ChatGPT

# State of the Art in World Models Across Robotics, XR/AR, Language, and Gaming

## Introduction

A **world model** generally refers to an internal representation of the state of an environment or "world" that an agent (or multiple agents) can use for perception, reasoning, and decision-making [1] [2]. In different domains, this concept takes on varied meanings – from a robot's map of its surroundings, to a cloud-based spatial map for augmented reality, to the implicit knowledge inside a language model, or the game state managed by a game engine. At its core, a world model provides a **unified, persistent representation** of the environment that can be **shared across multiple components or agents**. This shared representation is critical for coordination between **sensing**, **planning**, and **acting** processes in robotics [2], for maintaining consistent augmented experiences across **devices and users** in XR [3], for allowing AI systems to retain context or knowledge over **time** in language applications [4], and for ensuring all players or NPCs in a **video game** see the same evolving world state. In this report, we survey the state of the art in world models across **robotics**, **extended reality (XR) and augmented reality (AR)**, and other domains such as **natural language AI** and **video games**. We focus on the last 3–5 years of developments (while noting foundational ideas where necessary), examining how richly detailed and consistent world state representations are built and maintained. We also compare architectural approaches to achieving a **persistent, scalable world model** that spans multiple actors, addressing key challenges like grounding (tying representations to reality), temporal consistency, partial observability, and multi-agent coordination.

## World Models in Robotics

In robotics, a world model typically manifests as an **internal data structure** that a robot maintains about itself and its environment [1]. Classic examples include maps for navigation (e.g. occupancy grids or SLAM maps), object models for manipulation, and knowledge bases for task reasoning. As far back as the 1970s, the Shakey robot used an internal model of room layouts for planning actions, an early illustration of a world model [5]. Modern robotic world models often integrate **multiple representations**: metric geometrical maps, symbolic knowledge (object types, relations), and even learned dynamics. For instance, the KnowRob framework maintains a *belief state* as a knowledge graph of objects and facts (ontology-based) while also incorporating a physics simulation as an "inner" world model for concrete states [6] [7]. This combination allows grounding high-level symbolic facts (e.g. *"cup is on table"*) in low-level geometric reality (3D poses, shapes), enabling both logical reasoning and motion planning. Another example is the **Robot Scene Graph (RSG)**, which provides a central 3D geometrical representation of the environment as a directed acyclic graph; nodes represent objects or reference frames (with transform nodes encoding positions over time) and edges define hierarchical spatial relationships [8] [9]. By time-stamping the transform nodes, the RSG can track a dynamic world in a temporally consistent way, storing historical states and enabling temporal queries (e.g. *"Where was object X at 10 AM?"*) [10] [11].

Modern robotic architectures emphasize **shared memory and modularity** in world models. A world model acts as a **common repository** where different subsystems (vision, localization, planning, etc.) can "tell" new

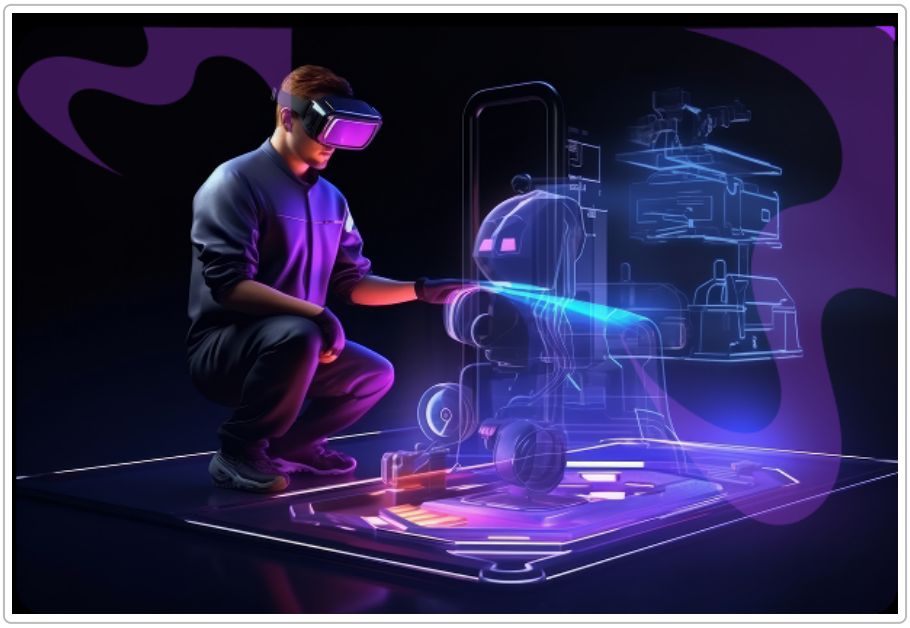
information or "ask" queries [2]. This reduces duplication of data and ensures consistency – a principle known in software engineering as *"single source of truth."* For example, a localization module might update the robot's pose in the world model ("tell"), while a task planner might query ("ask") the location of an object from the same model [12] [13]. Researchers have proposed design guidelines for such world models: they should be sufficiently **rich and accurate** to support all necessary decisions, but also **modular** and **low-coupling** so that components can share information without being tightly interwoven [14] [2]. In practice, this often leads to a **central world state** data structure (for instance, a database or blackboard) within the robot's control software. The world state may include separate layers for the robot's own state vs. the external environment, and for abstract (symbolic) vs. concrete (metric) information. Ensuring consistency between these layers is an active area of research – e.g. linking an object's ID and semantic properties to its geometric pose in the world model, so that symbolic planners and motion planners refer to the same object state [13] [15].

A significant challenge in robotic world modeling is **partial observability and uncertainty**. Robots rarely have complete information about their environment; sensors provide noisy, limited views. World models in robotics therefore often serve as an **accumulator of knowledge** – fusing sensor data over time to build a more complete and stable picture. Techniques like SLAM (Simultaneous Localization and Mapping) continuously update a map with new observations, treating unseen areas with uncertainty until explored. Some world models maintain probabilistic belief distributions for object locations or environmental features, especially in frameworks based on **POMDPs** (Partially Observable Markov Decision Processes). Others use assumptions to fill gaps (e.g. assuming walls or floors extend beyond the current view) and then update those assumptions when new data arrives [16]. The **temporal consistency** of the world model is crucial: as the world changes, the model must be updated without introducing contradictions. Recent literature highlights the importance of modeling time explicitly in the world state – distinguishing between **real-world time** (when events actually occur) and **belief time** (when the robot updated its knowledge) [10] [17]. In practice, however, many robotic systems only store the latest state, forgetting temporal history [18] [19]. Cutting-edge research prototypes have begun to include temporal reasoning, enabling robots to answer questions like "*Where did I last see object Y?*" or "*Is my knowledge about X up-to-date?*" by looking at timestamped beliefs [11] [19].

When multiple robots are involved, maintaining a **shared world model across a team** becomes challenging but vital. Multi-robot systems (e.g. robot fleets in a warehouse or robots playing soccer in RoboCup) require a common world state to coordinate actions and avoid conflicts. One approach is a **centralized world model server** or cloud-based map that all robots contribute to and query. For example, early work in RoboCup soccer had robots share via radio their sensed positions of the ball and players, merging these into a global game state despite communication delays [20] [21]. More recent approaches include using a **knowledge graph as a dynamic shared world model** for heterogeneous robot teams [22] [23]. A knowledge graph can encode diverse information (geometric, semantic, status of tasks) and is accessible to both machines and humans. Such shared world models must handle conflicting information (resolving discrepancies between agents' observations), and they raise issues of scalability (network bandwidth, data consistency protocols) and robustness to partial updates. Nonetheless, the trend in the past few years is toward **cloud robotics** and **digital twins** – offloading the world model to a powerful server or cloud that integrates data from many robots and sensors. This allows **long-term persistence** of the model (beyond any single robot's runtime) and **global scale** mapping. Industry examples include warehouse robotics platforms where a central system keeps a live map of inventory and robot locations, or self-driving car fleets sharing mapping data to collectively improve their world understanding. In summary, state-of-the-art robotic world models blend **detailed 3D maps**, **semantic knowledge**, and sometimes

**learned models** of physics or dynamics. They serve as a single, persistent knowledge source within a robot (or robot team) architecture, enabling consistent decision-making across modules and over time [14] [13]. Ongoing research in robotics is tackling how to ground abstract concepts to sensor data (the grounding problem), how to maintain up-to-date models in dynamic environments, and how to efficiently share massive world data across multiple robots and cloud services.

## World Models in Extended Reality (XR) and Augmented Reality



*Persistent augmented reality experiences are enabled by world models that anchor virtual content to the physical environment.* In the context of XR – which includes virtual, augmented, and mixed reality – a world model represents the system's understanding of the user's physical (and/or virtual) environment. **Augmented Reality (AR)** in particular requires a robust model of the real world so that digital content can be placed consistently in a user's surroundings. The recent shift toward **persistent AR** experiences has led to the rise of the **AR Cloud** concept: a shared digital map of the world stored in the cloud, functioning as a *"digital twin of the real world"* [24]. This AR Cloud contains **spatial maps** (e.g. 3D point clouds or meshes of real environments) and **anchor points** that serve as reference coordinates. By looking up the common map and anchors, multiple users' devices can **align their coordinate systems**, ensuring that if one user places a virtual object on a table, another user sees it in the exact same spot [3]. Industry leaders like Niantic (Lightship), Google (ARCore), and Apple (ARKit) have developed cloud anchor services that let developers pin content to real-world locations persistently [25]. These services typically work by having devices scan their environment (using camera, LiDAR, etc.), upload feature descriptors to the cloud, which then finds a match to a stored map and returns a coordinate transform – effectively **re-localizing** the device in the shared world model. Once re-localized to the common world frame, all devices render AR content in the correct place relative to real-world geometry.

The technical underpinnings of XR world models draw heavily on **spatial computing** and **SLAM**. Inside-out tracking (6-DoF tracking of a device via its own sensors) was a foundational breakthrough around 2017 (with ARKit/ARCore), enabling each device to build a local world model on the fly. Today's emphasis is on merging these local maps into a **global, persistent map**. Simultaneous Localization and Mapping algorithms are

continuously improving in accuracy and robustness, even in the face of moving objects, changing lighting, or large-scale outdoor environments [26]. For truly seamless multi-user XR, the system must achieve re-localization and map sync with very low **latency** (on the order of tens of milliseconds) so that users see updates in real time and virtual objects stay tightly locked to real positions [27]. **Networking and cloud infrastructure** are as crucial as the spatial algorithms here – fast cloud databases store the world models, and edge servers or 5G networks may be employed to reduce latency for nearby users. In fact, experts argue that "an AR system, by its very nature, is too big for a device… The world is too large to fit" entirely on a headset or phone, necessitating that the XR "operating system" live partly in the cloud [28] [29]. This distributed architecture is analogous to online games or web services, except it deals with mapping real physical space. An XR device thus continuously queries a cloud world model service to both **update** the global map with new scans and **retrieve** the latest environment model for its location.

Ensuring a **consistent and richly detailed world state across multiple XR users** involves several challenges. One is **coordinate grounding**: all users and devices must agree on a common frame of reference. Solutions include world-geographic coordinates (e.g. GPS lat/long with local offsets) or a designated "host" device's coordinate frame that others join (used in some ARCore Cloud Anchor sessions). Another challenge is **drift and alignment** – different devices' maps might not perfectly align, causing virtual content to appear offset. To mitigate this, world models incorporate **feature point clouds** or **visual landmarks** that serve as convergence points for map alignment [30]. There is also the matter of **persistence over time**: environments can change (furniture moves, buildings get remodeled). A robust AR world model needs mechanisms to update or invalidate parts of the map. Some systems schedule periodic re-scans or use **crowdsourced mapping** (multiple visitors contribute to refining the map). The AR Cloud effectively becomes a massive, evolving database of world observations – raising questions of **scalability** (storing and indexing billions of spatial features) and **privacy** (the map may inadvertently capture private spaces or people). Research and standards efforts (like the OpenARCloud initiative and OpenXR) are exploring ways to share and update these world models securely and interoperably [31] [32].

Interestingly, XR world models aren't just geometric; they are becoming **semantic**. Knowing pure geometry is useful for rendering, but understanding *what* an object or area is can enable richer interactions. For example, identifying a surface as a "floor" versus "wall" or recognizing a scanned object as a "chair" allows applications to place content more intelligently or enable realistic physics (making a virtual ball bounce on the floor, not hover in the air). Recent AR toolkits and research prototypes integrate **object recognition** and **scene understanding** into the world model. An extended reality operating system concept (DiOS, 2022) outlines that the XR OS should provide "shared access to the world's model" including feature points, semantic labels of areas, and recognized objects as a common resource for all apps [33]. This prevents each app from redundantly scanning and classifying the same scene, and ensures that different applications have a consistent view of the environment. We see early steps toward this in platforms like HoloLens 2, which offers a spatial mapping API and even some semantic labels (e.g. floor planes). The next generation of XR platforms is likely to maintain a continuously updated **scene graph** of the environment – mixing raw spatial data with semantic annotations (like room layouts, object identities) – and share it between apps and users.

For **multi-user XR** (shared AR or multi-player VR), synchronization of the world model is paramount. In collaborative AR, multiple users interact with the same virtual objects. This requires **real-time sharing** of object states (positions, animations, etc.) in addition to the static environment map. Typically, a **networked session** is established with a host server or peer-to-peer communication to replicate any changes to the world state to all participants. For example, if one user moves a virtual chess piece on a table, that action

must be sent through the server and applied on every other user's device so their world model reflects the new piece position. These systems draw on the same consistency techniques as online gaming (discussed later), including client-server authority and state synchronization. The key difference in XR is the mix of **physical and digital**: the base coordinate frame comes from the physical world (which is subject to tracking errors), and digital changes must stay aligned to it. Edge computing is being explored to host local "world model aggregators" that merge inputs from nearby XR devices to build a **global model** without too much cloud latency [34]. One project demonstrated aggregating the environment maps from multiple users in real time to create a more complete model than any single user has, enabling consistent occlusion and physics for all [35] [36]. This hints at a future where *city-scale AR world models* are continuously built by thousands of users and shared – essentially crowdsourced, real-world digital twins.

In summary, XR/AR world models have rapidly evolved from ephemeral, single-user maps to **persistent, cloud-backed models** that can be shared across sessions, devices, and users. The state of the art involves high-accuracy spatial mapping (often AI-enhanced SLAM), cloud distribution for scale, standards like OpenXR for cross-device interoperability, and emerging semantic layers to enrich raw geometry [37] [38]. Challenges remain in maintaining **temporal continuity** (keeping virtual content stable over days or months) and **grounding virtual content to a changing physical world**. Yet, the progress in the last 3–5 years – exemplified by systems like Niantic's Lightship AR Cloud and Microsoft's Azure Spatial Anchors – has made shared, persistent AR a practical reality. XR systems now treat the world model as a first-class component, much like a operating system service, that multiple applications and users can tap into simultaneously.

## World Models in Language Models and Conversational AI

Unlike robotics or XR, **large language models (LLMs)** and other conversational AIs do not interact with a physical space directly – their "world" is largely the world of **information and discourse**. Nevertheless, the concept of a world model is relevant in terms of how an AI represents and retains knowledge about the state of an ongoing conversation or a described scenario. By default, most LLMs (such as GPT-series, etc.) are **stateless** beyond their immediate context window: they generate text based on the input prompt and learned parameters, but they do not internally maintain a persistent model of the conversation or facts over multiple sessions [39] [40]. This has been identified as a limitation: without an ongoing memory or world state, AI assistants can't truly understand context or remember things in the way humans do [40] [41]. In response, recent advancements have introduced **external memory systems** and world modeling approaches for language models. For instance, **retrieval-augmented generation (RAG)** has become standard for grounding LLM outputs in a knowledge base [42]. In RAG, the language model queries an external datastore (vectors, documents, or a knowledge graph) to fetch relevant facts which are then provided in the prompt. This effectively gives the LLM a dynamic "world" of knowledge it can refer to, rather than solely relying on static training data. However, basic RAG alone supplies fragments of text and doesn't ensure the AI has a structured understanding of how those facts relate.

The next frontier is building more structured, persistent world models for language models – sometimes described as integrating **"world models" or knowledge graphs on top of RAG"** [43]. The idea is to have the AI not just retrieve isolated pieces of text, but to maintain an **internal knowledge base** that grows and updates with each interaction. For example, if an AI assistant has a long-running interaction with a user or a team, it could build a knowledge graph of important entities (people, projects, preferences) and their relationships, rather than forgetting them after each session. Sphere Inc. describes this as moving from "memory without meaning" (simply storing transcripts or facts) to a true contextual understanding where

the AI "integrates [new information] into an evolving model of the world" [4]. Such a model would capture not just that *Fact A* and *Fact B* were mentioned, but that A causes B, or A is part of B, etc., enabling more coherent reasoning. Academic projects like Microsoft's **KB-LaM** (Knowledge Base Language Model) and others have started to let LLMs query and even update a symbolic knowledge base as they converse [44]. By bridging high-level knowledge graphs with LLMs' generative ability, the AI maintains consistency over time and can avoid contradictions (for example, not forgetting a character's eye color halfway through a generated story, because it stored that detail in a world model).

Another approach to world models in language is seen in **embodied language agents** and simulators. Researchers have started to place language models inside simulated environments (text-based games, or even Minecraft) where the model must remember and act upon a world state. In these cases, an explicit world model is necessary because the agent's observations are **partial and sequential** (just like a robot's). For example, in text adventure games (like the classic Zork), an AI might build a mental model of which rooms are connected or which objects it has, to decide what to do next. Some solutions use **prompt engineering** to have the LLM summarize and keep track of the world state in text form that is carried along through the game. Others integrate a formal memory – e.g. a set of facts that get updated as the agent explores. A notable recent work is *Voyager (2023)*, where an LLM was used as a controller for a Minecraft agent that learns to survive and build in the game. Voyager employs an **external code memory** (storing skills as code scripts) and maintains an inventory of discovered items and world locations, effectively an explicit world model that persists across the agent's life [45] [46]. This helps the agent avoid forgetting how to craft tools it learned earlier or what resources it has gathered. Similarly, the "Generative Agents" project (Park et al., 2023) populates a sandbox game world with AI characters who have evolving memory streams – each agent records experiences and facts, does periodic reflection to form higher-level conclusions, and uses that to guide future behavior. The shared simulation (a 2D town) provides a common world state, but each agent's actions depend on its own remembered world model of who it has met, what it plans to do, etc. The result was remarkably human-like interactions emerging from agents that could recall and reason about past events (e.g. an agent remembering to throw a party after another agent told them about a birthday) [47] [48]. This exemplifies how explicit memory architectures can serve as a kind of world model for language-based agents in interactive contexts.

From an architectural standpoint, there is a push for a **unified context or memory layer** for AI systems. Enterprises deploying chatbots or LLM-based assistants have noticed that context and knowledge tend to become siloed – one bot might know the customer's order history, another knows the support knowledge base, but they don't share a common memory [49] [50]. To address this, visionaries talk about a **"shared memory" service for AI** in an organization [51]. This could be a centralized knowledge graph or database that all AI agents (and possibly other software) consult for the latest world state of the business (customer profiles, inventory, past interactions, etc.). By querying this shared context on demand, AI agents avoid inconsistency and have up-to-date information. Essentially, it's the analog of the robotic world model but for an enterprise's informational world. Key challenges here include data governance (who owns and updates the memory), privacy (especially if it contains personal data), and truth maintenance (ensuring outdated or incorrect info in the AI's memory gets corrected) [52] [53]. Some current systems implement this in rudimentary form: for example, a customer service chatbot might call an API to retrieve customer info at the start of each session (pulling from the company's database). Future systems could maintain a **long-term memory** per user that grows over repeated conversations (with user consent), making interactions more personalized and context-aware over time. Indeed, 2025 saw many platforms add "memory modes" where a chatbot can remember previous chats, but these often just store raw conversation logs without true understanding [39] [41]. The state of the art is moving toward more *semantic* memory – storing

structured representations of what was learned so the AI can generalize and reason, not just parrot back older messages [54] [55].

Finally, it's worth noting the intersection of language models with **world simulators** in a different sense: In model-based reinforcement learning and planning, there's a concept of a *learned world model* (a neural network that predicts environment dynamics). Some researchers are exploring using language models as a component of such world models for planning complex tasks described in language [56] [57]. For example, an LLM might be prompted with a description of the environment and asked to predict the outcome of a sequence of actions (essentially playing the role of a physics or strategy simulator in text form). While this is still preliminary, it suggests a convergence where language models not only generate text but also serve as general predictors for state transitions in an abstract world defined by language. In summary, the notion of world models in the language domain revolves around **extending the memory and knowledge capabilities** of AI: creating persistent, structured representations that the AI updates as it gains new information. Recent progress includes hybrid architectures combining LLMs with knowledge graphs [43], long-context or retrieval strategies for maintaining continuity, and multi-agent simulations with shared memory. The grand challenge remains **grounding** – unlike robots or XR systems, pure language models lack direct grounding in physical reality, which leads to problems like hallucinations. Bridging that gap (through tools, sensors, or multimodal learning) is an ongoing effort to give language AIs a more reliable world model of the real world they talk about.

## World Models in Video Games and Simulations

In video games, the “world model” is essentially the **game state**: all the information that defines the current condition of the game world, including the environment, objects, characters, etc., at a given time. Modern game engines maintain this world state explicitly, often using the **Entity-Component-System (ECS)** architecture [58]. In ECS, every entity (e.g. a character, a projectile, a tree) is an ID associated with various components that hold data (position, velocity, health, inventory, etc.), and systems are routines that update these components (physics system moves entities according to velocity, rendering system draws them, game logic system checks win/lose conditions, etc.) [59] [60]. The ECS design cleanly separates data from behavior and allows efficient processing of potentially **thousands of entities** in a game world. It is a cornerstone of how games represent a richly detailed world state internally – each tick of the game loop, the state is updated in a controlled manner by the systems, ensuring that all players and AI agents perceive a **consistent sequence of world states**.

For single-player games, the world model lives entirely on the player’s device (the game console or PC runs the simulation). However, in multiplayer and online games, maintaining a **unified world state across multiple players’ machines** is a central challenge. The typical solution is a **client-server architecture**: a server (or authoritative host) holds the master copy of the game state and pushes updates to clients. This way, all players ultimately synchronize to the server’s version of the world, ensuring consistency [61] [62]. Each client may simulate interim frames (client-side prediction) to reduce perceived lag, but any discrepancies are corrected by the server state (reconciliation) [63] [64]. This architecture has been refined over decades to support fast-paced games where even a few milliseconds matter for fairness. Techniques like **state interpolation** (to render smooth movement between discrete updates) and **lag compensation** (so that actions like shooting hit what the player saw on their screen) are all about keeping the distributed world model **temporally consistent** from each player’s viewpoint. Furthermore, game servers manage **partial observability** by design – for instance, in a strategy game with fog-of-war, the server will only send each player information about areas their units have explored, hiding enemy positions they shouldn’t see.

This ensures that each player's local world model contains exactly the information that player is meant to have, no more, no less. Such selective state disclosure is both a gameplay necessity and a bandwidth optimization.

As games have grown in scope, a major concern is **scalability of the world model**. Early online games and MMOs (massively multiplayer online games) often partitioned players into separate servers or "shards" to keep the number of concurrent agents per world manageable. Each shard is effectively an independent world model instance (e.g., 1000 players per shard). However, the vision of a single massive world (think *Ready Player One* or metaverse ideals) has pushed the development of architectures for **distributed world models**. One cutting-edge approach was introduced by Improbable's SpatialOS platform: it uses a **distributed Entity-Component-**<wbr>**Worker** architecture [65] [66]. Instead of one server simulating the entire world, different aspects of the world are handled by different workers (processes potentially on different machines). For example, the world can be spatially divided: one set of physics engine instances handles the north zone of the map while another handles the south zone. Or it can be divided by function: a dedicated AI worker process handles NPC decision-making, separate from the physics or rendering workers [67] [68]. All these workers collaborate to produce one coherent world simulation for players. The complexity of keeping this consistent is non-trivial – it involves managing network messages between workers, load balancing, and ensuring no single point becomes a bottleneck. But when done correctly, it allows virtually **unlimited scale**, as the world's simulation is no longer bound by the CPU/RAM of a single server. A developer can design the game as if it were a normal ECS system, and the underlying platform orchestrates the workers to abstract the distribution [69] [70]. This has enabled games with thousands of concurrent players in one shared space and large persistent worlds that continue evolving even when a player logs off. For example, **Worlds Adrift** (Bossa Studios, ~2018) utilized such a distributed world model to support a single unsharded world where players could physically congregate in the same region without instanced zones [71]. The state of the art in game world architectures thus mirrors distributed databases: techniques from cloud computing (sharding, eventual consistency, fault tolerance via consensus algorithms like Paxos/Raft) have been adapted to game state management [72].

It's also interesting to consider **world models for game AI**. Non-player characters (NPCs) in games often use simplified world representations for decision-making, such as navmeshes for pathfinding or threat trackers for enemy AI. Classic game AI might have a hard-coded model of the environment (like waypoints or area graphs for navigation). But with the advent of machine learning in games, there's overlap with the concept of learned world models from reinforcement learning. For instance, **AlphaGo** and **AlphaStar** (for Go and StarCraft II, respectively) are agents that implicitly learned models of the game environment through simulation. OpenAI's **hide-and-seek** agents (a multi-agent physics environment) developed memory of object locations and possible future states through recurrent neural networks – effectively internal world models to plan their next move. In **procedurally generated or open-world games**, game AI might need to build its own map as it explores, akin to a robot's SLAM in a virtual world. Some sandbox games like **Minecraft** have prompted research where agents learn world models to predict the consequences of their actions in the game (e.g., predicting that mining below you causes you to fall). These are parallel to robotics scenarios but in a simulated domain. One concrete example: the "**World Models**" paper by Ha and Schmidhuber (2018) famously trained a neural network to model the dynamics of a simple car racing game and a Doom-like environment, allowing the agent to imagine future frames and plan – that neural net was literally referred to as the agent's world model. The approach has since been extended (e.g. DeepMind's **Dreamer** agents use learned world models to achieve high sample efficiency on game-like tasks by planning in imagination).

From a **multi-agent coordination** perspective, video games also provide insight. In cooperative games, AI agents (or players) coordinate via the shared game state – e.g., players in a team know each other's locations and status through the game's UI which taps into the world model. If an AI team is controlling multiple units (like a squad tactics AI), it usually has a central world model for all units, enabling strategies like flanking that require knowledge of each unit's position relative to others. Some game AIs use a **blackboard system** – essentially a shared memory space where different AI "behaviors" post information or read about the world and other agents. This is analogous to the shared world model concept in robotics, transplanted into game AI architecture. It simplifies coordination because instead of each agent acting in isolation, they contribute to a common representation (the blackboard) of tactical knowledge (like "area clear" or "target spotted").

In summary, the domain of video games treats world models in a very **explicit and operational** manner: the world state is a well-defined data structure that the game engine updates and synchronizes. Industry practices have advanced from single-machine simulations to **cloud-distributed world simulations**, enabling persistent, large-scale worlds. The emphasis is on real-time consistency, efficient replication of state, and dealing with network/latency challenges so that multiple players (and AI) share a seamless experience. Meanwhile, at the intersection of gaming and AI, world models (either coded or learned) are key to agents planning and behaving intelligently in complex environments. Table 1 provides a comparative summary of how world models are represented and used across the domains we've discussed (robotics, XR, language, and video games):

**Table 1. World Models Across Different Domains – Representations and Approaches**

| Domain | Predominant World Model Representation and Approach | Examples / Notes |
|---|---|---|
| **Robotics** | Internal **belief state** combining metric maps (e.g. occupancy grids, 3D meshes) and **semantic knowledge** (objects, relations). Often implemented as a centralized module with tell/ask interfaces for sensor updates and queries [2]. May include learned dynamics models for simulation (internal "imaginary" simulators in model-based RL). | *Examples:* KnowRob knowledge base (ontology + facts) [6]; Robot Scene Graph (3D scene graph with temporal transforms) [8]; Ha & Schmidhuber's *World Models* (2018) for learned environment simulation in RL. |
| **XR / AR** | **Spatial maps** of real-world environments (point clouds, visual feature maps) stored persistently (AR Cloud). **Anchors** (coordinate frames) tied to real-world locations ensure cross-device alignment [3]. Often cloud-supported for multi-user sync and long-term persistence. Incorporates **SLAM** on device and cloud-based map fusion, increasingly with semantic labels (digital twin concept). | *Examples:* Google ARCore's Cloud Anchors for shared AR [3]; Niantic Lightship AR Cloud enabling multi-user games (e.g. Pokémon Go shared AR); Microsoft Mesh / Azure Spatial Anchors for holoportation and persistent holograms. Research XROS (Extended Reality OS) proposes a system-level world model shared between apps [34]. |

| Domain | Predominant World Model Representation and Approach | Examples / Notes |
|---|---|---|
| **Language AI** | Implicit world knowledge in model weights (from training) plus **external knowledge stores** for dynamic facts. Uses **text context** as a transient world state (conversation so far) and extends it with **retrieval** from databases or **knowledge graphs** for persistence [43]. Emerging architectures maintain a **long-term memory** (e.g. vector databases of dialogues, or a knowledge graph that updates over interactions). | *Examples:* Retrieval-Augmented Generation (LLM + wiki/DB lookup) [42]; MemoryGPT/MemGPT that logs and summarizes past dialogue to simulate long memory; Microsoft's KB-LaM integrating a knowledge base; Stanford Generative Agents with a memory stream for each agent (used in a simulated world). Efforts like LangChain frameworks allow LLMs to use tools/DBs as an external world model. |
| **Video Games** | **Explicit game state** managed by engine: typically ECS with entities & components. The world model is fully known to the simulation (no hidden variables except for designed gameplay uncertainty). For multi-player, a **networked shared state** with server authority ensures consistency across players. State can be huge (open worlds) but partitioned or distributed across servers for scale [69]. Game logic and AI read/write this state every tick. | *Examples:* Unity and Unreal engine's ECS for representing game objects [58]; MMO server clusters (e.g. EVE Online's single shard, SpatialOS distributed simulation) for one massive world [69]. Multiplayer uses techniques like state replication, delta updates, and snapshot interpolation to keep everyone in sync. AI systems may use their own abstractions (navmesh, etc.) derived from the game state. |

## Comparative Analysis and Key Challenges

Across these domains, several **common themes** and challenges emerge in building world models:

- **Grounding and Real-World Alignment:** In robotics and AR, grounding is literal – the world model must align with the physical world (through sensor calibration, coordinate frames, or visual features) so that actions based on the model have the intended real-world effect. A robot's world model is only useful if its coordinates map accurately to physical space (e.g., a robot arm's internal model of an object position must match where the object actually is). Similarly, an AR world model must align virtual graphics to real surfaces with high precision [73]. Language models face an abstract grounding problem: their "world" is words, so grounding means connecting text to actual entities or facts in reality. Without grounding, an LLM might confidently output false information because it lacks a mechanism to check against an external world model. Techniques like retrieval and tool-use are essentially ways to ground the LLM in factual reality by referencing an external source [42]. In video games, grounding is usually not an issue internally (the game state is the ground truth by definition), but if game AI uses learning, it must ground its internal representations to the game's state observations (pixels or game variables). Overall, ensuring that the world model stays tied to the true state of the world (physical or virtual) is fundamental – this requires continuous calibration, error correction, and sometimes human oversight to correct drift or misconceptions.

- **Temporal Consistency and Persistence:** All domains need to handle the passage of time gracefully in the world model. Robotics and games typically operate in real-time loops, updating the state as changes occur. Ensuring consistency means avoiding race conditions and handle latency – e.g., a multi-robot world model must not apply out-of-order sensor updates that might revert to an older state. AR must keep virtual content stable as users move (predictive tracking helps, as does smoothing map updates). A persistent world model (spanning long periods) also needs to reconcile *when* information was valid. The robotics review explicitly calls for distinguishing belief updates from real-world event times to manage stale data [74] [17]. In language applications, temporal consistency might mean remembering what was said earlier and not contradicting it later – something humans do naturally, but LLMs struggle with when context window is exceeded. Solutions like session memory or user profiles try to keep relevant history accessible. Meanwhile, video games often simulate time in discrete ticks which all clients agree on, achieving temporal consistency through lockstep or server time-stamps. **Persisting state** beyond a single run or session is another aspect: for instance, an MMO game world persists on a database so that if you log in tomorrow, the world reflects changes from yesterday. AR clouds persist spatial anchors so that returning to a location days later, the same AR content reappears [75] [76]. Persistence introduces the need for **data storage and versioning** – how to update the world model when the world changes, and how to perhaps maintain a history of changes. Few systems other than databases tackle this well; robotics and AR typically overwrite old state with new (though one might keep a log). Future world models might incorporate more **memory of past states** to, for example, allow querying how an environment changed over time or enabling an AI to reminisce previous contexts.

- **Partial Observability and Incompleteness:** Incomplete information is the norm outside of fully virtual environments. Robots cannot see through walls; AR devices only scan what the user points them at; language models only know what's in their training data or provided documents. Hence, world models must be robust to gaps. Robotics deals with this via probabilistic reasoning, conservative assumptions (e.g., assume unknown areas are blocked until proven clear for navigation), or active exploration (moving to gather new info). AR systems improve map completeness as more users scan an area – crowdmapping is a strategy to fill in holes in the world model (e.g., one user's scan might cover one side of a building, another user covers the other side, and combined they yield a full model). Language AI handles unknowns either by saying "I don't know" or by attempting to fetch information. A big failure mode for LLMs is **hallucination**, essentially the AI confabulating to fill gaps in its world model. Augmenting them with explicit world knowledge bases is a direct counter to that [42]. In games, partial observability is sometimes a game mechanic, but the underlying engine often has the full state (except in certain P2P cases where no single node has the entire state without communication). Techniques like fog-of-war are implemented by filtering the world model per agent. Another angle is **uncertainty**: real-world models benefit from representing uncertainty (e.g., probability distributions), while game world states are usually deterministic or have well-defined randomness. Bridging these, some advanced simulators might include uncertainty to mimic real sensors for training robots (domain randomization).

- **Multi-Agent Coordination and Consistency:** When multiple agents (humans or machines) share a world, the world model becomes the *medium of coordination*. All agents should agree on certain facts (e.g., "object O is at location L") or at least have a mechanism to resolve discrepancies (perhaps through communication or reference to an authoritative source). In multi-robot systems, communication of world state is crucial if they operate in close proximity – they might share pose

estimates of objects so they don't both pick up the same item, for example. Shared world models like knowledge graphs [22] or distributed ledgers have been proposed to ensure each agent is on the same page. In AR, multiple user devices achieve consistency by aligning to the same anchor frame; any new AR content or change one user makes (like moving a virtual object) must be sent to others, typically via a server that merges updates and redistributes them. This resembles multiplayer game state synchronization. Conflicts can occur (what if two users try to grab the same virtual object?) and are resolved by application logic (maybe one user is designated "owner" of that object's state at a time). Language agents (like conversational AIs) might coordinate by sharing a conversational world model – for instance, in a multi-bot customer service scenario, one bot should know what another bot already told the user, to avoid repetition or contradiction. Ensuring consistency might involve a shared session memory or a centralized dialogue manager that updates the state (e.g., "issue X is resolved, user provided info Y"). Without such coordination, agents can quickly diverge and confuse the human user. In games, coordination is handled by the server enforcing rules – players or AI may desynchronize on their local view, but the server's version of the world state is the final word and is propagated out to everyone. If two players attempt a conflicting action (both pick up the same item at the same time), the server's world model (with perhaps a lock or atomic update on that item state) will allow one and reject the other, keeping the world logically consistent. The analogy in robotics would be a central coordinator that arbitrates, and in distributed AI systems, perhaps a consensus algorithm or central memory as arbiter.

- **Scalability of Information and Computation:** World models can become extremely large – think of mapping an entire city in AR, or an MMO world with millions of objects, or an enterprise knowledge base with millions of facts. Handling this amount of information requires architectural strategies. We've seen that gaming uses sharding and distributed workers to scale computation [69]. AR will likely use **edge computing** and hierarchical maps (perhaps dividing the world into regions, so your device only loads the local region's model, not the whole Earth at once). Robotics might leverage cloud backends to store maps of facilities and stream relevant parts to robots as needed (akin to how a robot vacuum downloads the floor plan of a house from the cloud). Language models dealing with long-term knowledge may integrate with databases that index information for efficient retrieval (vector indices, graph queries). The notion of a **unified but distributed** world model is useful – logically one world, but partitioned in storage. Cloud robotics and IoT concepts often involve a digital twin for each asset, and a higher-level model that links them; this federated approach is scalable. Additionally, techniques like level-of-detail or abstraction help – not every agent needs every detail. A robot team might share a high-level map outline while each keeps fine local details to itself, syncing critical info. In games, far-away regions might be "out of core" (not loaded) until players approach, to save memory, which is a form of partial world model per client. Consistency is maintained by loading authoritative data when needed.

- **Heterogeneity of Actors:** Increasingly, world models are being accessed by a mix of humans and machines. In an AR scenario, humans see the augmentations visually, while an AI agent might query the same world model via an API. For example, a service robot in a smart building might use a shared digital twin that is also used by AR glasses worn by maintenance staff – the robot might ask "where is valve #3?" and get a coordinate from the digital twin, while the human sees a highlighted valve in AR from the same model. This means the representation must be **interpretable by multiple types of agents**. Semantic labels are human-friendly ("Valve 3 in Room 2A"), coordinates and geometry are machine-friendly; a good world model provides both. Likewise in language, if a human curator corrects the knowledge in a knowledge graph that an AI uses, the AI's responses will improve

– a symbiosis via the world model. A challenge here is creating interfaces and standards such that different systems can plug into the world model without ambiguity. In robotics, projects like ROS (Robot Operating System) have standardized messages for sharing map data or TF coordinate frames as a form of world state broadcast. In AR, standards like **OpenXR and OpenARCloud** aim to standardize how spatial anchors and map data are represented, so that different devices share anchors reliably [31].

In conclusion, world models are a unifying concept that, despite domain-specific differences, serve the same purpose: **to maintain a coherent, shared understanding of "what is where and what is happening"** in a given environment. Whether it's a robot remembering where it left a tool, an AR system letting two people see the same virtual object, a conversational AI recalling a user's preference, or a game server tracking every character in a virtual city – all these are powered by representations that model the world's state. The state of the art has advanced significantly in recent years: robotics has formalized design patterns for world models [14], XR has made persistent shared AR feasible at scale [27], language AI is extending its context via external knowledge [43], and gaming has pushed the envelope of real-time distributed simulations [69]. The ongoing research and development continue to tackle the grand challenges of grounding, consistency over time, and multi-agent scalability. By learning from each other – e.g. applying game networking techniques to multi-robot coordination, or using knowledge graphs from AI in robotic reasoning – these fields are collectively moving toward more **robust and intelligent world models** that can underpin long-term, multi-actor interactions in complex environments. Each domain contributes a piece of the puzzle, and together they hint at the future of pervasive world models: perhaps a **universal shared model** that seamlessly connects humans, machines, and virtual worlds with a common understanding of reality.

**Sources:** World model concepts and definitions in robotics [1] [2]; examples of robotic world models and design considerations [13] [10]; AR Cloud and persistent AR technology [3] [28]; XR shared world aggregation [34]; integration of knowledge bases with language models [43] [4]; game engine and distributed world state architectures [59] [69].

---

1 2 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 74 Robotic world models—conceptualization, review, and engineering best practices - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC10652279/

3 24 25 26 27 31 32 37 38 73 75 76 Augmented Reality
https://codora.io/augmented-reality-2025-immersive-experiences/

4 39 40 41 42 43 49 50 51 52 53 54 55 AI Memory Vs Context Understanding | AI Guide
https://www.sphereinc.com/blogs/ai-memory-and-context/

22 KG-MAS: Knowledge Graph-Enhanced Multi-Agent Infrastructure for ...
https://www.researchgate.net/publication/396462254_KG-MAS_Knowledge_Graph-Enhanced_Multi-Agent_Infrastructure_for_coupling_physical_and_digital_robotic_environments

23 KG-MAS: Knowledge Graph-Enhanced Multi-Agent Infrastructure for coupling physical and digital robotic environments
https://arxiv.org/pdf/2510.10325

28 29 30 Multiplayer AR — why it's quite hard | by Matt Miesnieks | 6D.ai | Medium
https://medium.com/6d-ai/multiplayer-ar-why-its-quite-hard-43efdb378418

33 34 35 36 (PDF) DiOS -- An Extended Reality Operating System for the Metaverse
https://www.researchgate.net/publication/357733375_DiOS_--_An_Extended_Reality_Operating_System_for_the_Metaverse

44 Introducing KBLaM: Bringing plug-and-play external knowledge to ...
https://www.microsoft.com/en-us/research/blog/introducing-kblam-bringing-plug-and-play-external-knowledge-to-llms/

45 46 [2305.16291] Voyager: An Open-Ended Embodied Agent with Large Language Models
https://arxiv.org/abs/2305.16291

47 48 Computational Agents Exhibit Believable Humanlike Behavior
https://hai.stanford.edu/news/computational-agents-exhibit-believable-humanlike-behavior

56 Reasoning with Language Model is Planning with World Model
https://openreview.net/forum?id=VTWWvYtF1R

57 Large Language Model: world models or surface statistics?
https://thegradient.pub/othello/

58 59 60 65 66 67 68 69 70 71 72 The Entity-Component-Worker architecture and its use on massive online games
https://www.gamedeveloper.com/programming/the-entity-component-worker-architecture-and-its-use-on-massive-online-games

61 62 63 64 Mastering Multiplayer Game Architecture: Choosing the Right Approach - Getgud.io
https://www.getgud.io/blog/mastering-multiplayer-game-architecture-choosing-the-right-approach/