Leveraging Large Language Models for Telemetry-Monitoring Code Generation for Life Support Systems

H. Sandidge[1], C. Williams[2], J. Hall[3], and J. Wall[4]
Marquette University, Milwaukee, Wisconsin, 53201

K. Sandidge[5]
Best Buy, Kohler, Wisconsin, 53044

and

J. Rohrig[6]
Collins Aerospace, Winsor Locks, Connecticut, 06096

Reliable monitoring of life support systems in space exploration is challenging. The number of telemetry-generating platforms continues to increase, and consequently, so does the need for health-monitoring software development and active real-time monitoring itself. For many platforms, the translation of engineering-created flight rules into software is both costly and timely. Additionally, the number of systems, states, and conditions that need active monitoring by both crew on the ground and on the edge pose challenges such as cognitive overload and reliance on memorization of recovery procedures. This paper proposes a novel methodology that uses large language models (LLMs) to automate the generation of telemetry-monitoring code based on flight rules. This approach has the advantage of reducing the need for extensive document interpretation and programming and can be adapted for any telemetry-generating platform. Additionally, we propose a standardized abstract base class (ABC) code pattern for flight rules. This standardized code structure allows for a platform-and-vehicle-agnostic monitoring system to be developed. We demonstrate a prototype of such a self-monitoring system (SMS) through an example derived from a simplified extravehicular mobility unit (EMU) portable life support system (PLSS), laying the groundwork for future empirical research into the performance of our proposed code-generation system. The proposed system offers a promising solution to streamline the development and monitoring of life-support systems, enhancing both efficiency and reliability in mission-critical environments.

Acronyms and Nomenclature

ABC	=	Abstract Base Class
AI	=	Artificial Intelligence
API	=	Application Programming Interface
CAPCOM	=	Capsule Communicator
ECLSS	=	Environmental Control and Life Support Systems
EMU	=	Extravehicular Mobility Unit
EVA	=	Extravehicular Activity
FDIR	=	Fault Detection, Isolation, And Recovery

[1] Director, AI Development and Value Creation, hunter.sandidge@marquette.edu.
[2] Undergraduate Research Assistant, AI Development and Value Creation, charles.williams@marquette.edu.
[3] Undergraduate Research Assistant, AI Development and Value Creation, jermauri.hall@marquette.edu.
[4] Chief of Strategy, AI Development and Value Creation, joseph.wall@marquette.edu.
[5] General Manager, kc.sandidge@bestbuy.com.
[6] Advanced Technology Discipline Chief, Advanced Research and Technology, jake.rohrig@collins.com.

GMSEC = Goddard Mission Services Evolution Center
HITL = Human-In-The-Loop
ISS = International Space Station
LEO = Low Earth Orbit
LLM = Large Language Model
MCT = Mission Control Technologies
NASA = National Aeronautics And Space Administration
ORM = Object-Relational-Mapping
PFD = Product Flow Diagram
PLSS = Portable Life Support System
SME = Subject Matter Experts
SMS = Self-Monitoring System
XML = Extensible Markup Language

I. Introduction

SPACEFLIGHT operations use flight rules to help improve astronaut safety and mission success. The rules are usually in a narrative format to help guide experts during operations and extravehicular activity. As telemetry streams continue to increase in quantity and complexity with each subsequent mission, so do the rules. Thus, human observers attempting to monitor this data manually and apply these rules experience increasing job complexity and difficulty over time. This complexity increases the chance of errors and human observers missing key data. Although automated telemetry monitoring tools are widely used, translating operational intent into executable monitoring logic still requires human configuration and interpretation. Because this translation is not inherently standardized, differences in implementation can introduce friction when scaling across missions, teams, or hardware environments [1].

This manual translation introduces safety and cost implications. Also, changes to the rules, telemetry naming conventions, or operational constraints need to be reviewed by multiple members of the flight control team [1]. Proprietary nature of the rules exacerbates the problem by limiting the number of tools and external service choices. Missions are becoming further, longer, and more autonomous. This combination increases the likelihood of needs for real-time decision support. Yet, this combination also decreases the likelihood of high-quality real-time decision support. Thus, there is a growing need for a systematic, extensible approach to converting narrative flight rules into reliable, machine-readable, creatable, and executable monitoring logic.

In this paper, we propose a modular approach to transform narrative flight rules into executable telemetry-monitoring code. Our approach uses unique modules for content extraction, rule parsing, validation, rule enrichment, code generation, and multi-layer validation to enable plug-and-play frameworks that are efficient, adaptable, and consistent. We use large-language-model (LLM) architecture to leverage structured outputs and abstract base classes. We include humans in the loop at critical junctures to ensure rules are being written and applied appropriately. Further, this system is portable when using private or local models. This overall approach helps ensure adaptability, consistency, and safety. We conclude by demonstrating feasibility through a proof-of-concept that uses synthetic telemetry and flight rules, laying the groundwork for future research into the empirical performance of our proposed system.

II. Background

Telemetry, the automatic transmission and remote collection of data to a central system for monitoring and analysis, had early documentation by Mayo-Wells (1963). Wells notes that Russian engineers employed electromagnetic circuits as early as the War of 1812 to remotely detonate naval mines. The principles of telemetry and the electric telegraph soon extended into scientific monitoring, including Olland's 1874 transmission of meteorological measurements from Mont Blanc to Paris over a 350-kilometer landline, establishing telemetry as a continuous remote observation mechanism [2]. By the early twentieth century, advances in aviation and rocketry transformed telemetry into an operational necessity for real-time measurement of propulsion performance, structural loads, and system health, a role that carried directly into spaceflight operations where telemetry became the backbone of spacecraft monitoring and mission safety. Modern space systems now integrate telemetry with prognostics and health management frameworks to move beyond passive data transmission toward predictive system awareness, enabling early failure detection, life extension assessment, and sustained human performance in space environments [3].

Existing approaches for automated telemetry monitoring and flight rule execution fall into five broad categories: early rule systems, mission operations infrastructure frameworks, fault detection and diagnosis algorithms, knowledge representation and autonomy frameworks, and emerging large language model applications in space systems. Early work on formalizing and automating flight rules demonstrated that narrative operational constraints could be translated into computer-interpretable representations. Bell et al. presented a system for encoding International Space Station (ISS) environmental controls and life support systems (ECLSS) flight rules using Extensible Markup Language (XML). This allowed for real-time monitoring against live telemetry streams [4]. While effective, this approach required manually encoding structured rules and was not built with the robustness required for large rule sets or rapid rule modification.

Mission operations frameworks, such as Goddard Mission Services Evolution Center (GMSEC) and Open Mission Control Technologies (MCT), focus on messaging standards and visualizations rather than rule translation. GMSEC also provides standardized messaging interfaces to support ground systems [5], [6], [7]. Further, NASA developed Open MCT, an open-source, web-based tool to visualize and analyze complex data [8], [9]. These frameworks have the advantage of being easier to integrate into other systems, but they generally carry the risk of assuming that executable rule logic already exists. This leaves the automation or semi-autonomous execution of flight rules unresolved.

Existing research examines fault detection, isolation, and recovery using model-based, data-driven, and hybrid techniques. Tipaldi and Bruenjes wrote a comprehensive overview focusing on Fault Detection, Isolation, and Recovery (FDIR). This technology allows a satellite to realize something is wrong, figure out what broke, and fix it or switch to a backup without human help. Further, modern satellite monitoring uses math and data trends to identify anomalies without requiring custom models for every component [10], [11]. While these methods are efficient, they do not solve the problem of manually translating human instructions (flight rules) into computer code.

Autonomy research explored structured representations of system behavior and decisions. Early NASA work turned graphical representations of diagnostics into automated rule generation [12]. Newer systems use "ontologies," essentially digital encyclopedias, that define how satellite parts and mission goals relate to one another. This allows a computer to "reason" through complex situations and ensure that every action taken by the satellite is logically connected to the original mission plan [13]. This approach requires a lot of work upfront, but it still often fails to automate translating rules into executable code.

While these knowledge-driven autonomy frameworks provide a principled way to model system behavior and decision logic for satellites, their practical deployment in mission operations remains challenging for human space exploration due to the complexity of formalizing and maintaining executable rule structures, especially when considering the non-mechanical human components. As a result, many operational environments continue to rely on human-validated constraints, most notably in the form of flight rules, to govern real-time decision-making [1], [14].

Flight rules are pre-approved operational constraints that govern how a space mission may be conducted under both nominal and off-nominal conditions. Flight rules encode agreed-upon responses to a wide range of anticipated scenarios. These rules are developed collaboratively by engineering disciplines, mission operations, safety authorities, and program management prior to flight [1]. Flight rules define allowable actions, prohibited actions, decision thresholds, and escalation criteria based on system state, environmental conditions, and mission phase. Their primary purpose is to reduce ambiguity during time-critical operations by ensuring that decisions affecting crew safety and vehicle integrity are consistent with pre-flight engineering analysis and programmatic risk acceptance.

During mission execution, the Flight Control Team uses flight rules as a real-time decision framework when interpreting telemetry and responding to anomalies. When off-nominal behavior is detected, either through automated monitoring systems or human observation, flight controllers consult applicable flight rules to determine whether operations may continue, must be modified, or require immediate termination or contingency actions. These rules are typically distributed across large, structured documents and mission-specific annexes, and are interpreted collaboratively by multiple discipline specialists before guidance is communicated to the crew.

Astronauts generally do not reference flight rules directly; instead, they receive clear, directive instructions derived from Mission Control's interpretation of flight rules. This human-centered process ensures safety and accountability, but also introduces latency due to the volume, complexity, and cross-referencing of the flight rule sets. This operational model is largely unchanged since the Apollo era. The model relies on centralized decision-making, with mission controllers providing real-time instructions and support to astronauts during extravehicular activity (EVA) operations [15].

The flight-rule–centric operational model has proven highly effective for low-Earth orbit (LEO) missions such as those on the ISS. However, it does not scale to deep-space missions where communication latency fundamentally alters the dynamics of anomaly response [16]. For missions to Mars, one-way communication delays ranging from several minutes to over twenty minutes eliminate the possibility of real-time ground interpretation and guidance [15].

Under these conditions, the current reliance on large, discipline-specialized ground teams to interpret telemetry, consult flight rules, and relay instructions becomes impractical. Time-critical decisions affecting crew safety and system integrity must instead be made onboard, as noted by Chullen in 2017, often with incomplete situational context and without immediate ground validation [17]. As a result, in the traditional model telemetry anomalies are detected by automated systems but resolved through human interpretation on Earth. This process introduces unacceptable delays and operational risk for deep-space missions, necessitating new approaches that enable faster, onboard interpretation of pre-approved constraints.

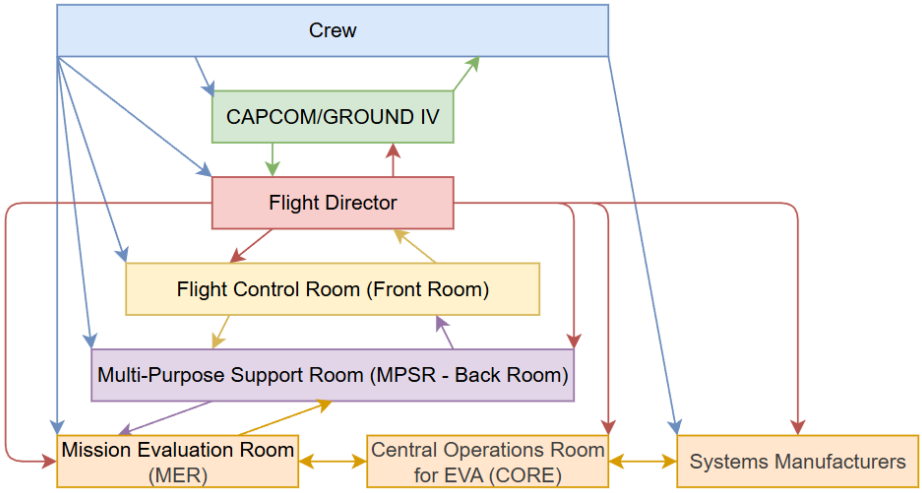

Figure 1. Diagram of EVA Communications Flow. The diagram presents a flow map of the audio communication paths supporting an EVA.

Current NASA mission operations rely on a distributed, multi-layered flight control structure, as seen in Figure 1, in which each front-room flight controller is supported by a team of back-room specialists responsible for monitoring highly detailed subsystem data [17]. These teams communicate through dedicated voice loop channels, coordinating anomaly detection, diagnosis, and response across multiple roles with differing scopes of responsibility [17]. Back-room specialists provide deep subsystem expertise, such as mechanical systems, crew equipment, or in-flight maintenance, while front-room controllers synthesize this information and escalate recommendations to the flight director for approval before instructions are relayed to the crew. Although this approach has proven effective for complex missions mentioned previously, it requires significant real-time coordination among numerous personnel, introducing procedural and communication latency that can become a limiting factor when rapid decision-making is required or when ground support is constrained.

When an off-nominal condition is detected, identifying and applying the appropriate flight rule is a non-trivial process that typically unfolds over several minutes. Flight controllers must first validate the anomaly, establish operational context, and determine which subsets of flight rules are applicable based on mission phase and system state. Relevant rules are then located within large, cross-referenced documents and interpreted collaboratively across multiple disciplines to ensure that constraints, exceptions, and precedence relationships are correctly understood. This deliberate, human-centered process, while essential for safety, introduces response latency that is driven by cognitive workload and coordination rather than by telemetry availability or communication speed.

To illustrate the current operational workflow, consider a scenario in which the telemetry alarm system indicates a decrease in spacesuit oxygen levels during an EVA. As shown in Figure 1 the front room flight controller responsible for suit systems first verifies whether the alert reflects sensor noise, a false positive, or a valid off-nominal condition. This initial assessment typically involves examining telemetry trends, correlated parameters, and mission context to determine whether the observed behavior is consistent with nominal operations, a process that can require several minutes. If the condition is confirmed to be outside expected limits, the controller then consults applicable flight rules, reviewing triggering conditions, constraints, and exceptions, and cross-checking referenced rules to ensure no higher-priority constraints apply. Given the size and cross-referenced nature of flight rule documentation, this step can require additional minutes to complete. Once a candidate response is identified, back-room specialists supporting the front-room controller are consulted to provide deeper subsystem expertise and confirm the proposed interpretation. The resulting recommendation is then relayed to the flight director for approval before being communicated to the crew

through the capsule communicator (CAPCOM) [17]. While each step is essential for safety, the cumulative process, from anomaly detection to crew guidance, can reasonably span tens of minutes during time-critical EVA operations.

Flight rules are standardized in their intent and governance across NASA human spaceflight programs, while remaining mission- and vehicle-specific in content. They are developed prior to flight through a rigorous, multi-disciplinary approval process and encode pre-authorized operational constraints for both nominal and off-nominal conditions. Although individual rule sets vary based on mission phase, system design, and environment, formalized language enables flight controllers to interpret them reliably during time-critical operations. However, the structure of each flight rule is independent of each other, thus making it potentially more difficult to interpret. This results in large, complex rule sets that must be manually searched and interpreted during anomalies with varying degrees of standardized structure in writing format.

III. Limitations

Despite the central role of telemetry in modern spaceflight operations, current systems face persistent challenges across data quality and volume, human workload and error, and the cost and scalability of monitoring infrastructure. Spaceflight telemetry systems generate vast volumes of heterogeneous, unstandardized data transmitted across long distances, subject to signal degradation, interference, and communication latency [18]. Interplanetary missions, such as those to Mars, experience delays of up to twenty minutes, limiting real-time intervention and amplifying the importance of accurate autonomous monitoring, especially for human spaceflight exploration. As described in the Leto Spaceflight Intelligence System developed by Collins Aerospace, “Each system, subsystem, and sensor has unique functional bounds and telemetry delivery systems which create immense volumes of unstandardized and unstructured data across many disparate sources.” [19]. This fragmentation complicates data aggregation, contextual interpretation, and timely anomaly detection.

Telemetry integrity further degrades under the harsh space environment, where radiation exposure, mechanical stress, and extreme temperature fluctuations impact sensor reliability. Corrupted or drifting sensors can generate misleading measurements, triggering false alarms or masking true system degradation [20]. Such false positives divert mission control resources from investigating genuine anomalies, delaying responses.

Interpreting complex telemetry streams remains heavily dependent on highly trained human analysts. Skilled flight controllers and subsystem specialists are required to identify subtle patterns, diagnose anomalies, and contextualize system behavior across interconnected subsystems. However, as telemetry volumes increase, available human capital has become a limiting factor [18]. Leto highlights that shortages in analytical staffing “result in missed anomalies, delayed diagnostics, and suboptimal use of specialist time” [19]. Communication delays inherent to deep-space missions further constrain effective human intervention, increasing operational risk in time-critical scenarios. Additionally, modern mission control environments rely on personnel to compensate for limitations in automated monitoring, often through continuous manual review, trend analysis, and cross-system reasoning. While this approach has historically ensured safety and reliability, it introduces vulnerability to cognitive overload, fatigue, and human error, particularly as system complexity continues to grow.

Current telemetry monitoring solutions predominantly rely on manually authored algorithms and rule-based logic developed collaboratively by subject matter experts (SMEs) that are then translated into code by software engineers. This paradigm provides high transparency, validation rigor, and operator trust, and has been foundational to safe human spaceflight operations, including ISS mission support and EVA monitoring [19]. Each rule is explicitly designed to reflect known system behavior and operational constraints.

However, manually encoding flight rules is inherently time-intensive and costly. Creating a single monitoring algorithm requires SMEs to interpret system documentation, identify relevant telemetry signatures, and, with software developers, iteratively translate those signatures into executable logic. Prior research in NASA mission operations and autonomy systems consistently identifies this process as labor-intensive and poorly scalable as telemetry volume and system complexity increase [21], [22]. Even conservative estimates of aerospace software development suggest that individual monitoring rules may require tens of engineering hours for design, verification, and deployment.

Consequently, operational systems typically implement only a limited subset of possible monitoring algorithms, prioritizing high-severity failure modes that pose an immediate safety risk. Lower-severity but operationally valuable behaviors, such as gradual consumable degradation, maintenance forecasting, and performance optimization, are often left unmonitored or assessed manually. This imbalance reflects not a lack of utility, but the prohibitive effort required to author and maintain comprehensive rule sets. Similar constraints have been documented in ISS operations, where ground teams routinely perform manual review of the telemetry and trend analysis to identify anomalous behavior and recommend corrective actions [19], [21].

The reliance on manual monitoring directly drives sustained operational cost. These include both direct labor associated with continuous telemetry review and indirect costs related to delayed detection, reactive interventions, and limited analytical coverage. NASA Office of Inspector General reports indicate that ground operations and mission support constitute a substantial portion of the ISS's annual operating cost, which has remained on the order of several billion dollars per year [23]. As future missions expand in duration, autonomy, and scope, these labor-driven costs are expected to grow disproportionately relative to available human expertise [19].

Despite widespread recognition of these challenges, scaling traditional monitoring approaches remains difficult. Effective algorithm development requires a rare intersection of deep domain knowledge in systems and proficiency in software engineering, data science, or artificial intelligence methods, skill sets that seldom coexist within a single team. Bridging this gap introduces coordination overhead, slows deployment cycles, and further limits the rate at which new monitoring capabilities can be introduced [22].

Yet, the advent of LLMs holds promise, especially as they begin to impact space systems engineering. Agentic AI using LLMs showed capability in simulations of autonomous spacecraft control [24]. Developers are training AI models on private satellite manuals and technical documents to make them smarter at handling space communications. This approach works better than having the AI search through a database for retrieval. However, pulling text from rich, deep datasets remains a problem. Further, keeping data private and ensuring accuracy remains challenging [25]. Taken together, this collective information suggests the ability to automate is increasing, but a standardized, validated process for translating narrative flight rules into executable telemetry monitors does not exist. This gap motivates the modular, validated rule translation framework proposed in this paper.

IV. Proposed Solution

The methodology proposed in this paper addresses these limitations by enabling SMEs to express expected system behavior, anomalous conditions, and recommended corrective actions directly in natural language. By leveraging large language models to translate this structured narrative knowledge into executable telemetry monitoring code, the approach reduces the manual burden traditionally associated with algorithm development. Importantly, this process preserves SME intent, interpretability, and human oversight while accelerating rule creation and iteration.

By reducing the cost and effort required to develop monitoring algorithms, this approach enables the creation of broader, more comprehensive suites of telemetry monitors. Increased coverage allows systems to detect a wider range of off-nominal behaviors earlier, supporting proactive intervention rather than reactive response. Prior work on intelligent ECLSS monitoring has demonstrated that expanding automated analytics can reduce labor requirements, improve situational awareness, and allow engineering staff to focus on higher-value decision-making rather than routine data preparation and inspection [19].

Ultimately, the ability to rapidly generate, validate, and deploy telemetry monitoring algorithms can improve both safety and operational efficiency. While this work does not seek to replace traditional rigor or human oversight, it provides a scalable mechanism for capturing expert knowledge and translating it into actionable monitoring logic. In doing so, it establishes a foundation for more resilient, data-driven, and autonomous mission operations as spaceflight systems and mission architectures continue to evolve.

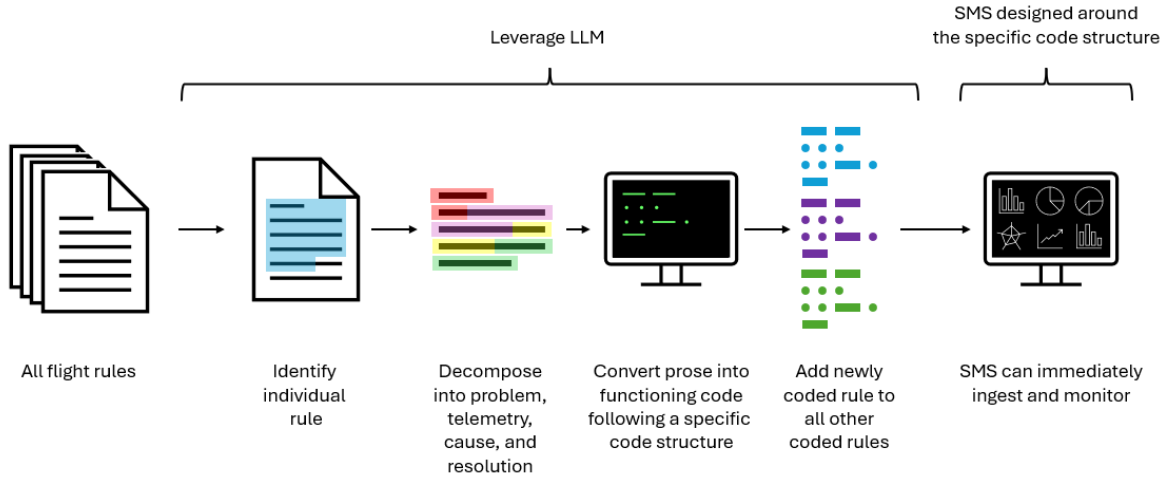

Figure 2. Infographic showing the goal of the proposed method. The infographic shows the generalized steps our process tries to automate and how the output, if structured properly, could be ingested into a platform-agnostic monitoring system to prevent a custom system from having to be developed for every platform/mission.

Before discussing the method itself, it is important to outline the research goal of our proposed system. At present, software engineers read through volumes of flight rules written by engineers and convert that narrative into functioning code [26]. Within that process, software engineers have to identify the core problem described by each flight rule, what the telemetry would look like when such problem manifests, the causes for the problem, as well as the resolutions to the problem [27]. They then code each flight rule manually into what are often fully unique monitoring systems [28]. Our goal is to automate that process through the use of an LLM, a form of AI that excels at taking instructions in natural language converting it into code, while concurrently presenting how using a specific code output structure could allow for a fully reusable, platform agnostic self-monitoring system (SMS) such that any platform (space station, satellite, launch vehicle, spacesuit, rover, etc.) could be monitored without having to develop unique systems for each. We provide an infographic in Figure 2 above to illustrate our goal.

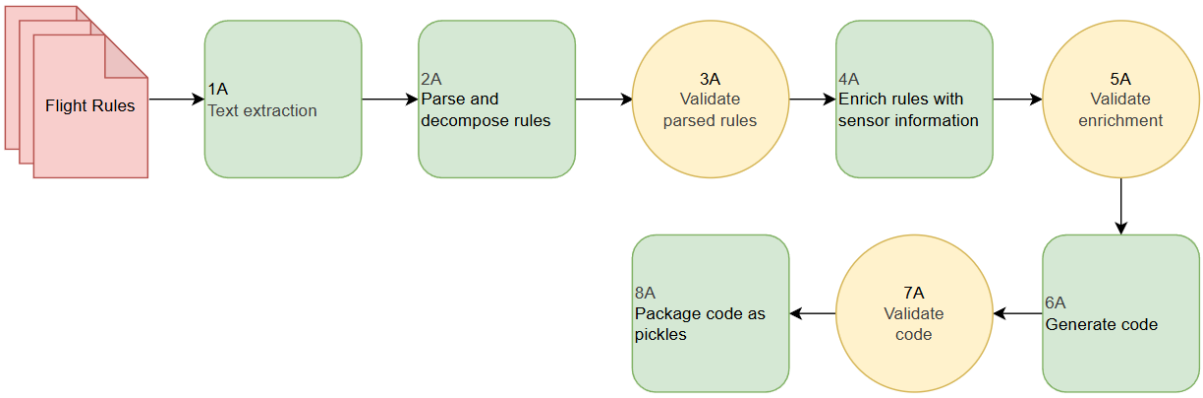

Figure 3. Diagram of Proposed Solution. The diagram presents a flow map of our proposed flight-rules-to-code process.

Our proposed process (Figure 3) can be broken out into the following steps: content extraction, rule parsing, parsing validation, rule enrichment, enrichment validation, code generation, code validation, and code packaging. Our entire architecture relies upon the concept of strategy patterns, providing future users and us an easy-to-use framework that is effectively plug-and-play code, such that alterations and custom versions of each of our sections can be drafted and injected into our system without having to rewrite or change anything beyond the edited section itself.

First, as seen in Figure 3-1A, the content of the rules must be extracted and made computer-readable. This layer may be challenging, depending on the era and the form of the flight rules. Modern documentation is typically digitized (e.g., PDF or Word document form), making it ready or near-ready to be read. However, even in modern documentation, images and tables may be embedded in a way that requires advanced information extraction techniques. Older documentation may only exist as hard copies or scanned images of hard copies. Because of this, we build an abstract base class (ABC) that allows a user to build the proper extraction methodology (direct text extraction, optical character recognition, etc.) for their respective flight rules that will work within our larger system. For our prototype, we use the Markdown Python library, which extracts information from a wide variety of document types and converts the extracted text into markdown (a text formatting language that LLMs work well with).

Once the text is extracted from the documents, we then propose using an LLM in combination with structured outputs (pre-defined code structures provided to an LLM to coerce it into responding in a particular shape and form) to parse individual rules (Figure 3-2A) from the aggregate body of text as well as decompose each discovered flight rule into the core components of rule title, problem, cause, resolution, and observable telemetry during the problem (we recognize that these components may not be standard to every type of flight rule). To do this, the extracted text is sent to an LLM via an application programming interface (API) along with the desired structured output for the response and a prompt instructing the LLM to parse the rules and elements. This prompt can be customized to provide the LLM with additional information idiosyncratic to the underlying rules and their structure. Again, we provide an ABC that performs parsing, allowing users to create their own prompts, and additional preprocessing steps (e.g., Regex) that work within our system. Additionally, we include an ABC that allows a user to provide any LLM of their choosing (so long as it is capable of structured outputs). This grants the user the ability to use local or private models, which is an important consideration given that most information related to flight rules are often proprietary or under strict export control. Once the rules are parsed and decomposed by the LLM into the various elements required by our system, we then inject the information into a SQLite database.

Following the extraction step, we implement an automated rule extraction validation layer (Figure 3-3A). The system employs an LLM to audit the output of the extraction step, comparing each parsed rule against its corresponding section in the original document. For our prototype, the validator identifies rule markers (e.g., "FR-###") in the source document and verifies that all rules were extracted. For each extracted rule, it performs completeness checks, identifying missing measurements, numeric thresholds, and technical acronyms through regex pattern matching and accuracy checks that detect hallucinated content, modified numbers, or corrupted special characters (such as subscripts or degree symbols). The validator prompts an LLM to identify any content from the original document that was omitted and any information in the extraction that does not appear in the source. This dual approach, combining programmatic pattern matching with LLM-based semantic review, provides layered assurance that the extraction is both complete and faithful to the original document.

Once the parsing validation is completed, we propose enriching the rules (Figure 3-4A) by directly embedding the sensor names into the text. In many instances, the flight rules are written in plain English (e.g., spacesuit oxygen pressure), while the telemetry being streamed from the hardware lacks such terms. Commonly, the column names of the telemetry are either entirely decoupled and encoded (e.g.: PUI000011) or are partially decoupled (e.g., suit_o2_pressure). Because of this, there is often a gap between the structured data of the telemetry and the flight rules that must be clarified before any form of code or monitoring can happen. To resolve this issue, we once again propose leveraging an LLM. We individually pass each parsed rule and the telemetry metadata (primarily the sensor name and sensor description) to an LLM and instruct it to embed the sensor name after each discovered instance of its underlying (e.g., "…spacesuit oxygen pressure [PUI000011]…"). We provide an ABC to help map the observed telemetry section of the flight rules to the actual telemetry sensor names being streamed. This layer allows the user to perform any such text enrichment they want.

Once telemetry descriptions are annotated with sensor identifiers, we validate the sensor mappings to ensure accuracy (Figure 3-5A). The mapper validator checks for four categories of errors: hallucinated sensor IDs (identifiers that do not exist in the sensor metadata catalog), text modification (any alteration to the original telemetry text beyond adding annotations), unit incompatibility (mismatches between the measurement units mentioned in the text and the sensor's declared unit in the catalog), and unmapped telemetry terms (technical keywords like "pressure" or "ppCO2" that lack a sensor ID annotation). When validation issues are detected, the system leverages an LLM to automatically correct the mappings. This corrector LLM receives the original telemetry, the erroneous annotation, the specific issues detected, and the full sensor catalog, then produces a corrected version with proper sensor ID annotations. The system re-validates the correction to confirm all issues have been resolved. This auto-correction loop transforms what would otherwise be a binary pass/fail system into a self-healing pipeline that improves mapping quality without human intervention while still flagging any issues that resist correction.

Once the rules have been ingested, parsed, broken into the four core components (problem title, cause, resolution, observable telemetry), and enriched with sensors (with intermittent validation), we begin the most important step in the process: code generation (Figure 3-6A). In this step, we propose leveraging an LLM to convert the narrative rules into Python code. First, we define a rigid ABC with strict expected properties, methods, inputs, and outputs to guarantee all coded flight rule functions are standardized and consistent. This is the quintessential step that ensures that all rules can be ingested into a SMS and successfully run. By forcing the LLM to generate code in a standardized structure, a platform-agnostic SMS can be leveraged so long as the SMS knows to seek for and use the predefined methods and properties. For each flight rule, we force the code to have properties for the title of the problem, the cause, the observable telemetry description, the resolution, and the number of time periods required to observe the issue. Additionally, we force the flight rule code to have three methods: assess, to_dict, and pickle. This allows any platform's flight rules to be passed into our system, code to be generated, and the resultant code to be immediately ingestible into a generalized monitoring system. This process opens an opportunity for a universal monitoring system could be created, saving time and money across missions, while also standardizing telemetry health monitoring.

The "assess" method requires the input of all columns of the telemetry and rows consisting of the most recent data point, going back to the maximum required time step to observe the problem. This method also requires the code to output a dictionary (defined by the required "to_dict" method) with a standardized set of keys. Because these keys are standardized, any SMS can easily identify if a problem is occurring and, if so, can display it in an intelligent manner to the user. The last required method, "pickle," converts the code into a pickle object, which can easily be discovered and ingested by the SMS, such that it can, without knowing anything about the flight rules, find, run, and assess each rule.

We follow the code generation with multi-layer validation (Figure 2-7A). The code generator validation uses a three-phase approach to ensure generated Python code is safe, syntactically correct, and semantically aligned with the flight rule. Phase 1 (library safety) checks to ensure that the written code only imports code from an approved library list (e.g., pyproject.toml, requirements.txt, etc.). Importing an unapproved library results in a hard block, preventing

the execution of potentially unsafe code. Phase 2 (Python validity) uses Python's “ast” module to parse the generated code without execution, verifying that it is syntactically valid, contains a class inheriting from the expected ABC, includes all required class attributes, and defines an “assess”, “to_dict”, and “pickle” methods with the correct signature. Again, failures here result in a hard block. Phase 3 (semantic validation) employs an LLM to verify that the code's logic matches the rule's intent: it checks whether sensor IDs from the telemetry appear in the code's data access patterns, whether numeric thresholds are preserved exactly, and whether comparison operators correctly implement the described conditions (e.g., "above 6.0 mmHg" translates to > 6.0, not < 6.0). Unlike the first two phases, semantic validation issues are flagged as warnings rather than hard blocks, allowing human review without halting the pipeline. Finally, after all code-and-LLM-driven validation is completed, we use a layer of human-in-the-loop (HITL) to validate the outputs of the validation. This final layer helps ensure astronaut safety and system explainability.

After validation is completed and whatever issues the HITL may have discovered are updated and resolved, a script that converts each flight rules class object into pickled code can be generated (Figure 3-8A). This script instantiates and saves the code into pickle files and deposits the pickles into a specific folder that the SMS has been coded to search through. This allows the SMS to identify how many rules exist and the maximum time window required across all rules (e.g., the longest-manifesting rule requires 30 minutes of data in order to be discovered).

In addition to only monitoring explicitly described items in the flight rules, a “breach detection” script is also included. Many sensors have specific and known ranges of nominal values (minimums and maximums) that may not have a specific rule allocated to them in the flight rules, yet are still monitored by Mission Control. As a result, we add an additional layer of automatic monitoring that assesses if each sensor is within its expected nominal range, which follows the same standardized output as the flight rules, such that the SMS can agnostically run the breach detection code similarly to the flight rules code. As a part of our system, we include a sensor metadata table that has a row for each sensor, its functional description, its units of measurement, its expected minimum, its expected maximum, etc. The breach detection script leverages this backend table when monitoring, such that no additional code needs to be created or validated, so long as the table is properly filled out.

V. Proof of Concept

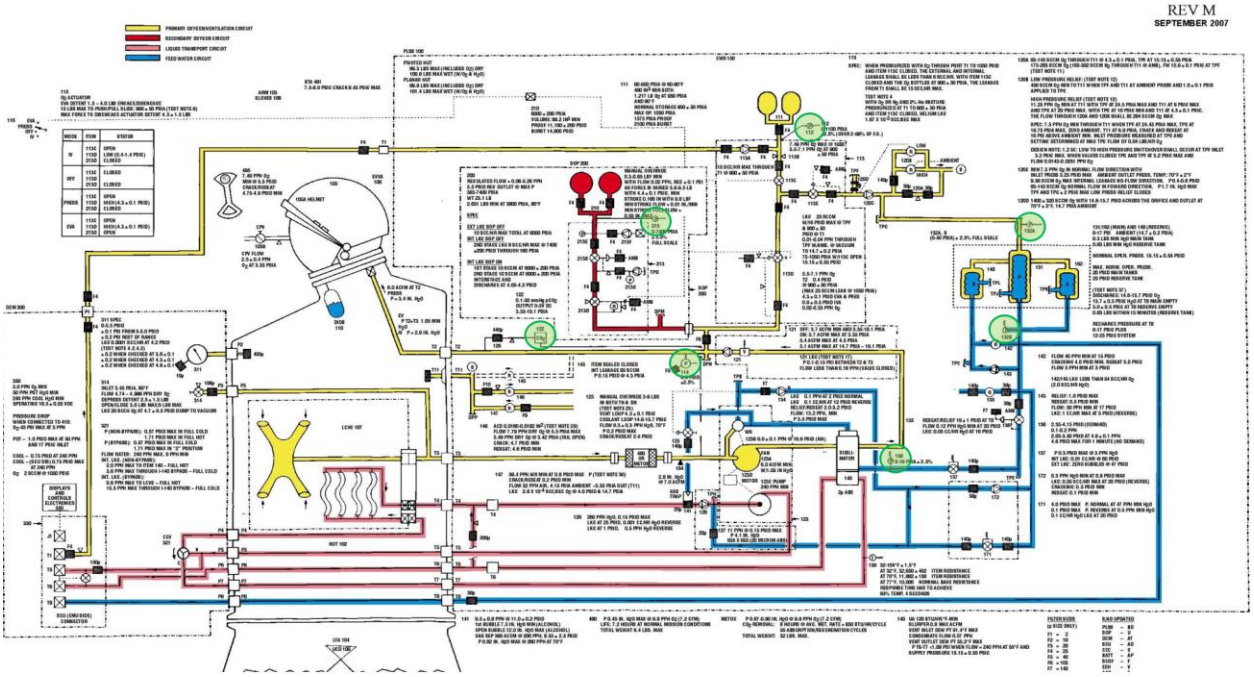

Figure 4. 2007 Rev M EMU PLSS PFD. The diagram presents a real PLSS PFD with green circles highlighting the simplified sensor-suite (not fully inclusive) we used to test our system.

To test the functional validity of our proposed system and provide a basis for future empirical evaluation, we begin by creating a small mock example similar to the existing caution and warning system on the extravehicular mobility unit (EMU) and the proposed informatics system for the advanced extravehicular mobility unit [29]. To begin, we

leverage the 2007 Rev M EMU PLSS PFD (Figure 4) as our baseline and down-select the total number of sensors [30]. From Figure 4, we select seven observed sensors, six other sensors known to be monitored not in Figure 4, and one calculated field. We present our simplified sensor suite in Table 1 below.

Table 1. Sensors selected for our mock test example.

Sensor Description	Sensor Name	Rev M Item #	System
Timestamp	EVA_Timestamp		Time
H2O Gas Pressure	PUI000001	132A	PLSS
H2O Water Pressure	PUI000002	132B	PLSS
Feedwater Pressure	PUI000003	138	PLSS
Primary O2 Pressure	PUI000004	112	PLSS
Ventilation Loop Pressure	PUI000005	114	PLSS
Sublimator Temperature	PUI000006		PLSS
IR CO2 Transducer	PUI000007	122	PLSS
Pressure Compensated CO2 Level	PUI000008		ECWS
Secondary O2 Pressure	PUI000009	215	SOP
Battery Voltage	PUI000010		DCM
Battery Current	PUI000011		DCM
Fan Pump Separator Motor Current	PUI000012		PLSS
Fan Pump Separator Motor Speed	PUI000013		PLSS

With the sensors defined, we then defined the nominal ranges and expected probability distributions for each sensor. Once this was completed, we generated synthetic data imitating a nominal spacewalk. The sensor metadata and synthetic data were then loaded into a SQLite database, which served as the backend for our system.

With the synthetic data created, we then created eight imitation flight rules based on our chosen sensor suite such as elevated CO_2 levels due to inadequate ventilation, primary oxygen depletion rate exceeds planned consumption, battery undervoltage during EVA, etc. Each rule was written in narrative form, similar in form, though less complex, to what we would expect to find in real flight rules, containing a title for the problem, the problem description, the expected cause of the problem, the telemetry that would be observed if the problem manifested, and the proposed solution to the problem. We include Figure 5 below to show an example of a synthetic flight rule:

FR 1: Elevated CO_2 Levels Due to Inadequate Ventilation

During an EVA, carbon dioxide (CO_2) may accumulate within the suit ventilation loop if ventilation effectiveness is reduced or if crew metabolic CO_2 production exceeds removal capability. This condition may result from degraded Fan Pump Separator (FPS) performance, reduced ventilation flow, increased crew workload, or diminished effectiveness of the CO_2 removal system. Elevated CO_2 constitutes a direct physiological hazard and may lead to increased respiratory effort, headache, cognitive degradation, and reduced task performance. Continued elevation without mitigation may compromise crew safety and require termination of the EVA.

Under nominal EVA conditions, pressure-compensated CO_2 is expected to remain within 1 to 3 mmHg. Values between 4 and 7.6 mmHg indicate degrading margin and require monitoring and workload management. A sustained pressure-compensated CO_2 value exceeding 7 mmHg for over 3 minutes constitutes the critical condition.

Additionally, ventilation system mechanical performance is assessed using an FPS fan speed measured by the FPS RPM sensor (Item 123A). During nominal EVA operation, FPS fan speed is expected to remain near its design operating point of approximately 19,300 RPM. A sustained fan speed approximately 13,000 RPM for over 1 minute may indicate degraded ventilation capability, will result with rising CO_2 or pressure-compensated CO_2 trends, and supports a diagnosis of reduced ventilation effectiveness.

Upon detection of rising pressure-compensated CO_2, Mission Control shall immediately assess the rate of increase and current EVA workload. The crewmember shall be directed to immediately reduce physical workload, pause translation or task execution, and report any subjective symptoms including increased breathing effort, headache, or dizziness.

If pressure-compensated CO_2 stabilizes or decreases below 4 mmHg following workload reduction, the EVA may continue at reduced intensity under heightened monitoring. If pressure-compensated CO_2 continues to rise or reaches the ECWS limit of 7.6 mmHg, Mission Control shall direct termination of the EVA. The crewmember shall return to the airlock via the most direct safe path consistent with crew safety. No in-EVA corrective action exists to restore CO_2 removal system performance. Mitigation is limited to workload reduction or EVA termination.

Figure 5. Example of a synthetic flight rule. The example above shows a synthetic flight rule related to elevated CO2 levels due to inadequate ventilation.

After the flight rules were generated, we returned to our nominal synthetic EVA data and created an additional synthetic set where we intentionally injected key issues (both nominal-bound breaches and specific flight rules). Finally, we loaded all of this into our repository and began the aforementioned code-generation process. At each of our three validation junctions (parsing of rules, enrichment of rules, and code generation), we use a HITL to validate our system's performance, similar to our expectations of how this system would be used in a real-world setting.

Table 2 below continues with the example shown in Figure 5, showing how the LLM decomposed and enriched Flight Rule 1. After this step, we confirmed that the system successfully converted this prose into code in our desired structure.

Table 2. Example of LLM identification, decomposition, and enrichment.

Title	Elevated CO2 Levels Due to Inadequate Ventilation
Problem	During an EVA, carbon dioxide (CO2) may accumulate within the suit ventilation loop if ventilation effectiveness is reduced. This condition may result from degraded Fan Pump Separator (FPS) performance, reducing ventilation flow. Elevated CO2 constitutes a direct physiological hazard and may lead to increased respiratory effort, headache, cognitive degradation, and reduced task performance. Continued elevation without mitigation may compromise crew safety and require termination of the EVA.
Cause	Degraded Fan Pump Separator (FPS) performance, reducing ventilation flow.
Telemetry	Under nominal EVA conditions, pressure-compensated CO2 is expected to remain within 1 to 3 mmHg. Values between 4 and 7.6 mmHg indicate degrading margin and require monitoring and workload management. A sustained pressure-compensated CO2 (PUI000008) value exceeding 7 mmHg for over 3 minutes constitutes the critical condition. Additionally, ventilation system mechanical performance is assessed using Fan Pump Separator (FPS) fan speed (PUI000013), measured by the FPS RPM sensor. During nominal EVA operation, FPS fan speed (PUI000013) is expected to remain near its design operating point of approximately 19,300 RPM. A sustained reduction in fan speed (PUI000013) below 17,000 RPM over 3 minutes may indicate degraded ventilation capability and, will result with rising pressure-compensated CO2 trends, supporting a diagnosis of reduced ventilation effectiveness due to degraded FPS performance.
Resolution	Upon detection of rising pressure-compensated CO, Mission Control shall immediately assess the rate of increase and current EVA workload. The crewmember shall be directed to immediately reduce physical workload, pause translation or task execution, and report any subjective symptoms including increased breathing effort, headache, or dizziness. If pressure-compensated CO2 stabilizes or decreases below 4 mmHg following workload reduction, the EVA may continue at reduced intensity under heightened monitoring. If pressure-compensated CO2 continues to rise or reaches the ECWS limit of 7.6 mmHg, Mission Control shall direct termination of the EVA. The crewmember shall return to the airlock via the most direct safe path consistent with crew safety. No in-EVA corrective action exists to restore CO2 removal system performance. Mitigation is limited to workload reduction or EVA termination.

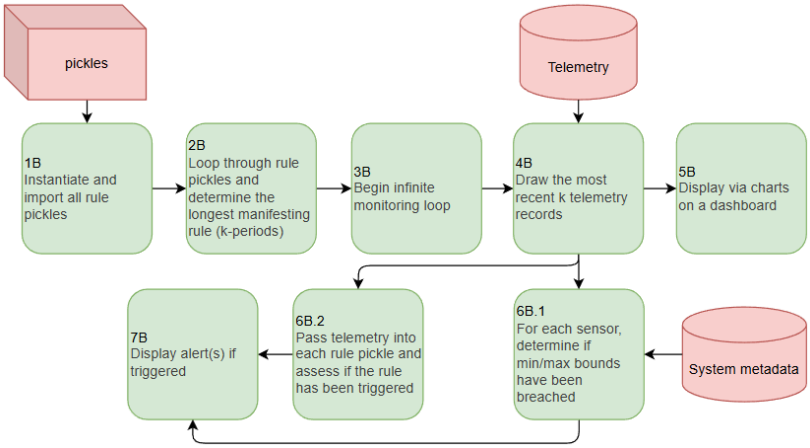

Figure 6. SMS Flow Map. The diagram presents the steps and order of our mock SMS, showing how the code generation from Figure 1 can be leveraged for real-time telemetry monitoring.

Once the flight rules code and pickles were generated, we then built a mock SMS to demonstrate a real-world analogue for how Mission Control might use our proposed solution and to demonstrate that a monitoring system could be developed that has no programming specific to an underly platform or vehicle. We demonstrate the steps the SMS

follows in Figure 6 above. Upon instantiation (Figure 6-1B), the SMS ingests all pickled code rules. After ingesting all code pickles, it loops through each to determine the maximum period of time any rule takes to manifest (Figure 6-2B). Then, in an endless loop (Figure 6-3B), the SMS pulls k data points (Figure 6-4B), where k is the maximum time period required across all rules, from a table in our SQLite database to which we stream our synthetic EVA data (emulating real telemetry). It then renders the new telemetry onto a dashboard for visualization (Figure 6-5B). Additionally, two forms of monitoring are then performed on the drawn telemetry. First, the SMS draws information from stored metadata about each sensor to check whether the latest record of telemetry breached any lower and upper bounds (Figure 6-6B.1). If any sensor breaches its known safety limits, an alert is then displayed on the SMS (Figure 6-7B). Concurrently, the SMS loads each pickle, passes in the telemetry, runs the pickle's "assess" method, and interprets the standardized result to determine if action needs to be taken (Figure 6-6B.2). If a rule has been triggered, then an alert is then displayed on the SMS (Figure 6-7B). If a problem arises during the EVA, the application generates a warning and displays the title, problem, expected causes, manifested telemetry, and proposed resolution such that the SME is immediately alerted and can then take action.

Figure 7. Screenshot of SMS. The screenshot above shows our proof-of-concept SMS identifying an alert discovered by auto-generated code.

To verify our system worked as intended, we passed our nominal and off-nominal synthetic data sets through our Streamlit-based SMS (example shown in Figure 7) to ensure the system caught and flagged all injected issues, while also not flagging false positive events.

VI. Conclusion

This paper presented a modular framework for transforming narrative flight rules into machine-executable telemetry monitoring code with humans in the loop. By breaking the process into separate structures with validation checkpoints, this architecture helps overcome problems associated with manual rule translation, including inconsistency, errors, and maintenance. Using abstract base classes and strategy patterns increases customization while minimizing the need for massive code rewrites. Structured LLM outputs and human-in-the-loop further improve accuracy and safety in

operations. Additionally, we show how this system could be developed in parallel with a platform-agnostic SMS such that a universal monitoring system could be created.

We use synthetic data and flight rules to demonstrate our proof-of-concept. The proposed system shows high potential for ingesting rules, generating standardized code, and supporting real-time monitoring. While the evaluation is limited in scope and does not claim operational readiness, it does suggest that further work on this concept appears promising. LLM-assisted rule translation seems to increase the quantity, quality, and speed of telemetry monitoring workflows without sacrificing transparency or violating controls.

Thus, LLMs hold great potential as interpreters between human-narrative-based rules and large live datasets to support decisions. As space missions become more autonomous and complex, architectures that preserve human intent and enable reliable automation will matter more. The framework presented here provides a foundation for future research. Industry collaboration focused on reducing costs, increasing safety, and improving consistency will find this helpful as they continue to develop scalable, automated telemetry monitoring.

VII. Caveats

While our work establishes a strong foundation for an automated telemetry monitoring code-generation system, it is essential to acknowledge that this is an early-stage proof-of-concept. The system remains a work in progress, and there are gaps in our ability to ensure it reflects real-world conditions due to a lack of flight rules and telemetry. We acknowledge that we took many liberties and made many assumptions in designing the underlying telemetry and flight rules. To name a few of many: we made assumptions about the structure and contents of the flight rules; our flight rules were in a computer-readable form (not a scanned picture); our flight rules were strictly text and did not contain images, schematics, tables, or other non-text elements; the observable telemetry for each flight rule was made to be relatively simple; our synthetic data set did not experience data drop out (though if our system was embedded on the edge at the point of telemetry generation, this may not be an issue); the synthetic data was generated ungrounded in real bounds and probability distributions, etc. Additionally, though the focus of this paper was on explaining a theoretical framework rather than performance optimization, our sample size of flight rules for evaluation was extremely small, preventing us from confidently claiming that our system would work under real-world conditions.

While the SMS was not the primary focus of this paper, we recognize that additional refinement is needed to optimize our self-code generation system to enable root cause analysis and prevent potential cognitive overload. For example, it is possible for multiple sensors to breach their nominal bounds and several correlated flight rules issues to occur simultaneously. In our current presentation, this would lead to multiple alerts all at once, potentially causing information-overload. Additionally, though there may be a single underlying cause for all alerts happening, it would be hard to discern what that would be using our system in its current form.

VIII. Future Direction

Our primary future objective is to find an industry partner with real flight rules and telemetry so that we can perform empirical tests on our system with real-world data. If we cannot find an industry partner, we will continue with our PLSS example, scaling the data and flight rules to get as close to the real thing as possible when performing our empirical evaluation. Additionally, we recognize that flight rules are inherently retroactive and corrective, yet safety demands proactive action. We believe that an important future direction would be to use an LLM to add a layer of code that proactively checks whether telemetry is trending towards a particular issue, so that a “soft” or “pre” alert could be issued to help prevent key issues from occurring.

We also see areas of opportunity to enhance the technical-side of our system. At present, our system relies on a SQLite backend, which we believe should be replaced with an object-relational-mapping (ORM) framework, allowing the system to be agnostic to the database type. Additionally, we should vectorize, thread, and parallelize our code to optimize runtime efficiency, especially on the SMS side, given that time-to-detection has life-or-death implications. Finally, as previously stated, we believe the current system may cause cognitive overload if many alerts are triggered simultaneously. We believe this may be a worthwhile area of research to advance our goal of increasing astronaut safety and independence while simultaneously reducing development and mission costs.

IX. Access, Distribution, and Contribution

Our prototype code is freely available and accessible on GitHub (https://github.com/profsandman/ices-auto-telemetry) as a Streamlit application (as seen above in Figure 7). Streamlit is a lightweight, Python-based framework for building interactive web applications with minimal code, allowing users to build full-stack applications in a single language. We have included full installation instructions, environment setup guides, and the synthetic flight rules and datasets in the repository, allowing anyone to modify, deploy, and test our proof-of-concept.

This project is fully open source, and we welcome contributions and partnerships from the aerospace community to enhance its capabilities, broaden its impact, and make the application more easily accessible to the various aerospace companies and international agencies engaged in space exploration. We would love the opportunity to advance this research on non-trivial, non-synthetic data and invite anyone interested in working with us to contact us for further discussion.

Acknowledgements

The authors would like to express their gratitude to the sponsors of ICES for the opportunity to research and to Marquette University for its support in this endeavor. In particular, we extend our appreciation to the AI Development and Value Creation and Accounting Programs for providing the resources and academic environment that facilitated this work. Furthermore, the team would like to thank the ICES chairs and reviewers for their thorough and thoughtful critique of this paper prior to publishing.

References

[1] J. Barreiro, J. Chachere, J. Frank, C. Bertels and A. Crocker, "Constraint and Flight Rule Management for Space Mission Operations," in *The International Symposium on Artificial Intelligence*, Sapporo, 2010.

[2] W. Mayo-Wells, "The Origins of Space Telemetry," *Technology and Culture,* vol. 4, no. 4, pp. 499-514, 1963.

[3] O. Kevorkova and A. Popov, "Personal Health Caare and corresponding technology with prognostic capability for astronauts - issues and challenges," in *2015 IEEE Aerospace Conference*, Big Sky, MT, 2015.

[4] S. Bell, D. Kortenkamp and J. Zaientz, "Real-time monitoring of ECLSS flight rules," in *39th International Conference on Environmental Systems*, Barcelona, Spain, 2010.

[5] A. Bowman, "GMSEC : Goddard Mission Services Evolution Center," NASA, 28th August 2024. [Online]. Available: https://www.nasa.gov/goddard/gmsec/. [Accessed 02 January 2026].

[6] A. Bowman, "GMSEC Components," NASA, 28 August 2024. [Online]. Available: https://www.nasa.gov/goddard/gmsec/components/. [Accessed 21 January 2026].

[7] M. Handy, "GMSEC Interface Specification," Goddard Space Flight Center, Greenbelt, Maryland, 2016.

[8] T. Quan Le, "OpenMCT," NASA, 2025. [Online]. Available: https://nasa.github.io/openmct/. [Accessed 02 January 2026].

[9] D. Lockney, "Open Mission Control Technologies (Open MCT)," NASA, 2025. [Online]. Available: https://software.nasa.gov/software/ARC-15256-1D. [Accessed 02 January 2026].

[10] M. Tipaldi and B. Bruenjes, "Survey on Fault Detection, Isolation, and Recovery Strategies in the Space Domain," *Journal of Aerospace Information Systems,* vol. 12, no. 2, pp. 235-256, 2015.

[11] A. Carlton, R. Morgan, W. Lohmeyer and K. Cahoy, "Telemetry fault-detection algorithms: Applications for spacecraft monitoring and space environment sensing.," *Journal of Aerospace Information Systems,* vol. 15, no. 5, pp. 239-252, 2018.

[12] W. Truszkowski, F. Paterra and S. Bailin, "Knowledge from Pictures (KFP)," in *In Technology 2002: The Third National Technology Transfer Conference and Exposition*, Washington D.C., 2002.

[13] M. Elaasar, N. Rouquette, K. Havelund, M. Feather, S. Bandyopadhyay and A. Candela, "Autonomica: Ontological modeling and analysis of autonomous behavior," in *33rd Annual INCOSE International Symposium*, Honolulu, Hawaii, 2023.

[14] A. Ganopol and M. Oglietti, "Space Modular Operations: A Powerful Method to Guarantee Flight Rules' Compliance," *Journal of Aerospace Information Systems,* vol. 19, no. 9, pp. 585-597, 2022.

[15] J. Rohrig, H. Sandidge and D. Olivas, "Asteria – A Spaceflight Artificial Intelligence Assistant," in *54th International Conference on Environmental Systems*, Prague, Czechia, ICES-2025-279, 2025.

[16] M. Parisi and et al., "The Impact of Delayed Communication," *Advances in Human Factors of Transportation,* vol. 148, pp. 49-57, 2024.

[17] E. Patterson, J. Watts-Perotti and D. Woods, "Voice Loops as Coordination Aids in Space Shuttle Mission Control," *Computer Supported Cooperative Work (CSCW),* vol. 8, no. 4, pp. 353-371, 1999.

[18] A. Fejjari, A. C. R. Delavault and G. Valentino, "A Review of Anomaly Detection in Spacecraft Telemetry Data," *Applied Sciences,* vol. 15, no. 10, p. 5653, 2025.

[19] J. Rohrig, T. Kirk, M. Torres, H. Sandidge, J. Wehr and J. Werschey, "Leto™ - A Spaceflight Intelligence System," in *53th International Conference on Environmental Systems*, Louisville, KY, ICES-2024-210, 2024.

[20] S. Tummala, M. Bhatia and R. Chun, "HERMES: An XGBoost Approach to Detecting Low-Quality Radio Transmissions on Martian Missions," in *2025 6th International Conference on Big Data, Artificial Intelligence, and Internet of Things Engineering*, Shanghai, China, 2025.

[21] D. L. Iversion, "System Health Monitoring for Space Mission Operations," in *Aerospace Conference, 2008 IEEE*, Big Sky, MT, 2008.

[22] D. L. Iverson and et al., "General Purpose Data-Driven System Monitoring for Space Operations," *Journal of Aerospace Computing, Information, and Communication,* vol. 9, no. 2, pp. 26-44, 2012.

[23] NASA Office of the Inspector General, "NASA's Management of Risks to Sustaining ISS Operations through 2030 (IG-24-020)," NASA, [Online]. Available: https://oig.nasa.gov/wp-content/uploads/2024/09/ig-24-020.pdf, 2024.

[24] A. Carrasco, V. Rodriguez-Fernandez and R. Linares, "Archive," 30 June 2025. [Online]. Available: https://arxiv.org/pdf/2505.19896. [Accessed 02 January 2026].

[25] A. Mozo and et al., "Adapting LLMs for Satellite Communications: Methodology, Challenges, and Impact," *IEEE Access,* vol. 13, pp. 155675-155696, 2025.

[26] E. Kurklu and K. Havelund, "A Flight Rule Checker for the LADEE Lunar Spacecraft," in *Theoretical Aspects of Computing – ICTAC 2020: 17th International Colloquium*, Macau, China, 2020.

[27] C. Culbert, G. Riley and R. T. Savely, "Verification Issues for Rule-Based Expert Systems," in *Third Conference on Artificial Intelligence for Space Applications*, Huntsville, Alabama, 1987.

[28] G. Falco and et al., "Mission Aware Cyber Safe Mode for Spacecraft," Sandia National Laboratories, Albuquerque, NM, 2025.

[29] C. Chullen, "Sensors and Systems for Spacesuits (JSC-CN-39828)," Johnson Space Center, Moffett Field, CA, 2017.

[30] UTC Aerospace Systems, "Lunar and Planetary Institute," September 2017. [Online]. Available: https://www.lpi.usra.edu/lunar/artemis/NASA-EMU-Data-Book-JSC-E-DAA-TN55224.pdf. [Accessed 23 January 2026].