Distributed Multi-Agent LLM Search Experiences

Microsoft NERD Center / Tufts University

Alekha Rao, Ankitha Raman, Carly Seigel, Connor Zamary

05.13.2026

Abstract
One of the primary challenges in autonomous systems is the transition from relatively simple information retrieval to complex multi-user negotiation and communication. This whitepaper investigates the efficacy of agentic communication as a solution for wildly complex and unpredictable transactions. Using the catering industry as a primary case study, we demonstrate how a multi-agent architecture can navigate a complex data environment to complete a transaction, guiding a user from initial requirements to a confirmed booking. Our system utilizes a central User Agent responsible for managing requirements and multiple business-specific agents to independently reason over unstructured vendor data. By allowing these agents to communicate, clarify intent, and negotiate constraints autonomously, we can demonstrate that agentic workflows can potentially replace the traditional manual search.

Introduction
The evolution of artificial intelligence has moved rapidly from static knowledge retrieval to agentic action, but a true test of an autonomous system lies in its ability to manage multi-party communication to resolve complex tasks. The primary challenge in these systems is bridging the gap between simply retrieving information and actively negotiating a transaction. Our research investigates the efficacy of agent-to-agent communication by building an autonomous platform that shifts the user experience from manual search to autonomous intent and execution.

To effectively test this architecture, we required a domain that could not be easily solved by traditional search engines, and we identified catering as the optimal use case. Booking a caterer requires a rigorous balance of objective filtering and deep qualitative reasoning, such as assessing presentation style or menu adaptability. In our initial investigation into the domain, we examined catering email transcripts to identify the time delay and slow points of communication. Because catering represents a high-friction transaction traditionally plagued by these circular email threads and inconsistent data formats, it serves as a good proof of concept for agentic communication.

Our architectural choices were directly informed by the realities of how businesses operate and how users actually behave, which was understood through email threat analysis. We quickly discovered that users often require structured guidance, and real-world catering data does not fit neatly into standardized rows. To address this, we implemented an intake wizard to capture essential constraints before any interaction with artificial intelligence begins. On the vendor side, we built a hybrid data model that includes a structured database for fast filtering and unstructured business folders containing raw menus, reviews, and past communications.

We also had to determine the most effective agent architecture to facilitate these transactions. We rejected the idea of a single omniscient User Agent evaluating all businesses simultaneously because granting a central model access to all proprietary vendor data can lead to hallucinated

comparisons. Instead, we landed on a streamlined, isolated framework orchestrated by Microsoft Semantic Kernel. The system uses a single central User Agent to manage the event context and spins up a single Business Agent for each viable vendor. This mapping enforces data privacy and mirrors real-world interactions, in which vendors evaluate their ability to meet client needs.

This paper details how this architecture successfully moves a user through a progressive pipeline from deterministic geographic filtering to agentic reasoning over absolute constraints to a consolidated, clarifying question exchange. We intentionally design the system to halt at the scheduling of a tasting appointment to preserve the essential human element required for final event decisions. While agentic search can be incredibly effective for fit, specific tastes are subjective and require a human decision for best results. Beyond proving the technical efficacy of agentic communication, we explore future considerations like tracking autonomous decisions to generate powerful data-driven analytics. This would provide a clear incentive for vendors to adopt the platform, as the system would definitely identify where they are losing customers in the filtering process due to things like pricing or a lack of specific dietary accommodations.

Solution

Our system is an end-to-end agentic catering platform that takes a user from raw event requirements to a set of curated vendor proposals and ultimately a tasting appointment booking. The design combines structured data, large-language-model reasoning, and multi-agent communication to progressively narrow a large vendor space to a small set of relevant options.

Data Foundation

The platform is built on a two-layer vendor data model that separates structured and unstructured information. All vendor data used in the system is synthetic and was created specifically for development and evaluation purposes, as access to the proprietary vendor data required for the product wasn't available. A structured Azure SQL database stores data on approximately 1,000 catering businesses, including fields such as location, cuisine types, and contact information. This layer allows for fast, deterministic, "search-like" filtering based on objective geography and cuisine. In addition, there is a set of business data folders, each containing a collection of documents, including menus, past email exchanges, pricing notes, and other unstructured data. This second layer captures the variability of real-world vendor information and provides the contextual depth needed to evaluate fit. These records were generated programmatically using a large language model to simulate realistic business distributions, service areas, and catering offerings. Agents operate strictly on these documents by using them as the primary context for reasoning and generation.

Structured Intake and Event Context

User requirements are collected through a structured intake wizard rather than an open-ended conversation. This design ensures that all relevant event details are captured before any vendor

evaluation begins. Additionally, the wizard enables much faster data collection, since clicking a few buttons is quicker than manually typing out your preferences and priorities. The wizard collects core parameters such as location, date, budget, and cuisine preferences, along with logistical considerations such as service style of choice and any dietary restrictions. It also requires users to explicitly rank key dimensions in terms of importance, distinguishing those must-have constraints from any softer preferences. The result is converted into a structured event context and injected into all the following agent prompts.

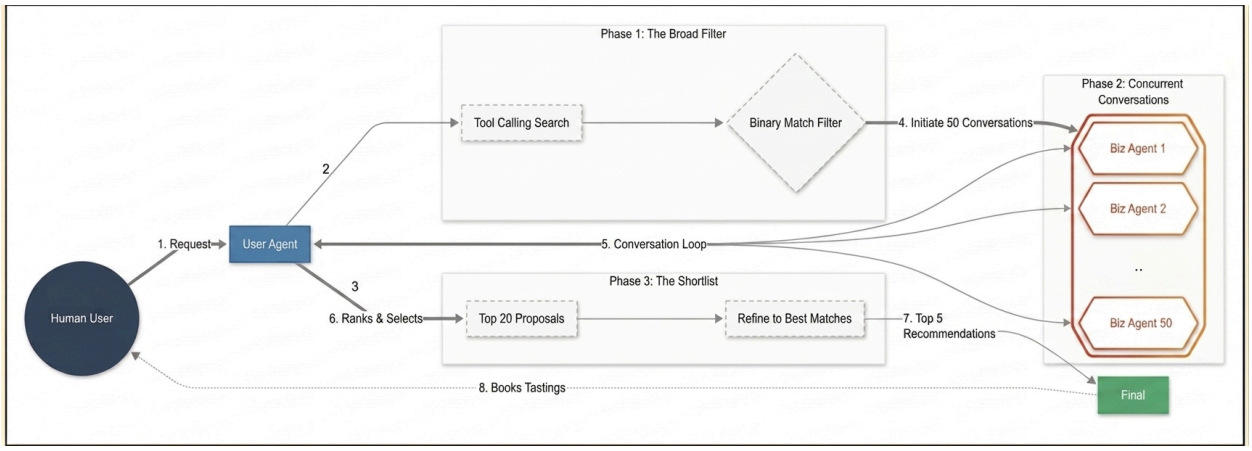


Figure 1: Architecture Diagram

Agent Architecture

As shown in Figure 1, the system is organized around a multi-agent architecture consisting of a single User Agent and multiple Business Agents. The User Agent acts as the central orchestrator, managing the system's overall flow, coordinating interactions, and determining when to transition between pipeline stages. In contrast, each Business Agent represents a single catering business and operates independently, with access only to that vendor's structured data folder. This design enforces strict isolation across vendors while also enabling parallel reasoning. Each Business Agent evaluates its vendor in isolation, which helps mirror real-world competition and prevents information leakage. As a result, vendor selection is not performed by a single centralized model but instead relies on multiple independent reasoning processes.

Filtering, Reasoning & Proposal Generation

The core of the system is a multi-stage filtering pipeline that progressively narrows the business pool while simultaneously increasing the depth of reasoning at each step. The pipeline begins with stage 1, a fast, rule-based process that applies location and cuisine deterministic constraints using the database. Location is enforced through Azure Maps geocoding and radius-based filtering, ensuring that vendors within a realistic distance of the event are considered. This initial stage reduces the candidate set significantly without requiring any language model computation.

Once a manageable set of vendors is left, the system transitions to stage 2, which introduces agent-based reasoning. For each remaining vendor, the platform instantiates a Business Agent that is controlled by the system and is responsible for evaluating that specific vendor. The agent

loads the vendor's unstructured business data into the LLM context and uses it to determine whether the vendor meets all must-have requirements. These are treated as hard constraints, and any vendor that fails to meet any of them is eliminated regardless of how well it satisfies other criteria.

After enforcing must-have constraints, the system moves to stage 3, which refines the candidate set based on the user's nice-to-have preferences. Business Agents that have passed the previous feasibility stage evaluate how well their vendor aligns with the user's softer priorities, such as presentation quality or dietary flexibility. This stage focuses on relative fit rather than eligibility, ensuring that the remaining vendors are not only acceptable but well-matched to the event.

Before generating final proposals, the system introduces a clarifying question exchange that mirrors the back-and-forth involved in a real vendor conversation. Each Business Agent independently identifies cases where the vendor does not yet have enough information from the user, and produces targeted follow-up questions to gather the missing details. Because multiple vendors often ask semantically similar questions, the User Agent merges overlapping questions into the minimum number needed, so the user is not asked the same question multiple times in different wording. The consolidated questions are then presented to the user, whose responses are routed back to each relevant Business Agent before proposal generation begins. This allows each final proposal to directly reflect the user's stated preferences in a way that the intake wizard alone could not capture.

Finally, in stage 4, each remaining Business Agent generates a detailed catering proposal grounded in its vendor's documents. These proposals include menu suggestions, pricing considerations, and a concise assessment of strengths and tradeoffs. The User Agent ranks them based on which are the best fit for the specific event and then presents the top five options. Because each Business Agent is constrained by its own data, the resulting proposals reflect realistic vendor offerings based on that data while remaining personalized to the event's needs.

Not every part of this system is agentic, and that is intentional. Stage 1 is a fast, deterministic filter that requires no reasoning. These components do not need agent-level autonomy and are better served by reliable, predictable logic. The agentic behavior is concentrated in stages 2 through 4 and the clarifying question exchange. In these stages, each Business Agent makes autonomous decisions by accepting or rejecting its own vendor based on reasoning over unstructured documents, generating targeted questions, and constructing a personalized proposal. This architecture enables the system to scale across thousands of vendors simultaneously while keeping each agent's reasoning grounded.

Booking
After reviewing the generated proposals, the user selects which vendor they want to move forward with. The corresponding Business Agent then proposes a set of tasting appointment options based on vendor availability. Once the user selects a time, the system generates a booking confirmation, completing the flow from requirements to a scheduled appointment. We intentionally end the system at the tasting stage to preserve a human decision point, recognizing that final catering decisions benefit from in-person evaluation and direct interaction. This step reflects the importance of human judgment in catering decisions and helps build user trust.

Tech Stack
The system is implemented using a modern, scalable architecture that supports real-time interaction. The backend is built in Python using FastAPI and communicates with the frontend through a persistent WebSocket connection. The frontend is built using React and Vite, with authentication handled through Clerk. Agent orchestration is handled using Microsoft's Semantic Kernel, which provides a structured framework for managing LLM interactions in multi-step, multi-agent workflows. Rather than calling the OpenAI API directly, Semantic Kernel manages kernel configuration, chat history, and prompt execution settings per agent, allowing each Business Agent to maintain persistent conversation history across pipeline stages. For each of our agents, we used GPT-5-chat models and Azure to host the database, deploy the models, and call Azure Maps.

Key Insights

Choosing a Domain
The incentive behind building this application was to explore the power and value of agent-to-agent communication. To achieve this, we had to pick a domain to ground our exploration in. We considered a variety of different domains before landing on catering. Two main categories led us to our decision: fit vs quality.

A key theme throughout this project was determining when and why you should rely on agents vs traditional search. Search queries involving metrics associated with “fit” are often well-suited for traditional search. These are characteristics that are objective with few–if any–major caveats. An example domain that falls into the “fit” category is plumbing. If there is a plumbing issue in your house, you want to find a plumber who is nearby, available, and highly rated. There are likely no other major caveats or qualifiers. We considered what a plumber-finding agent would look like and what problem that would solve. We ultimately decided this would not be a good domain to highlight multi-agent search, as it can be solved with traditional search.

On the other hand, search queries that depend on qualitative metrics are more subjective and cannot be answered simply by web browsing. An example domain that falls into the “quality”

category is restaurant searching. Maybe you are looking for a “cool” restaurant that serves the “best” burger-and-fries dish. What metrics should an agent use to decide what qualifies a burger and fries dish as “the best?” Is it the restaurant with the highest ratings that happens to serve a burger and fries dish (even though maybe their burger is the worst item on the menu)? The cheapest offering? The most expensive offering? The one with the crispiest fries? How does the agent decide what makes a restaurant “cool”? Is that solely based on reviews? Photos? What would an agent in this case look like? We liked that this domain left room for agentic reasoning and decision-making, but ultimately felt we had to rein it in.

We decided that the optimal use case for a multi-agent system is one that balances both fit and quality. That is, a scenario in which there are clear facts but also considerable variability. With this in mind, we chose catering as our domain. Using agents to find caterers seemed like a mix of objective and subjective criteria. For example, price is generally within a fixed range, dietary restrictions can vary but have a concreteness to them (e.g., vegetarian options or not), location is fixed to a specific region, the “vibe” of a menu is very much up to interpretation. Ultimately, this hybrid of fit and quality made catering a clear domain choice.

Formatting Vendor Data

To ensure our agents found the best vendors for the user’s event, we needed data they could use to base their decisions on. While determining what types of data would be necessary to select a caterer, we quickly realized that humans rely on far more than just the menu and pricing. Humans read reviews, look at photos, and contact businesses when questions arise, whether that be about dietary restrictions or the dimensions of certain menu items. As such, it was integral that we included a variety of data for the agents to synthesize.

Initially, we had a structured JSON schema containing all relevant business information: contact information, menu items, dietary restrictions, and menu substitutions/modifications. These set fields were complemented by a free-form facts section containing any notes and reviews about the business and specific food items. While this methodology yielded accurate results for the user, we found a significant flaw: the structured form of the business data did not reflect the nature of real catering menus. Different catering companies format their data differently; some price by person, some have tiered packages, and some price by item.

It was essential that our agents could handle a variety of data types and formats, so that our application could run successfully if a real business were to upload their own documents in any format. Our finalized approach consists of a database with basic information on each business for fast filtering– company name, location, cuisine– and a folder with more detailed information for each business. These folders contain synthetic business data including the menu, past email exchanges, reviews, and operational notes in order to give our agentic system a holistic overview of what each business can provide.

Collecting User Information

One of the earliest questions we found ourselves asking was: What information does the User Agent need to initiate agent-to-agent communication? In the first iteration of our catering-finder application, the UI was a simple chatbot. The user was greeted by the User Agent and prompted to enter any information about their event and the kind of food they wanted.

We soon realized that there was a key flaw to this design: people don't always know what they want or what they are looking for. We turned to sites like Airbnb and Google Flights, which both give you ample filter options with criteria you probably wouldn't have even realized you cared about unless explicitly asked. To collect the most useful information for the User Agent, we decided to refactor our UI. Instead of spinning up a chatbot right away, the program starts by presenting the user with a form wizard to gather multi-modal input. Once the user fills it out, the chatbot UI is introduced, the User Agent receives the user's input, and agent-to-agent communication can begin (after deterministic stage 1 filtering is complete). It is important to note that there is a text box at the end of the wizard where users can enter any additional information they wish to share. The use of the wizard in conjunction with the chatbot interface enables a more robust data-collection process and, subsequently, a more tailored, accurate end result than when we were using the chatbot alone.

Business Orchestrator Evolution

In our initial architecture diagram of the Business Agent, we envisioned each catering company represented by a series of AI agents–each one responsible for a certain role. For example, maybe each caterer would have a scheduling agent, a menu agent, a pricing agent, etc. We very quickly started to have concerns about how exhaustive that process might be. Since we were new to agentic communication–let alone multi-agentic communication–we wanted to keep it simple and limit the amount of agents in our system. We do, however, believe that a sub-agent type of architecture could be a good idea for future researchers to explore.

Throughout this journey, we have learned that agents are powerful but not the solution to every task. We now believe that the key to leveraging agent-to-agent communication is identifying when agents are useful and when they are unnecessary. Agents seem to be most useful for autonomous tasks like decision-making that extend beyond a deterministic algorithm. On the other hand, they seem to be unnecessary when it comes to binary outcomes like whether or not a catering company is within a user's specified location. Furthermore, one of the advantages of LLM agents is that they have the ability to synthesize a variety of unstructured data. With this in mind, we continuously asked ourselves the following questions: Would creating sub-agents, each one responsible for one concentrated area (ex. pricing, menu, scheduling, etc.), be an unnecessary–and perhaps even wasteful–layer of abstraction? Would this sort of pigeonhole-setup fail to reap the full benefits that AI agents have to offer?

As we began scaling up the number of businesses we wanted to have, we could not justify spinning up three (or more) agents for every one potential caterer. We started to ask ourselves if one agent (per business) could be responsible for ingesting all business-related information–whether it be about food items, dietary modifications, customer reviews, pricing, or past customer email exchanges. This idea–of one agent per business–was ultimately the path we chose. In our final architecture, each Business Agent has access to a folder containing unstructured data provided by their respective catering company. The agents can access their assigned business's data and make decisions autonomously. While we were overall satisfied with the results of this architecture, as mentioned above, we believe comparing this design to a "multi-agent per business" model is worth investigating.

Who is doing the heavy lifting?

When a business gets eliminated from not meeting the user's criteria, whether that be due to their needs or preferences, which agent is making that final decision? We chose the Business Agent itself to determine whether it can meet the user's demands. This approach ensures data confidentiality for catering companies, demonstrates true agentic communication, and most closely mimics real life. If you were calling a catering company to determine whether they would be a good fit for your catered event, you would ask questions and get responses. You yourself would not be reviewing their data to see whether they meet your criteria.

While we had concerns that these Business Agents could be manipulated to game the system through the data given to it, which would ensure their business always passed the screening, we found that explicit prompting mitigated this issue. Although it is slightly odd that the Business Agent self-eliminates its own company, the alternative, being the User Agent determining what businesses fit the user's event, brought up other problems.

If the User Agent were determining which businesses were well-suited and which were not, it would require access to each business's data. As a result, this methodology would constitute a breach of privacy because the User Agent would be able to read a business's proprietary information. Furthermore, if the User Agent was doing the "heavy lifting," our multi-agentic system would no longer consist of multiple agents interacting with each other. The sole role of the Business Agent is to ingest and synthesize the data of the business it represents to make decisions that best support the user. Had the User Agent been doing this work, we would unnecessarily have these Business Agents that simply store data.

Could we turn this into a real product?

Users have a wizard and chat interface to enter their event details and converse with an agent to filter results and find the top caterers that match their event. To turn this into a real product with real vendors, we needed a way for them to provide their data. We created a separate UI for businesses, with features for signing up and uploading their documents.

The value for a user of our application was clear from the start: a user could find a catering company that fit their event criteria in a short time and without the back-and-forth hassle and circular conversations that come with communicating the same information to multiple businesses. We needed to identify the value for the businesses. If this were a real product, why would a caterer decide to sign up with us? What is their incentive to give us their proprietary data?

We realized that we could provide business insights into their successes and failures in acquiring customers. With enough usage, we could learn the typical customer persona for a specific caterer and identify the shortcomings of that caterer. For example, we would be able to tell a business that they were losing potential customers because they lacked gluten-free options or because their per-person pricing was out of budget for most customers.

This concept of providing metrics to help a business gain insight gave us confidence that this senior capstone project could extend beyond the classroom. It was interesting for us to think more entrepreneurially than technically. And we hope that with the right resources, someone else could take these findings and expand upon them.

Results & Discussion

To evaluate the performance of our system, we developed a testing suite comprised of 50 distinct event scenarios designed to capture a wide range of real-world catering requests. These scenarios varied across key dimensions, including event size, budget, cuisine preferences, dietary restrictions, and more. Each scenario was run through the full pipeline, allowing us to measure system behavior across all stages. It is worth noting that all business data used was synthetic.

We evaluated four primary metrics: LLM cost distribution, pipeline runtime, effectiveness of filtering process, and user satisfaction.

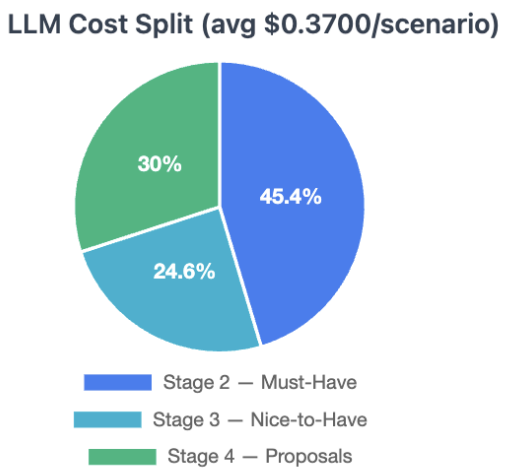


Figure 2: Average LLM Cost Distribution

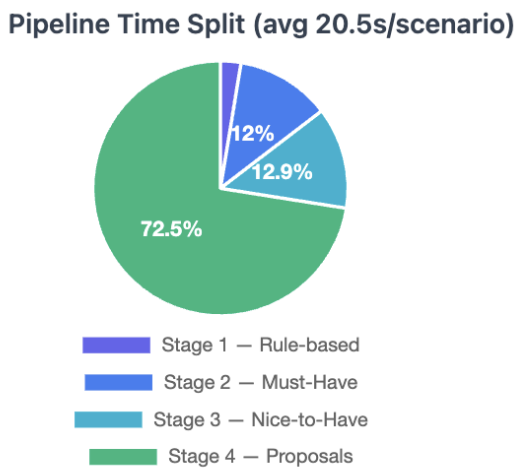


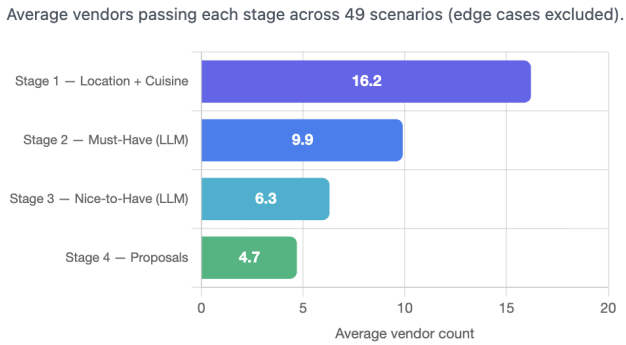
Figure 3: Average Pipeline Time breakdown

Cost Breakdown by Stage

Figure 2 shows the average LLM cost distribution by stage. The largest share of the cost is incurred during stage 2, with an average total cost of approximately $0.37 per scenario. It is not surprising that stage 2 is the most expensive. It is at this stage that the Business Agents are created, so it makes sense that most of the cost comes from here.

Time Breakdown by Stage

Figure 3 shows the time spent at each stage from the moment the app starts until a tasting appointment is booked. The total runtime per scenario is approximately 20.5 seconds, with the vast majority of time being spent in stage 4, the proposal generation stage. This can be explained by the fact that this is the most computationally exhaustive step for the Business Agents. When Business Agents create a proposal for the User Agent to evaluate, they consider all available business data to curate a menu and pricing scheme that best addresses the user's requests. This is the only stage in which Business Agents actually have to generate and deliver something. In stages 2 and 3, a Business Agent is only responsible for telling the User Agent if their business meets certain criteria and should therefore be moved to the next stage. So, it makes sense that much more time is spent in the final stage.



Stage	Avg vendors	% of S1
Stage 1 — Location + Cuisine	16.2	100.0%
Stage 2 — Must-Have (LLM)	9.9	61.1%
Stage 3 — Nice-to-Have (LLM)	6.3	38.9%
Stage 4 — Proposals	4.7	29.0%

Figure 4: Funnel Process Analysis

Effectiveness of Funnel Architecture

Figure 4 presents the funnel process analysis across all scenarios. On average, the number of vendors decreases at each stage of the pipeline, from approximately 16 after the initial filtering stage to around 4-5 in the final stage. This pattern is consistent across the testing suite and indicates a successful funnel process. There is a clear decrease in the number of vendors that make it through each stage, which is exactly what we were expecting to see.

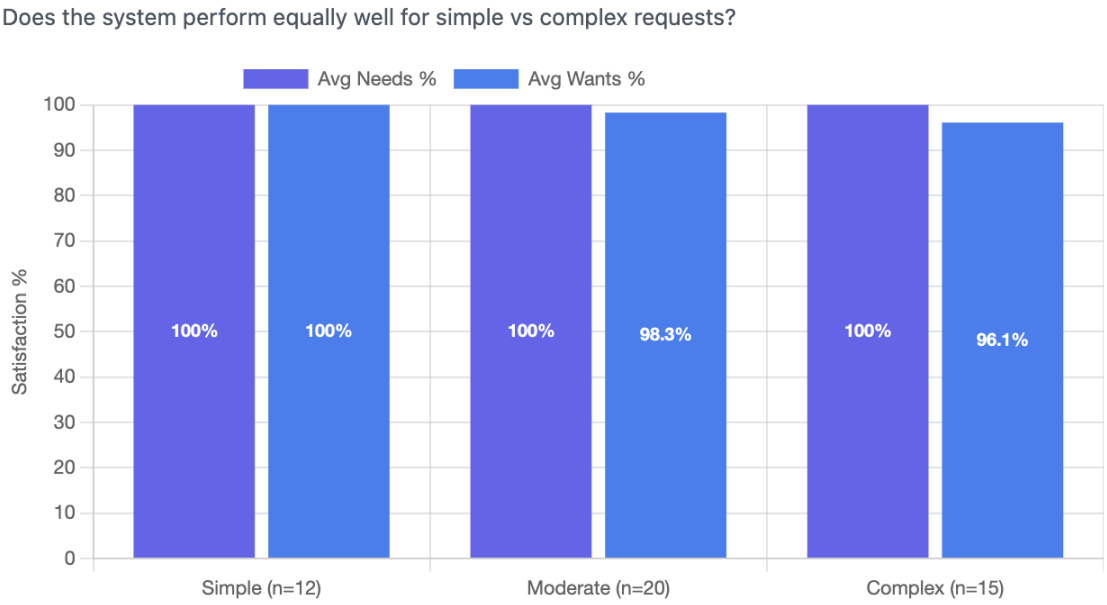


Figure 5: Request Complexity Analysis

Satisfaction Level vs. Complexity of Request

Because the end result of our application depends on requirement- and preference-based filtering, we analyzed how our system performed across a range of request complexities. In

Figure 5, we ran 47 scenarios–12 with simple requests, 20 with moderate requests, and 15 with complex requests. A simple request, for example, might not require dietary restrictions and might fall within a broad budgetary range, while a complex request might require a specific presentation of the dishes or specific dishes themselves.

Throughout these scenarios, we found that the user's requirements were always met. This finding was essential to our application's performance because we were strict about our system only presenting vendors that meet our users' needs. The preferences, however, were not always met. As the event complexity increased, the user satisfaction indicating preferences being met by recommended vendors decreased by about 2% from the last complexity group. These results emphasize that our system does its best to match the user's wants with the caterer's abilities, but since these are not required elements, our system will still present these caterers. Overall, this decrease in user satisfaction is extremely small, and our multi-agent system successfully matches a user's wants and needs.

Future Considerations

The current architecture provides a robust proof of concept for agentic negotiation, but the potential for expansion lies in deepening the autonomy of the business-to-user relationship and refining the commercial value proposition for vendors.

Autonomous Price Negotiation

While the current system evaluates static pricing data from unstructured documents, a logical next step is enabling active price negotiation. By providing Business Agents with a negotiation envelope, such as a range of acceptable profit margins or discretionary discounts, the agents could engage in real-time bidding. If a user's budget is slightly below a vendor's standard rate, the Business Agent could autonomously offer a modified menu or a mid-week discount to bridge the gap, mimicking the flexibility of a human sales representative without the administrative delay.

Expansion of the Vendor Analytics Suite

The most significant entrepreneurial opportunity lies in the data generated during the elimination stages. We envision a comprehensive dashboard for vendors that tracks Loss Reason Distribution. Vendors would receive automated reports indicating exactly where they were filtered out, whether due to a hard constraint, such as a lack of vegan certification, or a soft preference, such as presentation style. This creates a feedback loop in which businesses can use AI-driven insights to adjust their service offerings or menu structures to capture missed market segments.

Integrating Real-Time Availability and Scheduling

To move beyond the tasting appointment and toward a full transactional close, the system requires deep integration with third-party calendar and logistics software. Future iterations could allow Business Agents to cross-reference event dates with real-time staff availability and kitchen capacity. This would transform the system from a recommendation engine into a comprehensive logistics coordinator capable of managing the entire booking lifecycle, from initial inquiry to final confirmation.

Cross-Domain Adaptability

Finally, of course, the original intent of this project was to expand this concept to any complex searching requirement. The fit-versus-quality framework established in this research suggests that this multi-agent architecture is highly transferable to other high-friction industries. Fields such as commercial real estate leasing or corporate recruiting, which require a similar balance of hard constraints and qualitative reasoning, could utilize the same User-Business Agent split. Testing this architecture in a higher-stakes domain would further validate the reliability of agentic communication as a replacement for manual procurement processes, which were difficult to implement under current conditions, since large-quantity transactions are considered untrustworthy to be handled autonomously by AI.

Conclusion

The transition from simple information retrieval to complex autonomous conversation represents a significant jump in the evolution of AI. This research demonstrates that by utilizing a multi-agent architecture in the catering domain, agentic workflows can successfully bridge the gap between intent and execution for high-friction transactions. By isolating vendor data within independent Business Agents, the system addresses critical data privacy challenges while mirroring the competitive nature of real-world environments. The success of the funnel architecture is evident in its ability to systematically refine a large pool of vendors into a manageable set of curated proposals. Furthermore, the project highlights a clear path towards commercial viability by transforming autonomous decision-making into actionable analytics for vendors. While the current system preserves the essential human element at the tasting stage, future iterations that incorporate autonomous price negotiation and deeper logistics integration could further streamline the procurement process. Ultimately, the fit-versus-quality framework established here provides a robust blueprint for applying agentic communication to other complex, high-friction industries.