ChatGPT

Agentic AI: Foundations and Developments

Theoretical Foundations: In AI and cognitive science, an *agent* is typically defined as an entity that perceives its environment and autonomously takes actions to achieve goals [1]. At a basic level, "an agent is any entity able to act…to produce some causal effect and change in its environment" [2]. Formal definitions emphasize goal-directed behavior: for example, Russell & Norvig define an intelligent agent as one that **perceives** and **acts**, improving performance over time [1]. More recently, the term *agentic AI* has emerged to describe agents that not only react but proactively pursue goals over extended periods [3]. Thus, an agentic AI system integrates **autonomy** (self-directed action), **proactiveness** (goal-oriented initiative), **reactivity** (responding to environmental stimuli), and **social ability** (interacting with other agents) [4] [5]. Autonomy means an agent controls its own decisions without direct outside influence [4]. Reactivity implies continual perception-action loops that update beliefs in light of new inputs [4]. Proactivity means taking initiative and planning toward future goals [6]. Social ability requires communication or interaction protocols so agents can coordinate or compete with peers [6]. These four characteristics (as identified by Wooldridge et al.) distinguish intelligent agent systems from passive programs [7] [5].

Types of Agents: Classical AI distinguishes several agent types. **Reactive agents** (or *simple reflex agents*) choose actions based solely on current percepts via hard-coded condition-action rules [8]. They lack internal models or memory, acting as stimulus-response devices (e.g. a thermostat or simple robot avoiding obstacles) [8]. By contrast, **deliberative agents** (or model-based agents) maintain an internal representation of the world and use planning or reasoning to decide actions [9]. They can strategize by considering many possible future states (for example, path-planning in navigation) [9]. **Hybrid agents** combine reactive and deliberative layers: a fast reflexive subsystem handles urgent, time-critical reactions, while a higher-level planning subsystem handles long-term objectives [10] [11]. In a hybrid architecture (sometimes called *subsumption* or layered control), simple behaviors (e.g. "avoid obstacle") form the base layer, and higher layers (e.g. "wander" or "explore") subsume or override lower layers as needed [12] [13]. This yields robust, real-time behavior while retaining planning abilities. Finally, **learning agents** incorporate machine learning to improve over time. A learning agent starts with basic knowledge and adapts based on experience [14]. It typically has components for executing actions (performance element), evaluating results (critic), and updating its strategy (learning element), sometimes with a *problem generator* to explore new behaviors [14]. For example, a reinforcement-learning agent uses reward feedback to refine its policy.

Agent Architectures: Various architectures have been proposed. The **Belief–Desire–Intention (BDI)** model is a prominent cognitive architecture inspired by human practical reasoning [15]. In BDI, an agent maintains a set of *beliefs* (its internal model of the world), *desires* (states it would like to achieve), and *intentions* (desires the agent commits to pursuing). A typical BDI loop involves: updating beliefs from perceptions, generating goals (desires) consistent with beliefs and existing intentions, filtering which desires to commit to (intentions), and devising plans to achieve those intentions [15]【72†】. The following diagram illustrates a BDI architecture:

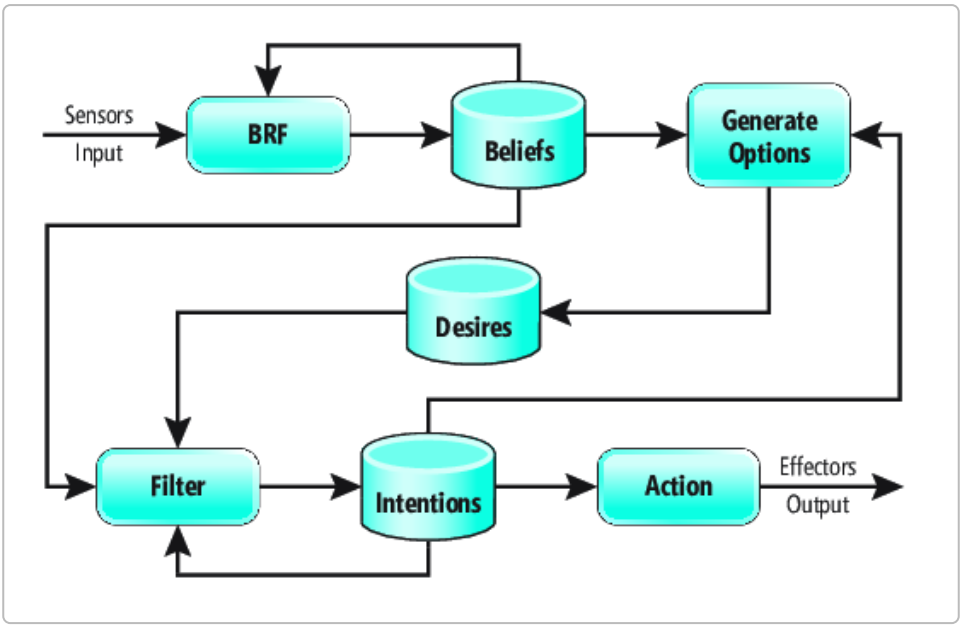

Figure: Schematic of a BDI agent. Percepts update the Beliefs database via a belief revision function (BRF). The agent generates options (potential desires) from beliefs and intentions, filters these into concrete Intentions, and then plans Actions to achieve them [15] [16] .

Another notable architecture is **subsumption** (Brooks, 1986), which organizes behavior in layered finite state machines without central control [12] [13] . In subsumption, low-level reactive behaviors (e.g. "avoid obstacle") operate continuously; higher layers (e.g. "wander around") can suppress or inhibit lower layers when higher-level goals demand it [12] [13] . This bottom-up design yields highly robust, real-time interaction with dynamic environments, though it lacks rich memory or symbolic reasoning [17] . A third paradigm is **utility-based** (or rational) agents, which select actions to maximize an explicit *utility function* [18] . Unlike goal-based agents that seek any state satisfying a goal, a utility-based agent evaluates how desirable each outcome is and chooses the action with highest expected utility [18] . This approach (common in economics and decision theory) yields consistently rational behavior under uncertainty. In practice, complex agents often blend these styles (e.g. using utility metrics within a BDI framework) to balance reactivity, goal reasoning, and optimization.

Software-Based Agentic Systems

The rise of large language models (LLMs) has given birth to a new class of *software agents* that wrap a conversational model in an autonomous decision loop. Early 2023 saw the debut of open-source "agentic" frameworks like **AutoGPT** and **BabyAGI**, which let users specify a high-level goal and then use GPT-4 (or similar models) to plan and act until the goal is met [19] [20] . These systems typically share a common architecture: the LLM is prompted with the agent's goal and instructions and outputs an action plus reasoning ("thought") at each step. The framework executes the action (for example, using web APIs or calling code), collects the result, and feeds it back to the LLM for the next iteration [21] . This *Thought→Action→Feedback* cycle extends classic ReAct prompting, allowing the agent to call tools or external knowledge sources between reasoning steps [22] [21] .

For example, **AutoGPT** (Significant Gravitas) spins up GPT-4 with a system prompt defining an autonomous agent role and a user prompt with the goal [19] [23] . In practice, AutoGPT breaks a task into sub-tasks: it chains together GPT-generated plans and uses internet search or code execution to complete each step [19] [23] . It maintains short-term memory in context (the chat history) so subsequent reasoning can build on earlier results [24] . Similarly, **AgentGPT** and **AgentVerse** follow this loop model.

In contrast, **BabyAGI** (Yohei Nakajima's script) introduced a minimalist *task loop*: the agent executes one task at a time, then creates new tasks from the outcome, and reprioritizes its queue [25]. Crucially, BabyAGI uses an external vector database (e.g. Pinecone) to embed and store past results, so it can recall relevant information on later loops [26]. This gives BabyAGI rudimentary long-term memory across the planning process.

ReAct prompting (Yao et al., 2023) is an influential paradigm used by many LLM agents. ReAct interleaves chain-of-thought reasoning with environment *actions* (API calls) so the model can update its plans based on real-time data [22]. For example, an agent using ReAct might think through a math problem step-by-step, pause to query a knowledge API for a fact, and then incorporate that result into its plan [22]. Studies show ReAct not only improves task success but also makes the process more interpretable to humans [27]. Many agent frameworks implicitly implement ReAct: the LLM's output includes both a "thought" and an action command, and the framework executes the command before continuing.

A notable embodied agent example is **Voyager** (Wang et al., 2023), an LLM-powered lifelong learning agent in a Minecraft environment [28]. Voyager continually explores its world, learns new skills, and adds them to a growing *skill library* (executable Python code). It uses an "automatic curriculum" to select exploration goals and an iterative prompting method: after executing each step in Minecraft, it uses the LLM to reflect on the outcome, correct errors, and refine its code [28] [29]. This approach lets Voyager acquire complex, compositional behaviors (like building machinery) without human coding. In effect, Voyager combines planning (deciding what to do next), memory (storing learned skills), and tool use (executing code) within an open-ended game environment.

Agent frameworks and libraries have also proliferated. **LangChain** is a popular Python framework for building LLM applications, including agents [30]. It provides modular components for prompts, memory modules, and tool interfaces. For instance, LangChain's *AgentExecutor* can orchestrate a single agent with a set of tools (e.g. search, code interpreter). Its extension **LangGraph** allows constructing graphs of specialized agents: each node in the graph is an LLM agent with its own prompt and possibly even a different model, and messages flow between them along defined edges [31]. This enables complex multi-agent workflows (for example, one agent extracts data, another summarizes, another makes decisions), with explicit control of information flow [32].

Microsoft's AutoGen is another agentic framework (open-source) designed for multi-agent workflows [33]. AutoGen v0.4 adopts an asynchronous, event-driven architecture: agents communicate via messages, allowing both request/response and fire-and-forget patterns [34]. It provides pluggable components for memory, tools, and models, and built-in observability (logging and tracing) for debugging agent interactions [35]. These features help scale to distributed systems and long-running agents. Other emerging tools include Hugging Face's **CAMEL** and **OpenAI Swarm** (a coordination API), though these are evolving rapidly.

Comparative Table of Agentic Frameworks: Below we compare key software frameworks and tools for agentic AI systems:

Framework / Tool	Description	Capabilities	Use Cases / Strengths	Limitations
OpenAI Function API	LLM (GPT-4/ GPT-3.5) with function calling and Plugins.	High-quality LLM reasoning plus tool calls (web API, code execution, etc).	Easy access to world knowledge and actions (search, browser, calculator). Good models.	Requires custom code/logic around the LLM; limited built-in agent loop or memory.
LangChain (+LangGraph)	Modular Python library for LLM apps/ agents [30].	Supports chains, memories, tool integration, agent executors, multi-agent graphs [31].	Highly flexible: choose prompts, tools, memory backend. LangGraph enables orchestrating multiple specialized agents.	More complex to set up; overhead of library layers; still requires developer to design agent loops.
Microsoft AutoGen	Open-source multi-agent framework [33].	Asynchronous agent messaging, role-based agents (e.g. “Director”, “Worker”), pluggable memories and tools.	Designed for multi-agent collaboration. Built-in observability, robust event-driven scheduling.	Newer project; developer community still growing; learning curve.
BabyAGI / AutoGPT	Example open-source agent programs using GPT-4 [19] [25].	Single-agent loops breaking tasks into sub-tasks. Uses vector DB memory (BabyAGI) or context window (AutoGPT) to remember past steps.	Easy to try as scripts; demonstrated open-ended task solving.	Very limited control/ oversight. Often brittle and hallucinating; not officially supported or tested at scale.
Hugging Face Agents / Swarm	Ecosystem of LLM agent tools (e.g. AgentOS, DeepSEEK).	Pre-built components, community models. Supports chaining and multi-agent.	Rapid experimentation, some standardized agent templates.	Still maturing; non-uniform standards; potential security/privacy concerns.

(Sources: AutoGPT documentation and wiki [19] [24]; analysis by Tech_with_KJ [25] [21]; Microsoft AutoGen docs [34]; LangChain blog [30] [31].)

In all these systems, memory and knowledge retrieval are key. Agents often use vector databases (e.g. Pinecone, Weaviate) to embed previous observations and recall them by similarity [26]. Tools/APIs extend their capabilities: e.g. search engines, databases, code interpreters, and even robotic control APIs can be invoked as actions. The LLM generates pseudo-code or structured queries, which the system executes and returns to the agent’s context. Thus, planning and reasoning are typically

implemented as LLM prompt engineering (chain-of-thought templates) inside an external execution loop [21].

Robotics and Physical Agents

In embodied domains, agentic AI controls physical robots. An **autonomous robotic agent** integrates perception, planning, and control. It fuses data from multiple sensors (cameras, LiDAR, radar, IMUs, etc.) into a coherent world model (often using Kalman filters, SLAM algorithms, or deep learning) [36]. Control modules implement feedback loops (e.g. PID controllers or learned policies) to actuate motors. High-level planners then reason over the fused sensor data to set goals or trajectories (e.g. path planners for navigation). This layered architecture (sensing → world model → planning → control) echoes the agentic principles: it is *reactive* to new sensor inputs yet *proactive* in pursuing waypoints and tasks. Modern autonomous vehicles, for example, use machine learning to interpret lidar and camera inputs into object detections, then employ motion planning algorithms to chart safe paths through traffic. In manufacturing, mobile warehouse robots (e.g. Amazon's Kiva) combine real-time obstacle avoidance with a central task-dispatcher to optimize inventory retrieval. Drones and robot swarms apply similar ideas: each drone has onboard fusion of GPS, lidar, and vision for collision-free flight, while decentralized coordination protocols allow swarms to self-organize tasks (such as area coverage or search) without centralized control.

Example: Autonomous cars (Waymo, Tesla) use sensor fusion (camera+lidar) to perceive environment and sophisticated planning stacks for lane-following and obstacle avoidance. Warehouse robots use agents to navigate dynamic environments. Swarm drones mimic insect colonies: simple flight controllers with local rules (e.g. maintain distance, align velocity) yield complex flocking and formation flying. Importantly, robotics inherits all four agent characteristics: robots act *autonomously* in real-world environments, they *react* to sensor inputs continuously, they *proactively* plan routes, and in multi-robot settings they exhibit *social* coordination (via communication or stigmergy). Recent research emphasizes *sensor fusion*: combining heterogeneous data with AI models is crucial for reliable perception [36]. Overall, agentic AI in robotics leads to self-guided machines that can adapt to new environments with minimal human oversight.

Multi-Agent Systems (MAS)

When multiple agents operate together, they form a *multi-agent system* (MAS), which can solve problems beyond the scope of any single agent [37]. In a MAS, each agent may have partial information or specialized skills, and agents must communicate and coordinate. **Communication protocols** are therefore vital. Agents often use standardized agent communication languages (ACLs) such as **FIPA-ACL** or **KQML**, which specify message formats (syntax, semantics, pragmatics) for information exchange [38]. Communication may be **explicit** (direct message-passing: requests, offers, data sharing) or **implicit** (signaling via the environment, e.g. leaving markers) [39]. Protocols like the *Contract Net* allow dynamic task allocation: a manager agent announces a task, and worker agents bid to take it. Auction-based and market-based schemes are common for resource sharing. Predefined protocols (set by developers) coexist with **emergent protocols** that agents learn or adapt over time [40]. Recent work even proposes new agent protocols (LangChain's Agent Protocol, Anthropic's Model Context Protocol, etc.) to standardize multi-agent coordination [41].

Coordination strategies include negotiation, consensus, and leader election. For instance, a team of robots exploring a building might use consensus algorithms to agree on an area map. In swarm robotics, simple local rules yield global coordination: e.g. each robot maintains distance from neighbors and shares a "task found" signal, which leads to emergent foraging or coverage behavior. Multi-agent

planning can be centralized (one agent plans for all) or decentralized (each agent plans locally while considering others' roles).

A hallmark of MAS is **emergent behavior**. Swarm intelligence (SI) is the study of how simple agent interactions produce intelligent global patterns [42]. In SI, agents typically follow simple rules and interact locally [43]. For example, birds in a flock each align with nearby neighbors, yielding cohesive flocking without any bird having a map of the group. Ant colony optimization algorithms let simulated ants find shortest paths via pheromone-like shared state. These natural metaphors inspire algorithms in engineering. In a MAS, unexpected group-level intelligence can emerge: Google's data centers used "ant" algorithms to optimize network routing, and swarms of drones can spread out to cover an area more efficiently than any solo unit. Importantly, MAS can solve problems via **collective intelligence**: multiple agents pooling knowledge can navigate complex spaces or simulate economies in ways single agents cannot [37] [42].

Applications: MAS are used in simulations (e.g. training agent teams in virtual environments), in economics (agent-based market and consumer simulations), in defense (UAV swarms and distributed sensor networks), and collaborative AI (multiple AI assistants working on subtasks). For example, multi-agent reinforcement learning (MARL) has been applied to games like StarCraft and real-world traffic light control. Swarm robots can autonomously assemble structures or perform search-and-rescue. Even on the web, simple "bots" acting as agents can coordinate (some use Ethereum smart-contract "agents" for decentralized tasks). In each case, communication and coordination enable tasks (like network routing, traffic management, or resource allocation) that exploit the parallelism and diversity of the agent team.

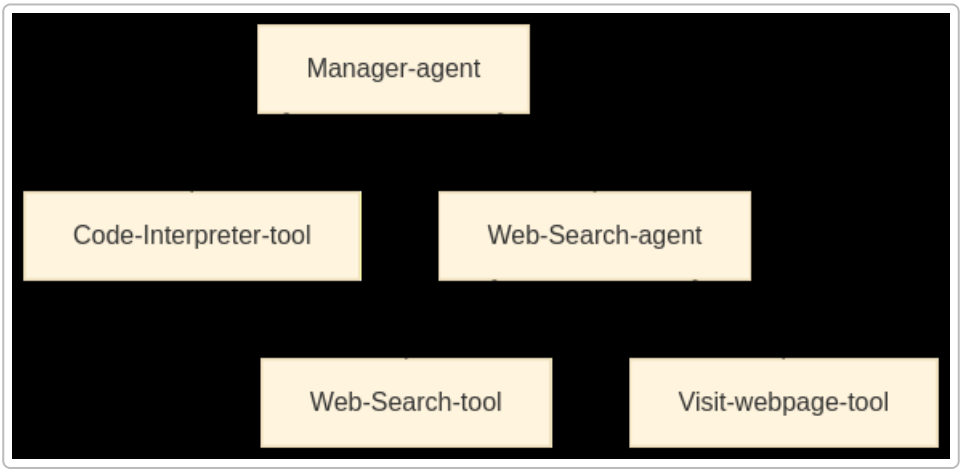

Figure: Example multi-agent architecture. A Manager/Orchestrator agent delegates subtasks to specialized agents (here a Code-Interpreter agent and a Web-Search agent). The web-search agent itself uses tools (e.g. a search API and page-fetcher) to gather information. Such hierarchies allow dividing complex tasks across agents with different roles [44].

Future Directions and Challenges

As agentic AI advances, several major challenges and trends emerge:

- **Scalability:** Real-world tasks may require orchestrating many agents and vast knowledge bases. Frameworks are evolving to support large-scale MAS. For example, AutoGen's asynchronous, event-driven design is intended to scale to distributed networks of agents across machines [34]. Cloud platforms allow running hundreds of agents in parallel. However, managing state and

coordination overhead grows with scale, and research is needed to efficiently allocate resources among agents without bottleneck.

- **Safety and Alignment:** Autonomous agents acting in the real world pose safety risks. Potential hazards include unexpected behavior, privacy breaches, or malicious exploitation [45]. Recent work highlights the lack of safety audits in many agent projects [46]. New guardrail libraries (e.g. NVIDIA's NeMo Guardrails) are being developed to enforce constraints on LLM outputs and limit harmful actions. Standards and regulations are also being proposed. Ensuring that agents reliably obey user intentions and do not override human control is a major open problem (part of the broader AI alignment issue) [45].

- **Interpretability and Trust:** Complex agent pipelines (LLMs + many tool calls) can be opaque. Explaining why an agent chose a given action is difficult. Approaches like chain-of-thought (e.g. ReAct) improve transparency by exposing intermediate reasoning [27]. However, post-hoc auditing of agent decisions remains essential, especially in high-stakes domains (e.g. medicine or law). Tooling for tracking an agent's reasoning trace and verifying each step (as in AutoGen's observability features [35]) will be important.

- **Multi-Modal and LLM Integration:** Future agents will combine multiple modalities (vision, audio, touch) with language. Already, models like GPT-4 integrate image understanding, enabling agents to *see* and *talk*. Embodied agents (in robotics or game worlds) combine LLM "brains" with sensorimotor perception. We expect more research on agents that can navigate and manipulate the physical world under LLM guidance. For instance, Vision-Language Navigation agents use LLMs to interpret directions and control robot motion. Integrating structured knowledge (e.g. knowledge graphs) with LLM reasoning is another promising direction for richer agency.

- **Agent Simulation Environments:** Simulators for agents are rapidly improving. **Meta's Habitat** (and similar platforms like AI2-THOR, CARLA) provide realistic 3D environments for embodied agents [47]. Habitat 3.0 even supports simulating human-robot collaborative tasks. Training agents in such worlds (rather than on static datasets) allows them to learn sensorimotor skills and long-term planning by trial and error. In parallel, research on "generative agents" has created sandbox social worlds where hundreds of LLM-driven NPCs interact in human-like ways [48]. For example, Park et al. built a virtual town of 25 generative agents that form memories, initiate conversations, and coordinate events purely via an LLM-based architecture [48]. These environments demonstrate how agents can simulate complex social behaviors. Going forward, we expect agent researchers to develop open benchmarks and simulators (e.g. 3D worlds, multi-agent games, digital twins of real systems) to rigorously evaluate agent behavior, collaboration, and long-term autonomy.

In summary, agentic AI unites classical AI agent theory with modern LLM-driven techniques. It encompasses a spectrum from purely software "agents" (AutoGPT, LangChain bots) to physical embodied robots, and from single-agent systems to rich multi-agent ecologies. The field draws on cognitive architectures (BDI, subsumption, utility) and cutting-edge ML. As agents become more capable, building them at scale demands robust architectures, standards for communication (some already emerging [41]), and principled safety measures [45]. The integration of multi-modal perception and long-term memory is crucial for next-generation agents. Finally, powerful simulation environments (e.g. Habitat, Minecraft, digital twins) will be essential for training and validating agentic AI. By combining clear theoretical foundations with these emerging tools and frameworks, researchers aim to create autonomous AI systems that are powerful yet understandable, collaborative yet controllable.

Sources: Key concepts and definitions are drawn from AI literature [1] [4]. Agent characteristics are detailed by Wooldridge et al. [4]. Reactive/deliberative/hybrid agents are described in AI textbooks and tutorials [8] [9] [10]. Architectures like BDI and subsumption are documented in classical AI research [15] [12]. Details of modern LLM-based agents come from recent articles, whitepapers, and the open-source community [19] [25] [34]. Robotics content is supported by recent robotics surveys [36]. Multi-agent and swarm concepts use standard AI references [38] [42]. Future directions cite current work on simulation and generative agents [47] [48] [45]. All information is up-to-date as of early 2025.

---

[1] [3] [41] [45] [46] Intelligent agent - Wikipedia
https://en.wikipedia.org/wiki/Intelligent_agent

[2] [4] [5] [6] [7] [15] loa.istc.cnr.it
https://www.loa.istc.cnr.it/wp-content/uploads/2019/10/FerrarioOltramariFOIS2004.pdf

[8] [9] Reactive vs Deliberative AI Agents | GeeksforGeeks
https://www.geeksforgeeks.org/reactive-vs-deliberative-ai-agents/

[10] [11] SmythOS - Understanding Hybrid Agent Architectures
https://smythos.com/ai-agents/agent-architectures/hybrid-agent-architectures/

[12] [13] [17] Subsumption architecture - Wikipedia
https://en.wikipedia.org/wiki/Subsumption_architecture

[14] Agents in AI | GeeksforGeeks
https://www.geeksforgeeks.org/agents-artificial-intelligence/

[16] Machine Learning - Leveraging the Beliefs-Desires-Intentions Agent Architecture | Microsoft Learn
https://learn.microsoft.com/en-us/archive/msdn-magazine/2019/january/machine-learning-leveraging-the-beliefs-desires-intentions-agent-architecture

[18] Utility-Based Agents in AI | GeeksforGeeks
https://www.geeksforgeeks.org/utility-based-agents-in-ai/

[19] [24] AutoGPT - Wikipedia
https://en.wikipedia.org/wiki/AutoGPT

[20] [21] [23] [25] [26] [30] [31] [32] Agentic AI: AutoGPT, BabyAGI, and Autonomous LLM Agents — Substance or Hype? | by Tech_with_KJ | Apr, 2025 | Medium
https://medium.com/@roseserene/agentic-ai-autogpt-babyagi-and-autonomous-llm-agents-substance-or-hype-8fa5a14ee265

[22] [27] [2210.03629] ReAct: Synergizing Reasoning and Acting in Language Models
https://arxiv.org/abs/2210.03629

[28] [29] [2305.16291] Voyager: An Open-Ended Embodied Agent with Large Language Models
https://arxiv.org/abs/2305.16291

[33] [34] [35] AutoGen - Microsoft Research
https://www.microsoft.com/en-us/research/project/autogen/

[36] Sensor-Fusion Based Navigation for Autonomous Mobile Robot
https://www.mdpi.com/1424-8220/25/4/1248

[37] [38] [39] [40] Communication in Multi-agent Environment in AI | GeeksforGeeks
https://www.geeksforgeeks.org/communication-in-multi-agent-environment-in-ai/

[42] [43] Swarm intelligence - Wikipedia
https://en.wikipedia.org/wiki/Swarm_intelligence

44 Multi-Agent Systems - Hugging Face Agents Course
https://huggingface.co/learn/agents-course/en/unit2/smolagents/multi_agent_systems

47 AI Habitat
https://aihabitat.org/

48 [2304.03442] Generative Agents: Interactive Simulacra of Human Behavior
https://arxiv.org/abs/2304.03442