Engage 2

ChatMET

Natural Language Solution For Historical Meteorological Information

Final technical report

Engage 2 catalyst fund project

Coordinator: MetSafe

Consortium partners: N/A

Thematic challenge: TC3 Disruptive ATM system modernization

Edition date: 28/11/2025

The ChatMET project has been supported by the SESAR 3 Joint Undertaking and its founding members under the Grant Agreement nr. 101114648. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or SESAR 3 JU. Neither the European Union nor the SESAR 3 JU can be held responsible for them.

SUPPORTED BY sesar JOINT UNDERTAKING

Co-funded by the European Union

Authoring & approval

Author(s) of the document

Organisation name	Date
Maxime Warnier, MetSafe	30/09/2025

Reviewed by

Organisation name	Date
Kamel Rebai, MetSafe	30/09/2025
Engage 2 mentors, University of Belgrade – FTTE; University of Westminster	31/10/2025
Engage 2 Task Leader, University of Westminster	19/11/2025

Approved for submission to the SESAR 3 JU by[1]

Organisation name	Date
Micol Biscotto, Deep Blue srl	19/11/2025

Rejected by[2]

Organisation name	Date

Document history

Edition	Date	Status	Company Author	Justification
01.00	30/09/2025	Submitted to the Engage 2 consortium	Maxime Warnier	Initial version
02.00	28/11/2025	Ready for publication	Engage 2	Final version

[1] Representatives of all the beneficiaries involved in the project

[2] Representatives of the beneficiaries involved in the project

Engage 2

THE SESAR 3 KNOWLEDGE TRANSFER NETWORK

Engage 2

This document is part of a project that has received funding from the SESAR 3 Joint Undertaking under grant agreement No 101114648 under European Union's Horizon Europe research and innovation programme.

Table of contents

List of figures

1 Introduction

1.1 Abstract

Accessing meteorological data is fundamental to post-operational analysis, safety investigations, and airspace design within Air Traffic Management (ATM). However, current processes are often manual, inefficient, and require significant human interaction between ATM and meteorological services. The ChatMET project was initiated to investigate the feasibility of leveraging Large Language Models (LLMs) to create an intuitive, natural language interface for accessing these complex datasets. The core of the research focused on addressing the inherent reliability challenges of LLMs, such as model hallucinations. This investigation led to the identification and validation of a “tool-calling” agentic architecture where the LLM acts not as a source of knowledge, but as a secure translator that converts user queries into API calls to an authoritative source of truth. The project's primary outcome is a Technology Readiness Level (TRL) 3 proof-of-concept that serves as a foundational blueprint for the future development of trustworthy AI-assisted tools in ATM.

1.2 Executive summary

In the modern Air Traffic Management (ATM) ecosystem, post-operational analysis is a cornerstone of continuous improvement in safety, efficiency, and capacity planning. Activities such as justifying weather-related regulations, analysing the impact of adverse weather on specific sectors, or conducting search and rescue missions all depend on timely and accurate access to historical meteorological data. Despite the critical nature of this information, access remains a significant operational bottleneck. The process is poorly automated, often relying on a series of human interactions between ATM operators and meteorological service providers, including manual data gathering and email-based requests. This inefficiency not only consumes valuable expert time but also limits the scope and responsiveness of analytical activities. The emergence of LLMs, popularized by platforms like ChatGPT, presented a disruptive opportunity to fundamentally change this paradigm by enabling users to query vast data repositories using simple, natural language

1.2.1 Architectural Findings

The ChatMET project was launched with a fundamental research question: how can the natural language capabilities of LLMs be safely and reliably harnessed to query structured, time-series meteorological data within the constraints of the ATM environment.

Initial benchmarking of general-purpose LLMs quickly confirmed that their direct application was infeasible due to the unacceptable risk of model hallucinations (the generation of confident but factually incorrect information). This finding invalidated any direct application and shifted the project's focus from building a product to conducting foundational architectural research.

The project's primary scientific contribution is the deliberate selection and validation of a tool-calling agentic architecture. This paradigm redefines the role of the LLM from a knowledge source to a sophisticated translator. The LLM converts a user's natural language query into a structured API call, which is then executed against an external, authoritative data source (in our case, the pre-existing HistoMet platform).

The structured data returned by HistoMet is then presented to the user, with the LLM providing a natural language wrapper. This decoupling of language processing from deterministic data retrieval effectively eliminates data hallucination and ensures that every query is traceable and auditable. Alternative architectures, such as Retrieval-Augmented Generation (RAG), were deliberately rejected as ill-suited for the structured and time-series nature of meteorological data.

1.2.2 Key Results

The ChatMET project delivered a TRL 3 prototype that serves as a proof-of-concept, validating the core functionality and viability of the tool-calling concept. The research also yielded as the necessity of advanced data visualization for operational usability and the development of strategies for managing large-scale data queries that exceed LLM context limits.

1.2.3 Future Outlook

The architectural blueprint and the lessons learned during the ChatMET project represent our contribution to the state of the art in applying AI and specifically LLM usage within ATM. This work provides a pathway for future development. The next steps are to mature the ChatMET concept into an industrial and operational product, potentially through integration with established ATM portals like CDM@DSNA or as a service integrated into next-generation Air Traffic Controller (ATCO) decision support systems. The foundational research conducted in this project is the critical first step toward realizing that vision.

2 Overview of catalyst project

2.1 Operational/technical context

The operational environment of Air Traffic Management is increasingly data-driven. While real-time meteorological data has been effectively integrated into ATM operations through initiatives like the System-Wide Information Management (SWIM) services, access to historical meteorological data remains a significant challenge. This historical data is indispensable for a wide range of post-operational activities. For example, Air Navigation Service Providers (ANSPs) must analyse past weather events to justify the implementation of capacity-impacting regulations. Safety investigators require detailed meteorological reconstructions to understand incident causal factors. Furthermore, airspace designers use long-term climatological data to optimize route structures and sector configurations.

Currently, retrieving this information is a labour-intensive process, characterized by manual searches, inter-departmental requests, and a lack of automated, user-friendly tools.

The technical context for this project is shaped by the rapid democratization of Large Language Models. These sophisticated artificial intelligence systems have demonstrated an unprecedented ability to understand and generate human-like text, offering the potential for highly intuitive user interfaces. However, this potential is counterbalanced by the inherent unreliability of current LLM technology. Issues such as “hallucinations” where the model generates factually incorrect information with a high degree of confidence, and a lack of traceability for its outputs, make the direct application of LLMs in safety-critical domains like ATM a high-risk proposition.

This led to a main scientific challenge that the ChatMET project aims to solve: how to use the helpful features of large language models (LLMs) for users while making sure the system is reliable, secure, and verifiable, as required for air traffic management (ATM).

2.2 Project scope and objectives

2.2.1 Initial Scope and Use Cases

The objective is to develop and validate ChatMET, a natural language solution for accessing historical meteorological information, with the explicit goal of reaching a Technology Readiness Level (TRL) 3 prototype while trying to achieve a TRL 5 usable version.

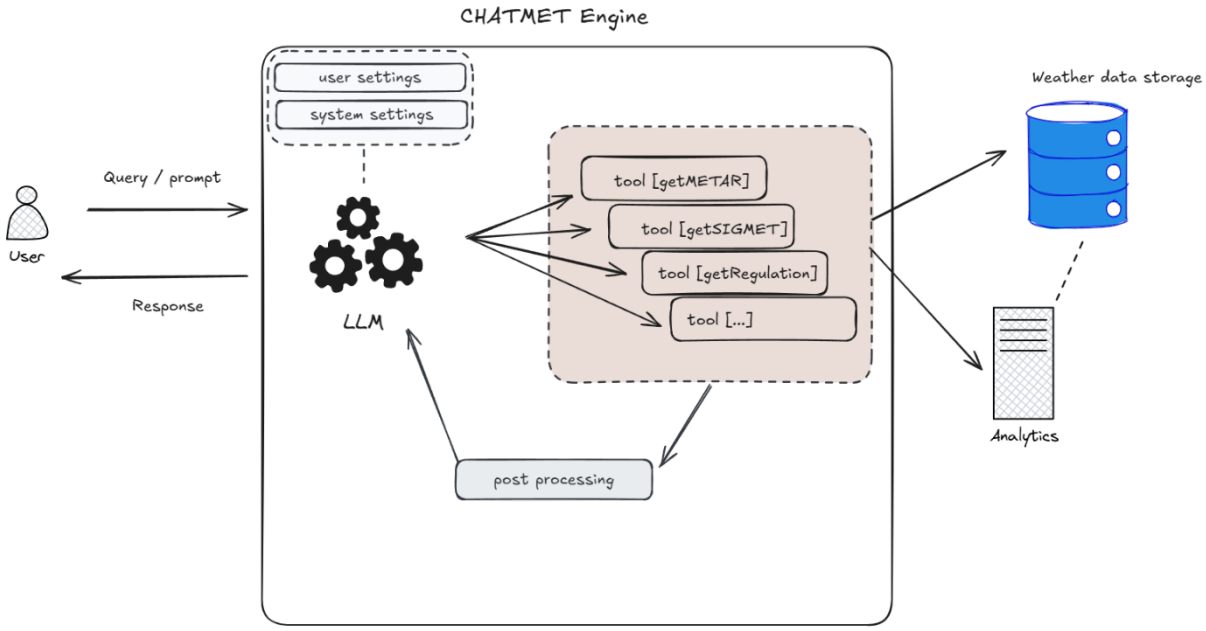


Figure 1: Overview of the first simplified architecture

The scope of the research was initially defined by a set of operationally relevant use cases, developed in consultation with ATM experts, primarily from the French ANSP, DSNA:

- **Use Case #1: Access to Basic Meteorological Data.** This use case focused on fundamental post-operational queries, such as retrieving OPMET data (e.g., METAR, SIGMET) for a specific airport or airspace over a defined period to support the analysis of sensible weather situations.
- **Use Case #2: Justification of Regulations**. This addressed the need for operators to efficiently gather all relevant weather observations (e.g., radar, satellite, SIGMETs) that justify the implementation of a specific weather-related air traffic flow regulation.
- **Use Case #3: Analysis of Weather Impact.** This use case involved more complex analytical queries, such as comparing the frequency of thunderstorms in a particular sector between two different years or identifying the most weather-impacted areas within a Flight Information Region (FIR) over a summer season.

For weather data storage source, we rely on our existing platform: **HistoMet**

HistoMet is a platform designed to provide access to up to ten years of validated meteorological data. This system serves as a reliable, authoritative data source, storing structured spatio-temporal data in a PostgreSQL/PostGIS database. For access to geographical information, HistoMet offers several APIs, including the WFS (Web Feature Service). For "gridded data" (weather data from numerical models), the platform uses NetCDF storage, complemented by a specialized API for optimized extractions on specific points, areas or trajectories.

2.2.2 Refined Research Objectives

As the project progressed and the complexities of integrating LLMs into the ATM domain became clearer, the initial scope evolved into a set of more specific objectives. The project's focus shifted from simply building a tool to investigating the principles of safe and reliable LLM integration. The refined objectives were:

- To conduct a comparative analysis of different LLM integration patterns, specifically evaluating their suitability for ensuring data fidelity and mitigating hallucination risks in the context of structured meteorological data.
- To design and prototype a system architecture that strategically decouples the non-deterministic, generative nature of the LLM from the deterministic, verifiable process of data retrieval from an authoritative source.
- To investigate the feasibility and advantages of utilizing smaller, specialized LLMs (e.g., models with fewer than 14 billion parameters) to enable on-premise deployment, thereby addressing the stringent security, privacy, and performance requirements of the ATM environment.
- To develop a TRL 3 proof-of-concept that serves as a tangible validation of the chosen architectural pattern and provides a foundational basis for future, higher-TRL development.

2.3 Research carried out

The research conducted under the ChatMET project followed an iterative methodology, progressing from initial hypothesis testing to the development and validation of a novel system architecture.

2.3.1 Initial Hypothesis and Benchmarking

The project's initial phase involved a benchmarking study to evaluate state-of-the-art LLMs, specifically to see if they could serve as a direct knowledge source for meteorological information. We tested models like ChatGPT with queries related to historical weather data.

Our tests provided clear evidence of the “hallucination” problem; the models frequently generated confident but incorrect information, inventing plausible-sounding METAR reports or fabricating data for specific dates and locations. This outcome showed that a simple approach of using a generic LLM as a direct source for ATM data was not viable. It highlighted that any effective system must use the LLM as a component to be carefully controlled within a more robust framework, rather than as a primary source of truth.

While large models like ChatGPT have implemented “guardrails” that allow them to say “I don't know” or state that they lack access to specific data, this isn't always the case for smaller, open-weighted models. These models, often lacking the sophisticated safety training of their larger counterparts, are more prone to generating plausible but false information. This highlights the need for a robust external framework to validate and control the LLM's output, especially for critical applications.

2.3.2 Architectural Investigation: The Rejection of RAG

Following the initial findings, the research focused on established architectural patterns for grounding LLMs in factual data. The most prominent of these is Retrieval-Augmented Generation (RAG). In a RAG system, a user's query is first used to retrieve relevant documents or text chunks from a knowledge base; this retrieved context is then provided to the LLM along with the original query to generate a factually-grounded answer.

However, the RAG architecture was unsuitable for the ChatMET use cases, for several fundamental reasons:

- **Data structure**: RAG is optimized for unstructured or semi-structured text-based sources. The meteorological data in the HistoMet platform is highly structured time-series data. Applying RAG would require converting this data into text, an inefficient process that would lead to a loss of precision.
- **Lack of Scalability for Complex Data Types**: The project's scope includes the future integration of complex formats like gridded numerical weather prediction data or 3D vectorial data (e.g., turbulence volumes). RAG is fundamentally ill-equipped to handle such data types.
- **Lack of Precision and Verifiability**: RAG systems are prone to summarization errors or misinterpretations of the retrieved context. For ATM applications, where precision is paramount (e.g., the exact wind speed or visibility value), a system that performs direct, unambiguous queries against a structured database is vastly superior. RAG could not guarantee the 100% data validity required.

Based on this analysis, the RAG pattern was deliberately rejected.

2.3.3 The Tool-Calling Concept

The core contribution of the ChatMET project is the adoption and validation of the “tool-calling” (also known as function-calling) concept as the baseline solution. This paradigm establishes a clear and robust separation of concerns, leveraging the strengths of the LLM for language understanding while relying on a deterministic system for data integrity [arXiv:2504.00914].

The workflow of the tool-calling architecture is as follows:

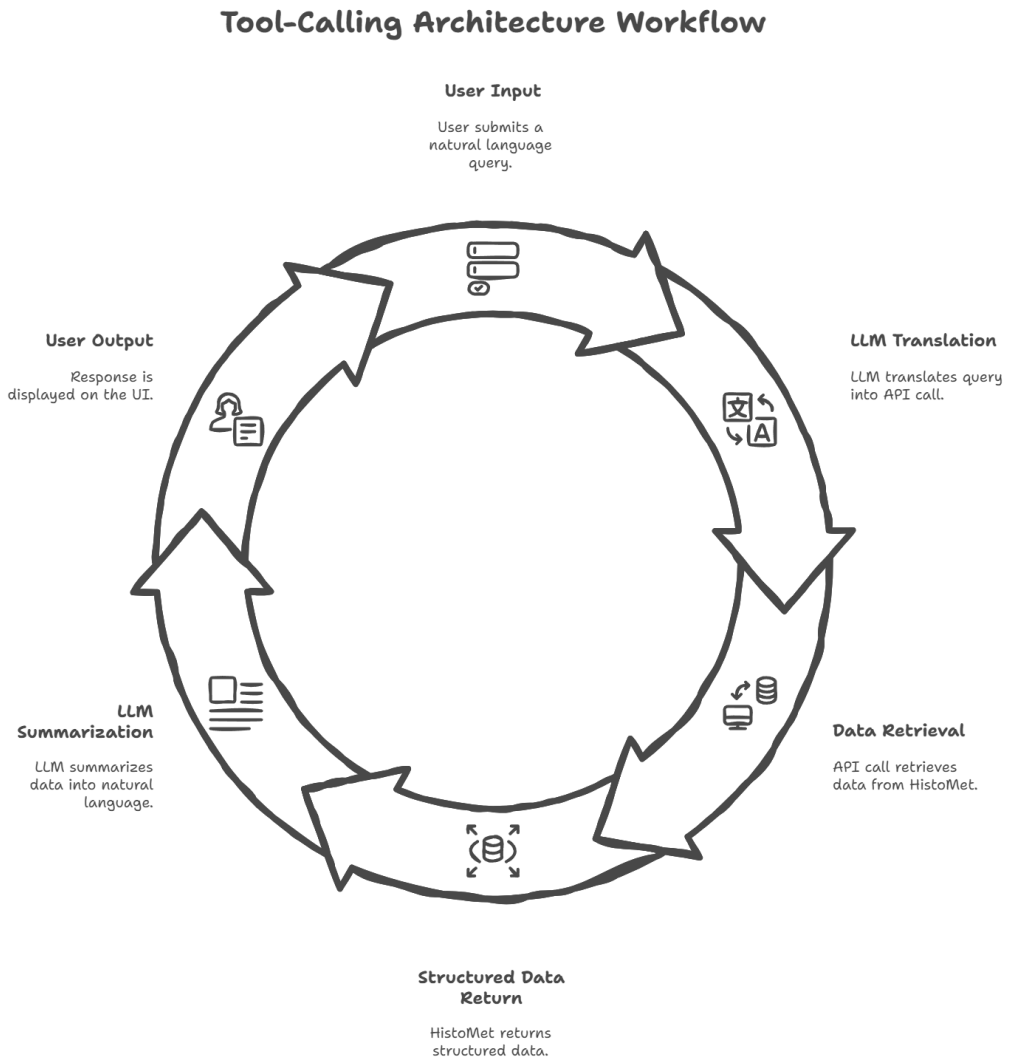


Figure 2: Simplified tool-calling Architecture Workflow

Natural Language Input: The user submits a query in natural language from the user interface, for example:

"Show me the SIGMETs for severe turbulence issued over the Bordeaux FIR yesterday"

LLM as Translator: The LLM's primary role is to parse this natural language input and translate it into a structured, machine-readable API call. It identifies the user's intent and extracts the necessary parameters (e.g., data type, location, time period).

For the example query, the LLM would generate a call like:

getData(type="SIGMET", area="LFBB", startTime="2024-12-24T00:00Z", endTime="2024-12-24T23:59Z",filterType:"type",filterValue:"TURB")

Execution against Authoritative Source: This structured API call is then executed against the HistoMet platform. *HistoMet is a pre-existing, validated database containing up to ten years of decoded OPMET data and other meteorological products from authoritative sources like Météo-France, Meteorage. The query is executed using standard, deterministic database logic.*

Structured Data Return: HistoMet returns a precise, structured data output (e.g., a JSON object containing…) to the ChatMET engine.

LLM as Summarizer: The structured data output is passed back to the LLM, which then transform this factual data in a user-friendly, natural language response.

This architecture is not merely a technical implementation; it is a framework for establishing trust in an inherently non-deterministic technology. The LLM is effectively placed in a "sandbox", prevented from ever acting as a source of factual data. Its role is confined to interpretation and presentation. Every piece of data shown to the user can be traced back to a specific, auditable API call made to a validated source. This creates a system that is trustworthy by design, not by attempting to force trustworthiness onto the LLM itself, but by building a reliable system around it.

2.3.4 Investigation into specialized agentic LLMs

In order to meet the requirements for a secure, on-premise infrastructure, ChatMET opted for Small Language Models (SLM), specifically models with a parameter count below 14 billion.

This implementation decision was driven by two primary factors: operational efficiency and architectural strategy. The use of SLMs ensures a reduced computational footprint and lower operational costs for on-premise deployment, which is a prerequisite for compliance with stringent security and privacy standards.

Fundamentally, this selection served as the cornerstone for an agentic architecture. By moving away from a monolithic, generalized model, the system facilitates the orchestration of specialized agents. This paradigm allows for the decomposition of complex tasks and their parallel execution by dedicated SLMs, effectively mitigating the significant computational overhead and performance bottlenecks inherent in a single, large-scale model.

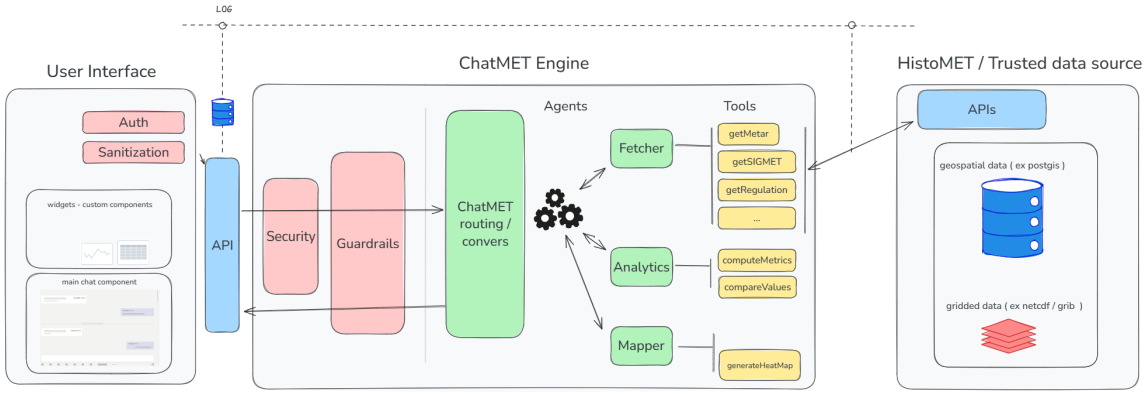


Figure 3: ChatMET Full Architecture illustration

2.3.4.1 A Modular, Agent-Based Architecture

This approach of breaking down the system into smaller entities is directly inspired by agent-based architecture concepts. Each module is designed as an autonomous agent, endowed with a specific objective and a defined scope of capabilities. This philosophy paves the way for a modular and resilient system where specialized models collaborate. It enables fine-grained upgrades and precise scoping of each module while reducing the non-determinism often associated with LLM-driven architectures.

2.3.4.2 Specialization, Orchestration, and Communication

The core of this architecture is specialization. Each agent is conditioned for a specific task (e.g., data retrieval, data analysis, interacting with a specific API), which justifies and enables the use of significantly lighter language models.

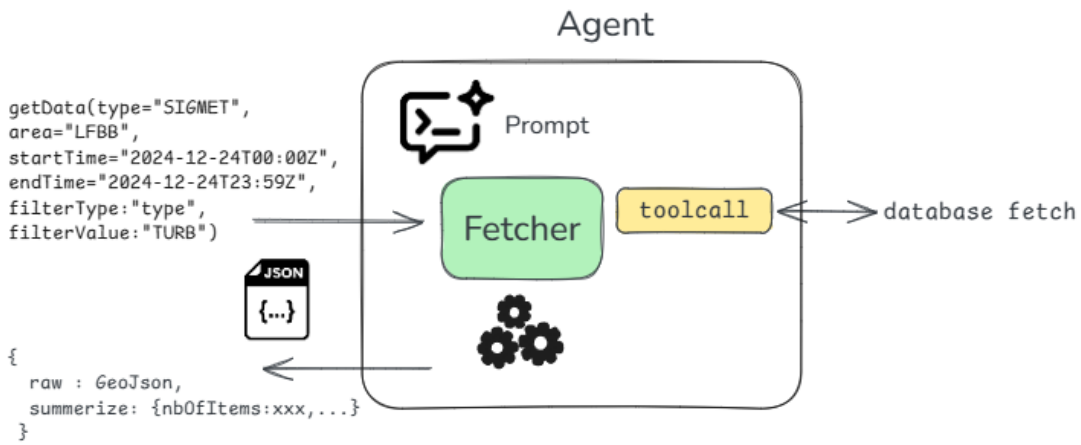


Figure 4: Agent "fetcher" illustration

A main agent, or orchestrator, is responsible for analyzing the user's initial request and delegating it to the most relevant agent or sequence of agents. If a task requires a collaboration of skills, agents can also call upon each other.

To perform concrete actions, agents have a set of tools they can invoke as we have seen previously in the tool-calling concepts. Inter-agent communication is standardized and conducted exclusively through structured text messages in JSON format. This format ensures perfect interoperability, as it is natively understood and generated by LLMs. It is also easier to log.

2.3.4.3 Asynchronous data flow and dual output

To optimize the user experience and performance, agents produce a dual output stream upon completing their execution:

- A light and concise summary intended for integration into the context of the main conversational LLM that interacts directly with the user. This text allows it to formulate a natural and relevant final response.
- A structured data (e.g., GeoJSON objects, numerical tables) that is sent directly to the front-end to be displayed in dedicated widgets.

This second output intentionally bypasses the conversational LLM. Integrating complex and dense data like geographic coordinates into an LLM's context would not only be inefficient in terms of performance but would also risk introducing interpretation errors (hallucinations).

The entire process is orchestrated asynchronously, and data is pushed to the interface via streaming as it becomes available. This approach ensures a fluid and responsive user interaction, even when complex tasks are running in the background.

This approach prepares a modular, resilient system in which specialized models collaborate, enabling fine-grained upgrades and scoping of each module while reducing the non-determinism typically associated with LLM-driven architectures.

2.3.5 The role of fine-tuning as an optimization vector

With the tool-calling architecture established, the potential role of fine-tuning was evaluated as a vector for optimization rather than a prerequisite for core functionality. The theoretical objective of fine-tuning within the ChatMET project was not to teach the LLM meteorology from first principles, but to improve its fluency in the specific dialect of ATM and enhance the precision of its generated tool calls.

The standard process for such fine-tuning utilizes a dataset of prompt-completion pairs designed to train the model on domain-specific terminology and common query patterns. For example:

Prompt: "Show me the SIGMETs for thunderstorms in the Paris ACC last summer."

Completion (Tool Call): get_sigmet("LFFF", "2023-06-01", "2023-08-31", event_type="TS")

This targeted training can help an LLM better map user intent to a specific function and refine its ability to generate clear, contextually appropriate natural language summaries from the structured data returned by an API.

2.3.6 Strategic focus on a flexible, Prompt-driven model

A pragmatic evaluation led us to prioritize a flexible, prompt-driven model over a dedicated fine-tuning pipeline. This decision was driven by two critical factors: the evolution of LLM capabilities and the imperative of long-term project maintainability. With the latest 2025 models demonstrating vastly superior instruction-following abilities [arXiv:2503.01743], the performance gap that fine-tuning once filled has significantly narrowed. Furthermore, we chose to avoid the architectural rigidity inherent to fine-tuning that would tightly couples a model to specific tool schemas and necessitates costly retraining cycles for any future updates. In contrast, a prompt-centric approach ensures essential agility.

Consequently, our development efforts were redirected toward iteratively refining instructional prompts and engineering a resilient agent logic with robust error handling. To validate the efficacy of our prompts, we began by evaluating various candidates, a process that led to the development of a lightweight testing framework. This framework is built with python and the langchain library to handle to different LLM model calls. The outputs are exported in a json file that can easily by visualised. For that we added a standalone viewer as a web app to analyse the outputs from the tests.

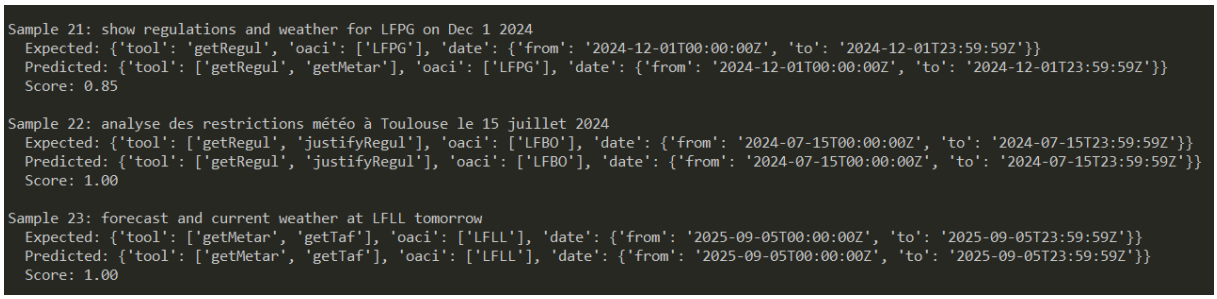


Figure 5: Output from the CLI testing tool

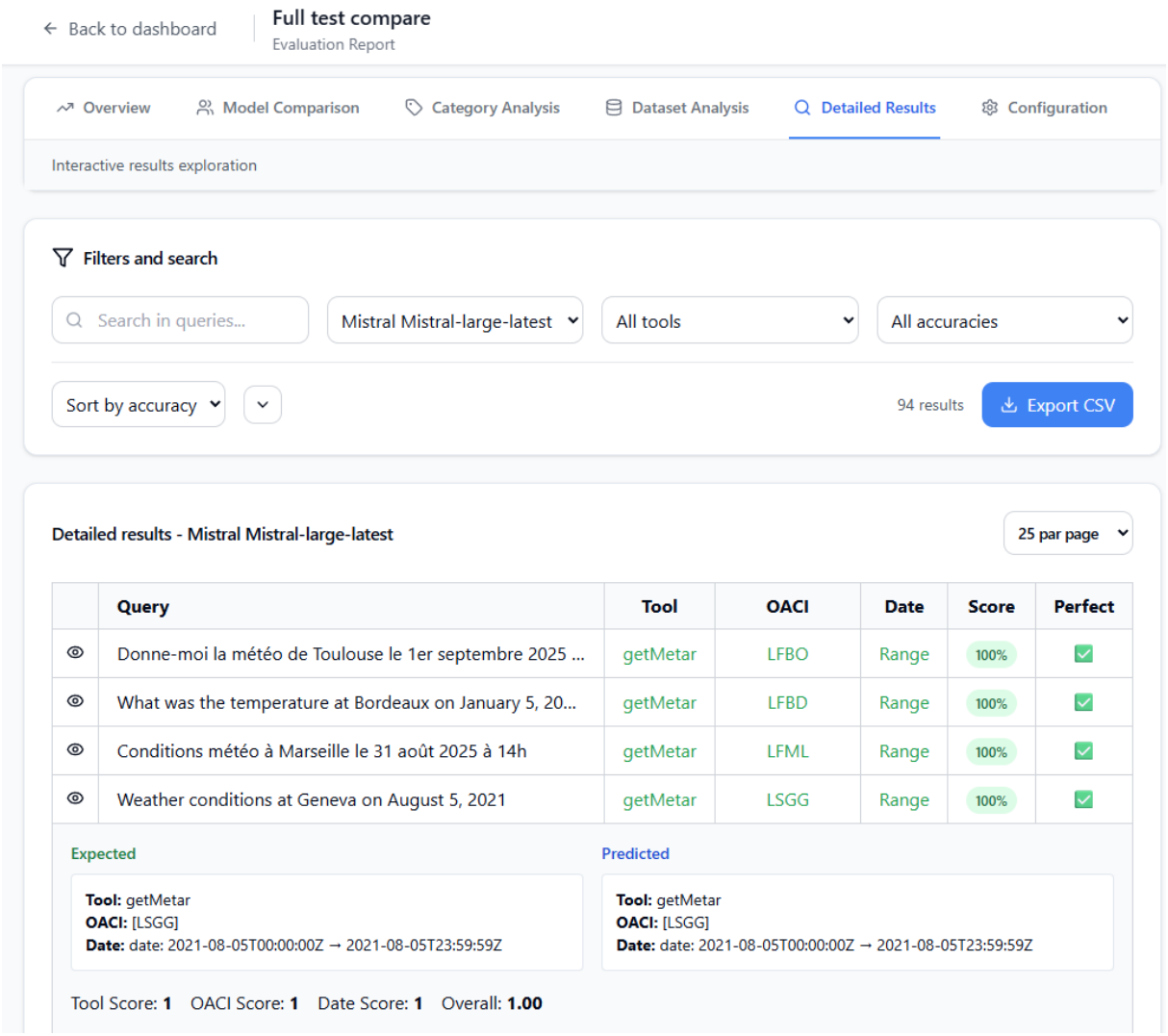


Figure 6: Screenshot of the viewer developed to analyse the test outputs

Within this framework, we systematically assess the ability of different model types and prompt structures to manage critical requirements such as strict output formatting, date handling, ICAO airport code recognition, and proper tool selection.

Model	Accuracy	Overall Score	Perfect Matches	Total Samples	Tool Score	OACI Score	Date Score
Mistral Mistral-large-latest	36.2%	76.1%	34/94	94	77.0%	83.0%	66.0%
Mistral Mistral-small-latest	34.0%	75.7%	32/94	94	78.0%	83.0%	63.8%
Ollama Llama3.1:8b	30.9%	75.4%	29/94	94	81.7%	80.9%	61.7%
Openai Gpt-4o	34.0%	73.6%	32/94	94	76.4%	78.7%	63.8%
Ollama Mistral-nemo	31.9%	73.6%	30/94	94	82.3%	77.7%	59.6%
Ollama Qwen3:14b	29.8%	73.4%	28/94	94	76.4%	79.8%	61.7%
Openai Gpt-4o-mini	29.8%	71.9%	28/94	94	77.1%	75.5%	61.7%
Ollama Phi4-mini	14.9%	62.1%	14/94	94	77.5%	69.1%	37.2%

Figure 7: Score overview of different model testing. Highlight on locally tested llama3.1 8b outperforming gpt-4o

The results demonstrate that smaller, more efficient models can perform exceptionally well on these targeted tasks, often rivalling the performance of counterparts that are hundreds of times larger. This finding reinforces our strategy, confirming that meticulous prompt engineering can unlock high-level capabilities without relying on massive-scale models.

2.3.7 Addressing visualization

Early user feedback and prototyping highlighted an important operational requirement: the Human-Machine Interface (HMI) is not merely a display layer but an integral part of the analytical tool. Feedback indicated that for post-operational analysis, interactive graphical representations, maps, and structured tables are often more valuable than purely text-based responses. This elevated the importance of a tightly integrated, flexible HMI capable of rendering complex visualizations.

To address this, our architecture was designed so that the LLM produces more than just conversational text. It is engineered to generate structured data payloads, specifically in JSON format, alongside its natural language summaries. This JSON object acts as a precise set of rendering instructions for the HMI. For instance, a query about convective activity along a flight path would not only yield a textual summary but also a JSON payload defining a map view, the route's coordinates, and the specific weather layers (e.g., SIGMETs, thunderstorm cells) to be overlaid. Similarly, a request to compare landing conditions at several airports would generate a JSON object structured to populate a comparative table.

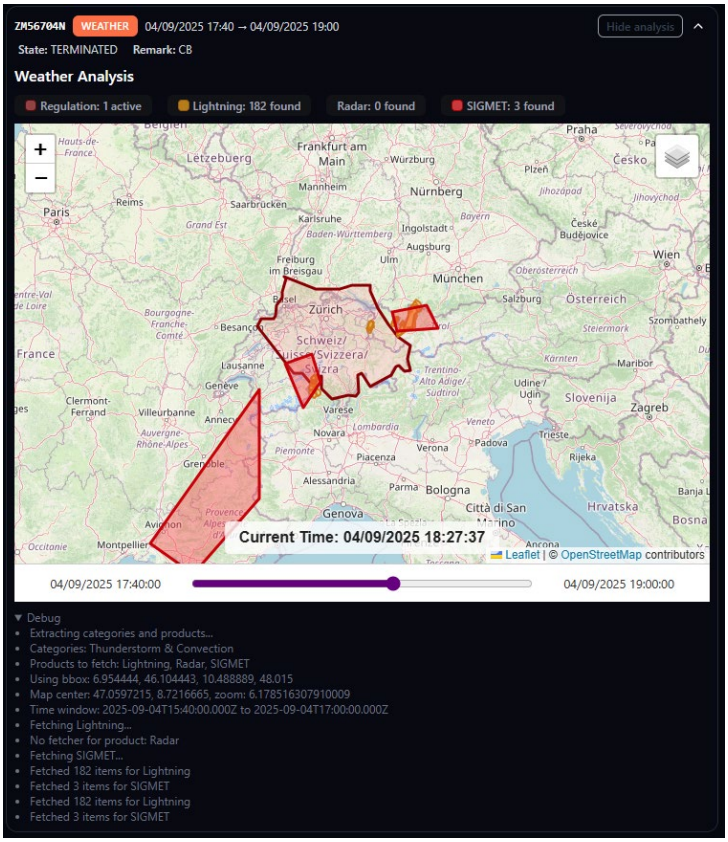


Figure 8: Widget for regulation analysis showing debug logic and interactivity

This approach decouples the agent's reasoning core from the front-end presentation logic. The LLM's role is to understand the user's intent and package the necessary data and visualization commands. The HMI's role is to interpret these commands and render the corresponding interactive components. HMI developments could also include an 'expert mode' where users can inspect and validate the generated data payloads, providing a feedback loop for system improvement.

2.4 Results

A Blueprint for AI LLM usage in ATM

The primary achievement of this project was developing a product that serves as a reference model for integrating AI technologies like LLMs into the ATM domain, especially for weather data. This work helped to mitigate initial risks associated with using these technologies in a safety-critical environment. As a result, the project was able to focus on developing a rigorous testing process.

Validation was a critical phase for both technical performance and user adoption. The system was tested to verify its ability to translate natural language queries into accurate API. This process led to the creation of a validation framework that will be made available to the community.

Operational Feedback and HMI Insights

Initial feedback from operational experts confirmed the relevance of our use cases and the intuitive nature of the natural language interface. However, it quickly became clear that text-only responses are insufficient for post-operational analysis because they lack the necessary context. The team learned that robust data visualization and illustration are core requirements for true usability and trust in reporting across all levels of the system.

While this new layer of complexity has postponed our initial ambition to deliver a fully autonomous and demonstrably functional prototype by the end of the project, the finding provides a solid, validated foundation for defining the project's next steps and future development.

3 Conclusions, next steps and lessons learned

3.1 Conclusions

The ChatMET project met its core research objective: to identify, develop, and validate a trustworthy architectural pattern for integrating Large Language Models with historical meteorological databases in Air Traffic Management. By using a “tool-calling” architecture, the project demonstrated that the power of natural language interfaces can be safely harnessed while mitigating the risks of data hallucination.

While the initial ambition was to reach a TRL 5 usable product, the final TRL 3 proof-of-concept represents a necessary outcome. The early research phases revealed that the problem was more complex than anticipated. Simple, off-the-shelf solutions and architectures like RAG were found to be unsuitable. This required a pivot toward foundational research to focus on two crucial areas: rigorous technical validation to ensure a reliable core architecture, and a deep dive into HMI requirements to confirm that data visualization was key for user acceptance.

This TRL 3 achievement provides the scientific and technical foundation needed to confidently build a more advanced TRL 5 system. Without this crucial foundational work, attempting to reach TRL 5 would have been premature, resulting in an unreliable and unusable system.

3.2 Next steps

The completion of the ChatMET project provides a roadmap for future work, with a long-term vision of transitioning this research into an operational capability.

3.2.1 Maturation to TRL 5/7

The immediate next step is to advance the prototype from TRL 3 to TRL 5. This will involve several key activities:

- Prototype Hardening: Evolving the current proof-of-concept into a more robust and feature-complete application.
- Expansion of Toolset: Integrating a wider range of meteorological data types and analytical functions through the development of new API tools and agents. This includes handling more complex data like satellite data and gridded numerical weather prediction outputs.
- HMI Development: Improve our prototype interface based on the feedback from operational experts to make it operational.
- Developing the core agent's ability to handle ambiguity and prompt users for clarification when a query is incomplete.
- Formal Validation: Conducting a formal user-in-the-loop validation exercise. This will involve comparing the efficiency and effectiveness of ChatMET against baseline methods (e.g., static forms, manual data requests) for a series of realistic post-operational analysis tasks.

- Operational Monitoring Tools: Building a dedicated monitoring interface to provide full observability into the routing agent's behavior, allowing for easier debugging and performance analysis.

3.2.2 Future Research

The project has opened several avenues for future research that can further enhance the system's capabilities:

Integration of new data sources: expanding the system's knowledge base by integrating additional data sources, such as real-time flight data, airport operational data, and advanced meteorological reanalysis datasets like ERA5. This would enable more sophisticated cross-domain analysis.

Advanced agent specialization (beyond prompting): building upon the initial success of prompt-engineered agents, future research must tackle the challenge of achieving specialization through fine-tuning on increasingly lightweight models. This research stream will focus on methods like Parameter-Efficient Fine-Tuning (PEFT) to adapt smaller LLMs to specific tasks, ensuring they deliver superior precision and performance while maintaining an ultra-low computational footprint for distributed operational environments.

Integration and Industrialization

The ultimate goal is the operational deployment of the ChatMET service. The recommended path forward is to release it either as a standalone tool, integrate it into existing systems such as ATM portals used by ANSPs or organizations like EUROCONTROL, or deploy it as a plugin that can be integrated into enterprise solutions like Mattermost.

Further, the techniques developed can be applied beyond post-operational analysis. We can collaborate with ATC system manufacturers to explore integrating a ChatMET-like AI assistant directly into controller working positions or decision-support tools. This would bring the capability right into the operational workflow, supporting more dynamic decision-making.

3.3 Lessons learned

The experience gained through the project and the SESAR Engage 2 catalyst framework led to the following:

3.3.1 Technical Lessons Learned

For data-critical applications, the design of the integration architecture is more important than the specific choice of LLM. The tool-calling pattern provides the necessary control, verifiability, and safety guarantees that are absent in more direct LLM implementations. **The focus should be on building a reliable system around the LLM**.

Naively passing large volumes of data through an LLM's context window is technically and economically infeasible. A hybrid data management strategy, combining server-side pre-processing and

summarization with direct-to-UI rendering of raw data, is essential for building scalable and responsive systems.

3.3.2 Operational Lessons Learned

For meteorological and other complex, multi-dimensional data, the value for the end-user is often in the visualization, not the text. **The HMI should not be considered an afterthought but a core component of the solution**, co-designed with the backend to enable rich, interactive data exploration.

3.3.3 Project Management Lessons Learned

The catalyst funding approach proved to be highly effective for this type of exploratory, high-risk/high-reward research. It provided the flexibility for the project to pivot its technical approach based on early scientific findings. Specifically, treating the hallucination problem without being rigidly constrained by an initial technical plan. This agility was key to the project's success to evolve to deliver a meaningful result.

4 Dissemination

To ensure we share only consolidated results, our dissemination will occur in the final phase. We will begin by releasing our testing framework and visualization tool as open-source projects on GitHub. This release will be announced on LinkedIn and will occur just before we attend the SESAR Innovation Days 2025, where we will present our findings and engage with the community.

- ChatMET LLM Testing Framework
  - A framework designed to test the performance and reliability of the LLM in the ChatMET context.
  - *Source code available at:* https://github.com/MetSafe-atm/chatmet-test-engine

- ChatMET Report Viewer
  - A dedicated user interface for analyzing outputs from the testing framework, provided with sample testing datasets.
  - *Source code available at:* https://github.com/MetSafe-atm/chatmet-report-viewer

5 References

5.1 Project outputs

Internal Deliverable

- ChatMET Engine Prototype
  - An internal prototype of the core engine, integrated with the UI analytics platform for initial testing and validation. This deliverable is confidential.

Public Outputs

- LLM Software Requirements Specification
  - A document detailing the baseline requirements for integrating Large Language Models within Air Traffic Management (ATM) applications.

- Article on Project Findings
  - An article detailing the project's methodology and key outcomes.
  - *[Forthcoming - To be published on social media (LinkedIn)]*

5.2 Other

References used to study the ChatMET architecture

References

[1] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le et D. Zhou, «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,» chez *Advances in Neural Information Processing Systems 35*, 2022.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser et I. Polosukhin, «Attention is All You Need,» chez *Advances in Neural Information Processing Systems 30*, 2017.

[3] B. Shneiderman, «Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy,» *International Journal of Human–Computer Interaction,* vol. 36, p. 495–504, 2020.

[4] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda et T. Scialom, «Toolformer: Language Models Can Teach Themselves to Use Tools,» *arXiv preprint arXiv:2302.04761,* 2023.

[5] J. Lee, H. Su, D.-Y. Ji et T. Minami, «Engineering artificial intelligence: framework, challenges, and future direction,» *Machine Learning: Engineering,* vol. 1, p. 013001, July 2025.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et others, «Language Models are Few-Shot Learners,» chez *Advances in Neural Information Processing Systems 33*, 2020.

[7] E. M. Bender, T. Gebru, A. McMillan-Major et S. Shmitchell, «On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?,» chez *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, 2021.

[8] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins et others, «Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,» *Information Fusion,* vol. 58, p. 82–115, 2020.

[9] European Union Aviation Safety Agency (EASA), «EASA Artificial Intelligence Roadmap 2.0,» 2023.

The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models : https://gorilla.cs.berkeley.edu/leaderboard.html

6 List of acronyms

Acronym	Description
AI	Artificial Intelligence
ANSP	Air Navigation Service Provider
API	Application Programming Interface
ATC	Air Traffic Control
ATCO	Air Traffic Controller
ATM	Air Traffic Management
DSNA	Direction des Services de la Navigation Aérienne
FIR	Flight Information Region
HMI	Human-Machine Interface
JSON	JavaScript Object Notation
LLM	Large Language Model
METAR	Meteorological Aerodrome Report
OPMET	Operational Meteorological Information
PEFT	Parameter-Efficient Fine-Tuning
RAG	Retrieval-Augmented Generation
SESAR	Single European Sky ATM Research Programme
SIGMET	Significant Meteorological Information
SLM	Small Language Model
SWIM	System-Wide Information Management
TRL	Technology Readiness Level
WFS	Web Feature Service

Appendix A System/Software Requirements Specification - AI/LLM Assistant for ATM

Version 0.1

A.1 High-Level System Requirements (SYS-HLR)

- **REQ-SYS-HLR-001:** The system shall provide a natural language interface for users to query authorized Air Traffic Management (ATM) data sources.
- **REQ-SYS-HLR-002:** The system shall ensure that all factual data presented to the user is exclusively sourced from predefined, authoritative data systems via a deterministic software interface.
- **REQ-SYS-HLR-003:** The system shall operate entirely within a secure, private network infrastructure, with no data transmission to or from public networks or third-party services.
- **REQ-SYS-HLR-004:** The system shall provide a complete and verifiable trace from any data presented to the user back to the specific query executed against the authoritative data source.
- **REQ-SYS-HLR-005**: The system shall be capable of operating in a gracefully degraded mode. In the event of a critical failure of the LLM/NLP component, the system shall fall back to a structured, form-based query interface that allows users to directly execute the underlying data tools.

A.2 Performance & Availability (SYS-PRF)

- **REQ-SYS-PRF-001:** The system shall provide a response to 95% of standard user queries defined in seconds (end-to-end latency, from query submission to response display).
- **REQ-SYS-PRF-002:** The system shall have an availability rate defined.
- **REQ-SYS-PRF-003:** In the event an authoritative data source is unavailable, the system shall clearly inform the user and reject any query targeting that source, rather than providing an incomplete or erroneous answer.
- **REQ-SYS-PRF-004:** The system shall be architected to support **X concurrent active users** while maintaining the latency specified in REQ-SYS-PRF-001, where X is a configurable parameter defined based on expected operational load.

A.3 Software Requirements (SWR)

A.3.1 Natural Language Processing (SWR-NLP)

- **REQ-SWR-NLP-001:** The software shall parse a user's natural language input to identify a primary operational intent from a predefined set of supported intents.
- **REQ-SWR-NLP-002:** The software shall perfectly extract specific entities from the user's input, including but not limited to: ICAO location identifiers, date ranges, time ranges.
- **REQ-SWR-NLP-003:** The software shall normalize extracted date and time entities to Coordinated Universal Time (UTC).
- **REQ-SWR-NLP-004:** If the user's input is ambiguous and cannot be resolved to a single intent or a complete set of required entities, the software should prompt the user for clarification by offering specific, context-aware choices (e.g., suggesting known airport codes or regions).
- **REQ-SWR-NLP-005:** The software shall detect and reject any user query whose identified intent is not on the list of supported operational intents.
- **REQ-SWR-NLP-006:** The software shall manage the context of a multi-turn conversation session. Pronominal references (e.g., "what about for that airport?") and follow-up queries shall be correctly resolved within the immediate conversational context.

A.3.2 Tool Calling & Execution (SWR-TCE)

- **REQ-SWR-TCE-001:** The software shall map each supported operational intent to a single, corresponding software function (a "tool").
- **REQ-SWR-TCE-002:** The software shall construct a structured, syntactically correct function call using the entities extracted from the user's input as parameters.
- **REQ-SWR-TCE-003:** The software shall validate the data type and format of each parameter before executing the function call.
- **REQ-SWR-TCE-004:** The software shall execute the validated function call against the corresponding internal API endpoint.
- **REQ-SWR-TCE-005:** The software shall capture and log any error state returned by the API endpoint during function execution.

A.3.3 Data Handling & Response Generation (SWR-DHR)

- **REQ-SWR-DHR-001:** Upon receiving a structured data response from an API, the software shall pass this data to the LLM to generate a natural language summary.
- **REQ-SWR-DHR-002:** The software shall not pass raw data payloads exceeding a configurable size limit (e.g., 4096 tokens) into the LLM's context for summarization.

- **REQ-SWR-DHR-003:** For queries that return a data set larger than the configured limit, the software shall perform server-side statistical pre-processing (e.g., calculating min, max, mean) and pass only the result of this pre-processing to the LLM.
- **REQ-SWR-DHR-004:** The software should provide a mechanism for the user to download the complete, raw data set in CSV format.
- **REQ-SWR-DHR-005:** The final output presented to the user shall be composed of the LLM-generated summary and the structured data (e.g., table, chart) received from the tool.
- **REQ-SWR-DHR-006:** Any data presented as a summary or as the result of statistical pre-processing shall be accompanied by an explicit and visible notice indicating that the information is a summary and not the complete dataset.

A.4 Safety Requirements (SAF)

- **REQ-SAF-HLR-001:** The system shall be designed to prevent the LLM from generating or altering factual data. Its role shall be limited to query interpretation and summarization of tool-provided data.
- **REQ-SAF-LLR-001:** The system shall implement a "pass-through" mechanism for raw data, ensuring that the data retrieved from the authoritative source is displayed to the user without modification by the LLM.
- **REQ-SAF-LLR-002:** In the user interface, the LLM-generated text summary shall be visually and textually distinct from the display of factual, raw data.
- **REQ-SAF-LLR-003:** The system shall have a deterministic mechanism to reject the execution of any tool call that is malformed or contains invalid parameters, preventing unintended operations.
- **REQ-SAF-LLR-004:** The system shall integrate a semantic monitoring mechanism that, where feasible, compares the LLM-generated summary against the raw factual data to detect blatant contradictions. If a contradiction is detected, the response shall be blocked, and a standard error message shall be displayed.
- **REQ-SAF-LLR-005:** The user interface shall apply standardized and unambiguous color-coding and labels to differentiate information types (e.g., one for raw data, another for AI-generated summaries, and another for system warnings).

A.5 Security Requirements (SEC)

- **REQ-SEC-AUT-001:** The system should require user authentication before granting access to the natural language interface.
- **REQ-SEC-INP-001:** The software shall sanitize all user input strings to remove executable scripts or characters that could trigger injection vulnerabilities.

- **REQ-SEC-API-001:** All internal API communications between the orchestration service and data tools shall be encrypted using Transport Layer Security (TLS) 1.2 or higher.
- **REQ-SEC-API-002:** All internal API endpoints shall require a valid authentication token for access.
- **REQ-SEC-LOG-001:** The system shall maintain an immutable audit log of all tool calls executed, including the function name, parameters, and a timestamp. The user's original natural language query should not be logged.

A.6 Human-Machine Interface (HMI) Requirements

- **REQ-HMI-DSP-001:** The HMI shall provide a single text input field for the user to enter their natural language query.
- **REQ-HMI-DSP-002:** The HMI shall display the system's response in a structured layout, containing separate, clearly identified areas for the natural language summary, the structured data (e.g., table), and any data visualizations.
- **REQ-HMI-FDB-001:** The HMI shall provide a visual indicator to the user when a query is being processed.
- **REQ-HMI-FDB-002:** The HMI should include a simple user feedback mechanism (e.g., "thumbs up / thumbs down" buttons) for each generated response.
- **REQ-HMI-TRC-001:** The HMI should allow an authorized user to view the specific, structured tool call that was generated from their natural language query.
- **REQ-HMI-HLP-001:** The HMI may provide example queries or a list of supported capabilities to guide the user.
- **REQ-HMI-HLP-002:** The HMI may provide suggested follow-up actions or clarification choices in response to the LLM's output

A.7 LLM Model Selection and Lifecycle Management (SWR-LLM)

- **REQ-SWR-LLM-001:** The selected LLM shall have its neural network weights and parameters fixed (frozen) upon deployment into the production environment.
- **REQ-SWR-LLM-002:** The LLM component shall not have the capability to self-modify or retrain its parameters based on user interactions or operational data post-deployment.
- **REQ-SWR-LLM-003:** Any update to the LLM's weights or fine-tuning data shall be treated as a major software change and shall follow the full verification, validation, and release management process.
- **REQ-SWR-LLM-004:** The LLM's behavior shall be deterministic, meaning that for an identical input prompt and system state, the output shall be consistently reproducible.
- **REQ-SWR-LLM-005:** The LLM selected for the system should be of a size and architecture that permits on-premise deployment and inference to ensure data privacy and security.

A.8 LLM Data Management (SWR-DAT)

- **REQ-SWR-DAT-001:** The training and fine-tuning datasets used for the LLM shall be treated as certified configuration items, subject to version control and quality assurance.
- **REQ-SWR-DAT-002:** All data used for fine-tuning shall be relevant, complete, accurate, and representative of the intended operational domain.
- **REQ-SWR-DAT-003:** The fine-tuning dataset shall be reviewed and approved by a human subject matter expert to ensure its quality and the absence of biases.
- **REQ-SWR-DAT-004:** The software shall ensure that no personally identifiable information or sensitive operational data is present in the training or fine-tuning datasets.
- **REQ-SWR-DAT-005:** A dedicated and representative test dataset, separate from the training and validation datasets, shall be used to perform the final performance evaluation of the LLM.

A.9 LLM Verification and Validation (SWR-VAV)

- **REQ-SWR-VAV-001:** The LLM's performance shall be validated against a predefined set of test cases covering both typical and known edge-case scenarios for the covered ATM domain.
- **REQ-SWR-VAV-002:** The verification process shall measure the LLM's accuracy in correctly identifying user intent and generating syntactically valid tool calls.
- **REQ-SWR-VAV-003:** The verification process shall measure the rate of "hallucinations" (factually incorrect or fabricated statements) in the LLM's natural language outputs, and this rate shall be below a predefined, acceptable threshold.
- **REQ-SWR-VAV-004:** Every LLM-generated statement or piece of code that is part of a safety case or assurance argument shall be treated as experimental and require line-by-line review and approval by a qualified human engineer.
- **REQ-SWR-VAV-005:** The validation of the LLM should include adversarial testing to assess its robustness against prompt injection and jailbreaking attempts.
- **REQ-SWR-VAV-006:** The validation process shall formally define the metrics and acceptance thresholds (e.g., Correct Intent Recognition Rate > 99.5%; Factual Hallucination Rate < 0.01%).
- **REQ-SWR-VAV-007:** Continuous performance monitoring of the model in production shall be implemented. The system shall periodically re-run a validation subset to detect any performance drift or unintentional regression.

A.10 LLM Explainability and Trustworthiness (SWR-EXP)

- **REQ-SWR-EXP-001:** The system shall be designed such that the human operator remains the ultimate decision-making authority.

- **REQ-SWR-EXP-002:** The system shall provide a clear explanation of how it reached a conclusion by making the generated tool call and its parameters visible to an authorized user.
- **REQ-SWR-EXP-003:** The LLM's outputs shall not contain expressions of opinion, judgment, or advice, particularly on safety-critical matters. Its function is limited to interpretation and summarization.
- **REQ-SWR-EXP-004:** The system should be designed to align with the principles of trustworthy AI as defined by regulatory bodies like EASA (e.g., transparency, accountability, fairness).
- **REQ-SWR-EXP-005:** A "Trustworthy AI" Compliance Dossier shall be maintained, explicitly mapping each system requirement to core principles defined by regulatory bodies.

Appendix B Use Case Mapping

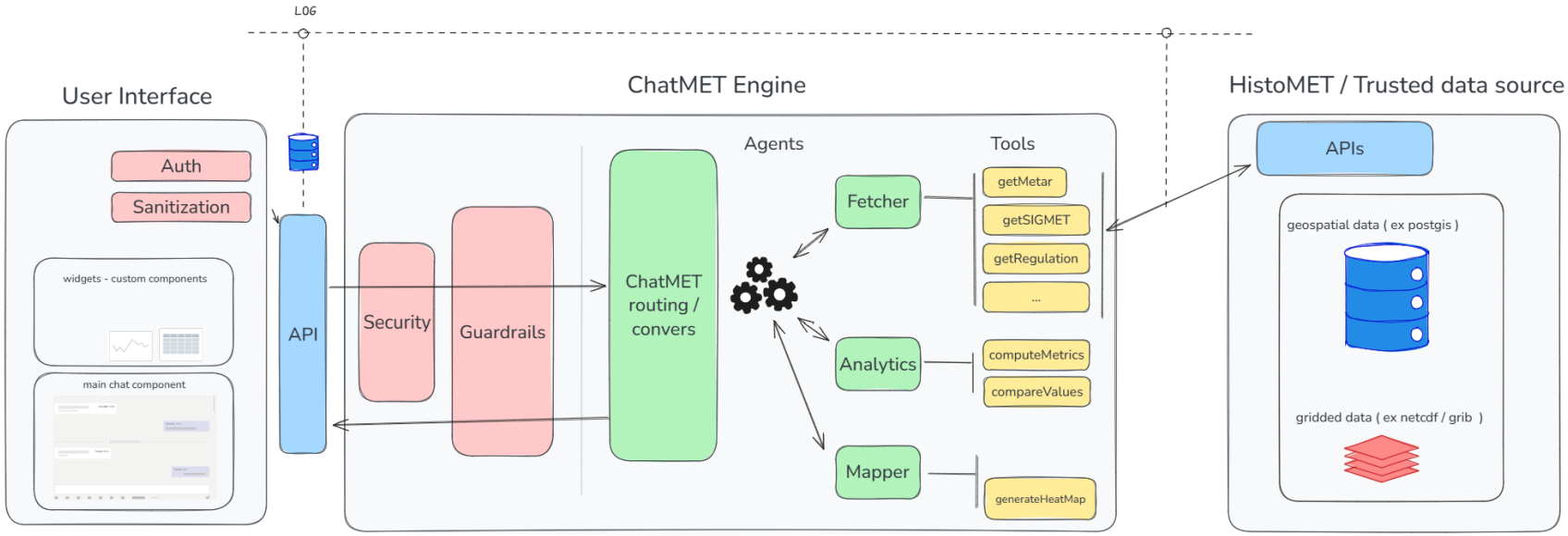
Main Use Case (UC)	Example User Query (Natural Language)	LLM's Detected Intent	Required Data	Generated Tool Calls (Unitary & Combined)	Resulting Output / Visualization
UC #1: Basic Data Access	"Get me all METARs for Nice airport on December 24th, 2024, between 8 AM and 12 PM UTC."	Fetch raw OPMET data for a specific location and time.	• HistoMet DB: metar table	getData(type="METAR", area="LFMN", startTime="2024-12-24T08:00Z", endTime="2024-12-24T12:00Z")	Table of raw and decoded METARs.
	"Show me the SIGMETs for severe turbulence issued over the Bordeaux FIR yesterday."	Find a specific weather event over a large geographical area.	• HistoMet DB: sigmet table	1. sigmets = getData(type="SIGMET", area="LFBB", startTime="2024-12-24T00:00Z", endTime="2024-12-24T23:59Z",filterType:"type",filterValue:"TURB") 2. plot_map(base_layer="LFBB_FIR", overlays=[sigmets])	Map displaying the SIGMET polygons over the Bordeaux FIR.
	"Show me the composite radar image for France on August 15th, 2024, at 18:00 UTC."	Retrieve historical geospatial weather data.	• HistoMet DB: radar_polygons table (Source: Météo-France)	1. radar_data = getData(type="RADAR_COMPOSITE", area="FR", startTime="2024-08-15T17:30Z", endTime="2024-08-15T18:30Z") 2. plot_map(base_layer="FR", overlays=[radar_data])	Radar map

Main Use Case (UC)	Example User Query (Natural Language)	LLM's Detected Intent	Required Data	Generated Tool Calls (Unitary & Combined)	Resulting Output / Visualization
	"Compare the TAF forecasts for Lyon on the morning of September 1st with the actual METARs observed."	Cross-reference two OPMET data types to verify a forecast.	• HistoMet DB: taf & metar tables	1. tafs = get_opmet_data(locations=["LFLY"], date="2024-09-01", period="morning", types=["TAF"]) 2. metars = get_opmet_data(locations=["LFLY"], date="2024-09-01", period="morning", types=["METAR"]) 3. generate_comparison_table(dataset1=tafs, dataset2=metars)	Side-by-side comparison table of TAF forecasts and the corresponding METAR observations.
UC #2: Regulation Justification	"Was the thunderstorm regulation on sector LFFFAM yesterday from 14h to 16h justified?"	Verify the presence of a hazardous weather phenomenon during a regulation period.	• HistoMet DB: lightning (Meteorage), radar_polygons • NM B2B: Regulation data	1. regulation = get_regulation_data(id="LFFFAM_...") 2. lightning = getData(type="LIGHTNING", area=regulation.area, startTime=regulation.start, endTime=regulation.end) 3. summary = generate_summary(context=[regulation], evidence=[lightning])	Text Report: "Yes, 125 lightning strikes were detected within the sector during the regulation period." with a supporting map.
	"A turbulence regulation was active on sector UM135, but I see no SIGMET. Check other sources."	Find evidence for non-SIGMET phenomena like Clear Air Turbulence.	• HistoMet DB: cat_products table (Clear Air Turbulence)	1. regulation = get_regulation_data(id="UM135_...") 2. cat_data = getData(type="CAT", area=regulation.area, startTime=regulation.start, endTime=regulation.end) 3. summary = generate_summary(context=[regulation], evidence=[cat])	Text + Map: "SIGMET was negative, but the CAT product showed a 75% probability of moderate turbulence at FL350."

Main Use Case (UC)	Example User Query (Natural Language)	LLM's Detected Intent	Required Data	Generated Tool Calls (Unitary & Combined)	Resulting Output / Visualization
UC #3: Weather Impact Analysis	"Which airport in France was most impacted by fog (visibility < 400m) in November 2024?"	Quantify and rank the impact of a weather phenomenon across multiple locations.	• HistoMet DB: metar table	1. fog_metars = getData(type="METAR", area="all_FR_airports", startTime="2024-11-01T00:00Z", endTime="2024-11-30T23:59Z",filterType:"vis":filtervalue:"lt400") 2. stats = calculate_statistics(data=fog_metars, metric="rank_by_event_count", group_by="location") 3. plot_chart(data=stats, type="bar")	Bar chart ranking airports by the number of hours with fog.
	"Compare thunderstorm activity and associated delays at Marseille airport for summer 2024 vs 2023."	Analyze the evolution of a weather-operational impact over two periods.	• HistoMet DB: lightning • NM B2B: Delay data	1. delays_24 = getData(type="DELAY_DATA", area="LFML", startTime="2024-06-01T00:00Z", endTime="2024-08-31T23:59Z") 2. ts_events_24 = getData(type="LIGHTNING", area="LFML_TMA", startTime="2024-06-01T00:00Z", endTime="2024-08-31T23:59Z") 3. # Repeat for 2023 ... 4. plot_chart(data=[...], type="comparative_bar")	Bar chart and a text summary of the key differences.
	"On a map of the Paris FIR, create a heatmap of sectors based on the number of thunderstorm-related regulations this summer."	Create an aggregated spatial visualization of operational impact.	• NM B2B: Regulation data • HistoMet DB: lightning, radar	1. regulations = getData(type="REGULATION_CONVECTION", area="LFFF_FIR", startTime="2025-06-01T00:00Z", endTime="2025-08-31T23:59Z") 2. stats = calculate_statistics(data=regulations, metric="count", group_by="sector") 3. plot_map(base_layer="LFFF_FIR_sectors", overlays=[{"type":"heatmap", "data":stats}])	Map + heatmap feature overlaid on the Paris FIR sector map.

Main Use Case (UC)	Example User Query (Natural Language)	LLM's Detected Intent	Required Data	Generated Tool Calls (Unitary & Combined)	Resulting Output / Visualization
	"What weather phenomenon (wind, TS, snow) caused the most cumulative delay minutes at Roissy CDG airport over the last 12 months?"	Identify the primary meteorological cause of delays over a long period.	• HistoMet DB: metar • NM B2B: Delay data	1. delays = getData(type="DELAY_DATA", area="LFPG", startTime="2024-09-28T00:00Z", endTime="2025-09-27T23:59Z") 2. weather = getData(type="METAR", area="LFPG", startTime="2024-09-28T00:00Z", endTime="2025-09-27T23:59Z") 3. correlation = correlate_data(dataset_A=delays, dataset_B=weather, time_window="1h") 4. stats = calculate_statistics(data=correlation, metric="sum_delay_minutes", group_by="phenomenon") 5. plot_chart(data=stats, type="pie")	Chart (Pie Chart feature) and a detailed report identifying the primary cause.

Appendix C Technical Architecture: ChatMET Engine



This document describes the architecture by detailing the data flow of a typical user query, from its initiation at the user interface to the final presentation of the processed data. The workflow is designed to be secure, deterministic, and fully traceable.

C.1 The Presentation Layer – User Interface

The workflow initiates at the Presentation Layer, where an authenticated user interacts with the system [REQ-SEC-AUT-001].

The user formulates a natural language query (e.g., « compare the wind speed at LFPG and LFPO yesterday ») and submits it through the main chat component's text input field [REQ-HMI-DSP-001]. Client-side logic performs initial input validation and sanitization as a first-line defense against common injection attacks [REQ-SEC-INP-001]. Upon submission, the query is securely transmitted to the ChatMET Engine, and the UI displays a processing indicator to the user [REQ-HMI-FDB-001].

C.2 Ingress and Validation: The Engine's Security Perimeter

The request enters the ChatMET Engine, which operates exclusively on a private, secure infrastructure [REQ-SYS-HLR-006]. It immediately passes through a sequence of security and validation modules:

API Gateway: The entry point that terminates the TLS-encrypted connection [REQ-SEC-API-001] and validates the user's session token [REQ-SEC-API-002].

Security Module: This layer enforces access control policies based on the user's credentials.

Guardrails: This final checkpoint before the core logic performs deep input analysis to mitigate prompt injection and filter any malicious or out-of-scope content.

C.3 Interpretation and Orchestration: The ChatMET routing agent

Upon passing the security perimeter, the sanitized query is processed by the ChatMET Routing Agent. This core component is a Small Language Model (SLM) whose function is not to answer the query but to interpret it and orchestrate the required actions. It can also be used as a direct conversational agent.

The SLM first parses the query to determine the user's operational intent (e.g., "comparison of metrics") from a predefined set [REQ-SWR-NLP-001]. It then extracts all necessary entities, such as {locations: ["LFPG", "LFPO"], parameter: "wind speed", time_range: "yesterday"} [REQ-SWR-NLP-002].

Based on the recognized intent, the agent selects the appropriate tool, in this case compareValues from the Analytics Agent's toolset [REQ-SWR-TCE-001]. It then constructs a structured, syntactically correct function call with the extracted entities as parameters [REQ-SWR-TCE-002]. This call is validated for correct data types and format before execution [REQ-SWR-TCE-003] [REQ-SAF-LLR-003].

C.4 Specialized agents and the Trusted data source

The validated function call is dispatched to the corresponding specialized agent. The Analytics Agent receives the call and executes the compareValues tool [REQ-SWR-TCE-004].

The tool's execution involves making one or more authenticated API calls to the HistoMET / Trusted Data Source. This ensures that all factual data is exclusively retrieved from a predefined, authoritative system [REQ-SYS-HLR-002] [REQ-SAF-HLR-001]. The HistoMET APIs query the relevant backend data stores, which may include a geospatial database (e.g., PostGIS) or gridded data files (e.g., NetCDF). Every tool call executed against the data source is recorded in an immutable audit log [REQ-SEC-LOG-001].

C.5 Post-processing and response assembly

The HistoMET data source returns a structured data payload (e.g., JSON) to the data post-processing & response generation layer of the ChatMET Engine.

The structured data is passed to the SLM, which now performs its secondary function to generating a concise, text-based summary of the results [REQ-SWR-DHR-001].

The final response payload is constructed by combining the SLM-generated summary with the original, unmodified raw data received from the tool [REQ-SWR-DHR-005] [REQ-SAF-LLR-001].

C.6 Final presentation and traceability

The assembled response is transmitted back to the presentation layer. The UI renders the information in a structured layout, using distinct custom widgets to display the AI-generated summary and the factual data table separately [REQ-HMI-DSP-002] [REQ-SAF-LLR-002]. This provides the user with both a convenient summary and the verifiable source data.

For full transparency, the interface allows the user to inspect the exact tool call that was generated, providing a complete and verifiable trace from the presented data back to its source [REQ-SYS-HLR-004] [REQ-HMI-TRC-001].

C.7 System Monitoring and Traceability

The primary purpose of this monitoring layer would be to log and visualize the agent's complete decision-making process for each user request, primarily for maintenance, debugging, and continuous improvement purposes. For any given query, the monitoring interface would provide a transparent, step-by-step trace of the agent's internal logic, including:

- The initial sanitized user input.
- The operational intent identified by the routing agent.
- The specific entities extracted (e.g., locations, time periods, meteorological parameters).
- The specialized agent and corresponding tool selected for execution.
- The final, structured API call generated and dispatched to the agent.