Bazham Khanatayev, Salah Elbakri, Zongze Li

Megan Hazen, Flipkart Group

DATA 591: Data Science Capstone II - Project Implementation

Mar 16, 2025

Final Delivery: Learning Document

1. Introduction and Preparation

a. Project Objectives and Timeline

Agentic frameworks transform Large Language Models into autonomous entities within an integrated system architecture, enabling robust inter-agent communication and coordinated task execution. These frameworks facilitate sophisticated multi-agent collaboration, where specialized agents leverage complementary capabilities to accomplish complex objectives with greater efficiency than single-agent approaches.

The implementation of a hierarchical agent architecture, powered by the Autogen framework, enables precise orchestration of conversational flow and procedural execution. This structure supports specialized agent roles, optimizing the resolution pathway for marketplace seller inquiries through deliberate agent interaction patterns and clearly defined communication protocols. The resulting system demonstrates enhanced problem-solving capabilities and significantly improved resolution rates for complex seller-related issues. One might ask if the Autogen framework is necessary to create such a system. Theoretically, LLMs can be prompted and configured from scratch, requiring advanced software engineering expertise and a custom orchestrator. One key advantage of Autogen is its built-in functions, which streamline communication between agents. Our team successfully leveraged the Autogen framework to configure the agents and define their interaction protocols efficiently.

To successfully design LLM agents for the extern project, establishing a clear timeline has been crucial. We set our initial project goal in October 2024, followed by the submission of our team bios and outlining of preparatory work in November. From December 2024 to January 2025, our primary focus was on building background knowledge by reviewing various online resources, including designing finite state machines and understanding the construction of autonomous LLM agents. In late January, we received instructions for the first major SOP and worked on its development through late February. During this phase, we refined the SOP by implementing additional functionalities and demonstrating the system's extensibility to accommodate other SOP requirements. By mid March, we successfully built the user interface using Streamlit.

b. Preparations before Implementation

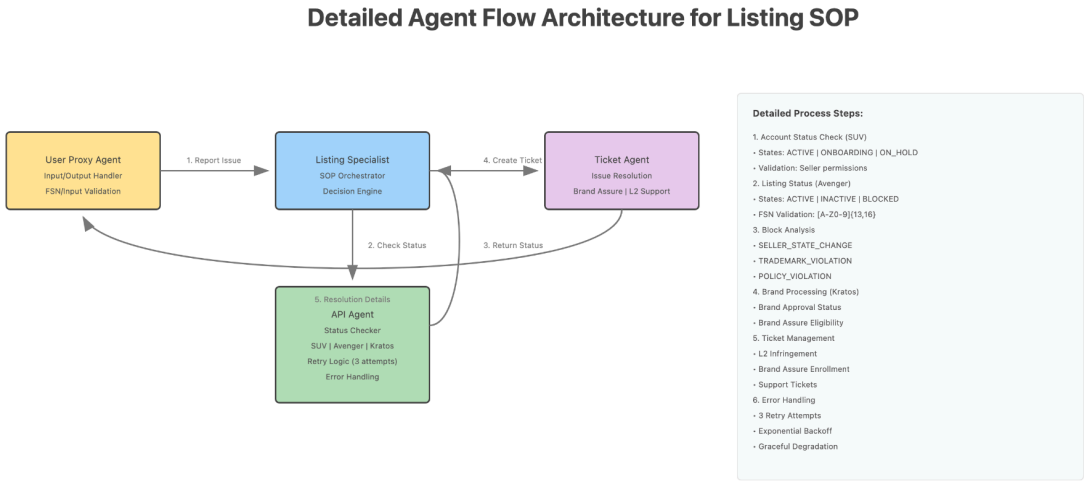
Before developing our LLM agents, we conducted extensive research to establish a strong theoretical foundation and evaluate existing frameworks. We explored some key concepts such as vision language models, agentic design, and multi agent architectures.

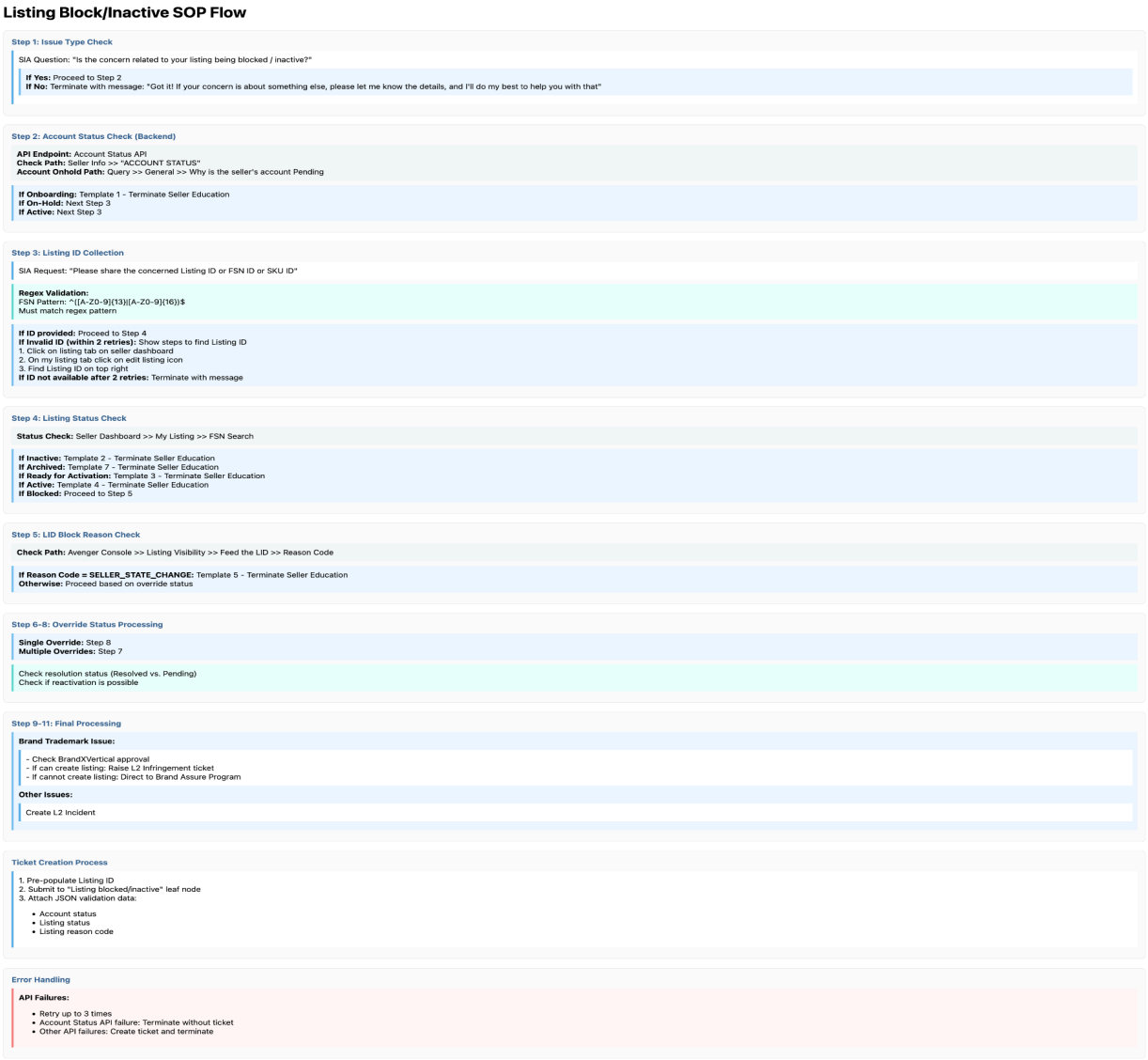
Among the resources we reviewed, Hugging Face's "A Dive into Vision-Language Models"[1] provided us with some valuable context on integrating multimodal learning into LLM workflows; DeepLearning AI's "AI Agentic Design Patterns with AutoGen"[2] course helped us understand how to build and customize multi-agent systems, allowing them to take on different roles and collaborate; Akira AI's "Microsoft AutoGen: Redefining Multi-Agent System Frameworks"[3] provided us some initial ideas in creating and configuring agents via Python; and Anthropic's "Building Effective Agents"[4] provided us with valuable ideas of how to sketch finite state machines. By synthesizing knowledge from these sources, we established a well

informed foundation that shaped our approach to designing and implementing a scalable, efficient multi-agent system.

c. Evaluation of Tools and Justification for Final Selection

During our project, we explored various tools to assist in designing and implementing our LLM agents. To structure our finite state machines, we first created a flowchart based on the steps outlined in the SOP before translating them into code. We used a drawing tool to visually map out agent interactions and state transitions, allowing us to clearly define how agents communicate and operate within the system. This visualization helped us identify potential gaps in the workflow and determine where additional agents might be needed to facilitate smoother transitions. By visualizing our design before implementation, we ensured a more structured and efficient development process.





Proceeding to the coding part, we initially considered implementing the finite state machine based on the Microsoft article we found, which outlined a structured methodology. However, this approach required the use of Azure services and extensive programming knowledge in C#, which posed a challenge since our team primarily worked with Python. Additionally, we encountered cross-platform compatibility issues, particularly with syncing between Mac and Windows systems, which hindered smooth collaboration and coordination

within the team. As a result, we decided to explore alternative solutions that would better align with our expertise and streamline our development process.

To address these challenges, we transitioned to VS Code and Python, which provided enhanced flexibility, seamless collaboration, and cross-platform compatibility. This shift simplified our development process by reducing setup complexities. Additionally, several articles we found on implementing agents using Python further guided and streamlined the setup process. With these improvements, we were able to work more efficiently.

2. Main Difficulties Encountered

a. SOP Functional Requirements and Objectives

During the implementation of the SOPs, we faced several challenges, particularly when attempting to connect to Flipkart's internal API. One minor issue we encountered was limited access to certain templates provided by Flipkart due to restrictions on our assigned Flipkart email addresses. Fortunately, this issue was resolved quickly. In the meantime, we leveraged our previous knowledge to design the finite state machine and wrote some basic functions for the agents. Despite the initial setbacks, these functions worked well and allowed us to continue progressing with the project while awaiting full access to the necessary materials.

One major technical difficulty we encountered was our inability to retrieve the account status or block reasons for user and listing IDs through Flipkart’s API call, which prevented us from accessing this data as originally intended. To maintain progress on our development, we decided to shift our approach. Rather than relying on the results from the API call, we implemented a hardcoded solution. Specifically, we created a rule that examined the starting or ending character of the provided user or listing IDs and assigned them predefined statuses. For

example, if the user ID started with “1”, we assigned the account status as “active,” and if the listing ID ended with “2”, we marked it as "blocked," bypassing the need for an API call. While this solution did not fully replicate the API based process, it offered a practical workaround that allowed us to meet the functional requirements and continue making progress with the project.

b. Integration in a Multi-Agent System

Aside from the technical difficulties, one of the major challenges we faced when integrating multiple agents into the system was that the SIA agent sometimes generated its own responses instead of passing the results directly from the assigned functions. To investigate this issue, we performed a series of tests on the functions, examining both regular and edge cases to pinpoint the source of the problem. For example, with the “get_listing_status” function, we passed in listing IDs that were designed to pass the regular expression check. We first created a function called “fetch_block_reason”, which randomly selected a block reason from a predefined list of possible values, such as “policy_violation”, “payment_issue”, “document_expired”, and “None”. We then tested the function with a sample listing ID, which successfully returned a dictionary with the correct status and a randomly selected block reason as shown in the example below: “{“status”: “blocked”, “message”: “Your listing is blocked due to document_expired.”, “block_reason”: “document_expired”}”. This result confirmed that the function itself was working as expected. However, it also indicated that the issue likely lay in the integration of the function within the agent system or in how the data was being passed between the agents, rather than in the function itself.

In response, we decided to modify the workflow so that the SIA agent would directly pass the results from the functions without any alterations or modifications. To ensure that this

change would work as expected, we added extensive debugging throughout the system to monitor the flow of data between the agents at each step.

The debugging code essentially printed messages at critical points to allow us to track the system's behavior and identify where things went wrong. For instance, before the SIA agent interpreted the result, we logged a message confirming that the FunctionExecutor had returned the expected output, signaling that SIA was about to process it. Additionally, we started each new chat session with a log statement to help us track when a new interaction began, ensuring that we could differentiate between old and new sessions. We also included debugging messages to display the order of agents in the system, which helped us understand the flow of execution and how the agents interacted with one another. Finally, we printed out SIA's system message, which contained key information about the agent's configuration and expectations.

To further address the issue of SIA generating its own responses, we customize the behavior of the group chat system in AutoGen by overwriting the built in communication methods. Specifically, we defined our own "select_speaker" function to ensure that responses were only generated by the intended function and not improvised by the agent itself. This modification allowed us to have better control over which agent should speak at each step in the interaction process. By explicitly directing the flow of conversation and specifying the conditions under which the SIA agent should relay function outputs rather than create its own messages, we were able to eliminate unexpected behavior. This approach ensured that all responses were strictly derived from the executed functions, leading to a more reliable and predictable system. This process helped us pinpoint where the breakdown occurred in the interaction between the agents. Once identified, we were able to make the necessary adjustments to ensure that the agents passed the data correctly and that the system functioned as intended.

c. Function Registration Consistency and Execution Correctness

Another major challenge was ensuring that functions were properly registered and recognized by AutoGen throughout different iterations. AutoGen has a built-in way for function registration, but we observed inconsistencies in how it handled function calls during execution, making it difficult to determine which functions were available at any given moment. To diagnose this issue, we closely examined how the function executor retrieved function names and mapped them to the registered functions, explicitly tracking which functions were recognized.

Our original attempt used a “FunctionExecutor” agent, which was designed to receive function calls and execute them accordingly. However, it lacked strict function call detection, relying on AutoGen’s internal function execution mechanism without explicitly verifying the format of function calls or identifying which functions were available at each step. These shortcomings led to the function executor failing to recognize or process function calls correctly. Additionally, it did not provide direct control over function execution. The system message in the function executor simply instructed the agent to execute functions when received, lacking an explicit method to extract function names and arguments systematically, which lead to unpredictable behavior when interacting with other agents.

To resolve these issues, we created a custom “FunctionExecutorAgent” class, which introduced explicit function call validation and controlled message processing. One key enhancement was overriding the replying method to directly check the last message from the SIA agent, extracting and executing the function call only if it matched the expected format. This prevented the agent from modifying function outputs. Additionally, we implemented strict function call detection using regex to ensure that only properly formatted function calls were

executed, reducing errors. We also enhanced function recognition by incorporating debugging and logging at key points, explicitly verifying both the sender and the structure of the function call, which addressed the registration inconsistencies in AutoGen.

d. Version Control and Synchronization Across Devices and Platforms

Last but not least, we also encountered challenges related to team collaboration, time management, and synchronizing efforts across different team members. These non-technical aspects were also crucial in ensuring the smooth progress of the project. To address these issues, we adopted a proactive approach by meeting both in person and online frequently. This allowed us to stay aligned and maintain open communication. Additionally, we established regular communication with the Flipkart team via Google Chat, which helped us address questions and issues outside of our regular weekly meetings.

In the initial stages of the code writing process, each of us worked independently on the same major parts of the SOP, which resulted in some redundancy in our work. We realized that this approach led to overlaps in effort and inefficiencies in progress. After some discussions, we decided to streamline our approach by selecting one person's code as the primary base for development, with the other two team members dedicating time to understanding and familiarizing themselves with that code. This allowed us to avoid redundant work, better synchronize our efforts, and ensure that we could move forward cohesively as a team.

3. Implementation Process and Learnings

a. Bridging the Gap Between Background Knowledge and Required Skills

While our previous coursework primarily focused on data science, with some exposure to software development, the skills and knowledge we acquired were largely aimed at data analysis

and engineering tasks. In contrast, the SOPs were heavily centered around software engineering. Although we possessed foundational software development skills from our data science background, the depth and breadth of Python programming required for this project proved to be a significant challenge.

Throughout the project, we frequently had to rely on online resources to bridge the gap between our existing knowledge and the new skills we needed to acquire. For example, designing the regular expression check for the provided IDs was a task that required us to dive deeper into string manipulation techniques. Additionally, we had to learn how to effectively implement debugging practices, such as adding critical print statements at various stages in the code to trace the system's behavior. These debugging techniques allowed us to monitor the flow of data and quickly identify where the system was deviating from expected results.

At times, we encountered roadblocks where our approaches led to dead ends. These instances required us to step back, reflect on our methods, and revisit our strategies after some time. It was a challenging yet valuable learning experience that pushed us to adapt and develop new problem solving and programming skills.

b. SOP Implementation and Error Handling

After extensive trials and iterations, we successfully implemented the functionalities outlined in the SOPs. In our system, the execution of these SOPs is seamlessly integrated through a combination of automated processes and robust error-handling strategies. The Autogen logic is specifically designed to automatically generate requests when certain conditions are met, such as detecting specific statuses from user ID or listing ID. In other scenarios, the system activates a retry mechanism to address potential network or system errors. This retry logic

ensures that each request is retried up to a predefined limit, effectively minimizing the chances of failure due to temporary issues.

When a valid function call operation is detected, the system efficiently dispatches the request to the appropriate function using the FunctionExecutor. For example, if the system needs to retrieve a listing status, it automatically triggers the "get_listing_status" function to gather the required information. This process is fully automated, ensuring accurate and timely data retrieval while eliminating the need for manual intervention.

The Coordinator agent plays a critical role in managing the flow of the system by orchestrating the sequence of actions based on the context of the conversation. It ensures that the appropriate components of the system are selected for each step, maintaining a structured flow and preventing errors. The system analyzes the input it receives and decides which process to activate next.

For example, if a valid function call is detected, the Coordinator routes the request to the FunctionExecutor to retrieve or update the relevant data. If a request encounters an issue, the Coordinator activates the retry mechanism. The retry count is tracked for each request to avoid exceeding the retry limit. The system is designed to handle both standard and exceptional cases, ensuring smooth operation under a variety of conditions. In cases where the retry mechanism does not resolve the issue, the Coordinator escalates the situation by generating a support ticket. This support ticket creation process ensures that unresolved issues, such as blocked listings or persistent errors, are quickly addressed by the appropriate team. By dynamically selecting the appropriate process or agent based on the context and input, the Coordinator ensures that each request is handled efficiently and in the correct sequence, thereby optimizing the overall performance of the system.

The integration of automated processes ensures that all steps, from request generation to error handling, are executed swiftly and without manual intervention. The combination of Autogen logic, retry mechanisms, and proactive error handling provides a robust framework that supports high availability and performance. By maintaining a comprehensive error handling strategy, the system promptly identifies and addresses issues, minimizing downtime and ensuring a smooth and reliable service. This approach not only adheres to the SOPs but also allows the system to adapt to unexpected situations, guaranteeing optimal performance even during unforeseen challenges.

c. Key Achievements and Future Opportunities

After completing the main and Streamlit Python files, we conducted thorough testing, covering all potential workflows. We achieved this by using various listing and brand approval request IDs, following the patterns outlined in the readme file. The results aligned with the expectations set by the SOPs, effectively cross-validating our implementation.

Furthermore, the system can also be applied to integrate seamlessly with Flipkart's customer systems, enabling a smooth flow of operations across platforms. As the system evolves, it can easily incorporate additional external data sources, broadening its ability to respond to a wider variety of inputs.

In the future, the system can be expanded to support additional chatbot functionalities, such as fraud detection, order tracking, after-sale customer support, and return processing. These enhancements will significantly improve the customer experience, offering a comprehensive solution that not only addresses immediate user needs but also adapts to emerging challenges and opportunities in the e-commerce landscape.

To summarize, the primary goal for this project was to explore the feasibility of developing an agentic process capable of navigating multiple SOP-driven workflows for seller issues. By leveraging the Autogen framework, our team demonstrated through this capstone project that such a process is indeed viable.

4. Additional Takeaways

As of the moment of compiling this learning document, most of the work for this project has already been finalized. However, there are still several important tasks to address and areas to consider as we move forward with refining the system and preparing for future applications.

a. Finalizing Version Control and Synchronization Methods

One key task that remains is establishing proper version control and progress syncing across the team. It is crucial to implement a reliable system for tracking changes, updates, and milestones to prevent duplication of efforts and ensure smooth collaboration. Finalizing version control will help the team capture, document, and manage all changes effectively throughout the development and deployment phases. Additionally, if our product is used either in whole or in part in the future, this comprehensive documentation will serve as a valuable reference, guiding future teams and users in understanding the evolution of the system and its functionalities.

b. Assessing Core Functionalities for Retention or Removal

Additionally, as we continue refining the system, it is important to evaluate and decide which functionalities should remain in the final implementation and which should be discarded. This evaluation will help streamline the system, removing any redundant or non-essential features. Conducting a thorough analysis of user needs and project goals will guide this

decision-making process, ensuring that the most relevant and impactful functionalities are retained.

c. Exploring Applicability for Related Topics and Future Research

Another important consideration is the potential applicability of the work done so far to similar topics or future research. The methodologies, frameworks, and technologies developed for this project can be adapted and applied to other domains or areas of research. It is important to explore these possibilities to maximize the impact and utility of the work. This examination could open up new opportunities for research and innovation in related fields.

d. Ensuring Data Integrity and Addressing Ethical Concerns

Last but not least, maintaining data integrity and addressing ethical concerns remain critical components of the project. Ensuring the accuracy, reliability, and security of the data used and processed by the system is important. Additionally, ethical considerations, such as privacy concerns, transparency in data handling, and fairness in automated decision making process, must be continuously assessed and addressed to ensure compliance with legal and ethical standards.

e. Future Communication Plans and Collaborative Engagement

Finally, ongoing communication and feedback are vital for continued success. Conducting surveys to gather insights from our sponsor will provide valuable feedback for improvements. Regular communication will ensure alignment with project goals, encourage collaboration, and address any concerns or challenges that arise during implementation.

By addressing these remaining tasks and considering these additional takeaways, we can ensure the continued success of the project and its long term impact.

Finally, we would like to express our most sincere thanks to Professor Hazen and the Flipkart Group for providing this invaluable externship opportunity. The guidance, feedback, and instructions provided throughout the process have been instrumental in shaping the project and ensuring its success. We greatly appreciate your support and the insights shared, which have enriched our learning experience and contributed to the overall growth of our team.

References:

[1] Dirik, A., & Paul, S. “A Dive into Vision-Language Models.” Hugging Face, Feb. 3, 2023, https://huggingface.co/blog/vision_language_pretraining.

[2] Wang, C., & Wu, Q. “AI Agentic Design Patterns with AutoGen” DeepLearning.AI, https://www.deeplearning.ai/short-courses/ai-agentic-design-patterns-with-autogen.

[3] Gill, J. K. “Microsoft AutoGen: Redefining Multi-Agent System Frameworks” Akira AI, Dec. 13, 2024, https://www.akira.ai/blog/microsoft-autogen-with-multi-agent-system.

[4] “Building effective agents” Anthropic, Dec. 19, 2024, https://www.anthropic.com/engineering/building-effective-agents.