ORCA: Retrieval-powered AI Conversations

Gaurav Ajit Patil[1], Vivek Suresh Chaudhari[2], Rakesh Dnyaneshwar Patil[3],
Harsh Ramakant Neve[4], K. D. Deore[5]
[1,2,3,4,5] *CSE (Data Science) Department R. C. Patel Institute of Technology, Shirpur, India*

***Abstract*—Conversational agents have evolved significantly with advancements in artificial intelligence and natural language processing, yet many chatbot development tools still require programming expertise. This creates barriers for educators, small organizations, and non-technical users seeking to harness conversational AI. This paper presents ORCA, a no-code platform that enables users to create intelligent, retrieval-augmented chatbots from diverse data sources including PDFs, URLs, JSON/CSV files, images, and hardcopy documents. ORCA employs a hybrid architecture combining a monolithic backend with microservices for high-compute tasks such as OCR and vector-based semantic search. The system integrates Groq's high-speed llm inference engine and DataStax AstraDB for scalable embedding manage- ment.**

***Index Terms*—No-Code, AI, Chatbot, Conversational AI, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Natural Language Processing (NLP), User-Centric Design**

I. INTRODUCTION

In recent years, conversational agents, commonly referred to as chatbots, have gained significant attention across industries such as education, healthcare, e-commerce, and customer service. These systems provide interactive, real-time communication between users and organizations, reducing workload, improving efficiency, and offering personalized experiences. With the rise of artificial intelligence (AI) and natural language processing (NLP), chatbots have evolved from simple rule- based scripts to intelligent systems capable of understanding context, intent, and user preferences. Despite this progress, many existing chatbot platforms remain challenging for non- technical users. Solutions such as Dialogflow, Microsoft Bot Framework, and Rasa offer powerful capabilities but often require programming expertise, steep learning curves, or ex- tensive configuration. This creates a barrier for small organizations, educators, or individuals who wish to integrate conversational AI into their workflows but lack the necessary technical background. Furthermore, existing platforms may restrict customization, limit deployment flexibility, or impose

Identify applicable funding agency here. If none, delete this.vendor lock-in, reducing accessibility and scalability. To ad- dress these challenges, we introduce ORCA, a user-centric platform that enables the development and deployment of custom chatbots without requiring deep technical expertise. ORCA provides an intuitive, low-code/No-code interface that allows users to design conversational flows, and integrate chat- bots seamlessly across multiple channels. By balancing sim- plicity and technical depth, ORCA empowers both developers and non-developers to create tailored conversational agents. This paper presents the design, implementation, and evaluation of the ORCA platform. We first review existing solutions and highlight the gap in accessible yet customizable chatbot development environments. We then describe ORCA's system architecture, which integrates natural language understanding (NLU), modular conversation management, and cloud-native deployment strategies. Following this, we discuss platform features, including a drag-and-drop builder, and omnichannel deployment support. Finally, we evaluate the system's perfor- mance through case studies and user testing, demonstrating ORCA's potential in real-world applications. By lowering the barrier to entry and enabling wider participation in chatbot development, ORCA contributes to the democratization of conversational AI. This work aims to bridge the gap between sophisticated AI models and end-user accessibility, making chatbot technology more inclusive, adaptable, and impactful.

II. LITERATURE REVIEW

The development of conversational agents has progressed significantly over the last two decades, driven by advance- ments in natural language processing (NLP), machine learning, and cloud technologies. Early chatbot systems such as ELIZA (Weizenbaum, 1966) and ALICE (Wallace, 2009) relied heav- ily on rule-based patterns and lacked contextual understanding. With the rise of machine learning and transformer-based ar- chitectures, LLM-powered chatbots achieved improved natural language comprehension and intent recognition. Retrieval-augmented generation (RAG) has become a dominant ap- proach for improving accuracy and reducing hallucinations.

Arslan et al. (2024) examined the strengths and limitations of RAG pipelines and highlighted the importance of efficient vec- tor search and context chunking. Vector database performance plays a critical role in retrieval accuracy. Pan et al. (2024) emphasized challenges such as high-dimensional indexing, approximate nearest-neighbor (ANN) search cost, and hybrid query filtering. ORCA incorporates these insights by using DataStax AstraDB, which provides scalable vector similarity search using advanced indexing algorithms such as HNSW. Koc¸ et al. (2025) explored the role of AI-assisted no-code platforms in accelerating software development. Their findings align with ORCA's vision of enabling non-technical users to build functional AI systems without programming overhead. Despite advancements, a recurring gap in the literature centers on balancing usability and flexibility. Existing systems either provide advanced customization at the cost of complexity or offer simplicity with limited extensibility. ORCA addresses this gap by combining a low-code interface with robust back- end intelligence.

III. SYSTEM DESIGN AND ARCHITECTURE

The architecture of ORCA has been designed to balance us- ability, scalability, and flexibility, ensuring that both technical and non-technical users can build and deploy conversational agents efficiently. The system adopts a modular, cloud-native architecture that separates frontend interaction and backend processing while maintaining smooth integration across all layers.

A. Orca Hybrid Architecture

The ORCA platform adopts a hybrid architecture that com- bines the simplicity of a monolithic backend with a compu- tationally intensive OCR service deployed as an independent microservice. This architectural split allows ORCA to retain ease of development and maintainability while isolating heavy processing workloads for improved scalability and perfor- mance. ORCA's choice of DataStax AstraDB as its vector store is consistent with current best practices in vector database management. According to Pan et al. (2024), modern Vector Database Management Systems (VDBMSs) must efficiently manage billions of vectors and support millisecond-level query latency while preserving high retrieval accuracy. AstraDB's distributed and horizontally scalable architecture addresses these performance demands, making it well-suited for large- scale semantic search applications such as ORCA. Addition- ally, ORCA's query engine incorporates similarity projection operations, which Pan et al. (2024) describe as fundamental operators that "project each vector in the collection onto its similarity score." Implementing these operators enables ORCA to rank retrieved documents based on semantic relevance, improving the quality of context supplied to the chatbot. Components of ORCA's Hybrid Architecture:

1. User / Mobile Client The mobile client represents the end user interacting with the ORCA platform. It communicates with the main application container through REST API end- points and integrates with Firebase for secure authentication.

The client also manages locally stored metadata used for maintaining application state and file-related information.

2. Locally Stored Metadata The system stores certain metadata such as file descriptors, upload status, or user preferences directly on the user's device. This metadata is periodically synchronized with the main application container to support faster access and reduce backend load.
3. Firebase Firebase functions as the platform's identity and lightweight data management layer. • Authentication: Provides secure user login and session handling. • Database (Firestore/Realtime Database): Stores non-vector data such as user

profiles, configuration settings, or lightweight application records.

4. Main Application Container (Deployed on Cloud Run) The core backend application is deployed as a serverless Docker container on Google Cloud Run, enabling automatic scaling based on user demand. It exposes multiple API end-points:
- /Upload: Uploads files or documents to the system.
- /Delete: Removes user-stored data.
- /Search: Executes vector-based search queries through the Query Engine.
- /owst: Performs organizational website search (OWST).
- /webupload: Uploads webpage content for processing. Key services within this container include:
- Data upload and delete services
- Query Engine
- Organizational Website Search Tool (OWST)
- Webpage data ingestion service
5. DataStax AstraDB (Vector Store) AstraDB acts as the primary vector database, storing high-dimensional embed- dings and supporting similarity-based retrieval for Retrieval-Augmented Generation (RAG). Its distributed architecture supports horizontal scaling and high-throughput vector search, aligning with requirements described in vector database liter-ature.
6. Groq Inference Engine (llama-4-maverick-17b-128e- instruct) The Groq inference accelerator hosts the llama-4- maverick-17b-128e-instruct LLM. The Query Engine sends retrieved context to this model, enabling fast, low-latency generation of responses. Groq hardware is optimized for high- speed inference, making it suitable for real-time conversational systems.
7. Temporary File Storage Temporary storage is used to hold intermediate files uploaded by users during processing. The main application container interacts with this location before final data is processed, vectorized, or transferred to persistent storage systems.
8. OCR Service Container (Deployed on Render) dedicated microservice performs Optical Character Recognition (OCR).
- /ocr: Endpoint used by the main application container to submit images or documents and receive extracted machine-readable text. This container is deployed sep- arately on Render to isolate computationally intensive OCR workloads from the main application logic.
9. Cloud Run (Hosting for Main Backend) Google Cloud Run provides the serverless hosting environment for the main application container. It offers automatic scaling, stateless execution, and HTTP-triggered deployments.
10. Render (Hosting for OCR Service) The OCR microser- vice is hosted on Render as a Docker-deployed container, ensuring dedicated resources for text extraction tasks and preventing OCR load from affecting main application perfor-mance.

B. Owst Architecture

The core purpose of this architecture is to overcome a limitation of standard Retrieval-Augmented Generation (RAG) by dynamically fetching the most relevant internal web page link instead of trying to cram vast, complex documents into a single LLM context window and return the filtered UI to client's web view component which simplifies and automates the process and user doesn't need to crawl through the website in search of required content. Components of ORCA Hybrid Architecture:

1. Input/Output Components:
- User Query: The process begins when the user submits a query or search term. This serves as the primary input that initiates the entire OWST retrieval and reasoning pipeline.
- System Response: The final output consists of a synthe- sized response generated by the system. This may be an LLM-generated answer or the presentation of selected web content, depending on relevance and user needs.

2. System/Process Components:

a. Setup and Initial Search
- Website Configuration: This module defines search pa- rameters such as target domains, filtering rules, and preferred search engine settings. It provides configuration metadata that guides the retrieval process.
- Google JSON API: The API receives the user query and Website Configuration parameters to

perform an external web search. It returns structured JSON data containing ranked search results.
- Search Results: These results represent the raw output of the Google JSON API, typically consisting of URLs, snippets, and metadata for the top retrieved webpages.

b. Information Selection and Augmentation
- User Query + Top-K Results: The system selects the top-ranked K results from the initial Search Results and combines them with the original user query. This enriched context is used both for LLM reasoning and for selecting a single best result.
- Large Language Model (LLM): A generative language model processes the augmented input (user query + Top- K documents) to synthesize an accurate, coherent, and contextually informed natural language response.
- Best Result: From the Top-K set, the system identifies the single most relevant result. This is surfaced to the user in cases where direct content inspection is necessary or where LLM output alone is insufficient.

c. Direct Web Content Flow
- Web Scraper: The scraper retrieves the full webpage content corresponding to the Best Result URL. This may include HTML, text, images, and embedded data.
- UI Filter: The extracted content undergoes cleansing and formatting. Noise such as advertisements, navigation bars, or irrelevant sections is removed to ensure a clean and user-friendly presentation.
- WebView: The filtered webpage is rendered in an embed- ded WebView component, enabling the user to view and interact with the original website content directly within the application.

IV. IMPLEMENTATION

The implementation of ORCA was carried out in modular phases to ensure seamless integration of core services, LLM inference, Vector Store Database, Cloud services and user interfaces. Each stage was designed to align with the hybrid architecture, where lightweight services remain bound within a monolithic backend, and computationally heavy tasks are executed through independent deployed services.

A. Technology Stack and Architecture Deci- Sions

ORCA's implementation leverages modern cloud-native technologies to achieve scalability, maintainability, and rapid development. The system is built on three primary layers:

a. Client Layer
- Frontend Framework: React Native for cross-platform mobile development (iOS/Android).
- State Management: Local state with React hooks; meta- data persistence via AsyncStorage.
- Authentication: Firebase Authentication for secure user management.

b. Application Layer
- Backend Framework: FastAPI (Python) for high- performance REST API.
- Deployment: Google Cloud Run for serverless container deployment
- API Gateway: Custom routing service for endpoint man- agement.
- Rationale: FastAPI offers async support crucial for han- dling concurrent LLM inference requests.

c. Data and Intelligence Layer
- Vector Database: DataStax AstraDB for embedding stor- age and similarity search
- LLM Inference: Groq inference engine with llama-4- maverick-17b-128e-instruct
- OCR Service: Custom microservice deployed on Render (Docker)
- File Storage: Cloud Storage for temporary document processing.

B. Key Implementation Challenges and Solu- Tions

a. Challenge: Efficient Document Processing Pipeline

Problem: Supporting six input formats (PDF, URL, JSON/CSV, images, hardcopy via OCR, webpage) required heterogeneous processing logic with varying computational requirements.

Solution: Implemented a plugin architecture where each document type has a dedicated processing logics:
- PDF Processing: Configurable extraction
- Web Scraper: Recursive crawling with depth limits and content filtering

- OCR Pipeline: Separate microservice to prevent blocking main application
- Structured Data: CSV/JSON parsers

b. Challenge: Context Window Management
Problem: Standard RAG approaches struggle when relevant context spans multiple documents or exceeds LLM context windows (128K tokens for llama-4-maverick).

Solution: Developed OWST (Organizational Website Search Tool) that:
1. Uses Google JSON API to search within organizational domains
2. Retrieves top-K results (typically K=5)
3. Combines query + results as LLM input for synthesis
4. Scrapes best result and applies UI filtering for direct viewing
5. Presents both synthesized answer and filtered webpage in WebView

Technical Innovation: Unlike traditional RAG that retrieves document chunks, OWST retrieves links to full documents and lets users navigate context-rich sources. This overcomes the fundamental trade-off between context completeness and token limits.
Implementation Detail: UI filtering removes distracting el- ements (ads, navigation, footers) using CSS selectors and heuristics.

c. Challenge: Dual-Agent Query Routing
Problem: Users sometimes ask questions unrelated to up- loaded documents (e.g.," What is the weather?" in a product documentation chatbot).
Solution: Implemented dual-agent architecture:
- Primary Agent: RAG-powered agent using vector simi- larity search on user's knowledge base
- Fallback Agent: General-purpose LLM for out-of-context queries
- Router: Classifies query intent using embedding similar- ity threshold

C. Deployment Architecture
a. Production Environment:
Main Application: Google Cloud Run
- Auto-scaling: 0 to 100 instances
- Per-instance resources: 2 vCPU, 4GB RAM
- Request timeout: 300s

OCR Service: Render Docker deployment
- Fixed instance: 2 vCPU, 4GB RAM (cost optimization vs auto-scaling)
- Separate deployment prevents OCR spikes from affecting main service

b. Data Persistence:
- Vector embeddings: DataStax AstraDB (cloud-managed)
- User metadata: Firebase Firestore
- Temporary files: Cloud Storage with 24-hour TTL
- Authentication tokens: Firebase Auth

V. EVALUATION

To validate ORCA's effectiveness in real-world scenarios and assess its accessibility for non-technical users, we de- ployed the platform across four distinct application domains. Each deployment tested different aspects of ORCA's capa- bilities, including document processing, retrieval accuracy, domain adaptation, and user experience. This section presents the implementation details, performance metrics, and user feedback from these case studies.
Our evaluation approach combined quantitative system per- formance metrics with qualitative user experience assessment across four diverse use cases:
1. Legal Advisory System - Testing document knowledge retrieval
2. Academic Question-Answering System - Evaluating structured data handling
3. College Website Chatbot - Assessing web content inte- gration via OWST
4. Automated Candidate-Expert Matching System - Testing complex query routing and dual-agent architecture

For each deployment, we measured:
- System Performance Metrics: Response latency, through- put, retrieval accuracy
- User Experience Metrics: Development time, ease of configuration, user satisfaction
- Functional Effectiveness: Task completion rate, answer accuracy, user engagement

VI. CONCLUSION

The development of ORCA demonstrates how conversa- tional AI platforms can be made more accessible, scalable, and customizable for both

technical and non-technical users. By combining a monolithic core for efficient backend orchestra- tion with microservice-based modules for compute-intensive tasks such as OCR and LLM inference, ORCA achieves a balanced hybrid architecture that supports performance and flexibility simultaneously. Its low-code environment allows users to design, train, and deploy chatbots with minimal effort, bridging the gap between traditional chatbot platforms and modern AI-driven assistants. Evaluation results indicate that ORCA provides a reliable, responsive, and user-friendly experience while maintaining modularity and cloud porta- bility through containerized deployment. The hybrid design ensures that lightweight services remain efficient, and heav- ier components can scale independently, making the system
robust under varying workloads. The integration of real-time knowledge retrieval and adaptive learning mechanisms will further strengthen ORCA's ability to serve as a dynamic and intelligent conversational platform.

REFERENCES

[1] M. Arslan, H. Ghanem, S. Munawar, and C. Cruz, "A survey on RAG with LLMs," Procedia Computer Science, vol. 246, pp. 3781–3790, 2024.

[2] O. Koc¸, ˙I. Yu¨cedag˘, and U¨ . S¸ entu¨rk, "The impact of artificial intel- ligence enhanced no-code software development platforms on software processes: A literature review," Du¨zce University Journal of Science and Technology, vol. 13, no. 1, pp. 383–401, 2025.

[3] J. J. Pan, J. Wang, and G. Li, "Vector database management techniques and systems," Tsinghua Univ. and Purdue Univ., preprint, 2024.

[4] R. S. Wallace, "The anatomy of A.L.I.C.E.," in Parsing the Turing Test, H. van den Herik, H. van Vugt, and J.-J. Ch. Meyer, Eds. Berlin, Germany: Springer, 2009, pp. 181–210.