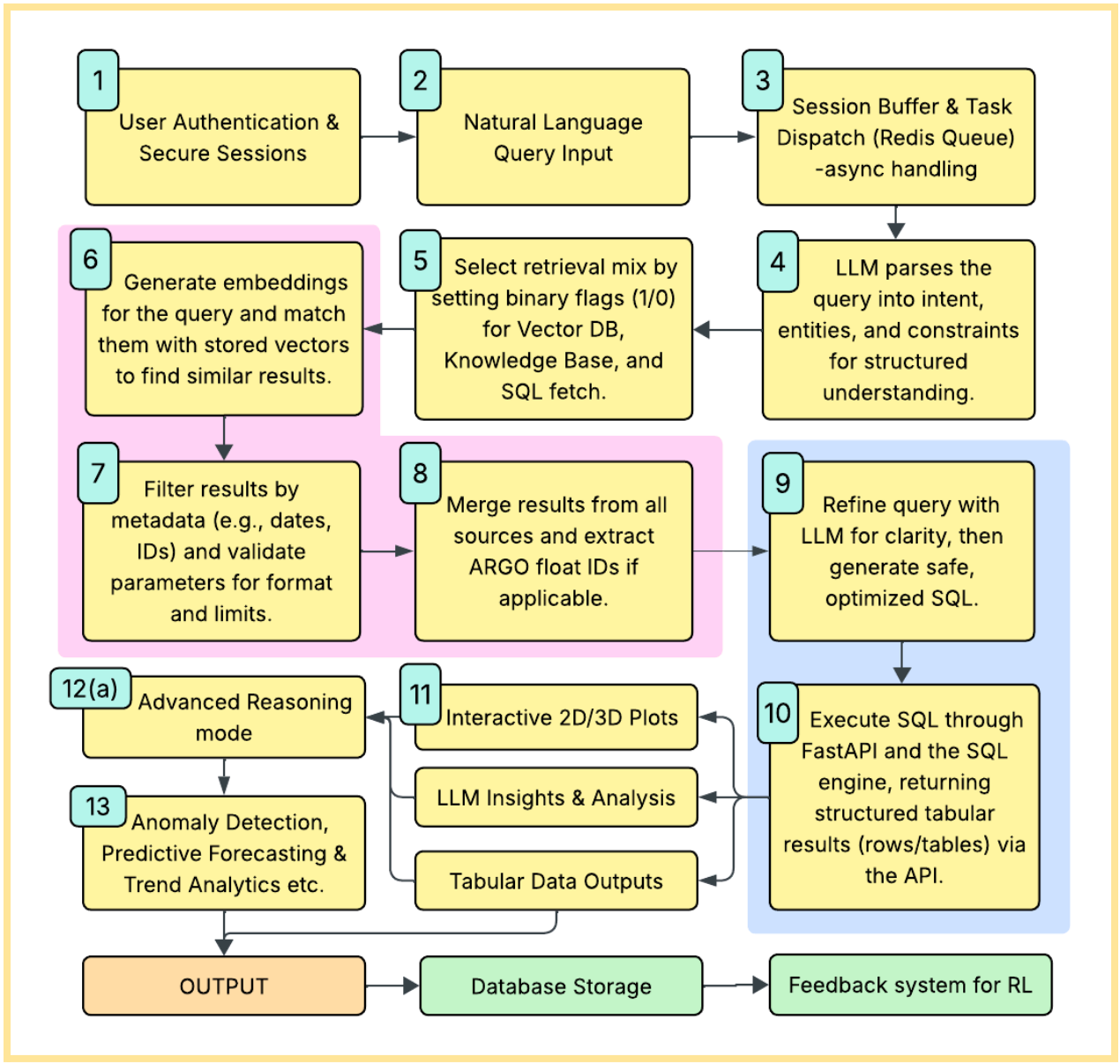
System Architecture and Processing Pipeline for Anantha - ARGO Conversational Interface

Team Name : Sirius
Team Id : 99896
Team Members : R. Advaith, V. Subhash, R. Yashwanth, S. Raviteja, M. Abhinav, J. Shreya

Topics Covered in This Document

15. User Feedback Capture and Integration into Reinforcement Learning Feedback Loops (RLHF)

Workflow Diagram :



1. User Authentication and Secure Session Management Initialization

The authentication entry point of the system architecture leverages **Firebase Authentication** as a managed identity provider, thereby externalizing credential storage, password hashing, and user lifecycle management into a hardened, cloud-native service. Authentication tokens are issued as **Firebase ID Tokens**, which are JWTs (JSON Web Tokens) signed by Google's public keys (rotated automatically via the Google Identity Platform). These tokens encapsulate user identity, verification state, and optional custom role claims, and they are verified server-side through signature validation against the published JWKS (JSON Web Key Set).

In contrast to traditional self-managed identity stores, this approach obviates the need for a local password hashing scheme (e.g., argon2id), while inheriting Firebase's built-in resistance to brute-force, credential stuffing, and replay attacks. Upon successful sign-in, clients receive a short-lived ID Token (≈1 hour TTL) alongside an automatically managed Refresh Token. The Refresh Token, scoped to the Firebase project, enables silent renewal of the ID Token without user re-authentication, ensuring stateless access flows in coordination with the FastAPI middleware layer that performs verification and role claim extraction.

Session continuity is achieved by binding Firebase-issued identity tokens to a **Redis-backed session context store**. Each active session is associated with a unique `session_id` and enriched with ephemeral metadata such as prior query embeddings, intent vectors extracted from conversational history, and access-tier annotations (e.g., Analyst, Researcher, Public Viewer). This allows downstream LLM-driven components to maintain **multi-turn dialogue coherence** while enforcing least-privilege access controls.

Security controls are layered:

- **CSRF protection** is enforced by delivering ID Tokens via explicit authorization headers (`Authorization: Bearer`) rather than cookies, thereby decoupling authentication from browser session semantics.

- **Rate limiting** is implemented through Redis token-bucket algorithms (e.g., 20 authenticated requests/second), mitigating denial-of-service or abuse scenarios.

- **Session invalidation** propagates from Firebase via revocation APIs: when a user password is reset, MFA is enforced, or explicit logout occurs, the corresponding refresh tokens are invalidated globally and sessions flushed from Redis.

- **Audit logging** captures all authenticated data accesses, recording `{uid, role, query_id, timestamp, resource, action}` in structured, immutable logs. These

support compliance with the FAIR data principles—**Findable, Accessible, Interoperable, and Reusable**—by ensuring traceability across human and programmatic interactions.

By integrating Firebase Authentication with Redis-based ephemeral session enrichment, the system establishes a hybrid model that unifies **stateless cloud-native identity guarantees** with **stateful conversational memory management**, ensuring both security and usability for non-technical ocean data consumers.

---

2. Natural Language Query Acquisition and Input Processing Interface

Upon successful authentication, users interact via a **frontend abstraction layer** constructed using a hybrid **Streamlit + React** stack. This layer facilitates **freeform natural language input capture**, augmented with rich widgets for contextual assistance (e.g., dropdowns for known float IDs, date-range pickers, and BGC parameter selectors).

Input queries are passed to a backend API endpoint (`/query/parse`) via asynchronous HTTP (Axios/FastAPI), tagged with user `session_id`, geospatial preference, and historical dialog context. Each input utterance is subjected to preprocessing routines, which include:

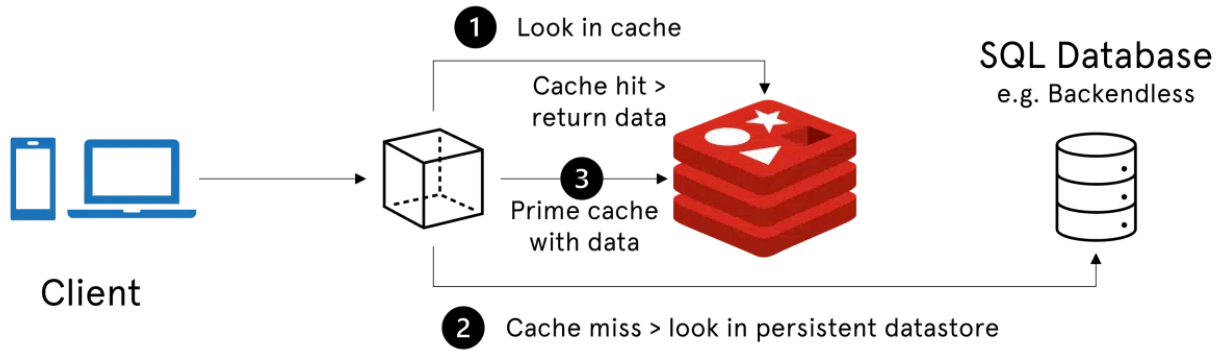
- **Normalization**: Unicode NFKC conversion, whitespace trimming, case folding.
- **Named Entity Recognition (NER)** for marine-specific entities: e.g., `"Arabian Sea"`, `"Argo float 3901234"`, using domain-adapted transformer-based models.
- **Temporal Resolution**: Expression parsing of phrases like “last 6 months”, “March 2023”, into canonical ISO 8601 timestamps using `Heideltime`-style interval resolvers.

This processed input is then appended to the user’s **conversation history buffer** in Redis, retaining both the raw and normalized query text alongside metadata such as `query_time`, `confidence_scores`, and `geo_flag`.

Moreover, the frontend UI integrates **autocomplete and semantic suggestion modules**, powered by shallow semantic embeddings retrieved from the vector database (ChromaDB). For example, typing "salinity" triggers dynamic completions like "salinity near 100m depth" or "salinity anomalies in Niño zones", reflecting frequent query structures observed in system logs.

These augmentations not only improve user experience but also **prime the LLM parser** for high-quality downstream intent extraction, ensuring minimal semantic drift in subsequent steps.

3. Session Buffer & Task Dispatch via Redis Queue (Asynchronous Handling)



Once the normalized query and associated metadata (from Step 2) are prepared, the system transitions into a **distributed task orchestration phase**, designed to handle high-concurrency user interactions and prevent LLM bottlenecks. This phase utilizes a **Redis-backed queueing infrastructure** (using `Redis` as the broker) to decouple frontend query capture from backend processing, providing **non-blocking, fault-tolerant dispatch semantics**.

Upon enqueueing a user query, the system generates a **task object** serialized as a JSON payload:

```
{
  "session_id": "abc123",
  "query_id": "q_xyz",
  "normalized_query": "Show salinity profiles near the equator in
March 2023",
  "timestamp": "2025-09-17T13:04:21Z",
  "user_roles": ["researcher"],
  "context_hash": "md5(embedding+NER+GeoLoc)",
  "conversation_state_ptr": "redis://ctx:session_abc123"}
```

Each task is then pushed to a **priority Redis queue** (`query:parse:priority`) based on heuristics like:

- User role (e.g., admins get priority latency),
- Query complexity (estimated via token count + semantic entropy),

- Queue length + age of pending tasks.

Worker nodes running on **Python asyncio event loops** pull from this queue and spin off **coroutine tasks** using `uvloop` or `Trio` backends for sub-50ms task startup latency.

Key Mechanisms Enabled Here:

- **Task TTL enforcement**: Tasks older than 120s are invalidated to prevent stale processing.
- **Concurrency-safe deduplication**: Using Redis' `SETNX` and Lua scripts to prevent race conditions in duplicate query dispatch.
- **Error propagation channel**: Failures (e.g., tokenizer overflow, rate-limiting from LLM API) are pushed to a secondary queue for logging and fallbacks.
- **Task chaining**: If semantic ambiguity is detected (via confidence scores from LLMs in step 4), a secondary task is enqueued automatically to elicit clarification from the user.

This architecture enables **high-throughput LLM orchestration**, allowing the system to handle hundreds of concurrent user queries while isolating LLM invocation costs and latency from the frontend.

---

4. Advanced Semantic Query Parsing and Structural Interpretation Using LLMs

This stage forms the **core semantic translation engine**, wherein natural language queries are decomposed into formal, machine-interpretable semantic frames using a **domain-tuned Large Language Model (LLM)**—specifically accessed via the **Gemini API** (multimodal, instruction-tuned, long-context capable).

Upon invocation, the query is embedded into a **Model Context Protocol (MCP)** template, ensuring contextual grounding and reproducibility. A typical MCP prompt structure:

Technical Notes:

- LLM prompting is **zero/few-shot**, augmented by **in-context metadata injection** for user-specific intent continuity.
- Entity detection includes both **explicit** (`"Arabian Sea"`) and **implicit** (`"last 6 months"`) resolution, resolved using internal libraries and verified against a **marine geospatial ontology** derived from GEBCO and WOD datasets.
- The model returns not just structured intents, but **probabilistic confidence scores** per field, used to trigger fallback chains if ambiguity exceeds thresholds.

- Outputs are parsed using **pydantic-based schemas**, allowing runtime validation and typed structuring of the LLM response into downstream classes (`SemanticIntent`, `GeoTemporalConstraint`, `DataSpecRequest`, etc.).

To ensure robustness, a **fallback controller** detects out-of-distribution queries (e.g., "show underwater volcanoes") and routes them to a fallback KB/QA retrieval engine or triggers clarification prompts.

This phase ensures that **human natural language** is transformed into a **machine-readable abstraction**, which becomes the basis for selecting retrieval strategies in the next step.

Enhanced Query Generation Example :

1. **User prompt : "presssute trenda over years"**

   Enhanced query : "Retrieve ARGO float profiles data with PRES (pressure) parameter measurements, suitable for analyzing long-term trends over multiple years."

2. **User prompt : "Give me floats after feb10 '23 and before 23 mar '24 , give me all"**

   Enhanced query : "Retrieve ARGO float profiles and associated oceanographic data, including temperature, salinity, pressure, and any bio-geochemical (BGC) parameters, for floats that commenced their missions or recorded profiles after February 10, 2023, and before March 23, 2024. Include details such as float IDs, WMO numbers, launch dates, mission durations, operating institutions, and specific sensor types."

3. **User prompt : "floats after mar 2023 in arabian sea"**

   Enhanced query : "Retrieve ARGO float profiles and associated oceanographic data, including temperature, salinity, pressure, and any bio-geochemical (BGC) parameters, for floats that commenced their missions or recorded profiles in the Arabian Sea region after March 31, 2023. Include details such as float IDs, WMO numbers, launch dates, mission durations, operating institutions, and specific sensor types."

---

5. Retrieval Strategy Classification: Hybrid Mode Selection Between Vector Search, SQL, and Knowledge Base

Upon successful semantic interpretation of the user query (via LLM-based parsing), the system enters the Retrieval Strategy Classification phase, where it determines the appropriate data access modality from three channels:

- Vector Semantic Search (V_DB): Based on natural language embeddings over ARGO float documents and summaries.

- Relational SQL Access (SQL): For strict geotemporal slicing and numerical filtering.
- Curated Knowledge Base (K_BASE): For static or factual queries (e.g., float counts, definitions, instrument specs).

Classification Engine

This phase is powered by a Bayesian Intent-Constraint Mapper (BICM), which uses the output semantic schema from the LLM to evaluate the following decision features:

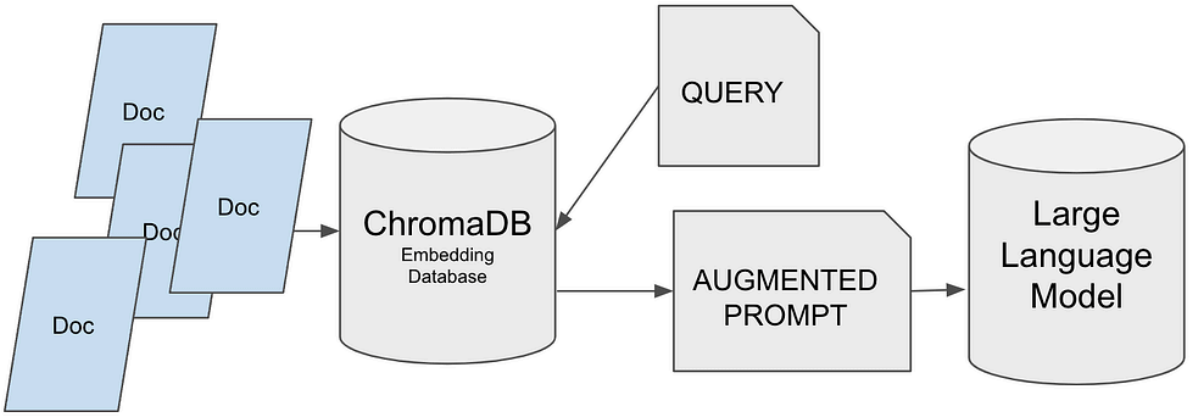
Feature	Description	Example
`intent_class`	From LLM output (e.g., `"compare_parameters"`)	“Compare BGC values in Arabian Sea”
`temporal_scope`	ISO-8601 parsed	`"2025-04-01" to "2025-09-17"`
`spatial_filtering`	Presence of lat/lon or named regions	`"Bay of Bengal"` → BBOX
`parametric_constraints`	Filters like `"salinity > 34"`	True → SQL
`query_semantic_density`	Token-level entropy	High → Vector DB
`entity_type`	Domain NER → Float ID, Sensor Name	`"float_1902674"`
`factuality_score`	Matches to static KB entries	Definitions, tags

Based on this, the system sets a retrieval vector $R = [V\_DB, SQL, K\_BASE] \in \{0,1\}^3$. An example result:

```
{
  "vector_db": true,
  "sql_enabled": true,
  "knowledge_base": false,
  "priority": ["sql", "vector_db"],
  "embedding_model": "all-MiniLM-L6-v2",
  "sql_tables": ["argo_profiles", "bgc_data"],
  "expected_return_type": "profile_data + float_metadata"
}
```

This guides the parallel execution pipeline in fetching relevant entries—via semantic similarity in ChromaDB and explicit filtering in PostgreSQL. If ambiguity or low confidence is detected, a clarification prompt is triggered, re-entering the user into a feedback loop with the LLM parser.

6. Embedding Vector Generation and Semantic Similarity Search in ChromaDB (With Metadata Constraints)



In cases where `vector_db: true` is asserted by the classifier, the system enters the semantic retrieval phase, using ChromaDB (built over FAISS) to perform approximate nearest neighbor (ANN) search against high-dimensional embeddings of ARGO float metadata and descriptive documents.

Embedding Space and Corpus

Each document in the ChromaDB collection represents a summarized description of a unique ARGO float profile, such as:

> *"Float 1902674 (WMO A12345), deployed by Dr. Smith from CSIR... Drift Summary: Arabian Sea (65% of profiles)... Anomalies: sensor calibration issue on 2019-11-22."*

The corresponding metadata is highly structured and includes nested scientific instrumentation info, geospatial extents, anomaly logs, mission durations, etc.

Document Schema:

```
{
  "id": "float_1902674_profile_1",
  "document": "...",  // Natural language description
  "embedding": [...], // 384/768-D vector
```

```
  "metadata": {
    "float_id": "1902674",
    "dominant_region": "Arabian Sea",
    "start_date": "2015-05-20",
    "end_mission_date": "2024-01-30",
    "sensors": [...],
    "tags": ["Salinity Monitoring", "High-Quality Data"],
      "special_features_or_anomalies": ["Sensor calibration issue on
2019-11-22"],
    ...
  }
}
```

Embedding Generation and Querying

The incoming natural language query (e.g., "Find floats with salinity anomalies in the Arabian Sea") is first embedded using a domain-aligned sentence transformer (e.g., `all-MiniLM-L6-v2`) to produce a $\texttt{query\_vector} \in \mathbb{R}^{384}$.

Then, a constrained vector similarity query is issued to ChromaDB:

```
results = chroma.query(
  query_embeddings=[query_vector],
  n_results=15,
  where={
    "dominant_region": "Arabian Sea",
    "tags": {"$contains": "Salinity Monitoring"}
  },
  include=["documents", "metadatas", "distances"]
)
```

Retrieval Mechanics:

- Similarity Metric: `cosine_similarity(query_vec, db_vec)` using FAISS HNSW.
- Metadata Filtering: Executed before similarity ranking to reduce vector comparison load.
- Payload Returned: Contains structured metadata, full float summaries, anomaly flags, and distance scores.

Example Output (Abstracted):

```
[
  {
    "float_id": "1902674",
    "score": 0.87,
```

```
    "dominant_region": "Arabian Sea",
    "special_features_or_anomalies": [
      "Sensor calibration issue on 2019-11-22"
    ],
    "mission_duration_days": 3185,
    "tags": ["Long-Term Deployment", "High-Quality Data"]
  },
  ...
]
```

This vector-based retrieval complements SQL-based slicing, particularly when:

- Users refer to latent concepts (e.g., “abnormal drift”),
- Queries are underspecified (“Find interesting floats”),
- Or need approximate recall across fuzzy region/time overlaps.

By integrating both semantic similarity and structured metadata filtering, this hybrid retrieval ensures both interpretability (via metadata) and flexibility (via neural embeddings), forming the backbone of contextual relevance in ARGO float discovery.

Structure of Metadata :

Field	Value
id	`float_1902674_profile_1`
float_id	`1902674`
wmo_inst_type	`A12345`
pi_name	`Dr. Smith`
operating_institution	`CSIR`
project_name	`Argo Project`
launch_date	`2015-05-20`
launch_latitude	`-6.01`
launch_longitude	`76.5`
start_date	`2015-05-20`

end_mission_date	2024-01-30
end_mission_status	Active
mission_duration_days	3185
mission_duration_years	8.7
num_profiles	500
sensors	Seabird SBE41 → Measures Temperature (°C), Salinity (PSU)
manufacturer	Seabird Electronics
model	SBE41
deployment_type	Profiling
dominant_region	Arabian Sea
pct_in_dominant_region	65%
regions_visited	Arabian Sea, Bay of Bengal
latitude_range	-10.5 → 5.1
longitude_range	60.0 → 80.0
centroid	Lat: -2.5, Lon: 70.0
first_region	Arabian Sea
last_region	Bay of Bengal
special_features_or_anomalies	Drift outside expected area on 2021-06-15; Sensor calibration issue on 2019-11-22
data_quality_score	High
last_data_update	2024-01-30
tags	Arabian Sea, Salinity Monitoring, Long-Term Deployment, High-Quality Data

7. Metadata-Driven Filtering, Validation, and Constraint Enforcement

After initial candidate documents are retrieved—either from **ChromaDB (semantic vector search)**, **PostgreSQL (structured SQL query)**, or both—an intermediate filtration layer applies **domain-informed constraint logic** over the metadata schema to enforce strict alignment with the parsed user intent.

This stage plays a critical role in ensuring that:

- **False positives from semantic retrieval** are pruned.
- **All numeric, geospatial, and temporal constraints** are respected.
- Returned profiles are both **scientifically meaningful** and **user-aligned**.

Constraint Types and Operators

The system parses and applies constraints over fields extracted in step 4 (`intent_struct`), using deterministic validation operators:

Constraint Type	Metadata Field	Operator	Example
Temporal	`start_date`, `end_mission_date`	`[start, end]` $\subseteq$ $[T_1, T_2]$	Profiles after `2023-03-01`
Spatial (Named)	`dominant_region`, `regions_visited`	$\in$	"Arabian Sea"
Spatial (Lat/Lon)	`centroid_lat`, `centroid_lon`	`BBox.contains(pt)`	`[-10, 10], [65, 75]`
Tag-based	`tags`, `anomalies`	`token_match(tag)`	"Salinity Monitoring"
Numeric	`mission_duration_days`, `num_profiles`	`>`, `<`, `=`	Mission duration > 1000
Categorical	`sensor_name`, `model`	`==`	`"Seabird SBE41"`

Filtering Engine Implementation

Metadata filtering is implemented using a **rule-based pipeline** over document payloads returned from ChromaDB or SQL. Each candidate is passed through a **filter stack**, modeled as a sequence of logical predicates:

```
def validate(doc):
```

```
    return (
        doc["metadata"]["dominant_region"] == "Arabian Sea"
                            and    "Sensor    calibration    issue"    in
doc["metadata"]["special_features_or_anomalies"]
        and doc["metadata"]["start_date"] >= "2023-03-01"
    )
```

For vector search outputs, the system ranks and retains only documents **meeting ≥80% constraint satisfaction** (configurable threshold). For SQL outputs, results are **hard-filtered** at the query layer to guarantee strict satisfaction.

Constraint Resolution Examples

User Query: *"Show me long-term ARGO floats with salinity sensors in the Arabian Sea that reported anomalies."*

→ Constraints derived:

- `mission_duration_days` > 1500
- `tags` contains `"Salinity Monitoring"`
- `dominant_region` = `"Arabian Sea"`
- `special_features_or_anomalies` non-empty

→ Matching document:

```
{
  "float_id": "1902674",
  "mission_duration_days": 3185,
  "dominant_region": "Arabian Sea",
  "tags": ["Salinity Monitoring"],
    "special_features_or_anomalies":  ["Sensor  calibration  issue  on
2019-11-22"]
}
```

This multi-constraint enforcement step is **not optional**, especially when multiple candidates have high vector similarity but diverge semantically in mission scope, region, or instrumentation.

This validator is also responsible for **error propagation**: if no candidates satisfy the constraints, a structured clarification request is generated and routed back to the chat interface with suggestions (e.g., "No floats with nitrate anomalies in that region; would you like to expand search bounds?").

8. Query Result Consolidation and ARGO Float Identifier Extraction

Following successful filtering, the system proceeds to the **result consolidation** phase, where data products from different modalities (semantic, SQL, KB) are unified into a canonical response structure. This serves two purposes:

1. Deliver a **consistent response format** to downstream visualization and chatbot renderers.

2. Derive critical identifiers (e.g., `float_id`, `profile_id`) for further API chaining and drill-down queries.

Output Schema Normalization

Each selected result—regardless of retrieval origin—is cast into a normalized, typed data model, such as:

```
{
  "float_id": "1902674",
  "wmo_inst_type": "A12345",
  "deployment_region": "Arabian Sea",
  "mission_start": "2015-05-20",
  "mission_end": "2024-01-30",
  "duration_days": 3185,
  "sensors": ["Seabird SBE41"],
  "parameters": ["Temperature", "Salinity"],
  "anomalies": ["Sensor calibration issue on 2019-11-22"],
  "geo_summary": {
    "lat_range": [-10.5, 5.1],
    "lon_range": [60.0, 80.0]
  },
  "tags": ["Salinity Monitoring", "Long-Term Deployment"]
}
```

This schema is materialized as a **list of structured objects**, which can be ingested by multiple backend renderers:

- **Chat summarizer (LLM)** → Converts structured float metadata into human-readable summaries.

- **Geospatial mapper (Plotly/Leaflet)** → Uses `lat_range`, `lon_range`, `centroid_*`.

- **Dashboard interface (Streamlit)** → Renders profile-by-profile comparisons.

- **Drilldown link generator** → Enables the user to view all profiles from a float ID (e.g., `/float/1902674/profiles`).

Identifier Extraction

In this step, the system extracts and caches identifiers such as:

- `float_id` (e.g., "1902674")
- `profile_ids` (associated SQL rowkeys)
- `session_query_hash` (unique fingerprint for reproducibility)
- `source_retrieval_mode` (e.g., `"sql+vector"`, `"vector_only"`)

This metadata is persisted into Redis under the current `session_id`, enabling:

- **Jumpback capability** to revisit or refine earlier queries.
- **Drilldown operations**, e.g., *"Show me all profiles from float 1902674 with depth > 1000m".*
- **Cross-query reasoning**, e.g., comparing floats across missions, sensors, or anomalies.

Finally, the consolidated result object is passed downstream to the **post-processing module** (step 9), where format selection, visualization, and data export options (PNG, CSV, NetCDF) are triggered.

---

9. Context-Aware Query Refinement and Augmentation Leveraging LLM Feedback

At this stage, the system transitions from raw intent detection and preliminary entity extraction into **canonical query specification**. The goal is not to directly emit SQL but to produce an intermediate, schema-aware representation that is both **machine-validated** and **user-interpretable**.

The refinement loop leverages a schema-conditioned LLM (Gemini API) to normalize ambiguous user expressions into **deterministic query objects**. These objects explicitly encode temporal intervals in ISO-8601, spatial constraints in standardized geospatial formats (BBOX or GeoJSON), and parametric constraints mapped to database column names. Importantly, the model is forced to emit **strict JSON structures**, eliminating freeform narrative and enabling direct ingestion by backend validators.

To ensure robustness, all outputs undergo **formal schema validation** (e.g., via Pydantic) and **semantic checks**: invalid date ranges, missing units, or unsupported aggregation operators

trigger a clarification pathway. When ambiguity is detected (e.g., “near the equator” without bounding coordinates), the LLM sets a clarification flag, prompting the system to solicit explicit user input rather than hallucinating assumptions.

This refinement stage thus transforms a noisy linguistic input into a **canonical query specification language (CQSL)**, which becomes the contract between natural language and structured database operations. Beyond accuracy, this layer introduces **safety guarantees**: no direct LLM-to-SQL path exists, and only vetted parameters advance to the execution engine.

---

10. Automated Generation of Optimized SQL Queries from Refined Parameters

Following canonicalization, the system performs **safe, deterministic translation into SQL**. Unlike naive text-to-SQL pipelines, query generation here is **programmatic and index-aware**, designed to exploit relational optimizations and spatial indexing in PostgreSQL/PostGIS.

The translation process applies **operator templates** parameterized by the refined query object: temporal constraints translate into range predicates, spatial constraints into `ST_Within` or `ST_DWithin` calls, and parameter filters into equality or inequality clauses. To handle oceanographic workloads, queries frequently incorporate **common table expressions (CTEs)** to isolate candidate float profiles before aggregation.

Aggregation is treated as a first-class construct. User requests such as “mean salinity” or “trend in nitrate” are mapped to canonical aggregation functions (`AVG`, `STDDEV`, `COUNT`) with explicit grouping dimensions (e.g., float identifier, depth bins). These groupings are enforced to match scientific conventions, ensuring reproducibility across repeated queries.

Optimization is not static. Each generated SQL statement is subjected to **execution plan inspection** (`EXPLAIN/ANALYZE`). Plans are recorded and analyzed to detect pathological behaviors such as sequential scans over multi-terabyte NetCDF-derived tables. When inefficiencies are detected, the system automatically suggests **index creation, partitioning strategies, or materialized aggregates**. Optionally, condensed query plans are reintroduced into the LLM feedback loop, enabling adaptive query rewrites informed by execution statistics.

To preserve accountability, every query—along with its refined JSON specification, generated SQL text, and execution plan—is logged into an **audit table**. This enables downstream **reinforcement learning from human feedback (RLHF)** and system introspection, turning operational telemetry into future optimization signals.

---

11. Interactive 2D/3D Visualization and LLM-Driven Analytical Insights

Once structured query results are obtained, the system pivots from raw tabular outputs into **multi-modal analytical visualization**. The core design principle here is to fuse **geospatial interactivity** with **scientific interpretability**, ensuring that domain experts can explore ARGO float data dynamically rather than through static tables.

The visualization layer is two-tiered:

1. **Geospatial Rendering** – Profiles and float trajectories are projected in two or three dimensions using high-precision geospatial libraries. Point geometries derived from float locations are integrated into map frameworks (Leaflet/Cesium), enabling zoomable trajectory exploration, heatmap overlays, and temporal brushing. Bathymetric and regional boundaries are layered to contextualize float dynamics within oceanographic domains.

2. **Scientific Plotting** – Depth–time diagrams, temperature–salinity (T–S) plots, and biogeochemical cross-sections are automatically generated from structured queries. These plots preserve **scientific accuracy** by respecting units, quality control flags, and canonical parameter names. Advanced users may toggle between ensemble summaries (mean/variance envelopes) and single-float profiles, thereby scaling from macroscopic ocean basin insights to float-specific diagnostics.

An important innovation is the integration of **LLM-driven narrative overlays**. Instead of presenting raw figures alone, the system couples each visualization with an interpretive summary generated by the language model, e.g., “Salinity gradients increase toward the equatorial band in late summer, with float 1902674 showing anomalously high values compared to basin averages.” This capability reduces cognitive load for decision-makers who may not be specialists in physical oceanography.

By unifying geospatial rendering, time-series analytics, and natural-language explanations, the system elevates raw ARGO outputs into **explanatory visual knowledge artifacts**.

---

12. Advanced Reasoning Mode for Complex Scientific Inference

While most queries can be satisfied by direct retrieval and aggregation, certain research-grade use cases demand **higher-order reasoning** that spans beyond database joins. Advanced reasoning mode activates when the LLM detects either (a) user requests involving prediction, causality, or anomaly attribution, or (b) contextual ambiguity requiring simulation or model-based inference.

This mode involves a multi-step augmentation of the core pipeline:

- **Cross-Modal Synthesis** – Instead of limiting retrieval to SQL outputs, the system fuses tabular ARGO profiles, metadata tags, and vectorized summaries of historical literature (e.g., regional climatology, monsoon variability studies). The LLM performs a **retrieval-augmented synthesis**, integrating heterogeneous evidence into a coherent scientific explanation.

- **Pattern Extraction and Anomaly Attribution** – Temporal and spatial residuals are compared against climatological baselines derived from long-term ARGO archives. If float profiles deviate beyond statistical thresholds (e.g., >3σ anomaly in mixed-layer depth), the reasoning engine classifies the anomaly (instrumental vs. geophysical) and links it to potential drivers such as monsoon circulation, eddy activity, or sensor drift.

- **Hypothetical Modeling** – When users pose counterfactuals ("What if the float had drifted 5° further west?"), the reasoning mode extrapolates profiles into adjacent domains using geostatistical interpolation (kriging) or physics-guided regression models. These extrapolations are not deterministic predictions but **probabilistic scenarios**, explicitly annotated with confidence intervals.

Critically, advanced reasoning is **transparent**: outputs include traceable provenance metadata specifying which retrievals, baselines, or models contributed to the inference. This guards against the opacity of LLMs by grounding their higher-order reasoning in verifiable scientific data sources.

Thus, advanced reasoning mode transforms the system from a **query engine** into an **exploratory research assistant**, capable of hypothesis generation, anomaly attribution, and contextual synthesis within the oceanographic domain.

---

13. Anomaly Detection, Forecasting, and Trend Analytics

Beyond descriptive retrieval, the system incorporates a **diagnostic and prognostic analytics layer** designed to extend ARGO data utility into **early-warning, climate monitoring, and operational forecasting** domains.

1. **Anomaly Detection** – Incoming float profiles are continuously benchmarked against **climatological baselines** and historical float ensembles. Multivariate thresholds (temperature, salinity, density) are applied using robust statistical metrics such as **median absolute deviation (MAD)** and **empirical orthogonal function (EOF) outlier tests**. This ensures that anomalies reflect genuine oceanographic deviations rather than transient sensor noise. Each anomaly is **classified and tagged** as either:
    - *Instrumental* (e.g., calibration drift, pressure sensor offset).
    - *Geophysical* (e.g., intraseasonal Kelvin wave, eddy-induced salinity intrusion).
2. **Trend Analytics** – Time-series decomposition techniques (Fourier, wavelet, and seasonal-trend decomposition using LOESS) are applied to long-duration profiles. These reveal **basin-scale secular changes** (e.g., Arabian Sea salinity freshening) alongside **interannual variability signals** tied to ENSO or monsoon oscillations. Trends are stored as **metadata extensions** linked to floats, thereby enriching future retrievals with context-aware insights.
3. **Forecasting Layer** – While ARGO data alone cannot yield deterministic ocean forecasts, the system leverages **data-driven models** (vector autoregression, Gaussian process regression) and **physics-informed priors** (mixed-layer heat budget formulations) to project near-term scenarios. Forecasts are always accompanied by **confidence intervals** and **uncertainty maps**, explicitly stating the degree of reliability.

The value proposition of this module lies in converting raw ARGO float observations into **decision-relevant intelligence**, supporting stakeholders ranging from academic researchers to operational agencies monitoring fisheries, monsoon impacts, or coastal hazards.

14. Structured Outputs and Multi-Format Data Export

Scientific workflows demand not only analysis but also **portable, reusable outputs**. To this end, the system provides structured outputs across multiple modalities, each designed for distinct stakeholders:

- **Tabular Summaries** – Query results are normalized into CSV and SQL-ready tables, maintaining schema fidelity (column names, units, QC flags).
- **Geospatial Layers** – Float trajectories, profile centroids, and anomaly maps are exportable as **GeoJSON, NetCDF, and shapefiles**, compatible with GIS systems.
- **Publication-Ready Visuals** – Analytical plots (T–S diagrams, anomaly scatterplots, drift trajectories) are exportable in **high-resolution vector formats (SVG, PDF)** for direct inclusion in manuscripts.
- **Narrative Reports** – LLM-generated interpretive summaries are formatted into **LaTeX and Word-compatible documents**, enabling seamless integration into technical reports and research briefs.

A critical design choice is **traceability**: every exported file is stamped with **provenance metadata** (timestamp, float IDs, query signature, processing pipeline hash). This ensures that downstream users can reproduce or audit analyses — a cornerstone of scientific credibility.

---

15. Integration, Extensibility, and Human-in-the-Loop Oversight

The final layer emphasizes that the system is not a closed black box but a **modular, extensible framework** embedded within the larger oceanographic research ecosystem.

1. **Integration Hooks** – REST APIs, gRPC endpoints, and message-queue connectors allow seamless interfacing with external climate models, institutional databases, and national data repositories (e.g., INCOIS, NOAA, Euro-Argo).
2. **Extensibility** – New sensor modalities (e.g., biogeochemical floats with oxygen, nitrate, pH) can be onboarded with minimal schema adjustment, preserving backward compatibility.
3. **Human-in-the-Loop Oversight** – Analysts retain supervisory control: they can validate anomaly classifications, override automated forecasts, and append expert annotations. This ensures that **machine intelligence complements rather than replaces domain expertise**.

Thus, the pipeline culminates in a system that is **analytical, explainable, interoperable, and extensible**, bridging the gap between ARGO float raw data and actionable scientific knowledge.

THANK YOU