[SSC26-RAI-07]

Toward Trainable AI Agents for Satellite Operations

Haley Calderon, Vaibhav Bhosale, Ketan Bhardwaj, Ada Gavrilovska
Georgia Institute of Technology
266 Ferst Dr, Atlanta, GA 30332
{hcalderon6,vbhosale6}@gatech.edu, {ketanby,ada}@cc.gatech.edu

ABSTRACT

Low Earth orbit is increasingly crowded, and a growing share of it is flown by universities and small operators. These teams must detect and respond to on-orbit anomalies with far less staff, expertise, and tooling than large operators, and a single missed anomaly can end a mission. We present COMET, a ground-based operator agent that helps a small team both detect anomalies and diagnose them. COMET pairs a learned detector with a retrieval-augmented diagnosis stage. An LSTM autoencoder flags anomalous telemetry windows, and a language model explains each one against the mission's own documentation, procedures, and prior operational records. Because the detector is trained on a mission's own telemetry and the diagnosis is grounded in its own documents, an organization can build an agent tailored to its spacecraft, before launch and during operations. The agent is advisory rather than autonomous, and every output is traceable back to the telemetry and documents that produced it. We describe the design and a proof-of-concept implementation, and evaluate COMET on a public benchmark of simulated multivariate spacecraft telemetry with injected anomalies. The detector recovers most of the labeled anomalies, favoring recall so that few are missed, and for a flagged event the diagnosis stage produces a traceable, documentation-grounded explanation. These results show the pipeline is feasible and its outputs auditable, a step toward reducing operator burden as deployments grow.

INTRODUCTION

The number of active satellites in low Earth orbit (LEO) keeps rising, and now exceeds twelve thousand.[1] This growth is not only mega-constellation traffic. More than three thousand nanosatellites, most of them CubeSats, have been launched to date, many built and operated by universities and small teams.[2] Programs such as the University Nanosatellite Program continue to add to this number. Every one of these spacecraft has to be watched. When something goes wrong on orbit, an operator has to spot it in the telemetry, work out what happened, and decide how to respond. For a small team, this is where a mission can be lost. Large operators staff control rooms around the clock and build tooling to match. A small team has neither. A single missed anomaly can turn a healthy satellite into space debris.[3]

Several factors make this difficult. On orbit, satellites are exposed to space weather, including solar activity and coronal mass ejections, that can push a system into unexpected behavior.[4] Such effects appear only indirectly, in telemetry. That telemetry is high-dimensional and tightly coupled to mission-specific hardware, software, and procedures. Interpreting it takes extensive training. Missions are often locked into specific software versions, which limits the reuse of tooling and expertise across deployments. Both detecting an anomaly and choosing a safe response demand scarce operator expertise. An operator might place a satellite in safe mode during a power fault, or disable a thruster that is causing unwanted spin.

Recent work has applied machine learning and artificial intelligence to ease this burden. A range of methods has been used on spacecraft telemetry, spanning classical classifiers, deep sequence models, and graph-based models.[5,6] Long Short-Term Memory (LSTM) networks,[7] a form of recurrent neural network, have proven effective at detecting anomalies in high-dimensional telemetry.[8] Most of this work targets detection alone. In spacecraft operations, fault detection, isolation, and recovery have traditionally been handled as separate stages.[9] Diagnosing a flagged anomaly and selecting a response still fall to the operator. Retrieval-augmented generation (RAG)[10] offers a way to close part of this gap, by grounding a large language model in mission-specific documents.

We present COMET,[1] a ground-based opera-

[1] Code available at `https://github.com/GTkernel/COMET`.

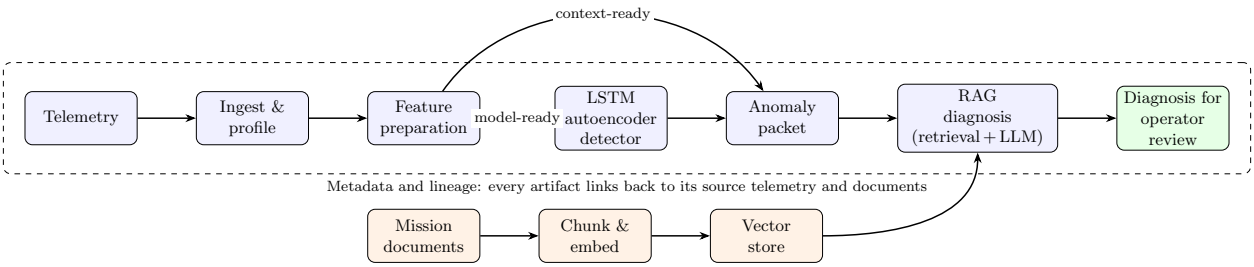

Figure 1: COMET architecture. Telemetry is ingested, profiled, and prepared into a model-ready representation for the detector and a context-ready representation for diagnosis. An LSTM autoencoder scores telemetry windows, and each flagged window becomes an anomaly packet. The packet is diagnosed against mission documents retrieved from a vector store, and the result is presented for operator review. Metadata links every artifact back to its source.

tor agent that addresses both detection and diagnosis. COMET combines an LSTM detector with a RAG diagnosis stage built on an existing large language model. The LSTM component learns patterns of nominal and anomalous behavior from historical telemetry. The RAG component contextualizes a detected anomaly using mission documentation, procedures, and prior operational data. COMET lets an organization train a mission-specific agent from its own telemetry and documents. This spans two phases. Before launch, a team trains COMET on historical or simulated telemetry. It also loads the mission documents, such as telemetry dictionaries, command references, and anomaly response guides. The agent is then ready when the satellite begins operations. After launch, COMET scores telemetry, flags anomalous windows, and retrieves the relevant procedures to support a response. The broader aim is to reduce operator burden and improve the scalability of ground operations as LEO deployments grow.

As a proof of concept, this paper contributes a metadata-driven ground-segment pipeline that preserves raw telemetry and links every anomaly back to its source records, prepared features, and retrieved documentation. The detector is an LSTM autoencoder, and its reconstruction output is packaged into structured anomaly packets. These packets form the interface between detection and diagnosis. A retrieval-augmented diagnosis stage then grounds language-model recommendations in mission documentation rather than in model parameters alone. We evaluate COMET end to end on the Controlled Anomalies Time Series (CATS) benchmark, a public dataset of simulated multivariate telemetry with labeled anomalies.[11] The detector recovers most of the labeled anomalies at a recall-oriented operating point, and for a flagged event the diagnosis stage aligns its top contributing features with the labeled root cause. The evaluation exercises the full detect-to-diagnose loop and establishes the feasibility and traceability of the pipeline.

BACKGROUND AND RELATED WORK

Machine learning for spacecraft telemetry anomaly detection is an active area, surveyed in detail by recent work.[5] Approaches range from classical classifiers to deep sequence and graph models. A widely cited example pairs LSTM networks with a nonparametric threshold to flag anomalies in multivariate telemetry.[8] Closer to the detector used here, an LSTM autoencoder with temporal attention reconstructs telemetry sequences and flags anomalies from the reconstruction error.[12] More recent methods use spatio-temporal graph learning to capture relationships across telemetry channels.[6] These methods share a common goal. They learn a model of nominal behavior and flag deviations from it.

Detection is only the first step. In spacecraft operations, fault detection, isolation, and recovery are traditionally treated as distinct functions.[9] A detector flags that something is wrong. An operator still has to identify the affected subsystem, consult the relevant procedures, and choose a response. Large language models have begun to assist with operations work of this kind. In cloud incident response, they have been used to recommend root causes and mitigation steps.[13] In the space domain, they have been studied as autonomous spacecraft operators[14] and as agents for anomaly detection in ground-segment data.[15] Retrieval-augmented generation grounds such a model in a document corpus rather than in its weights.[10] This suits operations knowledge, which usually lives in mission documentation.

Using a language model to detect anomalies directly has limits. A recent benchmark finds that such models struggle on multivariate aerospace telemetry, and that retrieval does not reliably improve their detection.[16] COMET takes this as a design cue. It keeps a learned detector for the detection step and uses retrieval only for diagnosis. An LSTM autoencoder scores telemetry, and a retrieval-augmented stage then explains a flagged anomaly using the mission's own documentation. COMET is advisory rather than autonomous, it runs on the ground, and every output is linked back to the telemetry and documents that produced it. This emphasis on a learned detector, on grounded and auditable diagnosis, and on a single mission's own data is what sets COMET apart from agents that let the language model both detect and act.

SYSTEM DESIGN AND ARCHITECTURE

COMET is a staged ground-segment pipeline that transforms raw telemetry into a diagnosis suitable for operator review. Figure 1 shows the stages and their data flow. This section presents the architecture and design decisions in COMET.

Telemetry schemas vary across missions and subsystems. COMET treats each telemetry source as an independent dataset, so a new mission can be ingested without aligning it to a common schema. To produce model input, COMET profiles each source and infers a role for every column. These role assignments are written to a human-readable feature configuration that an operator can inspect and edit before preparation. Automatic profiling handles the routine cases, and mission-specific knowledge can override it where needed.

The detector and the diagnosis stage place different demands on the same telemetry. The detector requires normalized numeric input, while diagnosis requires human-readable context. A single representation cannot serve both well. Feature preparation therefore produces two. The model-ready table holds normalized numeric and encoded categorical features for the detector. The context-ready table retains timestamps, identifiers, categorical states, and event notes for interpretation. This lets the detector operate on numeric input without discarding the context that diagnosis needs.

COMET assigns detection and diagnosis to different components. A recent benchmark reports that language models detect anomalies poorly in multivariate telemetry,[16] which is the regime of interest here. Detection is therefore performed by an LSTM autoencoder, which is suited to multivariate temporal sequences. The language model is used only for diagnosis, to explain a flagged anomaly against retrieved documentation.

Anomaly packets. Detection and diagnosis are separate stages, and they communicate through a single structured record, the anomaly packet. Each flagged window produces one packet, containing the anomaly score, the affected interval, the top contributing features, and the relevant context. The diagnosis stage consumes only this packet and does not access any raw telemetry schema. This decouples the two stages and provides a consistent unit of evidence for operator review.

Advisory, not autonomous. COMET produces diagnoses for review rather than commands. It identifies a likely cause, cites the relevant procedures, and states what additional data is required when the evidence is insufficient. It does not act on the spacecraft. When evidence is insufficient, the design favors escalation or abstention over a low-confidence recommendation. This keeps an operator in the loop, consistent with the safety constraints of flight operations.

Traceability by construction. Operators must be able to justify a response after the fact, and opaque model output is difficult to act on. COMET preserves the raw telemetry unchanged and separate from all derived artifacts, and links every artifact to its source. A diagnosis references its anomaly packet. The packet references the prepared features and the original telemetry rows. The diagnosis also references the documents it retrieved. This allows an operator to trace any output back to the underlying evidence.

Modular stages. Each stage has distinct data, failure modes, and responsibilities, so stages can be replaced independently. Ingestion is the clearest case. The current proof of concept reads static files, whereas an operational deployment would consume a live telemetry stream. Only the ingestion stage would change. Detection, packaging, and diagnosis would remain the same.

IMPLEMENTATION

This section describes how the stages in Figure 1 are implemented in the current proof of concept. The implementation is a set of command-line stages backed by a local SQLite database, invoked in dependency order. Each stage consumes the output of the previous one, carrying raw telemetry rows step by step toward a reviewable diagnosis.

Telemetry Ingestion

Telemetry enters as CSV files in a local directory, and each file is treated as an independent dataset. Before loading, COMET validates that a file is readable, has a header, has consistent row lengths, and contains at least one data row. A file that fails validation is skipped and logged, so one malformed file does not block the others. For each valid file, COMET records source metadata, including the file name, path, size, modification time, row count, and a SHA-256 content hash. The hash lets COMET skip files that have already been ingested. Raw rows are stored in a per-dataset table, separate from the metadata tables that track datasets, files, and columns. Column names are sanitized for the database during loading. This keeps the original telemetry intact while later stages build derived tables from it.

```
You are a satellite operator helping diagnose
telemetry anomalies.

Use only the provided anomaly packet, telemetry
summaries, and retrieved mission documentation.
If the evidence is insufficient, say what
additional information is needed.

Do not claim that the anomaly has a specific
category, event class, severity, or root cause
unless that value is explicitly present in the
anomaly packet or retrieved documentation. If
you infer a likely subsystem from feature names,
label it as an inference.

Anomaly Packet: {anomaly_packet}
Retrieved Mission Documentation: {retrieved_chunks}
Controlled Telemetry Tools Available:
{telemetry_tools}

Return:
1. Summary of the anomaly
2. Most likely subsystem or behavior
3. Evidence from telemetry
4. Relevant documentation references
5. Recommended next actions
6. Additional data needed, if any
```

Figure 2: The diagnosis prompt. The bracketed fields are filled at run time with the anomaly packet, the retrieved documentation chunks, and the available telemetry tools.

Profiling

Profiling takes a raw dataset and decides how each column is used downstream. For every column, COMET computes the missing-value ratio, the count and ratio of unique values, example values, and an inferred type, together with name-based hints. From these signals it assigns a role: timestamp, numeric telemetry feature, categorical feature, identifier, contextual field, event note, or excluded. The rules are deterministic, so the same dataset yields the same roles on every run. The result is a feature configuration written to a human-readable file. An operator can review and adjust it before preparation, which is where mission-specific knowledge enters the pipeline.

Feature Transformation

Feature transformation applies the feature configuration to the raw rows and produces the two derived tables. Numeric telemetry is converted to floating point and standardized, so channels with different units are comparable. Missing values are filled by interpolation where possible, with boundary fill at the ends of a sequence and mean fill as a fallback. Categorical fields are one-hot encoded, which avoids imposing an order on discrete states. Timestamps are standardized and retained, and row order is the fallback ordering when no timestamp is available. The model-ready table holds the standardized numeric and encoded categorical features. The context-ready table holds the human-readable fields used later for interpretation.

Anomaly Detection

Detection operates on the model-ready table and flags anomalous windows. The detector is an LSTM autoencoder. COMET orders the model-ready rows by timestamp, or by row identifier when no timestamp is present, and forms overlapping sliding windows. During training, the autoencoder learns to reconstruct these windows. At detection time, it reconstructs each window, and the reconstruction error is the anomaly score. The error is the mean squared difference between a window and its reconstruction, taken over the time and feature dimensions. A threshold is set from the training errors at a configurable percentile, and a window is flagged when its error exceeds the threshold. Window size, stride, the hidden size, the number of layers, and the threshold percentile are all configurable. COMET also computes the reconstruction error per feature, averaged over time, and records the highest-scoring features as the top contributors for that window. These rankings are diagnostic aids, not proof of causal fault origin. A high score means a feature was hard to reconstruct in that window, not that its subsystem caused the anomaly. When the per-feature scores show no useful variation, COMET falls back to simpler feature-change

information across the window. COMET saves a model checkpoint with the trained weights, the feature list, the configuration, and the threshold. A separate database record stores the model path, the threshold and its method, and a training summary. Together they let a later run reproduce the same scoring. Detection is currently batch-oriented, and the same scoring extends to streaming telemetry by keeping a rolling buffer of recent rows.

Anomaly Packets

Each flagged window is assembled into an anomaly packet, the record passed to diagnosis. COMET combines the stored anomaly record, the model metadata, the prepared rows, the context rows, and a reference to the raw telemetry into a single object. A packet carries the anomaly and dataset identifiers, the model identifier, the time and window range, the score and threshold, a numeric summary of the affected features, any categorical changes over the window, the top contributing features, and a reference back to the raw rows. The numeric summary reports the start and end values, the minimum and maximum, the change, and the percent change where it applies. The packet is the only input the diagnosis stage needs.

Retrieval-Augmented Diagnosis

The final stage turns an anomaly packet into a diagnosis, using mission documentation as context. These documents are ingested separately from telemetry, and include telemetry dictionaries, command references, and anomaly response guides. COMET reads them from a document directory, splits them into chunks, and stores the chunks in a local vector store. For each chunk it records metadata, including the source path, the file name, a document version, the chunk index, the text span, and a creation time. Diagnosis then follows a retrieval-augmented generation approach.[10] COMET turns an anomaly packet into a retrieval query built from the score, the threshold, the contributing features, the numeric summary, and the categorical changes. It retrieves the most similar chunks, combines them with the packet and a prompt template, and queries a locally hosted language model. The model is asked to summarize the anomaly, identify the likely subsystem, cite the relevant documentation, recommend next actions, and state what further data is needed when the evidence is insufficient. The prompt constrains the model to rely only on the provided packet and retrieved documentation, to avoid unsupported claims, and to label any inferred subsystem as an inference rather than a fact. Figure 2 shows the prompt. The output is stored as a diagnosis record, with the retrieved chunk references and the prompt metadata, so the diagnosis can be reviewed and reproduced. If the language model is unavailable, COMET stores a fallback record that reports the anomaly evidence and notes that further review is required, so the pipeline still completes.

EVALUATION

Dataset and Documents

Measuring detection performance requires labeled anomalies, which a mission's own telemetry does not provide. We therefore evaluate on CATS, a public benchmark of simulated multivariate telemetry with injected anomalies.[11] CATS contains about 5 million samples at 1 Hz across 17 variables, spanning sensor readings, control commands, and external stimuli, with 200 anomalies of known location and category. Two of its properties suit COMET. The signal is noise-free, so any reconstruction error the detector reports comes from the dynamics rather than from sensor noise. And only the later part of the series contains anomalies, so the early part is effectively nominal and gives the autoencoder clean data to learn from. COMET ingests CATS as a single dataset and trains on the first fifth of the windows, which falls in that nominal portion.

Diagnosis draws on the CATS reference documentation, a subsystem mapping, an anomaly-category description, and a mission overview, which COMET chunks and stores in a local vector store.

Configuration and Workflow

The detector is a single-layer LSTM autoencoder with a hidden size of 32. COMET forms 60-sample sliding windows with a stride of 20 samples, which at the 1 Hz cadence are 60-second windows that advance 20 seconds at a time. It trains on the first 20% of the windows for three epochs, with a batch size of 128 and a learning rate of 0.001, and sets the anomaly threshold at the 95th percentile of the training reconstruction error. Diagnosis runs a Gemma 2B model locally through Ollama, retrieving over a ChromaDB vector store. The stages run in dependency order from a command line:

```
python -m src.cli ingest
python -m src.cli profile --dataset-id 1
python -m src.cli prepare --dataset-id 1
python -m src.cli train --dataset-id 1
python -m src.cli detect --dataset-id 1 --model-id 1
python -m src.cli anomaly-packet \
    --anomaly-id anomaly_dataset1_model1_window137988
python -m src.cli ingest-docs
```

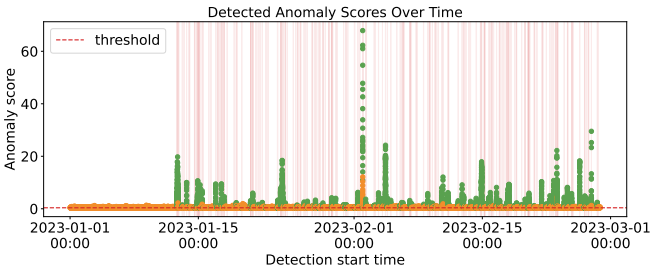

Figure 3: Anomaly score for each window over time. The dashed line marks the reconstruction-error threshold. Detections that match a labeled CATS interval and unmatched detections are shown in different colors.

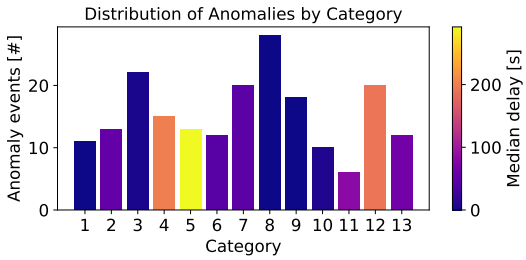

Figure 4: Number of anomaly events by category. Bar color indicates the median detection delay for each category.

```
python -m src.cli diagnose \
    --anomaly-id anomaly_dataset1_model1_window137988
```

The run used an isolated temporary database and artifact directory so it would not disturb working data.

Detection Results

Run over the full series, COMET scored about 250,000 windows and flagged roughly 7% of them. The 95th-percentile threshold fell at a reconstruction error of about 0.38. Measured against the 200 labeled intervals, the detector recovered 185, a recall of 0.925. Precision was lower, at 0.267, because many flagged windows lay outside the labeled intervals. On the intervals it did catch, it was reasonably prompt, with a mean detection delay of 112 seconds and a median of 29 seconds.

Other models were trained and tested following similar methodologies. We summarize the differences across model outputs in Table 1. Model 2 features a wider anomaly window of 120 seconds while model 3 includes both a wider anomaly window of 120 seconds and a second LSTM layer.

Metric	COMET	Model 2	Model 3
Precision	0.27	0.29	0.29
Recall	0.93	0.85	0.86
F2 Score	0.62	0.61	0.62
Mean [s]	112	103	112

Table 1: Comparison of model performance metrics.

The detector recovered most anomalies but raised many false alarms. For COMET this tradeoff is intended rather than incidental. The system surfaces candidates for a human to review, where a missed anomaly is the costly error and a spurious flag costs only a brief check. On that basis the operating point is recall-oriented. It recovers most labeled events while narrowing attention to a small fraction of the windows. The threshold sets the position on this tradeoff. A lower percentile would flag more windows and push recall higher, and a higher one would tighten precision. Flagged windows are ones whose reconstruction error is above the anomaly threshold. Currently, this means that a window is detected to be anomalous if any portion of the window overlaps with a labeled anomaly interval from the CATS metadata.

The delays were not uniform across anomaly types. However, identifying the reason for the distribution in anomaly detection delay is nontrivial given the CATS categories are unknown (Figure 4).

Diagnosis

Detection only identifies a window. Whether the diagnosis that follows can be trusted is a separate question, and to examine it we looked at the highest-scoring window, window 137988. It lies within a labeled CATS event whose root cause is channel bed2 and whose affected channel is ced1. The anomaly packet ranked ced1 and bed2 as the top contributing features, matching the labeled root cause and affected channel. Retrieval pulled in the subsystem mapping that links bed2 to ced1, and the model described the event as a disturbance in bed2 propagating to ced1, citing the large change in bed2 and the high reconstruction error on ced1. For this event the chain held, with the detector's evidence, the retrieved documentation, and the explanation all agreeing with the ground truth.

This is one hand-picked case, so it illustrates the diagnosis stage rather than measuring it, and it also

exposes a failure mode. The model referred to a period of high noise activity, which CATS does not contain, since the benchmark is noise-free. The claim is unsupported by the data. It is the kind of error the prompt constraints are meant to limit, and the kind a systematic diagnosis evaluation would have to catch.

DISCUSSION AND FUTURE WORK

The pipeline runs end to end, from ingestion through a documentation-grounded diagnosis, with every output traceable to its source. For a small operator, the value is less in any single model than in this combination. The detector is trained on a mission's own data, the diagnosis is grounded in the mission's own documents, and the agent advises rather than acts. These are the properties that make the tool auditable and suited to a small team.

The evaluation has clear limits. CATS is a simulated and noise-free benchmark, not real flight telemetry, so the detection numbers likely overstate what the detector would achieve on noisy field data. Precision is low at the recall-oriented operating point, which suits a surfacing aid but would burden an operator unless the threshold is calibrated. The diagnosis stage was examined on a single case, which illustrates its behavior but does not measure it.

The clearest next step is validation on real flight telemetry, where noise, gaps, and mission-specific behavior are present. Public datasets of real spacecraft telemetry with labeled anomalies, such as the NASA SMAP and MSL sets, would serve this purpose.[8] Training, validation, and test intervals should be kept separate, so the threshold is not tuned on the data it is judged against. The detector should also be measured against simpler baselines, including statistical thresholding, isolation-based methods, an autoencoder without temporal structure, and against a language model used directly as a detector, to show where the temporal model helps.

Operational use would require moving from batch to streaming. The current batch scoring extends to online inference by keeping a rolling buffer of recent rows and scoring each new window, which also means persisting the normalization parameters learned during training. Repeated training raises a model lifecycle question. An operational system needs explicit model selection, promotion, rollback, and audit, and a threshold calibrated from validation data and an operator's risk tolerance rather than the fixed percentile used here.

The diagnosis stage needs its own evaluation, which the single case here only motivates. Per-feature reconstruction error guides attention but does not establish cause, so stronger attribution and temporal localization within a window would help. The language-model output should be checked for grounding, since even the one case produced a claim the data does not support. Retrieval metrics, expert review, and explicit hallucination and safety checks would all be needed, because an inaccurate or unsafe recommendation is worse than none.

Finally, deployment raises operations and governance work. An operator interface would need alert acknowledgement, notes, and disposition tracking. The local store would be replaced by mission-grade persistence for concurrency, backup, and monitoring. Access control, audit logging, and controls around model promotion and any command recommendation would be required before the system could support live operations.

CONCLUSION

Small satellite operators face the same anomalies as large fleets, but without the staff and tooling to handle them. COMET is a ground-based agent that addresses this by pairing a learned detector with a retrieval-augmented diagnosis stage, both built from a single mission's own telemetry and documents. The detector flags anomalous telemetry windows, and the diagnosis stage explains each one against the mission's documentation, with every output traceable to its source. We presented the design and a command-line proof of concept, and evaluated it on the CATS benchmark. The detector recovers most labeled anomalies at a recall-oriented operating point, and for a flagged event the diagnosis aligns its top contributing features with the labeled root cause. Validating detection on real flight telemetry, evaluating the diagnosis stage systematically, and hardening the system for operations are the main steps toward use in the field.

Acknowledgments

This work was supported in part by NSF grant CNS-2345827. We thank Mason Starr for pointing us to the CATS dataset.

References

[1] Jonathan C. McDowell. Space statistics: Satellite and debris population. Jonathan's Space Report, `https://planet4589.org/`

space/stats/active.html, 2026. Accessed June 2026.

[2] Erik Kulu. Nanosats database, 2026. https://www.nanosats.eu, accessed June 2026.

[3] Craig Smith. An analysis of the 2016 Hitomi breakup event. *Earth, Planets and Space*, 69(1):51, 2017.

[4] Ryuho Kataoka, Daikou Shiota, Hitoshi Fujiwara, Hidekatsu Jin, Chihiro Tao, Hiroyuki Shinagawa, and Yasunobu Miyoshi. Unexpected space weather causing the reentry of 38 Starlink satellites in february 2022. *Journal of Space Weather and Space Climate*, 12:41, 2022.

[5] Asma Fejjari, Alexis Delavault, Robert Camilleri, and Gianluca Valentino. A review of anomaly detection in spacecraft telemetry data. *Applied Sciences*, 15(10):5653, 2025.

[6] Yi Lai, Ye Zhu, Li Li, Qing Lan, and Yizheng Zuo. STGLR: A spacecraft anomaly detection method based on spatio-temporal graph learning. *Sensors*, 25(2):310, 2025.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[8] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18)*, pages 387–395, 2018.

[9] Massimo Tipaldi and Bernhard Bruenjes. Survey on fault detection, isolation, and recovery strategies in the space domain. *Journal of Aerospace Information Systems*, 12(2):235–256, 2015.

[10] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wentau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020.

[11] Patrick Fleith. Controlled anomalies time series (CATS) dataset, 2023.

[12] Zhaoping Xu, Zhijun Cheng, and Bo Guo. A multivariate anomaly detector for satellite telemetry data using temporal attention-based LSTM autoencoder. *IEEE Transactions on Instrumentation and Measurement*, 72:1–13, 2023.

[13] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 1737–1749, 2023.

[14] Alejandro Carrasco, Victor Rodriguez-Fernandez, and Richard Linares. Large language models as autonomous spacecraft operators in Kerbal Space Program. *Advances in Space Research*, 76(6):3480–3497, 2025.

[15] Evan J. Chou, Lisa S. Locke, and Harvey M. Soldan. Automating the deep space network data systems: A case study in adaptive anomaly detection through agentic ai, 2025. arXiv:2508.21111.

[16] Yang Liu, Yixing Luo, Xiaofeng Li, Xiaogang Dong, Bin Gu, and Zhi Jin. Evaluating large language models for time series anomaly detection in aerospace software, 2026. arXiv:2601.12448.