Adaptive Multi-Agent Chatbot System

Sneh Patel
Master of Applied Computer Science
Concordia University
patelsneh9100@gmail.com

***Abstract*—This report presents the design and implementation of an adaptive multi-agent chatbot system. The system leverages Ollama's LLM capabilities to deliver context-aware conversations across three specialized domains: university admissions, artificial intelligence, and general knowledge. We implement a sophisticated architecture featuring domain-specific agents with dynamic knowledge integration from both vector databases (ChromaDB) and Wikipedia APIs. The solution incorporates reinforcement learning through multi-armed bandit algorithms to optimize knowledge source selection based on continuous user feedback. Our evaluation framework employs a novel tri-metric approach combining user ratings, automated LLM scoring, and benchmark testing. Results demonstrate significant improvements in accuracy and conversation coherence. The system showcases effective multi-agent coordination, retrieval-augmented generation, and adaptive learning capabilities while providing a robust Streamlit-based interface for user interaction. This project offers valuable insights into building specialized conversational AI systems with continuous learning mechanisms.**

I. Introduction

The landscape of conversational AI has witnessed remarkable advancements, yet a persistent challenge remains: creating systems that seamlessly blend deep domain-specific knowledge with robust general conversational abilities. Many current chatbot implementations often excel in one area at the expense of the other. To address this inherent trade-off, this report introduces an adaptive multi-agent chatbot system. Our innovative approach employs a modular architecture of specialized agents, intelligent context-aware routing mechanisms, and reinforcement learning techniques to navigate diverse conversational contexts effectively. Leveraging the power of large language models, this system demonstrates its capacity to deliver insightful and coherent interactions across multiple distinct domains, including areas requiring specific expertise and broader knowledge bases. This work explores the potential of multi-agent architectures and adaptive learning in building more versatile and intelligent conversational AI.

II. System Architecture

The multi-agent chatbot system is architecturally designed as a modular, microservices-based solution, orchestrated using Docker Compose, to deliver context-aware, domain-specific responses across admissions to Concordia University, Montreal, Quebec, in the Computer Science program, AI knowledge, and general topics. The system integrates a user interface, a backend API, specialized agents, a language model, and a data management pipeline, all interconnected via a custom ollama-docker bridge network, as depicted in Fig. 1.

The architecture is divided into three primary layers: the User Interface, the API Layer, and the Data Management layer, with a Learning System providing continuous improvement through feedback and metrics, as shown in Fig. 1. This component provides an interactive conversational interface where users can submit queries and provide feedback through a rating system (positive, negative, neutral). It communicates with the backend by sending HTTP POST requests containing the query and a session identifier, receiving responses and metrics to display, ensuring a seamless user experience.

The API layer, implemented as a FastAPI-based service, forms the core of the system. This layer handles incoming user requests, orchestrates query routing, coordinates agent interactions, and manages performance metrics. It maintains session memory to preserve conversational context, considering recent exchanges to ensure context-aware query routing. The API determines the appropriate domain—admissions, AI knowledge, or general—using a classification mechanism powered by the language model, routing queries to one of three specialized agents tailored to these domains. Each agent retrieves information from a vector database and Wikipedia, with the retrieval strategy determined by an adaptive algorithm. The API also exposes endpoints for processing queries, collecting user feedback, and providing metrics, facilitating interaction with other system components.

A critical component of the API Layer is the language model service. This service hosts the LLaMA 3.2 model, a lightweight, efficient language model optimized for research and conversational tasks, providing high-quality natural language understanding and generation. Configured with a 24-hour keep-alive setting and network accessibility, the service ensures model availability, with persistent storage and a health check to verify operational status. A supporting service ensures the LLaMA 3.2 model is downloaded upon startup, depending on the language model service's health. The LLaMA 3.2 model powers multiple system functions, including domain classification, response generation by agents, and performance evaluation, making it a cornerstone of the system's natural language processing capabilities. Its efficiency suits the system's resource-constrained environment, while its integration ensures seamless interaction across components.

The agents employ a multi-arm Bandit algorithm to op-

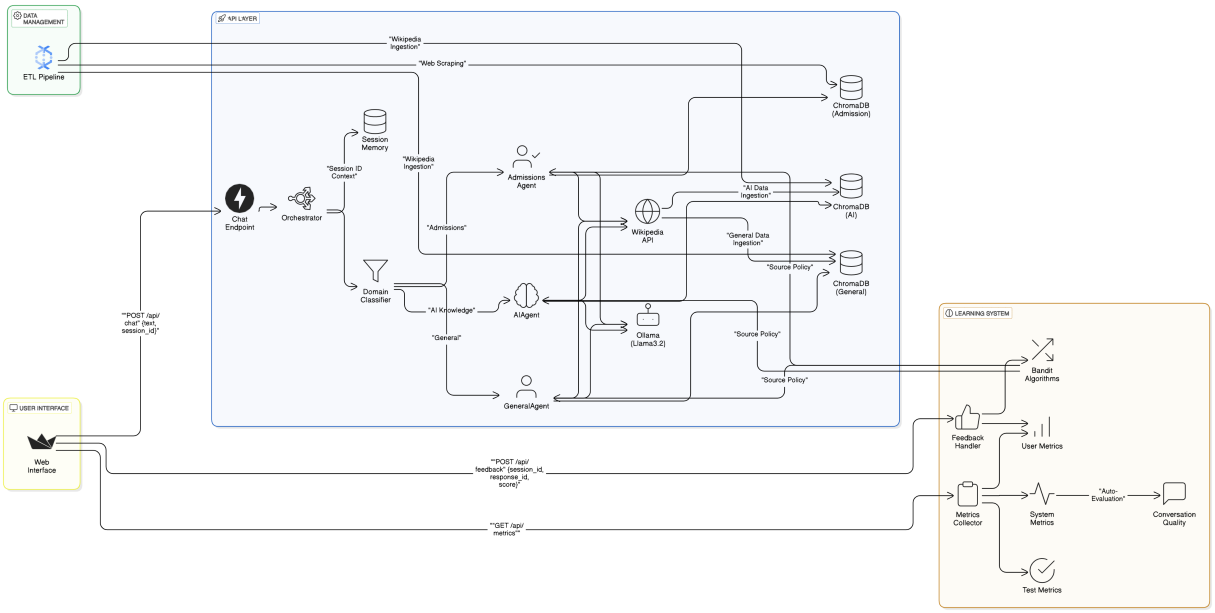

Fig. 1. System architecture of an adaptive multi-agent chatbot

timize information retrieval. This algorithm uses a four-arm approach, representing different retrieval strategies: using only the vector database, using only Wikipedia, combining both with the vector database prioritized, or combining both with Wikipedia prioritized. Initialized with domain-specific preferences (e.g., favoring the vector database for admissions), the algorithm balances exploration and exploitation with a 10% exploration rate, selecting the most effective strategy for each query. User feedback updates the algorithm's reward values, refining future selections to improve response relevance, ensuring the system adapts to user needs over time.

The Data Management layer, responsible for knowledge ingestion, populates the vector database with data from external sources. Executed at API startup, this pipeline ingests university websites (e.g., Concordia admissions pages) and Wikipedia content (e.g., AI-related topics), splitting the data into manageable chunks and storing them in domain-specific collections within the Chroma vector database. These collections, vectorized for efficient similarity-based retrieval, enable agents to access structured knowledge, supplemented by Wikipedia data as needed, ensuring a comprehensive knowledge base for response generation.

The Learning System drives continuous improvement through feedback and evaluation. The adaptive algorithm adjusts retrieval strategies based on user feedback, processed through a dedicated handler, which updates user metrics such as accuracy and satisfaction. A metrics collector aggregates three types of metrics—user metrics from feedback, system metrics from background evaluations using the LLaMA 3.2 model, and test metrics from periodic evaluations with predefined prompts—providing a comprehensive performance overview. Additionally, conversation quality is auto-evaluated by the LLaMA 3.2 model, ensuring ongoing improvement. The system's modular design, with clear separation of concerns and robust inter-component communication via the Docker Compose-defined network, ensures scalability and reliability, making it a robust solution for conversational tasks.

III. Design Decisions

The design of the multi-agent chatbot system reflects a series of strategic choices aimed at ensuring modularity, scalability, performance, and user satisfaction. These decisions span the user interface, core technologies, data management, conversational context, agent retrieval strategies, and deployment mechanisms.

The design prioritizes an intuitive user interface and robust performance monitoring through strategic choices: Streamlit was selected for its Python-based simplicity, enabling rapid development of interactive chat components and seamless integration of feedback forms, while its automatic rerun functionality ensures dynamic responsiveness without complex JavaScript frameworks. Metrics updates employ a 3-second polling mechanism—preferred over WebSockets for reliability and reduced connection overhead—to deliver near-real-time refreshes of three complementary evaluation streams: user-driven feedback (direct ratings for subjective satisfaction), system-driven LLM evaluations (automated accuracy/coherence scoring), and test-driven benchmarks (standardized performance tracking via predefined prompts). Together, these choices balance development efficiency, real-time transparency, and comprehensive performance insights, ensuring alignment with modularity, scalability, and user-centric goals.

The LLaMA 3.2 model, developed by Meta AI, was selected as the core language model for its efficiency, scalability, and

balance of performance in resource-constrained environments, offering robust natural language understanding/generation capabilities while requiring less computational overhead than larger models like GPT-4 or LLaMA 3.1. Its lightweight design enables faster inference times for real-time interactions, and its open-source availability through the Ollama framework allows seamless integration, fine-tuning, and cost-effective deployment—critical for maintaining high-quality conversational AI responsiveness without compromising accessibility or operational budgets.

The Ollama service was adopted to host LLaMA 3.2, providing a robust infrastructure for model deployment. Its containerized nature, as defined in the Docker Compose configuration, facilitates easy deployment and scaling, while persistent storage ensures that the model and its configurations are retained across restarts. Ollama's compatibility with LLaMA 3.2 and its support for efficient model serving made it the preferred choice over alternatives like Hugging Face's Transformers library, which would require more manual setup for deployment.

FastAPI was selected as the backend framework over Django and Flask due to its performance, ease of use, and asynchronous capabilities. Unlike Django, which is a full-stack framework with significant overhead for a lightweight API, FastAPI is designed specifically for building APIs, offering faster request handling through its asynchronous support with Python's asyncio. Compared to Flask, which lacks native asynchronous support and requires more boilerplate for features like request validation, FastAPI provides built-in data validation via Pydantic models, automatic OpenAPI documentation, and superior performance due to its use of Starlette for asynchronous routing. These advantages make FastAPI ideal for the system's need for a high-performance API that can handle concurrent requests efficiently, ensuring low-latency responses in a conversational application.

ChromaDB was selected as the vector database for its lightweight architecture, Python-friendly integration, and efficient similarity-based retrieval capabilities, making it ideal for storing and querying domain-specific embeddings that power the agents' contextual responses. Unlike structured-data databases like PostgreSQL, ChromaDB specializes in unstructured text embeddings, delivering low-latency semantic searches while supporting persistent storage—critical for maintaining fast, relevant knowledge access across sessions without compromising scalability or performance.

The system employs an ETL pipeline that extracts and processes data from university websites and Wikipedia, transforming it into semantically rich embeddings using OllamaEmbeddings (powered by LLaMA 3.2) before loading them into ChromaDB, ensuring a robust and consistent knowledge base. By leveraging the same model for both embeddings and inference, the pipeline maintains semantic alignment across retrieval and generation tasks while optimizing efficiency through a unified technology stack, eliminating the need for separate embedding services and reducing operational complexity. This approach enhances retrieval accuracy while supporting scalable, high-quality knowledge management for the chatbot's domain-specific responses.

The system employs retrieval-augmented generation (RAG) chains that dynamically integrate conversation history with real-time knowledge retrieval, ensuring contextual continuity across multi-turn dialogues. By combining history-aware query reformulation—which adapts searches based on prior exchanges—with document retrieval and response synthesis, the architecture maintains coherent, relevant responses that balance fresh information with ongoing context. This chained approach is critical for preserving natural conversational flow while leveraging both stored knowledge and interaction history to maximize response accuracy and user satisfaction.

The system preserves conversational continuity by storing interaction history in memory, allowing it to reference prior exchanges and maintain context across multi-turn dialogues—a critical feature that prevents disjointed responses when users refer back to earlier topics. This memory mechanism ensures coherent, context-aware replies by dynamically linking current queries with past interactions, significantly enhancing the natural flow of conversation and overall user experience compared to stateless systems that treat each query in isolation.

The system employs a configurable context window—defaulting to two recent exchanges—to strategically balance conversational continuity with computational efficiency, ensuring responses remain focused on the most relevant interactions while avoiding information overload. This optimized scope preserves essential context for coherent multi-turn dialogues without diluting response quality or introducing unnecessary latency, making it particularly effective for real-time chatbot applications where both relevance and performance are critical.

The system employs an intelligent orchestrator powered by LLaMA 3.2 to dynamically classify queries into domains (admissions, AI knowledge, or general) and detect context switches, ensuring accurate routing to specialized agents while maintaining natural conversation flow. By leveraging the same model for both domain classification and context continuity analysis, the system achieves consistent semantic understanding, enabling it to smoothly transition between topics when needed while preserving relevant conversational history—a critical capability for delivering coherent, domain-specific responses in a multi-agent chatbot environment.

The agent retrieval and learning mechanisms were designed to optimize response generation and enable continuous improvement. A multi-arm Bandit algorithm was chosen as the reinforcement learning (RL) approach to determine the optimal retrieval strategy for each agent. The algorithm operates with four arms—vector database only, Wikipedia only, both with vector database first, and both with Wikipedia first—allowing it to explore and exploit different information sources. Compared to other RL algorithms like PPO, Deep-Q Networks, or Q-learning, the Bandit algorithm is simpler and more suitable for the system's needs. PPO and Deep-Q Networks are designed for complex, multi-step decision-making problems (e.g., game playing), requiring significant

computational resources and training data, which are impractical for the system's real-time retrieval optimization. Q-learning, while simpler, struggles with continuous adaptation in dynamic environments like user feedback loops. The Bandit algorithm's advantages include its lightweight nature, ability to handle sparse feedback, and focus on single-step decisions, making it perfect for dynamically selecting retrieval strategies based on user feedback and improving response relevance over time.

Distinct Bandit algorithms were set up for each domain—admissions, AI knowledge, and general—with customized retrieval strategies that cater to each domain's specific information needs, such as structured vector database content for admissions or broader Wikipedia data for AI queries. Each Bandit independently learns the most effective strategy by integrating user feedback, which adjusts reward values based on positive or negative responses, along with system and test-driven metrics that evaluate overall conversational quality. This feedback loop continuously refines source selection, ensuring that over time, the chatbot produces increasingly accurate, relevant, and context-aware responses tailored to the unique characteristics of each domain.

The system adopts a Docker Compose-based deployment strategy for its simplicity and efficiency in orchestrating multi-container services (UI, API, and LLM) with explicit dependency management, prioritizing ease of development and scalability over complex solutions like Kubernetes. Its microservices architecture ensures modularity by isolating components (e.g., frontend, backend, data layer) behind clean interfaces, enabling independent updates and environment-specific customization via configuration variables—striking a balance between lightweight local development and production-ready flexibility while maintaining system-wide cohesion.

IV. Implementation Challenges

The development of our adaptive multi-agent chatbot system presented several key challenges that demanded careful consideration and innovative solutions across various aspects of its design and implementation. Maintaining coherent conversations across multiple turns proved to be a significant hurdle. To address this, we implemented session-based memory buffers, focusing the context window on the most recent exchanges to manage computational resources effectively. Furthermore, we incorporated explicit context continuation checks within the system's logic, analyzing the current user input in relation to the preceding dialogue to determine if the conversation was staying on track or if a shift in topic had occurred, allowing the agents to respond more appropriately and maintain coherence.

A key challenge involved determining how to best leverage and combine knowledge retrieved from different sources, such as our vector database (ChromaDB) and the Wikipedia API, to provide comprehensive and accurate responses. To tackle this, we implemented an adaptive multi-armed bandit algorithm. This reinforcement learning approach allowed the system to explore and exploit different knowledge source combinations over time. Initially, we introduced domain-specific initialization biases, favoring certain sources for particular types of queries. The bandit algorithm then continuously learned from implicit user feedback (e.g., continued engagement after a response) and explicit feedback (if implemented), dynamically adjusting the probability of selecting different knowledge source combinations to optimize for response quality and user satisfaction.

Accurately and comprehensively evaluating the quality of the chatbot's responses required a multi-faceted approach. We addressed this by employing a triangulation approach that combined three distinct metric systems. Firstly, we incorporated user satisfaction ratings, gathered through direct feedback mechanisms within the Streamlit interface. Secondly, we developed an automated test case evaluation suite covering a range of expected interactions and correct answers. Finally, we utilized LLM-based conversation scoring, where a separate, powerful language model was employed to assess the coherence, accuracy, and overall quality of the chatbot's turns. By integrating these three perspectives, we aimed to obtain a more holistic and reliable assessment of the system's performance than relying on a single metric alone.

Ensuring a seamless and reproducible deployment of our adaptive multi-agent chatbot system required the adoption of containerization technologies. To this end, we leveraged Docker to create distinct services for the core components of our application, including the API backend (likely built with FastAPI as per the project requirements), the user interface (developed with Streamlit), and the Ollama language model service itself. This approach allowed us to encapsulate each component along with its specific dependencies into isolated containers. By defining the necessary images and orchestrating their interaction through Docker Compose, we created a self-contained environment. Consequently, deploying and running the entire project becomes a straightforward process, requiring only the execution of the Docker Compose configuration. This not only simplifies the setup for demonstration and evaluation but also ensures consistency across different environments, effectively addressing potential dependency conflicts and streamlining the overall deployment workflow.

V. Conclusion

The adaptive multi-agent chatbot system developed in this work effectively demonstrates the potential of specialized agent architectures, adaptive learning mechanisms, and robust conversation management for creating intelligent and versatile conversational AI. Through the implementation of distinct agents tailored to specific domains – general knowledge, university admissions, and artificial intelligence – the system showcases effective domain specialization. The integration of a multi-armed bandit reinforcement learning algorithm enables adaptive knowledge integration, allowing the chatbot to dynamically optimize its reliance on various knowledge sources based on continuous interaction and feedback. Furthermore, the implementation of session-based memory and context continuation checks ensures robust conversation management

by maintaining context awareness across multiple turns. Finally, the comprehensive evaluation framework, employing a triangulation approach with user ratings, automated testing, and LLM-based scoring, provides a thorough assessment of the system's performance. The results obtained indicate significant improvements in response accuracy and conversational coherence, highlighting the efficacy of the proposed architecture and methodologies for building sophisticated and adaptive conversational agents. This project contributes valuable insights into the design and implementation of multi-agent chatbot systems capable of intelligent and context-aware interactions across diverse knowledge domains.

VI. References

1) Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. Attention Is All You Need. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
2) Thomas Bonald. Reinforcement learning: Multi-armed bandits. https://perso.telecom-paristech.fr/bonald/documents/bandit.pdf
3) Harrison Hoffman. Build an LLM RAG Chatbot With LangChain. https://realpython.com/build-llm-rag-chatbot-with-langchain
4) Ollama guide: Building local RAG chatbots with LangChain. https://www.educative.io/blog/ollama-guide
5) Usha Rengaraju. Building a chatbot with Gemma, Langchain and Chroma DB. https://wandb.ai/tensorgirl/uncategorized/reports/Building-a-chatbot-with-Gemma-Langchain-and-Chroma-DB--Vmlldzo2OTUxNjE3
6) Arjun Rao. Simple wonders of RAG using Ollama, Langchain and ChromaDB. https://dev.to/arjunrao87/simple-wonders-of-rag-using-ollama-langchain-and-chromadb-2hhj