ChatGPT

# Self-Assembling AI Agent Architectures for Dynamic Corporate Knowledge Systems

## Introduction

Modern enterprises are exploring **self-assembling AI agent systems** that can dynamically construct, update, and share a collective knowledge base in real time. These systems combine the autonomy of multi-agent frameworks with the richness of **knowledge graphs** to capture corporate knowledge and context. The goal is to enable AI agents to ingest and embed corporate data, generate contextually relevant prompts for each other, and track performance – all while maintaining secure knowledge flows across the organization. Recent advances in **agent orchestration** and knowledge graph technology have made such architectures feasible [1] [2]. This report delves into current implementations, use cases, research publications, integration approaches, and proposes a conceptual architecture (and potential patent outline) for a multi-agent knowledge graph system in enterprise settings. Short, structured sections with clear headings are used for readability, and key concepts are summarized in tables and lists where appropriate.

## Current and Emerging Multi-Agent Knowledge Frameworks

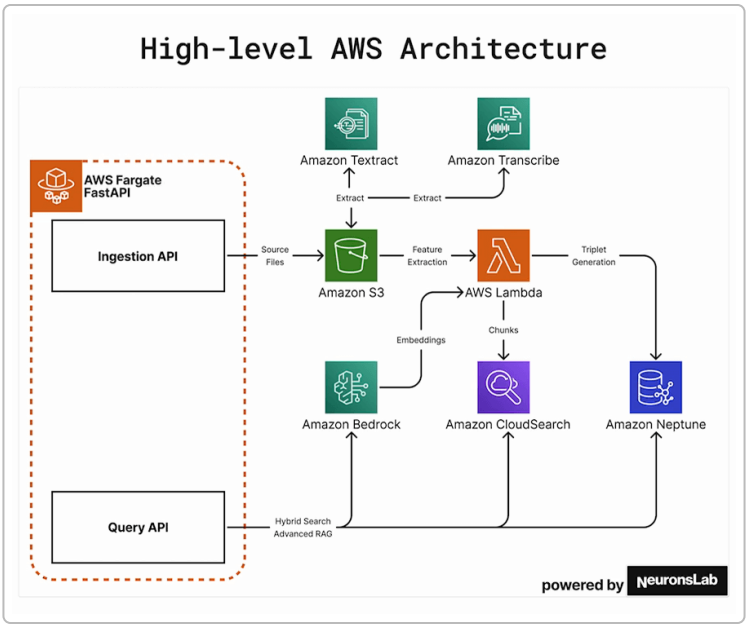
Multiple architectures and frameworks now enable *agentic AI systems* to build and share knowledge in real time. These frameworks blend large language models (LLMs) with tool integration, multi-agent collaboration, and knowledge graph memory. Below we describe some notable implementations and emerging patterns:

- **Graphiti (Zep Memory)** – *Graphiti* is an open-source framework designed for AI agents to build and query **temporally-aware knowledge graphs** in dynamic environments [2]. Unlike traditional static Retrieval-Augmented Generation (RAG) on documents, Graphiti **continuously integrates** new inputs (e.g. user chats, enterprise data streams) into a coherent, queryable graph [3]. It supports incremental updates and historical queries without full recomputation, allowing agents to "remember" and reason over evolving corporate knowledge. Graphiti powers the memory layer of Zep AI and has demonstrated state-of-the-art agent memory performance in benchmarks [4]. In essence, it gives AI agents a living knowledge network that they **autonomously expand and update** as new information arises [5]. This live graph memory helps reduce hallucinations and maintain context over long interactions.

- **SAP's Joule Copilot** – Enterprise software vendor SAP has introduced a multi-agent copilot ("Joule") that orchestrates specialized AI agents for business tasks [6]. Joule acts as a *conductor* agent coordinating an ensemble of domain-expert agents (finance, scheduling, HR, etc.), each trained for specific functions [6]. Together, they collaborate to execute complex workflows (e.g. resolving an invoicing dispute or preparing financial closing reports) by sharing context and results. SAP has also unveiled an **SAP Knowledge Graph** service to ground these agents in business semantics and data relationships [7]. This enterprise knowledge graph connects core business objects (purchase orders, invoices, customers, etc.) with their interrelationships, providing a **common, graph-based memory**

that all the agents can draw from [8]. According to SAP, this grounding in a shared ontology reduces irrelevant or incorrect outputs and allows more reliable generative AI applications [8]. In SAP's architecture, the human user remains "in the loop" as the composer of requests, Joule is the orchestrator, and the specialist agents are the musicians executing subtasks [9].

- **Neurons Lab Agentic System** – The AI consultancy Neurons Lab described a multi-agent system using a real-time knowledge graph to power a better customer service chatbot [10] [11]. In their design, multiple AI agents handle different conversational roles (one agent focuses on progressing the dialogue, another on negotiating, another injects humor or handles objections, and another closes the deal) [11]. A top-level *orchestrator layer* manages these agents, ensuring they work in concert for a smooth customer experience [12]. Crucially, Neurons Lab avoids a purely vector-based memory; instead they maintain a **knowledge graph of product and customer data** (e.g. plans, pricing, usage patterns) to capture relationships and factual details [10]. This prevents errors like conflating similar items – for example, two plans with different data limits that might appear similar in an embedding vector space are explicitly distinct nodes in the graph [13] [10]. The knowledge graph (built on AWS Neptune in their stack) is updated nearly on the fly as data changes, and the agents' prompts are grounded in this structured memory, improving factual accuracy. The result is a **network of specialized agents** coordinated by an orchestrator and supported by a live knowledge graph, yielding more context-aware and correct responses.




*High-level architecture from an agentic system by Neurons Lab, illustrating how corporate data flows into both a knowledge graph (AWS Neptune) and vector index, enabling hybrid search (graph + embeddings) and LLM-based queries. Multiple AWS services (Textract, Transcribe, Lambda) preprocess data into triples (for the graph) and embeddings (for search). An orchestrator (via Bedrock LLM) can then use GraphRAG – combining graph relationships with retrieval – to answer complex queries or guide agent workflows.* [14] [10]

- **LangChain, LangGraph, and Other Orchestration Frameworks** – A number of open-source frameworks have emerged to help developers build multi-agent systems with shared context:

- *LangChain* provides abstractions to manage chains of LLM calls and tools, and can be configured for multi-agent setups (including hierarchical agent teams) [15]. It simplifies connecting agents to external data sources and orchestrating tool usage.
- *LangGraph* is a newer framework explicitly designed for **multi-agent orchestration with knowledge graphs**. In one use case, LangGraph was combined with a hybrid knowledge base (Neo4j graph + vector DB) to analyze complex legal case documents [16]. This "GraphRAG" approach uses graph-based queries for multi-hop reasoning (e.g. finding relationships between people and events in a case) alongside vector semantic search for relevant text [16].
- *AutoGen (Microsoft)* is a multi-agent conversation framework that enables agents to converse with each other, share tasks, and use tools collaboratively [17]. It provides a high-level API where agents (which could represent AI roles or humans or tools) send messages and results to each other. By automating the chat among agents, AutoGen lets them collectively solve tasks and even spawn new sub-agents autonomously [17].
- *CrewAI and others:* Several research prototypes and libraries (e.g. MetaGPT, ChatDev, BabyAGI) have explored "agent teams" that mimic human teams. For instance, MetaGPT spawns agents as software engineer, tester, project manager, etc., to cooperatively develop a software product. These emphasize prompt-generated specialization and division of labor among agents. While not all incorporate knowledge graphs, they inform patterns of **agent role assignment** and coordination in an unstructured way.

**Common architectural themes** across these implementations include: a **shared memory store** (often a knowledge graph or a hybrid KG+vector index) that all agents can read/write; a mechanism for **agent communication** (either a centralized orchestrator that routes information or peer-to-peer messaging with a protocol); and **specialized expert agents** that handle distinct tasks or domains, sometimes arranged in a hierarchy. The combination of these elements allows an agent network to *self-assemble* a knowledge base and maintain collective context. Table 1 summarizes a few of the current frameworks and their key features:

| Framework / System | Description | Notable Features |
|---|---|---|
| **Graphiti (Zep)** | Open-source memory layer using a temporal **knowledge graph** for agents [2]. | Real-time graph updates; historical queries; integrates unstructured chats & structured data [18]. |
| **SAP Joule Copilot** | SAP's multi-agent business copilot with domain expert agents [6]. Uses SAP **Knowledge Graph** for enterprise data grounding [7]. | Orchestrator (Joule) + specialized agents; human-in-loop; business ontology grounding to reduce errors [8]. |
| **Neurons Lab Agentic CX** | Multi-agent chatbot for customer experience with **orchestrator + specialist** agents [11] and a live knowledge graph memory [10]. | Graph-based memory for factual consistency; agents with distinct dialogue roles; AWS-based pipeline for real-time updates. |
| **LangGraph (GraphRAG)** | Framework for **multi-agent orchestration** with combined knowledge graph and vector retrieval [16]. Applied in complex info retrieval (e.g. legal cases). | Agents perform multi-hop reasoning via graph queries; hybrid search (graph + embeddings) for high precision results. |

| Framework / System | Description | Notable Features |
|---|---|---|
| **AutoGen (Microsoft)** | Programming framework for **conversational multi-agent** applications [17]. Agents (LLMs, tools, human proxies) converse to solve tasks. | Standardized agent messaging interface; supports tool use and dynamic agent creation; improves task success by collaboration. |

Table 1: Representative frameworks and systems enabling multi-agent knowledge graphs and orchestration.

## Use Cases in Corporate Environments

Agent-based knowledge systems have compelling applications across corporate functions. By embedding an organization's knowledge and workflows into a network of AI agents, companies can optimize processes in coaching, HR, software development, leadership, and more. Below we analyze a few key use cases and how such an agent system would function in each:

### Coaching and Employee Development

In a coaching scenario, an AI **coach agent** can continuously monitor an employee's interactions, performance metrics, and learning progress (with appropriate privacy safeguards). Using a shared knowledge graph of company training materials, performance data, and best practices, the coach agent can provide personalized guidance. For example, it might observe that a sales employee struggles with a product line and proactively surface a training module or talking points from the knowledge graph. In team settings, a *"Socratic" coach agent* (as explored by researchers at Rice University [19]) can analyze past team data and offer real-time collaboration tips. The agent network could include a **mentorship agent** that identifies skill gaps (by comparing an employee's project history to required competencies in the graph) and a **feedback agent** that generates periodic coaching reports. Together, these agents improve employee development by autonomously delivering the right knowledge at the right time. Such a system was shown to enhance teamwork by prompting reflection and suggestions in real time [19] [20]. The corporate knowledge graph here would embed training documents, SOPs, and even social network insights (who knows what) to tailor coaching to each individual.

### HR Process Automation and Optimization

**Human Resources** can benefit greatly from agentic AI systems that streamline recruiting, onboarding, talent management, and employee support. An example configuration is a **multi-agent HR assistant**: one agent acts as a conversational HR chatbot for employees (answering questions on policies, benefits, leave, etc.), while behind the scenes other agents handle specific tasks – e.g. a **policy analysis agent** fetches the latest policy documents from the knowledge graph, a **workflow agent** files the appropriate forms or updates HRIS records, and a **compliance agent** checks requests against regulations. These agents all share access to a corporate knowledge graph that contains HR policies, employee records (with privacy controls), org charts, and historical Q&A. This enables consistent and context-aware responses. IBM describes that a range of AI agents can be used in tandem for HR to autonomously execute workflows like talent acquisition or payroll, often turning HR into a more strategic function rather than administrative [21] [22]. With multi-agent orchestration, routine HR tasks (updating an address, answering benefits FAQs, scheduling trainings)

could be fully automated, freeing human HR staff for high-value work. Agents can also monitor HR data in real time: for instance, a **retention insights agent** could watch the graph for upticks in attrition signals and alert HR leadership with an analysis. By embedding all relevant HR knowledge and data securely in the graph, these agents ensure decisions are data-driven and consistent. Notably, IBM found that using AI self-service in HR can yield 50–60% cost savings in service delivery [23], illustrating the efficiency gains such agent systems can deliver.

## Developer Support and Feedback Loops

For software engineering teams, multi-agent systems can create continuous feedback loops and assistantship throughout the development lifecycle. One envisioned use case is an AI-driven **developer assistant suite**. Here, an agent integrated in the IDE (such as the Cursor IDE agent) helps with code generation and remembers the developer's preferences and project context across sessions [24] [25]. This agent would utilize a knowledge graph that ingests the project's codebase, documentation, coding standards, and past tickets. As the developer writes code, the agent can retrieve relevant knowledge (e.g. similar past implementations, known bug patterns) from the graph. Meanwhile, other specialized agents play complementary roles: a **code review agent** could analyze pull requests against the graph of best practices and prior bugs, providing annotated suggestions; a **test generation agent** could create unit tests by querying the requirements and edge-case history stored in the graph; and a **devops agent** might monitor build and incident data to give feedback on performance or errors. These agents coordinate through a common context – for instance, the IDE agent might call on the test agent when the developer finishes a function. Integration with tools like Cursor demonstrates how an **MCP (Model Context Protocol)** server can allow the IDE's AI agent to persist and query a knowledge graph of coding standards and project knowledge [25] [26]. This means the agent "remembers" the developer's style guides, API keys, and design patterns from the graph and can inject that context when generating code. Over time, the developer feedback loop becomes tighter: every bug fix or code review result updates the knowledge graph (e.g., marking a certain API usage as problematic), and the agents learn to avoid or flag similar issues in the future. The performance of each agent (e.g., how often code suggestions are accepted or lead to bugs) can be tracked to continually refine the system. In summary, a multi-agent, knowledge-centric assistant can boost developer productivity and code quality by ensuring **lessons learned are never lost** but instead are cyclically fed back into the project knowledge graph.

## Leadership Decision Support

Corporate leaders and executives can also leverage agent networks for data-driven decision support and strategy. In this scenario, the **agents assemble and analyze knowledge** from across the enterprise to provide timely insights. For example, a **"strategist" agent** could query a broad knowledge graph that spans financial reports, market intelligence, and internal performance metrics to answer a question like "What factors are driving a dip in product X sales this quarter?". That agent might break the problem into parts and invoke other agents: a **financial analysis agent** to parse recent sales data, a **marketing sentiment agent** to scan social media and customer feedback (ingested in the knowledge graph), and a **competitor monitor agent** to check news or knowledge base entries on competitor moves. Each specialized agent returns its findings (e.g., a drop in conversion rates linked to a specific region, or negative sentiment about a product feature). The higher-level strategist agent then synthesizes these inputs into a recommendation or report for the executive. Throughout this process, the corporate knowledge graph serves as a **single source of truth** linking data from different silos – CRM, ERP, support tickets, industry news, etc. – which the agents can traverse. This reduces the risk of each agent working with fragmented or contradictory data.

Additionally, an **advisory agent** could be set to proactively monitor the knowledge graph and alert leadership to anomalies or opportunities (for instance, detecting that the latest R&D results combined with market trends indicate a new product opportunity). Security and confidentiality are paramount here, so agents would be tiered by clearance: e.g., a **finance agent** only accesses financial nodes of the graph, while a **CEO's agent** might have broader read access but still follows policies on what it can output (preventing leaks of sensitive plans). By dynamically assembling insights from the knowledge network, these leadership advisor agents act like an AI-augmented analytics team that never sleeps. They enable faster, evidence-backed decisions in coaching C-suite strategy, with the ability to drill down on any rationale because the underlying knowledge graph retains the connections and sources for each insight.

## Related Patents and Research

The concept of dynamically assembled agent networks that share a knowledge graph is cutting-edge, and few patents directly cover the full architecture. However, several existing patents and publications hint at parts of this vision:

- **Temporal Knowledge Graph Memory (Zep)** – Rasmussen et al. (2025) introduced *Zep*, a memory service for AI agents built on a temporal knowledge graph [4] [27]. Their paper describes how Zep's core (Graphiti) **synthesizes unstructured conversations and structured business data into a unified graph** with historical context [4]. This allows agents to perform temporal reasoning and maintain long-term context far beyond LLM context windows. Zep significantly outperformed prior art (like MemGPT) on benchmarks of deep memory and cross-session consistency [28]. While this is a research paper (arXiv:2501.13956) and not a patent, it provides a blueprint for agent memory architectures which a patent could build upon.

- **Torch.AI's "Graph Compute Generated Knowledge Mesh"** – In 2024, Torch.AI was awarded U.S. Patent 11,842,285 for an **autonomous graph compute invention** that creates a "knowledge mesh" in real time [29]. The patented method performs in-memory AI to **autonomously decompose, fuse, and graph data streams in real-time**, rather than the traditional "store-then-analyze" approach [29]. Essentially, it's a system that ingests data on the fly and forms a knowledge graph (mesh) of entities and relationships, enabling immediate insights. This aligns with our use-case of continuously updating knowledge graphs. Although the patent text doesn't explicitly mention multi-agent orchestration, one can imagine each incoming data source or processing function as an agent in a broader pipeline. Torch.AI's focus is on hyperscale and defense applications, but the core idea of *real-time graph-based knowledge assembly* is directly relevant to corporate agent networks.

- **Agent Orchestration Patterns** – The general idea of **AI agent orchestration** (coordinating multiple specialized agents) has been discussed in various whitepapers and a few patents. IBM, for example, has published guidance on multi-agent orchestration where a higher-level agent manages task assignment and data flow among sub-agents [30]. One older patent (US20120158633A1, 2012) defined a knowledge graph-based search system with distinct agent roles like *initiator, negotiator, closer, regulator,* etc., essentially mapping business entities to agent functions [31] [32]. While that patent was more about semantic search and ontology alignment, the idea of categorizing agent roles bears resemblance to the specialized agents in modern systems (e.g., an agent that "closes" a deal, similar to Neurons Lab's closing agent [12]). Recent open-source projects like Microsoft's AutoGen (2023) and academic frameworks like HALO (Hierarchical Autonomous Orchestration for LLMs, 2025) propose standard patterns for multi-agent coordination. These typically include a

**messaging protocol** between agents and a way to establish hierarchies of control. Any new patent in this domain would likely cite such frameworks as prior art for orchestration logic.

- **Model Context Protocol (MCP)** – Introduced by Anthropic in late 2024, MCP is an emerging open standard (not patented) that defines how AI agents (LLMs) communicate with external tools and each other in a structured way [33]. MCP provides a unified protocol layer analogous to HTTP but for AI context exchanges [34] [35]. It includes an **MCP client** (the agent or user-facing side), an **MCP server** (which exposes tool or data functionality in a standardized schema), and the protocol for interaction [33]. For example, Graphiti's integration with Cursor IDE uses an MCP server to let the IDE agent store and retrieve memories from the knowledge graph [25] [26]. MCP is notable because it could be foundational for secure and standardized knowledge sharing among agents – a patent filing might leverage MCP or a similar protocol to claim a novel combination of agents interacting with a knowledge graph. While MCP itself is open-source, a patent could specify particular **protocol extensions for multi-agent knowledge graphs**, such as authentication, access control tags on graph queries, or multi-agent dialogue management through MCP.

- **Hybrid Retrieval and Knowledge Augmentation** – Several recent publications deal with combining knowledge graphs and LLMs for improved retrieval (often termed *Graph-RAG* or similar). For instance, one method involves agents that first query a knowledge graph to identify relevant entities or subgraphs, and then use those as context to retrieve specific documents via vector search [16]. This two-step approach has shown better precision in domains like legal case analysis [16]. Patents such as EP4369259A1 (2023) have claimed methods for updating knowledge graphs in real-time to support automated reasoning without human intervention (suggesting an autonomous agent updating the graph). Any comprehensive patent on self-assembling agent knowledge systems would likely incorporate these hybrid retrieval techniques – ensuring that unstructured corporate text and structured graphs are both leveraged by the agents.

In summary, while no single existing patent covers "a multi-agent system building and using a real-time shared corporate knowledge graph with prompt generation and performance tracking," the pieces are falling into place via research and targeted patents. The inventive step for a new patent might lie in the *particular orchestration of these elements*: e.g., a specific architecture where agents form a dynamic hierarchy and use a protocol like MCP to read/write a continuously evolving enterprise knowledge graph, all under governance and tracking mechanisms. In the next section, we propose such a conceptual architecture in detail.

## Integration with Tools and Collaboration Interfaces

To deploy a multi-agent knowledge system in practice, integration with existing corporate tools and intuitive interfaces for human collaboration are critical. Several integration points and interface considerations include:

- **Agent Collaboration Interfaces:** Users will need a way to interact with multiple agents or see the outcome of their collaboration. This could be presented as a single conversational interface where the orchestrator agent represents the "team" of agents, summarizing their findings. Alternatively, a dashboard could visualize contributions from each agent (e.g., which agent handled which subtask). For instance, Neurons Lab demonstrated an *Experience Design* interface where the conversation flow between agents (like introduction, needs analysis, objection handling, closing) is visualized as a

graph [12] [36]. This gave designers control over the dialogue stages and style. In a corporate setting, an **agent collaboration UI** might show a timeline of tasks with labels of which agent executed them, allowing a human overseer to drill into each step if needed. Human supervisors can then give feedback directly in context – effectively coaching the agents – which is then fed back into the knowledge graph or agent memory for learning.

- **Integration with Developer Tools (Cursor, IDEs):** As mentioned in the developer use case, agent systems can be embedded into IDEs like Cursor. Using Cursor's *Composer* or similar features, the IDE's AI assistant can call upon internal agent services. The **MCP server** approach used by Graphiti is a template: by running a local or cloud MCP server that wraps the knowledge graph and any custom tools, the IDE's agent (MCP client) can issue structured commands to store knowledge or retrieve context [33] [37]. This integration pattern can extend to other tools: imagine a spreadsheet plugin where an agent can query the company knowledge graph for data enrichment, or an email client that auto-drafts responses using the collective knowledge of past communications (with an agent pulling relevant facts from the graph). The key is **low-friction integration** – using standardized APIs or protocols so that adding the agent network to an existing workflow does not require starting from scratch. Companies like Microsoft and IBM are weaving agent orchestration into their office suites (Copilot, Watson Orchestrate), suggesting that soon these multi-agent systems will act as an overlay across common enterprise apps.

- **APIs and Connectors:** A robust self-assembling agent system should include connectors to corporate data sources: databases, knowledge bases (Confluence, SharePoint), CRM systems, etc. These connectors can be implemented as specialized **ingestion agents** or pipelines that continuously feed data into the central knowledge graph. For example, a "CRM sync agent" could listen for new customer tickets and convert them into structured triples or nodes in the graph (linking customer, issue, product, etc.). Cloud providers (like AWS in the Neurons example [38]) have services such as Textract or Transcribe that can be seen as preprocessing agents to pull information from PDFs or voice calls. Integration might also involve tool-calling by agents: an agent might need to execute an external API call (e.g., schedule a meeting via Outlook API, or run an SQL query on a data warehouse) as part of fulfilling a user request. Frameworks like LangChain facilitate such tool calling, and the agent orchestration must handle passing authentication and capturing results securely. Each external integration point is an opportunity to enrich the knowledge graph (with results or with log data) so that future queries benefit from that knowledge.

- **Security and Access Control:** Integration with corporate systems requires careful enforcement of permissions. The knowledge graph itself can implement row-level or node-level security labels – for instance, marking certain nodes as confidential, and only agents with a certain role or clearance can access them. The integration layer should map corporate identity and access management (IAM) to agent permissions. If an agent acts on behalf of a user, it should inherit that user's access rights. If it's an autonomous background agent, it might have a service role with limited scope. All API calls and data access by agents need to be auditable. Using one centralized knowledge repository (the graph) actually aids security: instead of each agent scraping data arbitrarily, they request it through the graph's controlled interface, which can log and filter queries. Secure integration also implies sandboxing agents when using tools (to prevent an agent from, say, executing unauthorized commands if compromised by a prompt injection). Organizations may opt to keep the entire multi-agent system within a virtual private cloud or on-prem environment to protect sensitive knowledge.

In summary, integration must not come at the expense of **secure knowledge flow** – maintaining confidentiality and integrity of data is a first-class requirement for corporate AI.

- **Collaboration and Feedback:** Finally, a truly effective interface will allow **human-agent collaboration**. Users (be it employees or admins) should be able to correct an agent's output, provide feedback (positive or negative), or insert new knowledge when the agents are unaware. This feedback loop can be integrated via the UI – e.g., after an agent provides an answer, the user can flag it as incorrect, and the system will trigger a re-learning or update in the knowledge graph for future. Collaborative interfaces might also allow a user to **delegate tasks to agent teams** with a natural language command (the orchestrator then breaks it down among agents). Think of it as a project management interface where you assign a goal to your AI team and monitor their progress. Providing transparency (like showing which sources the agents consulted, via citations or expandable nodes) will build trust with users. If users see the underlying knowledge graph connections that led to an answer, they can verify and correct them. Thus, integration with collaboration tools is not just about the agents plugging into software, but also about the humans plugging into the agent's cognitive loop.

# Proposed Architecture and Patent Concept Outline

Building on the above insights, we propose a conceptual architecture for a **self-assembling AI agent system with dynamic knowledge graph orchestration**. This serves as a blueprint for a potential patent, covering the system components, protocols, agent hierarchies, and novel features:

## System Architecture Overview

At the core is a **Shared Knowledge Graph Repository** – a graph database that stores the enterprise knowledge as interconnected entities, relationships, and attributes. This knowledge graph is **continuously updated** by the agents: it ingests structured data (databases, spreadsheets), unstructured data (documents, emails, chats), and real-time data streams (transactions, sensor data) through various connectors. The graph might be accompanied by a **Vector Index** for embeddings of unstructured text, enabling hybrid search (the graph provides the schema and candidates, the vector store provides semantic matching) [10]. Surrounding this central knowledge base is a network of **AI Agents** of several types: - **Orchestrator Agent** – a top-level agent (or a small set of agents) responsible for global planning and task delegation [30]. It receives high-level goals or queries (from a human user or another system) and breaks them into subtasks. The orchestrator maintains a view of which specialist agent is best suited for each subtask, and it oversees the workflow execution [39] [40]. It also mediates shared context, ensuring all agents work with the latest relevant knowledge from the graph. - **Specialist Agents** – each focuses on a particular domain or function. For example, *FinanceAgent*, *HRAgent*, *DevOpsAgent*, *ResearchAgent*, etc. These agents have expertise encapsulated either in fine-tuned LLM prompts or model parameters, and possibly their own toolsets. They consult the knowledge graph for domain-specific data (e.g., FinanceAgent retrieving revenue numbers, or DevOpsAgent retrieving server logs) and perform reasoning or actions. They might update the graph with new insights (e.g., ResearchAgent adds a new market trend node). Each specialist agent can also use **tools/APIs** specific to its domain (like an HR system API) through a standardized interface. - **Memory/Reflection Agent** – a meta-agent that monitors dialogues and task outcomes, populating an **episodic memory** in the knowledge graph. This agent tracks performance metrics: for instance, after a task is completed, it logs which agent contributed which info, whether the result was accepted or needed correction, and how long it took. This forms a *performance knowledge*

*subgraph* that links agents, actions, outcomes, and feedback. The reflection agent can analyze this to suggest improvements (e.g., noticing an agent often fails on certain queries might trigger a model update or a new agent creation). This component ensures continuous learning and optimization, akin to a self-improvement loop in the network. - **User Interface Agent(s)** – agents dedicated to interacting with humans (could be a chatbot interface, email interface, voice assistant, etc.). They translate user requests into internal objectives and format the multi-agent system's outputs into human-friendly responses. They also enforce any necessary content filtering or approval steps (for compliance). For instance, a *ChatUIAgent* might take a user's question, pass it to the orchestrator, and then gather the final answer and present it with explanations and citations from the knowledge graph. This agent sits at the boundary of the system, ensuring that secure and appropriate information is communicated outward.

These agents are *not static*. The architecture supports **dynamic instantiation of new agents** or roles as needed. If a novel task arises frequently, the system might spawn a new specialist agent (possibly by cloning and fine-tuning an existing one or by prompt-programming a new role). This is part of what we mean by "self-assembling" – the agent society can **reconfigure itself** by adding or deactivating agents based on the evolving knowledge and goals.

## Communication Protocol and Knowledge Sharing

All interactions between agents and between an agent and the knowledge graph occur via defined **protocols**. The proposed system would implement or extend something like the **Model Context Protocol (MCP)** [33] to standardize communication: - Agents communicate with the knowledge graph through a query interface: e.g., an agent can send a query like *"MCP::Query(KG, find related nodes to X)"* and get a structured response. Similarly, updates are done via a transaction protocol: *"MCP::Update(KG, assert new relationship Y)"*. This decouples the agents from the specific database technology and provides a **unified language for knowledge access**. - Agents communicate with each other using a message-passing format (could be natural language messages, or a structured format for requests and results). For instance, if the orchestrator assigns a task, it might send a JSON or YAML message to the chosen agent containing: goal description, context pointers (references to relevant graph nodes), and tool permissions. The specialist agent, upon completion, sends back a message with results and any graph updates it made. This is logged in a **conversation log** per task. - The protocol would support **multi-turn dialogues** among agents. If Agent A needs clarification from Agent B, it doesn't have to route through the orchestrator – it can directly query B using the protocol, though the orchestrator would be aware of this interaction (similar to how services communicate in a microservice architecture). An important aspect is a **negotiation protocol**: agents might have to reconcile conflicting suggestions. For example, in a leadership advisory scenario, a RiskAgent might advise caution while a SalesAgent pushes for expansion. The system could allow a moderated debate between them, either resolved by the orchestrator or by a voting mechanism encoded in the protocol. - **Security and permissions** are embedded at the protocol level. Each agent has a digital identity with roles that the MCP server recognizes, limiting what queries or tool calls it can make [34]. For example, an HRAgent's MCP token might simply not allow accessing financial data in the graph. If an agent tries to bypass this (intentionally or via a prompt exploit), the request is denied by the MCP layer (just as an HTTP 403 Forbidden). All communications are encrypted and authenticated, ensuring that a malicious actor can't impersonate an agent or eavesdrop on sensitive knowledge exchanges.

By using a standardized protocol for all interactions, the system achieves **modularity and scalability**. New tools or data sources can be plugged in as MCP services (for instance, a "LoggingService" that agents can write event logs to, or an "EmailService" an agent can call to send a report). Likewise, new agents can

register with the orchestrator via the protocol, advertising their capabilities. This dynamic service discovery could even be automated: the orchestrator might detect a gap (say, no agent is handling compliance checking) and request a new agent module be loaded or created, then register it in the network.

## Agent Hierarchy and Organization

The architecture inherently supports a **hierarchical agent organization**, which is often necessary to maintain order in complex tasks [30]. At the top is the orchestrator (or a small group of high-level agents) that have broad awareness. Below that, specialist agents might be grouped by function (for example, a *"Sales Team"* of agents vs. a *"Engineering Team"* of agents). Each team could have a coordinator agent that reports up to the orchestrator, forming a multi-level hierarchy. This is analogous to how a company has divisions and team leads.

However, the hierarchy can be fluid. For certain straightforward tasks, the orchestrator might directly delegate to a single agent and not involve others. For more complex, multi-faceted problems, the orchestrator might assemble a *temporary team* of agents. In that sense, the hierarchy is *task-dependent*. The proposed system could patent an approach where **agent hierarchies form dynamically** based on the context: e.g., when a task "Develop new product strategy" comes in, the orchestrator creates a sub-hierarchy with a *StrategyAgent* leading, under which a *MarketResearchAgent*, *FinanceAgent*, and *TechLeadAgent* collaborate. Once the task is done, that sub-team dissolves (though the knowledge of their interaction persists in the graph for later reference).

Each agent in the hierarchy has well-defined **responsibilities and decision rights**. Higher-level agents focus on planning, conflict resolution, and big-picture alignment (ensuring the solution meets the user's goal). Lower-level agents focus on execution details in their domain. This layered approach prevents chaos and ensures **accountability**: if something goes wrong, the system can trace whether it was due to a specialist agent's error or the orchestrator's instruction. In a patented design, one could claim a particular method for hierarchical control, such as a "mentor-critic" pairing where one agent double-checks another's work (a quality control sub-hierarchy).

Crucially, the knowledge graph underpins the hierarchy by serving as a **common ground** where all agents, regardless of level, record their outputs and fetch needed context [41]. This means a lower-level agent doesn't only rely on direct messages from above; it can also see the larger context in the graph that the orchestrator has prepared (for instance, a subgraph of relevant info for its subtask). Likewise, when it finishes, it writes results to the graph so that the orchestrator and others immediately see it. This *blackboard architecture* style (common in early AI systems) is reborn here via a knowledge graph, enabling both hierarchical and opportunistic collaboration.

## Secure Knowledge Flow and Governance

In an enterprise scenario, **secure knowledge flow** is paramount. Our conceptual architecture includes multiple layers of security and governance: - **Role-Based Access Control (RBAC):** Each agent is associated with one or more roles. The knowledge graph's nodes and edges have labels or attributes defining access requirements (public, confidential, HR-only, legal-only, etc.). An agent can only read or write a graph element if its role permits. For example, a leadership advisor agent might access company-wide financial summaries but not individual employees' medical records; an HR agent vice versa. This could be implemented by partitioning the graph or by attribute-based access control (ABAC) at query time. The MCP

layer enforces these rules uniformly [35]. - **Data Encryption and Privacy:** Sensitive data in the graph may be stored in encrypted form. Agents have to explicitly request decryption for specific use, which is logged. Alternatively, homomorphic encryption or differential privacy techniques might be employed for certain analytics agents (so they can compute insights without directly seeing raw sensitive data). A patentable idea here could be an **"enclave agent"** that has higher privileges but runs in a secure enclave, performing analysis on confidential data and only outputting aggregated or sanitized results to the rest of the agents. - **Knowledge Lifecycle Management:** The system tracks the provenance of every piece of knowledge in the graph – which agent added it, from what source, when, and how it has been used. This facilitates governance and compliance checks. If a particular data point must be purged (e.g., a user requests deletion of personal data under GDPR), the system can identify it in the graph and trigger all agents to forget or refrain from using that node. Agents might periodically "forget" or archive old information that's no longer relevant, to reduce risk and ensure they rely on the freshest data. Policies can be set that certain knowledge expires after a time unless reaffirmed by an agent. - **Agent Auditing and Overrides:** A supervisory interface allows authorized personnel (or even a watchdog agent) to audit agent decisions. Every recommendation or action proposed by the system can be traced back to the supporting knowledge and the reasoning path (this is recorded in the graph as a chain of inferences or a subgraph linking premises to conclusion). If an agent produces an unacceptable output (e.g., it tried to suggest something against company policy), the system can have an override mechanism: the orchestrator or a governance agent can intercept the output before it reaches the user. This is akin to having an AI compliance officer in the loop. Additionally, the performance tracking subgraph can be reviewed to spot anomalies – for instance, if an agent suddenly starts accessing data outside its usual pattern, it could indicate it's malfunctioning or compromised, and the system can automatically quarantine or reset that agent.

By incorporating these governance mechanisms, the architecture ensures that knowledge flow among agents is not only efficient but also **safe and aligned with corporate rules**. This is a key differentiator for enterprise AI versus general internet AI – it must follow the laws (both legal and internal policies). A patent could claim specific techniques for "secure multi-agent knowledge sharing," like the use of context-based encryption keys for different subgraphs or automated policy injection into agent prompts (e.g., always prepend a compliance guideline to certain agents' instructions).

## Performance Tracking and Continuous Learning

Finally, an innovative aspect of this system is how it tracks performance and learns over time: - The knowledge graph itself can serve as a **memory of outcomes**. Each task or query handled by the agents results in an entry recording what was asked, which agents participated, what the result was, and how it was received (user feedback or measured success). Over time, this builds a dataset of the agent network's activity. Using this, a **Learning Agent** can perform meta-analysis to identify where bottlenecks or failures occur. For example, it might learn that whenever a legal question comes, the agents take a long time or give unsatisfactory answers – indicating a gap in the knowledge or a need for a new legal specialist agent. - The system can use these records for **reinforcement learning or fine-tuning**. If certain actions consistently lead to good outcomes (say, the SalesAgent closing a deal after following a particular negotiation script), that pattern can be reinforced – the agent could be updated to prioritize that strategy. If an outcome is bad (e.g., an agent's suggestion caused an error), the agents involved can be penalized and their decision logic adjusted. This could be implemented via reward signals that are stored in the graph and occasionally used to fine-tune the LLMs driving the agents (analogous to RLHF – Reinforcement Learning from Human Feedback – but here it's from multi-agent performance feedback). - **Benchmarking and KPIs:** The performance tracking also allows setting KPIs for the AI system itself – e.g., average task completion time,

accuracy rate of answers, user satisfaction scores, etc. The orchestrator agent can have goals to improve these metrics. For instance, it might A/B test two different breakdown strategies for a task to see which yields better results, then standardize on the better one. This kind of self-optimization could be claimed as a novel feature: *a multi-agent system that tracks its own performance metrics and dynamically reconfigures prompts, agent roles, or knowledge retrieval strategies to improve those metrics*. - **Human Feedback Loop:** Users and domain experts should be able to inject feedback easily – marking answers as correct/incorrect, providing the right answer if the agent was wrong, or suggesting additional factors the agents missed. This feedback is ingested by the reflection component and added to the graph (perhaps linked to the specific context where it applies). The next time a similar context arises, the agents will have that feedback available. One could implement a **coaching agent** that specifically looks at human feedback and creates updated prompts or training data for the other agents. This closes the loop between human oversight and agent learning.

Overall, the architecture emphasizes a **network that learns and evolves**. New knowledge leads to new graph content; new graph content can lead to new agents or updated agent behavior; and the cycle continues. The self-assembling nature refers not just to adding agents, but also to how the knowledge structure self-organizes as more information and connections emerge.

## Conceptual Patent Claims (Hypothetical)

To tie this together in the form of a patent concept, we can outline a few key claims that such a patent might include: 1. **Multi-Agent Knowledge Graph Orchestration:** An AI system where a plurality of autonomous agents collaboratively build and consume a shared knowledge graph in real-time, under the guidance of a coordinator agent, to fulfill user-specified objectives. The novelty here is the combination of autonomous graph updates with orchestrated division of labor among agents. 2. **Dynamic Agent Team Formation:** A method for dynamically instantiating and dissolving agent sub-teams based on the context of a query or task, using the knowledge graph content to determine which expert agents are needed. This includes spawning new temporary agents for specialized tasks and retiring them after use. 3. **Model Context Protocol for Agents:** The use of a standardized context exchange protocol (like MCP) enabling agents to invoke tools and query the knowledge graph in a secure, controlled manner, including an authorization scheme that restricts agent access to knowledge based on role metadata. 4. **Hierarchical Memory and Learning:** A hierarchical memory structure where short-term interaction data and long-term corporate knowledge are integrated. The claim could cover the storage of agent interactions and outcomes in a graph format and the use of that data to adjust future agent behavior (closing the feedback loop). 5. **Secure Knowledge Flow Control:** Mechanisms to ensure secure handling of information within the agent network – for example, tagging sensitive knowledge graph elements with clearance levels and enforcing those at query-time, as well as monitoring agent requests for anomalies (essentially an AI firewall). 6. **Corporate Knowledge Embedding Integration:** The combined use of a symbolic knowledge graph and vector embeddings such that agents can perform hybrid reasoning (symbolic multi-hop queries followed by semantic similarity searches) to retrieve comprehensive context for decision-making. The claim might be that by linking nodes in the graph with embedding vectors, agents can seamlessly move between logical inference and deep content retrieval. 7. **Performance Tracking Graph & Self-Optimization:** Logging each multi-agent transaction as a graph structure (nodes for actions, agents, results, etc.) and applying algorithmic analysis to this structure to recommend or implement optimizations in the agent workflow. Essentially, patenting the idea of an “agent performance knowledge graph” that the system mines to improve itself.

These claims would highlight the unique synergy of technologies in this architecture: **LLM-driven agents, knowledge graphs, dynamic orchestration, and secure, continuous learning** – all within an enterprise context. Even if certain elements are known (LLMs or knowledge graphs individually), the inventive step is in how they are arranged to support a living corporate knowledge network orchestrated by self-directed agents.

## Conclusion

Self-assembling AI agent systems with real-time knowledge graphs represent a powerful paradigm for corporate AI. By enabling multiple specialized agents to collaborate, share a collective memory, and adapt on the fly, organizations can leverage AI for far deeper insights and automation than any single static model could offer. Current implementations from industry (SAP's multi-agent copilot, Graphiti's knowledge engine, etc.) and research (Zep's memory graph, multi-agent AutoGen frameworks) lay the groundwork, demonstrating feasibility and value [6] [2]. Use cases in coaching, HR, development, and leadership show how such systems can transform work by embedding intelligence into every workflow.

Our proposed architecture sketches a path toward an integrated solution that addresses not only the technical challenges of dynamic knowledge and prompt generation, but also enterprise necessities like security, auditability, and scalability. The focus on **secure knowledge flow** ensures that as agents assemble and disseminate corporate knowledge, they do so in compliance with policies and privacy – a non-negotiable in corporate settings. Meanwhile, **agent-driven prompt generation and task orchestration** unlock higher autonomy, with agents essentially writing the playbook as they execute it. Performance tracking closes the loop, making the system *self-improving* over time.

In contemplating a patent, it is clear that this is a nascent field with much innovation ongoing. A successful patent would likely cover the *combination* of these elements – a novel orchestration of multi-agent systems with knowledge graphs and continuous learning in a corporate environment. By claiming the architecture and protocols that enable a hive mind of AI workers to safely share and build knowledge, such a patent could protect a foundational design for the next generation of enterprise AI platforms.

Ultimately, the convergence of **agents + knowledge graphs + LLMs** points to AI systems that are more **adaptive, context-aware, and collaborative** than ever before. Corporations that adopt these will have AI that isn't just a tool, but a constantly evolving partner that grows with the organization's knowledge. The research and ideas discussed here map out both the state of the art and the next steps to realizing that vision. With careful design, the future corporate AI "brain" will be a dynamic graph of knowledge, tended and expanded by an orchestra of self-directed agents – working in unison to drive corporate optimization and innovation securely and intelligently.

**Sources:** [6] [7] [2] [4] [10] [11] [33] [29] [16] [17]

---


[1] [30] [39] [40] [41] What is AI Agent Orchestration? | IBM
https://www.ibm.com/think/topics/ai-agent-orchestration

[2] [3] [5] GitHub - getzep/graphiti: Build Real-Time Knowledge Graphs for AI Agents
https://github.com/getzep/graphiti

4 18 27 28 [2501.13956] Zep: A Temporal Knowledge Graph Architecture for Agent Memory
https://ar5iv.labs.arxiv.org/html/2501.13956

6 7 8 9 SAP launches collaborative AI agents, adds Knowledge Graph | CIO
https://www.cio.com/article/3551017/sap-launches-collaborative-ai-agents-adds-knowledge-graph.html

10 11 12 13 14 36 38 Using Multi-Agentic Systems and Knowledge Graphs for a Better Customer Experience - Neurons Lab
https://neurons-lab.com/article/using-multi-agentic-systems-and-knowledge-graphs-for-better-customer-experience/

15 Hierarchical Agent Teams - GitHub Pages
https://langchain-ai.github.io/langgraph/tutorials/multi_agent/hierarchical_agent_teams/

16 Multi-Agent Hybrid Knowledge Base Retrieval: Building a High-Precision Legal Case Analysis Platform - DEV Community
https://dev.to/jamesli/multi-agent-hybrid-knowledge-base-retrieval-building-a-high-precision-legal-case-analysis-platform-47p3

17 Multi-agent Conversation Framework | AutoGen 0.2
https://microsoft.github.io/autogen/0.2/docs/Use-Cases/agent_chat/

19 20 First-of-its-kind AI Coach improves human teamwork | Computer Science | Rice University
https://csweb.rice.edu/news/first-its-kind-ai-coach-improves-human-teamwork

21 22 23 AI Agents in Human Resources | IBM
https://www.ibm.com/think/topics/ai-agents-in-human-resources

24 25 26 33 37 Cursor IDE: Adding Memory With Graphiti MCP ⚡
https://blog.getzep.com/cursor-adding-memory-with-graphiti-mcp/

29 Torch.AI - Latest News, Articles & Stories - BigDATAwire
https://www.datanami.com/vendor/torch-ai/

31 32 US20120158633A1 - Knowledge graph based search system - Google Patents
https://patents.google.com/patent/US20120158633A1/en

34 35 Cognizant - Enterprise Agentic AI Guide
https://www.cognizant.com/en_us/industries/documents/the-enterprise-guide-to-agentic-ai.pdf