Literature Review and Validation of Anomaly Detection in Spacecraft Telemetry Data

Ishaan Agarwal

ABSTRACT

Spacecraft anomaly detection is critical for mission safety, but deploying capable models on space-grade hardware remains a practical barrier. Ground-based monitoring introduces delays that grow with distance and disappear entirely during communication blackouts, while on-board systems are constrained by processors that are orders of magnitude less powerful than their ground counterparts. This paper evaluates whether an existing LSTM-based detection pipeline can be adapted and compressed to better meet those constraints without sacrificing detection capability. Using the ESA Anomaly Detection Benchmark Mission 1 lightweight subset, the Telemanom framework from Hundman et al. (2018) is replicated, tuned, and quantized across three phases. Phase 1 establishes a baseline CEF0.5 score of 43.57%. Phase 2 adjusts two hardcoded parameters originally calibrated for NASA telemetry, bringing CEF0.5 to 81.0% — a substantial improvement that demonstrates the sensitivity of threshold-based anomaly scoring to dataset-specific error distributions. Phase 3 applies post-training quantization, compressing the model from 310.7 KB to 91.9 KB while preserving prediction accuracy within 0.14% of the original output, establishing a path to onboard deployment without retraining or architectural redesign. Taken together, the three phases show that a well-understood LSTM architecture, with targeted adjustments and post-training compression, can approach competitive detection performance within the memory constraints of real flight hardware.

INTRODUCTION

The time between telemetry collection and interpretation can determine whether a mission succeeds or fails. The complex systems of spacecraft lead to numerous devices accumulating telemetry data, and this volume will only grow as vehicles become more complex. Current ground-based monitoring introduces delays in transmitting the massive amount of information from thousands of different parameters, leading to critical gaps in data reception. Telemetry can only be received when the vehicle is within a ground station's line of sight, so there are stretches where no one on Earth is watching at all. For a crewed vehicle in flight, a delay of minutes — whether from a loss of line-of-sight or slow data transmission — can eliminate the crew's remaining abort options.

An inviting alternative is on-board anomaly detection: interpreting telemetry directly on the spacecraft rather than waiting for transmission to reach the ground. This approach minimizes the time delay and provides a safety measure in case ground communication is unavailable. The new solution offers new problems of its own, however; spacecraft computers can only handle a fraction of the processing power that machines on the ground can, and they must be

built with operational safety measures. These constraints mean that on-board systems have historically relied on threshold-based fault detection systems (FDIR), or at most, lightweight machine learning models small enough to run on space-grade hardware. Any viable on-board approach must therefore be optimized to do more with far less.

Hardware is not the only constraint: Public data for training detection systems, conducting anomaly research, and developing new algorithms is scarce, and labeled data is harder still to find. The scarcity is compounded by the anomalies themselves, as their infrequency leaves few examples to label. Supervised algorithms require labeled examples to train on, but the lack of labeled data forces most solutions to be unsupervised or semi-supervised: algorithms that can rely on ordinary, unmarked data to learn what normal looks like and flag anything that doesn't.

One recent approach by Goetze et al. (2026) addresses these constraints directly, achieving strong detection performance with models small enough for on-board deployment [1]. This paper builds on their work in three phases. First, it runs the original Telemanom LSTM pipeline from Hundman et al. (2018) [2] on the ESA-ADB Mission 1 lightweight subset [3] and records a CEF0.5 score. Second, it slightly adjusts the error-detection logic and changes parameters to reduce pattern memorization and adapt to the new dataset. Third, it tests quantization as an alternative edge optimization strategy — converting input telemetry from FP32 to INT8 and measuring both the change in detection performance and processing time. While Goetze et al. optimized for edge deployment by rebuilding the model architecture, this work asks whether quantization can achieve that without touching the model at all.

LSTM & Neural Network Methods:

The paper from Goetze et al. (2026) bases its algorithms on Hundman et al. (2018) [2], but replaces the type of neural network used. A neural network is a model loosely inspired by how the brain processes information — it passes data through layers of connected nodes, adjusting the strength of those connections during training [4] until it learns to map an input to the correct output.

The network originally used was the long short-term memory (LSTM), a type of neural network designed for sequential data [5], where the order of inputs matters. Telemetry is a time series, so each reading only makes sense in the context of the ones before it. An LSTM carries a memory of past inputs forward as it moves through the sequence: it learns how a signal normally behaves over time, predicts the next value, and flags anything that breaks from that pattern.

Standard neural networks treat each input independently. Once you feed it a reading from timestep 100, it has no memory of timesteps 1 through 99. LSTMs fix this with a cell state, an

internal memory that carries information across timesteps. At each step, three learned gates decide what to do with that memory:

- Forget gate: looks at the current input and memory, decides what to discard
- Input gate: decides what new information to add
- Output gate: decides what part of the memory becomes the prediction

Each gate is a learned function of the current input and everything that came before it. The forget gate, for example, is defined as:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Where σ is the sigmoid function that outputs a value between 0 and 1, h_{t-1} is the model's memory from the previous timestamp, x_t is the current input, and W_f and b_f are weights and biases the model learned during training [5]. A value close to 0 means forget, close to 1 means keep. The model learns during training when each gate should fire and when it shouldn't, which is what allows it to recognize that a pressure reading at minute 500 is only meaningful relative to the 499 readings that came before it.

Background & Related Work

The ESA-ADB is the first publicly available satellite telemetry dataset built around real operational data with expert-annotated anomalies [3]. It draws from two ESA missions covering roughly 17.5 years of flight, with 844 total annotated events across both, 148 of which are confirmed anomalies, each labeled by spacecraft engineers and verified algorithmically. The dataset spans 176 telemetry channels in total, producing over 1.55 billion data points. At standard FP32 precision, that's several gigabytes of raw numerical data. Converting each value to INT8 should drop memory cost from 4 bytes to around 1 byte per value. For a system running on flight hardware with severely limited memory, the reduction in space could make the difference between a model being deployable or not.

A telemetry channel is a single sensor stream that records a single physical measurement over the duration of a mission. The collection of all channels together gives engineers a real-time picture of spacecraft health, and failures often show up as subtle shifts in how multiple channels relate to each other over time rather than any single reading crossing a limit.

This work uses channels 41 through 46 from Mission 1's lightweight subset. The six channels behave independently from the rest of the mission's subsystems, making them suitable for onboard deployment testing and straightforward to analyze without needing broader mission context [3].

Before conducting the forecasting and threshold approach, two classical detection algorithms were implemented as a baseline to represent the rule-based logic characteristic of traditional FDIR systems. FDIR, or Fault Detection, Isolation, and Recovery, is the standard health monitoring architecture used across most operational spacecraft, assigning each telemetry channel fixed upper and lower limits and triggering an alert when a reading crosses one [6]. It has remained the industry standard for decades because it is deterministic and straightforward to certify for flight hardware. Its core limitation lies in evaluating each channel independently, which means anomalies that develop gradually across multiple channels or stay within individual bounds will go undetected entirely. The two algorithms implemented here replicate that same logic at different levels of adaptability, allowing a direct comparison of how well threshold-based detection performs against the LSTM approach that follows. Each algorithm processed channels 41–46 from the ESA-ADB Mission 1 dataset and was scored at the event level: a true positive required at least one flagged point inside a labeled anomaly window, and each contiguous flagged region outside a true window counted as a single false positive. The baselines were scored at the event level rather than the data point level to help visualize how many anomalies FDIR systems miss.

Methods	Window	True Positives (events)	False Negatives (events)	False Positives (events)	Detection %	Precision %
Static Threshold	—	193	1038	0	15.70%	100.00%
Rolling Average	500	184	1047	96	14.90%	65.70%
Rolling Average	1000	187	1044	116	15.20%	61.70%
Rolling Average	2000	188	1043	124	15.30%	60.30%
Rolling Average	5000	209	1022	370	17.00%	36.10%

Figure 1: Table representation of the 2 FDIR-simulated tests on Channels 41-46

Across both algorithms and every window size tested, the results were underwhelming. The static threshold missed the vast majority of anomalies, and the rolling average barely improved on it; shrinking the window cut false positives but pulled detection down with it, so the tradeoff just shifted without the overall picture improving. No window size broke through the detection floor because the underlying problem was the same regardless of how the average was

computed. When a sustained anomaly fell inside the computation window, it dragged the reference value down with it, so the anomaly stopped looking unusual relative to the new baseline and the algorithm stopped flagging it; when the signal eventually returned to normal, that normal data now sat above the contaminated reference and got flagged instead. This is window contamination, and it is structural. No amount of parameter tuning fixes it because the corruption happens at the level of the method itself, not the settings. These results illustrate what traditional FDIR systems run into when they define normal at runtime using the same signal they are trying to monitor, and why that approach has a ceiling. A machine learning model trained on historical telemetry fixes its definition of normal before testing begins [2], so anomalies cannot contaminate the baseline used to detect them.

METHODOLOGY

This work implements the LSTM-based forecasting approach from Hundman et al. (2018) [2], originally developed and validated on NASA's SMAP satellite [7] and MSL rover telemetry [8]. Their codebase is publicly available and well-documented, making it a practical foundation for replication on the ESA-ADB dataset.

The first phase of this experiment runs the pre-configured Telemanom LSTM directly on the ESA-ADB dataset. Each channel file and the labels CSV had to be reformatted to match what the algorithm expected, though the process was straightforward. The parameters were slightly adjusted to keep training time manageable: both the training and testing sets span 84 months of data, around 7.6 million data points each, with the LSTM being trained with 30 epochs and a batch size of 512. The code was also run on both the system's CPU and GPU to decrease the training time. The priority for all three phases is to track the testing time, not the training time, as training is generally done on the ground instead of on onboard computers in flight.

The second phase tuned two hardcoded parameters in Telemanom that were originally calibrated for NASA's SMAP and MSL datasets [2] and didn't carry over well to ESA-ADB. The first change lowered the scale check in for its error detection from 0.05 to 0.02, a check that skips windows where the error looks too small to be worth flagging. The LSTM trained on ESA-ADB produced smaller errors across the board, including inside real anomaly windows, so the original value was passing over true anomalies. Lowering the threshold mitigated overlooked anomalies, allowing more to be detected. The second change narrowed the detection threshold from 12.0 to 8.0 standard deviations. This parameter determines how far out the threshold search looks in the error distribution. ESA-ADB's errors are tighter than NASA's [3], so real anomalies never came close to 12 standard deviations, and the search was looking too far out and missing them as a result. Narrowing from 12 to 8 standard deviations brought the threshold back to a range that made sense for this data.

The third phase tests quantization as an alternative path to edge deployment. Goetze et al. achieved edge optimization by rebuilding the model architecture from scratch through neural architecture search, but this phase asks whether the same result is achievable without re-training or changing the model architecture. Specifically, post-training dynamic quantization converts the pre-trained FP32 LSTM model into 8-bit signed integers [9] using PyTorch, maintaining the same weights from the original LSTM model. Changing the data format should reduce the memory footprint of incoming data by 75%, dropping memory bandwidth demands from 4 bytes to 1 byte per feature. Simultaneously, converting the models should use the corresponding SIMD (single instruction, multiple data) [10] accelerated integer pipeline to reduce the matrix multiplication cycles–the number of clock cycles required to execute layer calculations–before reaching the LSTMs. Ultimately, this phase measures the change in accuracy from sample tests and the change in testing time.

To evaluate accuracy preservation, both the original FP32 model and the quantized INT8 model were run on 769,033 sliding windows per channel drawn from the test set, and the mean absolute error between their predictions was computed. A full CEF0.5 score wasn't calculated for Phase 3 because the quantization had to be done in PyTorch rather than TensorFlow, but prediction Mean Absolute Error (MAE) served as a solution: Prediction MAE measures the average difference between two sets of model outputs. If the INT8 model produces nearly the same predictions as the FP32 model on the same inputs, the anomaly detection results will be essentially the same.

SOFTWARE

Telemanom: How It Works

Telemanom [2] trains an LSTM to predict what normal telemetry should look like, then flags any timestep where the prediction is significantly wrong. During training, the model processes the signal as overlapping 250-point windows, asking at each step what should come next given everything that came before it. Two stacked LSTM layers, each with 80 hidden units, learn the temporal structure of the signal over 30 training epochs using the Adam optimizer with mean squared error as the loss function [4], stopping early if validation loss stops improving to prevent the model from overfitting to noise. The training set included the full ESA-ADB data rather than purely nominal segments, though this did not significantly affect the results since the labeled anomalies are sparse enough relative to the total volume that the model learned normal behavior regardless.

At inference time, the model generates a prediction at every timestep and records how far off each prediction was from what actually happened. Those errors get smoothed using an exponentially weighted moving average before being passed to a non-parametric thresholding algorithm that searches for the cutoff point best separating high-error clusters from the normal

noise floor [2], finding that threshold independently for each channel rather than applying a single global cutoff. Anything above the denoted threshold gets noted as an anomaly (Figure 2).

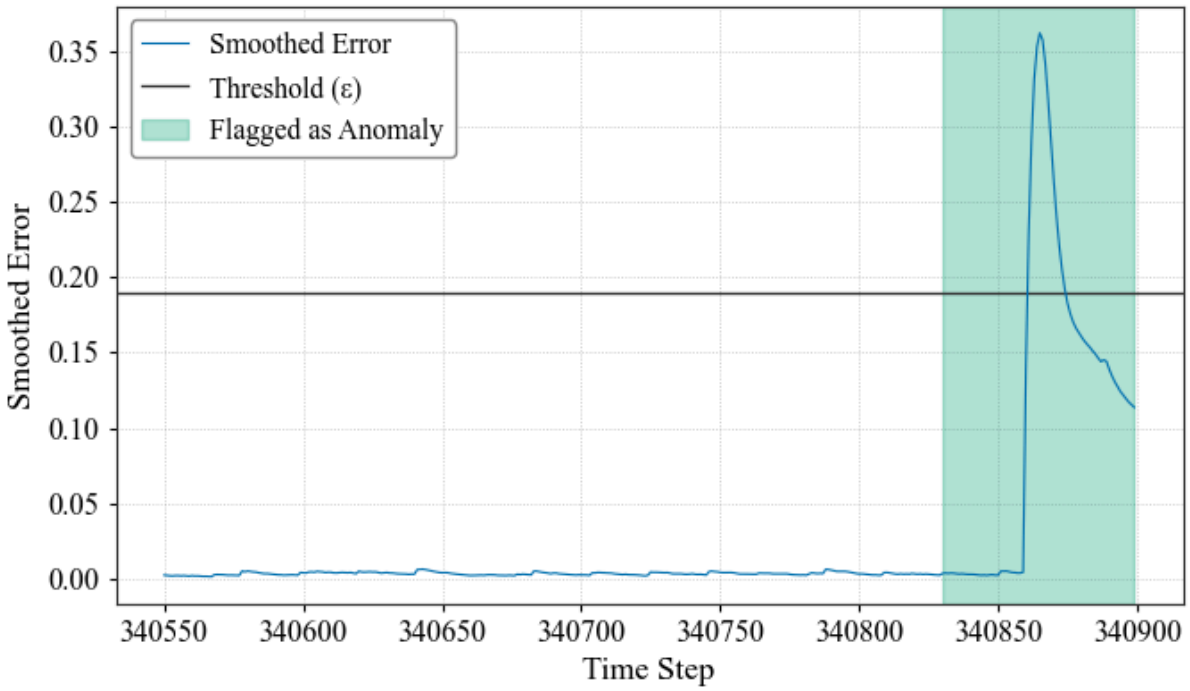

Figure 2: Non-Parametric Threshold Detection Example, Channel 41 (ε = 0.190)

Quantization: How It Works

The original plan was to compress the model using TensorFlow Lite [11], but that approach failed because Telemanom's LSTMs are trained with a GPU-optimized operation that TFLite's compression pipeline does not support, causing the conversion to break at the layer level. The fix was to pull the trained weights out of Telemanom's saved model files and rebuild the same architecture directly in PyTorch, using two stacked LSTM layers of 80 hidden units each followed by a single linear output layer. Keras and PyTorch store LSTM gate weights in different matrix orientations, so the four gate matrices had to be split and transposed into the shape PyTorch expects before the reconstructed model would behave identically to the original, though the gate order itself — input, forget, cell, output — stayed consistent across both frameworks.

With the weights correctly transferred, dynamic quantization was applied to the LSTM and linear layers, converting their stored weights from 32-bit floats to 8-bit integers in a one-time offline step before any inference runs [9]. Activations are handled differently because their value range is not known ahead of time, so the model computes a scale factor and converts to INT8 at each inference call, runs the operation, then converts the result back to floating point before passing it to the next layer. The quantization backend was set to QNNPACK when available, a backend built for ARM-oriented workloads and therefore relevant to eventual embedded deployment,

though any real performance advantage on an actual ARM flight processor remains unverified since the benchmark ran on the development machine [10]. Cutting weight storage from 32-bit to 8-bit reduces the serialized model from 310.7 KB to 91.9 KB regardless of what hardware runs the inference, and that memory reduction is what matters most for deployment on constrained flight computers.

To check that the conversion hadn't damaged accuracy, both the original and compressed models were run on the same 10% sample of 250-step test windows, drawn without replacement from the exact normalized test arrays Telemanom produced during Phase 1, rather than recomputing normalization from scratch. Their predictions were compared directly using mean absolute error. That MAE reflects how closely the FP32 and INT8 models agree with each other on identical inputs, not how close either one gets to the real telemetry values.

RESULTS

The first phase, with no changes made, resulted in a CEF0.5 score of 43.57%, which falls short of the 92.7% CEF0.5 baseline reported by Goetze et al. [1], though this was expected. The breakdown is more telling than the score itself: precision came in at 100% while recall sat at 13.38%, meaning the model never raised a false alarm but missed most of the anomalies it should have caught. The likely reason is that Telemanom's error thresholds were originally calibrated on NASA's SMAP and MSL datasets, and the ESA-ADB anomalies are sparse enough that those thresholds never got crossed — the model was effectively too conservative for this data, letting most anomalies pass through rather than risk being wrong.

Channel	Precision	Recall	CEF0.5	Prediction Error	Processing Time (s)
Channel 41	0.964	0.844	0.938	0.001584	16,230
Channel 42	0.551	0.871	0.595	0.002994	16,645
Channel 43	0.964	0.871	0.944	0.001834	16,592
Channel 44	0.818	0.844	0.823	0.002487	16,727
Channel 45	0.9	0.871	0.894	0.002013	16,557
Channel 46	0.628	0.9	0.668	0.003388	16,439
Overall	0.804	0.866	0.810	0.002383	99,190

Figure 3: Phase 2 Results (Telemanom, FP32)

The second phase produced an overall CEF0.5 of 81.0% across all six channels, which is a meaningful improvement over the 43.57% baseline from Phase 1 and demonstrates that the

trained LSTM is much better at catching anomalies than a static threshold approach. Channels 41 and 43 were the standouts, hitting 0.938 and 0.944 respectively with only one false positive each — the model was both precise and consistent on those channels. Channel 42 dragged the average down with 22 false positives and a CEF0.5 of 0.595, and Channel 46 had 16, suggesting those two channels have signal characteristics that the model found harder to distinguish from nominal behavior. The recall across all channels was strong, sitting between 0.844 and 0.900, meaning the model was generally catching the anomalies that were there, and the variance in CEF0.5 mostly came down to false positive rates, not missed detections. Still short of Goetze et al.'s 92.7%, but the gap is narrower, and the model is clearly learning meaningful patterns from the data.

Channel	FP32 Size (KB)	INT8 Size (KB)	Size Reduction	Speed Ratio (INT8/FP32)	Prediction MAE
Channel 41	310.7	91.9	3.4x	0.6x	0.003818
Channel 42	310.7	91.9	3.4x	0.5x	0.002345
Channel 43	310.7	91.9	3.4x	0.7x	0.000809
Channel 44	310.7	91.9	3.4x	0.6x	0.006164
Channel 45	310.7	91.9	3.4x	0.6x	0.001703
Channel 46	310.7	91.9	3.4x	0.6x	0.002455
Mean	310.7	91.9	3.4x	0.6x	0.002883

Figure 4: Post-Training Quantization Results for Channels 41-46

Across all six channels, quantization reduced model size from 310.7 KB to 91.9 KB, a consistent 3.4x reduction that falls just short of the theoretical 4x because the biases in the LSTM and linear layers stay in full 32-bit precision under dynamic quantization — only the weight matrices get converted to INT8 — with the added metadata and per-layer scale factors accounting for the rest of the gap (Figure 4). To check whether quantization hurt accuracy, both models were run on the same randomly sampled windows per channel, 10% of each channel's test set drawn without replacement, and their predictions compared directly using mean absolute error, which

averaged 0.002883 across channels, or about 0.14% of the full normalization range, meaning the INT8 model produces nearly identical outputs to the FP32 model on the same inputs.

Inference speed averaged a 0.6x ratio across channels, meaning the INT8 model ran slower rather than faster on the development machine. Dynamic quantization stores weights as INT8 but handles activations differently: each inference call quantizes the incoming activations on the fly, runs the matrix operations, then converts the result back to floating point before passing it to the next layer [9], and on a standard laptop CPU without vectorized low-precision execution paths, that per-call conversion adds overhead rather than removing it. On target hardware where those execution paths are available, the conversion cost disappears, and the reduced memory bandwidth produces a genuine speedup, though that remains unverified since the benchmark ran on the development machine [10]. The memory reduction holds regardless of hardware, since a model 3.4x smaller fits within the memory limits of onboard flight computers in a way the original FP32 version does not.

COMPARISON TO GOETZE ET AL.

Goetze et al.'s forecasting & threshold approach uses XceptionTimePlus, a convolutional network built on depthwise separable convolutions that processes an entire 224-point window in parallel [1]. This work's LSTM instead reads the sequence one timestep at a time, carrying a hidden state forward. Both feed into the same non-parametric dynamic thresholding from Hundman et al. [2] once forecasting is complete.

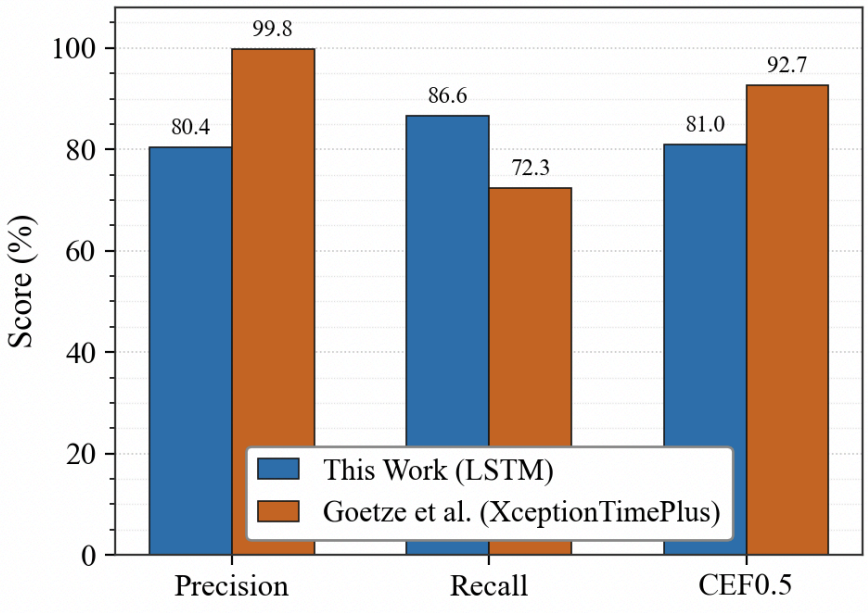

Figure 5: Comparison to Goetze et al.'s Forecasting & Threshold Baseline

The gap between the two CEF0.5 scores comes almost entirely from precision, not recall. Goetze et al.'s model was extremely conservative, catching 72.3% of anomalies while raising almost no false alarms. In comparison, this work's LSTM caught more anomalies overall at 86.6% recall but at the cost of far more false alarms, landing at 80.4% precision. Since CEF0.5 weights precision twice as heavily as recall by design [1], that precision difference explains most of what separates the two scores, not a shortfall in raw detection ability.

Even before any compression, this work's baseline LSTM was already smaller than Goetze et al.'s baseline CNN, 310.7 KB against 1,294 KB of read-only memory (ROM), the static storage size of a model on disk. This largely reflects a two-layer LSTM starting with far fewer parameters than XceptionTimePlus, not a difference introduced during training. After compression, the same pattern holds: this work's quantized model reached 91.9 KB against Goetze et al.'s NAS-optimized 166 KB.

This work's quantized model does not yet match Goetze et al.'s accuracy at a comparable size. Their optimized model retained a CEF0.5 of 88.8% after compression, while Phase 3 here relied on prediction MAE instead. The quantized model's predictions differed from the original FP32 model by an average of 0.14% of the normalized signal range, suggesting the compression itself introduced very little numerical error. That figure isn't directly comparable to Goetze et al.'s CEF0.5, though, since it measures agreement between the two models' raw outputs rather than detection accuracy against labeled anomalies. A direct accuracy-at-compressed-size comparison would require restoring TFLite compatibility, an open direction for this work.

DISCUSSION AND LIMITATIONS

Several implementation challenges came up during this study that are worth noting for reproducibility.

First, the Telemanom codebase has not been maintained since its 2018 release and needed modifications to run on modern hardware. The portion dedicated to errors included a section that passed the wrong data type as an array index, which newer versions of NumPy no longer accept. Fixing it required editing the library source code directly rather than adjusting any settings, which makes the pipeline harder to set up from scratch.

Second, the ESA-ADB dataset format does not match what Telemanom expects as input. Telemanom reads data from a specific folder structure of NumPy arrays, while ESA-ADB distributes its data as compressed files organized by channel. A preprocessing script was written to convert between the two formats. The anomaly labels had the same problem:

Telemanom expects labels as numerical index ranges, but ESA-ADB labels use timestamps and channel names, which had to be converted before Telemanom could score against them.

Third, the Phase 3 quantization was originally planned using TensorFlow Lite, which is the standard tool for deploying compressed models on embedded hardware [11] and would have allowed a direct accuracy comparison with Phase 2. That approach didn't work because the way Telemanom trains its LSTMs is incompatible with TFLite's compression process. The fix was to pull the trained weights out of Telemanom's saved files and rebuild the model in PyTorch, where the quantization worked without issue. The downside is that this created a gap between the quantized model and Telemanom's scoring pipeline, which is why Phase 3 reports a different accuracy metric than Phase 2.

CONCLUSIONS

This paper set out to ask whether machine learning-based anomaly detection could close the interpretation gap in spacecraft telemetry monitoring, and whether the resulting models could realistically run on the hardware that actually flies. Across three phases, the answer to both is cautiously yes.

The threshold baseline confirmed the fundamental limitation of traditional FDIR systems: fixed-limit monitoring catches a fraction of real anomalies regardless of how the window or threshold is configured. The trained LSTM brought overall CEF0.5 to 81.0%, a meaningful improvement that demonstrates deep learning detection is viable on real spacecraft telemetry. The distance between 81.0% and Goetze et al.'s 92.7% is largely attributable to architectural differences, since their approach rebuilt the model from the ground up through neural architecture search specifically optimized for the ESA-ADB, while this work applies the original Telemanom architecture with targeted parameter adjustments. Post-training quantization reduced the model to a third of its original size with prediction accuracy preserved to within 0.14% of the original output, establishing a path to onboard deployment without retraining or redesigning the model. The results show that viable anomaly detection within real flight hardware constraints is achievable with the right optimizations.

FUTURE WORK

Several directions follow naturally from the limitations identified here.

The most immediate is restoring TFLite compatibility by retraining Telemanom with standard LSTM operations rather than GPU-optimized ones. This would enable full INT8 quantization through TFLite's native pipeline and allow a direct CEF0.5 comparison between the FP32 and INT8 models, closing the evaluation gap that the PyTorch workaround introduced in Phase 3.

Extending the evaluation to all 176 channels across both ESA-ADB missions would test whether the Phase 2 results generalize across different spacecraft types, orbital profiles, and subsystems. The six channels used here were chosen for their independence and tractability, and broader coverage would give the deployment case considerably more weight.

Separately, whether missed anomalies in Phase 2 tend to produce error increases below the tuned threshold, rather than being invisible to the model outright, remains unexamined and would clarify where the remaining recall gap comes from.

On the model side, multivariate approaches that explicitly capture cross-channel relationships could detect the kinds of correlated drift signatures that single-channel LSTMs miss by design. Quantization-aware training, where INT8 rounding is simulated during the training process rather than applied after the fact, would also likely recover some of the accuracy that post-training compression leaves on the table.

Finally, an anonymized spacecraft anomaly repository modeled on NASA's Aviation Safety Reporting System [12] would give the broader research community access to labeled failure data that does not currently exist in the public domain. The detection performance achieved here is bounded in part by what can be done with the data that is available, and a shared repository would raise that ceiling for the entire field.

CODE AVAILABILITY:

The modified non-parametric thresholding module and the PyTorch quantization and benchmarking pipeline described in Phases 2 and 3 are available at https://github.com/ishaana7/esa-adb-quantization. The unmodified Telemanom codebase (Hundman et al., 2018) is available separately at https://github.com/khundman/telemanom.

REFERENCES

[1] Goetze, C., Schlippe, T., & Lakey, D. (2026). Deep Learning-Based Anomaly Detection in Spacecraft Telemetry on Edge Devices.
[2] Hundman, K., Constantinou, V., Laporte, C., Colwell, I., & Soderstrom, T. (2018). Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 387–395. https://doi.org/10.1145/3219819.3219845
[3] Kotowski, K., Haskamp, C., Andrzejewski, J., Ruszczak, B., Nalepa, J., Lakey, D., Collins, P., Kolmas, A., Bartesaghi, M., Martinez-Heras, J., & De Canio, G. (2024). European Space

Agency Benchmark for Anomaly Detection in Satellite Telemetry. arXiv. https://doi.org/10.48550/arXiv.2406.17826
[4] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536. https://doi.org/10.1038/323533a0
[5] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
[6] Jalilian, S., & Sabeghi, M. (2013). A survey on Fault Detection, Isolation and Recovery (FDIR) module in satellite onboard software. IEEE Conference on Space Operations (SpaceOps), 1–8. https://ieeexplore.ieee.org/document/6581270
[7] Entekhabi, D., Asrar, G. R., Betts, A. K., Beheri, M. A., & NASA SMAP Science Team. (2014). Soil Moisture Active Passive (SMAP) Mission Science Data Products. NASA Jet Propulsion Laboratory. https://smap.jpl.nasa.gov
[8] Grotzinger, J. P., Crisp, J., Vasavada, A. R., Anderson, R. C., Beegle, L. W., Bleacher, L. V., ... & Wiens, R. C. (2012). Mars Science Laboratory Mission and Science Investigation. Space Science Reviews, 170(1-4), 5–56. https://doi.org/10.1007/s11214-012-9892-2
[9] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2704–2713. https://doi.org/10.1109/CVPR.2018.00286
[10] Hennessy, J. L., & Patterson, D. A. (2017). Computer Architecture: A Quantitative Approach (6th ed.). Morgan Kaufmann.
[11] Warden, P., & Situnayake, D. (2019). TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. O'Reilly Media.
[12] NASA Ames Research Center. (2014). NASA Aviation Safety Reporting System (ASRS). NASA Technical Reports Server. https://ntrs.nasa.gov/citations/20140000574