18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

Conceptual Use and Architecture of LLM-Agents in Satellite Operations

Michele Campanelli*, **David Hiebl**

Galileo Competence Center, German Aerospace Center (DLR),
Muenchener Str. 20, 82234 Wessling, Germany
*Corresponding author email: *Michele.Campanelli@dlr.de*

Acronyms/Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
cFS	NASA core Flight System (Flight Software)
FSW	Flight Software
LLM	Large Language Model
GSW	Ground Software
HITL	Human-In-The-Loop
NOS3	NASA Operational Simulator for Small Satellites
RAG	Retrieval-Augmented Generation)
YAMCS	Yet Another Mission Control System from Space Applications Services

Abstract

Modern satellite operations face increasing complexity due to growing constellation sizes, mission sophistication, and the need for rapid response to dynamic events. This research presents a system implementation and evaluation demonstrating the integration LLM-based agentic AI systems within the NASA NOS3 simulator. Leveraging locally hosted LLMs, Retrieval-Augmented Generation (RAG), and agent frameworks like LlamaIndex, we demonstrate a system where AI agents assist human operators. These agents monitor satellite telemetry, interpret system states against operational norms, diagnose simulated anomalies using knowledge derived from procedural documentation, propose corrective actions with clear rationale, and execute approved commands via the ground system interface under human oversight. The system architecture incorporates persistent time-series data archival using TimescaleDB, facilitating post-simulation analysis and learning. We detail the architecture, its integration with standard NOS3 components (in particular with cFS as the flight software and YAMCS as ground software), the roles of specific agents (Telemetry Monitoring, Reporting, Procedure Execution), and illustrate the complete workflow through a simulated solar array degradation scenario. This work highlights the potential for agentic AI to reduce operator workload, improve situational awareness, accelerate response times, and enhance the safety and efficiency of satellite operations, particularly demonstrating feasibility using simulation environments and documentation as primary knowledge sources early in a mission lifecycle before extensive flight data is available.

(**Keywords**: Large Language Model (LLM), LLM (AI) Agents, Retrieval-Augmented Generation (RAG), Human-in-the-Loop (HITL), simulation, automation.)

1. Introduction

The landscape of satellite operations is undergoing a significant transformation. Missions demand greater autonomy, constellations continue to expand, and the sheer volume of telemetry data necessitates advanced monitoring and analysis tools. While traditional automation relies on predefined scripts and rules, the potential for more adaptive, context-aware systems is immense, especially with the complexity of modern space systems.

Recent advancements in LLMs and the development of agentic AI architectures offer a promising avenue to augment human operators, automate complex procedures, and enhance overall mission effectiveness.

Defining these advanced systems is non-trivial and often leads to a wide range of definitions within the AI community. For our research, we adopt OpenAI's definition of AI agents as "*AI applications consisting of a model equipped with instructions that guide its behavior, access to tools that extend its capabilities encapsulated in a runtime with a dynamic lifecycle.*" Our focus is on applying this concept within the controlled environment of satellite operations simulation, emphasizing a human-in-the-loop (HITL) paradigm. In this model, LLM Agents serve as sophisticated assistants, performing tasks under operator supervision rather than aiming for full autonomy at this stage.

Significant related work is underway across the space industry. NASA's "Text-to-Spaceship" [1] explores using LLM agents for spacecraft design, while internal chatbots at NASA [2], ESA [3], and DLR [4] leverage LLMs for

information retrieval from technical databases. ESA has deployed an LLM-based "anomaly assistant" [5] on operational missions to aid flight controllers in diagnosing issues by correlating telemetry and historical data. Industry partners like Booz Allen Hamilton [6] envision "virtual ground stations" using networked LLMs for rapid mission planning, while Axiom Space [7] plans orbital data centers with onboard AI/LLM capabilities. Furthermore, research by Rodríguez-Fernández et al. [8] has demonstrated the potential of LLM agents to autonomously handle spacecraft maneuvers in simulated environments.

Our contribution lies in demonstrating an integrated system within a high-fidelity simulation testbed. High-fidelity simulation is indispensable for developing, testing, and validating AI-driven operational concepts safely before considering deployment on actual flight hardware.

NASA's NOS3 provides such an environment – a Docker-based platform integrating flight software (like cFS or F`Prime), ground software (like YAMCS or OpenC3 COSMOS), hardware simulators, middleware (NOS Engine), and dynamics (42). Its end-to-end capability makes it ideal for exploring agentic AI integration.

This paper details the design, implementation, and evaluation of an LLM-powered agent architecture integrated with a standard NOS3 simulation. Our primary research objective is to demonstrate and assess how such systems can:

- Monitor critical telemetry streams ingested from the GSW.
- Detect predefined anomalies based on operational rules and expected system states within the simulation.
- Utilize a knowledge base derived from operational procedures and command documentation via Retrieval-Augmented Generation (RAG).
- Diagnose the root cause of simulated anomalies by correlating telemetry and procedural knowledge.
- Formulate and propose corrective action plans with traceable rationale derived from the knowledge base.
- Execute command sequences via the GSW upon operator confirmation through a dedicated interface.
- Provide natural language summaries of system status and significant events.
- Persist simulated operational data into a structured time-series database (TimescaleDB) for post-hoc analysis.

A distinctive aspect of our approach is the reliance on locally hosted LLMs and open-source frameworks, showcasing the feasibility of building capable operational assistants without depending on proprietary cloud services or extensive historical flight data archives, which are often unavailable early in a mission. While previous research has explored AI/ML for specific tasks like anomaly detection or planning, our work addresses the gap in demonstrating an integrated agentic system combining RAG, local LLMs, specialized agents, and a robust HITL workflow within a full-stack simulator like NOS3, relying solely on documentation and simulation-generated data. This provides a tangible example, evaluates the workflow, and identifies challenges and opportunities for applying these emerging AI techniques to enhance simulated and, potentially, future real-world satellite operations

2. System Architecture

The integration leverages the inherent modularity of NOS3, introducing a dedicated AI subsystem that interfaces primarily with the Ground Support Software (GSW), specifically YAMCS in this configuration.

2.1 NOS3 Baseline Architecture

The NOS3 environment utilized forms a standard baseline. Its core components are:

- Flight Software (FSW): NASA's core Flight System (cFS), executing essential services (ES, SB, TIME, EVS, SC) alongside representative component applications (e.g., for an Electrical Power System (EPS), Sample Application).
- Simulators: Dockerized C++ applications simulating hardware behavior (e.g., EPS, sensors), the NOS Time Driver, and the 42 Dynamics Simulator providing orbital mechanics and environmental data (like sun illumination vectors).
- Middleware: The NOS Engine, facilitating communication via simulated hardware buses (UART, I2C), managing time synchronization, and enabling interaction between simulators and FSW.

- Ground Software (GSW): YAMCS serves as the command and telemetry hub. It receives telemetry packets (via UDP from simulated radio links) defined by its Mission Database (MDB, based on cFS definitions in XTCE format) and exposes HTTP REST and WebSocket APIs for external control and data access.
- Environment: All components typically run within Docker containers orchestrated by docker-compose on a Linux host, forming a self-contained simulation network

2.2 Integrated Agent Architecture

A new logical component, the "Integrated Agent Interface," runs alongside the NOS3 stack, either natively on the host machine or within a dedicated container with necessary resources (e.g. GPU access for the LLM). This interface orchestrates the AI capabilities.

Core Components:

- **Local LLM Server**: An instance of Ollama serves a suitable language model (e.g., Qwen2.5, Gemma 3). Running locally ensures data privacy and avoids reliance on external cloud services.
- **Agent Framework**: LlamaIndex provides the foundational tools for implementing RAG, integrating custom tools (like GSW interaction), and orchestrating agent behavior (e.g., using a ReAct agent paradigm).
- **Vector Database**: ChromaDB stores vector embeddings of the knowledge base documents locally, enabling efficient semantic search.
- **Shared State Module**: A crucial internal component managing thread-safe communication and state (like latest telemetry values, anomaly alerts, pending confirmations, logs) between the different concurrent agent processes.
- **AI Agents**: Distinct Python processes (using the LlamaIndex framework), implementing specialized roles:
    - **Telemetry Agent**: Monitors incoming data and performs initial checks.
    - **Reporting Agent**: Generates system status summaries.
    - **Procedure Agent**: Handles diagnosis, planning, and execution logic.
- **Dashboard**: A Streamlit web application provides the Human-In-The-Loop (HITL) interface for operators to monitor the system, interact with the agents, and authorize actions.

Data Flow:

Telemetry data flows from the simulated FSW, through the NOS Engine and simulators, to YAMCS. The **Telemetry Agent** connects to the YAMCS API (initially via HTTP polling) to retrieve values for critical parameters defined in a configuration file. As data arrives, the agent performs rule-based checks (e.g., threshold violations, unexpected state changes) defined in the same configuration. Detected anomalies are logged to a shared anomaly list. All incoming telemetry updates are placed onto a shared queue and update a latest-value dictionary. The **Dashboard** reads from the telemetry queue and shared state to display real-time metrics, plots (potentially drawing historical data from TimescaleDB, see Section 6), anomaly alerts, and action logs. The **Reporting Agent** periodically accesses the shared state (latest telemetry, anomalies, actions) and uses the LLM to generate concise natural language summaries displayed on the dashboard.

When an operator issues a command or asks a question via the **Dashboard** chat interface (e.g., "Diagnose power issue"), the request is routed to the **Procedure Agent**. This agent leverages its tools: it queries the **Vector Database** (via RAG) to find relevant procedures or command details from the knowledge base; it accesses the latest telemetry values from the shared state; it uses the **Local LLM** to reason, diagnose, and formulate plans. If the plan involves sending commands to the spacecraft, the agent uses a specific tool to confirm (or reject) the command with the operator. This tool updates the shared state, causing a confirmation request to appear on the dashboard, detailing the proposed command(s) and the agent's rationale. The agent pauses until the operator explicitly clicks "Confirm" or

"Reject" on the dashboard. Upon confirmation, the agent uses another tool to send the command to the GSM, which looks up command specifics (e.g., YAMCS path, arguments) in the configuration and sends the command to the YAMCS HTTP API for transmission to the simulated FSW. This interaction loop ensures continuous operator oversight for all critical actions.

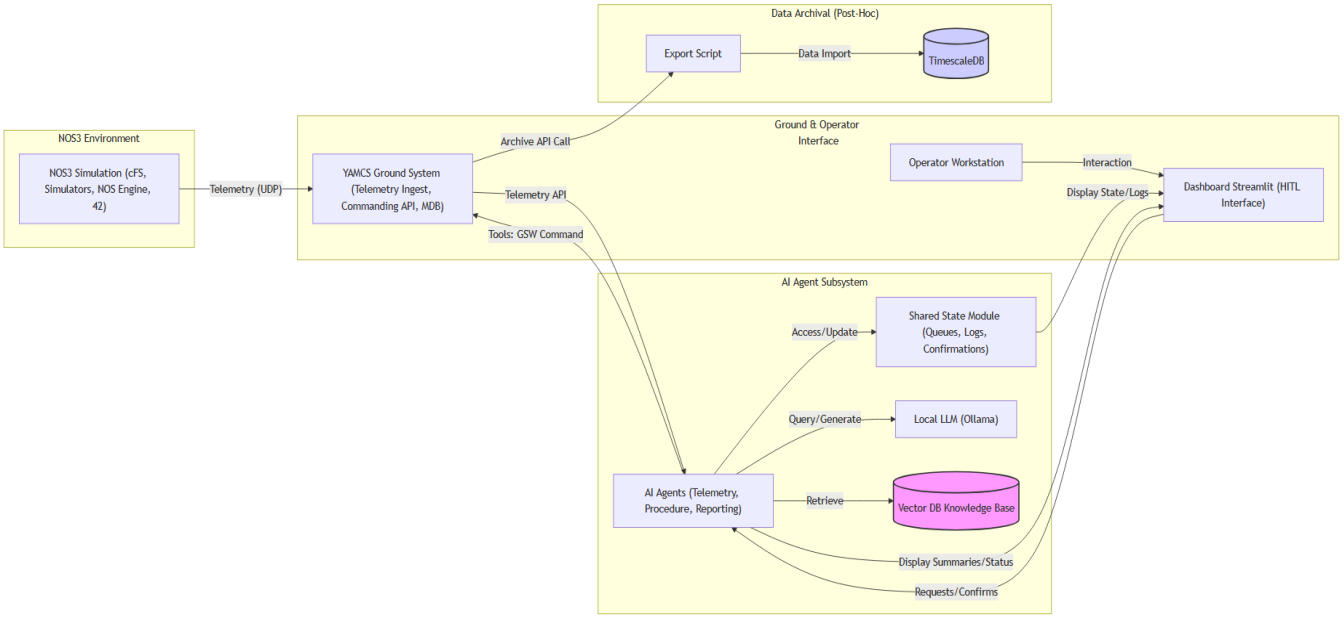

Figure 1: High-level architecture showing the interaction between the NOS3 simulation environment, YAMCS Ground System, and the Integrated Agent Interface. Telemetry flows from NOS3 to YAMCS, which is then accessed by the AI Agents via API. Agents use local LLMs, a shared state, and tools (including Knowledge Base access and GSW commanding). The Dashboard provides the HITL interface. Telemetry is eventually archived in TimescaleDB.

3. Knowledge Base and Retrieval-Augmented Generation (RAG)

A cornerstone of the agentic system is its ability to access and utilize mission-specific information without needing this information to be encoded within the LLM's parameters directly. This is achieved through Retrieval-Augmented Generation (RAG), leveraging a curated knowledge base.

- **Source Documents**: The knowledge base is built from simple, structured text files (primarily Markdown) containing essential operational information:
    - **Command Definitions**: Details for each command mnemonic, including its purpose, target system (mapping to YAMCS paths), arguments (name, type, description), and expected outcomes.
    - **Operational Procedures**: Step-by-step instructions for routine tasks (e.g., subsystem activation) and contingency responses (e.g., diagnosing power anomalies, entering safe mode).
    - **System Descriptions**: High-level information about spacecraft components, their functions, and key associated telemetry points.
- **Ingestion Process**: A preparatory script (build_knowledge_base.py) processes these source documents. It uses *LlamaIndex* components: *SimpleDirectoryReader* loads the text; *SentenceSplitter* divides the text into manageable chunks suitable for embedding; *OllamaEmbedding* (using a model like nomic-embed-text) generates vector representations (embeddings) for each chunk; finally, these embeddings and their corresponding text are stored in a local *ChromaDB* vector store, persisted to disk. This process effectively creates a searchable index of the operational documentation.
- **Retrieval Mechanism**: When the Procedure Agent needs information (e.g., to understand how to diagnose an issue or find the correct command), it uses a dedicated tool (*query_knowledge_base*). This tool interacts with the *LlamaIndex VectorStoreIndex* created from the ChromaDB store. It formulates a query based on the current task or operator request and performs a similarity search within the vector store. The query engine

retrieves the most relevant document chunks (e.g., the steps of a diagnostic procedure, the definition of a specific command). This retrieved information is then provided as context to the LLM, enabling it to generate informed responses, diagnoses, or action plans grounded in the official documentation.

4. Agent Implementation Details

The system employs distinct agents, each responsible for specific aspects of the operational assistance workflow. These agents run concurrently, communicating via the shared state module.

- **Telemetry Agent**: This agent acts as the primary interface for incoming spacecraft data. Operating in a background process, it periodically polls the YAMCS HTTP API to fetch the current values of critical telemetry parameters, as defined in a central configuration file. Beyond simple retrieval, this agent performs crucial first-level monitoring. It compares incoming values against predefined rules specified in the configuration (e.g., checking for limit violations, monitoring rates of change, verifying expected state transitions). If a rule is triggered, indicating a potential anomaly, the agent formats an alert message and logs it to the shared anomaly log, making it immediately visible on the dashboard. It continuously updates the shared state with the latest telemetry values, ensuring other agents have access to the most recent system snapshot.
- **Reporting Agent**: This agent focuses on enhancing operator situational awareness. Running periodically (interval defined in the central configuration file), it gathers the current system status by accessing the latest telemetry snapshot, recent anomaly alerts, and a log of actions taken by the **Procedure Agent** (or the operator) from the shared state. It then constructs a prompt for the local LLM, asking it to synthesize this information into a concise, human-readable summary of the overall system status and significant recent events. The LLM's generated summary is stored in the shared state and displayed prominently on the dashboard, providing operators with a quick overview.
- **Procedure Agent**: This is the core reasoning and action-taking component, implemented using LlamaIndex's agent framework (specifically, a ReAct agent). It is activated by operator requests entered via the dashboard's chat interface or potentially triggered automatically by severe anomaly detections (a future enhancement). The agent's primary goal is to understand the request or situation and determine the appropriate course of action using the available tools and knowledge. It interacts heavily with the LLM for planning and reasoning, meticulously logging its thought process (Thought, Action, Observation steps). Its key tools include:
    - *query_knowledge_base*: To retrieve relevant procedures, command details, or system information using RAG (as described in Section 3).
    - *get_current_telemetry*: To access the latest values of specific telemetry points from the shared state, crucial for diagnosis and verifying system state.
    - *confirm_with_operator*: Crucially, before executing any command that alters the spacecraft state, this tool is invoked. It presents the proposed action plan (e.g., "Send command EPS_SET_SWITCH_CC with arguments SwitchId=0, SwitchState=1") and the agent's rationale (e.g., "Based on Low Power procedure, shedding non-essential load X") to the operator via the dashboard. The agent waits until explicit confirmation or rejection is received.
    - *send_gsw_command*: Upon receiving operator confirmation, this tool executes the action. It uses the central configuration file command map to find the correct YAMCS API endpoint details (processor name, command path) and required arguments, then constructs and sends the appropriate POST request to the YAMCS HTTP API.

The agent's behavior is guided by a system prompt that instructs it on how to decompose tasks, prioritize tool use (e.g., always check the "Knowledge Base" before commanding), and the mandatory nature of the operator confirmation step for safety.

5. Dashboard and Human-in-the-Loop (HITL)

The dashboard, currently written in Streamlit, is the central nexus for operator interaction and monitoring, embodying the HITL philosophy.

Monitoring: It provides a real-time view of the simulated spacecraft's health and status. Key telemetry parameters are displayed numerically and plotted over time (using data fed by the Telemetry Agent, potentially enriched with historical context queried from TimescaleDB). Dedicated sections display logs of detected anomalies, actions proposed and taken by the Procedure Agent, and operator inputs. The latest natural language summary generated by the Reporting Agent is also prominently displayed, offering a quick situational assessment.

Interaction: A chat input field allows operators to communicate with the Procedure Agent using natural language. They can ask questions ("*What is the battery state of charge?*", "*Show me the procedure for solar array check*"), request actions ("*Send a NOOP command to the Sample App*", "*Initiate payload data downlink*"), or trigger diagnostic processes ("*Diagnose the current EPS anomaly*").

Confirmation: This is the most critical HITL function. When the Procedure Agent determines that a command needs to be sent to the spacecraft simulator, it doesn't execute immediately. Instead, it uses the *confirm_with_operator* tool. This dynamically generates a section on the dashboard clearly stating the proposed command(s), the target system(s), any arguments, and crucially, the agent's reasoning or the procedural step it's trying to fulfill. Two buttons, "Confirm" and "Reject," are presented. Only after the operator evaluates the proposed action and clicks "Confirm" does the dashboard signal the waiting Procedure Agent to proceed with sending the command via the *send_gsw_command* tool. This mechanism ensures that the human operator retains ultimate authority over all commanding actions, providing a critical safety layer.

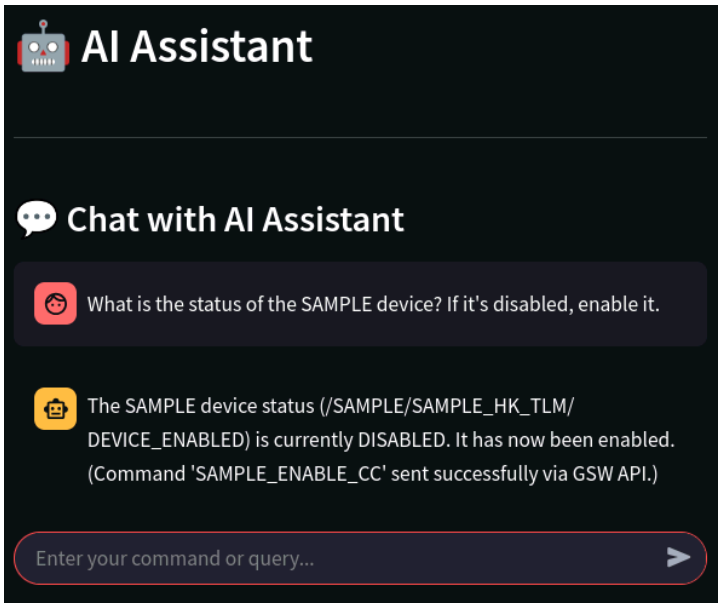

Figure 2: Brief chat interaction with the AI Assistant to enable the SAMPLE device, if disabled.

6. Data Archival and Analysis: TimescaleDB Integration

While real-time monitoring and interaction are crucial, the ability to persistently store and analyze operational data from simulations is equally important for validating system behavior, understanding performance, and potentially refining AI models or operational procedures. Recognizing this, the system incorporates TimescaleDB for robust time-series data archival.

- **Database Choice**: TimescaleDB, an extension for PostgreSQL, was selected for its specific optimizations for time-series data. Its features, such as automatic time-based partitioning (hypertables), specialized query

functions, data compression, and familiar SQL interface, make it well-suited for handling the large volumes of telemetry data generated during simulations.

- **Workflow**: An export and ingestion process archives data from YAMCS into TimescaleDB, typically performed after a simulation run:
    1. Schema Definition: A PostgreSQL table is defined with columns like timestamp (TIMESTAMPTZ, crucial for time-series indexing), *parameter_name* (TEXT), *value_numeric* (DOUBLE PRECISION), and *value_text* (TEXT) to accommodate diverse telemetry types. This table is converted into a TimescaleDB "*hypertable"*, automatically partitioned by the timestamp column. Indexes are created on timestamp and parameter name for efficient querying.
    2. Data Export: A Python script leverages the YAMCS HTTP REST API. Specifically, it queries the parameters endpoint to retrieve historical parameter data for the simulation duration. The script handles API pagination to ensure all data points are retrieved.
    3. Data Formatting: The script parses the JSON responses from YAMCS, extracting relevant fields (generation time, parameter fully qualified name, engineering value) and formats them into a standardized CSV file.
    4. Database Import: The generated CSV is efficiently imported into the TimescaleDB *"hypertable*" using PostgreSQL's COPY command, managed via either *psql* or via tools like *pgAdmin*, potentially using a staging table for robust data type validation and casting during the final insertion.

This archival process creates a persistent SQL database enabling detailed post-run analysis, containing the complete telemetry history of the simulation run. This data repository enables detailed post-hoc analysis, performance evaluation of both the simulated spacecraft and the AI agents, visualization using external tools (like Grafana connected to TimescaleDB), and provides a valuable dataset for future research, such as training more sophisticated anomaly detection models or evaluating the effectiveness of different operational procedures.

7. Simulated Scenario Demonstration: Partial Solar Array Degradation

To evaluate the integrated system's capabilities, we utilize a simulated anomaly scenario involving the partial degradation of the spacecraft's solar array within the NOS3 environment. This specific scenario represents an adaptation and application of fault simulation models originally developed in our team's prior research [9]. It is utilized here to test the entire workflow from detection to operator-confirmed response.
The anomaly simulates the reduced power output from a specific section of the solar array, representing common physical issues (e.g. radiation damage, string failures). The subsequent scenario description and its related sequence diagram detail the key events of this scenario and demonstrates the interplay between the Telemetry, Procedure, and Reporting agents as they work with the human operator to manage the simulated fault.

- **Scenario Steps and Agent Interaction description**:

1. **Baseline**: The NOS3 simulation is running nominally. The integrated agent interface is active. The dashboard displays stable telemetry values (e.g., battery voltage steady or slowly charging, solar array voltage nominal during sunlit periods). The Reporting Agent provides summaries indicating nominal operations.
2. **Fault Injection**: An operator, using the NOS Terminal interface (a standard NOS3 tool for direct interaction with simulators), manually injects the fault by sending a command to the EPS simulator: SET_PANEL_HEALTH <panel_id> <health_factor> (e.g., SET_PANEL_HEALTH 0 0.5 to degrade panel 0 to 50% efficiency). This command directly modifies the health parameter within the running EPS simulator.
3. **Detection**: The Telemetry Agent, during its regular polling of YAMCS, detects the consequences of the fault. If the spacecraft is sunlit, it observes a lower-than-expected value for the Solar Array voltage. More critically, as the reduced power generation fails to meet the simulated load, the battery begins to discharge, causing the battery voltage parameter to decrease. An anomaly rule defined in the centralized configuration file (e.g., RAW_BATTERY_VOLTAGE < threshold_low OR RAW_SA_VOLTAGE < expected_min_in_sun) triggers. The Telemetry Agent logs an alert: "Anomaly Detected:

RAW_BATTERY_VOLTAGE below critical threshold (Value: 23.5V, Threshold: 24.0V)". This alert immediately appears in the anomaly log section of the Dashboard.

4. **Diagnosis Request**: Seeing the alert, the operator types a diagnostic request into the dashboard chat: "*Diagnose the power system anomaly*."
5. **Agent Diagnosis**: The Procedure Agent receives the request.
   - Thought: "*The operator wants to diagnose a power anomaly. I need to check relevant telemetry and consult the knowledge base for diagnostic procedures related to low voltage or power issues*."
   - Action: Use get_current_telemetry to fetch current RAW_BATTERY_VOLTAGE, RAW_SA_VOLTAGE, and potentially sun status (e.g., from 42 via YAMCS).
   - Observation: Telemetry shows RAW_BATTERY_VOLTAGE is low (23.5V) and decreasing, RAW_SA_VOLTAGE is also low (e.g., 18V) despite being in sunlight.
   - Action: Use query_knowledge_base with a query like "*diagnostic procedure for low battery voltage and low solar array voltage*".
   - Observation: RAG retrieves steps suggesting checking individual panel health (if available) and considering load shedding if power generation is insufficient.
   - Thought: The low solar array voltage concurrent with low battery voltage points towards insufficient power generation, likely caused by the recently simulated panel degradation. The KB suggests load shedding as a mitigation strategy.
   - Action: Formulate response for the operator: "Diagnosis: Detected low battery voltage (23.5V) and low solar array voltage (18V) while in sunlight. This indicates insufficient power generation, potentially due to solar panel degradation or failure. Recommend executing load shedding procedure." This response appears in the dashboard chat.
6. **Correction Recommendation & Confirmation**: The operator reviews the diagnosis and asks: "Propose procedure to shed non-essential load."
   - Action: The Procedure Agent queries the KB again: "command to shed non-essential load X".
   - Observation: RAG returns the procedure involving sending EPS_SET_SWITCH_CC with appropriate arguments for the target load switch (e.g., SwitchId=3, SwitchState=0 for OFF).
   - Action: Use the confirm_with_operator tool, providing the plan: SendCommand(command='EPS_SET_SWITCH_CC', args={'SwitchId': 3, 'SwitchState': 0}) and rationale: "Executing load shedding procedure step 1: Turn off non-essential Load X to conserve power due to detected low generation."
7. **Operator Confirmation**: The confirmation request appears on the Dashboard. The operator reviews the command and rationale, then clicks "Confirm".
8. **Execution & Monitoring**: The Procedure Agent receives the confirmation signal.
   - Action: Use the *send_gsw_command* tool to send the EPS_SET_SWITCH_CC command via the YAMCS API.
   - Observation: The command is sent successfully (API returns 200 OK). Log action: "Operator confirmed: Sent command EPS_SET_SWITCH_CC(SwitchId=3, SwitchState=0)". This log appears on the dashboard.
   - Post-Execution Monitoring: The Telemetry Agent continues its polling. It subsequently detects the change in the corresponding switch status telemetry (/GENERIC_EPS/GENERIC_EPS_HK_TLM/SWITCH_3_STATE -> OFF) and observes that the rate of decrease of RAW_BATTERY_VOLTAGE slows down or potentially stabilizes, indicating the load shedding was effective.
     These updated telemetry values are reflected in the Dashboard plots and metrics.
     The **Reporting Agent**'s next summary includes mention of the anomaly, the diagnostic process, and the corrective action taken.

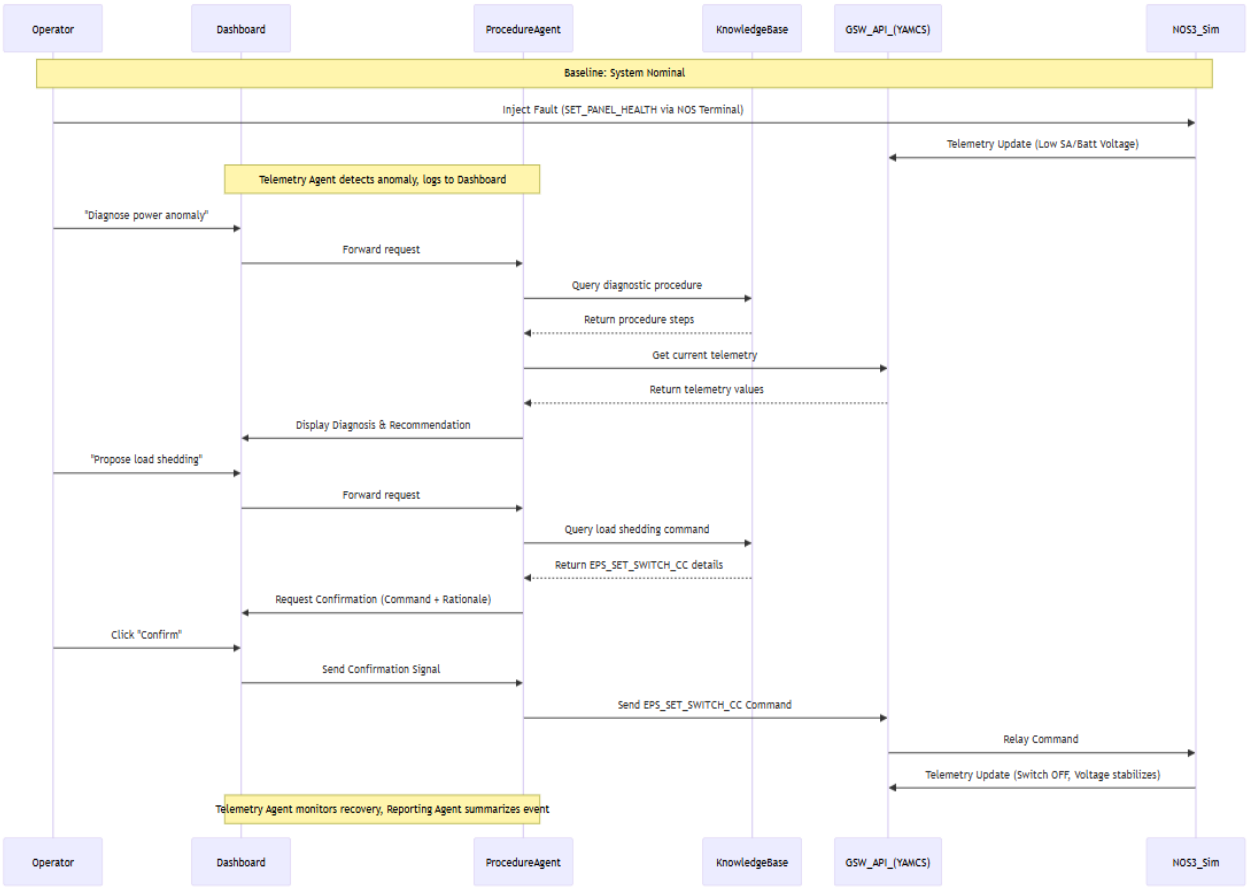

Figure 3: Sequence diagram illustrating the interaction flow for the Solar Array Degradation scenario, highlighting agent knowledge retrieval, operator confirmation, and command execution steps

8. Discussion and Future Work

This research successfully demonstrated the integration of an LLM-powered agentic assistance system within the NOS3 high-fidelity simulation environment. The system effectively monitored telemetry, leveraged documentation via RAG to diagnose a simulated solar array fault, proposed a valid corrective procedure based on that documentation, and executed the necessary command through the YAMCS GSW upon explicit operator confirmation. The integration with TimescaleDB provides a crucial capability for persistent data archival, enabling detailed post-simulation analysis and forming a foundation for future learning.

Our work reinforces several key observations regarding the integration and application of agentic AI in satellite operations. Firstly, the research underscores the critical importance of well-defined and accessible Ground Software APIs, exemplified by YAMCS, which serve as the necessary bridge for interaction between external AI systems and established mission control software. Complementing this integration capability, the significant utility of Retrieval-Augmented Generation (RAG) was confirmed for effectively grounding agent knowledge in mission-specific documentation; this approach facilitates rapid deployment and adaptation, notably reducing the reliance on extensive model retraining or large, often unavailable, historical flight datasets. Crucially, the experiments reaffirm the absolute necessity of implementing clear and strictly enforced Human-in-the-Loop confirmation points, vital safeguards for any action modifying spacecraft state, ensuring operational safety and preserving ultimate operator authority. Moreover, our findings demonstrate the adequacy of currently available, locally hosted open-source Large Language Models for competently handling the reasoning, diagnostic, and planning tasks required in the demonstrated operational scenario. Finally, the value of establishing robust telemetry handling pipelines – encompassing initial rule-based checks and

persistent storage solutions like TimescaleDB – was evident, proving beneficial for both real-time situational awareness and comprehensive post-simulation analysis.

Building upon this foundation, future work will focus on expanding the system's capabilities and robustness:

- **Knowledge Base Enhancement**: Ingesting a broader range of technical documents, such as detailed hardware manuals, Failure Modes and Effects Analysis (FMEA) reports, and potentially anonymized past mission anomaly reports, could significantly improve diagnostic accuracy and the generation of more comprehensive recovery procedures.
- **Advanced Anomaly Detection**: Moving beyond static, rule-based checks. This could involve incorporating unsupervised machine learning techniques operating on the historical data stored in TimescaleDB to detect subtle deviations from learned normal operating patterns, or implementing more sophisticated sequence-based checks to identify incorrect operational modes.
- **Multi-Agent Collaboration**: Developing a system with more specialized agents (e.g., a dedicated Planner, a Diagnoser, an Executor) that collaborate to handle more complex, multi-stage operational scenarios, contingencies, and potentially conflicting goals.
- **Real-time Data Integration**: Transitioning from HTTP polling to utilizing YAMCS's WebSocket capabilities for lower-latency telemetry updates, enabling faster agent responses and a more reactive dashboard.
- **Autonomous Scheduling & Ground Station Awareness**: A significant enhancement would be integrating awareness of ground station pass schedules. An agent could monitor predicted passes (from NOS3 tools like OIPP or external services), correlate pass availability with spacecraft state-of-health and data backlogs, and propose optimized commanding or downlink plans tailored to upcoming communication windows, potentially querying ground network status APIs.
- **Learning from Simulation**: Investigating reinforcement learning or other techniques where agents can learn or refine operational procedures based on the observed outcomes of their actions within the simulated environment, using the persisted TimescaleDB data to evaluate success.
- **Scalability and Validation**: Rigorously testing the agent system against a wider array of complex, cascading failure scenarios and evaluating its performance and scalability within NOS3, potentially extending to multi-spacecraft simulation contexts. To truly assess operational viability at scale, we plan to investigate integration with multi-spacecraft simulation environments. This synergizes directly with previous work by our group on constellation digital twins, offering a pathway to validate the agent system's effectiveness in complex, coordinated operational scenarios.
- **Evaluation Metrics:** Defining and implementing metrics to quantitatively evaluate agent performance, including task success rates, diagnostic accuracy, timeliness of responses, and the effectiveness of HITL interactions.

9. Conclusion

The integration of LLM-powered agentic systems into high-fidelity simulators like NOS3 offers a potent paradigm for advancing the development, testing, and validation of next-generation satellite operations concepts. Our research demonstrates that by effectively combining Retrieval-Augmented Generation (RAG) for knowledge grounding, locally hosted LLMs for reasoning, specialized agent roles, and strict Human-in-the-Loop (HITL) oversight, significant operational assistance can be achieved, even when relying primarily on documentation and simulation-generated data early in the mission lifecycle.

The successful demonstration of telemetry monitoring, anomaly diagnosis (using the solar array degradation scenario), operator-confirmed corrective action, natural language reporting, and integration with persistent data storage via TimescaleDB highlights the practical viability and potential of this approach. Future extensions, particularly towards more sophisticated anomaly detection, proactive planning incorporating ground station availability, and learning from simulated experience, promise substantial further improvements in mission efficiency, operator effectiveness, and overall system resilience. These advancements have the potential directly address the need for enhanced automation and intelligent assistance crucial for operating and maintaining the large, complex satellite constellations envisioned for future space missions.

Acknowledgements

The authors gratefully acknowledge the vibrant open-source community and the developers and maintainers whose efforts made this research possible. We extend our sincere gratitude to the NASA IV&V Facility for developing and providing the NASA Operational Simulator for Small Satellites (NOS3), which served as the foundational environment for this work.

Our research heavily relied upon components frequently used within or alongside NOS3, including NASA's core Flight System (cFS) for flight software simulation and YAMCS (Yet Another Mission Control System) as the ground software interface. The agentic AI capabilities were implemented using Python, leveraging powerful frameworks such as Ollama for serving local large language models (including models like Qwen and Gemma), LlamaIndex for agent orchestration and Retrieval-Augmented Generation (RAG), and ChromaDB for efficient vector storage. Specific embedding models like nomic-embed-text were also crucial for the RAG pipeline. Data visualization and operator interaction were facilitated by the Streamlit framework, while long-term time-series data persistence and analysis were enabled by TimescaleDB built upon PostgreSQL. The entire system was developed and deployed using Docker containerization, primarily within a Linux operating environment. The availability, maturity, and collaborative nature of these open-source tools were instrumental to the successful execution and demonstration of this project.

References

List of references

[1] Synera (2025). "AI Agents build NASA spaceship from text" (Aviation Week, Mar 27, 2025). – *Reports on NASA's Text-to-Spaceship project using LLM agents for spacecraft design; multi-agent AI cuts preliminary design time drastically* (Aviation Week article: NASA tests AI's ability to design a spaceship with Synera support).

[2] NASA Advanced Supercomputing Division (2024). *NASA-GPT: Searching NASA Technical Reports Using AI*. – *Describes an internal NASA chatbot that leverages LLM technology to retrieve information from technical report databases* (NAS 2024 Wrap-Up: 12 Significant NAS Division Science & Engineering Projects of the Year).

[3] European Space Agency (2024). "Building ChatGPT-style tools with Earth observation." – *ESA news article on developing a natural language assistant for querying Earth observation data, leveraging foundation models for accurate answers* (ESA - Building ChatGPT-style tools with Earth observation).

[4] C. Schefels *et al.* (2023). *Evaluating Large Language Models for Space Operations*. DLR/GSOC. – *Explores LLM use for mission control tasks; found local models can answer ops queries, with fine-tuning needed for accuracy* (Evaluating Large Language Models for Space Operations).

[5] ESA European Space Operations Centre – AI and Data Foundation (2024). *A²I Roadmap – Highlights ESA's development of an LLM-powered anomaly investigation assistant deployed on missions* (A²I Roadmap | OPS Portal).

[6] J. Perrius & G. Cevasco (2025). "Taking the Ground Out of Ground Systems." *Booz Allen Velocity* **V3**. – *Discusses virtual ground station concepts using networked LLMs to accelerate military satellite operations and decision-making* (Taking the Ground Out of Ground Systems).

[7] Axiom Space (2025). *Press Release: Axiom to Launch Orbital Data Centers*. – *Details planned in-orbit data centers enabling on-board AI/LLM processing for real-time satellite decision support* (Axiom Space to Launch Orbital Data Center Nodes to Support National Security, Commercial, International Customers).

[8] V. Rodriguez-Fernandez & A. Carrasco (2024). *Language Models are Spacecraft Operators* (arXiv:2404.00413). – *Demonstrates a GPT-4 based agent performing autonomous satellite maneuvers in a game environment, illustrating LLMs' potential in closed-loop control* (Language Models are Spacecraft Operators).

[9] U. Kling et al. (2024). *"Prototype of a Workflow for a Digital Twin in Small Satellite Operations." presented at the 75th Int. Astronautical Congress (IAC), session IAC-24-B4.3.2, Milan, Italy, Oct 14-18 2024.*