UAV-GUARD: Efficient UAV Anomaly Detection via Normality-Guided Channel Selection and Statistical Temporal Descriptors

Ahmed Jlidi and Noura Aljeri
Computer Science Department, Kuwait University, Kuwait
s2231171736@ku.edu.kw; aljeri@cs.ku.edu.kw

***Abstract*—Unsupervised anomaly detection is important for UAV monitoring because labeled fault data are scarce and operating conditions vary across platforms. Existing methods often rely on costly sequence matching or deep models with high training and deployment overhead. We propose UAV-GUARD, a lightweight framework for anomaly detection in multivariate UAV telemetry. UAV-GUARD combines Patch-PCA representation, Normality-Influence Channel Selection (NICS), statistical feature extraction, and KNN anomaly detection with kernel density estimation. Patch-PCA creates a less noisy and more informative representation, NICS removes weak or redundant channels, Catch24 produces a small fixed set of descriptive features, and KDE selects the anomaly threshold automatically. Experiments on the ALFA UAV flight-fault dataset show that UAV-GUARD achieves strong detection performance while maintaining low runtime and lightweight processing, supporting its use for real-time UAV fault monitoring.**

***Index Terms*—UAV anomaly detection, UAV telemetry, mobility monitoring, multivariate time series, unsupervised learning, lightweight detection.**

I. Introduction

Unmanned aerial vehicles (UAVs) are widely used in applications such as traffic surveillance, border security, environmental monitoring, disaster assessment, and wildlife tracking [1]–[6]. Their ability to operate in large, hazardous, or hard-to-access areas makes them useful for missions where direct human deployment is difficult or unsafe.

Reliable UAV operation depends on continuous interaction between sensing, control, and communication systems. During flight, the UAV receives commands from the ground control station and returns telemetry such as position, speed, acceleration, orientation, and inertial measurements. These signals may become abnormal because of sensor faults, hardware failures, communication degradation, or cyber-attacks. Such abnormalities can appear as spikes, drift, persistent bias, frozen values, or changes in the relationship between multiple telemetry channels [7].

This makes anomaly detection an important component of UAV monitoring. The problem is naturally time-dependent because UAV measurements are ordered sequences rather than independent observations. Some faults may only become visible through temporal evolution, while others may appear through abnormal relationships between channels. Therefore, effective UAV anomaly detection must consider both temporal behavior and the multivariate structure.

Existing time-series anomaly detection methods include classical and deep learning approaches [8]. Classical methods are often simple and interpretable, but they may be limited when the data contain complex temporal and cross-channel patterns. Deep learning methods can learn more complex patterns, but they usually require more training cost, more parameters, and higher deployment cost.

Another challenge is the lack of labeled fault data. In real UAV missions, faults and attacks are rare, hard to reproduce, and may differ across platforms, environments, and operating conditions. For this reason, unsupervised anomaly detection is more practical, since it learns from normal flight data and identifies deviations without requiring labeled examples of every abnormal condition.

Motivated by these challenges, we propose UAV-GUARD, a UAV monitoring framework via Graph-aware Unsupervised Anomaly Recognition and Detection. UAV-GUARD combines patch-based temporal representation, graph-aware channel selection, statistical time-series feature extraction, and density-based thresholding. The framework is designed to detect anomalies from multivariate UAV telemetry while maintaining a lightweight structure suitable for real-time monitoring settings.

The remainder of this paper is organized as follows. Section II reviews related work on UAV anomaly detection and channel selection. Section III presents UAV-GUARD. Section IV reports the experimental setup and results. Section V concludes the paper and outlines future work.

II. Related Work

The related literature can be divided into three areas. The first covers UAV fault monitoring from flight telemetry, the second examines anomaly detection methods for time-series data, and the third considers channel selection for multivariate time series.

A. UAV Telemetry-Based Fault Monitoring

Anomaly detection for UAV monitoring aims at identifying abnormal behavior caused by faults, sensor irregularities, communication problems, or cyber-attacks [7]. UAV telemetry logs contain repeated measurements collected during flight, including position, velocity, attitude, inertial measurements, and control commands. These readings can be represented as

979-8-3195-0167-7/26/$31.00 ©2026 IEEE

a multivariate time series, where the detector examines how signals change over time and how they behave together.

This representation is useful because UAV anomalies may appear as drift, sudden signal changes, frozen values, abnormal attitude or velocity behavior, or mismatch between control commands and the measured UAV response [7], [9], [10]. Since abnormal cases are difficult to collect exhaustively and may vary across platforms and operating conditions, it is more convenient to use an unsupervised setting, where normal data are used to model expected behavior and deviations are treated as anomalies [7], [10], [11].

B. Classical Time-Series Anomaly Detection

Classical time-series anomaly detection methods usually rely on similarity, distance, or density measures. Distance-based methods compare a test sequence with normal reference sequences and assign higher anomaly scores to samples that are far from normal behavior [8], [12]. Dynamic Time Warping (DTW) is a common baseline because it compares temporal sequences while allowing local time shifts [13]. When combined with KNN scoring, distant windows receive higher anomaly scores.

However, direct sequence comparison can be expensive for multivariate UAV telemetry, especially with many channels and overlapping windows. It also uses the full temporal representation, which may preserve noisy or redundant signal variations. In contrast, our framework keeps the nearest-neighbor anomaly-scoring principle but applies it after filtering noisy channels and extracting statistical time-series features.

C. Deep Learning Methods for UAV Anomaly Detection

Deep learning methods for UAV anomaly detection learn normal behavior using reconstruction, encoding, likelihood estimation, or learned representations. UAV-focused examples include unsupervised encoding and thresholding [10], convolutional autoencoders with support-vector data description [14], stacked recurrent autoencoders [15], STE-RS [16], and MeNU [11]. General multivariate time-series detectors follow similar ideas, including OmniAnomaly [17], USAD [18], DAGMM [19], and GDN [20]. These models can learn complex temporal patterns, but often introduce many parameters, training cost, and deployment overhead.

In contrast, UAV-GUARD avoids deep model training and uses lightweight feature construction followed by density-based thresholding. It also relies on a small number of main parameters, making it easier to tune and more suitable for resource-constrained UAV settings.

D. Channel Selection in Multivariate Time Series

Reducing the channel space is one way to lower the cost before feature extraction and anomaly scoring. Existing channel-selection methods such as ECP [21] and TSelect [22] show that large reductions are possible, but they mainly target supervised classification and therefore rely on labeled data to perform pruning. This assumption is not suitable for the normal-only anomaly detection setting considered in this work. In contrast, our framework selects channels without explicit labels by measuring how much each channel affects the nearest-neighbor structure and score ordering of benign samples.

III. Methodology

We propose UAV-GUARD, a UAV monitoring framework via Graph-aware Unsupervised Anomaly Recognition and Detection. The general flow of the framework is shown in Fig. 1, and the main steps are summarized in Algorithm 1. UAV-GUARD is trained using only benign samples and detects anomalies according to their deviation from the learned normal behavior.

The framework is designed for telemetry windows generated during a UAV mission. Incoming flight measurements are streamed as overlapping multivariate windows and passed through four stages. Patch-PCA first reduces each window into a compact patch-based representation while preserving temporal order (Section III-A). NICS then performs graph-aware channel selection on the computed representation by measuring how much each channel affects the normal nearest-neighbor graph (Section III-B). From the selected channels, Catch24 extracts descriptive time-series features (Section III-C). KNN-KDE then computes the anomaly score, compares it with the learned threshold, and flags windows above the threshold as anomalous (Section III-D).

A. Patch-PCA Representation

Raw multivariate UAV telemetry can contain noise, local fluctuations, and redundant temporal detail. Using these measurements directly can also increase the cost of input-processing and anomaly detection. Patch-PCA addresses these issues by encoding short overlapping patches with PCA while preserving their temporal order in the final sample.

Let $X \in \mathbb{R}^{N\times C\times T}$ denote the training set, where C is the number of input channels. Each sample is divided into P overlapping patches of length w and stride s. For sample i, the patch starting at position t_p is the concatenation of the channel values within that local window:

$$\mathbf{a}_{i,p} = \big(X_{i,c,\,t_p+r}\big)_{c=1,\ldots,C;\ r=0,\ldots,w-1} \in \mathbb{R}^{Cw}. \tag{1}$$

All normal training patches are then fit to a PCA model, and each patch is projected onto the first D principal components:

$$\mathbf{z}_{i,p} = W^{\top}\mathbf{a}_{i,p}, \qquad W \in \mathbb{R}^{Cw\times D}. \tag{2}$$

Finally, the projected patches are arranged in their original temporal order to form the final sample:

$$Z_i = (\mathbf{z}_{i,1}, \mathbf{z}_{i,2}, \ldots, \mathbf{z}_{i,P}) \in \mathbb{R}^{D\times P}. \tag{3}$$

B. Normality-Influence Channel Selection (NICS)

UAV monitoring data form multivariate time series, where each sensor or communication source contributes one temporal channel. Some channels may be noisy, redundant, or weakly related to normal operating behavior; using all channels can therefore increase computation without improving the anomaly

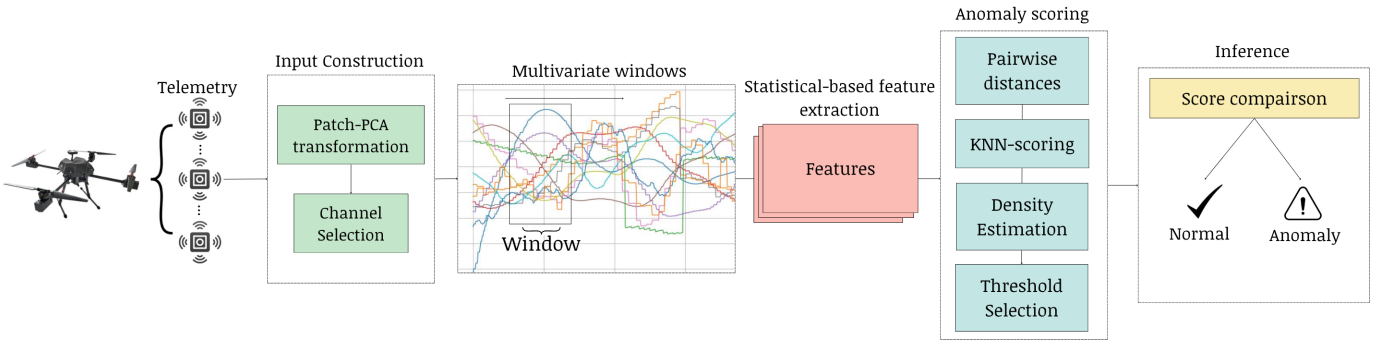


Fig. 1: Overview of the proposed UAV-GUARD framework.

Algorithm 1: UAV-GUARD framework

Input: Normal training windows X_{tr}, test windows X_{te}
Output: Predicted anomaly labels $\hat{y}$

```
Fit Patch-PCA using normal training windows;
Transform training and test windows with Patch-PCA;
Apply graph-aware NICS to the normal Patch-PCA representation;
Retain the selected Patch-PCA channels in both training and test representations;
Extract Catch24 features from the selected channels;
Fit KNN on normal training features;
Compute anomaly scores for the evaluated samples;
Estimate the threshold using KDE valley detection, with Otsu fallback;
Assign labels by thresholding the anomaly scores;
return ŷ;
```

detector. NICS selects channels by measuring their influence on the nearest-neighbor structure formed by normal training samples.

Let $X^* \in \mathbb{R}^{N\times C\times T}$ denote the normal training set after Patch-PCA, where N, C, and T are the number of samples, channels, and time steps, respectively. For each channel c, the pairwise squared Euclidean distance between samples i and j is

$$D_c(i,j) = \left\| X^*_{i,c,:} - X^*_{j,c,:} \right\|_2^2 . \tag{4}$$

The all-channel distance matrix is obtained by summing across channels:

$$D(i,j) = \sum_{c=1}^{C} D_c(i,j). \tag{5}$$

Using D, the k-nearest-neighbor set of sample i is denoted $N_k(i)$, and its neighborhood score is

$$s_i = \frac{1}{k} \sum_{j\in N_k(i)} \sqrt{D(i,j)}. \tag{6}$$

To estimate the influence of channel c, NICS removes it from the distance computation:

$$D^{(-c)}(i,j) = D(i,j) - D_c(i,j), \tag{7}$$

and recomputes the neighbor sets $N_k^{(-c)}(i)$ and score vector $\mathbf{s}^{(-c)}$ from $D^{(-c)}$. A channel is considered influential when its removal changes either the nearest-neighbor structure or the ordering of normal samples by their scores.

These two effects are captured by separate loss terms. The graph term measures the average loss of neighbor overlap:

$$L_c^{\text{graph}} = 1 - \frac{1}{N} \sum_{i=1}^{N} \frac{\left| N_k(i) \cap N_k^{(-c)}(i) \right|}{k}. \tag{8}$$

The score term uses Spearman correlation between $\mathbf{s}$ and $\mathbf{s}^{(-c)}$, since the relative ordering of samples is important:

$$L_c^{\text{score}} = 1 - \max\left(\rho_s\left(\mathbf{s},\, \mathbf{s}^{(-c)} \right), 0 \right). \tag{9}$$

The two terms are combined with equal weight into a single influence score:

$$I_c = \frac{1}{2} L_c^{\text{graph}} + \frac{1}{2} L_c^{\text{score}}. \tag{10}$$

After computing I_c for all channels, NICS applies Otsu thresholding to determine the cutoff point. Otsu thresholding selects the cutoff that best separates a score distribution into two groups by minimizing the within-group variance. This is suitable here because we assume that the channel scores form two major groups: channels that strongly affect the normal neighborhood structure and channels with lower influence. The selected subset is

$$\mathcal{C}^* = \{ c \mid I_c > \theta_{\text{Otsu}} \}, \tag{11}$$

and the remaining stages of the framework operate only on these channels.

C. Catch24 Feature Extraction

After channel filtering and Patch-PCA reduction, the next stage extracts features from the reduced time-series samples. Several time-series feature extraction methods could be used at this stage, including kernel-based transforms [23]. Since the goal of UAV-GUARD is to remain lightweight, we use

Catch24 [24] as a statistical feature extractor. Catch24 summarizes each temporal channel using a small set of 24 descriptive time-series features. These features capture different properties of the signal, such as distributional behavior, variation, autocorrelation, and temporal fluctuation patterns. Another advantage of this approach is that the extracted features can be passed directly to lightweight anomaly-scoring methods. This avoids the need for direct sequence matching, as in DTW-based methods, or recurrent model training.

Let $Z_i \in \mathbb{R}^{D\times P}$ denote the Patch-PCA representation of sample i, where D is the number of PCA channels and P is the number of ordered patches. For each PCA channel d, the corresponding univariate time series is

$$\mathbf{z}_{i,d} = (Z_i(d,1), Z_i(d,2), \ldots, Z_i(d,P)). \tag{12}$$

Catch24 is applied independently to each $\mathbf{z}_{i,d}$. It computes the canonical catch22 features together with the mean and standard deviation, producing a feature vector consisting of 24 features for each PCA channel:

$$\boldsymbol{\phi}_{i,d} = \mathrm{Catch24}(\mathbf{z}_{i,d}). \tag{13}$$

The final representation of sample i is obtained by concatenating the Catch24 feature vectors extracted from all PCA channels:

$$\boldsymbol{\Phi}_i = [\boldsymbol{\phi}_{i,1}^\top, \boldsymbol{\phi}_{i,2}^\top, \ldots, \boldsymbol{\phi}_{i,D}^\top]^\top. \tag{14}$$

$\boldsymbol{\Phi}_i$ represents the fixed-length feature vector for each sample, which is used by the anomaly-scoring stage.

D. KNN-KDE Anomaly Detection

After Catch24 feature extraction, each sample is represented as a fixed-length feature vector. We use a nearest-neighbor detector because it provides a lightweight measure of how far a sample lies from the normal training distribution. The detector is fitted using only normal samples.

Let

$$F_{\mathrm{tr}} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N\}, \qquad \mathbf{f}_i \in \mathbb{R}^M,$$

denote the Catch24 feature vectors extracted from the normal training set. For an input sample with feature vector $\mathbf{f}$, the anomaly score is computed as the average Euclidean distance to its k nearest normal training samples:

$$s(\mathbf{f}) = \frac{1}{k} \sum_{\mathbf{f}_j \in \mathcal{N}_k(\mathbf{f}; F_{\mathrm{tr}})} \|\mathbf{f} - \mathbf{f}_j\|_2 , \tag{15}$$

where $\mathcal{N}_k(\mathbf{f}; F_{\mathrm{tr}})$ is the set of k nearest neighbors of $\mathbf{f}$ in the normal training feature set F_{tr}. Larger scores indicate that the sample is farther from the learned normal behavior.

A common way to convert anomaly scores into binary decisions is to use a fixed high quantile of the normal training scores as the threshold. For example, if S_{tr} denotes the anomaly scores of the normal training samples, the threshold can be written as

$$\theta_q = Q_q(S_{\mathrm{tr}}), \tag{16}$$

where $Q_q(\cdot)$ is the q-th quantile. This rule is simple, but its performance depends on the selected value of q. A quantile that works for one score distribution may be too strict or too permissive for another, especially when the score scale changes across datasets or monitoring sources.

Instead, we estimate the threshold from the score distribution of the evaluated batch using kernel density estimation (KDE). Let

$$S = \{s_1, s_2, \ldots, s_n\}$$

denote the unlabeled anomaly scores of the evaluated samples. KDE estimates the one-dimensional score density as

$$\hat{p}(s) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{s - s_i}{h}\right), \tag{17}$$

where $K(\cdot)$ is a Gaussian kernel and h is the bandwidth. The bandwidth is selected automatically using a Silverman-style rule.

The estimated density is evaluated on a dense score grid. The lowest-score density peak is treated as the normal-score mode, and the next peak to its right is treated as the higher-score mode. Let p_{low} and p_{high} denote the locations of these two peaks. The KDE threshold is selected as the density valley between them:

$$\theta_{\mathrm{KDE}} = \arg \min_{s\in[p_{\mathrm{low}}, p_{\mathrm{high}}]} \hat{p}(s). \tag{18}$$

If the KDE does not show a valid two-peak structure, Otsu thresholding is used as a fallback to obtain an automatic split of the score distribution. The final threshold is denoted by θ, where $\theta = \theta_{\mathrm{KDE}}$ when a valid KDE valley is found.

The final anomaly decision for sample i is

$$\hat{y}_i = \begin{cases} 1, & s_i > \theta, \\ 0, & s_i \le \theta, \end{cases} \tag{19}$$

where $\hat{y}_i = 1$ denotes an anomalous prediction and $\hat{y}_i = 0$ denotes a normal prediction. This thresholding strategy avoids manually selecting a fixed quantile and instead places the decision boundary according to the observed separation in the anomaly-score distribution.

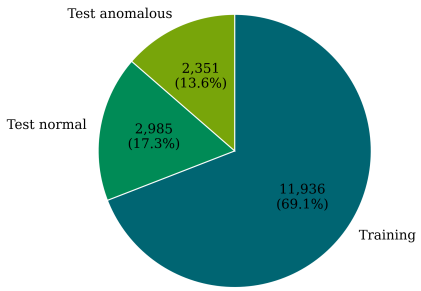


Fig. 2: Distribution of labels in the pre-processed ALFA dataset

IV. Performance Evaluation

A. Dataset

We evaluate the proposed framework on the ALFA dataset, which represents physical UAV flight-fault detection from

real telemetry logs. ALFA contains flight data collected from a Carbon-Z T-28 fixed-wing UAV under benign and failure conditions. The anomalous cases include engine, elevator, aileron, and rudder faults, allowing the evaluation to cover different types of physical UAV faults.

The raw ALFA logs contain telemetry variables related to aircraft motion, orientation, navigation, and control response. Variables that are unrelated to the UAV state or fault behavior such as timestamp are excluded.

After feature processing and synchronization, the data are converted into multivariate time-series samples using a fixed window length of $T_{\text{win}} = 200$ time steps and a stride of $S_{\text{win}} = 5$ time steps. Since 5 time steps correspond to 0.25 s, the stride is 0.25 s and each window covers approximately 10 s. Each resulting sample is represented as

$$X_i \in \mathbb{R}^{24 \times T_{200}}, \tag{20}$$

where 24 is the number of input variables before Patch-PCA and channel selection, and T_{win} is the number of time steps in each sampled window. Figure 2 shows the sample distribution after time-series construction.

B. Experimental setup

The experiments follow an unsupervised anomaly detection setting. Only benign samples are used for training, while the test set contains the remaining benign samples together with all anomalous samples. The original labels are converted into a binary form, where benign samples are assigned label 0 and all fault samples are assigned label 1. All preprocessing steps that require fitted statistics, including normalization and PCA, are fitted only on the normal training data and then applied to the test set.

1) Metrics: The evaluation reports accuracy, precision, recall, F1-score, and ROC-AUC. To assess the lightweight nature of the framework, we also report channel reduction, end-to-end runtime, peak memory usage, mean processing latency, mean decision delay, real-time factor (RTF), and throughput. All experiments were implemented in Python 3.14.6, with CPU execution restricted to a single thread. Channel reduction measures the percentage of input channels removed by NICS. End-to-end runtime includes the full evaluation pipeline, while processing latency measures the time required to process an available test window during the online decision stage. Peak memory usage is recorded during execution to estimate memory pressure. The real-time factor compares the mean processing latency per window with the window stride duration:

$$\text{RTF} = \frac{\text{processing latency per window}}{\text{window stride duration}}. \tag{21}$$

An RTF below 1 indicates that the method can keep up with streaming windows. Lastly, Throughput is computed as

$$\text{Throughput} = \frac{\text{number of windows}}{\text{total processing time}} \tag{22}$$

2) Parameter Configuration: UAV-GUARD uses a small set of fixed design parameters and one tunable parameter. The Patch-PCA stage is fixed with a patch size of 10, stride of 5, 40 ordered patches, and 32 PCA components. This produces a reduced time-series segment with 32 channels and 40 time steps, corresponding to a 10-second window.

The tunable parameter is the number of nearest neighbors k, which is used in both NICS and the final KNN anomaly detector. We test $k \in \{1, 3, 5, 10\}$ and report the results for each setting.

3) Baselines: We compare UAV-GUARD with representative classical and deep unsupervised anomaly detection baselines. DTW-KNN is used as a classical sequence-matching baseline, where DTW measures time-series similarity and KNN computes the anomaly score [8]. For deep learning baselines, we include LSTM-AE [7] as recurrent reconstruction, OmniAnomaly [17] as variational sequence modeling, MeNU [11] as memory-augmented detection, and LVA-SP [25] as lightweight VAE-based.

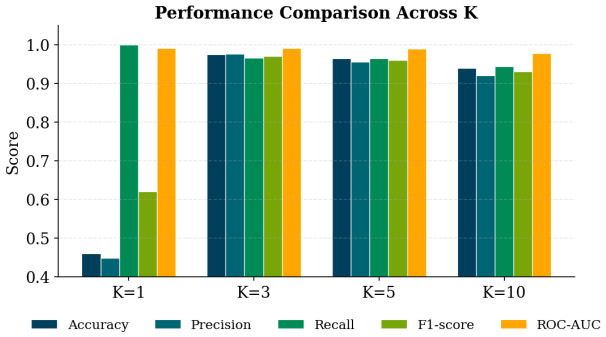


Fig. 3: Detection performance obtained using different nearest-neighbor values. The setting $k = 3$ provides the best balance across the evaluated metrics.

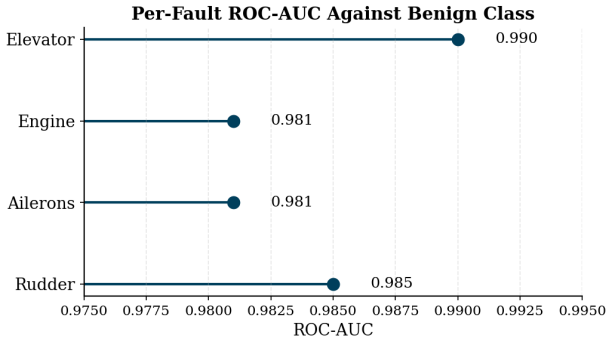


Fig. 4: Per-fault ROC-AUC against the benign class on ALFA.

C. Results

1) Detection Performance: Figure 3 shows the effect of changing the nearest-neighbor parameter k. The best setting is obtained at $k = 3$, with an accuracy of 0.974, precision of 0.976, F1-score of 0.970, and ROC-AUC of 0.991. Using

only one neighbor gives a high ROC-AUC but poor accuracy and precision, because the decision becomes too dependent on the closest normal sample. This leads to many benign windows being assigned high anomaly scores. Beyond $k = 3$, the performance gradually drops: the F1-score decreases from 0.970 at $k = 3$ to 0.960 at $k = 5$ and 0.931 at $k = 10$. Therefore, $k = 3$ is used for the remaining experiments.

Table I reports the final detection results using the selected setting $k = 3$. The framework achieves an accuracy of 0.974, precision of 0.976, recall of 0.965, F1-score of 0.970, and ROC-AUC of 0.990. Figure 5 further shows the anomaly-score distribution used for thresholding. Benign windows are concentrated in the lower-score region, while most anomalous windows appear at higher scores. The KDE valley threshold is placed between these two regions at 10.64, which is consistent with the strong precision and recall reported in Table I. The clear separation in the score distribution also supports the use of KDE valley thresholding.

Since the ALFA test set is imbalanced, Fig. 4 reports the ROC-AUC of each fault type against the benign class to examine whether all fault types are handled consistently. The detector obtains high ROC-AUC for all faults: 0.990 for elevator, 0.985 for rudder, and 0.981 for both engine and aileron faults. The most difficult cases are engine and aileron faults, but their ROC-AUC values remain above 0.98.

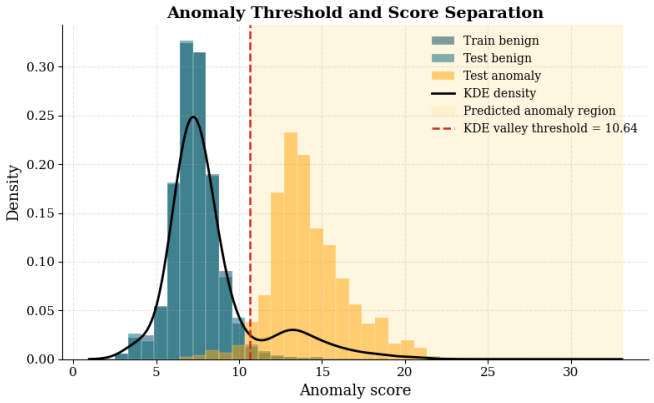


Fig. 5: Anomaly-score distributions with the KDE valley threshold. The threshold lies between the benign score region and the higher anomaly-score region.

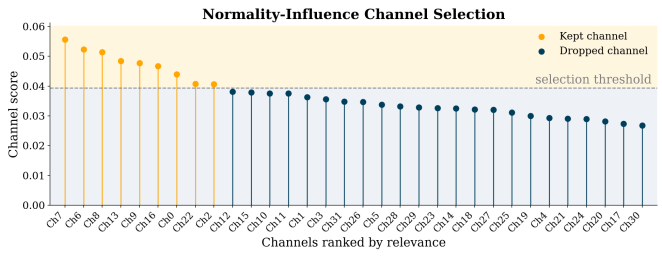


Fig. 6: NICS channel influence scores and cutoff threshold. Channels above the threshold are retained for the following stages.

TABLE I: Detection performance and efficiency results of UAV-GUARD on the ALFA dataset.

Metric	Result
Detection Performance	
Accuracy	0.974
Precision	0.976
Recall	0.965
F1-score	0.970
ROC-AUC	0.990
Efficiency	
Channel Reduction (%)	71.9
Runtime (s)	104.75
Peak Memory Usage (KB)	490
Mean Decision Delay (ms)	5.07
Real-Time Factor	0.0142
Throughput (windows/s)	281.4

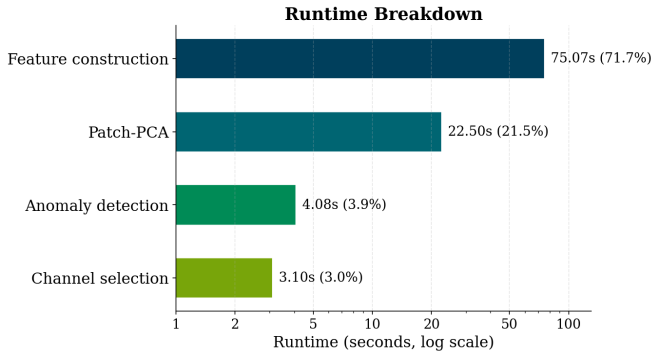


Fig. 7: Runtime breakdown of UAV-GUARD.

2) Efficiency: Figure 6 shows the channel scores produced by NICS. The method keeps 9 channels from the original 32 and removes the remaining 23 channels. This corresponds to a channel reduction of 71.9%. The selected channels are those above the automatic threshold, while the dropped channels have lower influence on the normal neighborhood structure.

The runtime results are reported in Table I and Figure 7. The full framework requires 104.75 seconds end-to-end. Channel selection adds only 3.10 seconds, or 3.0% of the total runtime, while providing the 71.9% channel reduction. Most of the runtime is spent in feature construction, which takes 75.07 seconds, or 71.7% of the total cost. Patch-PCA requires 22.50 seconds, while anomaly detection requires 4.08 seconds. The runtime breakdown therefore shows that the main cost comes from feature construction, not from channel selection or anomaly scoring.

The online processing results also support the lightweight behavior of the framework. The peak memory usage is 490 KB above the process baseline. The mean decision delay is 5.07 ms, and the real-time factor is 0.0142. The throughput reaches 281.4 windows/s, which means the framework processes data approximately 70 times faster than the rate at which telemetry windows are generated. Taken together, the detection performance and efficiency results indicate that UAV-

TABLE II: Ablation results of the proposed framework variants.

Variation	Patch-PCA	NICS	Accuracy	Precision	Recall	F1-score	ROC-AUC	Channels Selected	Runtime (s)
Var 1	✓	✓	0.974	0.976	0.965	0.970	0.990	9	104.75
Var 2	✓	–	0.974	0.987	0.951	0.969	0.995	32	264.32
Var 3	–	✓	0.572	0.959	0.030	0.058	0.757	1	50.65
Var 4	–	–	0.938	0.927	0.932	0.930	0.979	32	613.49

GUARD is suitable for real-time UAV monitoring under the measured setting because it achieves accurate results across different faults, while being computationally lightweight.

D. Ablation Study

Table II reports the ablation results of UAV-GUARD. Var 1 is the full configuration with both Patch-PCA and NICS, Var 2 removes NICS while keeping Patch-PCA, Var 3 keeps NICS but removes Patch-PCA, and Var 4 removes both modules.

Comparing Var 1 and Var 2 shows the effect of NICS when Patch-PCA is used. The full framework selects 9 out of 32 channels, corresponding to a 71.9% reduction in the input channel space. This reduction is achieved while keeping the detection results close to the all-channel variant. Var 1 achieves an accuracy of 0.974, precision of 0.976, F1-score of 0.970, and ROC-AUC of 0.990. Var 2 obtains the same accuracy (0.974) and a close F1-score (0.969), but requires all 32 channels. Thus, NICS removes a large part of the input space with only a small change in detection quality.

The runtime difference between Var 1 and Var 2 shows why this reduction is important. Removing NICS increases runtime from 104.75 s to 264.32 s, although the detection scores remain close. The all-channel variant therefore requires more than twice the runtime for only marginal changes in detection metrics. This makes NICS useful mainly as an efficiency module: it reduces the number of channels passed to the later stages without degrading the final operating performance.

The role of Patch-PCA is shown by Var 3 and Var 4. When Patch-PCA is removed but NICS is kept, the framework becomes the fastest variant (50.65 s), but the detection performance drops sharply. The F1-score falls to 0.058, and the ROC-AUC decreases to 0.757. The high precision (0.959) is misleading in this case, because the very low F1-score means that most anomalous windows are missed. In this setting, NICS selects only one channel, which explains why it is the fastest variant, though one channel is not enough to represent the multivariate fault patterns in ALFA.

Var 4 keeps the raw multichannel representation by removing both Patch-PCA and NICS. It obtains reasonable detection performance, with accuracy 0.938, F1-score 0.930, and ROC-AUC 0.979, but it is the slowest configuration, requiring 613.49 s. Compared with Var 4, the full framework improves accuracy, F1-score, and ROC-AUC while reducing runtime by about 82.9%. This shows that Patch-PCA is not only a compression step. It also provides a representation that is more suitable for the later feature construction and anomaly-scoring stages.

TABLE III: Performance comparison of UAV-GUARD and evaluated baselines on the ALFA dataset. The best result in each column is shown in bold.

Model	**Accuracy**	**ROC-AUC**	**Runtime (s)**
UAV-GUARD (ours)	**0.974**	**0.990**	104.75
MeNU	0.751	0.806	**42.53**
OmniAnomaly	0.633	0.744	830.42
LSTM-AE	0.610	0.674	78.87
DTW	0.583	0.643	361.84
LVA-SP	0.458	0.450	563.84

Overall, the ablation results show that the best configuration is obtained when both Patch-PCA and NICS are used.

E. Comparison with Baselines

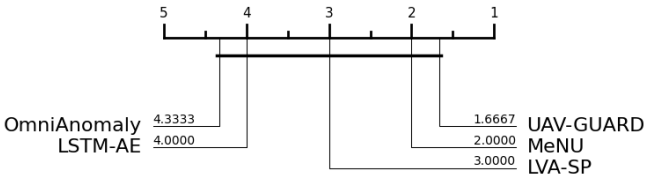


Fig. 8: Critical-difference diagram comparing the evaluated methods based on their average ranks.

Table III compares UAV-GUARD with the evaluated baselines on the ALFA dataset. UAV-GUARD achieves the best detection performance, with an accuracy of 0.974 and a ROC-AUC of 0.990. The closest baseline is MeNU, which obtains an accuracy of 0.751 and a ROC-AUC of 0.806. OmniAnomaly, LSTM-AE, DTW, and LVA-SP obtain lower scores, showing weaker separation between benign and anomalous flight windows.

The runtime results show a different trade-off. MeNU and LSTM-AE are faster than UAV-GUARD, requiring 42.53 s and 78.87 s, respectively, compared with 104.75 s for UAV-GUARD. However, their detection performance is substantially lower. UAV-GUARD is also much faster than OmniAnomaly, DTW, and LVA-SP, which require 830.42 s, 361.84 s, and 563.84 s, respectively. Therefore, UAV-GUARD provides the strongest detection quality while keeping runtime within a practical range.

Figure 8 reports the critical-difference ranking of the evaluated methods. UAV-GUARD obtains the best average rank,

followed by MeNU and LVA-SP. OmniAnomaly and LSTM-AE rank lower. This ranking is consistent with the table results, where UAV-GUARD gives the strongest overall performance across the evaluated criteria.

V. Conclusion

This paper presented UAV-GUARD, a lightweight unsupervised framework for anomaly detection in multivariate UAV telemetry. The framework combines Patch-PCA representation, NICS channel selection, Catch24 feature extraction, and KNN-KDE anomaly detection. Patch-PCA reduces local noise and temporal redundancy, NICS removes weak or redundant channels without fault labels, and KNN-KDE assigns anomaly scores with an automatically selected threshold. Experiments on the ALFA dataset showed strong detection performance while maintaining low memory usage, low decision delay, and high throughput. The ablation study further showed that Patch-PCA and NICS provide notable efficiency gains without affecting detection quality.

Future work will focus on further improving the efficiency and robustness of UAV-GUARD. Since feature construction is the main runtime component, we will study which Catch24 features are most useful and design UAV-specific statistical descriptors to improve the efficiency aspect of the framework. We also plan to extend the online evaluation to more realistic simulated mission settings to assess robustness under a wider range of operating conditions, such as degraded communication.

References

[1] I. Kabashkin and et al., "Synchronized multi-point uav-based traffic monitoring for urban infrastructure decision support," *Drones*, vol. 9, no. 5, p. 370, 2025.

[2] Y. Wang, I. W. H. Ho, and Y. Wang, "Real-time intersection vehicle turning movement counts from live uav video stream using multiple object tracking," *Journal of Intelligent and Connected Vehicles*, vol. 6, no. 3, pp. 149–160, 2023.

[3] N. Ahmadian, G. J. Lima, M. Torabbeigi, and S. J. Kim, "Smart border patrol using drones and wireless charging system under budget limitation," *Computers & Industrial Engineering*, vol. 165, p. 107891, 2022.

[4] R. Zhang, Z. Wang, X. Li, Z. She, and B. Wang, "Water quality sampling and multi-parameter monitoring system based on multi-rotor uav," *Water*, vol. 15, no. 11, p. 2129, 2023.

[5] D. K. Singh and V. Hoskere, "Post disaster damage assessment using ultra-high-resolution aerial imagery with semi-supervised transformers," *Sensors*, vol. 23, no. 19, p. 8235, 2023.

[6] M. B. Bagaram, D. Giuliarelli, G. Chirici, F. Giannetti, and A. Barbati, "Uav remote sensing for biodiversity monitoring: Are forest canopy gaps good covariates?" *Remote Sensing*, vol. 10, no. 9, p. 1397, 2018.

[7] A. Ghasemi, A. Ghaffari, N. Derakhshanfard, N. iBRAHIMOĞLU, and A. pakmehr, "Anomaly detection in unmanned aerial vehicles flight data: A survey," *Ad Hoc Networks*, vol. 178, p. 103989, 2025.

[8] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines," *IEEE Access*, vol. 9, pp. 120 043–120 065, 2021.

[9] R. Puchalski and W. Giernacki, "Uav fault detection methods, state-of-the-art," *Drones*, vol. 6, no. 11, p. 330, 2022.

[10] K. H. Park, E. Park, and H. K. Kim, "Unsupervised fault detection on unmanned aerial vehicles: Encoding and thresholding approach," *Sensors*, vol. 21, no. 6, p. 2208, 2021.

[11] J. D. Yoo, G. M. Kim, M. G. Song, and H. K. Kim, "Menu: Memorizing normality for uav anomaly detection with a few sensor values," *Computers & Security*, vol. 150, p. 104248, 2025.

[12] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM computing surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[13] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data mining and knowledge discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[14] H. Chen, Y. Lyu, J. Shi, and W. Zhang, "Uav anomaly detection method based on convolutional autoencoder and support vector data description with 0/1 soft-margin loss," *Drones*, vol. 8, no. 10, 2024.

[15] V. Bell, I. Moral Arce, J. M. Mase, D. Rengasamy, B. Rothwell, and G. P. Figueredo, "Anomaly detection for unmanned aerial vehicle sensor data using a stacked recurrent autoencoder method with dynamic thresholding," *SAE International Journal of Aerospace*, vol. 15, no. 2, pp. 219–229, Sep. 2022. [Online]. Available: https://doi.org/10.4271/01-15-02-0017

[16] C. Feng, J. Fan, G. Jin, J. Liu, and S. Chen, "Ste-rs: A novel framework for robust anomaly detection in uav flight data," *Results in Engineering*, p. 106801, 2025.

[17] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2828–2837.

[18] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Usad: Unsupervised anomaly detection on multivariate time series," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3395–3404.

[19] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International conference on learning representations*, 2018.

[20] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 5, 2021, pp. 4027–4035.

[21] B. Dhariyal, T. Le Nguyen, and G. Ifrim, "Scalable classifier-agnostic channel selection for multivariate time series classification," *Data Mining and Knowledge Discovery*, vol. 37, no. 2, pp. 1010–1054, 2023.

[22] L. Nuyts, L. Perini, and J. Davis, "Tselect: selecting relevant and non-redundant channels for multivariate time series classification," *Data Mining and Knowledge Discovery*, vol. 39, no. 6, p. 76, 2025.

[23] A. Dempster, F. Petitjean, and G. I. Webb, "Rocket: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, p. 1454–1495, Jul. 2020.

[24] C. H. Lubba, S. S. Sethi, P. Knaute, T. Tiumentsev, B. D. Wang, T. P. Scholzel, N. Y. Hammerla, T. Ploeg, and N. S. Jones, "catch22: Canonical time-series characteristics," *Data Mining and Knowledge Discovery*, vol. 33, no. 6, pp. 1821–1852, 2019.

[25] Y. Ren, K. Feng, F. Hu, L. Chen, and Y. Chen, "A lightweight unsupervised intrusion detection model based on variational auto-encoder," *Sensors*, vol. 23, no. 20, p. 8407, 2023.