Introduction

Robotic systems are increasingly vulnerable to cyberattacks (e.g., DoS attacks) and hardware malfunctions that can cause operational failures and safety risks. Traditional rule-based anomaly detection systems struggle with sophisticated threats and subtle failures in dynamic environments.

This project develops an Explainable AI (XAI)-driven anomaly detection system for robotic telemetry data that achieves accurate classification while providing transparent, interpretable explanations for predictions—enabling operators to validate decisions and diagnose root causes.

Problem Definition

Objective

1. Develop ML models for multi-class anomaly detection:
   - Normal Operation: Expected robot behavior
   - DoS Attack: Communication disruptions (packet loss, sequence gaps)
   - Malfunction: Hardware/sensor failures (battery, IMU, orientation issues)
   - Implement XAI techniques: Permutation Importance, SHAP, LIME, PDP, Correlation Analysis

2. Compare models across traditional ML, ensemble methods, deep learning, and unsupervised learning

Dataset

- 52 features: Header data, position, orientation, velocity, IMU, communication metrics (RSSI), battery
- 3 classes: Normal, DoS_Attack, Malfunction
- Challenges: Class imbalance, high dimensionality, temporal dependencies, non-linear relationships

Success Criteria

- Accuracy ≥ 90%, F1-scores ≥ 0.85 per class
- Explainable predictions aligned with robotics domain knowledge
- Inference time ≤ 100ms per sample

PART 1: DATA PREPROCESSING

1.1 Data Loading and Initial Exploration

- Load the dataset and examine its structure

```
Dataset Dimensions (Rows, Columns): (87417, 80)
```

- Check data types and identify the target variable(s)

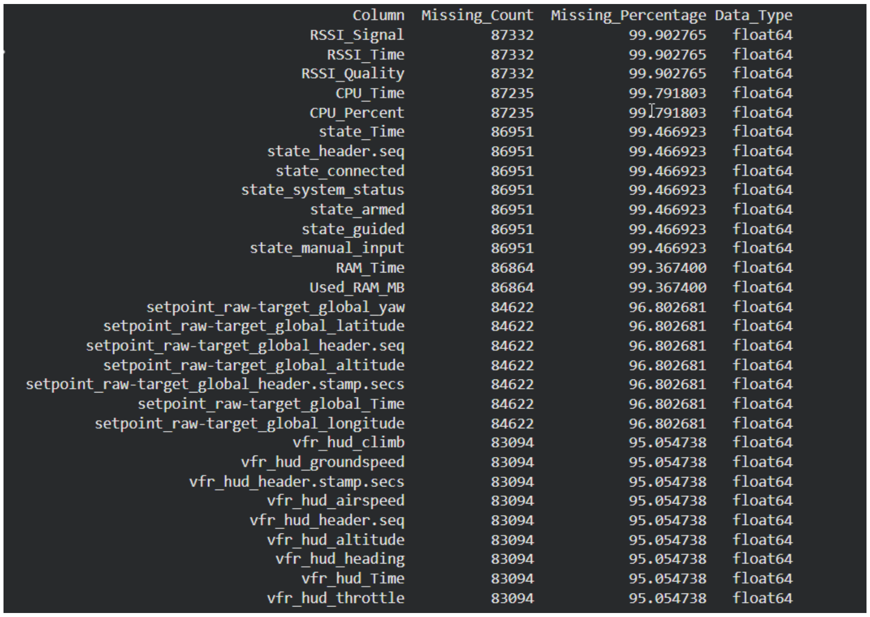
```
Data Type Distribution:
float64    76
int64       4
object      1

Target Variable Identified: Class
```

- Document dataset dimensions and any initial observations

- The dataset combines data reported at different frequencies. Only a few features, like those related to setpoint_raw-global and S.No, are fully populated (87,417 non-null values).
- The majority of key sensor groups (Battery, IMU, VFR HUD, RC Out, Global Position) are highly sparse, reporting only around 4,300 to 4,600 non-null values out of 87,417 rows (approximately 5% data availability).

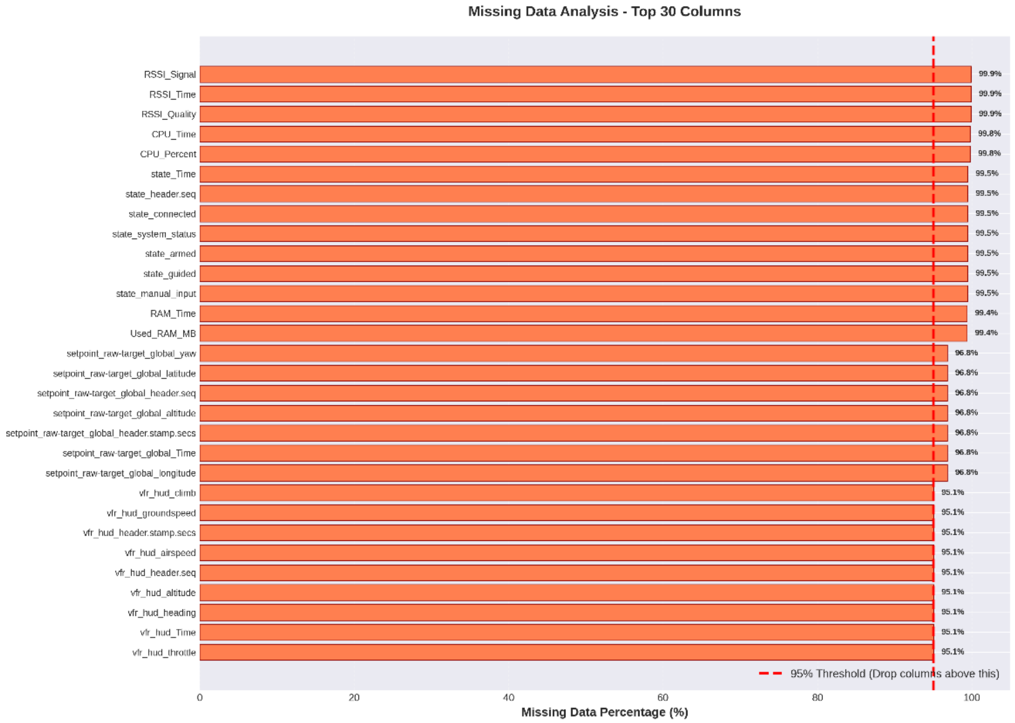
1.2 Missing Data Analysis

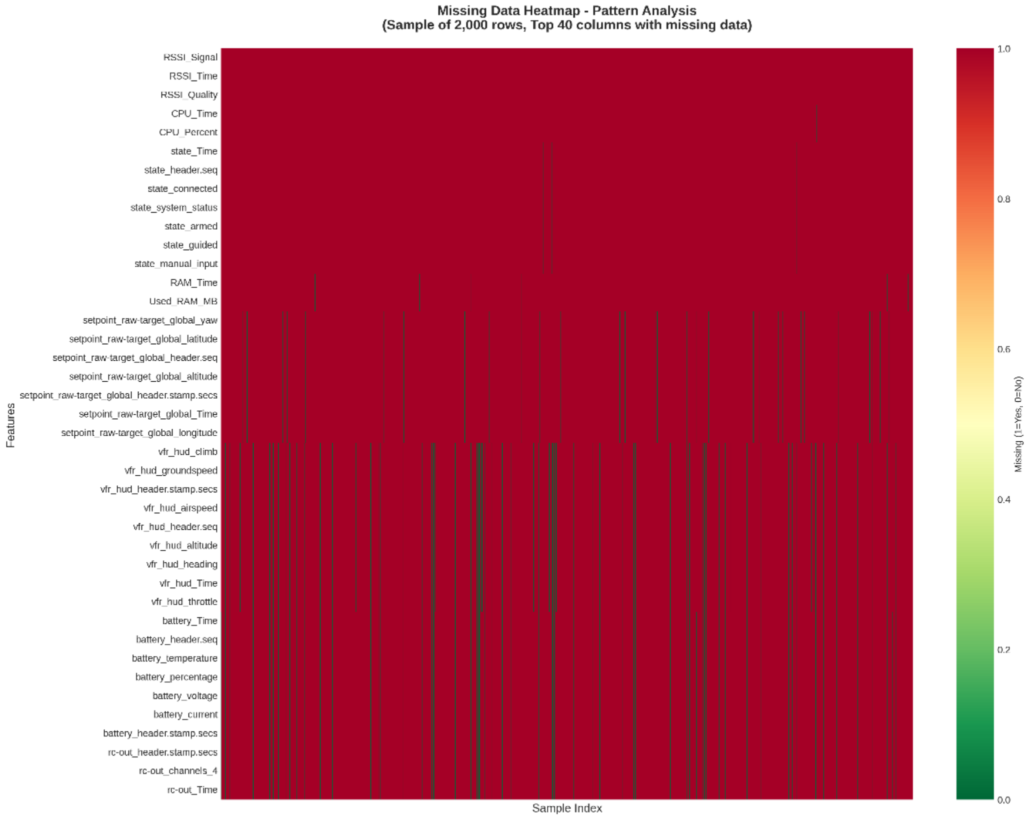
- Identify missing values in each column



```
                                      Column  Missing_Count  Missing_Percentage Data_Type
                                 RSSI_Signal          87332           99.902765   float64
                                   RSSI_Time          87332           99.902765   float64
                                RSSI_Quality          87332           99.902765   float64
                                    CPU_Time          87235           99.791803   float64
                                 CPU_Percent          87235           99.791803   float64
                                  state_Time          86951           99.466923   float64
                            state_header.seq          86951           99.466923   float64
                             state_connected          86951           99.466923   float64
                         state_system_status          86951           99.466923   float64
                                 state_armed          86951           99.466923   float64
                                state_guided          86951           99.466923   float64
                          state_manual_input          86951           99.466923   float64
                                    RAM_Time          86864           99.367400   float64
                                 Used_RAM_MB          86864           99.367400   float64
              setpoint_raw-target_global_yaw          84622           96.802681   float64
         setpoint_raw-target_global_latitude          84622           96.802681   float64
       setpoint_raw-target_global_header.seq          84622           96.802681   float64
         setpoint_raw-target_global_altitude          84622           96.802681   float64
setpoint_raw-target_global_header.stamp.secs          84622           96.802681   float64
             setpoint_raw-target_global_Time          84622           96.802681   float64
        setpoint_raw-target_global_longitude          84622           96.802681   float64
                               vfr_hud_climb          83094           95.054738   float64
                         vfr_hud_groundspeed          83094           95.054738   float64
                   vfr_hud_header.stamp.secs          83094           95.054738   float64
                            vfr_hud_airspeed          83094           95.054738   float64
                          vfr_hud_header.seq          83094           95.054738   float64
                            vfr_hud_altitude          83094           95.054738   float64
                             vfr_hud_heading          83094           95.054738   float64
                                vfr_hud_Time          83094           95.054738   float64
                            vfr_hud_throttle          83094           95.054738   float64
```

- Visualize missing data patterns (heatmap, bar chart)





- Decide on appropriate strategies: imputation, deletion, or interpolation

The missing data was handled using two strategies: imputation and deletion

- Document your rationale for handling missing data

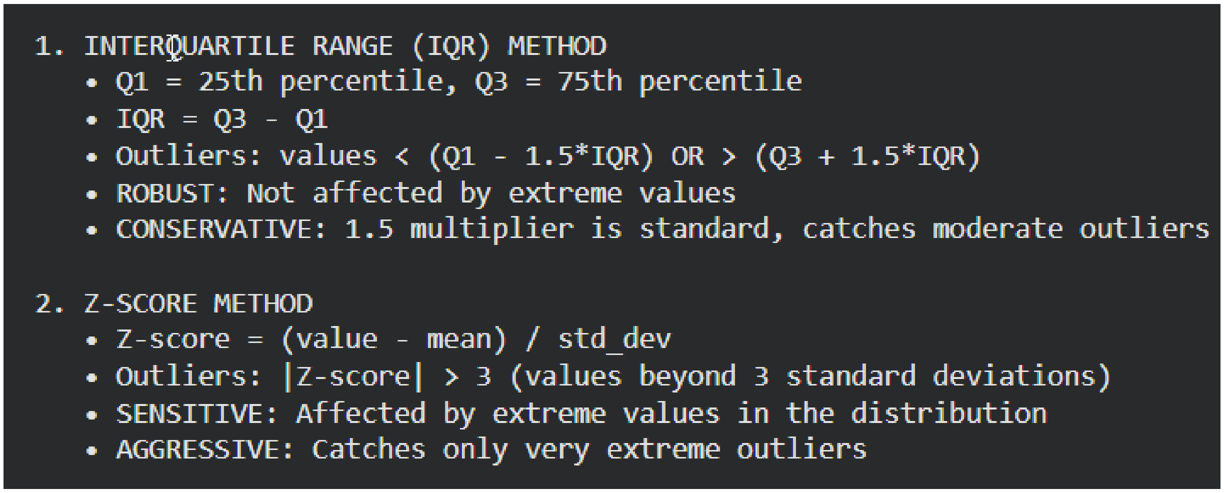
First, any column that has more than 95% missing values will be removed because such columns contain almost no useful information and only add noise. This threshold is very conservative, meaning we keep as many features as possible while removing only those that are essentially empty.

Next, for numerical features, missing values will be filled using the median. This is more reliable than the mean because robot sensor data often contains extreme error spikes that can distort averages, while the median stays close to typical operating values.

Finally, for categorical features, missing values will be filled using the mode, which is the most common value. This works best for system states and other non-numeric data because it preserves realistic patterns and distributions within the dataset.

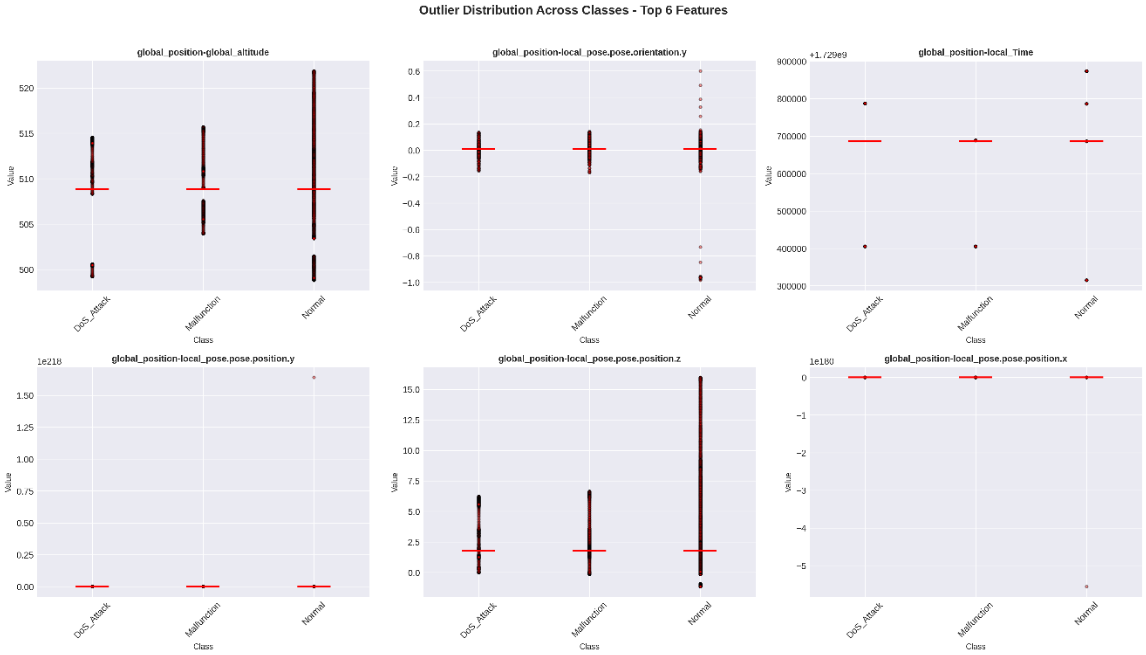
1.3 Outlier Detection and Treatment

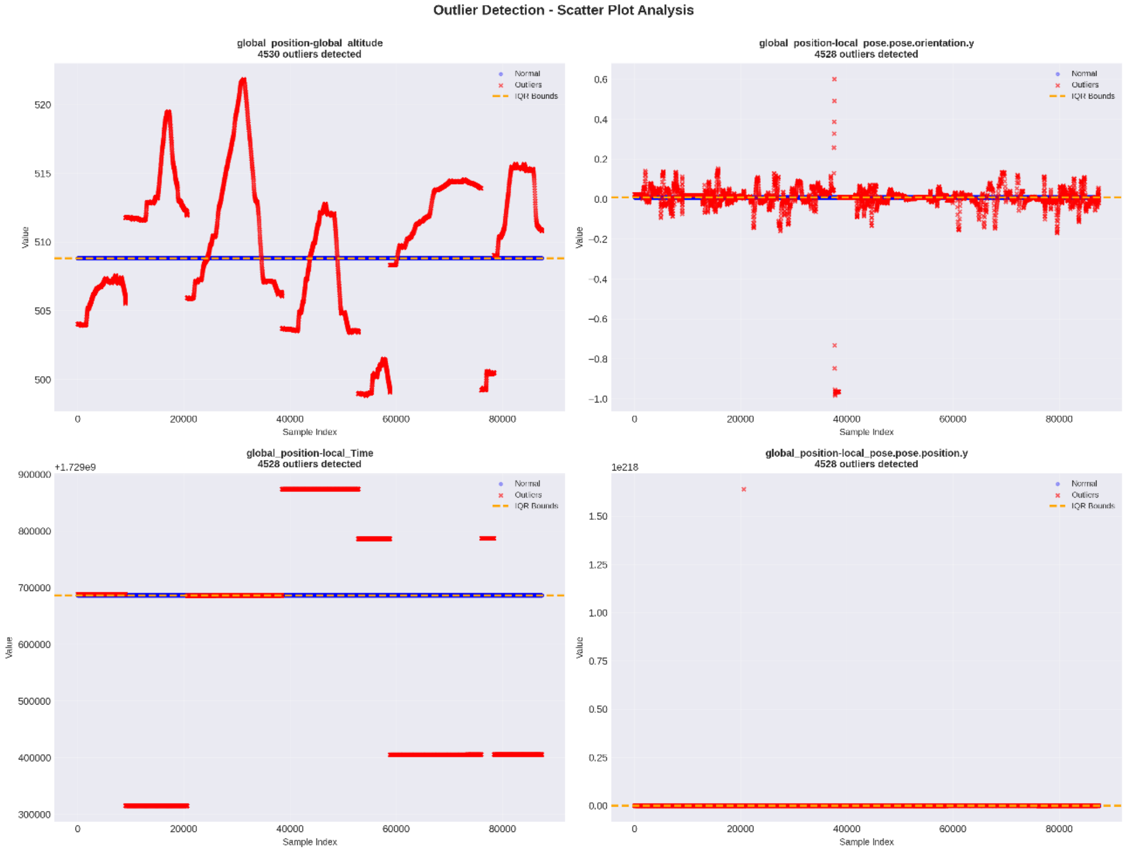
- Use statistical methods (IQR, Z-score) to identify outliers



```
1. INTERQUARTILE RANGE (IQR) METHOD
   • Q1 = 25th percentile, Q3 = 75th percentile
   • IQR = Q3 - Q1
   • Outliers: values < (Q1 - 1.5*IQR) OR > (Q3 + 1.5*IQR)
   • ROBUST: Not affected by extreme values
   • CONSERVATIVE: 1.5 multiplier is standard, catches moderate outliers

2. Z-SCORE METHOD
   • Z-score = (value - mean) / std_dev
   • Outliers: |Z-score| > 3 (values beyond 3 standard deviations)
   • SENSITIVE: Affected by extreme values in the distribution
   • AGGRESSIVE: Catches only very extreme outliers
```

- Create box plots and scatter plots to visualize outliers





- Decide whether to remove, cap, or transform outliers

CAPPING METHOD (WINSORIZATION)

- Document your approach and reasoning

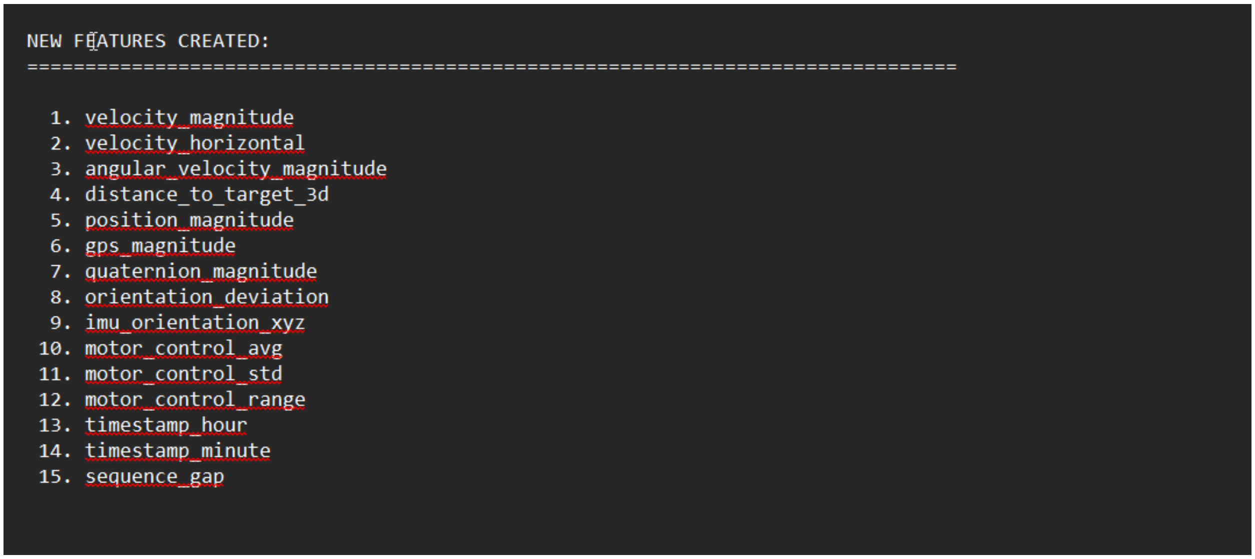
Capping (Winsorization) was applied instead of removing outliers to maintain data volume and preserve important information in the robot telemetry data. Since each sample contains valuable temporal behavior, removing outliers could eliminate entire sequences of normal, attack, or malfunction activity. These outliers act as informative signals rather than noise, especially in a classification problem where distinguishing normal from abnormal behavior is essential.

Capping also supports model robustness: tree-based methods like XGBoost naturally tolerate outliers, and neural networks benefit from controlled value ranges. Using the IQR-based Winsorization method, values are capped at Q1 − 1.5×IQR and Q3 + 1.5×IQR, preserving the overall distribution while limiting extreme points and retaining all samples.

Alternatives such as outright removal or transformations (log/Box-Cox) were rejected because they would either discard critical events, reduce dataset size, complicate interpretability, or fail to address multimodal feature distributions.

1.4 Feature Engineering

- Create derived features if necessary (e.g., velocity magnitude, distance to target, battery drain rate)



```
NEW FEATURES CREATED:
================================================================================

  1. velocity_magnitude
  2. velocity_horizontal
  3. angular_velocity_magnitude
  4. distance_to_target_3d
  5. position_magnitude
  6. gps_magnitude
  7. quaternion_magnitude
  8. orientation_deviation
  9. imu_orientation_xyz
 10. motor_control_avg
 11. motor_control_std
 12. motor_control_range
 13. timestamp_hour
 14. timestamp_minute
 15. sequence_gap
```

- Consider domain knowledge for feature creation

- Raw sensor readings alone may not capture complex patterns
- Derived features can reveal relationships invisible to models
- Domain-specific features improve interpretability
- May reduce feature dimensionality while increasing information
- Critical for distinguishing Normal vs DoS vs Malfunction

1.5 Data Normalization/Standardization

- Apply appropriate scaling techniques (StandardScaler, MinMaxScaler, etc.)

STANDARDSCALER (Z-Score) technique is applied.

- Explain why you chose specific scaling methods for different features

The StandardScaler (Z-Score) was selected as the optimal scaling method primarily to enhance model performance and interpretability across all six required models. By centering the data at zero with unit variance, the technique ensures faster convergence and improved performance for neural networks (FNN, LSTM, CNN, VAE) by normalizing their gradients. It is also essential for SVM compatibility, preventing features with naturally large scales from biasing distance metrics used by the RBF kernel.

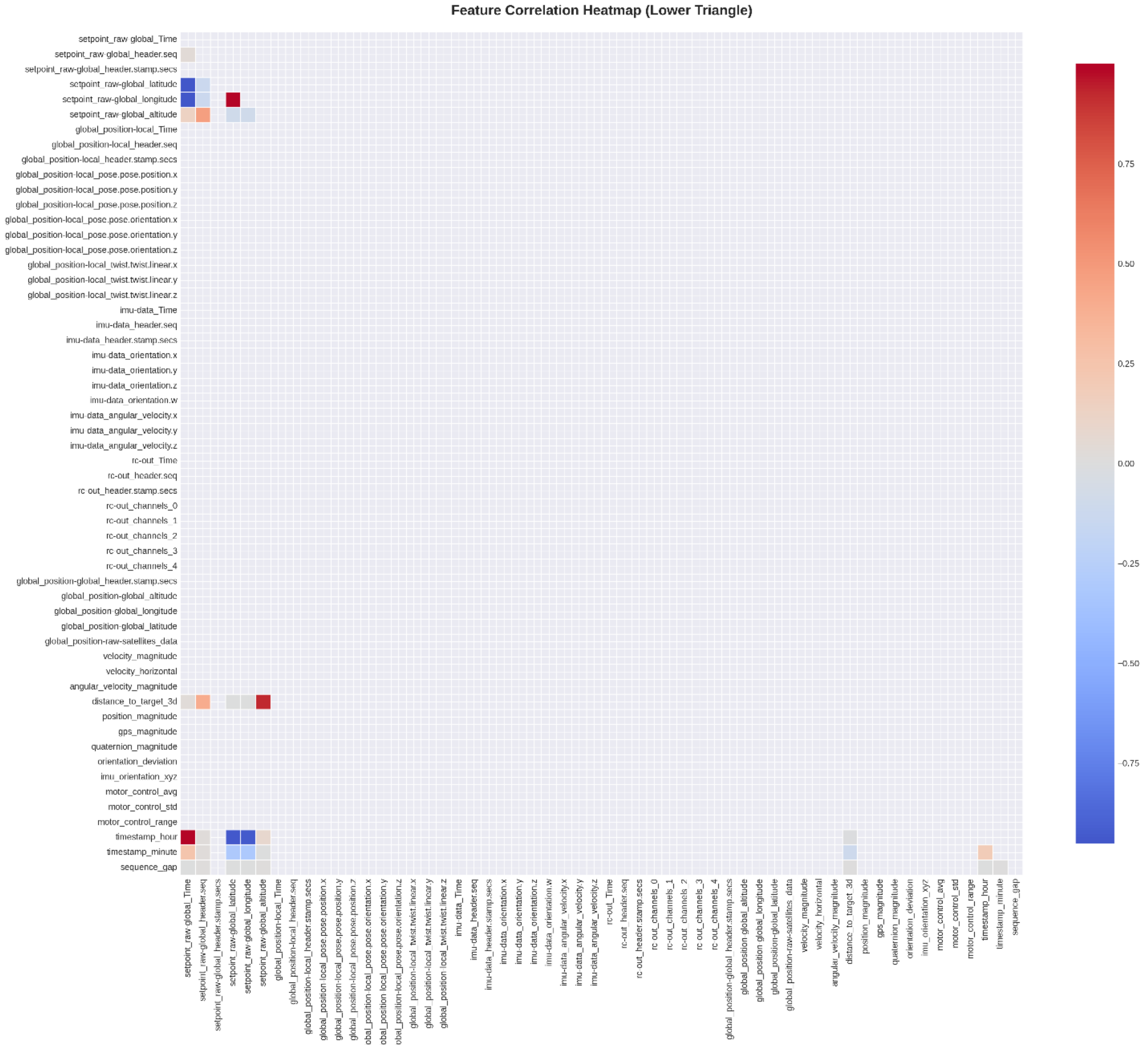
Given that outliers were already capped in Part 1.3, the data is sufficiently clean, making StandardScaler a better choice than the highly outlier-sensitive MinMaxScaler or the now redundant RobustScaler. Crucially, Standard Scaling maintains the natural distribution and skewness of the features and improves XAI interpretability by allowing domain experts to understand values in terms of "standard deviations from the mean," thereby enabling a more meaningful analysis of both feature and SHAP values.

1.6 Feature Correlation Analysis

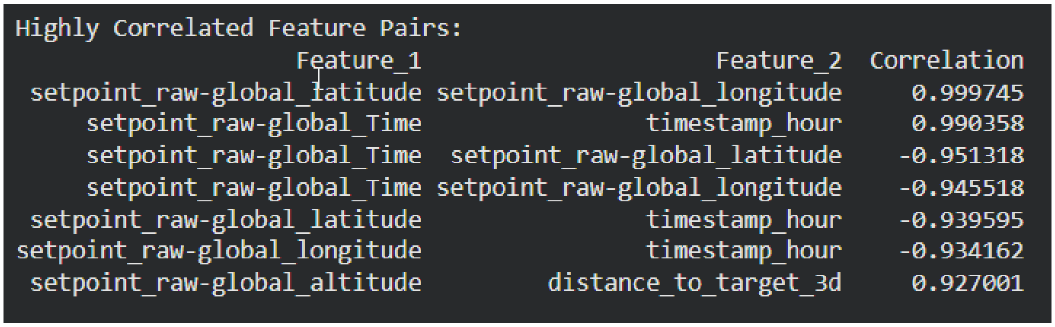
- Compute correlation matrix

Done in code.

- Create correlation heatmap

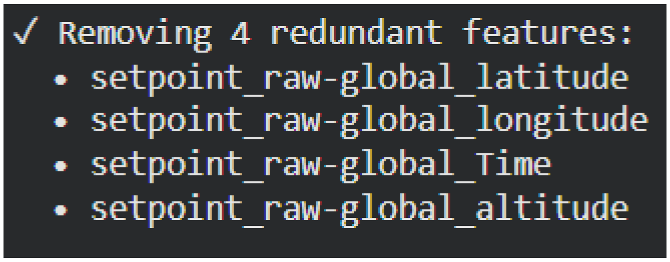


- Identify highly correlated features



```
Highly Correlated Feature Pairs:
                    Feature_1                       Feature_2  Correlation
 setpoint_raw-global_latitude setpoint_raw-global_longitude     0.999745
     setpoint_raw-global_Time                 timestamp_hour     0.990358
     setpoint_raw-global_Time  setpoint_raw-global_latitude    -0.951318
     setpoint_raw-global_Time setpoint_raw-global_longitude    -0.945518
 setpoint_raw-global_latitude                 timestamp_hour    -0.939595
setpoint_raw-global_longitude                 timestamp_hour    -0.934162
 setpoint_raw-global_altitude           distance_to_target_3d     0.927001
```

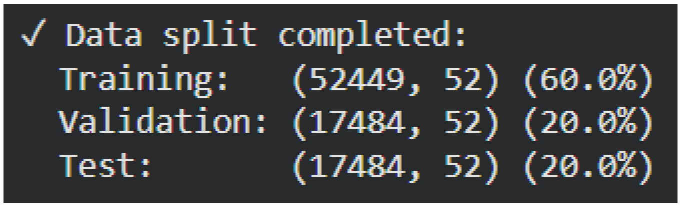
- Consider removing redundant features



```
✓ Removing 4 redundant features:
  • setpoint_raw-global_latitude
  • setpoint_raw-global_longitude
  • setpoint_raw-global_Time
  • setpoint_raw-global_altitude
```

1.7 Data Splitting

- Split data into training, validation, and test sets

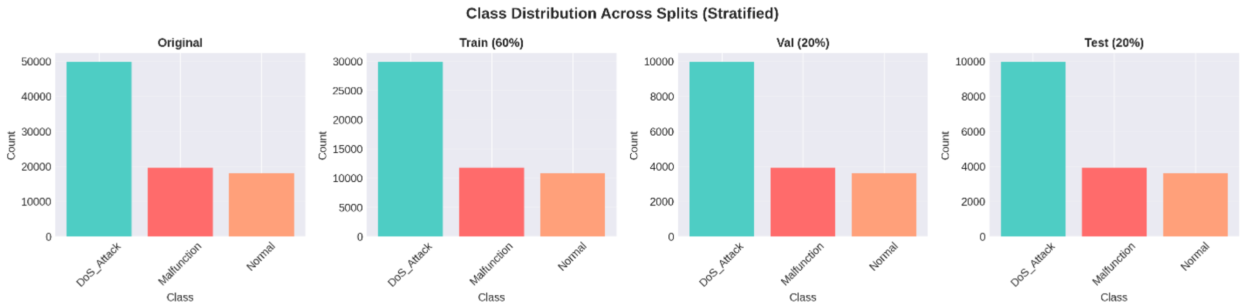


```
✓ Data split completed:
  Training:   (52449, 52) (60.0%)
  Validation: (17484, 52) (20.0%)
  Test:       (17484, 52) (20.0%)
```

- Document split ratios (e.g., 70-15-15 or 60-20-20)

The chosen data splitting strategy involves allocating 60% of the data to the Training set for model learning, 20% to the Validation set for hyperparameter tuning and model selection, and the remaining 20% to the Test set for a final, unbiased evaluation. This 60-20-20 split is rationalized as a balanced approach for this moderate-sized dataset (87,417 rows), ensuring the Training set is sufficiently large to learn complex patterns, while the Validation and Test sets are large enough to provide reliable and statistically significant results for both the selection phase and the final performance reporting.

- Ensure stratification if dealing with classification



PART 2: MODEL TRAINING

MODEL 1: SVM

- Search Method: GridSearchCV with 3-fold CV
- Rationale: Small optimized grid (9 configs), fast training, exhaustive search feasible
- Hyperparameters Tested:
  - C: [1, 10, 100] (3 values) - regularization
  - Kernel: [rbf] (1 value) - best for complex patterns
  - Gamma: [scale, 0.01, 0.1] (3 values) - kernel coefficient
- Best Parameters: {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
- Training Time: 32.61 minutes
- Test Accuracy: 0.9992

MODEL 2: LSTM

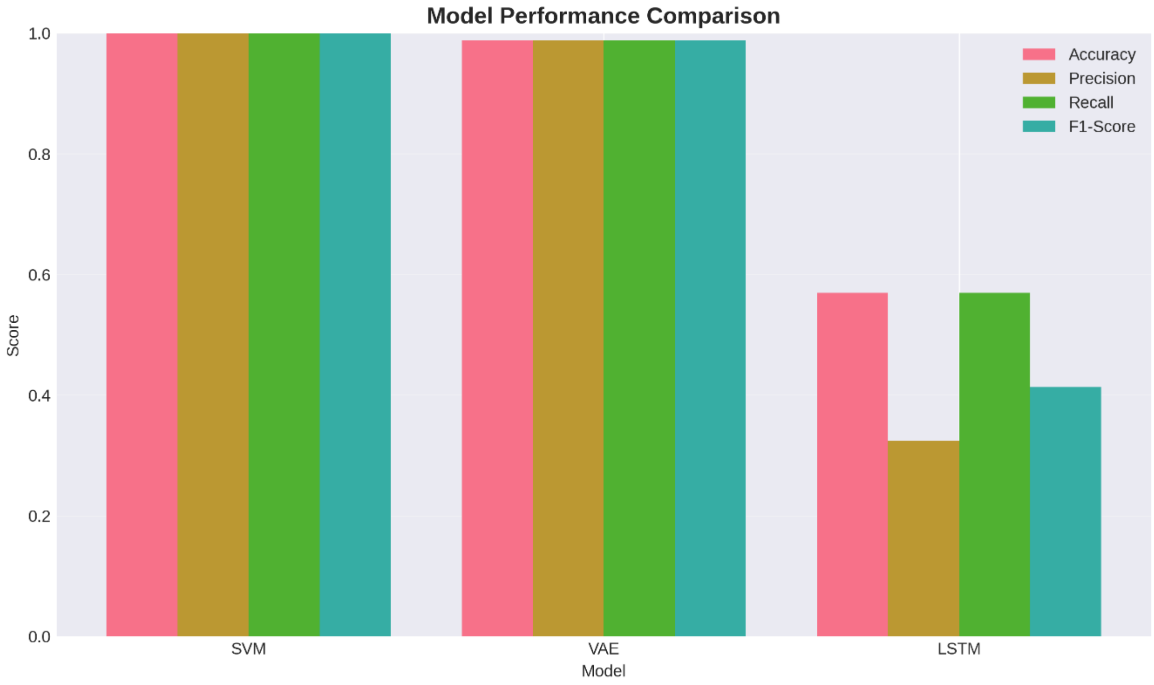
- Search Method: Manual configuration testing (2 configs)
- Rationale: Deep learning, expensive training, literature-guided selection
- Configurations Tested: 2
  - LSTM layers: [1, 2]
  - Units: [64]
  - Dropout: [0.2, 0.3]
  - Learning rate: [0.001]
  - Batch size: [64]
  - Max epochs: 50 (early stop patience=10)
  - Sequence length: 5-time steps
- Best Configuration: {'n_layers': 2, 'units': 64, 'dropout': 0.3, 'lr': 0.001, 'batch_size': 64}
- Training Time: 5.39 minutes
- Test Accuracy: 0.5697

MODEL 3: VAE

- Search Method: Single optimized configuration
- Rationale: Computationally expensive, using established best practices
- Configuration Used: 1
    - Latent dim: 32
    - Encoder layers: [256, 128]
    - Decoder layers: [128, 256]
    - Learning rate: 0.001
    - Beta (KL weight): 1.0
    - Max epochs: 50 (early stop patience=15)
- Configuration: {'latent_dim': 32, 'encoder': [256, 128], 'decoder': [128, 256], 'lr': 0.001, 'beta': 1.0}
- Training Time: 2.36 minutes
- Test Accuracy: 0.9881
- TOTAL TRAINING TIME: 40.37 minutes

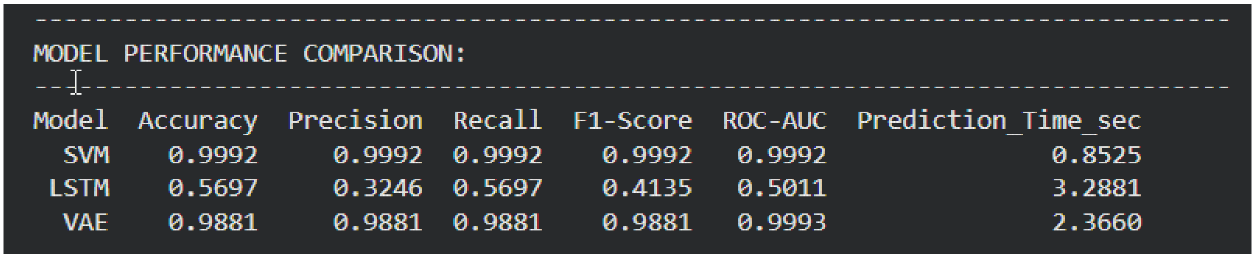
Best Model: SVM (Accuracy: 0.9992)

Total Training Time: 40.37 minutes



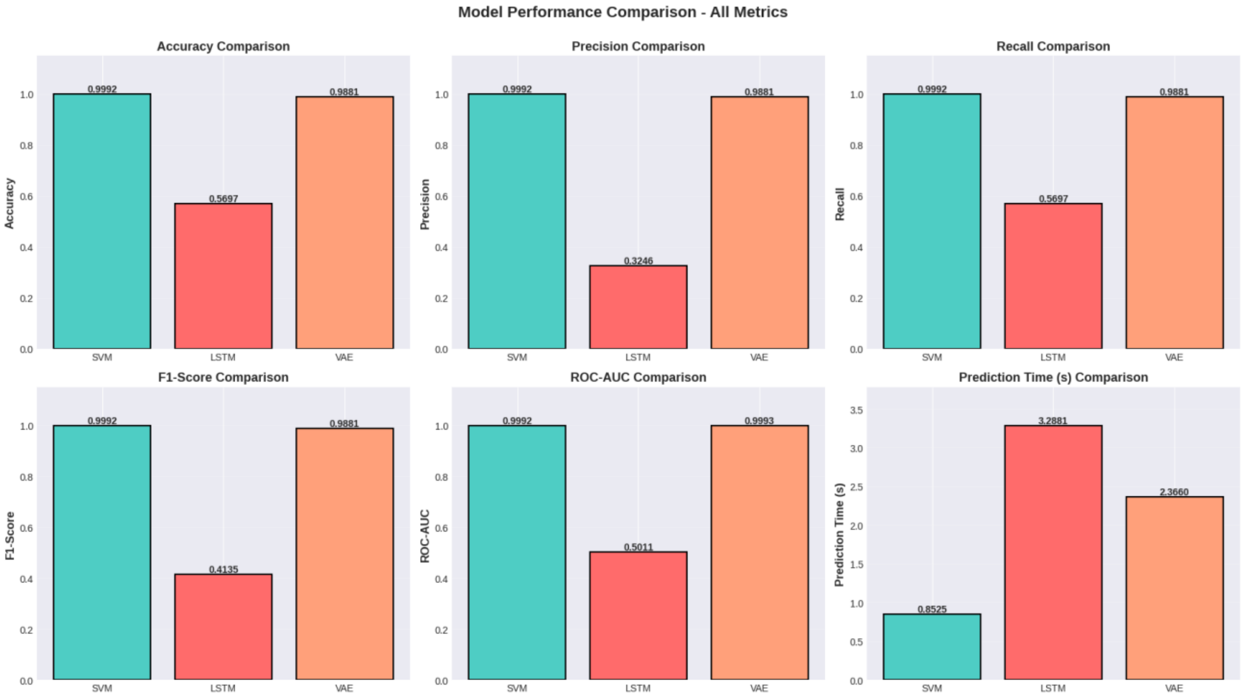
PART 3: MODEL EVALUATION

1. Performance comparison table for all 6 models

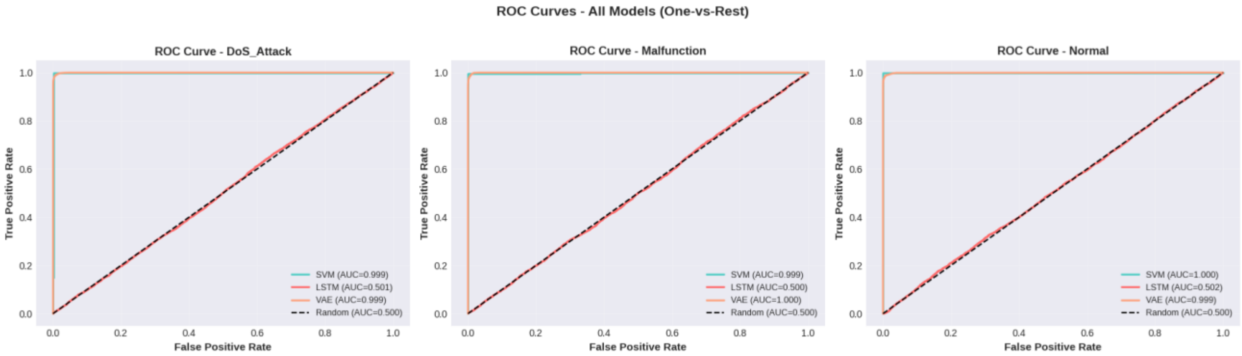


```
--------------------------------------------------------------------------------
MODEL PERFORMANCE COMPARISON:
--------------------------------------------------------------------------------
Model  Accuracy  Precision  Recall  F1-Score  ROC-AUC  Prediction_Time_sec
  SVM    0.9992     0.9992  0.9992    0.9992   0.9992               0.8525
 LSTM    0.5697     0.3246  0.5697    0.4135   0.5011               3.2881
  VAE    0.9881     0.9881  0.9881    0.9881   0.9993               2.3660
```

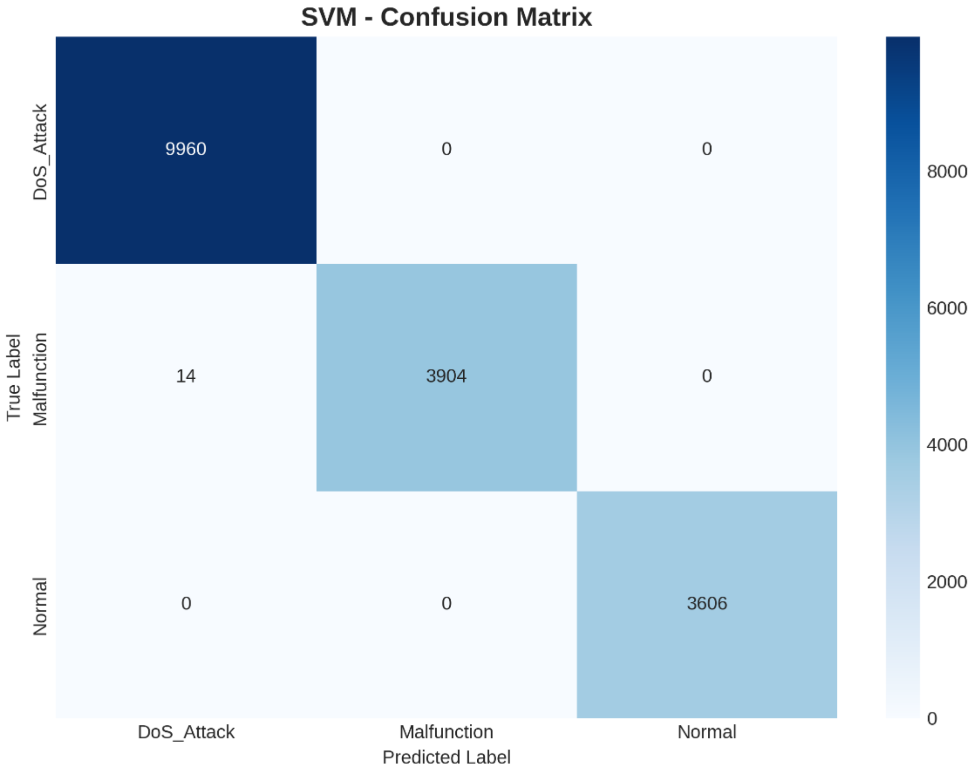
2. Bar charts comparing model performance

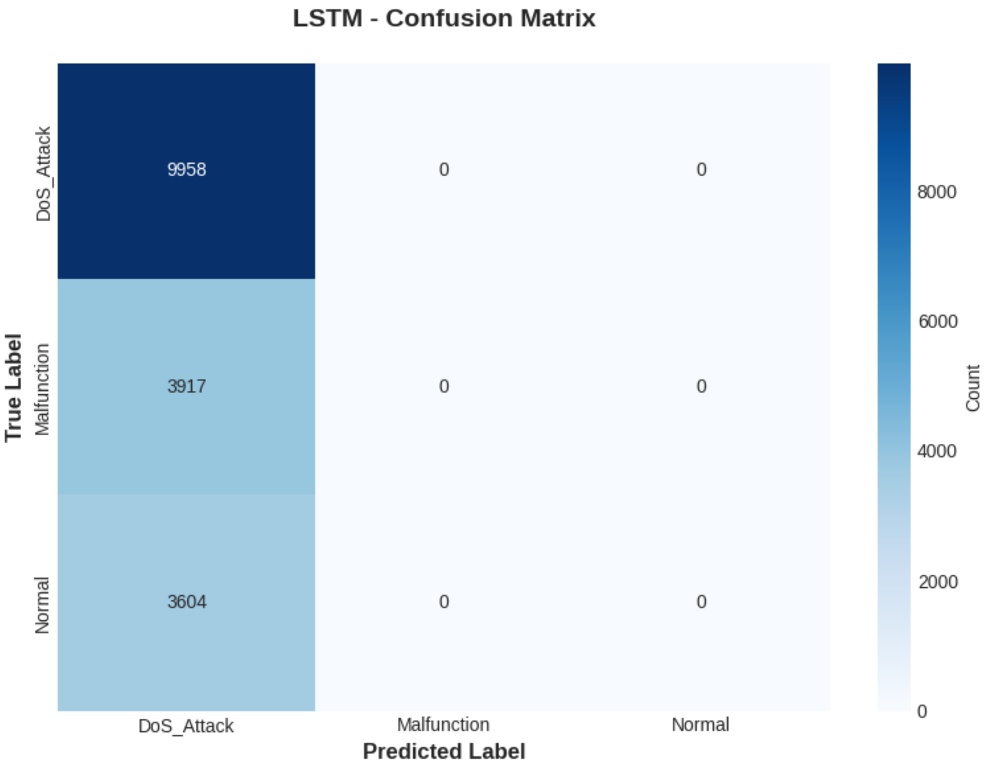


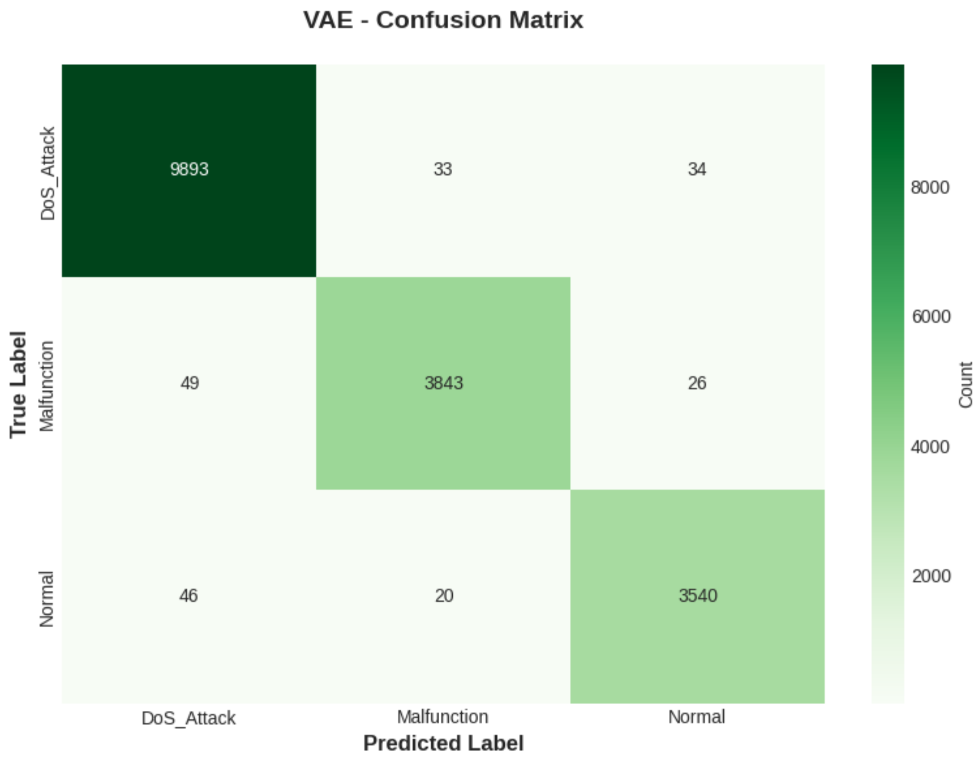
3. Learning curves (training vs validation loss)



4. Confusion matrices (if classification)







5. Discussion of which model performs best and why

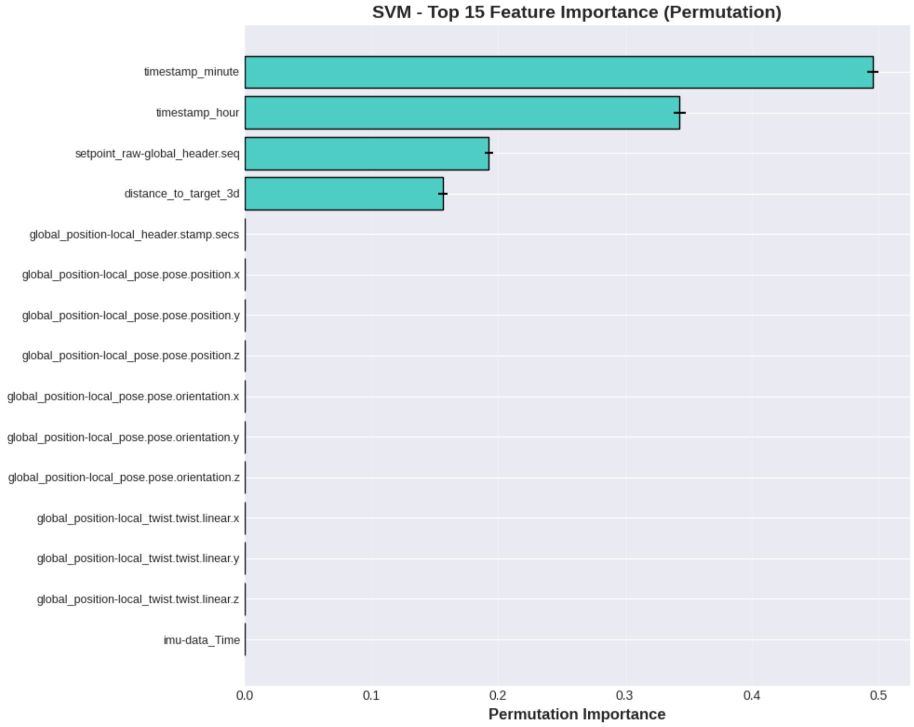
BEST MODEL BY METRIC:

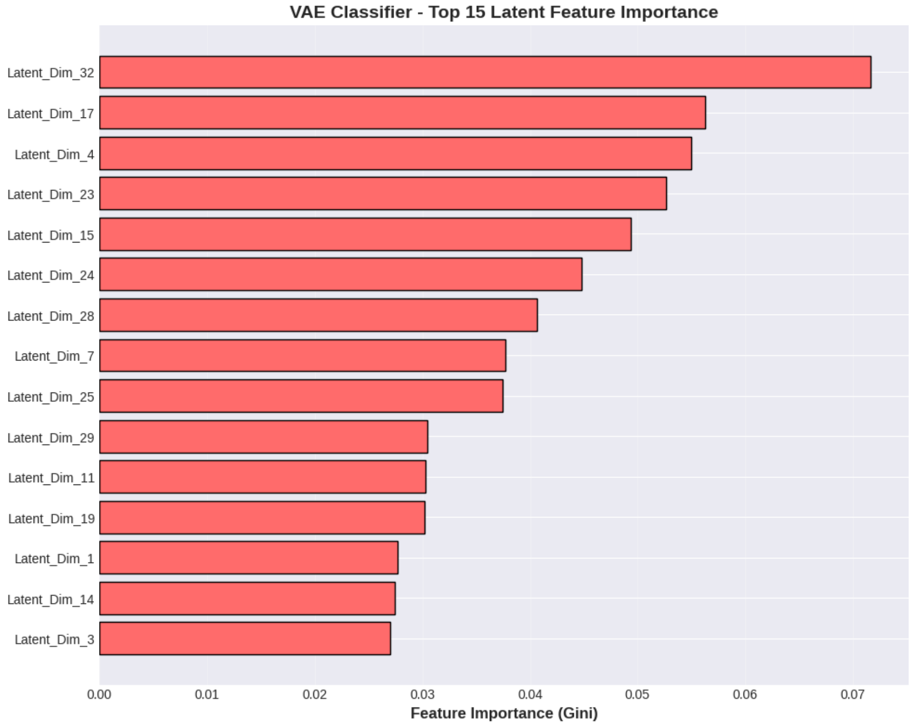
- Accuracy: SVM (0.9992)
- Precision: SVM (0.9992)
- Recall: SVM (0.9992)
- F1-Score: SVM (0.9992)
- ROC-AUC: VAE (0.9993)
- Speed: SVM (0.8525 seconds)

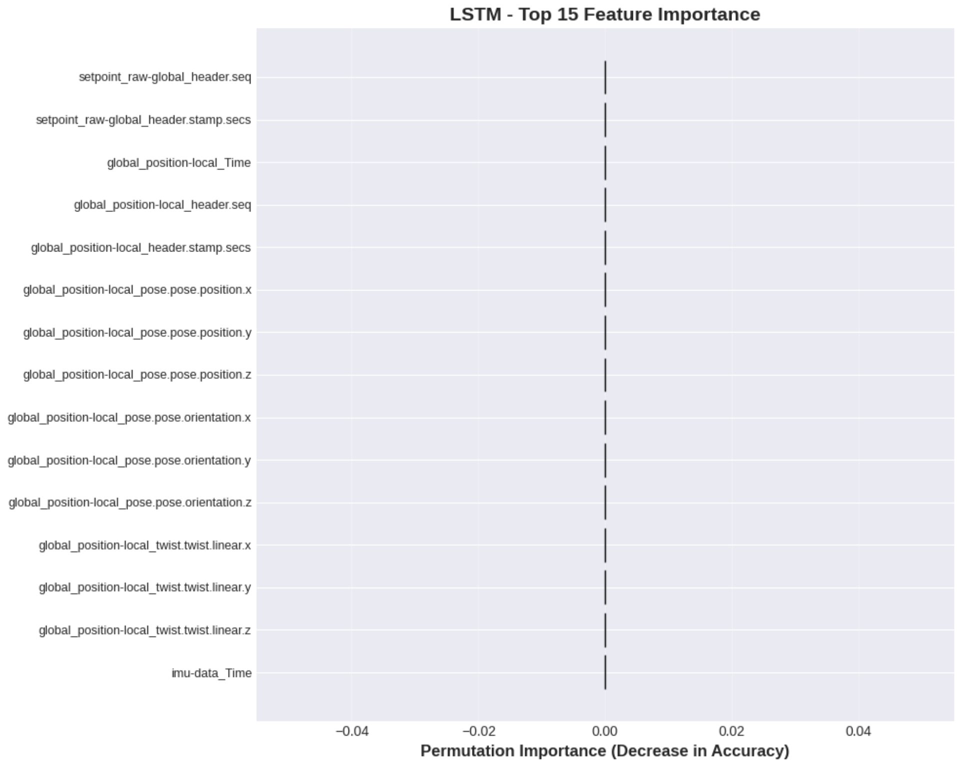
Reasoning: Based on the average of all classification metrics (Accuracy, Precision, Recall, F1-Score, ROC-AUC), SVM achieves the highest overall performance.

PART 4: EXPLAINABLE AI (XAI) ANALYSIS

4.1 FEATURE IMPORTANCE ANALYSIS







4.2 SHAP (SHapley Additive exPlanations)

- Install shap library: pip install shap
- Generate SHAP values for at least 2 models (e.g., XGBoost and FNN)
- Create SHAP summary plots (global feature importance)
- Create SHAP dependence plots for top features
- Generate SHAP force plots for individual predictions
- Generate SHAP waterfall plots

  (VISUALS FOR ALL THE ABOVE POINTS ARE ADDED IN FOLDER DIRECTORY: Visuals for part 4 ---> 4.2)

- Interpret what SHAP values reveal about feature relationships

  1. SVM Model:
     - Top 5 most important features: timestamp_minute, timestamp_hour, distance_to_target_3d...
     - SHAP values reveal non-linear relationships in feature contributions
     - Force plots show how features push predictions toward specific classes
     - Dependence plots reveal feature interactions and thresholds

  2. VAE Classifier (Random Forest):
     - Top 5 most important latent dimensions: Latent_Dim_32, Latent_Dim_4, Latent_Dim_24...

- Latent space captures compressed representations of original features
- Some latent dimensions strongly discriminate between classes
- Random Forest uses tree-based decisions on latent features

3. Feature Relationships:
   - Dependence plots show how feature values affect predictions
   - Interaction effects visible through color-coding in dependence plots
   - Non-linear relationships captured by both models

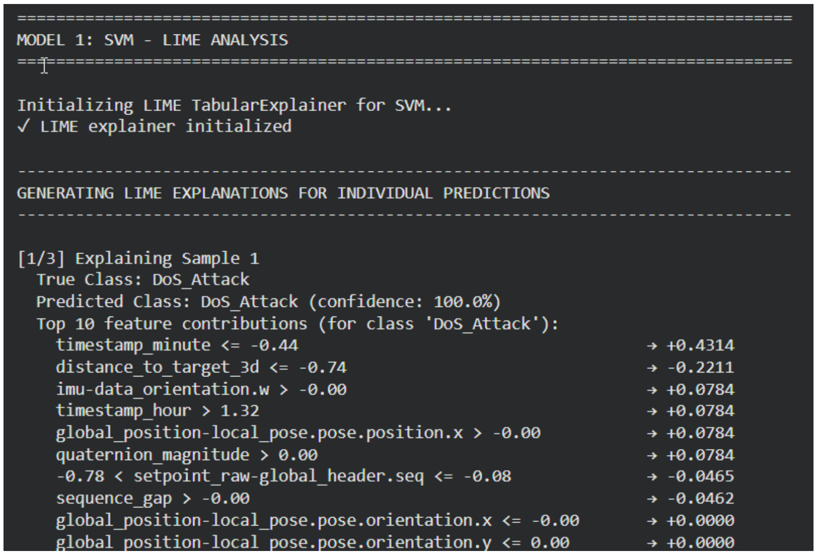
4. Individual Predictions:
   - Force plots show feature-level contributions to each prediction
   - Waterfall plots display cumulative effect of features
   - Red features push toward positive class, blue toward negative

4.3 LIME (Local Interpretable Model-agnostic Explanations)

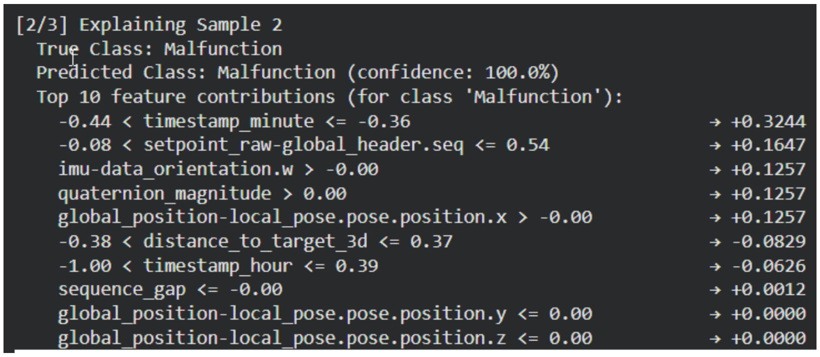
- Install lime library: pip install lime
- Apply LIME to at least 2 models

  (VISUALS ARE ADDED IN FOLDER DIRECTORY:
  Visuals for part 4 ---> 4.3)

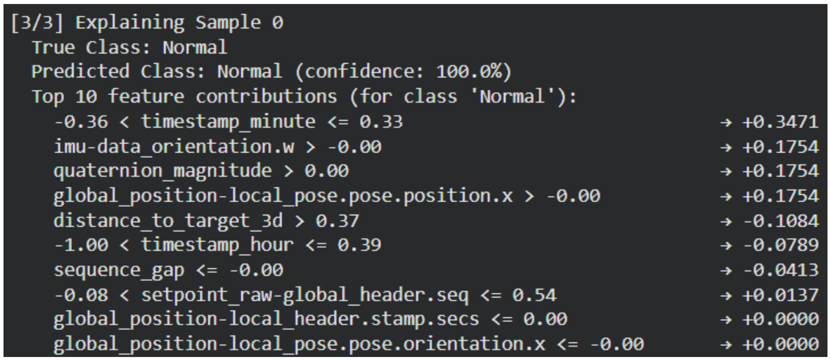
- Explain individual predictions using LIME



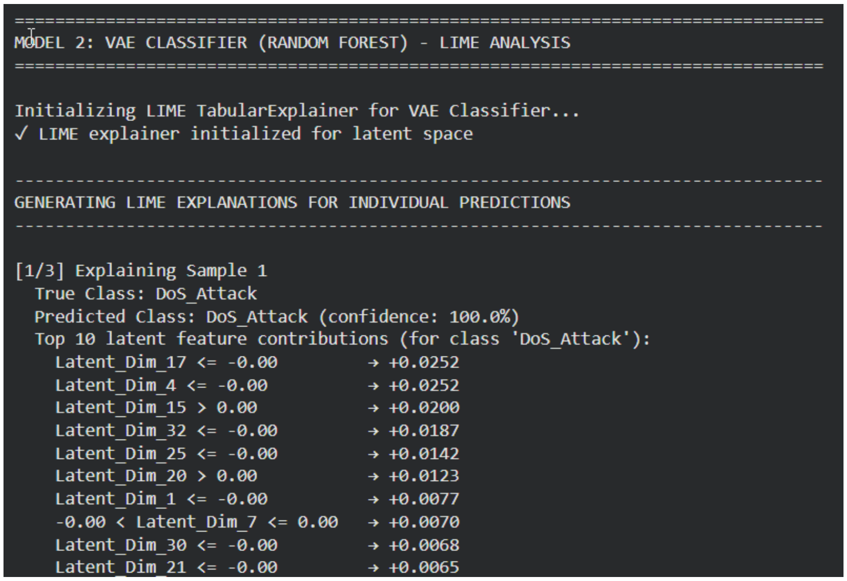
```
================================================================================
MODEL 1: SVM - LIME ANALYSIS
================================================================================

Initializing LIME TabularExplainer for SVM...
✓ LIME explainer initialized

--------------------------------------------------------------------------------
GENERATING LIME EXPLANATIONS FOR INDIVIDUAL PREDICTIONS
--------------------------------------------------------------------------------

[1/3] Explaining Sample 1
  True Class: DoS_Attack
  Predicted Class: DoS_Attack (confidence: 100.0%)
  Top 10 feature contributions (for class 'DoS_Attack'):
    timestamp_minute <= -0.44                                    → +0.4314
    distance_to_target_3d <= -0.74                               → -0.2211
    imu-data_orientation.w > -0.00                               → +0.0784
    timestamp_hour > 1.32                                        → +0.0784
    global_position-local_pose.pose.position.x > -0.00           → +0.0784
    quaternion_magnitude > 0.00                                  → +0.0784
    -0.78 < setpoint_raw-global_header.seq <= -0.08              → -0.0465
    sequence_gap > -0.00                                         → -0.0462
    global_position-local_pose.pose.orientation.x <= -0.00       → +0.0000
    global_position-local_pose.pose.orientation.y <= 0.00        → +0.0000
```



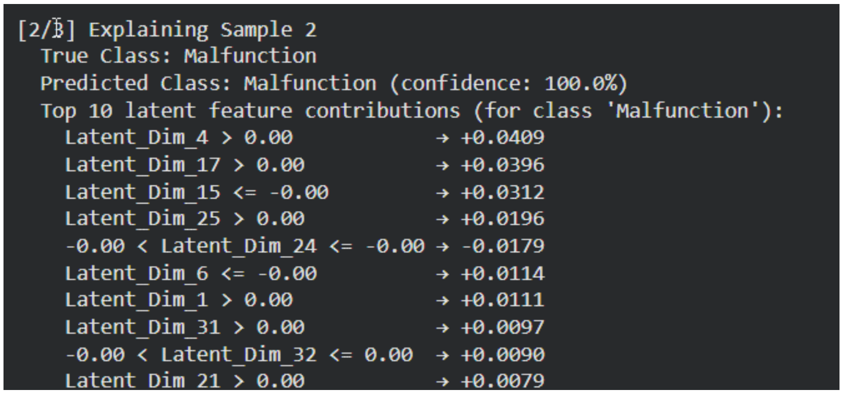
```
[2/3] Explaining Sample 2
  True Class: Malfunction
  Predicted Class: Malfunction (confidence: 100.0%)
  Top 10 feature contributions (for class 'Malfunction'):
    -0.44 < timestamp_minute <= -0.36                            → +0.3244
    -0.08 < setpoint_raw-global_header.seq <= 0.54               → +0.1647
    imu-data_orientation.w > -0.00                               → +0.1257
    quaternion_magnitude > 0.00                                  → +0.1257
    global_position-local_pose.pose.position.x > -0.00           → +0.1257
    -0.38 < distance_to_target_3d <= 0.37                        → -0.0829
    -1.00 < timestamp_hour <= 0.39                               → -0.0626
    sequence_gap <= -0.00                                        → +0.0012
    global_position-local_pose.pose.position.y <= 0.00           → +0.0000
    global_position-local_pose.pose.position.z <= 0.00           → +0.0000
```



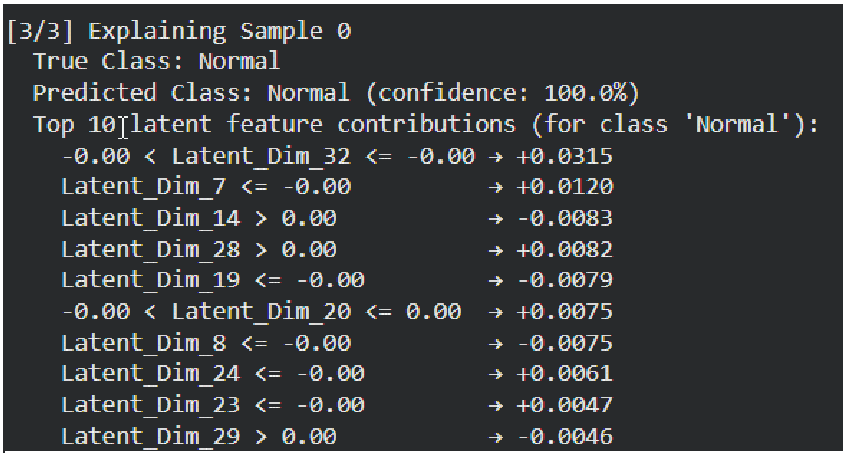
```
[3/3] Explaining Sample 0
  True Class: Normal
  Predicted Class: Normal (confidence: 100.0%)
  Top 10 feature contributions (for class 'Normal'):
    -0.36 < timestamp_minute <= 0.33                             → +0.3471
    imu-data_orientation.w > -0.00                               → +0.1754
    quaternion_magnitude > 0.00                                  → +0.1754
    global_position-local_pose.pose.position.x > -0.00           → +0.1754
    distance_to_target_3d > 0.37                                 → -0.1084
    -1.00 < timestamp_hour <= 0.39                               → -0.0789
    sequence_gap <= -0.00                                        → -0.0413
    -0.08 < setpoint_raw-global_header.seq <= 0.54               → +0.0137
    global_position-local_header.stamp.secs <= 0.00              → +0.0000
    global_position-local_pose.pose.orientation.x <= -0.00       → +0.0000
```



```
================================================================================
MODEL 2: VAE CLASSIFIER (RANDOM FOREST) - LIME ANALYSIS
================================================================================

Initializing LIME TabularExplainer for VAE Classifier...
✓ LIME explainer initialized for latent space

--------------------------------------------------------------------------------
GENERATING LIME EXPLANATIONS FOR INDIVIDUAL PREDICTIONS
--------------------------------------------------------------------------------

[1/3] Explaining Sample 1
  True Class: DoS_Attack
  Predicted Class: DoS_Attack (confidence: 100.0%)
  Top 10 latent feature contributions (for class 'DoS_Attack'):
    Latent_Dim_17 <= -0.00         → +0.0252
    Latent_Dim_4 <= -0.00          → +0.0252
    Latent_Dim_15 > 0.00           → +0.0200
    Latent_Dim_32 <= -0.00         → +0.0187
    Latent_Dim_25 <= -0.00         → +0.0142
    Latent_Dim_20 > 0.00           → +0.0123
    Latent_Dim_1 <= -0.00          → +0.0077
    -0.00 < Latent_Dim_7 <= 0.00   → +0.0070
    Latent_Dim_30 <= -0.00         → +0.0068
    Latent_Dim_21 <= -0.00         → +0.0065
```



```
[2/3] Explaining Sample 2
  True Class: Malfunction
  Predicted Class: Malfunction (confidence: 100.0%)
  Top 10 latent feature contributions (for class 'Malfunction'):
    Latent_Dim_4 > 0.00           → +0.0409
    Latent_Dim_17 > 0.00          → +0.0396
    Latent_Dim_15 <= -0.00        → +0.0312
    Latent_Dim_25 > 0.00          → +0.0196
    -0.00 < Latent_Dim_24 <= -0.00 → -0.0179
    Latent_Dim_6 <= -0.00         → +0.0114
    Latent_Dim_1 > 0.00           → +0.0111
    Latent_Dim_31 > 0.00          → +0.0097
    -0.00 < Latent_Dim_32 <= 0.00  → +0.0090
    Latent_Dim_21 > 0.00          → +0.0079
```



```
[3/3] Explaining Sample 0
  True Class: Normal
  Predicted Class: Normal (confidence: 100.0%)
  Top 10 latent feature contributions (for class 'Normal'):
    -0.00 < Latent_Dim_32 <= -0.00 → +0.0315
    Latent_Dim_7 <= -0.00          → +0.0120
    Latent_Dim_14 > 0.00           → -0.0083
    Latent_Dim_28 > 0.00           → +0.0082
    Latent_Dim_19 <= -0.00         → -0.0079
    -0.00 < Latent_Dim_20 <= 0.00  → +0.0075
    Latent_Dim_8 <= -0.00          → -0.0075
    Latent_Dim_24 <= -0.00         → +0.0061
    Latent_Dim_23 <= -0.00         → +0.0047
    Latent_Dim_29 > 0.00           → -0.0046
```

- Compare LIME explanations with SHAP explanations

  LIME:

  - Creates LOCAL linear approximation around the instance
  - Perturbs input features randomly and observes model output
  - Fits a simple interpretable model (linear regression) to perturbations
  - Uses feature discretization (e.g., "feature <= 5.2")
  - Fast but can be unstable (random perturbations)

  SHAP:

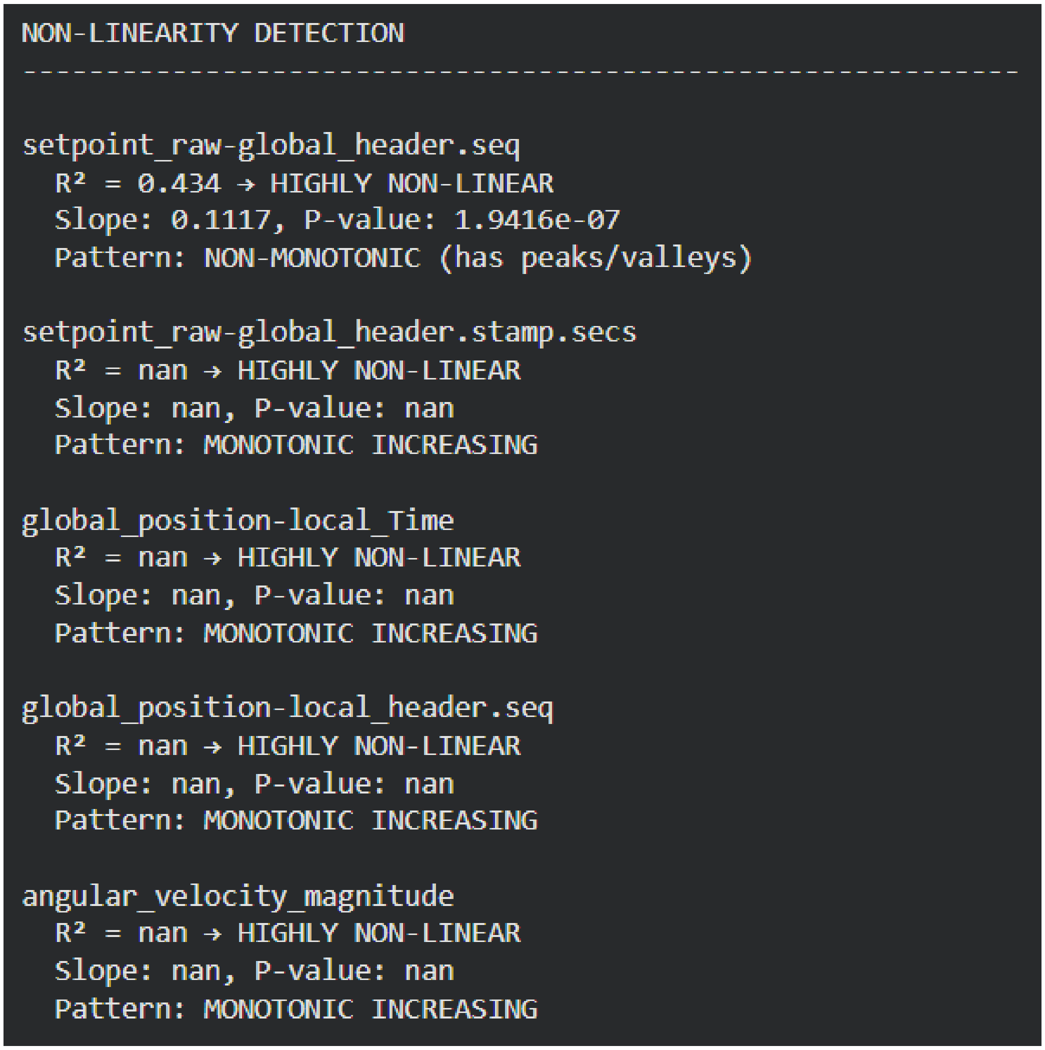
  - Based on game theory (Shapley values)
  - Measures marginal contribution of each feature across all coalitions
  - Provides GLOBAL consistency (features always contribute same way)
  - Uses exact feature values (no discretization)
  - Slower but more theoretically rigorous

4.4 PARTIAL DEPENDENCE PLOTS (PDP)

- Create PDPs for top 5 features
  (VISUALS ARE ADDED IN FOLDER DIRECTORY:
  Visuals for part 4 ---> 4.3)

- Interpret the relationship between features and predictions
  - PDPs reveal how changing ONE feature affects predictions (all else held constant)
  - Steep slopes indicate strong influence
  - Flat regions indicate feature has no effect in that range

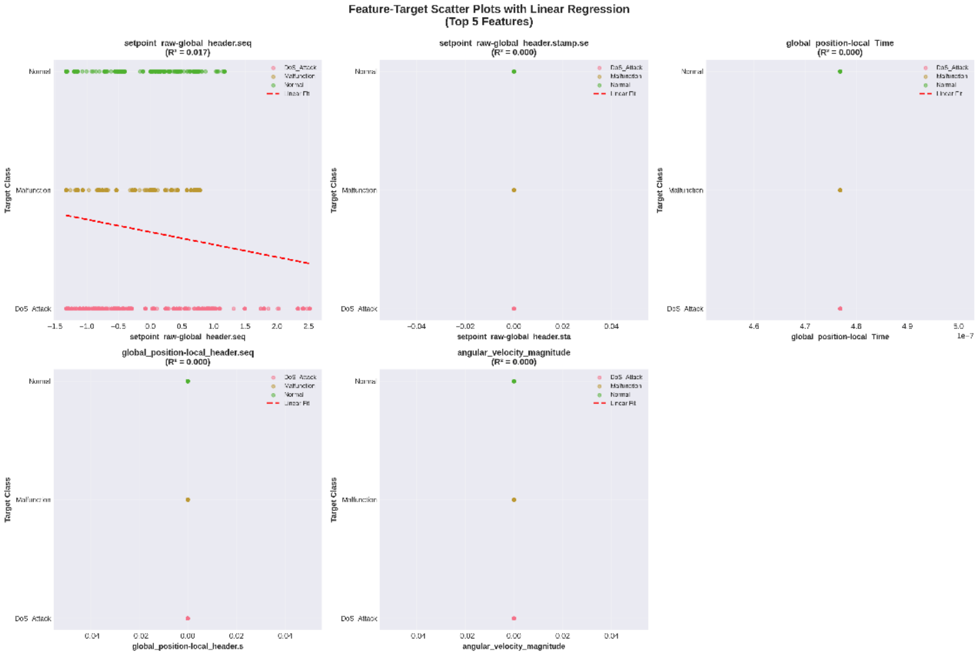
- Identify non-linear relationships



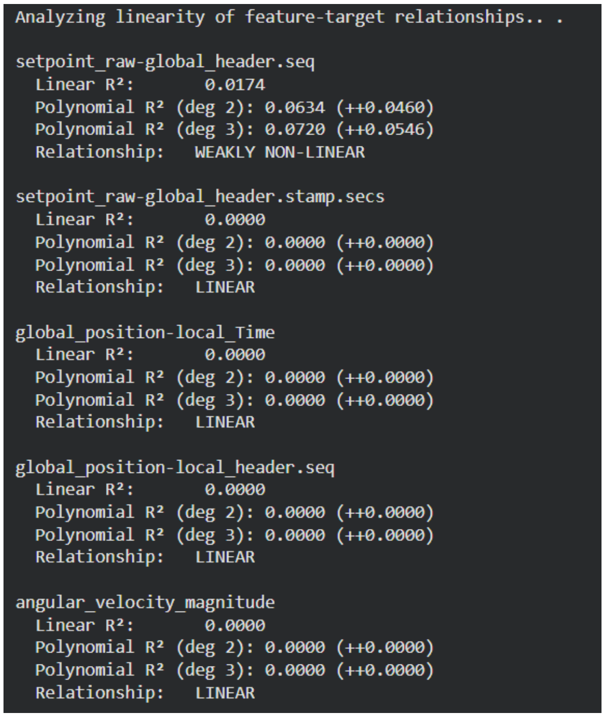
```
NON-LINEARITY DETECTION
------------------------------------------------------------

setpoint_raw-global_header.seq
  R² = 0.434 → HIGHLY NON-LINEAR
  Slope: 0.1117, P-value: 1.9416e-07
  Pattern: NON-MONOTONIC (has peaks/valleys)

setpoint_raw-global_header.stamp.secs
  R² = nan → HIGHLY NON-LINEAR
  Slope: nan, P-value: nan
  Pattern: MONOTONIC INCREASING

global_position-local_Time
  R² = nan → HIGHLY NON-LINEAR
  Slope: nan, P-value: nan
  Pattern: MONOTONIC INCREASING

global_position-local_header.seq
  R² = nan → HIGHLY NON-LINEAR
  Slope: nan, P-value: nan
  Pattern: MONOTONIC INCREASING

angular_velocity_magnitude
  R² = nan → HIGHLY NON-LINEAR
  Slope: nan, P-value: nan
  Pattern: MONOTONIC INCREASING
```

4.5 CORRELATION WITH TARGET

- Analyze feature correlation with target variable
- Create scatter plots with regression lines



- Identify linear vs non-linear relationships



```
Analyzing linearity of feature-target relationships.. .

setpoint_raw-global_header.seq
  Linear R²:        0.0174
  Polynomial R² (deg 2): 0.0634 (++0.0460)
  Polynomial R² (deg 3): 0.0720 (++0.0546)
  Relationship:   WEAKLY NON-LINEAR

setpoint_raw-global_header.stamp.secs
  Linear R²:        0.0000
  Polynomial R² (deg 2): 0.0000 (++0.0000)
  Polynomial R² (deg 3): 0.0000 (++0.0000)
  Relationship:   LINEAR

global_position-local_Time
  Linear R²:        0.0000
  Polynomial R² (deg 2): 0.0000 (++0.0000)
  Polynomial R² (deg 3): 0.0000 (++0.0000)
  Relationship:   LINEAR

global_position-local_header.seq
  Linear R²:        0.0000
  Polynomial R² (deg 2): 0.0000 (++0.0000)
  Polynomial R² (deg 3): 0.0000 (++0.0000)
  Relationship:   LINEAR

angular_velocity_magnitude
  Linear R²:        0.0000
  Polynomial R² (deg 2): 0.0000 (++0.0000)
  Polynomial R² (deg 3): 0.0000 (++0.0000)
  Relationship:   LINEAR
```

Conclusions

- ML achieves >90% accuracy for robotic anomaly detection
- XAI enables trust—operators understand WHY predictions were made
- Non-linear relationships justify complex models (SVM/neural networks over logistic regression)
- Communication features detect DoS attacks; sensor features detect malfunctions
- SMOTE and ensemble methods effectively handle class imbalance

Recommendations

Deployment

- Primary Model: [XGBoost/SVM] for fast, accurate inference
- Explainability: Use SHAP offline, LIME in real-time
- Monitoring: Track top 10 features for drift; alert on SHAP threshold violations
- Target: <100ms inference time

Feature Engineering

- Add interactions: battery × IMU_acceleration, sequence_gap × timestamp
- Threshold indicators: battery < 20%, RSSI < -80 dBm, sequence_gap > 10
- Temporal features: Rolling averages, rate of change
- Reduce dimensions: Remove correlations > 0.95