyager, MINDcraft, and Minecraft AI), LLMs are used to dynamically generate executable code, define new skills, or modify behaviors in real time. This allows AICs to self-improve, expand their capabilities, and adapt to tasks beyond their original programming.

Through these mechanisms, LLMs transform static virtual bots into dynamic, self-guided cognitive agents. As the field continues to evolve, deeper integration between LLMs and embodied frameworks will be key to developing intelligent systems capable of human-level interaction, adaptation, and collaboration in virtual worlds.

9.2 Memory: Generating Contextual and Long-Term Narratives with LLMs

Within the Minecraft AI framework, LLMs serve as the primary engine for memory summarization. Rather than feeding long raw logs directly into the LLM context (which quickly exceeds token limits), the system maintains two key types of high-level summaries:

- **General-purpose memory summary:** Periodically generated from recent interaction history, this compact summary captures the agent's recent experiences, current activities, and notable interactions. It ensures continuity and behavioral coherence during short- to mid-term decision-making.
- **Long-term thinking summary:** A higher-order reflective layer, this summary encodes the agent's long-standing goals, personal beliefs, and evolving preferences. It serves as a stable backbone across sessions and provides strategic guidance during planning or decision conflicts.

At runtime, both types of memory summaries are used when calling LLM to select action or perform reflection, ensuring that immediate choices are made in light of both recent experience and persistent intentions. In Prompt 9.1, it shows an example.

Update memory summary and long-term thinking.

You are an AI assistant helping to summarize memory records for a Minecraft bot. Your goal is to produce a new summary that captures key facts and useful context for the bot's future planning. When generating the memory summary, you should also consider the personality profile and the long-term thinkings of the bot.

Given:
1. A list of history records, representing recent events and messages (including user input and in-game events);
2. An old memory summary that contains previously known information;
3. The profile of the bot;
4. The long-term thinking of the bot.

Please:
1. Generate a new summary that preserves important past information while integrating new, relevant updates from the recent history. Pay special attention to the following:
- If there are any task requirements – including newly assigned tasks from other players or ongoing tasks that have not yet been completed – be sure to include them in the summary. This helps the bot stay aware of its current objectives and prevents forgetting unfinished work.
2. Generate a update of the longterm thinking, if necessary. Pay attention to the following:
- Only make changes if significant events have occurred that meaningfully impact the agent's behavior patterns or values;
- The long-term thinking should remain relatively stable over time and reflect high-level intentions (e.g., a preference for collaboration, caution, or efficiency);
- Avoid overfitting to short-term fluctuations or isolated incidents.

Prompt 9.1: Update memory summary and long-term thinking.

To ensure that the LLM can generate accurate and coherent memory summaries, the prompt is constructed using four key contextual components. These include:

- a list of recent **history records**, which capture the AIC's latest messages, actions, and observations;
- the **previous memory summary**, which serves as a condensed record of prior relevant activities and helps preserve temporal continuity across summarization iterations;
- the **AIC profile**, which encodes identity-defining attributes such as name, personality traits, role, and behavioral tendencies; and
- the AIC's current **long-term thinking**, a high-level narrative representing its overarching goals, preferences, and evolving self-concept.

These inputs are bundled into a single prompt and used as the full context window for LLM-based memory reasoning. Furthermore, to enable reliable downstream parsing and integration into the AIC's memory store, the LLM is instructed to return results in a standardized structure.

This LLM-powered summarization pipeline transforms raw interaction data into narrative memory artifacts that guide the agent's future behavior. The use of structured input and output protocols ensures consistency, relevance, and interpretability—while enabling memory evolution in response to dynamic gameplay and social contexts.

To better scale memory usage and enable more context-sensitive recall, an extended topic-based memory system can be applied, where several summary pieces with various themes or topics are maintained. These topic summaries capture different dimensions of the AIC's life, social connections, and mission progress. Examples might include:

- `[Friendship with Lucy]`: recording shared goals, emotional tones, and key dialogue exchanges with a specific character.
- `[Conflict with Zombies]`: capturing learned defense strategies, threats encountered, and consequences of past battles.
- `[Village Construction Project]`: summarizing building progress, material requirements, and unresolved tasks.

The number of topic summaries is configurable, and a relevance-based RAG filter determines which topics are retrieved and fed into the LLM prompt during planning, action selection, or self-reflection.

To maintain high-quality topic-based memory over time, the LLM must go beyond simply updating existing summaries with new information. It should also make higher-order decisions that reflect changes in the agent's narrative structure. This includes detecting which topics are affected by recent events, determining whether certain themes have become outdated or redundant, and identifying when entirely new topics should be introduced to reflect emerging patterns in the agent's experiences.

Such operations involve identifying topic relevance based on the current context, merging overlapping topics to reduce redundancy, and pruning stale or insignificant ones to avoid memory bloat. When novel, persistent threads of activity arise—such as a new social bond, long-term mission, or environmental shift—the LLM must autonomously instantiate new topic summaries to capture these developments. This adaptive memory shaping allows agents to manage growing histories without sacrificing relevance or coherence.

To support this, the prompt must include guiding instructions—informing the LLM how to interpret history contextually, match it with known topics, and judge whether new thematic summaries are warranted. For instance, if the agent begins recurring interactions with a new player character or enters a novel phase of exploration (e.g., moving from forest to mountain biome), the LLM should recognize and instantiate a relevant topic memory slot.

9.3 Action Selection: Contextual Decision-Making with LLMs

In AIC, the core of autonomous behavior lies in the ability to select the most appropriate next action based on current status, dialogue input, memory, and environment. Unlike rule-based systems that rely on rigid pipelines, LLMs as adaptive decision engines—capable of interpreting context and responding dynamically to both structured objectives and informal player interactions.

LLMs are invoked at key decision points to determine what the agent should do next, balancing goal fulfillment, player alignment, and ongoing narrative coherence. The prompt design reflects not only the current game state, but also the agent's evolving memory and possible action repertoire. Importantly, this prompt-based architecture allows for natural extensibility: new skills and behaviors can be added without retraining the model, simply by augmenting the set of available actions. This approach supports the following capabilities:

- **Context-sensitive planning:** The LLM can prioritize different actions based on the evolving environment, user interactions, and internal goals.
- **Goal-driven behavior:** The action selection pipeline integrates memory, task status, and prior actions to maintain continuity and direction.
- **Grounded language understanding:** Dialogue from human players is treated as dynamic instructions, allowing fluid transitions between verbal commands and physical actions.
- **Extensibility and modularity:** New types of action—such as constructing structures, offering trade, or triggering group behavior—can be appended without architectural changes, merely by defining new entries in the available action list.

Overall, this LLM-driven action framework gives rise to Minecraft agents that feel intentional, responsive, and consistent—key properties for passing the Embodied Turing Test. It is through prompts like Prompt 9.2 that language models move beyond symbolic reasoning and into grounded, embodied intelligence.

The set of "Available Actions" provided to the LLM during action selection follows a structured format inspired by the MCP. Each action is defined as a Dictionary-like object that includes: `name`, `description`, `params`, and `perform`, as shown in Action 9.1.

This structured format makes the action set interpretable to both LLMs and human developers. It also supports modular extension: new skills or plugins can register new actions with minimal overhead, allowing Minecraft AI agents to grow in capability over time.

As the number of available actions increases—especially when dynamic plugins, tools, or contextual behaviors are involved—it becomes impractical to inject the entire action list into the LLM prompt. To address this, RAG can be employed to filter and surface only the most relevant actions.

9.4 Reflection: Task Reassessment and Long-Term Intent Continuation

In addition to responding to immediate instructions or environment cues, intelligent agents must exhibit the capacity for self-driven behavior. Reflection is the mechanism by which an AIC pauses to assess its current state, review past tasks, and decide whether to resume long-term goals. This reflective loop enables autonomous continuity, reduces redundancy, and enhances the believability of the agent.

LLMs play a central role in this process by synthesizing memory traces, short-term status, and high-level personality traits into structured deliberation. The prompt shown in Prompt 9.3 provides an example of such a reflection instruction, which is usually structured around several sequential checks and reasoning heuristics:

Select the action to perform.

You are an AI assistant helping to plan the next action for a Minecraft bot. Based on the current status, memory, instruction, and the list of available actions, your task is to determine what the bot should do next.

You must output:
1. An updated status summary, if any change is needed. This should briefly describe the bot's current situation, goals, recent actions, and important outcomes that help track progress.
2. The next action for the bot to execute. Choose from the list of 'Available Actions', and provide appropriate parameter values based on the definitions.
3. A message to the player who sent the latest message, if necessary.

Important Guidelines:
1. Only update the status if new context or progress has occurred. Otherwise, set it to null.
2. Only generate an action if one is needed. Otherwise, set action to null.
3. You should consider the latest message with the highest priority when generating the action.
4. When choosing an action, consider all available actions and select the one that is most consistent with the task requirement.
5. When choosing an action, prioritize non-chat actions to ensure the task progresses.
6. If movement is required, choose an action to respond; `chat` should be the last resort.
7. If the requirement is already satisfied, respond with `chat` to explain why no further action is needed.

The selected action's parameters must follow the types and domains described under 'Available Actions'.

Prompt 9.2: Select the action to perform.

- **Short-Term Task Completion**: The LLM first scans recent messages and memory for unfulfilled short-term actions, such as accepted player instructions or acknowledged micro-tasks. If found, the agent should prioritize these immediately—unless it detects that they were already completed.
- **Redundancy Elimination**: Reflection includes a verification stage in which the LLM ensures that a task is not being needlessly repeated, referencing memory and message history to confirm outcomes.
- **Long-Term Goal Resumption**: If no short-term tasks are active, the LLM explores the memory for paused or deferred long-term objectives (e.g., building a base, exploring a cave). It evaluates their relevance in the current context and determines whether to resume them.
- **Initiative via Personality**: Personality profiles help modulate behavior. An active, initiative-taking agent may resume tasks or suggest new goals unprompted, while a passive or deferential agent may wait for input or check in with collaborators.
- **Graceful Idleness**: If no useful action is identified, the system returns a null message. This allows the agent to idle without issuing irrelevant behavior or consuming model resources.

Craft an item.

```
{
    'name': '!craftRecipe',
    'description': 'Craft the given recipe a given number of times.',
    'params': {
        'recipe_name': { type: 'ItemName', description: 'The name of the output item to craft
            .' },
        'num': { type: 'int', description: 'The number of times to craft the recipe. This is
            NOT the number of output items, as it may craft many more items depending on the
             recipe.', domain: [1, Number.MAX_SAFE_INTEGER] }
    },
    'perform': runAsAction(async (agent, recipe_name, num) => {
        await skills.craftRecipe(agent.bot, recipe_name, num);
    })
}
```

Acion 9.1: Craft an item.

Perform the reflection.

You are an AI agent reflecting on your recent activities in a Minecraft world. Your goal is to assess whether any short-term tasks are still pending, and to consider if any long-term objectives should be resumed. Your reflection should also take into account your personality profile to adjust your tone and initiative level.

Please follow the steps below:
- Short-Term Task Check: Review whether you were recently given a short-term task (e.g., "do a dance", "pick up an item", "go to a location") that you acknowledged or talked about but have not yet completed. If such a task exists, it becomes your immediate next step.
- Avoid Redundant Actions: Analyze the task carefully. Do not repeat an action if it has already been completed, unless it is meant to be repeated (e.g., a continuous or time-based task).
- Long-Term Task Resumption: If there are no remaining short-term tasks, check whether you had previously started a long-term task and paused it. Consider whether now is a good time to resume it, and determine what the next appropriate action would be.
- Validate Against Memory: Before deciding what to do next, analyze the memory and message history to confirm that any task you are considering has not already been completed. Avoid redundant or irrelevant actions.
- Adjust Initiative Based on Personality: Refer to your personality profile (e.g., active vs. passive, teamwork-oriented vs. solo, builder vs. explorer).
  * If your current situation aligns with your preferences, you may take a more proactive role.
  * In that case, feel free to suggest a next step even if no one asked you to—but ideally, ask others for their opinion before proceeding.
- Return Null if Idle: If there are no pending or paused tasks and nothing needs to be done, return a null message to indicate that no further reflection is needed for now.

Prompt 9.3: Perform the reflection.

9.5 Coding: Extending Behavior Through LLM-Generated Functions

While reflection and action selection allow agents to operate within a predefined behavioral space, true adaptability requires the ability to expand that space dynamically. This is where LLMs demonstrate transformative power—not just in choosing from existing options, but in creating new executable behaviors on the fly [Wang et al., 2023a]. When an agent encounters a task that cannot be solved using existing actions, the system prompts the LLM to generate a new executable function, effectively extending the bot's capabilities at runtime. This "on-demand coding" process is guided by a well-structured prompt and a strict safety protocol.

To generate a new action, LLM is given a coding task described in natural language. The prompt should carefully structured to provide all relevant context necessary for producing a valid, efficient, and safe function. At its core, the task description concisely explains what behavior or capability the new function should implement. This can range from low-level motion planning (e.g., "climb the nearest hill") to high-level routines (e.g., "build a three-level tower near the player's base").

If available, the system also provides the LLM with logs and source code from previous failed or incomplete attempts. This historical trace helps the model avoid redundant mistakes, identify points of failure, and iteratively refine its approach. Alongside the task description, a curated list of available APIs is included—typically encompassing the AIC's runtime interfaces, such as environment control functions (e.g., navigation, inventory access, or building commands), helper utilities, and domain-specific methods. These APIs are explicitly documented within the prompt, ensuring that the generated code remains grounded in the capabilities of the execution environment.

To ensure compatibility and prevent uncontrolled behavior, the LLM is instructed to output a function using a fixed signature and naming convention. The function must be written in the required programming language (e.g., Python or JavaScript, depending on the system) and adhere strictly to the execution constraints. This ensures the resulting code can be safely compiled, injected, and executed within the AIC's environment without further manual intervention.

Furthermore, generated code should be safe, executable, and useful. Thus the LLM must be told to follow some key rules in coding. For examples:

- **Strict Interface Compliance.** The output must define a Python function named `generated_action` that takes exactly one parameter: `agent`. The body of the function must be self-contained and operate entirely within the bot's environment.
- **Sandboxed Execution.** The LLM must adhere to strict safety constraints when generating code. Specifically, it must not use any potentially unsafe features such as file input/output operations, system calls, or network access. Additionally, dynamic evaluation functions are explicitly prohibited to avoid arbitrary code execution risks. The generated code should rely only on standard language features and a limited set of approved libraries that are deemed safe and contextually appropriate within the AIC's execution environment.
- **Precise and Focused Behavior.** The generated function should be concise and focused, ideally implementing a single logical step toward achieving the overall task goal. It must include appropriate condition checks before attempting any actions—for example, verifying that the required item is present in the inventory before attempting to place a block. Complex control structures should be avoided unless absolutely necessary, in order to maintain clarity, traceability, and ease of integration within the agent's larger behavior loop.

To further enhance coding quality, several additional strategies are employed during LLM-driven code generation. For example, the prompt context could include prior task examples, relevant API definitions, and earlier execution histories, enabling the LLM to align its output with expected coding patterns and interface constraints. When generating code that invokes time-sensitive or asynchronous actions (such

as movement or mining), the LLM could be instructed to insert brief delays or an outcome validation action to ensure task completion and system stability. These practices collectively support more reliable, interpretable, and iterative agent programming.

9.6 Leveraging LLMs for Structured and Creative Task

While memory, action selection, reflection, and code generation represent the core pillars of the MARC architecture, LLMs can also play a pivotal role in high-level planning, particularly for tasks that are structurally complex or open-ended—such as construction, resource gathering, or multi-stage quests.

9.6.1 LLM-Assisted Planning for Structured, Multi-Step Tasks

While the MARC architecture allows agents to manage both long-term and short-term goals through memory-informed reasoning, this mechanism primarily relies on natural language summaries and reflection-based planning. However, for tasks that involve a large number of tightly interdependent steps—such as constructing a multi-room building or executing a complex crafting pipeline—this approach can become unreliable.

The core challenge stems from the mismatch between the structured nature of such tasks and the unstructured format of natural language memory. As the number of steps grows and their relationships become more intricate (e.g., temporal dependencies, spatial layout, material constraints), it becomes increasingly difficult to maintain coherence through episodic summaries alone. The agent may forget critical dependencies, misorder actions, or re-interpret goals inconsistently across sessions.

To address this limitation, a viable strategy is to leverage an LLM not just for decision-making on a step-by-step basis, but as a dedicated planning module capable of producing a structured execution roadmap. In this planning-centric setting, the LLM plays the role of a high-level task architect. Rather than selecting actions one step at a time, it is prompted to generate or refine a comprehensive and structured execution plan—often before any concrete behavior is performed. The resulting plan serves as an externalized, persistent guide for the AIC's long-term operation. Depending on the nature of the task, this plan can take various forms, such as:

- **A sequenced list of subgoals or atomic actions:** Clearly numbered and ordered steps that outline the procedural logic of the task (e.g., "1. Gather 10 oak logs", "2. Craft planks", "3. Place foundation blocks"), helping the agent maintain orientation and reduce redundancy.
- **A multi-phase execution timeline:** For larger-scale objectives (such as multi-room buildings or dynamic quests), plans may be broken into discrete phases (e.g., "Phase I: Site preparation", "Phase II: Foundation and walls", "Phase III: Interior decoration"), each encapsulating its own set of goals and dependencies.
- **A resource-oriented material checklist with linked build instructions:** The LLM can first infer what materials are required to fulfill the design goal, generate a checklist (e.g., "30 cobblestone blocks, 15 glass panes"), and pair each requirement with the steps for how and when it will be used in the construction process.
- **Spatially-aware construction schemas:** If prompted with blueprints or location constraints, the LLM may also produce coordinate-bound layout suggestions (e.g., "Place door at $(x+2, y, z-1)$") or symbolic floorplans to be executed later by the bot.

These structured plans offer multiple benefits: they offload reasoning from token-limited memory, reduce hallucination risks by explicitly stating preconditions, and enable easier verification, progress tracking, and replanning. Critically, they also allow the AIC to pause, resume, or delegate parts of the task without losing sight of the overarching goal. ect deviations, or prompt replanning if needed.

By decoupling high-level structure from natural language memory and embedding it in an explicit planning representation, this method ensures greater precision, reduces forgetting, and improves consistency in carrying out complex, long-horizon goals.

9.6.2 Use of Contextual References

While LLMs are capable of generating general plans from high-level task descriptions, the quality, efficiency, and relevance of those plans can be significantly enhanced by incorporating contextual references. These references provide grounding information that helps the model tailor its output to the specific situation, environment, and user preferences. In practice, the LLM may be guided with a combination of the following inputs:

- **Visual or textual building blueprints:** These may include floor plans, sketches, or written descriptions of the desired structure. For example, a prompt might specify "build a Japanese-style pagoda with four tiers and a central lantern", or provide a schematic indicating relative dimensions and materials.
- **Previously completed plans or exemplar builds:** Existing builds—either created by other agents or imported from curated datasets—can serve as stylistic references or templates for reuse. This allows the agent to reproduce architectural motifs, replicate prior successes, or generate consistent variations on known themes.
- **Environmental descriptions:** These include parsed information about the current terrain, biome type, surrounding structures, available space, and resource distribution. For instance, an agent might be informed that it is working in a snowy taiga with limited wood but abundant stone, prompting it to adjust construction style accordingly.

By combining these contextual cues with a planning prompt, the LLM can produce structured plans that are not only functionally effective but also aesthetically aligned and environmentally adaptive. This ability to blend reference-based retrieval with generative reasoning supports a more robust and user-aligned planning process. It transforms the AIC from a generic executor of commands into a context-sensitive, creatively guided collaboratory.

Ultimately, this reference-driven approach illustrates how LLMs, when properly prompted and informed, can help bridge the gap between symbolic plans and grounded action within embodied environments.

9.7 Discussions

LLMs have emerged as the central reasoning engine for embodied AICs, enabling them to perceive, plan, act, and adapt within rich interactive environments. From summarizing memory and maintaining long-term identity, to selecting next actions, reflecting on progress, and even writing custom code to expand capabilities, LLMs provide the flexible cognitive substrate that allows AICs to move beyond scripted behavior toward emergent, human-like intelligence.

However, despite these strengths, LLMs remain fundamentally limited by their reliance on language as the sole modality of reasoning. Their inputs are shaped by text-based training data, which means they must infer complex spatial, temporal, and social dynamics from linguistic descriptions alone. This abstraction introduces challenges when AICs are asked to interpret ambiguous or noisy perceptual input; navigate evolving 3D spaces or manipulate physical affordances; or handle nuanced social dynamics involving timing, emotion, and implicit norms.

Even in simulated environments —which offer simplified, controlled worlds—AICs driven solely by language struggle with fine-grained coordination, implicit causality, and context preservation over long time horizons. Language models may hallucinate state, misattribute causality, or lack grounded understanding of environment-specific constraints.

Overcoming these limitations will require expanding beyond text. AIC systems will be more powerful when they integrate LLMs with multimodal models (e.g., vision-language or action-conditioned transformers), structured memory systems, and sensorimotor feedback loops to close the gap between textual inference and grounded understanding. Such hybrid systems hold the promise of achieving truly embodied intelligence: AICs that do not merely describe the world, but meaningfully exist and act within it.

Chapter 10

The Future with Multimodal LLMs

While Minecraft AI currently relies primarily on text-based large language models (LLMs) to drive cognition and behavior, the emergence of multimodal LLMs with vision—signals a transformative shift in how AICs can perceive, understand, and act in virtual environments. These models are capable of processing and generating across multiple modalities including images, audio, video, and structured data, significantly enhancing an agent's ability to interpret its world and engage more naturally with users.

10.1 Visual Perception for Human-Like Understanding

The integration of Vision-Language Models (VLMs) offers a transformative pathway for equipping AICs with more human-like perception. VLMs such as GPT-4o, Gemini-2.5 and Claude-4 extend beyond textual inputs to interpret and reason over visual data—enabling agents to *see*, *understand*, and *respond* to their environment in fundamentally new ways.

In a richly interactive world like Minecraft, relying solely on textual representations—such as inventory logs, coordinate lists, or chat-based commands—limits the expressiveness and efficiency of AICs. Visual input provides a high-dimensional yet compressed abstraction of the environment, which allows the AIC to process global spatial context, detect layout patterns, and assess real-time states without needing to parse lengthy textual descriptions. This is particularly important under the token-limited context windows of current LLMs. A single image can compactly convey environmental complexity that would otherwise require thousands of tokens to describe. By accepting images as part of their prompt context, AICs gain access to:

- **Rich spatial layouts:** Recognizing terrain formations, elevation changes, or object placements that influence pathfinding or building decisions.
- **Holistic status monitoring:** Capturing overall scene status—including time of day, nearby threats, or teammate positions—at a glance.
- **Multi-object tracking:** Understanding the spatial relationship between entities such as players, mobs, and constructions.

In addition to improving environmental perception, VLMs enhance the AIC's ability to understand and interact with human players. Screenshots, drawings, and visual demonstrations are intuitive for players to create and share. By interpreting these visual cues, AICs can understand the intended construction styles based on examples; the task expectations; inferred from before/after images or annotated blueprints; and the social context and focus by interpreting the player's postures. This visual grounding makes AIC behavior more intuitive and interpretable, reducing the cognitive gap between human and AI collaborators.

It also enables more fluid workflows where players can "show" rather than "tell" agents what needs to be done, aligning closely with natural human instructional behavior.

10.1.1 Incorporating Visual Input into VLM-Powered Perception

To extend the perceptual capabilities of AICs through vision-language models, visual data must be formatted and supplied to the VLM in ways that align with both system design and interaction flow. Two primary strategies have emerged for providing images to VLMs, each with distinct trade-offs in terms of fidelity, integration complexity, and runtime responsiveness.

- **External Vision via UI or Companion Tools.** In this method, a player or developer manually captures a screenshot or uploads a visual reference to an external interface—such as a web-based tool, model playground, or custom API wrapper—where the image is passed to the VLM. The VLM is then prompted to analyze the image and return a structured textual description, such as:
  - current location features and terrain type,
  - number and type of visible entities or objects,
  - construction state of an in-progress building,
  - notable visual cues (e.g., time of day, health indicators, errors).

  This extracted summary is then sent to the AIC as natural language context—integrated into its memory, reflection, or planning modules. This method is ideal for human-AI collaboration and debugging, offering rich offline interpretation and reusable visual input. However, it adds latency and limits real-time automation since the process often relies on human mediation or out-of-band analysis.
- **Internal Vision from Agent Perspective.** An alternative approach involves treating visual capture as an `action` that can be triggered autonomously by the AIC or scheduled within its reasoning loop. When invoked, this action captures an image (e.g., a first-person view from the AIC or another player) and sends it to the VLM alongside a structured prompt. The model returns context-aware insights that are directly parsed into short-term memory or used to influence immediate decisions.

  For example, an AIC may internally issue a `!capture` command to scan the construction site, prompting the VLM to answer: *"The tower is half complete. The left side is missing windows, and four stone blocks are misaligned."*

  This design allows real-time integration of visual understanding into agent behavior without user intervention, supporting autonomous perception loops. However, it introduces runtime complexity, and the quality of insight depends heavily on camera positioning, frame selection, and prompt design.

Both methods are complementary and may be used in tandem: external vision for pre-task analysis or long-term memory updates, and internal vision for immediate perception and context-sensitive decision-making. Together, they help bridge the perceptual gap between human and AI understanding of the world.

10.1.2 Challenges in Integrating VLMs into AIC Logic

While VLMs offer exciting capabilities for enriching perception and reasoning in embodied agents, integrating them into the operational logic of AICs presents several technical and practical challenges. These limitations must be addressed before VLMs can reliably serve as first-class cognitive components in the MARC architecture.

Most publicly accessible or lightweight VLMs (particularly those with fewer parameters) struggle with nuanced spatial reasoning, fine-grained entity recognition, or domain-specific concepts like block types or construction progress in Minecraft. Compared to language-only models, these smaller VLMs often produce vague or shallow responses to detailed visual prompts.

As a result, developers must carefully constrain their usage: VLMs are better suited for coarse-grained description, broad scene recognition, or generating candidate hypotheses—rather than deep semantic analysis or precise action planning, and the following challenges should be carefully evaluated :

- **High Computation Cost.** Calling a VLM typically incurs higher computational cost and latency than querying a standard LLM. This is due to the increased input complexity, larger model size, and more involved multimodal tokenization and inference procedures. In real-time applications such as agent decision-making or interactive dialogue, this delay can disrupt responsiveness, especially when frequent or repeated vision queries are needed. To mitigate this, visual processing should be used judiciously—triggered only when essential and possibly batched or cached for downstream reuse.
- **Prompt-Response Alignment.** Perhaps the most critical challenge is ensuring tight alignment between the AIC's prompt (i.e., what it asks the VLM to analyze) and the returned output (i.e., what the VLM chooses to report). Unlike structured APIs, VLM outputs are often freeform and may omit relevant details, introduce ambiguity, or fail to satisfy the AIC's decision-making needs.Improving this alignment requires careful prompt engineering, iterative prompt tuning, and potentially multi-step prompting where a first-stage vision summary is refined by a second-stage query tailored to the agent's goals.

Despite these challenges, the integration of VLMs into AIC pipelines holds transformative potential. By combining vision with structured memory, symbolic planning, and language-based reasoning, AICs can begin to perceive and act in a manner far closer to human gameplay and cognition. Future work will focus on model selection, response validation, multimodal grounding, and context-sensitive fusion to close the perception-action loop in more robust and efficient ways.

10.2 Video and Audio Perception

Beyond static images and text, providing embodied AI characters (AICs) with access to **video** and **audio** inputs offers the potential to dramatically expand their situational awareness, emotional perception, and interactive capabilities. In complex environments like Minecraft or other virtual worlds, temporal and auditory information can play a critical role in understanding events, detecting anomalies, and responding to player intent. Incorporating video and audio streams into AIC reasoning unlocks a range of advanced capabilities:

- **Video-based event understanding:** Agents can observe a replay or ongoing video of the world to interpret sequences of actions, detect motion patterns, or recognize changes over time.
- **Gesture and movement interpretation:** By analyzing video frames, AICs may learn to respond to non-verbal communication such as pointing, jumping, or waving.
- **Speech and tone recognition:** Audio inputs can allow AICs to differentiate emotional states, detect urgency, or interpret spoken instructions in more natural forms.
- **Environmental audio awareness:** Agents may learn to respond to ambient sounds, alerts, or cues like footsteps, explosions, or rainfall.

Despite these promising applications, current multimodal models (even the most advanced like GPT-4V or Gemini) face significant limitations when it comes to video and audio inputs:

- **Temporal Reasoning in Video.** Most VLMs are optimized for single-image understanding. While some emerging models can process short video clips, they often lack robust temporal reasoning—struggling to infer cause-and-effect across frames or track entities over time. Their outputs are typically descriptive rather than actionable.

- **Audio-Only Understanding Is Rudimentary.** While some multimodal models can process audio or transcribed speech, true auditory understanding—such as recognizing environmental context from sound, interpreting tone shifts, or separating overlapping sound sources—remains limited. Audio comprehension is generally constrained to short clips and coarse classification tasks.
- **Computational Costs and Latency.** Processing video or audio adds significant computational burden. Long input sequences require extensive tokenization and compression, increasing latency and memory usage. This can make such inputs impractical for real-time or frequent inference cycles within AIC logic.

Incorporating video and audio perception is a promising frontier for AIC research. In the near term, these inputs may be best processed through offline or batched interactions, with human-in-the-loop validation. Future model advances—especially in *video-language modeling*, *audio grounding*, and *temporal attention mechanisms*—will pave the way for fully multimodal agents that can perceive, reason, and act with human-level awareness across all sensory channels.

10.3 Understanding Structural Data in Virtual Worlds

While current multimodal LLMs can interpret images and text, they still lack a deep, accurate understanding of the underlying structural data that defines complex virtual environments such as Minecraft. These structural representations—such as block-level world states, entity graphs, and spatial voxel maps—are essential for precise reasoning, planning, and action in a digital world where spatial accuracy and environmental logic matter.

10.3.1 Limitations of General-Purpose Perception

General multimodal LLMs primarily interpret text and images based on real-world priors or general knowledge of virtual games. Even when a model is familiar with Minecraft as a concept, it typically understands it in terms of simplified textual descriptions (e.g., "a house with a wooden roof"); and general logic of games or survival simulations.

However, in practical embodied AI tasks within Minecraft, such information is insufficient. The agent often requires access to:

- **Block-level structure:** The exact material and position of thousands of blocks in a given volume,
- **World state schemas:** Inventories, crafting recipes, redstone circuits, or biome layouts,
- **3D spatial reasoning:** Navigation through layered terrain, occlusion-aware path planning, or airspace manipulation.

Existing models lack grounding in these data modalities and struggle to reason over them as structured inputs. Textual or visual representations can only partially capture the underlying complexity, making agent behavior brittle or hallucinated.

10.3.2 Toward Structure-Aware Multimodal LLMs: Needs and Challenges

It is an interesting direction to build multimodal LLMs which are able to understand and reason over structured data modalities that represent the virtual world's internal logic and geometry. This may includes:

- **3D block arrays or voxel grids:** These capture the spatial layout of blocks within the world. Understanding such data enables agents to make sense of vertical structures, occluded areas, and volumetric constraints, which are essential for tasks like building, tunneling, or complex object interaction.

- **JSON-based world state graphs:** These represent the game's internal state in a structured form, including information about agents, inventories, locations, entity attributes, crafting status, and active goals. A model that can reason over these graphs can better track dependencies, coordinate actions, and reason symbolically over long time horizons.
- **Semantic maps and annotated scene graphs:** These representations encode spatial and functional relationships between objects, agents, and landmarks, such as "bed next to window," "tree cluster northeast of spawn point," or "zombie hiding behind wall." Such annotations facilitate spatial reasoning and contextual grounding, enabling more realistic behavior and dialogue.

However, integrating these structured inputs into multimodal LLMs presents several formidable challenges:

- **Data Representation and Standardization:** Minecraft's world state is inherently complex and dynamic. There is currently no standard for encoding voxel-based environments and symbolic state graphs in a way that is both efficient for training and expressive enough for detailed reasoning. Designing such a format is a foundational challenge.
- **Dataset Collection and Annotation:** Building a high-quality dataset that links structured world states with corresponding agent behaviors, natural language explanations, or multimodal goals is labor-intensive. The diversity of Minecraft's gameplay means that the dataset must span a wide range of scenarios—including exploration, construction, combat, farming, and dialogue—to be broadly effective.
- **Model Architecture and Modality Fusion:** Existing MLLMs are not optimized for structured, non-sequential inputs like graphs or voxel volumes. New hybrid architectures may be required that fuse large language models with graph neural networks, 3D convolutional encoders, or symbolic planners, allowing these models to perform spatial and logical reasoning over structured contexts.
- **Training Scale and Compute Requirements:** Models capable of integrating multiple modalities—including structured Minecraft data—will likely require scale comparable to or exceeding that of current frontier models. Training these models from scratch or through continued pretraining demands substantial computing infrastructure and careful task-specific optimization.
- **Evaluation and Grounding Alignment:** It remains difficult to determine whether a model's performance reflects genuine structural understanding or superficial pattern matching. Reliable benchmarks, probing techniques, and alignment methods are needed to verify that models interpret structured inputs faithfully and produce grounded, goal-aligned behavior.

As Minecraft and other simulation platforms continue to evolve into testbeds for embodied intelligence, the development of structure-aware multimodal language models represents a critical frontier. Building such models will not only significantly expand the cognitive scope of AICs, but also open up new directions in embodied AI research. These include areas such as *multimodal grounding*, where models align structured representations with sensory inputs; *symbolic planning*, where logical task decomposition is grounded in spatial affordances; and *interactive reasoning*, where agents adapt their behavior through real-time collaboration and feedback.

Moreover, structure-aware multimodal LLMs promise to bridge the gap between human-centric representations (such as blueprints, maps, and task instructions) and machine-executable behavior. This could revolutionize the design of AICs and co-creative agents—not just in Minecraft, but across a wide range of simulation-based learning environments, digital twin systems, and robotics simulators.

Ultimately, this convergence of multimodal perception, symbolic structure, and adaptive reasoning may lay the foundation for a new generation of embodied agents—ones that do not simply mimic intelligence through language, but truly *understand, navigate, craft and communicate* within richly structured virtual worlds.

10.4 Discussions

The rapid advancement of multimodal LLMs marks a pivotal transition in the evolution of AICs—from purely text-based agents to embodied, perceptually grounded entities capable of navigating complex social and physical environments. As multimodal LLMs integrate vision, audio, action, and language within a unified reasoning framework, they offer unprecedented expressive capacity and contextual awareness for real-time character behavior.

At a high level, multimodal LLMs dramatically expand the perceptual bandwidth available to AICs. Agents are no longer blind narrators or rule-bound text processors—they can now see, hear, and interact with the world in modalities that align with human experience. This multimodal grounding reduces the semantic gap between an agent's internal state and its surroundings, enabling more coherent reactions to visual input, spatial reasoning based on physical layouts, or affective responses to tone and gesture. Coherence, in this context, becomes not just a linguistic property but a holistic alignment between perception, cognition, and action.

Second, the performance gains from recent multimodal LLMs—enabled by architectural innovations, larger training datasets, and GPU-accelerated inference—allow real-time, low-latency deployment in interactive environments. Fast inference unlocks rapid response loops, empowering agents to engage in fluid conversations, react promptly to visual changes, or synchronize gestures with speech in tightly coupled multimodal timelines. This temporal coherence is crucial for immersive character believability, particularly in games, simulations, and AR/VR contexts.

Third, powerful multimodal LLMs simplify the agent design stack. Where coherence once required painstaking manual wiring of text, image, and logic modules, multimodal LLMs collapse these interfaces into a single embedding space. This convergence enables developers to specify character traits, world models, and behavioral policies using natural abstractions—such as images, video clips, diagrams, or structured tool-use traces—rather than brittle symbolic grammars. As a result, character memory, perception, and reflection can all be encoded and queried within the same foundation model, reducing architectural complexity while enhancing expressivity.

Ultimately, multimodal LLMs do not merely add new senses to AICs—they reframe what it means to be an agent. The future of AICs is not limited to speaking convincingly, but to acting coherently, sensing responsively, and remembering autobiographically in richly textured environments. These models will underpin AICs that can not only understand the world as we describe it, but as we see, hear, and feel it—marking a decisive step toward human-aligned intelligence in interactive media.

Chapter 11

The Roadmap of Minecraft AI

As Minecraft AI evolves into a long-term open research and development initiative, its roadmap reflects a dual commitment to technical advancement and community collaboration. This chapter outlines the major directions that will guide the future of Minecraft AI—including infrastructure development, benchmark construction, and ecosystem engagement. Together, these efforts aim to push forward the frontier of embodied intelligence in virtual environments.

11.1 Maintaining and Expanding the Core Platforms

Ongoing development will continue to support both platforms—**Minecraft AI** (JavaScript version) and **Minecraft AI-Python**—as first-class citizens in the Minecraft AI ecosystem. While each version caters to different technical needs and user communities, they share a unified vision centered on modularity, extensibility, and agent-level coherence under the MARC architecture.

The JavaScript version is optimized for real-time responsiveness, ease of deployment, and plugin-driven experimentation. It is particularly well-suited for scenarios involving online interaction, live demos, and lightweight prototyping.

The Python version, on the other hand, prioritizes stability, introspection, and research-oriented workflows. It offers tighter integration with Python's scientific stack (e.g., for memory management, planning, or visualization) and is ideal for developing long-term, memory-augmented AICs in controlled environments.

Future releases across both platforms will introduce a coordinated set of enhancements:

- **Enhanced Plugin Support:** A standardized plugin interface will allow developers to inject new behavior modules into AICs with minimal boilerplate. Plugins can define new actions, memory strategies, dialogue policies, or reflection routines—enabling a vibrant ecosystem of community-driven extensions.
- **Cross-Version Compatibility:** Shared abstractions for agent logic, behavior trees, and memory structures will enable better interoperability between the JavaScript and Python versions. This ensures that core features, benchmarking tools, and evaluation protocols can operate consistently across both platforms.
- **Modular Support for Advanced Architectures:** Both versions will be upgraded to support emerging technologies such as: multimodal model APIs; distributed inference pipelines for scalable agent deployment across machines; and external cognition services, e.g. remote memory servers, world state graphs, or cloud-based reasoning agents.
- **Developer-Facing Toolkits:** Improvements will include better logging, debugging tools, and agent dashboards to monitor behavior, memory states, and decision flows in real time.

Both versions will serve as reference implementations for embodied AICs, aligned with the core principles of the MARC architecture. They will share:

- A unified action-planning interface for LLM integration,
- Standardized formats for memory, goals, and environment feedback,
- Interchangeable configurations for agents' personality profiles, reflection frequency, and planning depth.

By maintaining two mature, interoperable implementations, Minecraft AI ensures that developers and researchers can choose the toolchain that best matches their needs—without sacrificing functionality or architectural alignment. This dual-platform strategy guarantees both flexibility and stability as the ecosystem continues to evolve.

11.2 Establishing Benchmarks and Shared Databases

To promote rigorous, reproducible progress toward the Embodied Turing Test, Minecraft AI will prioritize the development of a standardized benchmarking and data-sharing ecosystem. This infrastructure will support researchers, developers, and educators working on LLM-driven embodied agents by offering shared tasks, reproducible metrics, and curated datasets. The core components of this initiative include:

- **Scenario Templates:** A diverse library of task configurations and world setups will be provided to assess AICs in domains such as navigation, crafting, exploration, construction, survival, and social interaction. Each scenario will be parametrizable for difficulty, world layout, and agent role—supporting both controlled experiments and open-ended exploration.
- **Evaluation Protocols:** To assess agent performance in a meaningful way, benchmarks will offer hybrid evaluation tools that combine:
  - *Quantitative metrics*, such as task success rate, resource efficiency, completion time, and policy robustness.
  - *Human-in-the-loop evaluations*, such as ratings on believability, social coherence, goal alignment, and perceived intelligence.
  - *Memory and coherence scoring*, evaluating whether agents can recall commitments, stay consistent over time, and exhibit human-like behavior patterns.
- **Standardized Memory and Dialogue Logs:** All agent runs will produce structured logs of decision traces, memory states, and interaction dialogues. This ensures experiments are auditable and facilitates comparative analysis across architectures and parameterizations.
- **Curated and Annotated Datasets:** Public datasets will be built from recorded episodes of AIC behavior, annotated with action traces, reflective summaries, and language-grounded planning. These datasets will enable:
  - Supervised training for specific behaviors,
  - Imitation learning and fine-tuning of foundation models,
  - Evaluation of planning, reflection, and memory use in realistic scenarios.

To further support openness and reproducibility, the entire benchmarking framework will be integrated directly into the Minecraft AI infrastructure. This includes:

- Scripts for automated scenario generation and evaluation,

- Interfaces for uploading and visualizing experiment results,
- Tools for live replay and annotation of agent runs.

All benchmark resources—including templates, evaluation tools, and datasets—will be made publicly available under open licenses. By fostering a shared foundation for experimentation, Minecraft AI aims to accelerate the collective progress of the embodied AI research community and provide robust tools for future studies on memory, planning, and interaction in large-agent environments.

11.3 Fostering a Vibrant and Inclusive Community

The success of Minecraft AI depends not only on the quality and extensibility of its technical platforms, but also on the creativity, diversity, and engagement of its community. We envision a future where AICs are not isolated test cases in research environments, but fully integrated participants in dynamic, co-creative virtual worlds—interacting with players, collaborating with developers, and contributing to shared narratives over extended periods of time.

To realize this vision, Minecraft AI will actively support and grow a community that includes:

- **Workshops and Tutorials:** We will provide accessible, hands-on materials for users at all skill levels—from beginner-friendly guides to advanced modules on AIC memory architecture, planning, and multimodal integration. Live and recorded workshops will be hosted regularly, with a focus on building, deploying, and experimenting with AICs using both JavaScript and Python platforms.
- **Community Events and Competitions:** To inspire experimentation and reward innovation, we will organize events such as:
  - *Build-a-Bot Challenges* where participants create AICs that solve themed tasks or express unique personalities,
  - *Roleplaying Scenarios* involving long-form social interaction between humans and AICs in narrative-rich environments,
  - *Multi-Agent Missions* requiring coordinated behavior among multiple autonomous characters with complementary roles.

  These events will serve both as community celebrations and as informal benchmarks for progress in embodied AI.
- **Forums and Online Spaces:** A dedicated communication hub will be maintained—featuring discussion boards, plugin repositories, benchmark sharing, and user showcases. These forums will foster peer-to-peer learning and support open-source contributions. Integration with platforms like GitHub, Discord, and Hugging Face will ensure the community remains active and interconnected.
- **Educational Outreach:** Minecraft AI will support efforts to bring Embodied AI into classrooms, workshops, and public education. This includes curriculum-aligned teaching kits, school-based AI challenges, and collaborations with educators to make AI development more approachable and exciting for students and hobbyists.

As the ecosystem grows, we aim to create a space where experimentation and storytelling meet technical rigor—a sandbox where students, researchers, and players alike can prototype new ideas in AI character design. By fostering an inclusive, supportive, and open-source culture, Minecraft AI will not only accelerate progress in embodied intelligence, but also democratize access to the tools and concepts shaping the future of human-agent interaction.

11.4 Summary

This roadmap is not a fixed blueprint, but a living framework to guide the collective evolution of Minecraft AI. Through continued platform maintenance, rigorous benchmark development, and active community engagement, we aim to cultivate a research and development environment where AICs are not only technically capable—but also meaningfully human-like. With the Embodied Turing Test as our guiding goal, we look forward to growing a generation of AICs that understand, adapt, and thrive in virtual worlds.

What we want to build in Minecraft AI is not merely an open-source project or a collection of LLM-driven agents. It is a step toward a future where artificial intelligence is embodied, interactive, and meaningfully situated in the environments it inhabits. Through blocks, mobs, tools, and terrain, we simulate not only tasks but cognition—memory, perception, collaboration, and adaptation.

By placing agents in a world with physical consequences and social expectations, we expose both their strengths and their limitations. This journey is as much about exploring what AI can do, as it is about understanding what intelligence means when it's no longer just textual or abstract.

It is not meant to present a perfect solution, but to serve as a living, modular exploration of Embodied AI. As language models grow more capable, and as we better understand intelligence in motion, the tools built here may seed the way for AICs that are not only responsive, but relatable. From one block to the next, the world is still being built.

We warmly invite developers, researchers, educators, and curious minds to get involved. Whether you're optimizing pathfinding, building new plugins, improving memory systems, or exploring how agents can collaborate, Minecraft AI is an open and evolving space for real innovation. All contributions are welcome, from code and experiments to feedback and big-picture questions.

Bibliography

Altera. AL, Andrew Ahn, Nic Becker, Stephanie Carroll, Nico Christie, Manuel Cortes, Arda Demirci, Melissa Du, Frankie Li, Shuying Luo, Peter Y Wang, Mathew Willows, Feitong Yang, and Guangyu Robert Yang. Project sid: Many-agent simulations toward ai civilization, 2024. URL https://arxiv.org/abs/2411.00114.

John R Anderson. Act: A simple theory of complex cognition. *American psychologist*, 51(4):355, 1996.

John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological review*, 111(4):1036, 2004.

Michael C Anderson and Collin Green. Suppressing unwanted memories by executive control. *Nature*, 410 (6826):366–369, 2001.

Anthropic. Introducing model context protocol. https://www.anthropic.com/news/model-context-protocol, 2023. URL https://www.anthropic.com/news/model-context-protocol. Accessed: 2025-05-23.

Anthropic. Introducing claude 4. Web page, 2024. URL https://www.anthropic.com/claude. Retrieved from https://www.anthropic.com/claude.

Ariel Flint Ashery, Luca Maria Aiello, and Andrea Baronchelli. Emergent social conventions and collective bias in llm populations. *Science Advances*, 11(20):eadu9368, 2025.

Axel Backlund and Lukas Petersson. Vending-bench: A benchmark for long-term coherence of autonomous agents, 2025. URL https://arxiv.org/abs/2502.15840.

Shaojie Bai, Te-Li Wang, Chenghui Li, Akshay Venkatesh, Tomas Simon, Chen Cao, Gabriel Schwartz, Ryan Wrench, Jason Saragih, Yaser Sheikh, et al. Universal facial encoding of codec avatars from vr headsets. *arXiv preprint arXiv:2407.13038*, 2024.

Lawrence W Barsalou. Grounded cognition. *Annu. Rev. Psychol.*, 59(1):617–645, 2008.

Frederic Charles Bartlett. *Remembering: A study in experimental and social psychology*. Cambridge university press, 1995.

Peter F. Bladin. W. grey walter, pioneer in the electroencephalogram, robotics, cybernetics, artificial intelligence. *Journal of Clinical Neuroscience*, 13(2):170–177, 2006. ISSN 0967-5868. doi: https://doi.org/10.1016/j.jocn.2005.04.010. URL https://www.sciencedirect.com/science/article/pii/S096758680500398X.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.

Michael Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Rodney A Brooks. Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159, 1991.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 11079–11091. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/47e288629a6996a17ce50b90a056a0e1-Paper-Conference.pdf.

Character.AI. Character profile. https://book.character.ai/character-book/character-profile, 2025. Accessed 1 June 2025.

Character.ai. Character.ai | personalized ai for every moment of your day, 2025. URL https://character.ai/. Accessed: 2025-06-01.

Jiaqi Chen, Yuxian Jiang, Jiachen Lu, and Li Zhang. S-agents: Self-organizing agents in open-ended environments. *arXiv preprint arXiv:2402.04578*, 2024.

Valerie Chen, Abhinav Gupta, and Kenneth Marino. Ask your humans: Using human instructions to improve generalization in reinforcement learning. *arXiv preprint arXiv:2011.00517*, 2020.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2(4):6, 2023.

Marvin M Chun, Julie D Golomb, and Nicholas B Turk-Browne. A taxonomy of external and internal attention. *Annual review of psychology*, 62(1):73–101, 2011.

ACM Communications. A brief history of embodied artificial intelligence and its future outlook, 2023. URL https://cacm.acm.org/blogcacm/a-brief-history-of-embodied-artificial-intelligence-and-its-future-outlook/.

Michael T. Cox. Metacognition in computation: A selected research review. *Artificial Intelligence*, 169 (2):104–141, 2005. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2005.10.009. URL https://www.sciencedirect.com/science/article/pii/S0004370205001530. Special Review Issue.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Cyprien de Masson D'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems*, 32, 2019.

DeepSeek AI. Deepseek v3. Web page, 2024. URL https://www.deepseek.com/products/deepseek-v3. Retrieved from https://www.deepseek.com/products/deepseek-v3.

Daniel C. Dennett. *The Intentional Stance*. MIT Press, 1987.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2169–2176. IEEE, 2017.

Guodong Ding, Fadime Sener, and Angela Yao. Temporal action segmentation: An analysis of modern techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(2):1011–1030, 2023.

Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 16290–16314, 2024.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023.

Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6 (2):230–244, 2022. doi: 10.1109/TETCI.2022.3141105.

Nouha Dziri, Ehsan Kamalloo, Kory W Mathewson, and Osmar Zaiane. Evaluating coherence in dialogue systems using entailment. *arXiv preprint arXiv:1904.03371*, 2019.

Glenn Ekaputra, Charles Lim, and Kho I Eng. Minecraft: A game as an education and scientific learning tool. *ISICO 2013*, 2013, 2013.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge, 2022. URL https://arxiv.org/abs/2206.08853.

Adam W Feinberg, Alexander Feigel, Sergey S Shevkoplyas, Sean P Sheehy, George M Whitesides, and Kevin K Parker. Muscular thin films for building actuators and powering devices. *Science*, 317(5843): 1366–1370, 2007.

Takashi Gomi, editor. *Evolutionary Robotics: From Intelligent Robotics to Artificial Life*, Tokyo, Japan, 2001. Springer.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, et al. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*, 2023.

Google. Gemini pro 2.5. Web page, 2024. URL https://ai.google.dev/models/gemini. Retrieved from https://ai.google.dev/models/gemini.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Significant Gravitas. Autogpt. https://github.com/Significant-Gravitas/AutoGPT, 2025. Accessed: June 3, 2025.

Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature communications*, 12(1):5721, 2021.

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

Pitoyo Hartono and Sachiko Kakita. Fast reinforcement learning for simple physical robots. *Cognitive Computation*, 1(1):69–79, 2009. doi: 10.1007/s12293-009-0015-x. URL https://link.springer.com/article/10.1007/s12293-009-0015-x.

Junqing He, Liang Zhu, Rui Wang, Xi Wang, Reza Haffari, and Jiaxing Zhang. Madial-bench: Towards real-world evaluation of memory-augmented dialogue generation, 2024. URL https://arxiv.org/abs/2409.15240.

David Herman. *Story logic: Problems and possibilities of narrative*. U of Nebraska Press, 2004.

Todd Hester, Michael Quinlan, and Peter Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2369–2374. IEEE, 2010. doi: 10.1109/ROBOT.2010.5509886. URL https://www.researchgate.net/publication/221067840.

William Hill, Ireton Liu, Anita De Mello Koch, Damion Harvey, Nishanth Kumar, George Konidaris, and Steven James. Mineplanner: A benchmark for long-horizon planning in large minecraft worlds, 2024. URL https://arxiv.org/abs/2312.12891.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

Shankar Kumar Jeyakumar, Alaa Alameer Ahmad, and Adrian Garret Gabriel. Advancing agentic systems: Dynamic task decomposition, tool integration and evaluation using novel metrics and dataset. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Jinhao Jiang, Changlin Chen, Shile Feng, Wanru Geng, Zesheng Zhou, Ni Wang, Shuai Li, Feng-Qi Cui, and Erbao Dong. Embodied intelligence: The key to unblocking generalized artificial intelligence, 2025. URL https://arxiv.org/abs/2505.06897.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572.

Cameron R. Jones and Benjamin K. Bergen. Does gpt-4 pass the turing test?, 2024a. URL https://arxiv.org/abs/2310.20216.

Cameron R Jones and Benjamin K Bergen. People cannot distinguish gpt-4 from a human in a turing test. *arXiv preprint arXiv:2405.08007*, 2024b.

Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. doi: 10.1109/TPAMI.2010.57.

Michael Kirchhoff, Thomas Parr, Ensor Palacios, Karl Friston, and Julian Kiverstein. The markov blankets of life: autonomy, active inference and the free energy principle. *Journal of The Royal Society Interface*, 15 (138):20170792, 2018. doi: 10.1098/rsif.2017.0792. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsif.2017.0792.

Benjamin Klieger, Charis Charitsis, Miroslav Suzara, Sierra Wang, Nick Haber, and John C Mitchell. Chatcollab: Exploring collaboration between humans and ai agents in software teams. *arXiv preprint arXiv:2412.01992*, 2024.

Arne Köhn, Julia Wichlacz, Álvaro Torralba, Daniel Höller, Jörg Hoffmann, and Alexander Koller. Generating instructions at different levels of abstraction. *arXiv preprint arXiv:2010.03982*, 2020.

Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, and et al. Ai2-thor: An interactive 3d environment for visual ai. In *CVPR Workshop*, 2017.

Jeff Kuhn. Minecraft: Education edition. *CALICO Journal*, 35(2):214–223, 2018. doi: 10.1558/cj.34600.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, pages 1378–1387. PMLR, 2016.

Emanuele La Malfa, Gabriele La Malfa, Samuele Marro, Jie M Zhang, Elizabeth Black, Micheal Luck, Philip Torr, and Michael Wooldridge. Large language models miss the multi-agent mark. *arXiv preprint arXiv:2505.21298*, 2025.

John E Laird. *The Soar cognitive architecture*. MIT press, 2019.

LangChain Developers. Langchain documentation. https://python.langchain.com/docs/introduction/, 2024. Accessed June 4, 2025.

LangChain Inc. Plan-and-execute agents in langchain. https://blog.langchain.dev/planning-agents/, 2024. Accessed 31 May 2025.

LangChain Team. Langgraph: Balance agent control with agency. https://www.langchain.com/langgraph, 2024.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-2: Multimodal minecraft agent with goal-observation-action conditioned policy. *arXiv preprint arXiv:2502.19902*, 2025.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Mircea Lică, Ojas Shirekar, Baptiste Colle, and Chirag Raman. Mindforge: Empowering embodied agents with theory of mind for lifelong collaborative learning. *arXiv preprint arXiv:2411.12977*, 2024.

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 69900–69929. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/dd03f856fc7f2efeec8b1c796284561d-Paper-Conference.pdf.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL https://arxiv.org/abs/2307.03172.

Weijie Liu, Zecheng Tang, Juntao Li, Kehai Chen, and Min Zhang. Memlong: Memory-augmented retrieval for long text modeling, 2024a. URL https://arxiv.org/abs/2408.16967.

Xiao Liu, Zhenping Xie, and Senlin Jiang. Personalized non-player characters: A framework for character-consistent dialogue generation. *AI*, 6(5):93, 2025.

Yang Liu, Weixing Chen, Yongjie Bai, Xiaodan Liang, Guanbin Li, Wen Gao, and Liang Lin. Aligning cyber space with physical world: A comprehensive survey on embodied ai, 2024b. URL https://arxiv.org/abs/2407.06886.

LlamaIndex Team. Llamaindex: Release notes and memory architecture. https://docs.llamaindex.ai/en/stable/, 2024. Accessed June 4, 2025.

Qian Long, Zhi Li, Ran Gong, Ying Nian Wu, Demetri Terzopoulos, and Xiaofeng Gao. Teamcraft: A benchmark for multi-modal multi-agent systems in minecraft. *arXiv preprint arXiv:2412.05255*, 2024.

Max Lungarella, Giorgio Metta, Rolf Pfeifer, and Giulio Sandini. Developmental robotics: a survey. *Connection Science*, 15(4):151–190, 2003.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents, 2024. URL https://arxiv.org/abs/2402.17753.

Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473.

James L McGaugh. The amygdala modulates the consolidation of memories of emotionally arousing experiences. *Annu. Rev. Neurosci.*, 27(1):1–28, 2004.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Meta AI. Llama 3. Web page, 2024. URL https://ai.meta.com/llama/. Retrieved from https://ai.meta.com/llama/.

Anton Meta Fundamental AI Research Diplomacy Team, Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624): 1067–1074, 2022.

Microsoft. Introduction to semantic kernel, 2025. URL https://learn.microsoft.com/en-us/semantic-kernel/overview/. A lightweight, open-source development kit for building AI agents and integrating AI models into C#, Python, or Java codebases.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

Milvus Team. Milvus: The high-performance vector database built for scale. https://milvus.io, 2020. Accessed: 2025-06-03.

Justin M Mittelstädt, Julia Maier, Panja Goerke, Frank Zinn, and Michael Hermes. Large language models can outperform humans in social situational judgments. *Scientific Reports*, 14(1):27449, 2024.

Lynn Nadel and Morris Moscovitch. Memory consolidation, retrograde amnesia and the hippocampal complex. *Current opinion in neurobiology*, 7(2):217–227, 1997.

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

Yohei Nakajima. Babyagi. https://github.com/yoheinakajima/babyagi, 2025. Accessed: June 3, 2025.

Allen Newell. *Unified theories of cognition*. Harvard University Press, 1994.

NVIDIA. Nvidia ace: Avatar cloud engine. https://developer.nvidia.com/ace, 2024. Accessed 31 May 2025.

OpenAI. OpenAI Embeddings: Text-Embedding-Ada-002. https://platform.openai.com/docs/guides/embeddings, 2022. Accessed: 2025-06-03.

OpenAI. Gpt-4. Web page, 2023. URL https://openai.com/gpt-4. Retrieved from https://openai.com/gpt-4.

OpenAI. What is memory. https://help.openai.com/en/articles/8983136-what-is-memory, 2024a. Accessed June 4, 2025.

OpenAI. Openai swarm: An experimental framework for multi-agent orchestration, 2024b. GitHub repository https://github.com/openai/swarm.

Charles L Ortiz Jr. Why we need a physically embodied turing test and what it might look like. *AI magazine*, 37(1):55–62, 2016.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

Giuseppe Paolo, Jonas Gonzalez-Billandon, and Balázs Kégl. Position: a call for embodied ai. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Jeiyoon Park, Chanjun Park, and Heuiseok Lim. Enhancing consistency and role-specific knowledge capturing by rebuilding fictional character's persona. *arXiv preprint arXiv:2405.19778*, 2024.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint*, arXiv:2304.03442, 2023.

Junyeong Park, Junmo Cho, and Sungjin Ahn. Mrsteve: Instruction-following agents in minecraft with what-where-when memory. In *The Thirteenth International Conference on Learning Representations*.

Chandana Paul. Morphological computation: A basis for the analysis of morphology and control requirements. *Robotics and Autonomous Systems*, 54(8):619–630, 2006. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2006.03.003. URL https://www.sciencedirect.com/science/article/pii/S0921889006000613. Morphology, Control and Passive Dynamics.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models, 2023. URL https://arxiv.org/abs/2309.00071.

Rolf Pfeifer and Christian Scheier. *Understanding intelligence*. MIT press, 2001.

Rolf Pfeifer, Fumiya Iida, and Gabriel Gómez. Morphological computation for adaptive behavior and cognition. In *International Congress Series*, volume 1291, pages 22–29. Elsevier, 2006.

Pinecone Systems, Inc. Pinecone documentation. https://docs.pinecone.io/, 2025. Accessed June 4, 2025.

PrismarineJS Contributors. Mineflayer: Create minecraft bots with javascript. https://github.com/PrismarineJS/mineflayer, 2017. Accessed: 2025-05-22.

Qdrant Team. Qdrant: High-performance vector search at scale. https://qdrant.tech/, 2025. Accessed: 2025-06-04.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large-language-model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*, 2024.

Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick. Machine theory of mind. In *International conference on machine learning*, pages 4218–4227. PMLR, 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019. URL https://arxiv.org/abs/1908.10084.

Alexander Richard, Colin Lea, Shugao Ma, Jurgen Gall, Fernando De la Torre, and Yaser Sheikh. Audio- and gaze-driven facial animation of codec avatars. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 41–50, 2021.

Roblox Corporation. Real time facial animation for avatars. https://corp.roblox.com/newsroom/2022/03/real-time-facial-animation-avatars, 2023. Accessed 31 May 2025.

Manolis Savva, Angel X Chang, Alexey Dosovitskiy, and et al. Habitat: A platform for embodied ai research. *Proceedings of the IEEE International Conference on Computer Vision*, pages 9339–9347, 2019.

Roger C Schank and Robert P Abelson. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology press, 2013.

Terrence J Sejnowski. Large language models and the reverse turing test. *Neural computation*, 35(3): 309–342, 2023.

Yunfan Shao, Linyang Li, Junqi Dai, and Xipeng Qiu. Character-llm: A trainable agent for role-playing. *arXiv preprint arXiv:2310.10158*, 2023.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021. URL https://arxiv.org/abs/2010.03768.

Thomas R Shultz, Jamie M Wise, and Ardavan Salehi Nobandegani. Text understanding in gpt-4 vs humans. *arXiv preprint arXiv:2403.17196*, 2024.

SillyTavern Contributors. Sillytavern: Llm frontend for power users, 2025. URL https://github.com/SillyTavern/SillyTavern. Beginning in February 2023 as a fork of TavernAI 1.2.8, with over 200 contributors and 2 years of independent development.

SillyTavern Maintainers. Character design. https://docs.sillytavern.app/usage/core-concepts/characterdesign/, 2025. Accessed 31 May 2025.

Herbert A. Simon. Theories of bounded rationality. *Decision and Organization*, 1:161–176, 1972.

Alexey Skrynnik, Aleksey Staroverov, Ermek Aitygulov, Kirill Aksenov, Vasilii Davydov, and Aleksandr I Panov. Forgetful experience replay in hierarchical reinforcement learning from demonstrations. *arXiv preprint arXiv:2006.09939*, 2020.

Linda B Smith. Cognition as a dynamic system: Principles from embodiment. *Developmental Review*, 25 (3-4):278–298, 2005.

Larry R Squire. Memory and the hippocampus: a synthesis from findings with rats, monkeys, and humans. *Psychological review*, 99(2):195, 1992.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

Simeng Sun, Yang Liu, Shuohang Wang, Chenguang Zhu, and Mohit Iyyer. Pearl: Prompting large language models to plan and execute actions over long documents. *arXiv preprint arXiv:2305.14564*, 2023.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long T. Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, Anand Iyer, Tianlong Chen, Huan Liu, Chen-Yu Lee, and Tomas Pfister. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents, 2025. URL https://arxiv.org/abs/2503.08026.

Meiling Tao, Xuechen Liang, Tianyu Shi, Lei Yu, and Yiting Xie. Rolecraft-glm: Advancing personalized role-playing in large language models. *arXiv preprint arXiv:2401.09432*, 2023.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Deepak Trivedi, Christopher D Rahn, William M Kier, and Ian D Walker. Soft robotics: Biological inspiration, state of the art, and future research. *Applied Bionics and Biomechanics*, 5(3):99–117, 2008.

Endel Tulving. Elements of episodic memory. 1983.

Endel Tulving. Memory and consciousness. *Canadian Psychology/Psychologie canadienne*, 26(1):1, 1985.

Endel Tulving et al. Episodic and semantic memory. *Organization of memory*, 1(381-403):1, 1972.

A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 00264423. URL http://www.jstor.org/stable/2251299.

Barbara Tversky. Structures of mental spaces: How people think about space. *Environment and behavior*, 35(1):66–80, 2003.

Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. Learning to speak and act in a fantasy text adventure game. *arXiv preprint arXiv:1903.03094*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a. URL https://arxiv.org/abs/2305.16291.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023b.

Weaviate Team. Weaviate: The ai-native database developers love. https://weaviate.io, 2021. Accessed: 2025-06-03.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Isadora White, Kolby Nottingham, Ayush Maniar, Max Robinson, Hansen Lillemark, Mehul Maheshwari, Lianhui Qin, and Prithviraj Ammanabrolu. Collaborating action by action: A multi-agent llm framework for embodied reasoning, 2025. URL https://arxiv.org/abs/2504.17950.

Wikipedia contributors. Elmer and elsie (robots) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Elmer_and_Elsie_(robots)&oldid=1256964200, 2024a. [Online; accessed 22-May-2025].

Wikipedia contributors. William grey walter — Wikipedia, the free encyclopedia, 2024b. URL https://en.wikipedia.org/w/index.php?title=William_Grey_Walter&oldid=1262061548. [Online; accessed 22-May-2025].

Wikipedia contributors. Diplomacy (game) — Wikipedia, the free encyclopedia, 2025a. URL https://en.wikipedia.org/w/index.php?title=Diplomacy_(game)&oldid=1291752837. [Online; accessed 1-June-2025].

Wikipedia contributors. Minecraft — Wikipedia, the free encyclopedia, 2025b. URL https://en.wikipedia.org/w/index.php?title=Minecraft&oldid=1291573674. [Online; accessed 22-May-2025].

Wikipedia contributors. Original character — Wikipedia, the free encyclopedia, 2025c. URL https://en.wikipedia.org/w/index.php?title=Original_character&oldid=1288200447. [Online; accessed 1-June-2025].

Wikipedia contributors. Shakey the robot — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Shakey_the_robot&oldid=1287404230, 2025d. [Online; accessed 22-May-2025].

Wikipedia contributors. Turing test — Wikipedia, the free encyclopedia, 2025e. URL https://en.wikipedia.org/w/index.php?title=Turing_test&oldid=1291137123. [Online; accessed 27-May-2025].

John T Wixted. The psychology and neuroscience of forgetting. *Annu. Rev. Psychol.*, 55(1):235–269, 2004.

World Wide Web Consortium (W3C). Rdf 1.2 concepts and abstract syntax. https://www.w3.org/TR/rdf12-concepts/, 2025. Accessed: 2025-06-04.

Di Wu, Xian Wei, Guang Chen, Hao Shen, Xiangfeng Wang, Wenhao Li, and Bo Jin. Generative multi-agent collaboration in embodied ai: A systematic review, 2025. URL https://arxiv.org/abs/2502.11518.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Fei Xia, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018. doi: 10.1109/CVPR.2018.00945.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.

Qiang Yi, Yangfan He, Jianhui Wang, Xinyuan Song, Shiyao Qian, Xinhang Yuan, Miao Zhang, Li Sun, Keqin Li, Kuan Lu, et al. Score: Story coherence and retrieval enhancement for ai narratives. *arXiv preprint arXiv:2503.23512*, 2025.

Xianhao Yu, Jiaqi Fu, Renjia Deng, and Wenjuan Han. Mineland: Simulating large-scale multi-agent interactions with limited multimodal senses and physical needs. *arXiv preprint arXiv:2403.19267*, 2024a.

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning. *arXiv preprint arXiv:2410.02052*, 2024b.

Jeffrey M Zacks, Nicole K Speer, Khena M Swallow, Todd S Braver, and Jeremy R Reynolds. Event perception: a mind-brain perspective. *Psychological bulletin*, 133(2):273, 2007.

Ruihong Zeng, Jinyuan Fang, Siwei Liu, and Zaiqiao Meng. On the structural memory of llm agents, 2024. URL https://arxiv.org/abs/2412.15266.

Ruihan Zhang, Fanbo Yu, Ozan Sener, Silvio Savarese, and Yuke Zhu. Unrealzoo: Enriching photo-realistic virtual worlds for embodied ai. *arXiv preprint arXiv:2412.20977*, 2024. URL https://arxiv.org/abs/2412.20977.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? *arXiv preprint arXiv:1801.07243*, 2018.

Zhonghan Zhao, Kewei Chen, Dongxu Guo, Wenhao Chai, Tian Ye, Yanting Zhang, and Gaoang Wang. Hierarchical auto-organizing system for open-ended multi-agent navigation. *arXiv preprint arXiv:2403.08282*, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.

Jordan Zlatev and Christian Balkenius. Introduction: Why epigenetic robotics? In *Proceedings of the First International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, volume 85, pages 1–4. Lund University Cognitive Studies, 2001.