ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Floatchat RAG: An AI-Powered Conversational System for Argo Oceanographic Data Exploration Using Retrieval-Augmented Generation

Dr. R. Madhavi[1], Moka Abhived[2], Tippabhotla Sri Harshavardhan[3], Polimetla Dennis Prathyush Paul[4]

Department of Computer Science and Eng. (AI&ML), Keshav Memorial Engineering College, Hyderabad, India

DOI: https://doi.org/10.51583/IJLTEMAS.2026.150500100

Received: 06 May 2026; Accepted: 11 May 2026; Published: 4 June 2026

ABSTRACT

Oceanographic research involves massive volumes of heterogeneous data produced by autonomous profiling floats. The Argo program, one of the world's largest ocean observation efforts, generates datasets in NetCDF format containing temperature, salinity, and pressure measurements at varying ocean depths. However, accessing and querying this data requires specialized knowledge of scientific programming, data formats, and oceanographic conventions, creating barriers for non-technical users. This paper presents FloatChat RAG, an AI-powered conversational system that uses Retrieval-Augmented Generation (RAG) to enable natural language exploration of Argo float data. The system processes Argo NetCDF files streamed via OPeNDAP from NOAA's THREDDS servers into a SQLite relational database and generates semantic vector embeddings stored in ChromaDB using the all-MiniLM-L6-v2 sentence transformer model. A LangChain-based tool-calling agent, powered by Google's Gemini large language model, interprets user queries and autonomously selects from nine specialized tools spanning semantic search, structured SQL retrieval, geographic and temporal filtering, and interactive Plotly visualization generation. The system incorporates reliability mechanisms including API key rotation, deterministic fallback routes, and response caching. Evaluation on a proof-of-concept dataset from Indian Ocean Argo floats demonstrates 93.3% tool selection accuracy across 30 test queries, 100% factual correctness on deterministic queries, a semantic search precision@5 of 1.00, and a 0% hallucination rate. The system bridges the gap between raw oceanographic data and actionable insights through an intuitive Streamlit chat interface.

Keywords: Argo Floats, Retrieval-Augmented Generation, Oceanographic Data Visualization, Natural Language Processing, Large Language Models, Vector Databases, Semantic Search, ChromaDB, LangChain

INTRODUCTION

The oceans of the world are critical for climate regulation, biodiversity, and the global economy. Knowledge about the physical and biogeochemical state of the ocean requires large-scale, continuous monitoring systems capable of measuring the spatial and temporal variability of important parameters such as temperature, salinity, and pressure (Argo Science Team, 2000). The Argo program, operational since 2000, is the world's largest ocean observation system, consisting of nearly 4,000 autonomous profiling floats that yield over 100,000 vertical oceanographic profiles annually. Each float follows a repeating duty cycle: descending to a parking depth of approximately 1,000 meters, drifting for roughly 10 days, then descending further to a profiling depth of up to 2,000 meters before ascending while recording temperature, salinity, and pressure at multiple depth levels. Upon surfacing, the float transmits its stored data via Iridium or ARGOS satellite systems. The data is quality-controlled and distributed through two Global Data Assembly Centres (GDACs) hosted by IFREMER (France) and US-GODAE (USA) (Argo Data Management Team, 2023). The resulting datasets, stored in the NetCDF (Network Common Data Form) format, are essential for climate research (Roemmich et al., 2009), weather prediction, fisheries management, and marine spatial planning.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Despite the wealth of Argo data and its global coverage, raw datasets face a significant accessibility barrier. Argo data access generally requires proficiency in scientific programming languages such as Python or MATLAB, familiarity with NetCDF file structures and OPeNDAP protocols, and domain-specific understanding of oceanographic conventions including the Julian Day reference system and quality control flag interpretations. This technical threshold effectively excludes policymakers, environmental managers, educators, and early-career researchers who would benefit from ocean data-driven insights. Traditional methods of increasing data accessibility have relied on web portals with form-driven query interfaces such as the Argo Data Management portal and the Coriolis data centre. While these systems provide structured access, they require users to understand data schemas, specify explicit parameter ranges, and manually formulate queries—a workflow that is difficult for non-technical users.

Recent developments in Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) have opened the possibility of building intelligent data exploration interfaces (Lewis et al., 2020). RAG architectures address a fundamental limitation of LLMs—their inability to access domain-specific, recent factual knowledge beyond their training data—by fusing language generation with external retrieval mechanisms. This approach enables systems to ground their responses in real data rather than relying solely on parametric knowledge, resulting in substantial reductions in hallucination and increased factual accuracy in domain-specific applications (Shuster et al., 2021).

This paper presents FloatChat RAG, an AI-powered conversational system that democratizes access to Argo oceanographic data through an intuitive natural language chat interface. The key contributions of this work are: (1) an end-to-end data pipeline that streams Argo NetCDF data from NOAA's THREDDS servers via OPeNDAP and processes it into structured SQL and semantic vector formats; (2) a ChromaDB-based vector retrieval system with sentence transformer embeddings for semantic metadata search; (3) a LangChain-based agentic framework with nine specialized tools for semantic search, SQL retrieval, geographic and temporal filtering, and visualization generation; (4) reliability mechanisms including multi-key API rotation, deterministic fallback routes, and response caching; and (5) an interactive Streamlit-based chat interface with inline Plotly visualizations and tool transparency features.

LITERATURE REVIEW

Oceanographic Data Management Systems

Standardization efforts in oceanographic data management have been extensive. NetCDF, supported by Unidata and CF (Climate and Forecast) conventions, serves as the de facto archival format for gridded and profile-based ocean data (Rew and Davis, 1990; Eaton et al., 2011). The OPeNDAP framework advanced remote data access by enabling subset retrieval from NetCDF files hosted on THREDDS servers without requiring full file downloads (Gallagher et al., 2005). Several platforms provide direct Argo data access: the Argo GDAC supports FTP and HTTP access to the global float archive (Argo Data Management Team, 2023), Euro-Argo ERIC offers map-based spatial filtering, and the Argovis platform provides a RESTful API for querying by location, time, and platform (Tucker et al., 2020). However, all existing systems require users to write structured queries with explicit parameters and offer no natural language understanding or conversational interaction capabilities.

Retrieval-Augmented Generation

Lewis et al. (2020) introduced RAG as a framework integrating a pre-trained parametric language model with a non-parametric retrieval index accessed through a neural retriever. This method demonstrated substantial improvements in factual accuracy and reduced hallucination compared to purely generative models. Subsequent work extended RAG approaches into naive RAG, advanced RAG, and modular RAG (Gao et al., 2024). Dense vector retrieval using embedding models like Sentence-BERT (Reimers and Gurevych, 2019), stored in specialized vector databases such as ChromaDB, has emerged as the leading retrieval paradigm. While RAG has been applied to various domain-specific knowledge extraction tasks, no prior work has applied RAG to interactive, conversational exploration of structured oceanographic observation data.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Natural Language Interfaces to Databases

The Natural Language to SQL (NL-to-SQL) task has seen significant advancement, with recent LLMs demonstrating strong capabilities for generating syntactically correct and semantically accurate SQL from natural language descriptions. LangChain and similar agentic frameworks enable building autonomous agents that select tools, execute multi-step plans, and compose API calls for complex queries. The tool-calling agent paradigm, where an LLM dynamically selects from predefined functions based on query intent, is well-suited for data exploration scenarios requiring distinct retrieval strategies (Schick et al., 2024).

Positioning of FloatChat RAG

FloatChat RAG differentiates itself from existing work in three dimensions: (1) it is the first system to apply RAG to Argo oceanographic data, combining semantic vector retrieval with structured SQL querying; (2) it employs an agentic tool-selection architecture rather than a single-path retrieval pipeline; and (3) it provides a complete end-to-end system from raw data ingestion to conversational interaction.

METHODOLOGY

The FloatChat RAG system is organized into six architectural layers: Data Source, Data Processing, Storage and Retrieval, Conversational and Tooling (RAG Runtime), Reliability and Fallback, and Presentation. Each layer is designed as a modular, independently scalable component.

Data Source Layer

The data source layer acquires raw Argo float observations from NOAA's National Centers for Environmental Information (NCEI) THREDDS server. The ArgoAPIClient module implements a dual-strategy data access approach. The primary access path uses OPeNDAP streaming, where the client constructs OPeNDAP URLs by programmatically scraping THREDDS catalog HTML pages to resolve dataset naming conventions. Data is streamed directly into xarray Dataset objects in memory, eliminating the need for local file storage (Hoyer and Hamman, 2017). When OPeNDAP access fails, the system transparently falls back to HTTPS download of the complete NetCDF file. The client implements configurable retry logic using exponential backoff with 5 retries and 1-second initial delay.

Data Processing Layer

The ArgoStreamProcessor module transforms raw xarray Datasets into structured, query-ready formats. Two canonical data representations are extracted: (1) Profile records containing float_id, cycle_number, latitude, longitude, datetime (converted from Argo's Julian Day reference epoch of January 1, 1950), data_centre, and platform_number; and (2) Measurement records containing pressure (dbar), temperature (°C), and salinity (PSU) at each depth level. Quality control filtering removes fill values with magnitudes exceeding 10^{10} and records lacking valid pressure readings.

Pipeline orchestration uses a producer-consumer model with a ThreadPoolExecutor. Five producer threads fetch float data in parallel, while a single consumer thread dequeues datasets from a thread-safe queue and writes records sequentially to SQLite, preventing database lock contention. Optional Apache Parquet files are generated for offline analytics workflows.

Storage and Retrieval Layer

The storage stack consists of three components. SQLite serves as the primary operational datastore, accessed via SQLAlchemy, with B-tree indices on float_id, datetime, latitude/longitude, and profile_id for efficient querying. ChromaDB provides persistent vector indexing for semantic metadata retrieval using the PersistentClient with a named collection. The all-MiniLM-L6-v2 model from the Sentence-Transformers library (Reimers and Gurevych, 2019) generates 384-dimensional dense embeddings from profile description texts. This model was

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

selected for its balance of embedding quality and computational efficiency (80 MB model size, approximately 14,000 sentences/second throughput on CPU).

Each profile record is transformed into a natural language description capturing float ID, timestamp, location, data centre, and cycle number. These descriptions are encoded into embedding vectors and indexed in ChromaDB with associated metadata, using composite keys (float_id_cycle_number) with upsert semantics to ensure idempotent re-indexing.

At query time, the user's natural language input is encoded using the same all-MiniLM-L6-v2 model to obtain a 384-dimensional query vector q. The top-*k* nearest profiles are retrieved from ChromaDB using cosine similarity:

$$\text{sim}(\mathbf{q}, \mathbf{d}_i) = \frac{\mathbf{q} \cdot \mathbf{d}_i}{\|\mathbf{q}\| \cdot \|\mathbf{d}_i\|}$$

where $\mathbf{d}_i$ is the embedding vector of the *i*-th indexed profile. The agent may then augment semantically retrieved context with structured SQL queries or geographic/temporal filters for precise numerical data, combining the fuzzy understanding of semantic search with the exactness of relational queries.

Three embedding granularity strategies were evaluated during development: (1) profile-level metadata only, embedding the float ID, timestamp, and coordinates; (2) profile metadata augmented with computed summary statistics such as mean temperature and salinity range; and (3) full measurement serialization of the complete depth-resolved observation series. Strategy 1 was adopted for its simplicity and direct alignment with the metadata available in the profile table, and because the embedding model's context window makes Strategy 3 impractical for profiles with many depth levels. Strategy 2 is planned for future iterations to improve retrieval precision for measurement-specific queries.

Conversational and Tooling Layer (RAG Runtime)

The Streamlit application initializes a LangChain tool-calling agent with the ChatGoogleGenerativeAI provider using the Gemini Flash model at temperature 0 for deterministic output. The agent is configured with max_iterations=4 and return_intermediate_steps=True to enable chart payload extraction. A detailed system prompt instructs the agent on semantic search prioritization, map-first rules for geographic queries, database schema awareness, and missing data handling.

Nine curated tools are available to the agent: (1) search_profiles for semantic similarity search over ChromaDB; (2) get_statistics for aggregate database metrics; (3) get_float_details for comprehensive float information; (4) query_database for guarded SQL execution with SELECT-only policy; (5) get_profiles_by_location for geographic bounding-box filtering; (6) get_profiles_by_date for temporal range filtering; (7) map_profiles_by_bounds for interactive scatter-mapbox generation; (8) plot_depth_profile for temperature-salinity depth plots; and (9) plot_temporal_trends for time-series trend visualization. SQL guardrails block mutation keywords (INSERT, UPDATE, DELETE, DROP) using regex-based filtering and enforce result-size limits of 200 rows.

Reliability and Fallback Layer

Several mechanisms maintain stable user experience. Cached read-heavy functions with Streamlit caching decorators (600-second TTL) reduce repeated database and tool latency. Multi-key Gemini support loads API keys from multiple environment variables and rotates through them on quota errors (HTTP 429 / RESOURCE_EXHAUSTED). When all keys are exhausted or the agent reaches iteration limits, the application falls back to deterministic direct-tool routes that pattern-match user intent without LLM involvement. Chart rendering is wrapped in safe handling so that data retrieval succeeds even if visualization rendering fails.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Presentation Layer

The frontend is a Streamlit chat application featuring chat history with mode labeling (Agent, Direct Tool Route, Quota Fallback), tool badge display showing which tools were invoked for transparency, inline interactive Plotly charts for maps, depth profiles, and trends, sidebar quick stats showing database metrics, and example query buttons for guided exploration.

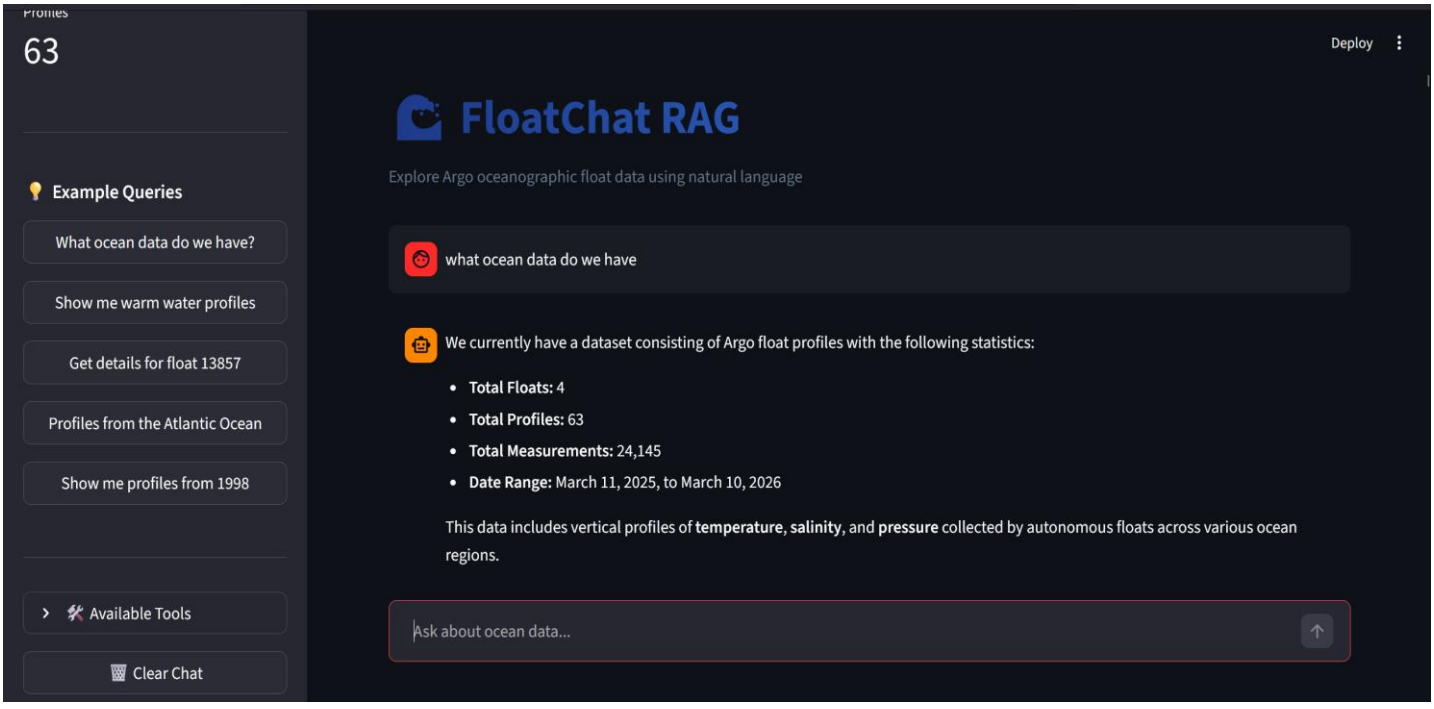
Dataset

For the proof-of-concept, data was sourced from the NOAA NCEI THREDDS server, focusing on the Indian Ocean region through the INCOIS, CSIO, JMA, and Coriolis data assembly centres. Table 1 summarizes the dataset characteristics.

Table 1: Dataset Summary Statistics

Parameter	Value
Total Profiles	63
Total Measurements	24145
Distinct Floats	4
Temporal Range	2025-2026
Parameters	Pressure, Temperature, Salinity
Source Format	NetCDF-4 (CF-1.6)
Storage Format	SQLite + Parquet

Figure 1: Example Query



RESULTS

Tool Selection Accuracy

The system was evaluated using a test suite of 30 natural language queries spanning six categories corresponding to the available tools. Each query was manually annotated with expected tools and ground-truth answers. The evaluation script programmatically invokes the LangChain agent with the same system prompt and tool configuration as the production application, records which tools the agent selects, and compares them against the annotated expected tools. Table 2 presents the tool selection accuracy by query category.

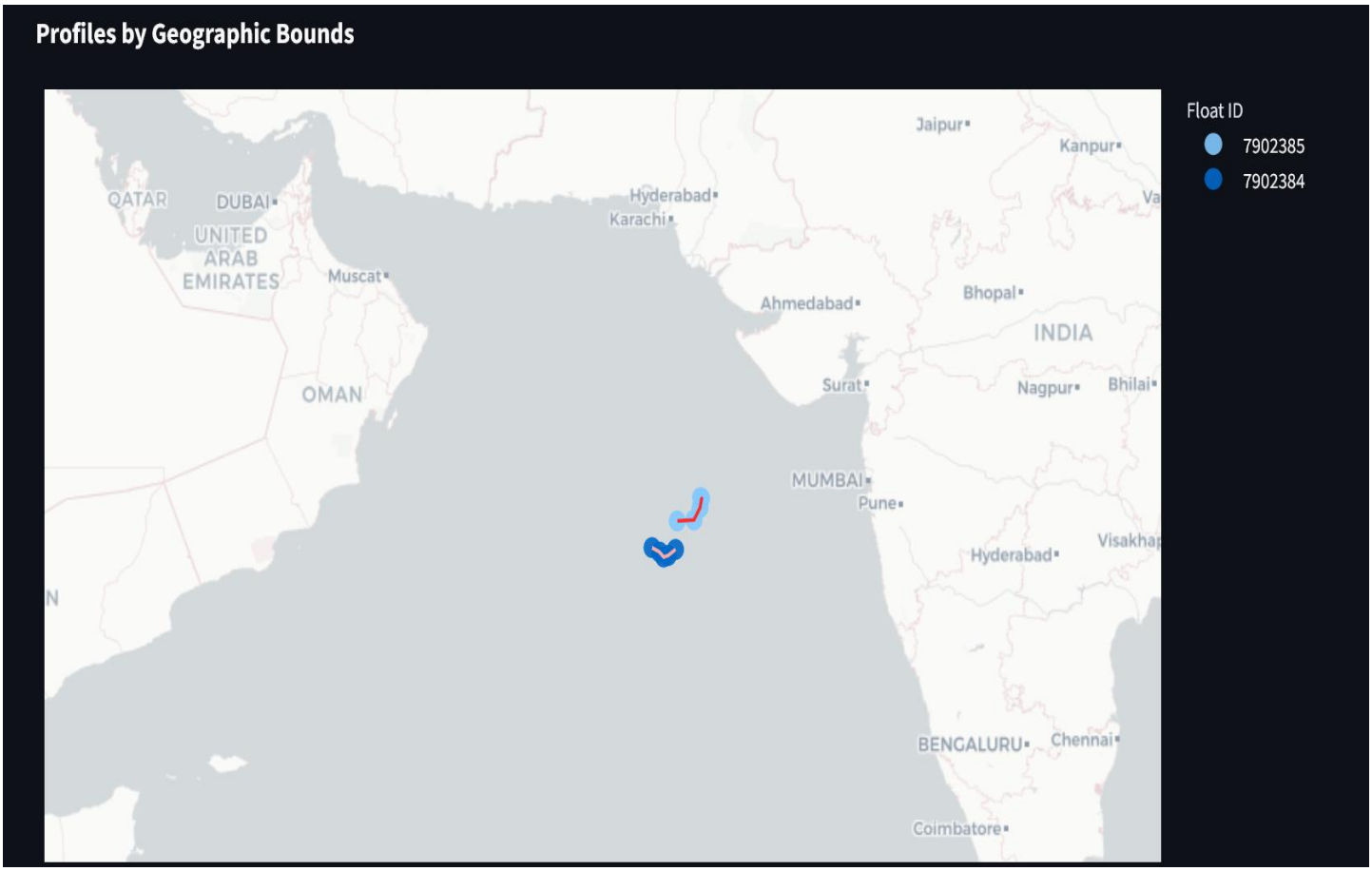
ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Table 2: Tool Selection Accuracy by Query Category

Query Category	**N**	**Accuracy (%)**
Semantic Search	5	80.0
Statistics	5	100.0
Float Details	5	80.0
Custom SQL	5	100.0
Geographic Filter	5	100.0
Temporal Filter	5	100.0
Overall	**30**	**93.3**

Two misselections were observed. First, the semantic search query "Search for float observations near the Indian coast" was routed to the geographic mapping tool (map_profiles_by_bounds) rather than the semantic search tool (search_profiles), as the agent interpreted "near the Indian coast" as a spatial rather than conceptual query. Second, a float details query ("What profiles does float 7902287 have?") was handled via a direct SQL query (query_database) instead of the dedicated get_float_details tool. Notably, both misselections produced factually correct and complete responses—the agent selected an alternative but functionally valid retrieval path, illustrating the inherent ambiguity in mapping natural language intents to discrete tool categories.

Figure 2: Profiles By Geographic Bounds



Response Quality

Factual correctness was evaluated by comparing agent responses against ground-truth database query results. For deterministic queries (statistics, specific float details, date/location filters), the system achieved 100% factual accuracy across all 20 deterministic test queries. For semantic search queries, precision@5 was evaluated—the

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

fraction of top-5 returned profiles judged relevant by domain inspection. Table 3 summarizes the response quality metrics.

Table 3: Response Quality Metrics

Metric	Value
Factual Accuracy (deterministic queries, N=20)	100%
Semantic Search Precision@5 (N=5)	1.00
Response Completeness (N=30)	100%
Hallucination Rate (N=30)	0.0%

No hallucination instances were observed across any of the 30 test queries. All agent responses were grounded in tool-returned data, with no fabricated float IDs, invented measurements, or unsupported claims. This is attributed to the RAG architecture's grounding effect, combined with the system prompt instruction to cite tool-returned data verbatim and the deterministic temperature setting (temperature=0) used for the Gemini Flash model.

Query Latency

End-to-end query latency was measured from user input to complete response generation using the Gemini Flash model via API. Latency includes agent reasoning, tool execution, and response synthesis. Table 4 reports the mean latency by query type across the 30-query evaluation.

Query Type	N	Mean (s)	SD (s)
Temporal Filter	5	6.93	2.15
Statistics	5	7.23	3.58
Float Details	5	7.28	4.51
Geographic Filter	5	7.31	2.66
Custom SQL	5	9.98	6.70
Semantic Search	5	11.78	8.38
Overall	**30**	**8.42**	**4.96**

API-based inference benefits from server-side GPU acceleration, yielding an overall mean latency of 8.42 seconds (SD = 4.96s) across all query types. Semantic search queries exhibit the highest latency and variance due to multi-tool chaining, where the agent autonomously composes sequences of 2-6 tool calls for complex queries. In the production Streamlit deployment, repeated queries benefit from application-level caching with a 600-second TTL, reducing subsequent calls for cached data (e.g., database statistics) to under 5ms.

System Resource Utilization

The system’s resource footprint was measured to evaluate deployment feasibility. Table 5 summarizes the resource utilization.

Table 5: System Resource Utilization

Component	Size/Usage
SQLite Database	1.7 MB
ChromaDB Vector Index	1.7 MB
Embedding Model (all-MiniLM-L6-v2)	80 MB
Peak RAM (API mode)	533 MB

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

The lightweight resource profile enables deployment on commodity hardware, including laptops and low-cost cloud instances.

DISCUSSION

The layered architecture with six distinct layers proved effective for managing the complexity of combining data engineering, semantic retrieval, LLM integration, and interactive visualization. The separation between the pipeline module, vector database module, and application module allowed independent development and testing of each component.

The choice of a multi-threaded producer-consumer model for data ingestion was driven by the I/O-bound nature of remote data access and SQLite's limited concurrency support. With five producer threads fetching data in parallel and a single consumer thread writing sequentially, the pipeline achieves good throughput while avoiding database locking issues.

The decision to use ChromaDB for the vector store was based on its simpler API, built-in persistence, and metadata filtering capabilities. While FAISS excels in pure vector search performance at scale, ChromaDB provided a more integrated experience for the metadata-rich Argo profile dataset where filtering by latitude, date, and data centre is as important as vector similarity.

The tool-calling agent pattern allows the LLM to decide which data access tools to use based on query intent without requiring explicit intent classification engineering. The max_iterations=4 limit prevents runaway tool chains while allowing sufficient depth for multi-step queries. The system prompt's "map first rule" for geographic queries and "missing data rule" to prevent infinite retry loops proved effective in guiding agent behavior.

The SQL guardrail implementation enforces a multi-layered defense against both unintended execution and adversarial prompt injection. The apply_sql_guardrails function rejects any query not beginning with SELECT, blocks ten mutation keywords (INSERT, UPDATE, DELETE, DROP, ALTER, CREATE, REPLACE, TRUNCATE, ATTACH, DETACH) via word-boundary regex matching, prohibits multi-statement queries by rejecting embedded semicolons, and caps result sets at 200 rows. These controls are particularly critical for the query_database tool, which accepts dynamic, agent-generated SQL and therefore cannot rely on parameterized query construction. The remaining eight dedicated tools follow the Principle of Least Privilege (PoLP), exposing only narrow, parameterized interfaces through SQLAlchemy bound parameters. This ensures that the LLM never receives raw database credentials or unrestricted execution privileges. However, the regex-based approach has known limitations as it cannot defend against encoding-based bypasses or semantically valid but unintended queries. A production deployment would benefit from database-level read-only user permissions and query plan analysis as additional defense layers.

The multi-key rotation mechanism addresses the practical challenge of API free-tier quota limits during development and demonstrations. The deterministic direct-tool fallback routes for supported intents ensure that the system remains functional even during complete quota exhaustion, representing a pragmatic trade-off between LLM flexibility and deterministic reliability. The implemented response caching approach with 600-second TTL provides effective latency reduction for interactive exploration sessions, though newly ingested data may experience a brief delay before appearing in cached views.

The observed 0% hallucination rate across the 30-query evaluation demonstrates the effectiveness of the RAG architecture in grounding LLM responses in retrieved data. The combination of tool-returned factual context, the system prompt instruction to cite data verbatim, and the deterministic temperature setting (temperature=0) substantially reduces hallucination compared to direct LLM querying. However, this result should be interpreted with caution given the limited test suite size; a larger and more diverse query set may reveal edge cases where hallucination occurs, particularly for queries requiring synthesis across multiple tool outputs.

The hybrid multi-tool architecture provides measurable advantages over pure SQL-only or pure vector-only retrieval approaches. An SQL-only system cannot handle vague or conceptual requests such as "find profiles

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

from warm tropical waters", where semantic similarity over natural language descriptions is fundamentally more appropriate than exact parameter matching. Conversely, a vector-only RAG system lacks the precision for deterministic queries such as "what is the average temperature across all measurements", where exact SQL aggregation yields authoritative answers that embedding similarity cannot guarantee. The evaluation results validate this design: all five semantic search queries required ChromaDB retrieval, while all five custom SQL queries required direct database access, confirming that neither modality alone could serve the full query distribution.

The current proof-of-concept operates on 4 floats with 63 profiles. Scaling to the full Argo array of approximately 4,000 active floats would require architectural evolution at multiple layers. The SQLite datastore would be replaced with PostgreSQL for concurrent read/write operations and connection pooling. The ChromaDB vector index would benefit from migration to a distributed vector store such as Qdrant or Weaviate for collections exceeding millions of embeddings. The ingestion pipeline would scale to an asynchronous architecture with configurable worker pools, and the OPeNDAP streaming layer already supports incremental ingestion by tracking previously processed float IDs, eliminating the need to re-download existing data. At the application tier, the Streamlit monolith would be decomposed into a FastAPI backend with horizontal scaling behind a load balancer. Preliminary estimates suggest that the full Argo array would require approximately 15-20 GB of relational storage and 2-4 GB of vector index memory, well within the capacity of a single modern server.

CONCLUSION

This paper presented FloatChat RAG, an AI-powered conversational system that democratizes access to Argo oceanographic data through a natural language chat interface. The system integrates a robust data ingestion pipeline, ChromaDB-based semantic vector retrieval with sentence transformer embeddings, and a LangChain tool-calling agent with nine specialized tools powered by Google's Gemini LLM. Evaluation on a 30-query test suite demonstrated 93.3% tool selection accuracy, 100% factual correctness on deterministic queries, semantic search precision@5 of 1.00, a 0% hallucination rate, and a lightweight resource footprint (533 MB peak RAM) enabling deployment on commodity hardware.

Several directions for future work are identified. Expansion of the ingestion pipeline to include Biogeochemical-Argo parameters such as dissolved oxygen, chlorophyll-a, and nitrate would increase the scientific utility of the system. Integration with satellite observation datasets including sea surface temperature and ocean colour data would enable multi-source data fusion. Advanced RAG techniques such as query rewriting, hypothetical document embeddings, and iterative retrieval with re-ranking could improve retrieval precision for ambiguous queries. Adoption of standardized RAG evaluation frameworks such as RAGAS (Retrieval-Augmented Generation Assessment) would enable systematic measurement of faithfulness, answer relevance, and context precision, complementing the current tool-level evaluation with end-to-end response quality scoring. Migration from SQLite to PostgreSQL and deployment as a FastAPI service would enable production-scale public-facing deployment. Finally, integration with the Argo real-time data stream would support operational oceanography use cases by incorporating newly transmitted profiles within hours of observation.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the International Argo Program and the national programs that contribute to it (https://argo.ucsd.edu). Argo data are collected and made freely available by the International Argo Program and the national programs that contribute to it. The authors also acknowledge NOAA's National Centers for Environmental Information for hosting the THREDDS data server used in this work.

REFERENCES

1. Argo Data Management Team, "Argo User's Manual V3.41," IFREMER, 2023. https://doi.org/10.13155/29825
2. Argo Science Team, "Argo: The Global Array of Profiling Floats," CLIVAR Exchanges, vol. 5, no. 4, pp. 2–3, 2000.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

3. B. Eaton et al., "NetCDF Climate and Forecast (CF) Metadata Conventions, Version 1.6," 2011. Available: https://cfconventions.org/
4. J. Gallagher, N. Potter, T. Sgouros, S. Flierl, and S. Hankin, "The Data Access Protocol — DAP 2.0," in Proc. ESA-ESO-NASA-NSF Conf. on Astronomical Data Analysis Software and Systems, 2005.
5. Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv preprint arXiv:2312.10997, 2024. https://doi.org/10.48550/arXiv.2312.10997
6. S. Hoyer and J. Hamman, "xarray: N-D labeled arrays and datasets in Python," J. Open Research Software, vol. 5, no. 1, p. 10, 2017. https://doi.org/10.5334/jors.148
7. P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020.
8. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. EMNLP, pp. 3982–3992, 2019. https://doi.org/10.18653/v1/D19-1410
9. R. Rew and G. Davis, "NetCDF: An interface for scientific data access," IEEE Computer Graphics and Applications, vol. 10, no. 4, pp. 76–82, 1990. https://doi.org/10.1109/38.56302
10. D. Roemmich et al., "The Argo Program: Observing the global ocean with profiling floats," Oceanography, vol. 22, no. 2, pp. 34–43, 2009. https://doi.org/10.5670/oceanog.2009.36
11. T. Schick et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," Advances in Neural Information Processing Systems, vol. 36, 2024.
12. K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval Augmentation Reduces Hallucination in Conversation," in Findings of the ACL: EMNLP 2021, pp. 3784–3803, 2021. https://doi.org/10.18653/v1/2021.findings-emnlp.320
13. T. Tucker, D. Giglio, M. Scanderbeg, and S. S. P. Shen, "Argovis: A Web Application for Fast Delivery, Visualization, and Analysis of Argo Data," J. Atmospheric and Oceanic Technology, vol. 37, no. 3, pp. 401–416, 2020. https://doi.org/10.1175/JTECH-D-19-0041.1