

Orchestration: From Agents to Enterprise Intelligence

Approaches and lessons for scaling multi-agentic
orchestration products in enterprise

*Co-Created by ACCENTURE &
HORIZON3 AI LABS at Unilever*

Akshay Bardia

Gary Bogdani

April 2026

Contributing Authors

Alex Mersereau

Amir Ghorbani

Anna Bolislavsky

Anuradha Rajkumar

Ashwini Pingali

Ayberk Aksoy

Belange Nohra

Fahd Khan

Irina Berezovksy

Jenn MacLean

Jyotsna Seghal

Kehinde Oyetunde

Leanne Nosworthy

Mathena Angeles

Mayuresh Oak

Monika Viktorova

Ndubuisi Okpala

Prahalad Ravi

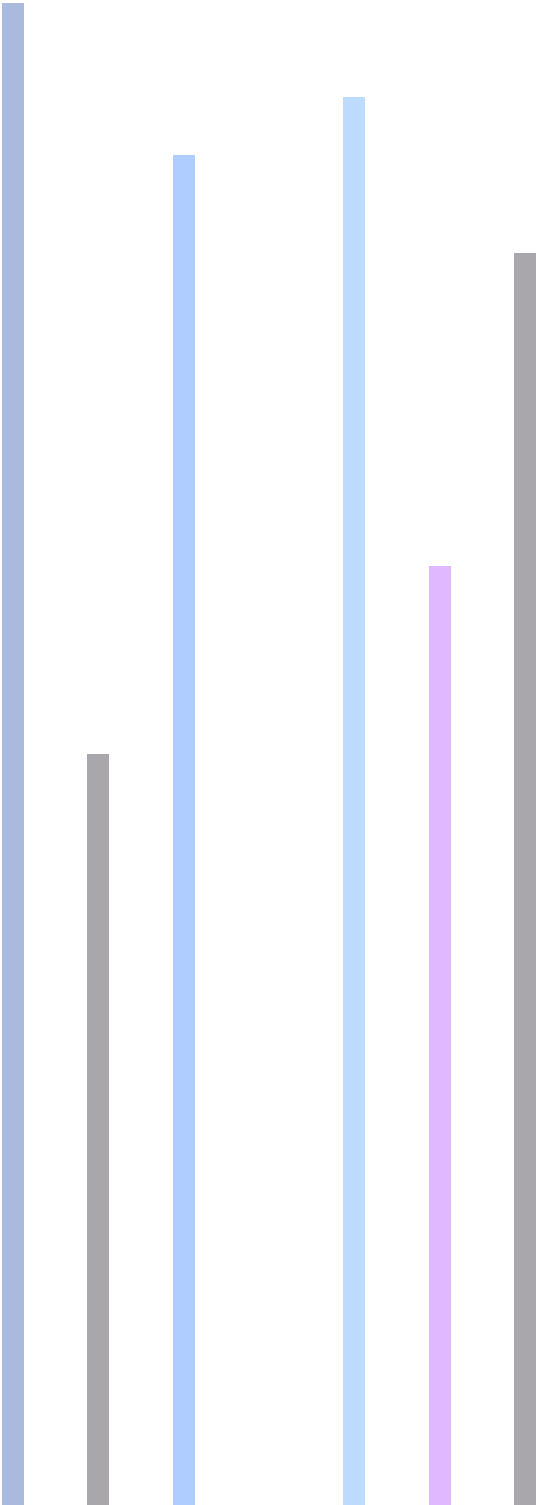
Quanhong Liu

Shrawan Malyala

Tianyu Han

Yuvraj Tomar

Contributing Organization



Contents

Executive Summary	4
Ch 1: Design choices for harnessing orchestration	6
Ch 2: Agentic communication, standards and controls	8
Ch 3: Expanding functionality without ambiguity bottlenecks	5
Ch 4. UI design for a seamless orchestration experience	13
Ch 5. Evaluation, monitoring and governance	16
Ch 6. Lessons for future innovation	16
Conclusion	16
Appendix	16

Executive Summary

As enterprises move from generative AI experimentation to scaled deployment, a new constraint has emerged: coordination. While foundation models and task-specific agents have advanced rapidly, organizations struggle to integrate them into coherent, governable, and reusable systems that align with real business workflows.

Unilever's experience scaling agentic AI across functions highlights a critical insight: the limiting factor is no longer model capability, but orchestration. Without a deliberate orchestration layer, enterprises encounter fragmented pilots, duplicated logic, inconsistent governance, and declining user trust. Productivity gains remain local and brittle rather than compounding across the organization.

In response, Unilever, in collaboration with Accenture and Larus Technologies, designed and deployed an Agentic Orchestrator, a centralized control layer that coordinates specialized AI agents through structured planning, execution, and human-in-the-loop oversight. Rather than optimizing individual agents and risking user overwhelm with too many tools and interfaces, the orchestrator provides a focal point to access the most important business workflows at speed. To provide a seamless user experience, key technical design choices enable the orchestrator to manage intent decomposition, task sequencing, agent selection, exception handling, and response synthesis across an evolving ecosystem of capabilities.

Today, this orchestrated system supports hundreds of business users, compressing workflows that previously required hours of manual effort into minutes, while preserving enterprise-grade standards for accuracy, transparency, and governance. Critically, orchestration enables reuse and scale: new agents can be onboarded without rebuilding workflows, and intelligence compounds as capabilities expand. This paper shares the design and architectural principles, operating assumptions, and hard-earned lessons from building and running agentic orchestration in a complex global enterprise.

As models commoditize and agents proliferate, **orchestration will become the primary differentiator** in enterprise AI, serving as the control plane that enables sustainable value creation, trust, and scale.



01

CHAPTER 01

Design choices for harnessing orchestration

Coordinating enterprise AI agents for scalable, unified decision-making

Why Orchestration is the Real Enterprise AI Opportunity

Modern enterprises generate and rely upon vast streams of data spanning multiple domains — finance, operations, marketing, and customer analytics, among others. To meet the growing demand for rapid insights, organizations are increasingly deploying agentic AI to decrease the manual effort and time taken to retrieve, analyze and generate insights from their data. When Agentic AI is scaled from pilots to production systems designed to support 100s of employees with daily tasks, the core challenge is no longer model quality but coordination.

As the number of specialized agents expands, so does the complexity of managing them. Business adoption of AI faces challenges when users are asked to navigate between multiple tools. Manually coordinating interactions across different systems introduces friction, diminishes user experience, and reduces AI adoption and its estimated productivity benefits.

This paper discusses approaches and learnings for the development of an Agentic Orchestrator (AO) to act as a central control and routing mechanism for an ecosystem of specialized “utility” or sub-agents meeting various business needs. When a user submits a query, the AO decomposes it into smaller, targeted subtasks and delegates each to the appropriate utility agent. It maintains shared memory and contextual understanding to ensure continuity and effective collaboration. The AO manages task dependencies, and sequences actions to preserve logical flow and avoid conflicts. Once the agents complete their assigned tasks, the orchestrator gathers and integrates the results into a coherent, unified response for the user. Additionally, the AO monitors agent outputs enabling feedback loops and iterations as needed to refine results and ensure successful task completion.

The Agentic Orchestrator delivers responses in the interface of the enterprise-wide communication platform used by all business stakeholders. This productionized system allows hundreds of users to access data retrieval, analysis, transformation and other data and research workflows across all domains relevant to them. In this paper we propose an agentic orchestration architecture and approach, including learnings and best practices for enterprises looking to coordinate workflow across a growing suite of agents.

KEY TAKEAWAYS

Orchestration provides the control plane that enables reuse, governance, and compounding value across agents, tools, and workflows. At large enterprises, orchestration becomes essential to reduce tool fragmentation, accelerate decision cycles, and maintain enterprise-grade trust as agent usage scaled across functions.

Effective orchestration requires modular architecture enabling separation of concerns

While individual agents can solve narrow tasks, enterprise environments require coordinated intelligence across data domains, security boundaries, and teams. Without orchestration, agents proliferate into fragmented copilots, duplicated logic, and brittle workflows. Orchestration enables enterprises to treat agents as modular specialists governed by a shared control layer rather than isolated tools.

The Agentic Orchestrator (AO) is the central coordination unit responsible for managing the end-to-end flow of user queries from interpretation and decomposition to routing, execution, and response collection and generation. To effectively orchestrate collaboration across a large group of sub-agents requires a multi-layered system design allowing for task distribution and hierarchization. As a result, the Agentic Orchestrator is designed as dual-supervisor system: it employs two specialized LLM "supervisors"—the Planning Supervisor, focused on strategic planning and query decomposition, and the Execution Supervisor, managing real-time execution and agent coordination. The Planning Supervisor and Execution Supervisor are the two twin brains of the Agentic Orchestrator; together with the Utility Agents they make up the AO system.

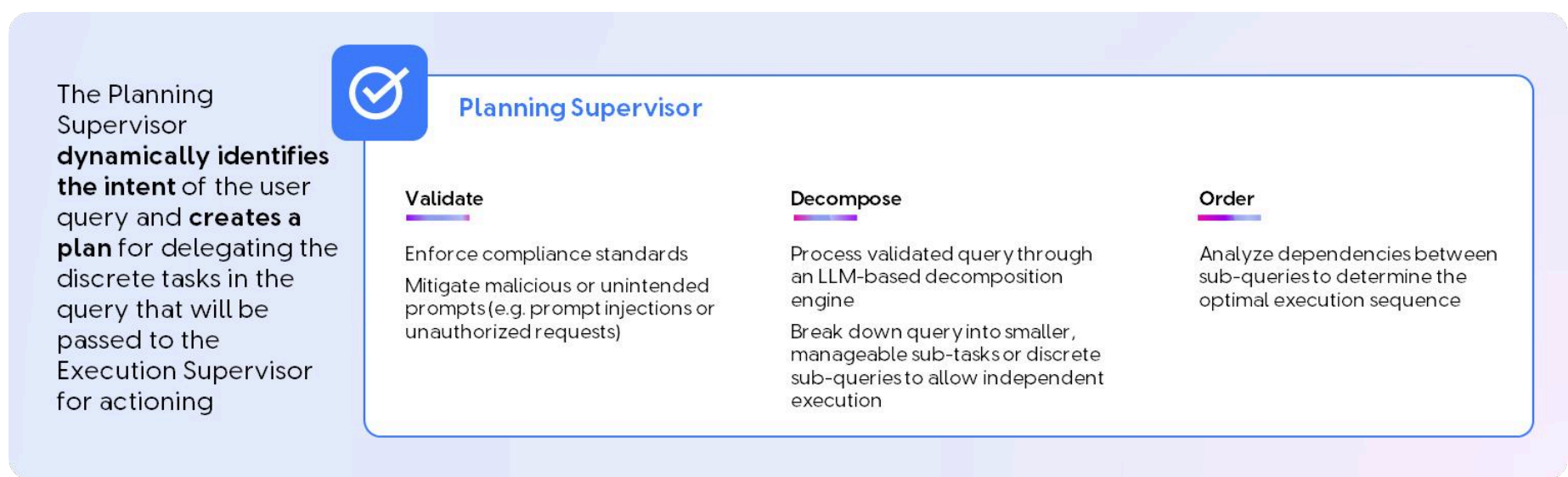
The Agentic Orchestrator system is built with a 'building-block' architecture: components can be connected various configurations, provided they adhere to established custom-built communication protocols described in Section 3. Vertical scaling focuses on enhancing the specialization of the two brains of the AO to achieve a separation of concerns: the Planning Supervisor and Execution supervisor will address different parts of the functional flow, but work together to achieve the overall goal. Horizontal scaling is achieved through collaboration with Utility Agents, each tailored to more complex or domain-specific tasks. These Utility Agents integrate seamlessly with the AO layer via a standardized communication protocol, ensuring extensibility without disrupting existing workflows.

This dual extensibility—horizontal for breadth and vertical for depth—ensures the architecture remains scalable, adaptable, and maintainable as requirements evolve. Moreover, it simplifies iteration cycles, enabling targeted updates and enhancements without impacting the stability of the broader system. Using a modular approach for orchestration serves as a foundation for building a robust, intelligent, and future-proof multi-agent system.

Orchestration begins with understanding user intent and making a Plan

One of the key goals of the AO is to present one easy access point to business users to get answers to data retrieval and analysis questions. Instead of accessing multiple agents for multiple queries, a user accesses a single frontend that engages the AO, which processes requests targeting one or more sub-agents. When a user submits a query through the user interface, the query is passed first to the Planning Supervisor. The Planning Supervisor executes three critical functions:

I. Task Routing and Delegation:



Once the user query is validated, decomposed and ordered, the Planning Supervisor aims to identify which of the available sub-agents (utility agents) will be suitable to answer the query. It does this using Agent Cards (described below), domain-specific knowledge bases containing agent role definitions, task-to-skill mappings, and functional capabilities. After mapping the query to the right utility agents using their Agent Cards, the Planning Supervisor provides the user with a plan, or high level definition of how it plans to answer their question.

II. Global Context Awareness:

The Planning Supervisor leverages state management to maintain a consistent context across various supervisors and agents, ensuring decisions are informed by previous interactions and intermediate results. This planning-first pattern proves critical in enterprise settings, where premature execution increases cost, risk, and user frustration. Explicit planning and approval shifted the system from 'answer generation' to 'decision collaboration'.

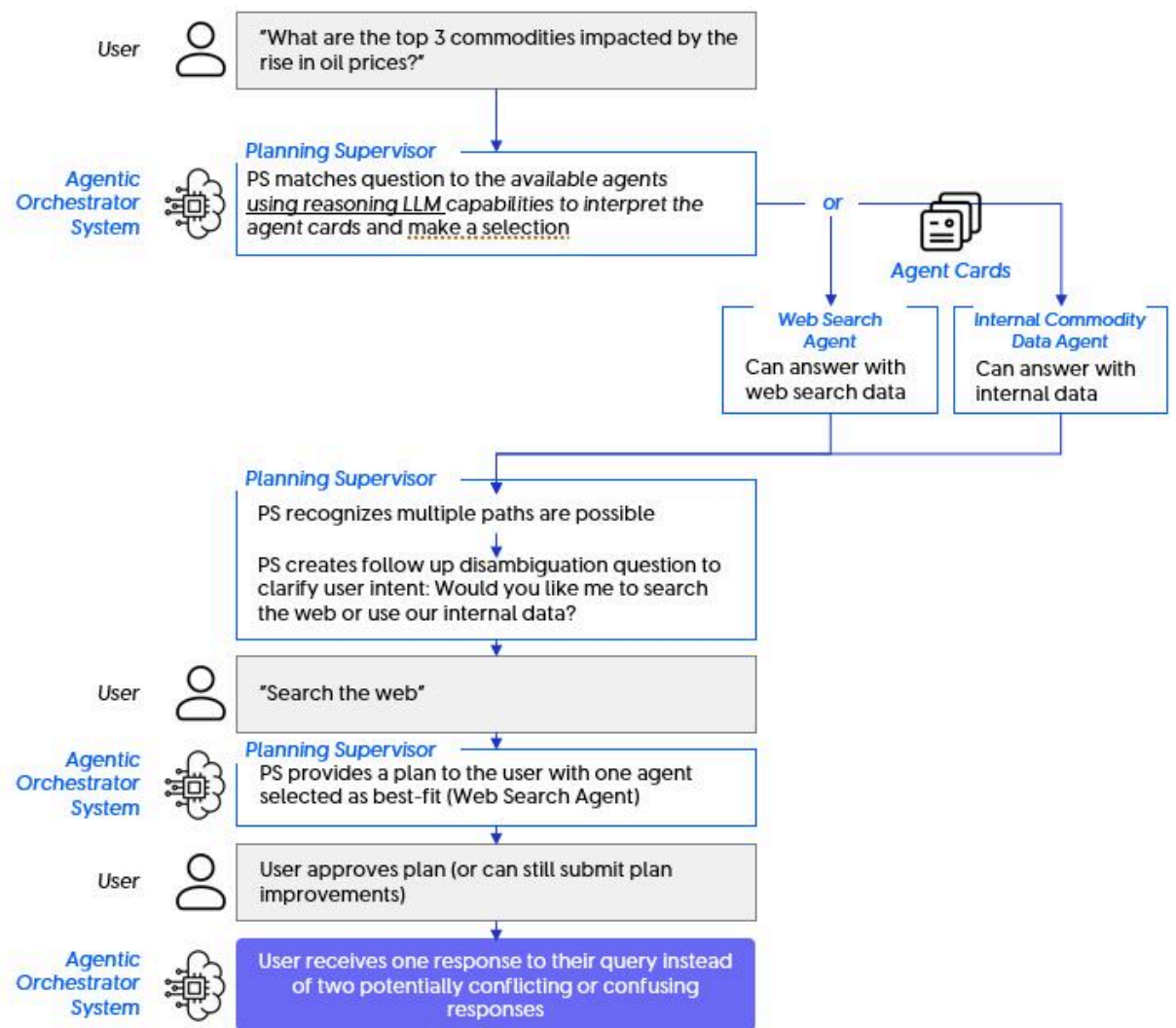
III. Plan Approval or Improvement:

The Planning Supervisor translates its technical strategy into a simple, jargon-free checklist called a “Plan” that is presented to the user. The Plan provides information on how the user query or queries will be answered, including data sources that will be used, to allow the user to review whether the agent has understood their ask and approve the workflow to complete it. A user can approve the plan or make suggestions on how to improve it. The Planning Supervisor will take the suggestions and generate a new plan, and this approve or improve cycle will run until the user is satisfied and approves. The approve or improve model gives users full transparency and control over the agent’s workflow.

Once the plan is approved, the Planning Supervisor dispatches the sub-queries in a defined execution sequence, ensuring that the Execution Supervisor receives the correct task scope and context for processing. The PS relies on Parallelization, instructing the Execution Supervisor on running sub queries meant for different agents in parallel to produce faster output and address user latency concerns.

(Human-in-the)-Loop provides clarification and improves time to value

If a user’s initial request is unclear, the Planning Supervisor loops them into the planning exercise through a multi-turn exchange with clarifying questions before it generates a plan. This human-in-the-loop exercise is meant to address ambiguity that results from a situation where two or more utility agents have different capabilities as outlined in their Agent Cards (prompts and data sources are distinct) and a user asks an ambiguous question whose wording could be linked to both



Similarly to the Planning Supervisor, the Execution Supervisor enables key points of interaction and feedback from users that enable managing real-time collaboration of the user and system. If a Utility Agent gets stuck or finds conflicting data, Execution Supervisor doesn't just fail. It intelligently pauses the plan and brings the user in with a simple follow-up question, allowing the user to make critical decisions mid-execution. This process provides "Transparent Execution" that contributes to a positive user experience. The steps taken by the Execution Supervisor are made available to the user in a dedicated pane to prevent clutter in the main conversation pane. The Execution Supervisor provides simple, real-time status updates (e.g., "cross-referencing supplier data..."), keeping the user informed and justifying the research time by showing the complex work being done in the background.

Transparent execution as a design principle addresses two key user needs: explain ability of how the agent works to create a response and reduction of perceived latency as the agentic system completes execution steps. Addressing both needs reduces potential for user and builds trust. We learned that pre-approved plans are rarely perfect. The system's true power comes from Ares's ability to dynamically pause and escalate exceptions (like data conflicts or agent follow-ups) to the user for a live decision. This makes the AI a true collaborator, not just a rigid script.

The Execution Supervisor leverages a Reporting LLM as a dedicated component responsible for aggregating the final results of the query. Once all sub-query tasks have been executed, any multi-turn disambiguation loops have been completed with the user, and all responses have been sent to the Execution Supervisor from the Utility Agents, the responses are ready to be compiled. The Reasoning LLM synthesizes these results into a coherent and comprehensive report to the user.

Maintaining a dynamic agent registry ensures effective execution flow

The Agentic Orchestrator system uses a dynamic agent registry to provide the Planning Supervisor with the full scope of sub-agent capabilities that can be leveraged to answer a user query. The dynamic agent registry is composed of Agent Cards: JSON file that acts as a "resume" for a Utility agent. The Agent Cards are declarative capability manifests configuration files specifying what each agent can do, example queries, and input constraints. Agent Cards are served to the Agentic Orchestrator via its prompt, enabling the Agentic Orchestrator to understand the capabilities, input requirements and constraints offered by each utility agent.

The concept of Agent Cards was heavily inspired from Google's A2A protocol. They are a novel and cutting-edge way to improve agentic orchestration planning accuracy. This registry-driven approach allows new agents to be integrated without modifying orchestration code and enables the supervisors to dynamically discover and select appropriate agents based on query requirements at runtime.

As larger numbers of Utility Agents came online, a need emerged for an easy way for the teams developing and maintaining those agents to prepare and maintain the Agent Cards. An Agent Card manager was developed as a straightforward and convenient interface for teams building and governing the Utility Agents to create and update their Agent Cards. The maintenance of Agent Cards is a continuous process: as Utility Agents gain or lose functionality, their descriptions, sample prompts, and data sources need to be updated in the Agent Cards to ensure that the Planning Supervisor has access to the best available information with which to make a selection. This tool also helps developers evaluate their agent cards for issues like syntax, logical or semantic errors. The Agent Card Manager further provides teams with an overlap detection feature that allowed easy, rapid comparison between with all the other agents, allowing teams to anticipate disambiguation issues. The insights also help guide creating Agent Card descriptions.

Using Agent Cards allows clearer visibility and management of multiple agents simultaneously, making it easier to track their responses and performance within the overall orchestration. The Agentic Orchestrator system demonstrates We learned that orchestrating auditable sub-agent specialists is superior to a single "mega-AI." Using "Agent Cards" allows the Panning Supervisor to build a custom expert team for each query. This model is more accurate and allows the teams supporting the solution easily add new capabilities (like a new agent) without rebuilding the entire system.

Scale, complexity and Design Constraints

To build at speed and at scale, several design assumptions were captured in the initial solutioning of the Agentic Orchestrator. The design for the Agentic Orchestrator was created while the build and deployment of the suite of Utility Agents was already in progress, with several UAs already deployed. As a result, there were unknowns about the final architecture, full desired scope of functionality, and real-world use of the Utility Agents. To mitigate the scope of unknowns, we captured key assumptions during our solution design:

Agentic Orchestrator design assumptions

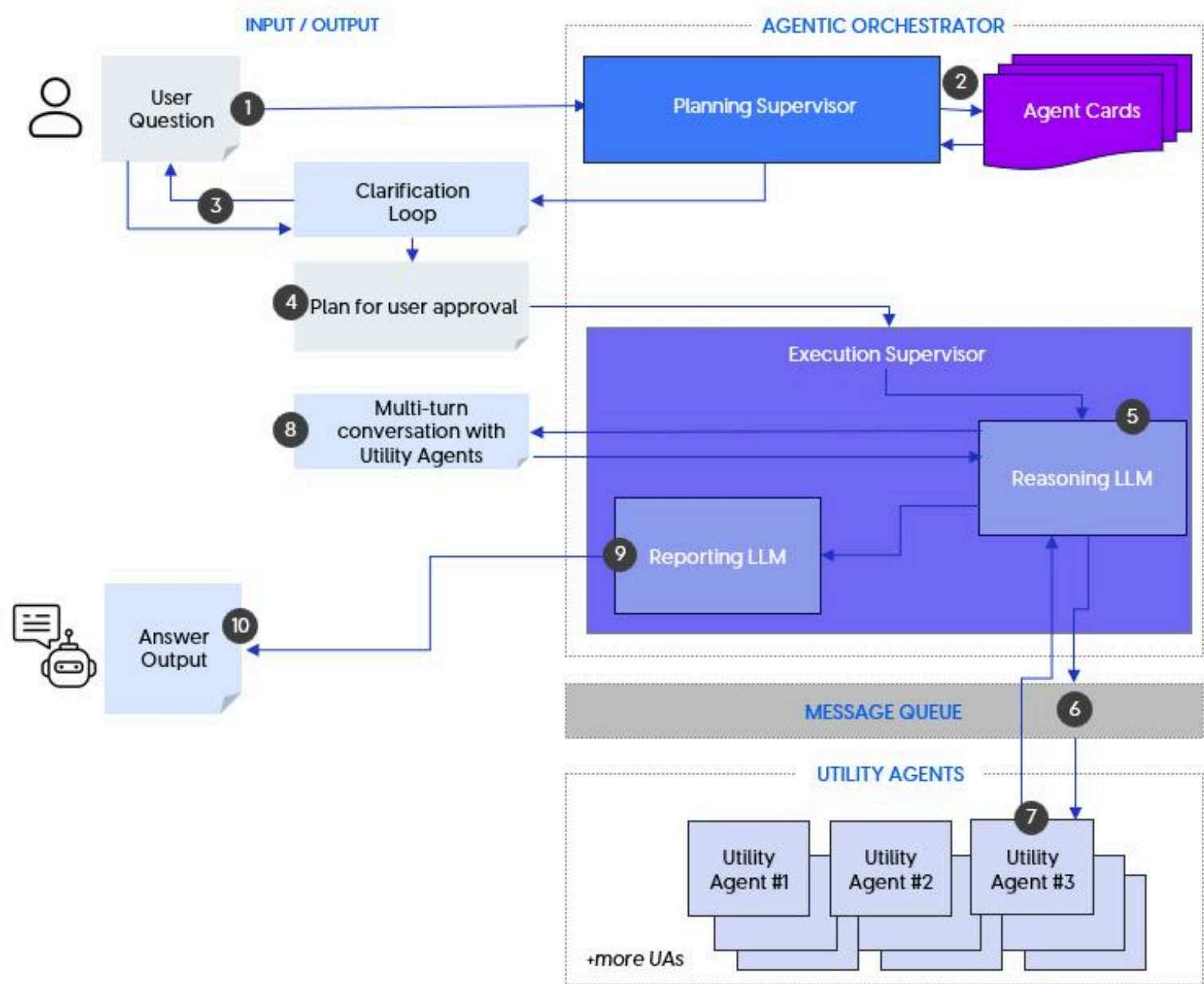
Captured during solution design with key business stakeholders, these assumptions allowed for accelerated build

No overlapping functionality	Utility Agents do not overlap in the types of queries they can answer, nor in the data sources they use to answer queries
No inter-agent communication	Utility Agents will not need to query each other's data or output mid-workflow to execute a task
No inter-Orchestrator communication	Agentic Orchestrator is a standalone product and will not interact with any other orchestrator agents
Natural language UX	Only natural language queries are supported by the agentic orchestrator system and sub-components
Tiered RBAC	Utility Agents will be solely accessible by users through the AO. Access to some, all or none agents RBAC must be implemented at the Agentic Orchestrator level. Data or functionality-related RBAC specific to a given utility agent must be implemented at the utility agent level

Several of these early design assumptions did not hold at scale. Functional overlap between agents proved inevitable as data domains converged, and strict isolation between agents limited flexibility in complex workflows. While teams worked to minimize these overlaps, the converging nature of the business processes the agentic workflows were designed to replicate meant that full elimination of overlap was not possible. These tensions surfaced disambiguation bottlenecks that required both architectural and human-in-the-loop mitigations.

The initial design assumptions underestimated how quickly agent boundaries blur in real enterprise workflows and how often users expect agents to collaborate implicitly. Designing for these realities earlier can reduce later complexity, but may hamper agile practices. Balancing ‘getting it done fast’ and ‘getting it done right’ remains a persistent challenge in designing agentic systems that integrate into business processes.

Overview of the orchestration user journey



Accenture's Cross-Industry Pattern Recognition and Orchestration at Scale

While the Agentic Orchestrator was purpose-built for Unilever's enterprise environment, its architectural principles were informed by cross-industry experience scaling AI systems in complex, regulated, and distributed organizations.

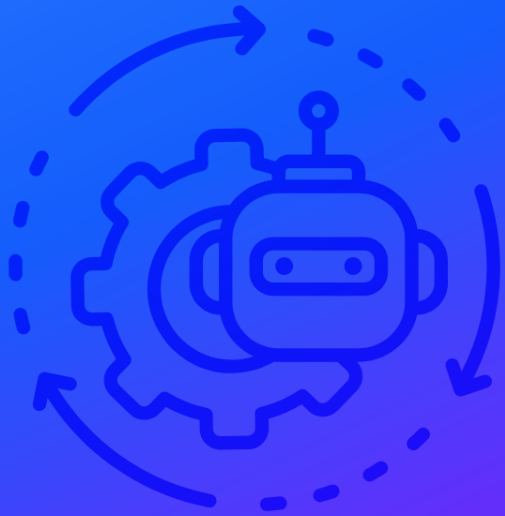
Across industries, we consistently observe a similar maturity curve in enterprise AI adoption. Early experimentation focuses on model performance and isolated pilots. As adoption expands, the bottleneck shifts from model capability to coordination: fragmented agents, duplicated logic, inconsistent governance controls, and limited reuse across business domains. Productivity gains plateau when orchestration is not deliberately designed.

The Agentic Orchestrator reflects lessons learned from this broader pattern recognition:

- Control plane first, agents second: Enterprises that scale successfully establish a coordination layer early. This includes defining routing logic, oversight mechanisms, and shared standards before proliferating domain-specific agents.
- Modularity over monoliths: Durable AI systems treat agents as composable, auditable specialists governed by shared protocols rather than building increasingly complex "mega-agents".
- Governance embedded in architecture: Risk, traceability, and compliance controls must be structural components of orchestration; not post hoc overlays.
- Human collaboration as design principle: In enterprise settings, autonomy without visibility erodes trust. Systems that integrate structured human-in-the-loop checkpoints achieve higher adoption and lower rework.

The dual-supervisor model, structured planning workflows, dynamic agent registry, and communication protocol design were shaped by these cross-sector insights. Rather than optimizing for a single use case, the architecture was designed to be extensible across domains, functions, and evolving business priorities.

This approach positions the Agentic Orchestrator as a reusable blueprint for enterprise AI coordination more broadly.



02

CHAPTER 02

Agentic communication, standards and controls

Modular dual-supervisor architecture for scalable
multi-agent orchestration

In enterprise environments, communication protocols are governance mechanisms as much as technical interfaces. Communication between the components of the Agentic Orchestrator system was designed to centralize authority in the Agentic Orchestrator while ensuring scalability and future-proofing to connect to new agents. While agent-agent and agent-tool communication protocols exist open source, the Agentic Orchestrator system leveraged best practices from those tools through a purpose-built communication protocol and libraries that facilitate communication between many orchestrated agents at scale.

Assuring that components adhere to the standards set by the Agentic Orchestrator for communication. We can reject the addition of utility agents that do not adhere to the communication protocols and standards. In an enterprise environment, an ecosystem of task-specific sub-agents will be scaled and pruned over time, including with off-the-shelf externally built agents. Orchestrator - Agent Communication Protocol future-proofs the Agentic Orchestrator system and enables extensibility across heterogeneity in the ecosystem.

Creating a custom-built orchestrator-agent communication protocol

To coordinate a team of agents, the Agentic Orchestrator system requires a common language. Orchestrator - Agent Communication Protocol (OACP) is a purpose-built novel protocol that enables reliable, seamless communication between the Planning Supervisor and Execution Supervisor and Utility Agents.

OACP is inspired by best practices from both Model Context Protocol (Anthropic) and A2A (Google) protocols for agent – agent integration. While MCP is primarily designed for agent-to-tool interactions, allowing agents to invoke external functions or services to complete specific tasks, it is not inherently optimized for orchestrator-to-agent communication. Particularly MCP is not useful in scenarios requiring structured control, task delegation, or multi-agent coordination, which were critical needs for the OA. In MCP's design, tools are assumed to be non-intelligent functions, and the protocol was not originally intended to support agents as callable endpoints. Further, MCP does not natively provide session management or structured bidirectional communication patterns necessary for effective orchestrator-to-agent workflows. While it's technically possible to use MCP for orchestrator-to-agent communication, it introduces architectural inefficiencies and complexity, making it a suboptimal choice for our use case and project timeline.

OACP inherits the benefits of MCP while optimizing for agent-level communication patterns. It is specifically designed to address the pain points encountered in orchestrator-agent interactions, making it the best fit for addressing the OA system's communication needs and pain points around parallel task execution, latency and scalability. OACP ensures the ES can manage all agents (new or old) seamlessly and without errors.

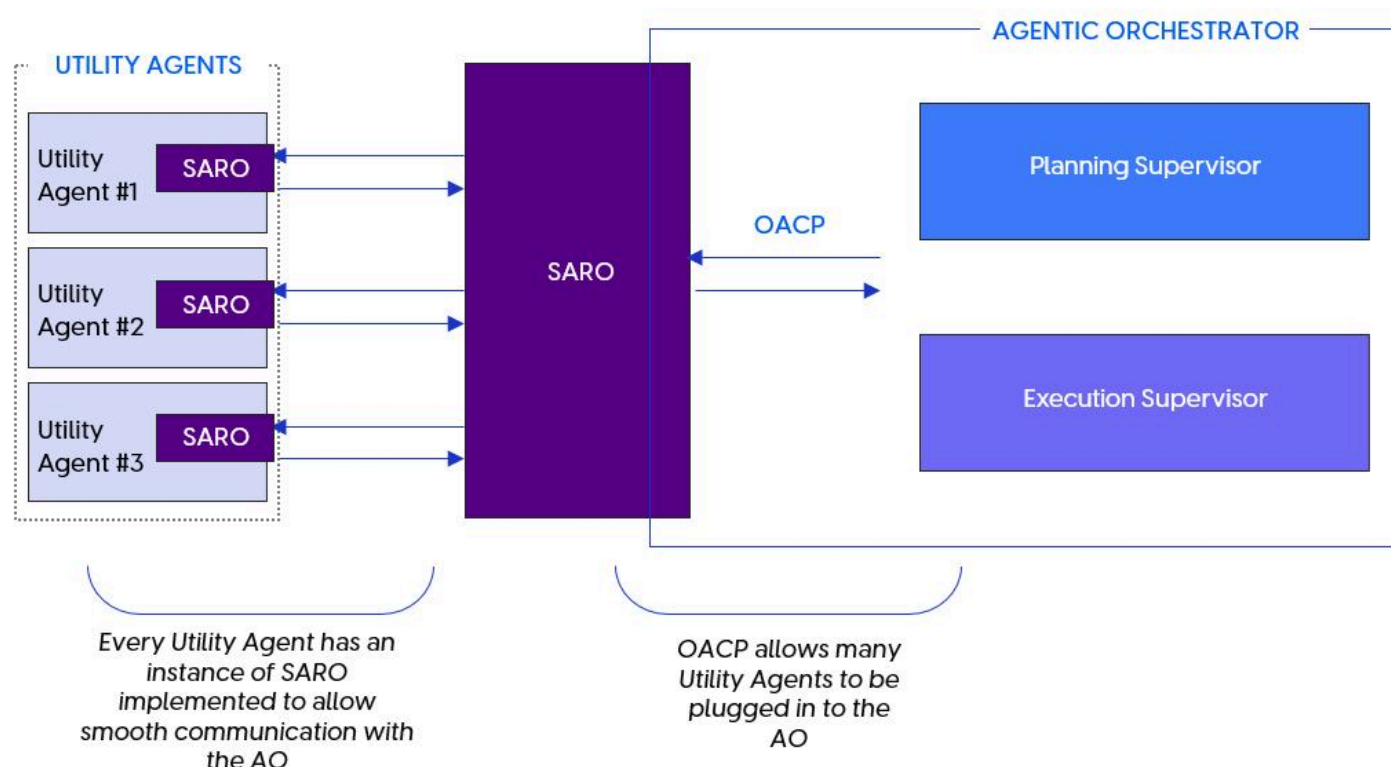
The choice to design OACP reflected a need for deterministic control, auditability, and predictable performance, requirements not met by existing agent-to-tool standards when applied at orchestrator scale.

Enabling communication at scale through a Server Adapter and Router for Orchestrator

To enable the OA and Utility Agents to use OACP, we created Server Adapter and Router for Orchestrator (SARO): a library comprising of a set of rules that standardize and structure orchestrator to agent communication regardless of agent function, build or deployment environment. SARO allows for faster utility agent integration by reducing load on changes required to utility agents. It also allows for rapid onboarding (< 1 day) of new utility agents, facilitating easier inter-team coordination necessary in an enterprise environment.

A key design principle in creating both SARO and OACP was to ensure reusability for future multi-agentic projects requiring orchestration and hierarchization of agents. Below is an illustration of how SARO and OACP are implemented across the system of the Agentic Orchestrator and the Utility Agents, facilitating smooth 1:1 conversation between each individual Utility Agent to the AO, while ensuring the many:1 conversations between the AO and its ecosystem of Utility Agents can happen robustly.

While the OA perceives Utility Agent #1 available through SARO, there may be multiple servers behind that agent load-balancing a request. Ensuring the orchestrator doesn't concern itself with what's behind the Utility Agent simplifies communication. Together, OACP and SARO allow the Agentic Orchestrator to ensure efficient and effective handling of complex orchestration tasks.





03

CHAPTER 03

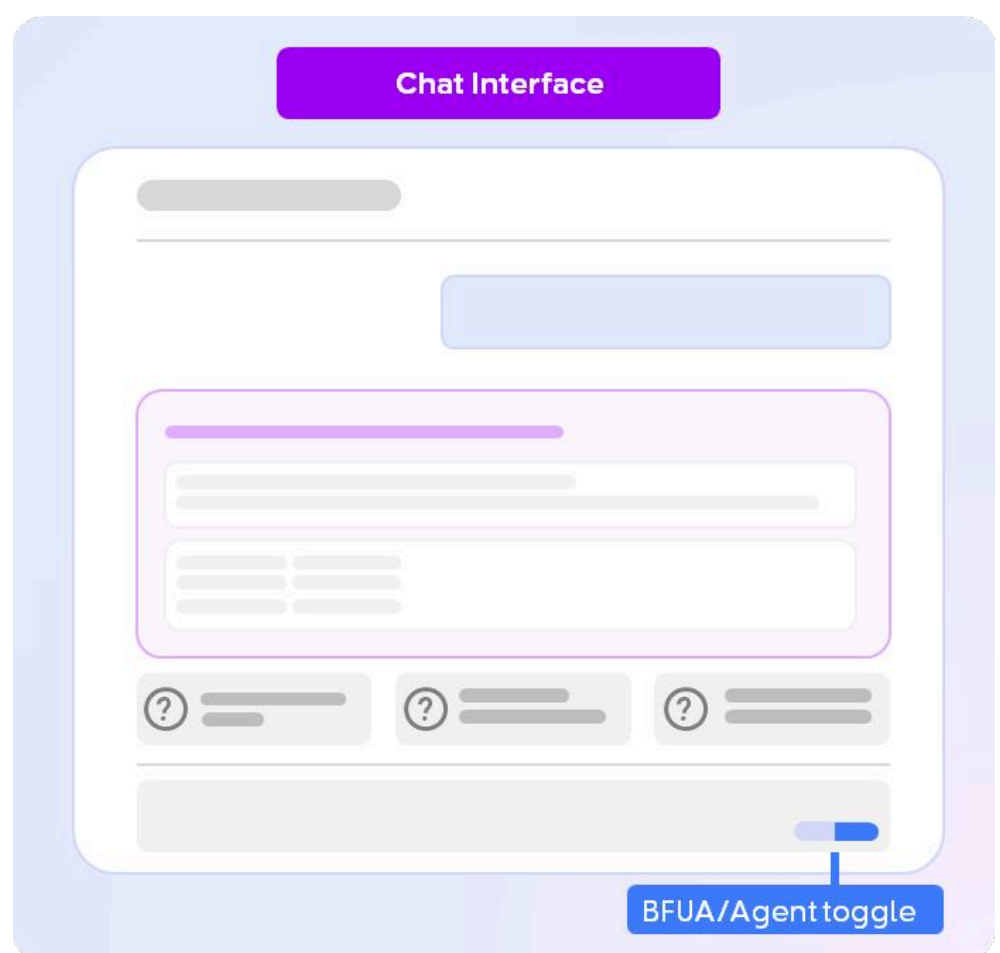
Expanding functionality without ambiguity bottlenecks

Modular dual-supervisor architecture for scalable
multi-agent orchestration

Orchestrating Broad and Narrow Functionality Agents

As the ecosystem of Utility Agents matured, users increasingly asked for broad, “ChatGPT-like” capabilities—ranging from open-ended insight generation to document uploads, ad-hoc analysis, and general web search. Meeting this demand required creating a new Broad Functionality Utility Agent (BFUA), powered directly by Gemini 2.5 Pro, to deliver secure, enterprise-grade general reasoning. Introducing BFUA surfaced a core governance tension: its broad scope overlapped with several specialist Utility Agents that handled web search or document generation. To prevent ambiguity bottlenecks for the Planning and Execution Supervisors, we made an explicit governance trade-off: instead of merging functionality, we designed two deliberately distinct operational modes—BFUA Mode and UA Mode—to partition capabilities between them.

BFUA mode complements the UAs by providing flexible, fast, everyday conversational support, while UA Mode remains the authoritative path for domain-specific operations that may use both external data from web search as well as internal data sources. BFUA Mode and UA Mode share conversation context, allowing users to switch between them without losing continuity. Users can easily toggle between the modes on the UI. This partitioning ensures BFUA does not undermine Utility Agents, which remain the default for authoritative workflows, ensuring traceability, accuracy, and domain accountability.



For organizations facing similar pressures, our primary recommendation is to understand the need for generalized conversational support early and adopt it in the solution design for the orchestration layer. Define where general-purpose intelligence helps in the users’ workflows, where it risks undermining governance controls, and where specialist agents must remain the source of truth. Clarifying this boundary—in governance, capability routing, and user experience—will save time and effort in implementing a system both powerful and enterprise ready.



04

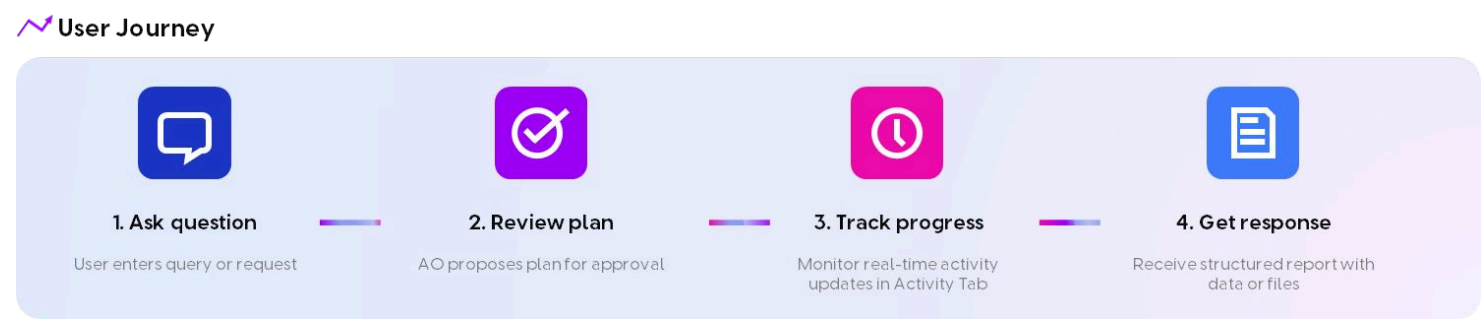
CHAPTER 04

UI design for a seamless orchestration experience

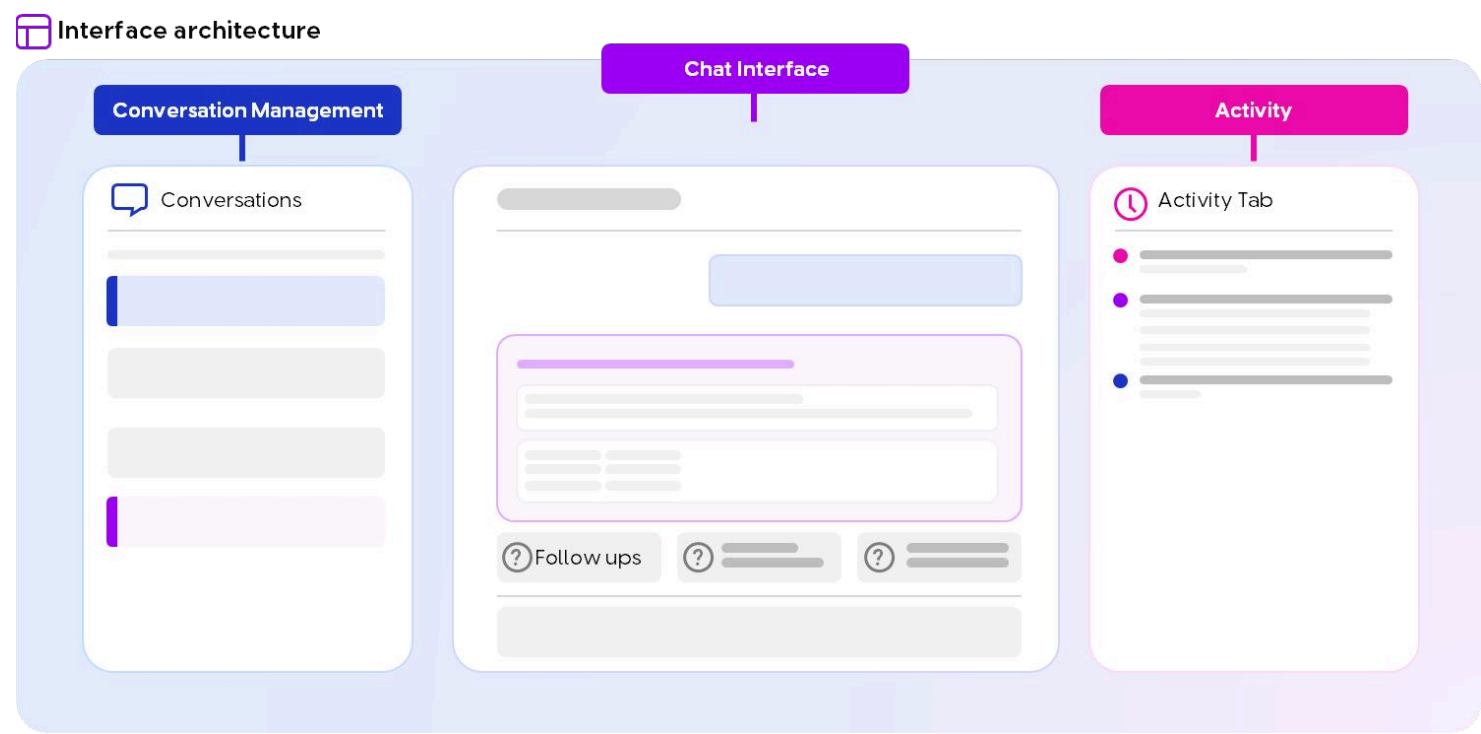
Modular dual-supervisor architecture for scalable multi-agent orchestration

Designing UI for a seamless experience of orchestration as a central tool

The user interface and user experience design of the Agentic Orchestrator (AO) system is tailored to complement the broad functionality enabled by orchestration. An optimal user experience ensures seamless integration with the existing ecosystem of tools that business users have access to, embedding the AO into the native enterprise software they use most. Users are provided with a light level of customizability for themes, and can expect a positive experience during long LLM execution paths through additional features that give them transparency and control.



The UI provides status updates, summary of ongoing work, and multi-turn follow ups with the user with minimal friction. The platform features a real-time activity feed inspired by Gemini Deep Research, displaying live agent operations with precise timestamps and built-in follow-up capabilities. It offers seamless authentication through both Teams Single Sign-On (utilizing silent token acquisition) and the MSAL browser flow for standalone web app access. Users can easily switch between Utility Agent Mode—designed for autonomous research and structured planning—and BFUA Mode for more conversational AI interactions.



The UI enables granular collection of user feedback with quick positive/negative feedback buttons, a detailed menu of options from which to select issues on responses, along with detailed suggestions to enhance AI performance over time. To help users get started quickly, it includes pre-configured sample prompts and templates that showcase available features and use cases. The interface automatically syncs with the Teams theme (Light, Dark, or High Contrast) and also includes a manual theme toggle for the standalone version. Before execution, users can review and approve Orchestrator-generated research plans with full version control and modification options, ensuring transparency and flexibility throughout the workflow.

KEY TAKEAWAYS

Key user feedback validated the choice to maintain simple interface design: fewer features with high stability won out over many features that introduced a buggy experience.



05

CHAPTER 05

Evaluation, Monitoring, and Governance

Modular dual-supervisor architecture for scalable
multi-agent orchestration

Observability as a Trust Mechanism

In agentic products, system observability is critical beyond its standard function in the technical monitoring of infrastructure: it is the foundation of trust, governance, and enterprise adoption.

Observability starts with a defined standard and mechanism for logging events across the full workflow, from the first interaction a user has with the system to all intermediate steps and system outputs, and downstream consumption of those outputs. While there are many open source tracing standards and libraries, a custom tracing standard was created for the AO to unlock development at speed. The system emits structured telemetry events throughout the orchestration lifecycle, stored in an in-memory cache with database persistence. These events are consumed by several tools :observability and monitoring were enabled through a dashboard; llm-based and human-in-the-loop evaluation through an in-dashboard events visualizer.

A web-based visualization dashboard renders these events as interactive timelines, enabling developers to trace execution flows, identify latency bottlenecks, and debug multi-agent interactions. This dashboard allows developers and business users alike to view the health of the system at a glance: whether all components like Utility Agents are live, what performance each component is exhibiting relative to pre-determined metrics, and the ability to export statistics about conversations over a give time frame to monitor metrics like latency. An events visualizer tool within the dashboard also allows users to zoom in to each event in a conversation, giving a full view of interactions between the user, AO, and Utility Agents. Providing this level of visibility allows for rapid troubleshooting when different enterprise teams collaborate to run testing and red-teaming of the system, and ensures visibility for ongoing monitoring.

Why non-determinism requires layered evaluation

Testing and evaluating agentic AI systems is critically important due to their complex, dynamic, and autonomous nature. Unlike traditional software systems, agentic AI systems are non-deterministic and possess reasoning capability. They lack the pre-determined execution steps and control of traditional computer systems and present unique challenges that make rigorous evaluation essential for reliable and safe deployment.

Testing and evaluating agentic AI systems is embedded throughout the agentic agile development lifecycle. Evaluating agentic systems is complex and relies on a number of layers of functional and automated approaches to ensure every component of the system is performing as expected. At each stage of development from ideation to deployment to continuous feature improvement and rollout, unit or “spot tests” were conducted by the build team to trouble shoot prompt design. Once system prompts were stable, automated testing using batch scripts and LLM-as-Judge were introduced, alongside Human-in-the-Loop evaluation.

The components of the evaluation pipeline created for AO system are described below.

Automated ‘Batched’ Tests allow high throughput stress-testing

The first step to designing the evaluation of the AO system is identifying unique failure modes across the flow steps from User Query to Response generation and designing test cases to address them. These test cases evaluate the performance of the Planning and Execution Supervisors and whether the final output matched user expectations across numerous evaluation metrics. Test cases are based on the expected workflows of the underlying Utility Agents.

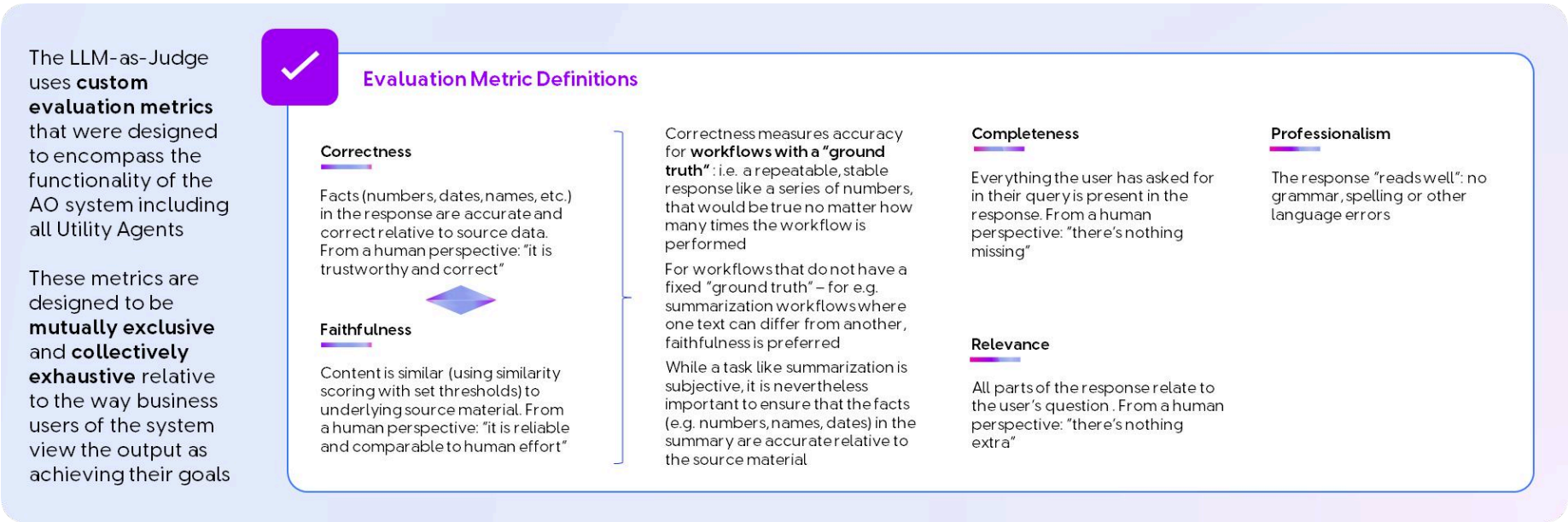
Variations of the test cases explored:

- Query variability: asking similar query structures about different data elements, where the workflow of the agent would be similar but the final answer would differ based on e.g. date range provided
- Language variability: humans may word the same question multiple ways
- Language ambiguity (e.g. intentionally testing for disambiguation protocols activated when questions could be interpreted by the AO to pertain to more than one agent
- Poor syntax/grammar and other common language errors made by humans
- Other edge cases designed to “break” the system or cause misalignment

Scripts are used to automate this process by performing tests in “batches” where many queries would be run through the system and outputs collected. These outputs are then processed automatically first by an LLM-as-Judge and also reviewed by human experts. Batching tests allows for high-throughput stress tests of the system and is effective to evaluate system stability and prepare for user onboarding and acceptance testing.

LLM-as-Judge with custom evaluation metrics provide a scaled health check

An LLM-as-Judge is an evaluation tool leveraging a large language model to review the output of another LLM or agentic system and provide a score on the quality of that output. The AO system leverages an LLM-as-Judge to review automated “batch” testing as well as human led conversations.



The LLM-as-Judge was is used as a first pass detection tool for unintended consequences or errors like hallucinations, bugs, prompt stability and performance and more. It allows for an at-scale approach to evaluation that can often simplify Human-in-the-Loop workflow. It also allows for granular measurement of response quality that gives early indication to the build team if something needs improvement. However, an we must note that the LLM-as-Judge can be fallible and hallucinate or misunderstand its task, so human oversight and evaluation remains the gold standard for judging the performance of an agentic system.

Human-in-the-loop evaluation ensures performance bench marking is trustworthy

The AO evaluation pipeline makes use of Human-in-the-Loop review of AO output. Humans-in-the-Loop included business users, Subject Matter Experts, and members of the build team. They evaluate responses using clear criteria such pre-prepared ground truths or access to source data and documents. In our pipeline, humans-in-the-loop focused primarily on scoring output for correctness or faithfulness, completeness, and relevancy. To facilitate evaluations, humans-in-the-loop made use of the monitoring dashboard, which provided granular visibility to a conversation traces, the LLM-as-Judge review of that conversation across the relevant evaluation metrics, and a first pass score.

Human evaluators provided scores as well as detailed written feedback, where appropriate, that described what went well, what went wrong, and how the answer could be improved. This feedback in turn was leveraged by the build team to troubleshoot AO and UA functionality. Human-in-the-Loop testing is invaluable and irreplaceable: LLM-as-Judge can often fail, and the deepest insights into system functionality are provided when humans are closely evaluating output. There is a tradeoff for scalability given the time-intensive nature of reviewing agentic output, but evaluation pipelines cannot be fully assured without at least some human oversight of performance.



06

CHAPTER 06

Lessons for Future Innovation

Modular dual-supervisor architecture for scalable multi-agent orchestration

Throughout our development and rollout of the Agentic Orchestrator (AO) system, key learnings emerged around building with cutting edge technology while ensuring enterprise goals, standards and best-practices were kept in the forefront. While the AO system has business-specific workflows, its overall design and structure are applicable across large enterprises looking to build orchestrated multi-agentic systems that perform discrete workflows. AO learnings can be leveraged in the design of similar enterprise products cross-industry and business-line.

User centric design is key to adoption and trust

Centering users in the design, development and deployment of enterprise multi-agentic systems is critical because of the high degree of embedding of these systems in previously human-led workflows. Our AO system executes actions on employees' behalf, directly producing work product previously only created by humans. This agentic work product may affect downstream user decisions, and with this incurs risks around errors and unintended outcomes. As a result, employees will only adopt agentic tools they trust to act in line with their goals, constraints, and existing risk parameters. Ensuring product strategy is co-created between technical and business teams is critical to both embedding user-centricity and ensuring later adoption.

User-centered design also focused on providing clear controls, transparency of agentic workflow, and auditability of outputs to reduce fear of loss of control and ensure the AO system felt like a partner instead of a black box. Centering users means giving them steerability: setting goals, guardrails, and preferences so the system augments their judgment instead of replacing it. This preserves human agency, improves satisfaction, and supports a culture where AI amplifies expertise rather than undermining professional identity.

System design requires co-creation of human and machine workflows

User-centric agentic design starts from real tasks and workflows, ensuring agents integrate into existing tools and processes instead of creating extra work or confusion. Building the system starts from understanding the processes we are looking to agentify or replace with agents. Co-creation with SMEs who understand and have performed these processes is crucial. Often the dynamic users will have with these systems would mimic asks from junior team members currently tasked with carrying out the manual workflow. Sometimes the user is asking the system for workflow that they themselves complete regularly, which can increase the likelihood of using colloquialisms or shorthands that will not be immediately understood by the agent.

User Acceptance Testing of the system revealed that users may use colloquial language, shorthands, enterprise-specific acronyms, and could often include spelling or grammatical errors while still having a high expectation that the agent would understand them. We found that a two-pronged approach was best to address these failures:

- Creating technical or process flow mitigation like the clarification loop, disambiguation follow ups, etc. and creating structural division of labor for identifying input intent and actioning on that intent via the Planning and Execution Supervisors
- Providing user education around the limitations of understanding of the system, clarifying that the system is not ‘learning’ from its experiences with user conversations and will not automatically absorb acronyms over time like a human employee would. Users benefited from ongoing touch points that explained features. We often had to remind users that getting the right answer required them to make good use of the clarification tools and human-in-the-loop steps of the system. In short, it required demystifying the use of Generative AI for users, who are often exposed to sensationalized marketing around the boundlessness of artificial intelligence and feel frustration when the user experience falls short of fuzzy expectations set by AI ads.

When multi-agentic systems align with how employees actually work, and take into account their goals, checkpoints, and hand-offs, they unlock the promised productivity gains rather than generating rework.

Real use and user expectations are driven by experiences with other Gen-AI systems

One of the issues consistently highlighted by users was a sense that early versions of the system were “taking too long”. Reducing both perceived and real latency were key goals of the development and production workstreams and relied on:

Perceived latency:

- Providing a “research pane” that broadcast real-time updates of Execution Supervisor activity to a user next to their conversation pane. Users gave positive feedback on both reducing perceived latency through understanding the steps the agent was taking as well as increased transparency on how the agentic system functions
- Providing time stamps for execution of intermediate steps of the process, like the Planning Supervisor providing a Plan to the user. "Live" Status Updates are Essential for Trust: A silent system feels broken. We learned that providing simple, real-time status updates (e.g., "Analyzing supplier data...") is critical. It transforms the user's experience from passively waiting for a result to actively supervising the work as it gets done.

Real latency:

- Architecture improvements to the AO as well as the utility agents
- Answer streaming from the Execution Supervisor to the user

Despite the continuous improvement of perceived and real latency across the system, we continued to receive feedback from users around their expectations for responses not being met. A curious phenomenon happened where data retrieval and analysis tasks that would normally take hours of human labor were reduced to sub 3 minute completion and yet some users remained dissatisfied with the time-to-answer. Based on user interviews, we hypothesized that this phenomenon was driven by power use of other Generative AI tools available on the market in users' spare time.

It highlighted the need for ongoing user education, even with power users of GenAI tools. Because most GenAI tools available from companies like Google, Anthropic and Open AI are extremely user-friendly, many users spend time with the tools but may not have a good underlying understanding of their limitations. For example, we presented a potential real latency improvements involving using a lightweight model for the Execution Supervisor alongside the clear reduction in task accuracy and completion. Users rejected this improvement: their expectation was that quality must remain consistent while latency reduced, citing their perceived lower latency experiences with other systems like ChatGPT. We found that ongoing education around the differences between the AO system and systems like ChatGPT, the types of workflows and integrations with other enterprise tools, and the accuracy limitations of other on-the-market tools that are not immediately obvious to users helped improve understanding and eventually right-size expectations.

Custom builds facilitate going at speed in complex enterprise environments

Throughout the project, we discovered building custom is often better in an enterprise setting because it allows for innovation at speed, tailored design, and quicker access to support.

Creating custom workflows and components for the Agentic Orchestrator and the Utility Agents ensured faithful mirroring of the original business processes, workflows, and risk profile in ways out-of-the-box tools simply could not match. This ensured we were able to build at speed and get the tool in the hands of users for testing and refinement and eventual release as quickly as possible. Designing components like OACP internally also meant that we could ensure rapid tool support and reduce drag on velocity at critical roadmap junctures.

However, custom build does present challenges to potential scaling and inter-agentic system communication in the future. As our Design Assumptions outlined, we had to assume that the Agentic Orchestrator is a standalone product and would not interact with any other orchestrator agent systems. Building at speed can then limit some of the extensibility that business leaders may envision as strategies around which workflows will evolve.

Enterprises must adapt AI risk and governance processes to the needs of agentic systems

As we moved through the build and deployment of the AO system, we worked in tandem with relevant teams across the enterprise involved in governing and assuring technology solutions. Enterprises nowadays often have robust and extensive technology governance across Cybersecurity, Privacy, Legal, and Responsible AI. All of these processes were designed with deterministic systems in mind and are challenged by the reality of agentic products. In an enterprise setting, agentic system may interface with sensitive data, produce different and non-repeatable outputs, take autonomous actions, and collaborate with humans across an entire workflow in ways that other technology solutions do not.

Mitigation strategies were implemented to build trust with users and AI risk and governance professionals in the enterprise. For example, the data sources to be used to answer the user query are described to the user by the Planning Supervisor at the Plan stage. Users also have access to citations in the final response that link to the underlying data sources. These techniques create a safety net that supports AI governance best practices, ensuring the tool would be evaluated positively by relevant governance teams in the organization.

We also found that ensuring cohesion in build practices across different teams responsible for building different Utility Agents was key in meeting the requirements for explainability and transparency often sought in AI governance processes. Standardizing trace collection across the full AO system, including the UAs, was a critical step to ensuring the system could be easily evaluated and those evaluations interpretable by relevant stakeholders.

In addition, we found a need to provide education for stakeholders not familiar with new techniques or functionalities leveraged in our build process. Ensuring all stakeholders have a level-set understanding of the system in laymans terms is critical to providing governance stakeholders the confidence that the AO system has been appropriately vetted, risks logged and mitigated.

Conclusion

The development of the AO represents a significant step forward in operationalizing multi-agent AI systems within a complex enterprise environment. By combining a dual-supervisor architecture, a rigorous communication framework, structured evaluation pipelines, and user-centric design, the AO demonstrates how modern enterprises can successfully coordinate a growing ecosystem of intelligent agents while maintaining control, transparency, and trust.

Our experience highlights several key truths about deploying agentic AI at scale. First, orchestration—not individual agent performance—is the defining factor in whether multi-agent systems can reliably deliver business value. Second, user expectations are shaped by their experiences with consumer-grade GenAI tools, making education and thoughtful UX design critical to adoption. Third, custom-built architectures remain essential for enterprises that require stringent governance, traceability, domain specialization, and deep integration with existing processes. Finally, while multi-agent systems unlock powerful new capabilities, they also introduce new forms of ambiguity and complexity that require continuous iteration, strong internal alignment, and robust governance frameworks.

As organizations continue to expand the scope of agentic workflows, scalable orchestration layers—supported by clear communication protocols, adaptive supervision, and human-in-the-loop safeguards—will be foundational to ensuring both performance and responsible deployment. The lessons captured through the design, testing, and rollout of the Agentic Orchestrator provide a roadmap for enterprises seeking to harness the full potential of agentic AI while maintaining the standards of reliability, explainability, and user trust required in real-world operations.

The Agentic Orchestrator is an evolving experiment: one that will continue to onboard new features, expanding Utility Agent capabilities, and evolve user expectations of what multi-agentic systems can do. We demonstrate that with the right architectural principles and a commitment to continuous learning, enterprises can unlock value from multi-agentic systems that are not only intelligent, but collaborative, governable, and truly transformative.

Editors Note

As the agentic field matures, in 2026 we are exploring major advances in how quickly new agents can be built. While the approaches presented in this paper outline how large enterprises coordinating complex teaming can build at speed, we also want to highlight the new opportunities in the field to build simpler, faster and with a more unified approach. Faster build cycles and growing standardization in agent development would reduce the tradeoffs needed to orchestrate layers built by multiple teams working concurrently, allowing for a more product-forward design that shapes the agent ecosystem by workflows within each domain of expertise. Designing utility agents in a more unified manner from the outset enables stronger guarantees around behavior, interfaces, and interoperability.

Teams can consider simplifying orchestration by exploring more direct interaction patterns between agents, including selective agent-to-agent handoff for well-scoped tasks via agent to agent protocols. While such patterns were not practical under our initial system constraints, they now offer a path to reducing orchestration overhead for simpler, domain-specific queries—allowing users or upstream agents to engage more directly with the most relevant capability. Recent advances in context-engineering —particularly modular, reusable context patterns like Anthropic's work in Skills—can be leveraged to implement cleaner, more efficient reasoning structures, reducing the need for heavy coordination logic while improving consistency and reuse across agents.

We are also exploring how to design agentic ecosystems with a platform-forward approach with a unified memory layer that supports cross-agent communication and global context awareness. As multi-agent systems and their supporting structures - shared-context architecture, flexible coordination patterns - have matured, teams can embrace failing fast and nimbly evolving their agent ecosystem with changing enterprise needs.

Appendix

Glossary

Agent: A process with agency (decision-making capabilities), typically powered by an LLM, comprising one or more agentic units.

Agentic Unit: Smallest independent component with agency.

Supervisor: An agentic unit that manages one or more agentic units; acts as the simplest form of a router.

Orchestrator: A higher-level concept consisting of one or more Supervisors; routes user requests effectively across multiple utility agents

Utility Agent: An agent designed to deliver a specific, clearly defined service or task to users.

Tool: Similar to Utility Agent in purpose but without any agency or autonomous functionalities.

Multi-Agent System: A coordinated environment consisting of multiple agents collaboratively working together, i.e. a utility agent can be a multi-agent system.

Implementation Approach

Technology Stack & Core Infrastructure

- o Python & FastAPI Framework: The AO is built using Python 3.11+ with the FastAPI web framework, chosen for its native asynchronous capabilities. This allows the system to handle multiple simultaneous LLM calls and agent communications without blocking—essential when coordinating several AI agents working in parallel on different aspects of a user's query.
- o Large Language Models (LLMs): The system integrates with Google's Gemini model family—specifically Gemini 2.5 Pro for complex reasoning tasks (planning, report generation) and Gemini 2.5 Flash for latency-sensitive operations (execution coordination, status updates). A provider abstraction layer enables runtime switching between models through an administrative interface, allowing teams to compare performance across different configurations without code modifications.
- o Document Storage & Database: MongoDB serves as the primary data store for conversation history, execution logs, and session management—selected for its flexible document schema that accommodates the varying structures of agent responses and execution traces. Azure Blob Storage handles user-uploaded files (invoices, reports, spreadsheets), with unique identifiers enabling multiple agents to process the same document concurrently

- o **Conversation History Management & Context Engineering:** The system implements a sliding-window context strategy that maintains the last N conversation turns in a structured JSON format, carefully balancing context richness against token limits. Each turn captures the user's original query, all follow-up exchanges, and the final synthesized response—enabling the orchestrator to reference prior answers, avoid redundant data requests, and maintain coherent multi-turn conversations. This context is selectively injected into prompts based on explicit plan directives, ensuring that only relevant historical information reaches each agent invocation.

Prompt Engineering & AI Design Patterns

- o **Dual-Supervisor Architecture:** Rather than using a single monolithic prompt, the system employs two specialized LLM "supervisors"—one focused on strategic planning and query decomposition, the other on real-time execution and agent coordination. This separation of concerns follows software engineering best practices, allowing each component to be optimized, tested, and iterated independently.
- o **Structured Prompt Design:** System prompts are engineered using a modular, section-based format with clearly delineated areas for role definition, behavioral directives, output schemas, and worked examples. Visual separators and hierarchical organization reduce instruction-following drift—a common challenge with long-context prompts—and improve the model's adherence to complex multi-step behaviors.
- o **Few-Shot Learning via Examples:** Each supervisor prompt includes 12-16 comprehensive scenario examples demonstrating expected behaviors across edge cases: parallel agent invocations, error recovery, user disambiguation, file handling, and graceful failure modes. This few-shot approach grounds the model's behavior in concrete patterns rather than relying solely on abstract instructions.
- o **Schema-Constrained Outputs:** All LLM responses are validated against strictly-typed data models (using Pydantic), ensuring that outputs conform to expected structures before being processed by downstream systems. Malformed responses automatically trigger intelligent retry logic with query refinement, implementing a bounded retry pattern that prevents infinite loops while maximizing recovery from transient failures.
- o **Dynamic Agent Registry:** Utility agents are defined through declarative capability manifests—configuration files specifying what each agent can do, example queries, and input constraints. This registry-driven approach allows new agents to be integrated without modifying orchestration code and enables the supervisors to dynamically discover and select appropriate agents based on query requirements at runtime.

