

Maintenance lifecycle of mission control applications: can AI help?

Diogo Vicente*, João Silva^b, Jose Feiteirinha^c, Luigi Palladino^d

^a *European Organisation for the Exploitation of Meteorological Satellites, diogo.vicente@eumetsat.int*

^b *European Organisation for the Exploitation of Meteorological Satellites, joao.silva@eumetsat.int*

^c *VisionSpace Technologies GmbH, jose.feiteirinha@external.eumetsat.int*

^d *European Organisation for the Exploitation of Meteorological Satellites, Luigi.Palladino@eumetsat.int*

* Corresponding Author

Abstract

EUMETSAT is an intergovernmental organisation based in Darmstadt, Germany, with 30 Member states that operates fleets of satellites for monitoring the weather, climate and the environment from space. With long duration satellite programmes like MSG or EPS, and respective follow-ons MTG and EPS-SG, EUMETSAT will continue to deliver data and products vital to weather forecasting and contributes significantly to the monitoring of the weather and climate change. To ensure efficiency and effectiveness, EUMETSAT endeavours to establish Multi-Mission Ground System approaches, where common solutions are deployed across multiple programs with minimal customisation. Despite the effort towards the adoption of harmonized application stacks, mission control functions are still provided by a mixture of heterogenous solutions and technologies and this increases the complexity of the maintenance activities and user support. The fact that EUMETSAT programmes are multi-decadal in scale (MTG and EPS-SG are both planned to be operational for 20+ years) creates knowledge management challenges for all participating teams including the ones focused in software maintenance. During the course of the programme, software products evolve substantially in response to user needs, new technology aspects are adopted, and systems accumulate a variety of workarounds and patches. Concurrently, maintenance teams undergo personnel turnover which lead to variability of institutional knowledge which constraints the team's ability to respond to different support requests. The knowledge retention problem is a real issue and that can be translated into a tangible risk that must be actively managed to ensure the continued reliability and effectiveness of the software system. To cope with this challenge, teams develop procedures, manuals and make use of different systems e.g. knowledge bases etc. to ensure These approaches can work well but are labour intensive and often lead to content stagnation, completeness issues, disorganization and overall fragmented knowledge. The retention of knowledge is also often hindered by the fact that information is scattered across a set of unintegrated systems and tools to support different processes and workflows e.g. source code management, issue tracking systems, continuous integration systems, project management systems, test management systems, etc. In this work we introduce Mofie, a RAG AI agent that can process different information sources e.g. requirements, anomaly reports, manuals in response to user queries. In addition, we introduce also three other AI agents (Fiona, Alex and Sara) to demonstrate how LLMs, few-shot prompting, tool calling and reasoning agents can be deployed in support to the investigation of anomalies and execution of flight control procedures.

Keywords: AI-agents, EUMETSAT, Spacecraft Monitoring and Control, Software Maintenance, Knowledge management

Nomenclature

This section is not numbered. A nomenclature section could be provided when there are mathematical symbols in your paper. Superscripts and subscripts must be listed separately. Nomenclature definitions should not appear again in the text.

Acronyms/Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ETL	Extract Transform Load
EPS	EUMETSAT Polar System
EPS-SG	EUMETSAT Polar System Second Generation
EUMETSAT	(European Organisation for the Exploitation of Meteorological Satellites)
LLM	Large Language Model
MOFIE	Mission Operations Facilities Intelligent Experiment
MSG	Meteosat Second Generation
MTG	Meteosat Third Generation
RAG	Retrieval-Augmented Generation
CAG	Cache-Augmented Generation
CLI	Command Line Interface
SECI	Socialisation Externalisation, Combination Internalization

1. Introduction

Since 2016, the industry has consistently outpaced academia in the development of "notable" machine learning models i.e. models that significantly advanced this field and are having real-world impacts. In 2023 alone, the number of new models released by industry was more than three times that of academia [1]. Additionally, the release of open foundational models [2] surged by more than threefold between 2022 and 2023 [3]. The number of AI-related projects on GitHub has also witnessed explosive growth since 2011, rising from 845 projects to an astounding 1.8 million by 2023. While AI publications have significantly increased from 88,000 in 2010 to 240,000 by 2022, the growth in 2023 was relatively modest, with an increase of just over 1%. Does this trend suggest that AI technology has now reached the maturity required for production-ready applications? Did the open-source software community manage to create a "mature-enough" ecosystem that allows transitioning from "cool demos" to "real use cases" that can drive and reshape the state of practice?

EUMETSAT is an intergovernmental organisation responsible for establishing, maintaining and exploiting satellite systems for monitoring the weather, the climate and the environment from space. It operates fleets of Geostationary satellites like MSG or MTG and Polar-orbiting satellites like EPS and EPS-SG. In addition, EUMETSAT is entrusted by the European Union to operate and exploit 4 Sentinel Missions part of the Copernicus Programme: Sentinel-3, Sentinel-4, Sentinel-5, Sentinel-6). EUMETSAT's Mission Control Centres are located in Darmstadt, Germany and provide the dedicated Flight Control teams with relevant Mission Control functions required to the monitor and control of the different satellites. EUMETSAT promotes the adoption of Multi-Mission Ground System approaches by deploying common solutions across multiple programmes that require minimal customization/extensions as a way to exploit synergies across programmes.

In this article we present the results of a series of experiments aimed at exploring how recent AI concepts, technologies, techniques, and frameworks could be applied in the context of the maintenance of spacecraft monitoring and control systems used at EUMETSAT. Similarly to other mission-critical domains, the software applications used within this domain, are mature, reliable and built on legacy technology stacks. Software architectures were designed long before the rise of AI-driven workflows and therefore create additional challenges that need to be overcome if one wants to benefit from the latest AI evolutions. This work demonstrates that the strategic incorporation of generative AI techniques into maintenance practices can modernize legacy systems, reduce entry barriers to users and help with sustainable knowledge retention which reduces long-term maintenance risks.

The long term maintenance of mission control applications presents a distinctive set of challenges that are deeply intertwined with the evolution of space exploration technology. These systems, which are often in operation for several decades, have grown in complexity as they accumulated modifications, patches, and workaround solutions. As mission control applications continue to underpin critical space operations, maintaining their reliability and safety has become increasingly difficult. Recent advances in artificial intelligence (AI) offer promising avenues to address these challenges by augmenting human expertise with automated analysis, predictive capabilities, and enhanced knowledge management.

1.1 Historical Context and Evolution of Mission Control Systems

Mission control systems have a rich history that reflects the evolution of space exploration. Early systems were designed with a focus on simplicity and reliability, but as missions grew more ambitious and technology advanced, these systems underwent significant modifications in response to mission and user needs. Like in other domains, in many cases, architectures and technologies did not evolve as could have and over time, the layered additions of new features across legacy and new components, the ad hoc fixes have, in many cases, resulted in scenarios where:

- **High Legacy Code Complexity:** Original design decisions have been obscured by decades of incremental updates, often leading to systems that are difficult to fully comprehend.
- **Institutional Memory Loss:** As personnel change and experienced team members retire, critical knowledge about system design and operational nuances is at risk of being lost.
- **Patchwork Solutions:** Temporary fixes and workarounds, while effective in the short term, may evolve into entrenched practices that obscure the ideal architectural vision.

Understanding this historical evolution is crucial because it frames some of the maintenance challenges and provokes recurring discussions on how to use technology to best address the scenario avoiding risks and exponential costs e.g. shall we upgrade or re-design? What is the impact on other teams? Can AI help managing this or at least improve the scenario? Would it be possible to delegate work on legacy chains to agents?

1.2 Maintenance challenges in Long-Lived Systems

Long-lived mission control applications face maintenance challenges that are markedly different from those encountered in more modern or transient systems, to name a few:

- **Knowledge Retention and Transfer:** With systems in service for decades, the departure of experienced engineers often leads to critical gaps in understanding the rationale behind legacy decisions, undocumented workarounds, and system idiosyncrasies.
- **Documentation Management:** As systems evolve, documentation frequently becomes outdated or fragmented. The challenge is not only in creating comprehensive documentation but also in ensuring that it remains relevant, accessible, and correctly synchronized with the system's current state.
- **Complex System Interactions:** Over time, the interdependencies between different components can become highly intricate. These interactions are often under-documented, making it difficult to predict how changes in one area may ripple through the entire system.
- **Bespoke approaches for application lifecycle management:** The fact that these systems/missions are complex, decisions were often made with a mission/project centred view and this lead to bespoke ecosystems with dependencies on specific companies, hardware, compilers, etc.

In order to avoid having dedicated teams of experts for a given missions, engineering teams tend to upgrade towards more modern technology stacks and methodologies and with this harmonize the application lifecycle management across different product families. In this context, paradigms like agentic AI could potentially offer the possibility of defining legacy workflows as AI workflows that could be delegated to system specific agents. On the knowledge management front, advanced RAG techniques [10] could enable knowledge management systems that could go beyond managing explicit knowledge but open a route to address tacit knowledge. Where are the main challenges? As a starting point, legacy ecosystems do not tend to "AI friendly" e.g. no/old APIs.

1.3 The promise of Generative AI

Artificial intelligence and machine learning has long been used in the scope of maintenance activities e.g. automotive where sensors historical data is monitored against models to identify and prevent failures on parts/components. proposed in the scope of maintenance [4]. More recently with the surgency of generative AI, the promise/potential of generative AI within software development and maintenance is becoming more present. In [5], the author identifies 4 topologies for using AI that lead to a reversal of roles and expose the potential of this technologies i.e. from the traditional approach with human in control and using automation, to AI in the loop where AI is used to automate some of the traditional use cases, to AI assuming selected roles specified by the user, to Human-in-the-loop scenario where AI manage operations across different roles and the human takes more a supervision stance.

Artificial intelligence, and in particular Large Language Models (or LLMs) have matured to the point where its application within maintenance activities is not only feasible but also highly beneficial. AI-driven systems can address many of the persistent challenges associated with the maintenance of long-lived mission control applications:

- **Enhanced Documentation and Knowledge Management:** AI tools can continuously synthesize and update documentation by integrating disparate sources of information, ensuring that both legacy and current system states are accurately reflected.
- **Automated Anomaly Reproduction and Analysis:** Modern agent technologies, and potentially computer use, can provide anomaly investigation and reproduction.
- **Natural Language Interfaces:** By enabling operators to interact with complex system data through natural language queries, AI can lower the cognitive load and improve operational efficiency, even for personnel who are less familiar with the underlying technical intricacies.

While the potential benefits are significant, it is essential to acknowledge the risks—such as oversimplification of complex systems and potential gaps in contextual understanding—which necessitate a hybrid approach that blends AI capabilities with expert human oversight.

1.4 Objectives and Scope of the Analysis

This work outlined in this article, aimed at evaluating how recent developments in generative AI can be used in support to the maintenance of spacecraft monitoring and control applications. The primary focus was not the development of production ready AI applications, but to: 1) learn how use these technologies and techniques 2) apply the techniques to real use cases 3) understand maturity of the existing frameworks. To achieve this, we identified specific challenges that we considered as possible to resolve with the current state of AI.

As depicted figure 1, and inspired on [5], we have defined 4 AI-driven roles able to perform common complex maintenance tasks:

Role (AI-Agent-As-...)	Acronym	Name
Mission-Control-Domain-Expert	Mofie	Mission Operations Facilities Intelligent Experiment
Mission Control Operator	Fiona	Flight Operations Navigation Assistant
Automation Logic Expert	Alex	Automation Logic Execution eXpert
Anomaly reproduction assistant	Sara	Software Anomaly Reproduction Assistant

Table 1: The AI agents

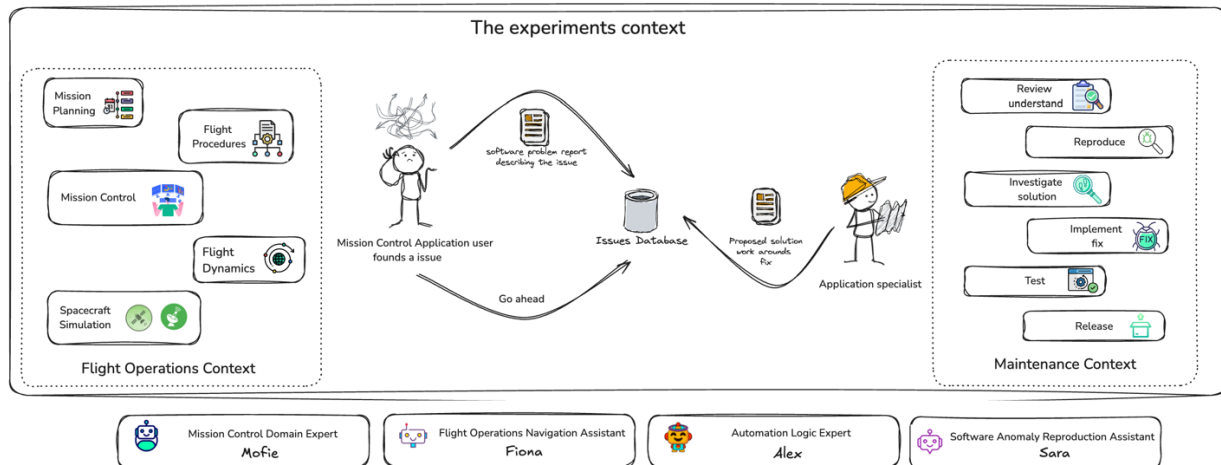


Figure 1: Scope of the work

1.4.1 Mofie, the AI driven mission control applications expert

The corrective maintenance of spacecraft monitoring and control applications tends to revolve around responding to problem reports raised by flight operations teams in areas related to mission planning, flight dynamics, mission control, satellite simulation, among others. These systems are highly specialised, and due to programmatic and historical reasons, are often not standardized i.e. each mission tend to have its own ecosystem of mission control tools with some level of overlap. Maintenance teams at EUMETSAT, working in a multi-mission approach must be expedite in the management of large amounts of information e.g. technical baselines, requirements, design, implementation, mission specificities. Furthermore, to ensure long term maintainability, teams need to be specialised on the technological stacks for the different systems. This need, led to the first and simplest use case, which aimed at creating an AI agent able to infer answers based on information spread across diverse sources e.g. requirements, user manuals, data bases of problems, among others. We named the agent MOFIE (**M**ission **O**perations **F**acilities **I**ntelligent **E**xperiment) which focuses on exploring RAG techniques [10] applied to this domain.

1.4.2 Fiona, the flight operations navigation assistant

The domain ecosystem around monitoring and control of spacecraft systems is very rich, i.e. it spans across several areas of knowledge from spacecraft operations, command and control, automation, simulation, flight dynamics, mission planning etc.. Each of these areas is often linked with specific tool suites that are tailored, specialised and integrated within the scope of a mission. At EUMETSAT, several of these systems were designed before “AI era” and, as such, do not make available mechanisms that can be easily exploited through modern AI frameworks e.g. lack of simple APIs. In an effort to harmonise and streamline mission control solutions, many of the roadmaps include alignments across missions, and this led to the creation of C-MCS and C-SATSIM which are mission control and satellite simulation suites. These suites are based on ESA/MICONYS and ESA/SIMULUS [6]. In this experience, we created an AI agent which we have named Fiona (**F**light **O**perations **N**avigation **A**ssistant). Fiona is able to interact with both Mission Control Systems based on C-MCS and with Satellite Simulators based on C-SATSIM. With Fiona, we demonstrate how this was achieved using few-shot prompting techniques [7].

1.4.3 Alex, the automation logic expert

An integral part of software maintenance is the execution of test cases to verify and validate the applications. Traditionally, this is first done in the scope of a specific application where software is evaluated at unit, integration and system testing level and then at “higher” levels of integration with other systems until the full ecosystem is build. In the case of mission control applications, the execution of a set of flight control procedures is often used as a quality gate to demonstrate if the ecosystem of monitoring and control applications deployed for a specific mission is ready to use. Flight control procedures describe the logical flows to perform specific spacecraft activities and executed

manually. In this experiment, we have created Alex (**A**utomation **L**ogic **E**xpert) who is able to follow procedures by interacting with C-SATSIM based simulators and C-MCS based mission control systems. Alex focuses on exploring the tool calling capability [8] made available in some large language models.

1.4.4 Sara, the software anomaly reproduction assistant

One critical challenge in mission control application maintenance is the time-intensive nature of anomaly analysis and reproduction. Maintenance engineers spend significant effort manually reviewing anomaly reports, often struggling to identify duplicate issues or connect them to previously resolved cases. This process is further complicated by incomplete or poorly structured anomaly reports that lack critical information needed for effective resolution. This need, led to the creation of an AI Agent able to outline a procedure to reproduce a given anomaly from the problem raised by and end user. With Sara (**S**oftware **A**nomaly **R**eproduction **A**ssistant), we explore agent-ReAct techniques [9] where the agent is able to demonstrate reasoning and corresponding actions.

2. Theory and calculation

This section presents the theoretical and conceptual basis driving the work presented in this article. It is assumed a level understanding on concepts such as: RAG [10], Chunking, Embeddings [11], Langraph, Langchain [12], among others.

2.1 Mofie, the AI driven mission control applications expert

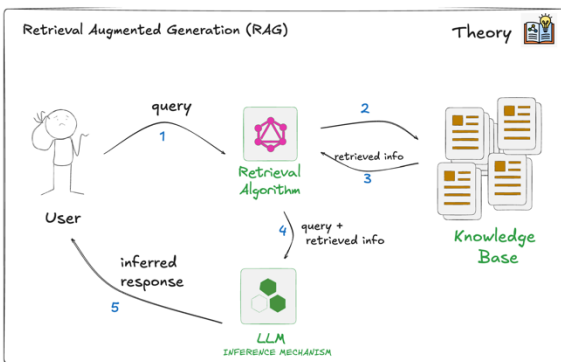


Figure 2: RAG concept

Large Language Models (LLMs) derive knowledge from information available on the training data and the user query. Training models is hard and expensive, so the RAG techniques come as a solution to augment the inference capability by adding external data to the process e.g. domain specific knowledge. Figure 2 depicts the process and the 3 main components of a RAG system: the retrieval algorithm, the inference mechanism and the knowledge base Mofie implements a RAG system using mission control applications data sources. In this experiment, we present how we have tailored/configured/implemented these components in our context.

Figure 3 depicts the Mofie experiment concept where we created a simple mechanism enabling users to raise questions related to mission control applications.

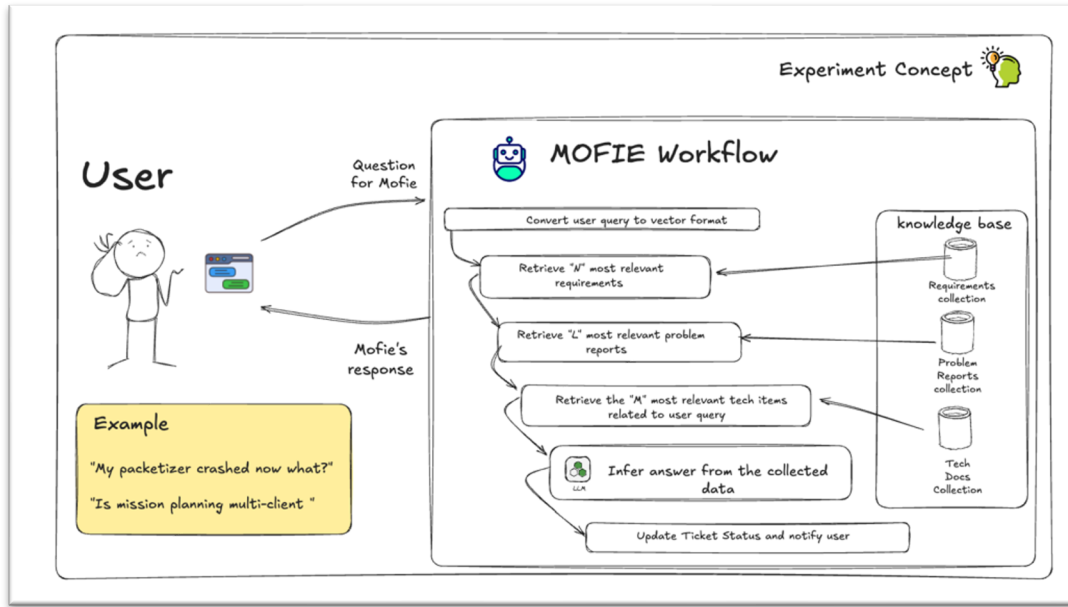


Figure 3: Mofie Workflow

These questions would then be answered by an LLM that would infer across diverse sources of data namely: requirements, user manuals, data bases of problems, among others. Due to infrastructure constraints e.g. hardware, we soon realised that the form factor had to be asynchronously i.e. a chatbot would be too slow in the provision of answers and this would not provide a suitable experience. To address this, the Mofie's user interface relied on the out-of-the-box GitLab mechanism. Users could create a new ticket with a specific question and this would trigger a pipeline that would answer the user query. As depicted in the [Figure 3](#), to provide the answers, Mofie would use a typical RAG strategy, where answers to user queries are derived as follows:

1. User input query is converted into a query vector;
2. Knowledge is extracted from the different collections based on vector similarity;
3. Answer is inferred based on the knowledge extracted, user query and LLM training context.

As expressed above, Mofie relies on a vector database where the documents persisted as collections which aggregate items of the same type. [Figure 4](#) depicts the creation of the knowledge base which follows a simple ETL pipeline that aims at ensuring that the information that is used as the basis of Mofie's answers is as relevant as possible. This pipeline included steps to read and clean the input data (e.g. removing table of contents, indexes, tables, images), followed by a summarisation step, where an LLM compressed the info being persisted as knowledge. After this step, the data items still had to be chunked to into blocks appropriate to the vector database.

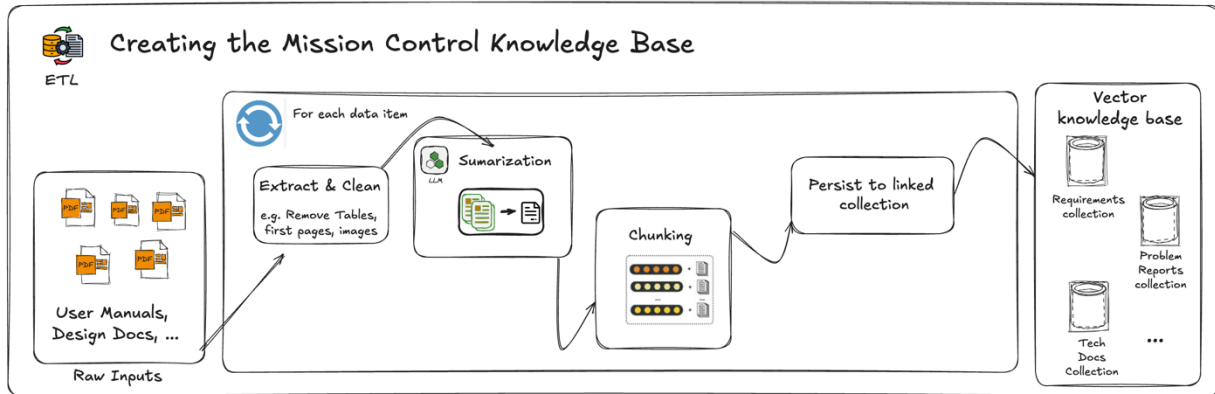


Figure 4: Mofie Knowledge creation: Extract, transform and load (ETL) pipeline

2.2 Fiona, the flight operations navigation assistant

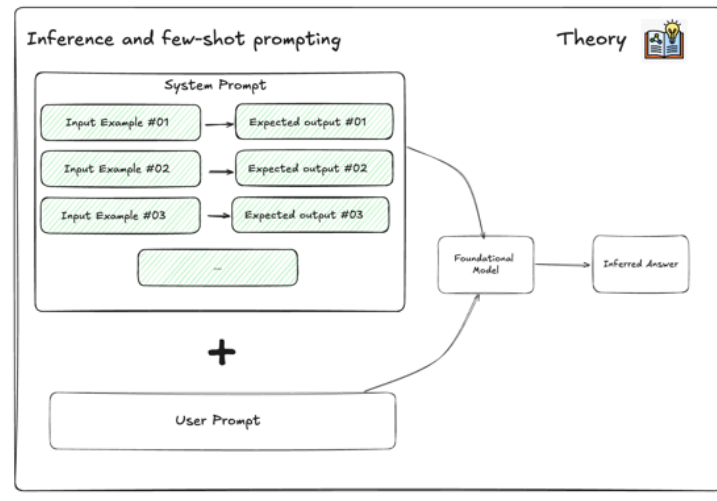


Figure 5: Inference and few shot prompting

Fiona explores the feasibility to command external systems solely by using few-shot prompting techniques. As depicted in [Figure 5](#), LLMs infer responses by combining the training knowledge embedded in the LLM weights, the user query and the system prompt information. The latter is used to nudge the LLM to respond in a specific way e.g. personality.

With a few-shot prompting approach, the system prompt is defined with a set of possible inputs examples and expected responses to nudge the LLM to infer in a specific way based on the user queries/prompts. With Fiona, we wanted to use this technique to call external tools.

The intention was exploring the feasibility of having an AI agent able to use “non-AI ready tools”. Specifically, we intended to create the possibility to have an agent able to interact with the Mission Control Systems and Satellite Simulators. From a form factor standpoint, this experiment made available a chat-like user interface, and the workflow is as depicted in [Figure 6](#) i.e.:

1. On reception of a user query, Fiona used an LLM to infer the Satellite Simulator or Mission Control System tool to be used to respond to the input request for example: “start_simulator”;
2. Programmatically, the relevant tool would be called using the relevant API endpoint or bespoke system specific interface;

- Based on the tool response, Fiona relied on a second LLM to translate the result to human readable format (again using few-shot prompting).

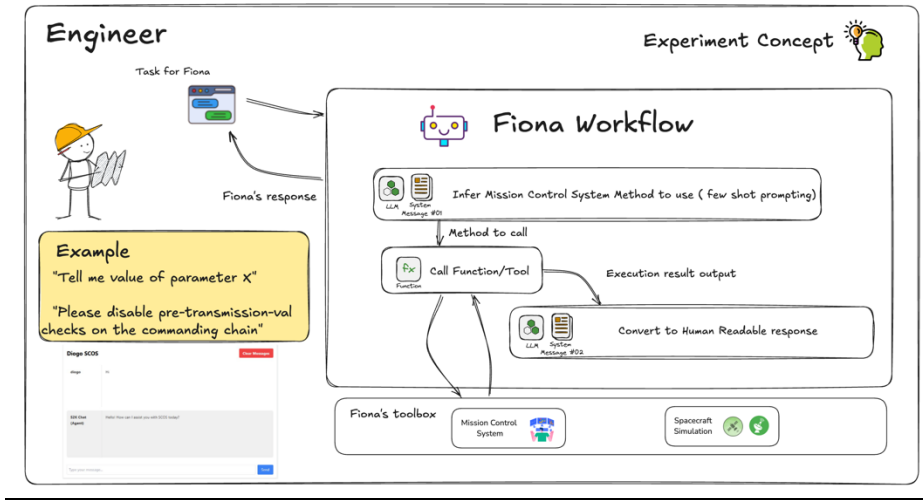


Figure 6: Fiona workflow

2.3 Alex, the automation logic expert

```
python
def multiply(a: float, b: float) -> float:
    """
    Multiplies two numbers together.

    Parameters:
    a (float): The first number.
    b (float): The second number.

    Returns:
    float: The product of a and b.
    """
    return a * b
```

Figure 7: Tool calling

Having demonstrated the ability of calling tools by nudging the LLM using system messages. In Alex we delegated this skill to the LLM. During the course of the work, LLMs starting to appear with tool support, meaning that during the ability for tool calling was part of the training phase and so, it could be removed from the context passed as part of the system context message. This was a great step towards simplifying the usage of LLMs in applications as resolves a lot of the challenges and optimisation needs to ensure the right tool is selected and creates room for more context specific system prompts. While, with Fiona we had to invest time in “prompt engineering cycles” to ensure the tools were called deterministically, in Alex this was somewhat transparent. As depicted in [Figure 7](#), the awareness of external tool and their semantic purpose was passed using “docstrings” i.e. the tool documentation. By connecting the LLM with the tools, the LLM is able to actually call these. In Fiona, this step was done programmatically i.e. the LLM would infer the name and parameters of the tool to be called and a bespoke function would perform the tool invocation.

In this experiment, the intention was to implement a prototype able to follow a test procedure in a “relatively” deterministic fashion i.e. reduced hallucinations. If feasible, this setup could be used for asking Alex to validate flight control procedures, configure the satellite simulator in accordance with a set of procedures, perform integration testing across systems etc. As depicted in [Figure 8](#), Alex makes use of an LLM mainly for: 1) tool calling purposes 2) interpreting procedure intermediate and results. Alex’s approach can be considered an evolution from Fiona’s in the

sense that more delegation is given to the foundational model. From a form factor standpoint, interactions were carried out via a chat-like user interface.

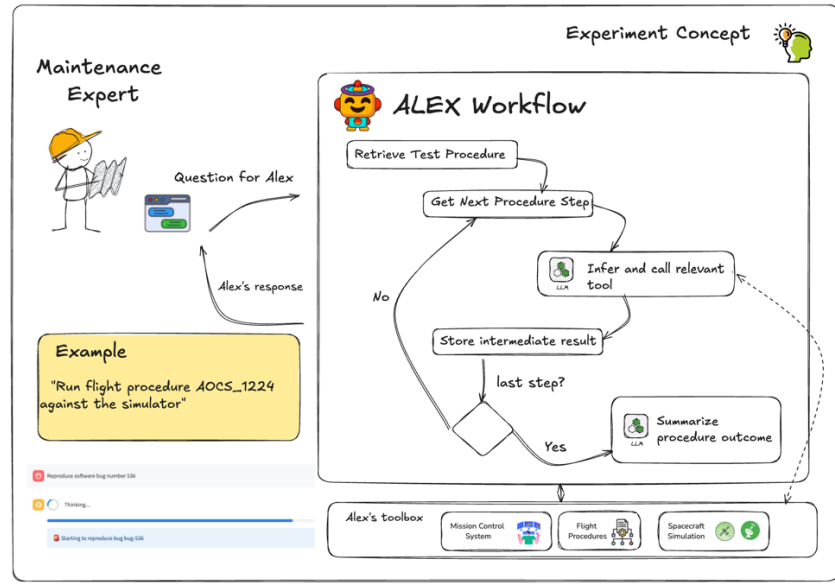


Figure 8: Alex's Workflow concept

2.4 Sara, the software anomaly reproduction assistant

With Sara, we tried to reproduce an anomaly based on user reports. This meant that, Sara had to be able to interpret a software report, derive the procedure to reproduce, reproduce the procedure by calling different tools and finally provide a result i.e. problem is reproducible or problem cannot be reproduced. This problem as many layers of complexity and to address this, Sara is implemented using an AgentReact [9] framework. With the example of Fiona and Alex, decision-making e.g. the tool to call was separated from the actual actions i.e. the tool invocation. ReAct agents represent a different approach where agents loop continuously through reasoning-action steps to resolve a specific complex task.

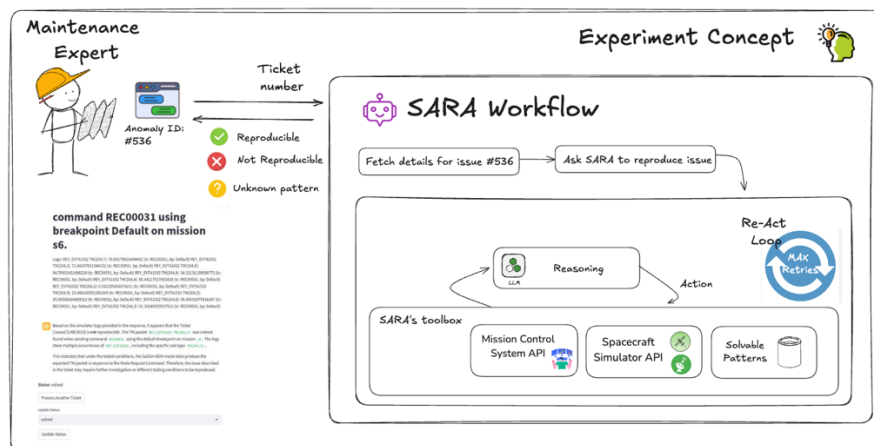


Figure 9: SARA's workflow concept

Figure 9 depicts Sara's workflow which in essence implements a loop "Reasoning" and "Act" where the latter is assisted by the satellite simulator and mission control tools. In this experiment, the novelty is was adding a specific

tool that lists the problems/patterns that can be solved. This tool helped nudging the agent to converge or to identify requests out-of scope.

3. Material and Methods

This section expands the work with technical details allowing the readers to reproduce the work or extend as needed.

3.1 Mofie, the details

Figure 10 details Mofie’s RAG technical context. As introduced in section 2, the prototype includes 3 main components: the knowledge base, the retrieving algorithm, and the inference mechanism.

- **Knowledge Base:** The knowledge base is implemented using ChromaDB [13], an open-source, vector database designed to handle the storage, retrieval, and management of embeddings and vectors used in machine learning applications. As depicted, this database is populated using a simple ETL pipeline that organises the information into different collections based on their type i.e. requirements, technical baselines, anomaly reports, maintenance documentation. As introduced, the ETL pipeline includes typical steps to clean the data: like removing headers, tables; transforming the data: summarizing the information using an LLM and loading to the vector database. Prior to the persistence into the different collections, the summaries are split into chunks of a specific size and overlap. Larger chunks provide the inference mechanism with more context but might make the system slower. Overlap across chunks usually affects the contextually correctness of the answer. In Mofie, these parameters were tuned empirically: chunk size: 2048, and overlap: 64.
- **Retrieving algorithm:** The retrieving mechanism simply pulled the 10 most relevant results for each collection. Again, this parameter was tuned empirically.
- **Inference Mechanism:** The inference mechanism relied on Ollama [13] as a pre-requisite was running the RAG locally. The large language model used was “command-r 35b-08-2024-q5_K_M” [14] which is a model optimised for conversational and long context tasks. Again, this was selected empirically, after using a set of different models.

Mofie’s workflow relied on GitLab [15] bespoke mechanisms to implement the user interface and overall orchestration. GitLab makes available the possibility to create issues (usually to report problems or bugs in software). This mechanism was used to trigger the inference pipeline to answer a dedicated user query.

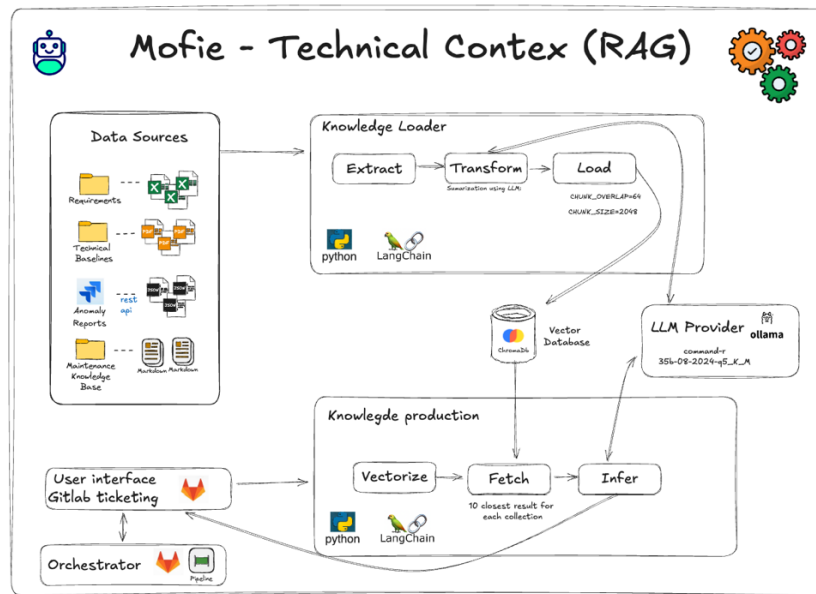


Figure 10: Mofie Technical context

3.2 Fiona, the details

As depicted in [Figure 11](#) Fiona was implemented as a Django application as this ensured quick prototyping. This specific experiment also covered aspects related to the handling of multiple AI and non-AI agents which is covered [16]. This section focuses on the 3 components in [green](#) as these are key to reveal the tool calling mechanism using few-shot prompting. From a system perspective, the prototype was implemented following the typical Model-View-Template architecture inherent to the Django framework [17]. In this framework, the **Template** system handles the layout of the facing application, the **Views** cover the business logic which relies on the **Models** that are responsible for the data and persistence layer. As detailed in [16], with Fiona, we also wanted to explore how to handle multiple users in a multi-agent setup, where a specific user could define its own playground where it was possible to choose the instantiated agents i.e. user 1 might want to have interactions between: Alex and Mofie while other user might only need to use Sara. This was also a driver to use Django as this framework provides user management out-of-the-box and this allowed us to focus on the actual AI experiment.

- **Tool Revelator:** The main purpose of this component was identifying the tool to use in response to a given user request. For this, we used LLM qwen2.5:32b [21] which we wanted to literally to answer with the method to call. To achieve this, we tailored the system message of the LLM with examples as depicted within the blue box in [Figure 12](#).
- **Tool Executor:** To interact with the Mission Control System there was an initial hinder we had to resolve. System based on C-MCS/Miconys only make available CORBA [18] interfaces, and these are not very useful i.e. semantically relevant for the LLM context i.e. were too low level and system oriented. So, we first generated python bindings from the IDL [19] definitions such that we could manipulate function calls and data structures made available by the mission control system. Then, we developed dedicated methods that we could bind to the LLM via the system message. The interaction with the Satellite Simulator was easier because within C-SATSIM there were already a Rest-API that we could use.
- **Tool output sanitizer:** The main purpose of this component was translating/interpreting the outputs resulting from the tool invocation. While for the SATSIM we had a rest API that provided JSON [20] responses that the LLM could handle, on the Mission Control System this was not the case. So, we had to configure the system message as depicted within the green box in [Figure 12](#) to ensure the user was provided with intelligible feedback.

In this specific example, it is worth noting that for better results one could have different LLMs and separate system messages to drive the Fiona's behaviour, but, as the objective was exercising the few-shot prompting the setup we had in place was deemed as sufficient for this phase.

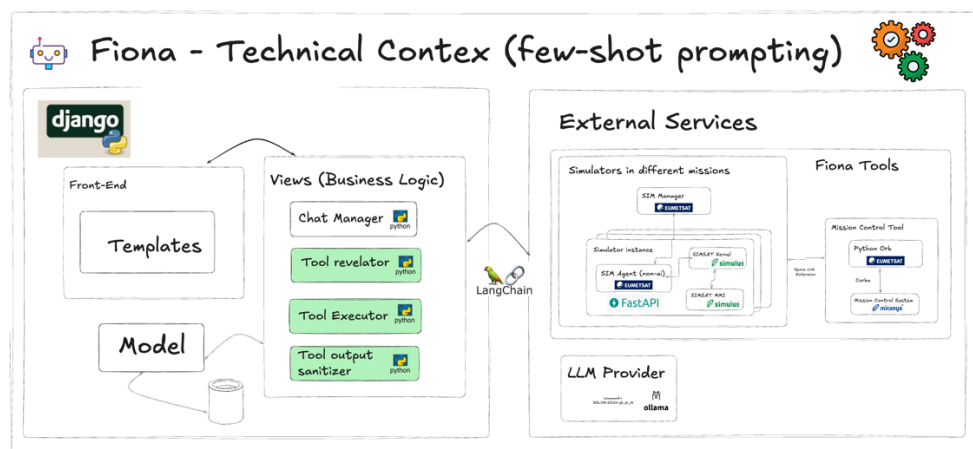


Figure 11: Fiona Technical Context

SARA is implemented as an agentic assistant designed to automate and streamline the process of reproducing software anomalies by analysing anomaly reports (AR) from JIRA tickets. Time saved in Ai solving simple tasks is re-invested by the maintenance team to tackle bigger challenges.

At a high level, SARA combines a ReAct agent with tool calling capabilities to analyse ticket descriptions, compare them against a database of known issues, and determine whether the issue falls within the scope of problems that can be addressed. The workflow begins when a ticket ID is provided; SARA uses JIRA to retrieve the corresponding ticket description. This description is then passed to the ReAct agent, which is equipped with a suite of tools designed to interact with various systems and databases.

The tools available to Sara are diverse, ranging from simple command-and-control interfaces for interacting with a set of instances of simulators and mission control systems (e.g., sending commands or reading telemetry) to more complex utilities such as LLM-based parsing and parameter extraction. These tools enable the agent to process natural language inputs, extract relevant information, and execute specific tasks. For instance, some tools are designed to filter log streams using regex patterns to verify if some scenarios are explicitly reproducible or not. Tools are implemented as callable python functions with LangGraph `@tool` decorator tag. Tools function mostly call either other specialized LLMs or perform calls to specific other tools or databases accessible with REST-API like EGS-CC MCS or the simulator.

Sara's implementation is based on a slightly modified implementation of the ReAct agent from LangGraph, which maintains a record of the agent's thought processes as it interacts with its environment. This allows for tracking continuous feedback between the agent and the tools it calls, balancing the autonomous experimentation and problem-solving with increased transparency and observability. While the system are hosted in the offline network of EUMETSAT, the LLMs are hosted locally at EUMETSAT in a Kubernetes cluster equipped with GPUs, running on Ollama, an instance of the Qwen2.5-Coder [24] accessed through OpenAI's API syntax.

The architecture of SARA is designed as a microservices-based application, with each component encapsulated as a Docker container orchestrated via Docker Compose. This modular approach ensures ease of integration and scalability. Each component is versioned and deployed through an automatic pipeline following a GitOps [23] approach to ensure full reproducibility and maintainability. The user

interface (UI) is built using Streamlit, while the backend services are implemented in Python as FastAPI applications. These services provide RESTful APIs for seamless interaction with external tools such as the simulator or the mission control system. In its final iteration, Sara posts the results of its analysis and actionable suggestions back to the Streamlit UI and JIRA once it assessed if the ticket is solved or whether it is reproducible and it requires more action to be acted by the maintenance team.

SARA's design is focused on flexibility and scalability, making it a powerful tool for automating anomaly reproduction and analysis while maintaining compatibility with existing and future infrastructure and processes.

4. Results

4.1 *Mofie, the results*

Figure 14, provides 4 examples of actual results we have achieved with Mofie. In Example 01, asks is a specific subsystem (mission planning) is able to handle more than one user in parallel. Mofie was able to understand the question, retrieve relevant requirements and answer the question sufficiently well. In the specific question, the user wanted to understand if the software allowed for planning and scheduling activities to be performed in parallel e.g. different spacecrafts. Mofie, was able to correctly state that "multi-user" is not in place by application design, but the some processes/activities can be performed in parallel.

In Example 02, the user does not really ask a question, it simply presents an observation. Mofie was able to retrieve relevant information (requirements and previous issues observed with this system). Furthermore, Mofie stated that, it was information to help the user only.

In Example 03, is more complex and challenging for Mofie as it requires cross-relating information across multiple systems. Mofie, hallucinated, stating it's possible to access the information the user needs (correct), provided a procedure how to do it (and this is also ok, from a behaviour standpoint). Unfortunately, the procedure is wrong, it provides steps using features of the Mission Control System instead of the Automation component. Still, the user found the answer helpful and manage to get the info needed based on Mofie's contribution, which, is still a helpful outcome.

In Example 04, Mofie simply failed. Mofie, just extracted because the question involves more than one system which makes it, the across systems. Mofie, grasped the question context but invented the results. It should be noted that the information requested is not available for consultation. Mofie would have had to compute these on its own. So, one

can easily argue that the challenge was too high, and that these answers could be resolved by empowering Mofie with tool calling mechanism.

	Context in the question grasped?	Data Retrieved is relevant	Answer, relevant?	Answer helpful?	Qualitative Score
Example 01	Yes	Yes	Yes	Yes	★★★★
Example 02	Yes	Yes	Yes	Yes	★★★★
Example 03	Ok	No	No	Yes	★★
Example 04	Yes	No	No	No	★

Table 2: Mofie results, qualitative evaluation

The screenshot displays four chat interactions with Mofie. Each interaction includes a user question, a Mofie response, and a red stamp indicating the quality of the response. Example 01 shows a detailed and correct response to a question about multi-client support. Example 02 shows a correct but less detailed response to a question about Packetizer crashes. Example 03 shows an incorrect response to a question about VC usage, marked with a 'WRONG' stamp. Example 04 shows an incorrect response to a question about anomaly percentages, also marked with a 'WRONG' stamp.

Figure 14: Mofie - the results

4.2 Fiona, the results

Figure 15 provides some of the results obtained using Fiona. In Example 01 Fiona was able to call the correct tool on request. On Example 02, Fiona hallucinated, the user asked to retrieve info on a specific parameter and Fiona was unable to do it and inferred the results incorrectly. Note the hallucination happened due to the longer context. Part of the reason is how we build and store the history (can be improved to avoid this) and the intelligence of the model. Right now we're using a 32b parameter model (qwen 2.5), for performance, but this can greatly be improved. Finally, in Example 03, and demonstrating the previous justification, Fiona is able to retrieve the parameter successfully using the C-MCS/MICONYS interface implemented.

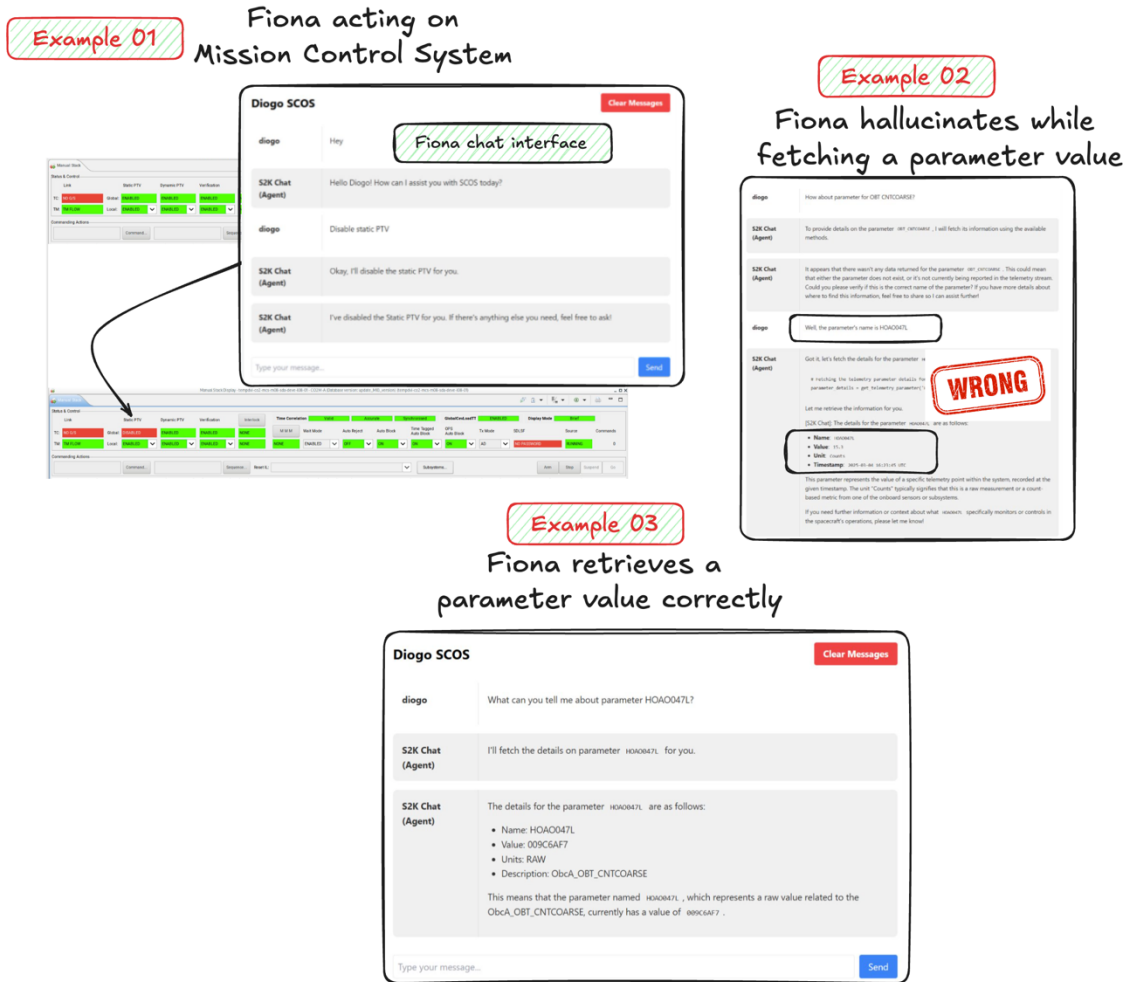


Figure 15: Fiona, the results

4.3 Alex, the results

Figure 16 provides an actual execution example of Alex working towards reproducing a specific anomaly. In the provided example, the different steps driving the execution are presented to the user. For instance, one can see, the execution flow on the left with the approach followed to execute the procedure with the triage agent at the centre. The diagram also presents the actions the agent takes to prior to execution (step-by-step Flow). It is possible to understand that the example below was massively simplified for demonstration purposes. For example, the procedure to be reproduced was defined in a json file, to ensure Alex did not diverge and called the different tools in sequence as per expectation. A second simplification in the example below, is that the tools called were stubs of the actual interfaces. This later fact would not affect the outcome.

This experiment, as well as the experience with Sara, clearly demonstrate the need that we still have today for observability. Agents and LLMs do not always responds as one expects and as such, it is fundamental to create useful mechanisms to understand the actual behaviour. This need increases as the more reliance on frameworks one has. In Fiona, there was also the need to expose what the agentReact was doing, as this was obfuscated by the framework and creating difficulties in the overall setup.

Reproduce software bug number 536

Thinking...

Starting to reproduce bug bug-536

Retrieving procedure steps for bug bug-536

Retrieved procedure: BUG_bug-536_REPRODUCTION

AI Workflow

System Interaction Flow

```

            graph TD
            subgraph Execution_Flow [Execution Flow]
            MC[Mission Control]
            ES[Execution System]
            SIM[Simulator]
            MC -- Command --> ES
            MC -- Forward --> SIM
            ES -- Response --> MC
            SIM -- Telemetry --> MC
            end
            subgraph Triage_Agent [Triage Agent]
            Triage[Triage Agent]
            LLM[LLM Reasoning]
            UI[Streamlit UI]
            end
            Triage -- "4b. Result" --> ES
            Triage -- "4a. Execute Step" --> SIM
            UI -- "1. User Request" --> Triage
            LLM -- "2. Analyze Request" --> Triage
            Triage -- "3. Select Tool" --> LLM
            Triage -- "5. Generate Response" --> UI
            
```

Procedure Flow Diagram

Step-by-Step Flow

```

            sequenceDiagram
            participant AI_Agent
            participant MCS
            participant Simulator
            AI_Agent->>MCS: 1. TC12345
            MCS->>Simulator: TC12345
            Note over AI_Agent: Purpose: Ping report request
            Simulator-->>MCS: TM11
            Note over AI_Agent: Purpose: Command Accepted
            AI_Agent->>MCS: 2. TM11
            MCS->>Simulator: TM17
            Note over AI_Agent: Purpose: Command Executed
            Simulator-->>MCS: TM172
            Note over AI_Agent: Purpose: Ping report
            
```

Consulting triage agent for step 1/4

Triage Agent Response:
 I have executed the Inject Telecommand TC12345 for Ping report request. The result is a Success, and the purpose of the command was to dump CELT. The timestamp shows that this occurred on January 3rd, 2025 at 10:23.

Executing Inject Telecommand - TC12345: Ping report request

Telemcommand TC12345 sent successfully

Consulting triage agent for step 2/4

Triage Agent Response:
 You have asked to execute the command Check Telemetry TM11 for Command Accepted. I will format an answer based on the tool's response.
 The command has been successfully executed. The telemetry parameters are now reporting as expected and showing that the command was accepted by the system.

Executing Check Telemetry - TM11: Command Accepted

Telemetry parameter TM11 checked

Consulting triage agent for step 3/4

Triage Agent Response:
 The TM17 telemetry check has been executed successfully. The command purpose was to dump the CELT, and it finished at 2025/01/03 10:23.

Executing Check Telemetry - TM17: Command Executed

Telemetry parameter TM17 checked

(...)

Consulting triage agent for step 4/4

Triage Agent Response:
 The telemetry parameters for the Ping report are up-to-date. The last dump of CELT was on January 3, 2025, at 10:23.

Executing Check Telemetry - TM172: Ping report

Telemetry parameter TM172 checked

Bug reproduction completed - 4 steps executed

Final Summary:
 The reproduction process was successful, and the telemetry parameters are reporting as expected.

Figure 16: Alex - the results

4.4 Sara, the results

We present 3 examples that we achieved with Sara. In Example 01, Sara is able to understand the task, call the right tools to solve the problem and provide a correct answer. In Figure 17, we externalise the thought and reasoning the agent performs to try to understand how the result is achieved and the approach seems reasonable i.e. check if simulator is running, get the logs, parse these to understand if issue occurred.

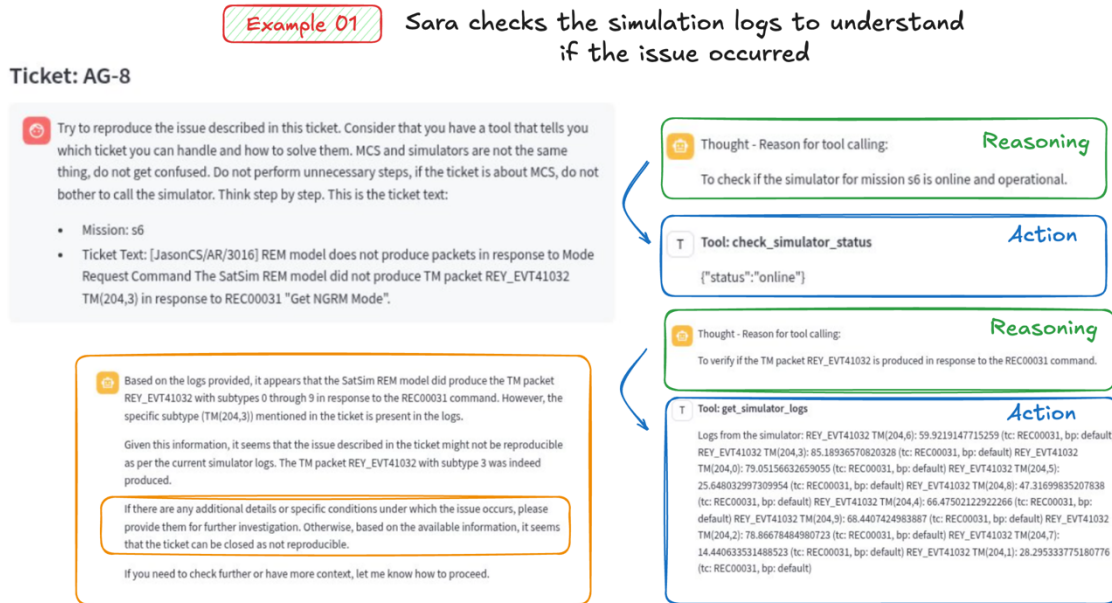


Figure 17: SARA, the results 01

In Figure 18, we ask Sara to resolve an activity outside of Sara's skill. In this case, Sara was able to detect that the request made by the user could not be resolved by its tool set.

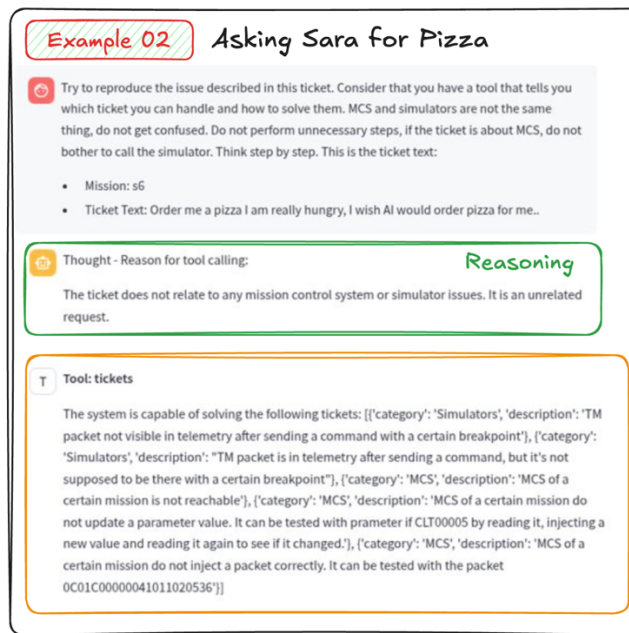


Figure 18: SARA, the results 02

Finally, with the example depicted in [Figure 19](#), we ask Sara to check an issue related to the Mission Control System, in this case, based on EGS-CC [24]. Again, in this example the steps to reproduce the issue seem reasonable i.e. considering the report that the injection mechanism was broken, Sara tries to inject the packet and then read a parameter.

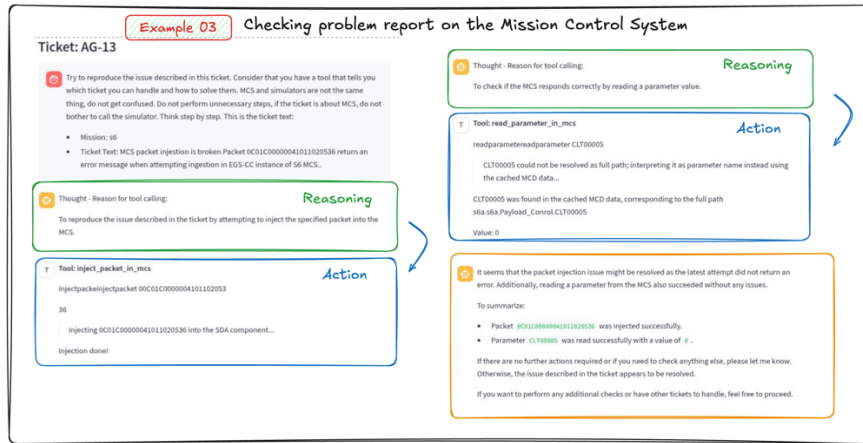


Figure 19: SARA, the results 03

5. Discussion

5.1 RAG

The work described in this article focused on understanding how recent developments in the domain of generative AI could be used to benefit the maintenance of spacecraft monitoring and control applications. In this domain, like others where systems have a very long lifetime, it is reasonable to expect that teams will experience relevant turnover throughout the years. From this viewpoint, the recent developments in generative AI and particular RAG present an enormous opportunity to evolve the current knowledge management ecosystems. Traditionally, content/knowledge management systems e.g. wikis, provide the users with repositories of explicit knowledge. From an architecture standpoint, these systems rely deterministic mechanisms such as indexes, tags, titles, categories, etc. to store and retrieve knowledge. From a form factor standpoint, this knowledge is made available through search mechanisms or tree views reflecting a specific viewpoint. This approach has limitations because the knowledge topology still needs to be managed, kept up to date, structured and re-structured as time passes.

RAG systems change the knowledge management paradigm in at least two ways. Firstly, users do not necessarily need to maintain the hierarchy of information e.g. hierarchy of tags to catalogue the information. Secondly, bring the capability to infer knowledge across multiple sources of information, something not really feasible in wikis and content management system. Of course, that, even if the inference mechanism is extremely useful, as demonstrated in Mofie, its value is driven by correctness of the inferred information. As demonstrated in Mofie's example 03, the ability to infer knowledge correctly depends on the ability to extract the correct information/chunks during the retrieval phase. In example 03, Mofie understood the search space/context, but provided the incorrect answer by inferring on the correlation of two valid inputs i.e. 1) MCS documentation allows identifying the Virtual Channel (VC) in use 2) What is the Auto PE system. These results were to be expected due to the Naive RAG, where the retrieval and generation phase are performed independently. In practice, the during the retrieval phase, Mofie still used a simple keyword matching strategy to identify the most relevant "X" chunks to be passed to the generator. More recent approaches such as Contextual RAG would have likely lead to better results. In Contextual RAG, the retrieval phase brings query awareness to the retrieval process by for instance reranking the chunks retrieved based on the user query. For instance, the retrieval mechanism would 1st identify the "M" most relevant chunks, but then this would be passed through a reranking mechanism (e.g. LLM that would sort the chunks for relevance with respect to the query) and finally the generator would pick the most relevant "X" chunks to be used for generation purposes. An additional or perhaps alternative route would be a combination of RAG and tool calling agent(s) – RAG 2.0. With this approach, in response to a user query, instead of simply retrieving + inferring an answer, the agent, could perform multi-step reasoning by using different tools to gather the relevant context and provide valuable answers.

A more complex knowledge management challenge regards addressing tacit knowledge. This knowledge is often flagged as not possible to persist because it is fundamentally bound to the knowledge, skills and abilities that individuals gain through experience. In [25], with Human-AI-Collaboration SECI model, the authors propose an extension to the SECI model where knowledge transfer is performed between human experts and their digital AI agent twin. The approach considers the usage of an agentic system where human and AI agents work together such that agents can learn, infer and persist tacit knowledge based on the interactions throughout the different experiences.

Will AI solve the knowledge management challenges in full? Some might argue it already did, still, there are many aspects to consider for instance, what are the limits to RAG ? How should the information be updated, replaced, extended? How does one ensure the boundaries of a dedicated RAG system i.e. should it be focused specifically in an area of expertise under a responsibility of a team? or should it be a single system across the organisation where teams are responsible for managing the collections i.e. areas of expertise ?

5.2 Agents and LLMs

As demonstrated in this work, AI agents can no longer be ignored. The ability to call external tools (Fiona, Sara, Alex), to implement reasoning loops (Sara) and organisation specific workflows (Alex, Sara) is today a reality that does not impose fundamental entry barriers for development/engineering teams. It should be noted that even if the agents created were developed and deployed with support to specialised hardware i.e. GPUs, the same agents were rehearsed in environments with lower specs and/or less requiring model setups e.g. lower quantization.

As introduced in section 1, the work aimed fundamentally at enabling engineering teams to understand the concepts, thinker with technology and the main frameworks within the context of mission control applications maintenance to be able to define realist roadmaps and appropriate goals. These technologies are fundamentally deflationary and will reduce development cycles and enabling individuals to enhance their abilities and achieve more with less. This will change the way maintenance and development activities are approached. For instance, will requirements be written and managed in the same way they are today? Or will these become more fluid and an integral part of the development flow/code generation? Will the design of applications be done using model-based engineering tools or will the design be achieved as the output of a discussion between an agent and different stakeholders? Will code reviews become more like coaching sessions against coding agents where the programmer gets feedback on how to improve? Will test campaigns be fully delegated to agents? And if so, how does one gain confidence on the results we will be delivered? Should the Technology readiness levels [28] evolve as a knock-on effect of this technologies as proposed in [27]? Even if reports such as [2] indicate that the scientific research did observe a major increase between 2022 and 2023, it is clear that there are several questions that deserve reflection before having these AI based applications in production for mission critical domains. Since most software is deemed as non-safety critical one can expect what is already demonstrated with the multiple AI frameworks i.e. there will be a proliferation of agents in support to different workflows within organisations.

In this regard, a relevant question would be, what should the best route be for development of agentic systems within organisations? Should we aim at a vertical approach where systems are high specialised and not so flexible but excel at specific tasks or should lean towards horizontal approach where agentic architectures are more general, less specialised, and do more but less well? Considering the work carried out, the pace of technology evolution and the fact that there are many systems today for mission control are non AI-enabled, it is easy advocate for aiming at vertical integration.

5.3 Form factor

One aspect not explicitly covered in this work regards the form factor of future applications. What should applications look like in a world with AI agents, RAG systems etc.?

Between 1950-1070's software interfaces were in essence command line interfaces (CLI) i.e. MS-DOS, Unix. The following decade presented the rise of GUIs with the first Windows and MacOS user interfaces. The 90's brought the web-ui's with browsers, html, javascript and hyperlinks. The 2000's brought an enrichment in terms of user experience to the web browser ecosystem with more powerful frameworks and later the transition to mobile. Finally, in recent years we observe the not so successfully attempt to introduce virtual-reality devices etc. With LLMs, agents that can infer, learn, keep short- and long-term memory will we finally depart from the paradigm in place since the appearance of modern computing where individuals are to a large extent locked interfacing with a computer using a keyboard and mouse? If so, what should the workflows look like and what should the observability criteria allow humans to monitor the execution status? Simple dashboards to observe the status of a workflow at this point seem more like a "debugging" tool rather than the abstraction layer needed to manage these future smart systems.

6. Conclusion

Today, within the domain monitoring and control of spacecraft systems, there is substantial research done around spacecraft fault detection and prediction using AI and specifically machine learning [29] [30], [31]. The work presented focuses on using LLMs for domain specific knowledge management and as a way to define workflows based on agents that can act on monitoring and control applications. LLMs are a type of machine learning models designed for natural language processing and generation applied AI and these are the core features we have demonstrated to interact with mission control applications that were developed before the advent of LLMs and therefore not really enabled for that purpose. The work shows that the entry barriers to the usage of this stack is not substantial and that non-AI engineering teams can and should adopt these ecosystems as a way to leverage/augment their capabilities. It is noted that, there we only have assessed few constructs in this field and even those were evaluated with a very exploratory mindset. The aim was understanding the technology and demonstrating the potential applied to the domain rather than having production ready applications. Production ready applications would require other considerations such as the usage of short and long term memory, agentic architectures and workflows. It is clear that the tool calling paradigm is relatively mature and can be easily used, as demonstrated with Alex, Sara and Fiona. ReAct [9] agents on the other hand, also proved a level of reliability but are still a bit engaging as the “thought” process leading to an action needs to be understood if one ones wants to have a better control on the outcome. This in turn requires having in the development phase observability mechanisms that explain the thought process and configuration mechanism that enable evaluating the outcome of changes in the setup e.g. Agent system message.

Finally, answering the paper main question, can AI help in the maintenance of monitoring and control applications? The answer is: yes, AI can help, still one must acknowledge that today, AI still needs a lot of help. As presented in this work, the agent behaviour is severely determined/affected by the quality of the input data and prompting provided to the LLM and this again points in the direction to have better tooling in support to the development of AI based applications.

References

- [1] Harvard, AI index report <https://aiindex.stanford.edu/report/> (accessed 19.02.25)
- [2] Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M.S., Bohg, J., Bosselut, A., Brunskill, E. and Brynjolfsson, E., 2021. On the opportunities and risks of foundation models, *arXiv preprint arXiv:2108.07258*.
- [3] Rick Merritt What are foundation models <https://blogs.nvidia.com/blog/what-are-foundation-models/> (accessed 19.02.25)
- [4] Vyasa, V. and Xu, Z., 2024. Maintenance in automotive and aerospace applications—An overview. *International Journal of Advances in Science Transactions*, 3(2), pp.349-361.
- [5] Sauvola, J., Tarkoma, S., Klemettinen, M., Riekkki, J. and Doermann, D., 2024. Future of software development with generative AI. *Automated Software Engineering*, 31(1), p.26.
- [6] ESA Operational Software Systems, <https://esoc.esa.int/services-software> (accessed 19.02.25)
- [7] Few shot prompting, <https://www.promptingguide.ai/techniques/fewshot> (accessed 19.02.25)
- [8] Tool Calling, <https://ollama.com/blog/tool-support> (accessed 19.02.25)
- [9] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. and Cao, Y., 2023, January. React: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR).
- [10] Marina Danilevsky (2023), What is Retrieval-Augmented Generation (RAG)? <https://www.youtube.com/watch?v=T-D1OfcDW1M&pp=ygUHUKFHIEICTQ%3D%3D> (accessed 19.02.25)
- [11] Vinay Kumar (2025) LLM building blocks -Tokens and Embeddings <https://vinaykuma201.medium.com/llm-building-blocks-tokens-and-embeddings-part-2-a18a531b4bdc> (accessed 19.02.25)
- [12] Martin Keen (2025), LangChain vs LangGraph: A Tale of Two Frameworks, <https://www.youtube.com/watch?v=qAF1NjEVHhY> (accessed 19.02.25)
- [13] Chroma, ChromaDb Docs, <https://docs.trychroma.com/> (accessed 19.02.25)
- [14] Command-r 35b https://ollama.com/library/command-r:35b-08-2024-q5_K_M/blobs/911476c254c7 (accessed 19.02.25)
- [15] GitLab, <https://en.wikipedia.org/wiki/GitLab> (accessed 19.02.25)
- [16] A Scalable Architecture for integrating AI and Non-AI Agents in Space Mission Control Applications, J. Feiteirinha et al, SpaceOps 2025

- [17] Basics of Django: Model-View-Template Architecture, <https://angelogentileiii.medium.com/basics-of-django-model-view-template-mvt-architecture-8585aecffbf6> (accessed 19.02.25)
- [18] CORBA, https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture (accessed 19.02.25)
- [19] IDL, https://en.wikipedia.org/wiki/Interface_definition_language (accessed 19.02.25)
- [20] What is JSON?, https://www.w3schools.com/whatis/whatis_json.asp (accessed 19.02.25)
- [21] Qwen2.5-32B, <https://huggingface.co/Qwen/Qwen2.5-32B> (accessed 19.02.25)
- [22] Streamlit, <https://streamlit.io/> (accessed 19.02.25)
- [23] GitOps, <https://about.gitlab.com/topics/gitops/>
- [23] Qwen2.5-Coder, <https://arxiv.org/abs/2409.12186> LLM (accessed 19.02.25)
- [24] European Ground System - Common Core (EGS-CC), <https://en.wikipedia.org/wiki/EGS-CC> (accessed 19.02.25)
- [25] Matsumoto, T., Nishikawa, R. and Morimoto, C., The Knowledge Management Model of the Experts' Tacit Knowledges with Augmented LLM-Based AI. In Agents and Multi-agent Systems: Technologies and Applications 2024: Proceedings of 18th KES International Conference, KES-AMSTA 2024, June 2024 (p. 135). Springer Nature.
- [26] Nonaka, I., Toyama, R. and Konno, N., 2000. SECI, Ba and leadership: a unified model of dynamic knowledge creation. *Long range planning*, 33(1), pp.5-34.
- [27] Lavin, A., Gilligan-Lee, C.M., Visnjic, A., Ganju, S., Newman, D., Ganguly, S., Lange, D., Baydin, A.G., Sharma, A., Gibson, A. and Zheng, S., 2022. Technology readiness levels for machine learning systems. *Nature Communications*, 13(1), p.6039.
- [28] TRL, https://en.wikipedia.org/wiki/Technology_readiness_level (accessed 19.02.25)
- [29] Siddique, I.M., 2024. Detection and Analysis of Anomalous Behavior in On-Orbit Satellites Using AI Algorithms. *Journal of Firewall Software and Networking*, 2(2), pp.10-48001.
- [30] Wang, Y., Yuan, X., Lin, Y., Gu, J. and Zhang, M., 2023. A semi-supervised multi-scale deep adversarial model for fan anomaly detection. *IEEE Transactions on Consumer Electronics*.
- [31] Aziz, T.A., Salama, G.I., Mohamed, M.S. and Hussein, S., 2023, November. Spacecraft fault detection and identification techniques using artificial intelligence. In *Journal of Physics: Conference Series* (Vol. 2616, No. 1, p. 012025). IOP Publishing.