Expert-Level Architectural Blueprint: Integrating Google A2A as the Multi-Agent Routing Backbone

I. Strategic Mandate: Interoperability as an Enterprise Necessity

The evolution toward Large Language Model (LLM)-powered autonomous systems has reached a critical juncture, defined by the need for seamless interaction between specialized components. The integration of the Google Agent-to-Agent (A2A) protocol is not merely an addition but a foundational architectural imperative designed to standardize this interaction.

1.1 The Fragmentation Challenge in Multi-Agent Systems (MAS)

The current state of the AI ecosystem is characterized by significant fragmentation, stemming from the deployment of specialized AI agents built using disparate frameworks, often by different vendors. Frameworks such as CrewAI and LangGraph automate intricate multi-agent workflows effectively but rely on bespoke communication layers and internal message passing mechanisms. This reliance on non-standard, internal communication creates architectural silos, fundamentally hindering the ability of agents to communicate and collaborate across technical boundaries.

This systemic lack of protocol-level standards directly prevents cross-framework composition, forcing enterprises to manage fragmented ecosystems where specialized capabilities cannot be easily combined. From a strategic perspective, adopting an open standard like A2A mitigates the significant risk associated with vendor lock-in. Since A2A is an open protocol, initially introduced by Google and subsequently donated to the Linux Foundation, it ensures that the core communication layer remains resilient against changes or deprecations within any single proprietary framework. This architectural flexibility ensures that current investments in agent development remain viable as the underlying technology stack evolves.

1.2 A2A as the Universal Standard for Agent Collaboration

The Agent-to-Agent (A2A) protocol was designed explicitly to serve as a common language—or, conceptually, a "universal translator"—that enables collaboration between AI agents irrespective of their developer, vendor, or underlying framework. A2A standardizes the interactions necessary for generative AI agents to function and collaborate effectively as autonomous peers, moving beyond the limitation of treating them merely as stateless tools.

For large-scale enterprise deployments, A2A provides a standardized method for managing, observing, and coordinating agents across diverse operational platforms and cloud environments. A cornerstone of the A2A design is its principle of preserving opacity. The protocol allows agents to engage in collaborative, multi-agent scenarios without requiring them to share their internal state, memory, tools, or proprietary context. This feature is vital for

enhancing security and protecting intellectual property when integrating agents from multiple internal business units or external partners. The broad industry support, involving a large coalition of technology partners, further solidifies A2A's position as the foundational standard necessary for a robust, interconnected AI ecosystem.

1.3 Defining the Supervisor Agent's Role in an A2A Ecosystem

In a supervisor-led multi-agent architecture leveraging A2A, the supervisory agent's role is precisely defined as the **Client Agent** within the A2A client-server model. The fundamental shift in the supervisor's function is from direct task execution to high-level orchestration, dynamic routing, and workflow management.
The supervisor's core responsibilities encompass a rigorous process chain. First, it involves **Task Formulation**, where a high-level user request is interpreted and decomposed into discrete, manageable sub-tasks. Second, the supervisor executes **Capability Discovery**, utilizing the standardized A2A mechanisms to dynamically identify and select the most appropriate specialized sub-agent, designated as the Remote Agent, capable of handling each specific sub-task. Third, it handles **Delegation and Monitoring**, which includes formulating the structured A2A Task message and actively monitoring the progress of delegated tasks via synchronous, streaming, or asynchronous channels. Finally, the supervisor performs **Aggregation**, synthesizing the results (Artifacts) received from the various Remote Agents into a final, cohesive, and actionable response for the initiating entity.

II. A2A Protocol Deep Dive: Architecture and Communication

The effectiveness of A2A as a routing backbone depends entirely on its rigorous architectural definition, which facilitates predictable, reliable, and standardized communication regardless of the agents' internal complexities.

2.1 The Client-Server Model and Core Components

A2A employs a foundational client-server structure tailored for autonomous communication. The two primary agent types are distinguished by their responsibilities:

- **A2A Client (Supervisor Agent):** The initiator of the interaction. This entity is responsible for formulating the tasks, selecting the appropriate remote agent, communicating the request, and ultimately consuming the results.
- **A2A Server (Remote Agent):** The executor. This entity hosts the A2A endpoint, receives the client's request, processes the delegated task using its internal logic and tools, and responds with status updates or the final results.

Communication is structured through several critical components:

- **Agent Card:** This is a mandatory, machine-readable JSON file that serves as an agent's public advertisement. It explicitly outlines the agent's capabilities, supported interaction modalities (text, forms, media), the security scheme required for connection, and the specific endpoint URL for communication.
- **Task:** This component represents the formal unit of work, clearly defined and delegated by the Client Agent. It ensures that the required execution parameters and scope of work are unambiguous.

- **Message:** Agents exchange information using structured, JSON-based messages. This standardization is critical, as it encapsulates the actions, intents, results, and necessary metadata, ensuring the interaction is predictable and easily processed by heterogeneous systems.
- **Artifact/Part:** These components represent the actual data payload exchanged. An Artifact is typically the final generated output, while a Part can represent distinct subtasks or chunks of information that are processed independently, aiding in task breakdown and parallel execution.

Table: A2A Component Roles in Task Routing

Component	Responsible Agent	Primary Function in Routing	Data Format/Protocol
Agent Card	Remote Agent (Server)	Capability advertisement, skill definition, and connection endpoint exposure	Structured JSON
Client Agent	Supervisor Agent	Intent mapping, discovery, task decomposition, and selection of appropriate Remote Agents	Framework-dependent Logic
Task Message	Client Agent	Formalized request package containing intent, input, and required actions	Structured JSON-RPC over HTTP(S)
Asynchronous Update	Remote Agent (Server)	Delivery of interim status or long-running task results	Webhook (Push) or SSE (Streaming)

2.2 Standardized Communication Mechanisms

To ensure enterprise-level reliability and minimize integration complexity, the A2A protocol leverages established web standards. The communication backbone operates using **JSON-RPC 2.0 over HTTP(S)**. This established transport layer provides a reliable and defined structure for requests and responses, which is fundamental for ensuring reliable data transmission and ease of parsing across systems built with different language SDKs (e.g., Python, Java, Go).

A distinguishing architectural feature of A2A is its robust mechanism for handling tasks of varying duration, a common challenge in multi-agent workflows where tasks may involve external API calls or human review. The protocol supports three modalities for result delivery:

1. **Synchronous Request/Response:** Used for immediate, low-latency tasks where the client waits for the result.
2. **Real-Time Streaming:** Achieved via **Server-Sent Events (SSE)**, which is utilized for continuous status updates, providing users or other agents with real-time feedback, or delivering very large outputs in chunks.
3. **Asynchronous Updates:** Critical for long-running tasks that may span hours or days. The protocol utilizes secure, client-supplied **webhooks** (push notifications) to deliver results or status changes. This mechanism is essential for operational resilience, ensuring

that the supervisor agent can delegate tasks without dedicating thread resources to continuous polling or maintaining a persistent connection.

The necessity of supporting asynchronous communication via webhooks dictates a crucial architectural requirement for the Supervisor Agent: it must be designed around an event-driven model. A supervisory system responsible for orchestrating dozens of concurrent sub-tasks cannot afford to block its processing threads while waiting for high-latency external dependencies, such as an agent waiting for a human approver or an external network query. Implementing push notifications maximizes concurrency and significantly improves the overall system throughput and responsiveness, representing a critical design departure from linear, synchronous tool-calling chains.

III. Dynamic Routing and Capability Discovery

The Supervisor Agent's primary function—routing—is executed not through predetermined logic but through a dynamic discovery process that treats the ecosystem as a variable resource pool.

3.1 The Centrality of the Agent Card for Routing Decisions

A2A routing is a capability-based process, relying on the machine-readable metadata provided by the Agent Card. Instead of hardcoding API keys or fixed endpoints, the Supervisor Agent dynamically selects the destination based on advertised competencies. Every A2A Server (Remote Agent) is required to expose its Agent Card via a standardized location, such as a well-known discovery endpoint (e.g., GET /.well-known/agent-card.json?assistant_id={assistant_id}).

The Agent Card must clearly articulate the agent's specific **Skills**, which are analogous to formalized function definitions or API specifications. These definitions allow the Client Agent to understand precisely what a Remote Agent can perform and what input schema it requires, enabling the system to identify the optimal match for a given intent.

3.2 Routing Logic Implementation: From Intent to Task Formulation

The Supervisor Agent (Client) executes a sophisticated decision cycle involving multiple steps to determine the correct routing path:

1. **Intent Translation:** The Supervisor's LLM processes the initial natural language request and converts it into a structured, formalized internal intent (e.g., mapping "check for fraud risks on this claim" to a structured intent like execute_fraud_analysis).
2. **Discovery Query:** The Supervisor queries its operational registry, referencing the capabilities defined in all accessible Agent Cards.
3. **Capability Matching and Delegation:** The Supervisor’s core routing logic is fundamentally dependent on the function-calling or tool-use capabilities of its underlying LLM. The LLM must semantically match the formalized intent against the detailed Skills schemas advertised in the Agent Cards. This capability-based matching ensures that routing is flexible and determined organically by the LLM’s interpretation of the best fit, rather than rigid, rule-based logic.

The reliability of the entire routing framework therefore depends on the robustness of the Supervisor Agent's language model. Any failure in the LLM's reasoning engine to consistently perform accurate semantic matching between an abstract user goal and the concrete,

structured schemas of the Agent Cards will lead directly to catastrophic routing failures. Furthermore, A2A infrastructure can be extended to support advanced routing beyond mere capability matching. For example, organizations like Twilio have demonstrated how **Latency-Aware Agent Selection** can be integrated. In this scenario, Remote Agents are configured to broadcast operational metrics such, as current processing latency or system load. This telemetry allows the Supervisor Agent to intelligently route tasks based not only on *what* an agent can do but also on *how quickly* it can do it. This extension transforms the A2A routing mechanism into a dynamic resource allocation and load-balancing layer, maximizing overall system efficiency and ensuring tasks are sent to the most responsive peer available.

IV. Enforcing Trust: Security and Authorization Architecture

For the A2A protocol to be viable in demanding enterprise environments, the security foundation must be absolute. A2A is architected to be **Secure by Default**, leveraging established standards to protect sensitive data and transactions throughout the multi-agent workflow.

4.1 A2A's Secure-by-Default Principles

The protocol's security posture is built upon widely adopted, reliable web technologies, minimizing proprietary risk and integration difficulty. Key principles include:

- **Standards-Compliant Security:** The protocol utilizes HTTPS for secure transport and integrates **OAuth 2.0** and **JSON Web Tokens (JWTs)** for identity and access management.
- **Agentic-First Operation:** Security is enforced based on the principle that agents operate independently, without shared memory or context by default.
- **Opacity and IP Protection:** The design ensures that agents can collaborate without exposing internal mechanisms or proprietary logic, which fundamentally protects intellectual property during multi-party transactions.

4.2 Authentication and Integrity Verification

A multi-layered approach to verifying identity and protecting message content is mandated by the A2A specification:

- **Mutual Authentication:** Agents must establish trust and identity via standard **OAuth 2.0** flows, providing secure authorization for access delegation.
- **Message Integrity:** To ensure that a message has not been tampered with and truly originated from the claimed source, communication employs **Digital Signatures**, often generated using RSA key pairs. These signatures verify the authenticity and integrity of the message payload, providing non-repudiation in the critical agent-to-agent exchanges.

4.3 Implementing Role-Based Access Control (RBAC) via JWT Claims

Authorization in A2A is enforced rigorously through **Role-Based Access Control (RBAC)**, which is implemented using claims embedded within JWTs.

- **Agent-Scoped Claims:** The issued JWTs must carry claims (metadata) that define the agent's identity, roles, and access scope for the duration of the transaction. These claims

are essential for enforcing the **Principle of Least Privilege**.

- **Fine-Grained Permissions:** A2A supports mapping fine-grained permissions directly onto the structural elements of the Task and Message schemas. This design ensures that an agent holding a JWT with a limited claim scope is physically prevented from executing Tasks that exceed its defined privileges (e.g., a 'read-only' agent cannot execute a write_database action).

A critical architectural consideration is ensuring **transitive security** throughout the workflow. The security boundaries established by the A2A protocol must not terminate at the Remote Agent's endpoint but must extend into any downstream enterprise data services accessed during task execution. If a Remote Agent needs to query a sensitive knowledge graph (such as Neo4j), the agent-scoped claims from the A2A JWT must be recognized by the database's security layer, typically via OpenID Connect (OIDC) or SSO integration. The JWT claims, defining the agent's specific roles, must map correctly to the RBAC schema enforced by the data store. If this mapping fails, the request, although successfully authenticated by the A2A layer, will be rejected by the data service, leading to task failure. The Supervisor must therefore be designed to issue transaction-specific JWTs with carefully defined claims (issuer iss, audience aud) that enable the principle of least privilege for the sub-agent's entire execution path.

Table: A2A Security Claims Mapping for RBAC

Security Element	A2A Protocol Implementation	Role in Authorization
Authentication Standard	OAuth 2.0 and JWTs	Secure agent identity, authorization, and access delegation tokens
Integrity/Authenticity	Digital Signatures (RSA Key Pairs)	Verifies message content non-tampering and confirms sender identity
Authorization Token	JSON Web Token (JWT)	Carries agent-scoped and user-scoped claims (roles, groups, permissions)
Access Control Mechanism	Role-Based Access Control (RBAC)	Enforces least privilege based on decoded JWT claims; permissions mapped to Task schemas
Transitive Security Check	IdP/OIDC Integration	Maps JWT claims (e.g., groups/roles) to permissions in downstream systems (e.g., Neo4j SSO)

V. Multi-Framework Integration and Coexistence

The strategic value of A2A lies in its capacity to serve as the unifying layer for agents built using heterogeneous architectures, resolving the isolation endemic to proprietary frameworks.

5.1 A2A as the Interop Layer for Diverse Agent Architectures

A2A functions as a universal communication bus that enables seamless interoperation between agents built on distinct technologies, including Google's Agent Development Kit (ADK), LangChain, LangGraph, and CrewAI. This is evident in practical demonstrations involving

agents from different frameworks, such as a CrewAI-backed agent successfully communicating and collaborating with a LangGraph-backed agent. A2A acts as a messaging tier that allows these agents to "talk" to each other effectively despite their differing internal architectures.

5.2 Pattern 1: Exposing Existing Framework Agents as A2A Servers

The first deployment pattern involves integrating specialized agents into the A2A ecosystem by making them discoverable and callable. This requires the developer to take an existing agent built using a specific framework (e.g., CrewAI) and implement an A2A Server SDK wrapper. This wrapper is responsible for two key actions:

1. Exposing the agent's capabilities publicly via the standardized Agent Card.
2. Implementing the A2A HTTP endpoint to receive and correctly parse incoming JSON-RPC Task messages.

For instance, the LangGraph Platform offers native support for this pattern, allowing agents deployed with a message-based state structure to automatically expose an A2A endpoint at a predefined URL, complete with an automatically generated Agent Card detailing capabilities.

5.3 Pattern 2: Incorporating Remote A2A Agents as Tools within a Supervisor’s Workflow

The second, equally critical pattern defines how the Supervisor Agent integrates the remote capabilities into its internal decision logic. The Supervisor uses an A2A Client SDK to query the Agent Directory, execute capability discovery, and ultimately delegate tasks to external A2A Servers.

Within the Supervisor's internal orchestration framework (e.g., ADK or LangGraph), the remote A2A Agent is treated as a highly sophisticated external function or tool. The Supervisor’s LLM uses the information retrieved from the Agent Card to dynamically construct the necessary inputs and invocation instructions, facilitating task delegation. This ensures that the supervisor can leverage the specialized skills of opaque external agents without needing to manage their internal context or code.

5.4 The Complementary Role of Model Context Protocol (MCP)

When designing the overall agent architecture, it is essential to delineate the roles of A2A and its complementary standard, the Model Context Protocol (MCP). These protocols address different concerns in the AI architectural stack:

- **A2A Focus:** Defines **Agent-to-Agent** communication. It manages delegation, complex collaboration, message security, and high-level interaction between autonomous entities.
- **MCP Focus:** Defines **Agent-to-Tool/Resource** communication. It standardizes how a single agent connects to its *internal* tools, APIs, and data services (such as retrieving graph data from Neo4j) to acquire real-time context needed for execution.

This layering provides necessary architectural clarity. The Supervisor Agent utilizes **A2A** to delegate complex tasks to specialized peer agents (e.g., delegating a market_research task to a Research Agent). The Research Agent, in turn, uses **MCP** to standardize its connection to internal execution resources, such as invoking a Cypher query tool to interact with a knowledge graph for data retrieval. This structural separation ensures that A2A focuses on external coordination, while MCP optimizes internal execution context acquisition, preventing a conflation

of peer-to-peer delegation logic with tool-calling mechanics.

VI. Operationalizing the A2A Framework for Scale

Achieving production readiness for an A2A framework requires meticulous attention to the underlying infrastructure, particularly the serving layer for the LLMs that power both the Supervisor and the Remote Agents.

6.1 Deployment Choices for High-Throughput Agent Systems

The Supervisor Agent's capability for reasoning, intent parsing, and routing is fundamentally dependent on the performance of its backing LLM. In an enterprise setting, where thousands of tasks may be routed concurrently, high-throughput LLM inference is critical. The choice of LLM serving framework directly impacts latency, cost-efficiency, and overall system scalability.

6.2 Comparative Analysis of LLM Serving Frameworks: vLLM vs. Ollama

The decision between prominent serving frameworks, vLLM and Ollama, should be dictated by the specific role of the agent and the deployment stage:

- **Ollama (Local/Prototyping):** Ollama prioritizes ease of deployment and local model execution, making it highly suitable for development, prototyping, and personal AI experimentation. It excels at abstracting complexity, allowing developers to rapidly test A2A flows with a simple setup. However, its architecture is not designed for high-concurrency workloads or enterprise-grade scalability.
- **vLLM (Production/Scale):** vLLM is specifically optimized for high-performance, large-scale LLM inference. It achieves industry-leading throughput and low latency through advanced techniques such as PagedAttention and optimized GPU scheduling. This framework is mandatory for production environments, particularly for high-traffic core agents like the Supervisor or critical specialized Remote Agents, where resource utilization and cost-efficiency under multi-user workloads are paramount.

The prescribed solution for enterprise adoption is a **Hybrid Deployment Model**. Development teams should utilize Ollama for rapid iteration and testing of A2A integration logic. However, the core production agents, including the Supervisor and any specialized Remote Agents handling high-volume A2A requests, must be deployed using vLLM to leverage its superior performance and scaling capabilities on data center-grade hardware.

Table: LLM Serving Framework Comparison for Production Agents

Feature	**Ollama**	**vLLM**	**Optimal Use Case in A2A System**
Primary Focus	Local development, ease of use, simple deployment	High-performance inference, throughput, resource efficiency	
Scalability	Limited; tailored for single-model, low-concurrency use	Enterprise-grade, highly scalable with optimized GPU scheduling	Development and isolated smaller agents
Architecture	Lightweight runtime,	PagedAttention,	Core production agents

Feature	Ollama	vLLM	Optimal Use Case in A2A System
	simple packaging	advanced GPU memory management	handling high-volume routing and task execution
Hardware Fit	Laptops, workstations, consumer GPUs	Data center GPUs (A100s, H100s)	Rapid prototyping and local simulation
Throughput/Latency	Sufficient for low-volume use	Industry-leading, designed for multi-user, high-concurrency API serving	Production deployment where cost efficiency per inference is critical

6.3 Observability, Logging, and Evaluation in A2A Workflows

The fundamental architectural feature of opacity, while beneficial for security and IP protection, simultaneously presents a significant challenge for debugging and auditing. A robust observability framework is non-negotiable for operating an A2A-based MAS in production. Standardized tracing is the foundational element required to mitigate opacity risk. The A2A integration must incorporate specifications like **OpenTelemetry** to provide end-to-end tracing across the entire multi-agent workflow. This allows developers to reconstruct the full sequence of A2A delegation calls, crucial for diagnosing distributed system failures. Without tracing, reconstructing the path of a Task delegated from the Supervisor, passed to Agent A, then routed to Agent B, becomes nearly impossible across disparate infrastructure.
Furthermore, comprehensive **Cloud Logging** and **Cloud Monitoring** are necessary to track low-level details, including message schema compliance, inter-agent latency, and explicit failure points between the Client and Remote Agents. The observability system must also track metrics specific to the supervisor's core competence: routing intelligence. Beyond standard system metrics, operational analysis must include evaluating the Supervisor’s choice of agent. The system should log the Supervisor's intermediate reasoning steps against the available Agent Cards, allowing architects to benchmark if the LLM successfully selected the optimal agent for the task. This ensures that resources are not wasted on routing errors (e.g., selecting a less capable or slower agent) and that the system’s performance bottlenecks can be accurately attributed to either a routing logic failure or an execution failure in a specific sub-agent.

VII. Conclusion and Implementation Roadmap

7.1 Synthesis of Architectural Advantages

The integration of the Google Agent-to-Agent (A2A) protocol provides the indispensable standardized communication framework necessary for scaling enterprise-grade multi-agent systems. By decoupling communication from proprietary frameworks, A2A ensures true interoperability among specialized agents, regardless of their underlying technology stack. The resulting architecture is characterized by:

1. **Interoperability:** Enables seamless communication between agents built on diverse frameworks (e.g., LangGraph, CrewAI, ADK).
2. **Security:** Implements robust, standards-compliant security through OAuth 2.0, JWT-based RBAC, and message signing, enforcing least privilege throughout the distributed workflow, including transitive security to downstream data services.

3. **Scalability:** Supports asynchronous communication (webhooks/SSE) for resilient orchestration and dictates the use of high-performance LLM serving layers (vLLM) for production traffic.
4. **Resilience:** Facilitates dynamic, capability-based routing, with the potential for incorporating real-time operational data like latency into routing decisions, maximizing overall system performance.

7.2 Key Implementation Roadmap Priorities

Successful deployment of the A2A routing backbone requires a methodical, layered approach focused on security, standardization, and performance:

1. **Protocol Foundation and Implementation:** Establish the foundational A2A Client endpoints for the Supervisor Agent and A2A Server endpoints for all specialized Remote Agents utilizing the appropriate SDKs (e.g., Python, ADK).
2. **Capability Standardization:** Define and rigorously enforce the structured JSON schema for Agent Cards across all specialized sub-agents, ensuring clear, unambiguous advertisement of Skills for dynamic routing.
3. **Security Architecture Deployment:** Implement the OAuth 2.0 and JWT authentication mechanism. Critically, design the Role-Based Access Control (RBAC) claims structure to ensure a functional, transitive security mapping between the A2A protocol layer and the access controls of core enterprise data repositories, such as Neo4j graph databases.
4. **LLM Serving Optimization:** Implement the hybrid deployment model, using Ollama for rapid development cycles and transitioning all production-critical agents (Supervisor and high-traffic Remote Agents) to the vLLM serving framework to achieve required performance and cost efficiency at scale.
5. **Observability Layer Integration:** Deploy a unified observability stack leveraging OpenTelemetry tracing and structured logging across the entire multi-agent system to ensure end-to-end auditability and rapid diagnosis of distributed failures, specifically tracking routing logic decisions against execution outcomes.

Works cited

1. A2A Protocol, https://a2a-protocol.org/ 2. Getting Started with Agent2Agent (A2A) Protocol: A Purchasing Concierge and Remote Seller Agent Interactions on Cloud Run and Agent Engine | Google Codelabs, https://codelabs.developers.google.com/intro-a2a-purchasing-concierge 3. Inside Google's Agent2Agent (A2A) Protocol: Teaching AI Agents to Talk to Each Other, https://towardsdatascience.com/inside-googles-agent2agent-a2a-protocol-teaching-ai-agents-to-talk-to-each-other/ 4. A Survey of Agent Interoperability Protocols: Model Context Protocol (MCP), Agent Communication Protocol (ACP), Agent-to-Agent Protocol (A2A), and Agent Network Protocol (ANP) - arXiv, https://arxiv.org/html/2505.02279v1 5. What is A2A protocol (Agent2Agent)? - IBM, https://www.ibm.com/think/topics/agent2agent-protocol 6. a2aproject/A2A: An open protocol enabling communication and interoperability between opaque agentic applications. - GitHub, https://github.com/a2aproject/A2A 7. Announcing the Agent2Agent Protocol (A2A) - Google for Developers Blog, https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/ 8. Building A Secure Agentic AI Application Leveraging Google's A2A Protocol - arXiv, https://arxiv.org/html/2504.16902v1 9. Agent2Agent (A2A) Protocol Explained: Improving Multi-Agent Interactions - AltexSoft, https://www.altexsoft.com/blog/a2a-protocol-explained/ 10.

What is A2A Protocol? - CopilotKit | The Agentic Framework for In-App AI Copilots, https://www.copilotkit.ai/blog/how-to-make-agents-talk-to-each-other-and-your-app-using-a2a-ag-ui 11. Proposal for Improving Google A2A Protocol: Safeguarding Sensitive Data in Multi-Agent Systems - arXiv, https://arxiv.org/html/2505.12490v1 12. A2A endpoint in LangGraph Server - Docs by LangChain, https://docs.langchain.com/langgraph-platform/server-a2a 13. Introducing Agent2Agent (A2A): Understanding Google's Protocol for AI Collaboration, https://priyalwalpita.medium.com/introducing-agent2agent-a2a-understanding-googles-protocol-for-ai-collaboration-10a46155c458 14. Multi-Agent Systems in ADK - Google, https://google.github.io/adk-docs/agents/multi-agents/ 15. Agent2Agent protocol (A2A) is getting an upgrade | Google Cloud Blog, https://cloud.google.com/blog/products/ai-machine-learning/agent2agent-protocol-is-getting-an-upgrade 16. A2A authentication - Axway Documentation Portal, https://docs.axway.com/bundle/FlowManager_20_allOS_en_HTML5/page/a2a_authentication.html 17. Attribute a role to a user for RBAC authorization when using OAuth? - Stack Overflow, https://stackoverflow.com/questions/79507020/attribute-a-role-to-a-user-for-rbac-authorization-when-using-oauth 18. Configuring Neo4j Single Sign-On (SSO) - Operations Manual, https://neo4j.com/docs/operations-manual/current/tutorial/tutorial-sso-configuration/ 19. Understanding Group to Role Mapping for SSO Authentication in Neo4j Aura - Support, https://support.neo4j.com/s/article/Understanding-Group-to-Role-Mapping 20. Authentication and authorization - Operations Manual - Neo4j, https://neo4j.com/docs/operations-manual/current/kubernetes/authentication-authorization/ 21. Role-based access control - Operations Manual - Neo4j, https://neo4j.com/docs/operations-manual/current/authentication-authorization/manage-privileges/ 22. Vertex AI Agent Builder | Google Cloud, https://cloud.google.com/products/agent-builder 23. Python A2A, MCP, and LangChain: Engineering the Next Generation of Modular GenAI Systems | by Manoj Desai | Medium, https://medium.com/@the_manoj_desai/python-a2a-mcp-and-langchain-engineering-the-next-generation-of-modular-genai-systems-326a3e94efae 24. Neo4j Live: MCP for LLM Agents, APIs & Graphs - YouTube, https://www.youtube.com/watch?v=6igWn_dckpc 25. Mastering LLM AI Agents: Building and Using AI Agents in Python with Real-World Use Cases | by Jagadeesan Ganesh | Medium, https://medium.com/@jagadeesan.ganesh/mastering-llm-ai-agents-building-and-using-ai-agents-in-python-with-real-world-use-cases-c578eb640e35 26. vLLM vs Ollama: Key Differences for LLM Success - Openxcell, https://www.openxcell.com/blog/vllm-vs-ollama/ 27. vLLM vs Ollama: Key differences, performance, and how to run them | Blog - Northflank, https://northflank.com/blog/vllm-vs-ollama-and-how-to-run-them 28. Ollama or vLLM? How to choose the right LLM serving tool for your use case, https://developers.redhat.com/articles/2025/07/08/ollama-or-vllm-how-choose-right-llm-serving-tool-your-use-case 29. Ollama vs vLLM: A Detailed Comparison of LLM Frameworks - DEV Community, https://dev.to/mechcloud_academy/ollama-vs-vllm-a-detailed-comparison-of-llm-frameworks-513m 30. Ollama vs. vLLM: The Definitive Guide to Local LLM Frameworks in 2025, https://blog.alphabravo.io/ollama-vs-vllm-the-definitive-guide-to-local-llm-frameworks-in-2025/ 31. Vertex AI Agent Engine overview - Google Cloud, https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/overview.
1. A2A Protocol, https://a2a-protocol.org/ 2. Getting Started with Agent2Agent (A2A) Protocol: A Purchasing Concierge and Remote Seller Agent Interactions on Cloud Run and Agent Engine | Google Codelabs, https://codelabs.developers.google.com/intro-a2a-purchasing-concierge 3.

Inside Google's Agent2Agent (A2A) Protocol: Teaching AI Agents to Talk to Each Other, https://towardsdatascience.com/inside-googles-agent2agent-a2a-protocol-teaching-ai-agents-to-talk-to-each-other/ 4. A Survey of Agent Interoperability Protocols: Model Context Protocol (MCP), Agent Communication Protocol (ACP), Agent-to-Agent Protocol (A2A), and Agent Network Protocol (ANP) - arXiv, https://arxiv.org/html/2505.02279v1 5. What is A2A protocol (Agent2Agent)? - IBM, https://www.ibm.com/think/topics/agent2agent-protocol 6. a2aproject/A2A: An open protocol enabling communication and interoperability between opaque agentic applications. - GitHub, https://github.com/a2aproject/A2A 7. Announcing the Agent2Agent Protocol (A2A) - Google for Developers Blog, https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/ 8. Building A Secure Agentic AI Application Leveraging Google's A2A Protocol - arXiv, https://arxiv.org/html/2504.16902v1 9. Agent2Agent (A2A) Protocol Explained: Improving Multi-Agent Interactions - AltexSoft, https://www.altexsoft.com/blog/a2a-protocol-explained/ 10. What is A2A Protocol? - CopilotKit | The Agentic Framework for In-App AI Copilots, https://www.copilotkit.ai/blog/how-to-make-agents-talk-to-each-other-and-your-app-using-a2a-ag-ui 11. Proposal for Improving Google A2A Protocol: Safeguarding Sensitive Data in Multi-Agent Systems - arXiv, https://arxiv.org/html/2505.12490v1 12. A2A endpoint in LangGraph Server - Docs by LangChain, https://docs.langchain.com/langgraph-platform/server-a2a 13. Introducing Agent2Agent (A2A): Understanding Google's Protocol for AI Collaboration, https://priyalwalpita.medium.com/introducing-agent2agent-a2a-understanding-googles-protocol-for-ai-collaboration-10a46155c458 14. Multi-Agent Systems in ADK - Google, https://google.github.io/adk-docs/agents/multi-agents/ 15. Agent2Agent protocol (A2A) is getting an upgrade | Google Cloud Blog, https://cloud.google.com/blog/products/ai-machine-learning/agent2agent-protocol-is-getting-an-upgrade 16. A2A authentication - Axway Documentation Portal, https://docs.axway.com/bundle/FlowManager_20_allOS_en_HTML5/page/a2a_authentication.html 17. Attribute a role to a user for RBAC authorization when using OAuth? - Stack Overflow, https://stackoverflow.com/questions/79507020/attribute-a-role-to-a-user-for-rbac-authorization-when-using-oauth 18. Configuring Neo4j Single Sign-On (SSO) - Operations Manual, https://neo4j.com/docs/operations-manual/current/tutorial/tutorial-sso-configuration/ 19. Understanding Group to Role Mapping for SSO Authentication in Neo4j Aura - Support, https://support.neo4j.com/s/article/Understanding-Group-to-Role-Mapping 20. Authentication and authorization - Operations Manual - Neo4j, https://neo4j.com/docs/operations-manual/current/kubernetes/authentication-authorization/ 21. Role-based access control - Operations Manual - Neo4j, https://neo4j.com/docs/operations-manual/current/authentication-authorization/manage-privileges/ 22. Vertex AI Agent Builder | Google Cloud, https://cloud.google.com/products/agent-builder 23. Python A2A, MCP, and LangChain: Engineering the Next Generation of Modular GenAI Systems | by Manoj Desai | Medium, https://medium.com/@the_manoj_desai/python-a2a-mcp-and-langchain-engineering-the-next-generation-of-modular-genai-systems-326a3e94efae 24. Neo4j Live: MCP for LLM Agents, APIs & Graphs - YouTube, https://www.youtube.com/watch?v=6igWn_dckpc 25. Mastering LLM AI Agents: Building and Using AI Agents in Python with Real-World Use Cases | by Jagadeesan Ganesh | Medium, https://medium.com/@jagadeesan.ganesh/mastering-llm-ai-agents-building-and-using-ai-agents-in-python-with-real-world-use-cases-c578eb640e35 26. vLLM vs Ollama: Key Differences for LLM Success - Openxcell, https://www.openxcell.com/blog/vllm-vs-ollama/ 27. vLLM vs Ollama: Key differences, performance, and how to run them | Blog - Northflank,

https://northflank.com/blog/vllm-vs-ollama-and-how-to-run-them 28. Ollama or vLLM? How to choose the right LLM serving tool for your use case, https://developers.redhat.com/articles/2025/07/08/ollama-or-vllm-how-choose-right-llm-serving-tool-your-use-case 29. Ollama vs vLLM: A Detailed Comparison of LLM Frameworks - DEV Community, https://dev.to/mechcloud_academy/ollama-vs-vllm-a-detailed-comparison-of-llm-frameworks-513m 30. Ollama vs. vLLM: The Definitive Guide to Local LLM Frameworks in 2025, https://blog.alphabravo.io/ollama-vs-vllm-the-definitive-guide-to-local-llm-frameworks-in-2025/ 31. Vertex AI Agent Engine overview - Google Cloud, https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/overview