DroneDetect V2 - Raw IQ Feature Extraction

Overview

This notebook implements Raw IQ (downsampled) feature extraction for RF-based drone classification using deep learning architectures.

Methodology

We follow a **two-stage pipeline**: (1) IQ Downsampling → (2) Deep Learning Training (RF-UAVNet)

The IQ extraction process:

1. Loads raw I/Q samples from .dat files (60 MHz sampling rate)
2. **Normalizes per-file** (Min-max [0,1]) before segmentation - **DIFFERENT from PSD/Spectrogram!**
3. Segments continuous signals into fixed-duration windows (20ms)
4. **Downsamples** from ~1.2M to 10k samples via linear interpolation
5. Saves complex IQ data as (2, 10000) = [real, imag]

Why Raw IQ Features?

Raw IQ: Preserves phase information and signal structure for end-to-end learning. Enables neural networks to discover representations beyond hand-crafted features.

Advantages:

- Preserves full signal information (magnitude + phase)
- No information loss from feature engineering
- End-to-end learnable representations
- Matches RF-UAVNet architecture

Parameter Selection

Aligned with RF-UAVNet reference:

- **Segment duration**: **20ms**
- **Downsampled size**: **10,000 samples** (from ~1.2M)
- **Normalization**: **Min-max [0,1] per-file** - Matches RadioML2016 protocol
- **Format**: **(2, 10000)** - [I channel, Q channel]
- **Downsampling method**: **Linear interpolation** - Preserves signal shape

Critical Difference: Min-Max Normalization

Unlike PSD/Spectrogram (Z-score), IQ uses **min-max [0,1]** normalization:

- Prevents negative values (important for some architectures)
- Bounded range [0,1] stabilizes training
- Matches RF-UAVNet paper implementation
- Per-channel normalization (I and Q independently)

Downstream Usage

IQ features are consumed by:

- `05_training_rfuavnet_COLAB.ipynb` - RF-UAVNet deep learning

Reference Alignment

Parameters verified against REFERENTIEL_DRONEDETECT_RFCLASSIFICATION.md Section 3.2 and arXiv:2308.11833.

In [1]:
```
import sys
sys.path.insert(0, '../src')
```

```
import gc
import psutil
import os
import numpy as np
from tqdm import tqdm
from typing import Any, List
from sklearn.preprocessing import LabelEncoder
import zipfile

from dronedetect import config, data_loader, preprocessing, features

# Memory monitoring utility

def get_memory_mb():
    return psutil.Process(os.getpid()).memory_info().rss / 1024 / 1024


# Create output directory
config.FEATURES_DIR.mkdir(parents=True, exist_ok=True)

print(f"Initial memory: {get_memory_mb():.0f} MB")
```

```
Initial memory: 429 MB
```

Specifications

All preprocessing parameters defined here for reproducibility.

In [2]:

```
# ============================================================================
# PREPROCESSING SPECIFICATIONS
# ============================================================================
# All parameters controlling feature extraction defined here for reproducibility.

# --- Signal Segmentation ---
SEGMENT_DURATION_MS = 20        # Window length (config.DEFAULT_SEGMENT_MS)
                                 # RFClassification results: 10ms→20ms→50ms improves 76.9%→83.6%→89.4%

# --- Feature Extraction Parameters ---
N_FFT = 1024                     # FFT size - Reference: REFERENTIEL Section 1.2.1 (nperseg=1024)
```

```
                                # Frequency resolution = 60MHz/1024 ≈ 58.6 kHz/bin

SPECTROGRAM_SIZE = (224, 224)   # Output image dimensions (config.DEFAULT_SPEC_SIZE)
                                # Matches VGG16/ResNet input for transfer learning

IQ_DOWNSAMPLE_TARGET = 10000    # Downsampled IQ samples (config.DEFAULT_IQ_DOWNSAMPLE)
                                # Original: ~1.2M samples/segment → 10k for memory efficiency

# --- Batch Processing ---
BATCH_SIZE = 2                 # Files per batch (adjust based on available RAM)

# --- Dataset Info ---
SAMPLING_RATE_MHZ = 60          # DroneDetect V2 sampling frequency
EXPECTED_SEGMENTS_PER_FILE = 100 # Approx (2s recording / 20ms window)

print("Specifications loaded:")
print(f"  Segment: {SEGMENT_DURATION_MS}ms | FFT: {N_FFT} | Batch: {BATCH_SIZE} files")
```

```
Specifications loaded:
  Segment: 20ms | FFT: 1024 | Batch: 2 files
```

Preprocessing Architecture

Modular pipeline system for flexible feature extraction.

In [3]:

```
# ============================================================================
# PREPROCESSING ARCHITECTURE
# ============================================================================


class PreprocessingStep:
    """Base class for preprocessing steps."""

    def process(self, segment: np.ndarray) -> Any:
        """Process a signal segment.

        Args:
            segment: Input signal segment
```

```
        Returns:
            Processed output (type depends on step)
        """
        raise NotImplementedError

    def __repr__(self):
        return self.__class__.__name__


class PSDStep(PreprocessingStep):
    """Power Spectral Density via Welch method with per-sample normalization."""

    def __init__(self, nfft=1024):
        self.nfft = nfft

    def process(self, segment):
        _, psd = features.compute_psd(segment, nfft=self.nfft)
        # Per-sample normalization (reference: REFERENTIEL Section 1.3.1, line 220)
        # Division by max with zero-protection
        psd_max = np.max(psd)
        if psd_max < 1e-15:  # Essentially zero power
            return np.zeros_like(psd)
        return psd / psd_max

    def __repr__(self):
        return f"PSDStep(nfft={self.nfft})"


class SpectrogramStep(PreprocessingStep):
    """Spectrogram via STFT + resize."""

    def __init__(self, target_size=(224, 224)):
        self.target_size = target_size

    def process(self, segment):
        return features.compute_spectrogram(segment, target_size=self.target_size)

    def __repr__(self):
        return f"SpectrogramStep(size={self.target_size})"
```

```python
class DownsampleIQStep(PreprocessingStep):
    """Downsample IQ via linear interpolation."""

    def __init__(self, target_samples=10000):
        self.target_samples = target_samples

    def process(self, segment):
        return preprocessing.downsample_iq(segment, target_samples=self.target_samples)

    def __repr__(self):
        return f"DownsampleIQStep(n={self.target_samples})"


class FeaturePipeline:
    """Pipeline orchestrating multiple preprocessing steps."""

    def __init__(self, name: str, steps: List[PreprocessingStep]):
        """Initialize pipeline.

        Args:
            name: Pipeline identifier (used for output filename)
            steps: List of preprocessing steps to apply in order
        """
        self.name = name
        self.steps = steps

    def process_segment(self, segment: np.ndarray) -> np.ndarray:
        """Apply all steps sequentially.

        Args:
            segment: Input signal segment

        Returns:
            Final processed features
        """
        data = segment
        for step in self.steps:
            data = step.process(data)
        return data

    def get_output_filename(self) -> str:
```

```
        """Get output filename for this pipeline."""
        return f"{self.name}_features.npz"

    def __repr__(self):
        steps_str = ' → '.join([str(s) for s in self.steps])
        return f"Pipeline({self.name}): {steps_str}"


print("Preprocessing architecture loaded")
```

```
Preprocessing architecture loaded
```

Pipeline Configuration

Define preprocessing pipelines for different feature types.

In [4]:
```
# ============================================================================
# PIPELINE CONFIGURATION
# ============================================================================

# Single pipeline for IQ features
pipeline = FeaturePipeline(
    name='iq',
    steps=[DownsampleIQStep(target_samples=IQ_DOWNSAMPLE_TARGET)]
)

print(f"Configured pipeline: {pipeline}")
```

```
Configured pipeline: Pipeline(iq): DownsampleIQStep(n=10000)
```

1. Scan Dataset

In [5]:
```
df = data_loader.get_dataset_metadata(config.DATA_DIR)
print(f"Total files: {len(df)}")
```

```
Total files: 195
```

In [6]:
```
df.describe(include="all")
```

Out[6]:

	drone_code	drone_folder	wifi	bluetooth	interference	state	index	file_path	in
count	195	195	195	195	195	195	195.000000	195	
unique	7	7	2	2	2	3	NaN	195	
top	AIR	AIR	True	True	BOTH	ON	NaN	/home/sambot/win_downloads/DATASETS/drones/Dro...	
freq	30	30	100	100	100	70	NaN	1	
mean	NaN	NaN	NaN	NaN	NaN	NaN	2.000000	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	1.417854	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	2.000000	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	3.000000	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	4.000000	NaN	

In [7]:
```
df.head()
```

Out[7]:

	drone_code	drone_folder	wifi	bluetooth	interference	state	index	file_path	interference_
0	AIR	AIR	True	True	BOTH	FY	0	/home/sambot/win_downloads/DATASETS/drones/Dro...	
1	AIR	AIR	True	True	BOTH	FY	1	/home/sambot/win_downloads/DATASETS/drones/Dro...	
2	AIR	AIR	True	True	BOTH	FY	2	/home/sambot/win_downloads/DATASETS/drones/Dro...	
3	AIR	AIR	True	True	BOTH	FY	3	/home/sambot/win_downloads/DATASETS/drones/Dro...	
4	AIR	AIR	True	True	BOTH	FY	4	/home/sambot/win_downloads/DATASETS/drones/Dro...	

In [8]:
```
df.tail()
```

Out[8]:

	drone_code	drone_folder	wifi	bluetooth	interference	state	index	file_path	interferen
190	PHA	PHA	False	False	CLEAN	ON	0	/home/sambot/win_downloads/DATASETS/drones/Dro...	
191	PHA	PHA	False	False	CLEAN	ON	1	/home/sambot/win_downloads/DATASETS/drones/Dro...	
192	PHA	PHA	False	False	CLEAN	ON	2	/home/sambot/win_downloads/DATASETS/drones/Dro...	
193	PHA	PHA	False	False	CLEAN	ON	3	/home/sambot/win_downloads/DATASETS/drones/Dro...	
194	PHA	PHA	False	False	CLEAN	ON	4	/home/sambot/win_downloads/DATASETS/drones/Dro...	

In [9]:
```
df.info(verbose=True, show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   drone_code           195 non-null    object
 1   drone_folder         195 non-null    object
 2   wifi                 195 non-null    bool  
 3   bluetooth            195 non-null    bool  
 4   interference         195 non-null    object
 5   state                195 non-null    object
 6   index                195 non-null    int64 
 7   file_path            195 non-null    object
 8   interference_folder  195 non-null    object
dtypes: bool(2), int64(1), object(6)
memory usage: 11.2+ KB
```

2. Extract Features (Batch Processing)

Process files in batches to avoid memory saturation. Features are written progressively to disk.

In [10]:

```
# ============================================================================
# BATCH PROCESSING - Extract Features
# ============================================================================

# Initialize label encoders
from sklearn.preprocessing import LabelEncoder

drone_encoder = LabelEncoder()
interference_encoder = LabelEncoder()
state_encoder = LabelEncoder()

drone_encoder.fit(df['drone_code'].unique())
interference_encoder.fit(df['interference'].unique())
state_encoder.fit(df['state'].unique())

print(f"Label encoders initialized")
print(f"  Drones: {drone_encoder.classes_}")
print(f"  Interference: {interference_encoder.classes_}")
print(f"  States: {state_encoder.classes_}")

# Batch processing
batch_files = []
batch_idx = 0

for batch_start in tqdm(range(0, len(df), BATCH_SIZE), desc="Processing batches"):
    batch_df = df.iloc[batch_start:batch_start + BATCH_SIZE]

    # Storage for current batch
    batch_features = []
    labels_batch = []
    file_ids_batch = []

    print(f"\nBatch {batch_idx}: files {batch_start}-{batch_start + len(batch_df)}")

    for idx, row in batch_df.iterrows():
        try:
            # Load IQ data
            iq_data = data_loader.load_raw_iq(row['file_path'])

            # CRITICAL: Min-max [0,1] normalization for IQ/RF-UAVNet
```

```
            iq_normalized = preprocessing.normalize_minmax(iq_data)

            # Segment into 20ms windows
            segments = preprocessing.segment_signal(iq_normalized, segment_ms=SEGMENT_DURATION_MS)

            # Process each segment through pipeline
            for seg in segments:
                seg = seg.copy()  # Break view to allow memory release

                # Extract features
                feature_output = pipeline.process_segment(seg)
                batch_features.append(feature_output)

                # Encode labels
                labels_batch.append({
                    'drone': drone_encoder.transform([row['drone_code']])[0],
                    'interference': interference_encoder.transform([row['interference']])[0],
                    'state': state_encoder.transform([row['state']])[0]
                })
                file_ids_batch.append(idx)

                del seg

            # Free memory
            del iq_data, iq_normalized, segments
            gc.collect()

        except Exception as e:
            print(f"  Error processing {row['file_path']}: {e}")
            raise Exception from e

    # Save batch to disk
    batch_file = config.FEATURES_DIR / f'batch_{batch_idx:03d}.npz'
    np.savez_compressed(
        batch_file,
        iq=np.array(batch_features),
        labels=np.array(labels_batch, dtype=object),
        file_ids=np.array(file_ids_batch, dtype=np.int32)
    )
    batch_files.append(batch_file)
```

```
    print(f"  Saved {len(batch_features)} samples to {batch_file.name}")
    print(f"  Memory: {get_memory_mb():.0f} MB")

    # Clean batch data
    del batch_features, labels_batch, file_ids_batch
    gc.collect()

    batch_idx += 1

print(f"\nBatch processing complete: {len(batch_files)} batches saved")
print(f"Memory: {get_memory_mb():.0f} MB")
```

```
Label encoders initialized
  Drones: ['AIR' 'DIS' 'INS' 'MA1' 'MAV' 'MIN' 'PHA']
  Interference: ['BOTH' 'CLEAN']
  States: ['FY' 'HO' 'ON']
Processing batches:   0%|          | 0/98 [00:00<?, ?it/s]
Batch 0: files 0-2
Processing batches:   1%|          | 1/98 [00:17<28:32, 17.65s/it]
  Saved 200 samples to batch_000.npz
  Memory: 501 MB

Batch 1: files 2-4
Processing batches:   2%||          | 2/98 [00:35<28:35, 17.87s/it]
  Saved 200 samples to batch_001.npz
  Memory: 478 MB

Batch 2: files 4-6
Processing batches:   3%|▌          | 3/98 [00:52<27:30, 17.37s/it]
  Saved 200 samples to batch_002.npz
  Memory: 478 MB

Batch 3: files 6-8
Processing batches:   4%|▋          | 4/98 [01:10<27:32, 17.58s/it]
  Saved 200 samples to batch_003.npz
  Memory: 532 MB

Batch 4: files 8-10
Processing batches:   5%|▊          | 5/98 [01:26<26:12, 16.91s/it]
```

```
  Saved 200 samples to batch_004.npz
  Memory: 532 MB

Batch 5: files 10-12
Processing batches:   6%|▌         | 6/98 [01:43<26:09, 17.06s/it]
  Saved 200 samples to batch_005.npz
  Memory: 534 MB

Batch 6: files 12-14
Processing batches:   7%|▋         | 7/98 [02:00<26:03, 17.18s/it]
  Saved 200 samples to batch_006.npz
  Memory: 531 MB

Batch 7: files 14-16
Processing batches:   8%|▊         | 8/98 [02:17<25:23, 16.93s/it]
  Saved 200 samples to batch_007.npz
  Memory: 533 MB

Batch 8: files 16-18
Processing batches:   9%|▉         | 9/98 [02:35<25:40, 17.31s/it]
  Saved 200 samples to batch_008.npz
  Memory: 479 MB

Batch 9: files 18-20
Processing batches:  10%|▉         | 10/98 [02:52<25:30, 17.39s/it]
  Saved 200 samples to batch_009.npz
  Memory: 479 MB

Batch 10: files 20-22
Processing batches:  11%|█         | 11/98 [03:11<25:34, 17.64s/it]
  Saved 200 samples to batch_010.npz
  Memory: 530 MB

Batch 11: files 22-24
Processing batches:  12%|█▏        | 12/98 [03:26<24:22, 17.00s/it]
  Saved 200 samples to batch_011.npz
  Memory: 479 MB

Batch 12: files 24-26
```

```
Processing batches:  13%|▉         | 13/98 [03:44<24:13, 17.10s/it]
  Saved 191 samples to batch_012.npz
  Memory: 527 MB

Batch 13: files 26-28
Processing batches:  14%|▉         | 14/98 [04:03<24:49, 17.73s/it]
  Saved 200 samples to batch_013.npz
  Memory: 531 MB

Batch 14: files 28-30
Processing batches:  15%|▉         | 15/98 [04:21<24:54, 18.01s/it]
  Saved 200 samples to batch_014.npz
  Memory: 531 MB

Batch 15: files 30-32
Processing batches:  16%|▉         | 16/98 [04:40<24:46, 18.13s/it]
  Saved 200 samples to batch_015.npz
  Memory: 531 MB

Batch 16: files 32-34
Processing batches:  17%|▉         | 17/98 [04:57<24:11, 17.92s/it]
  Saved 200 samples to batch_016.npz
  Memory: 531 MB

Batch 17: files 34-36
Processing batches:  18%|▉         | 18/98 [05:16<24:16, 18.21s/it]
  Saved 200 samples to batch_017.npz
  Memory: 532 MB

Batch 18: files 36-38
Processing batches:  19%|▉         | 19/98 [05:36<24:33, 18.66s/it]
  Saved 200 samples to batch_018.npz
  Memory: 479 MB

Batch 19: files 38-40
Processing batches:  20%|▉         | 20/98 [05:55<24:19, 18.72s/it]
```

```
  Saved 200 samples to batch_019.npz
  Memory: 479 MB

Batch 20: files 40-42
Processing batches:  21%|██▏        | 21/98 [06:12<23:39, 18.44s/it]
  Saved 200 samples to batch_020.npz
  Memory: 531 MB

Batch 21: files 42-44
Processing batches:  22%|██▏        | 22/98 [06:30<23:00, 18.17s/it]
  Saved 200 samples to batch_021.npz
  Memory: 531 MB

Batch 22: files 44-46
Processing batches:  23%|██▎        | 23/98 [06:48<22:44, 18.20s/it]
  Saved 200 samples to batch_022.npz
  Memory: 531 MB

Batch 23: files 46-48
Processing batches:  24%|██▍        | 24/98 [07:07<22:37, 18.35s/it]
  Saved 200 samples to batch_023.npz
  Memory: 531 MB

Batch 24: files 48-50
Processing batches:  26%|██▌        | 25/98 [07:26<22:27, 18.46s/it]
  Saved 200 samples to batch_024.npz
  Memory: 531 MB

Batch 25: files 50-52
Processing batches:  27%|██▋        | 26/98 [07:44<22:14, 18.53s/it]
  Saved 200 samples to batch_025.npz
  Memory: 533 MB

Batch 26: files 52-54
Processing batches:  28%|██▊        | 27/98 [08:04<22:09, 18.73s/it]
  Saved 200 samples to batch_026.npz
  Memory: 532 MB

Batch 27: files 54-56
```

```
Processing batches:  29%|██▉       | 28/98 [08:22<21:48, 18.69s/it]
  Saved 200 samples to batch_027.npz
  Memory: 532 MB

Batch 28: files 56-58
Processing batches:  30%|███       | 29/98 [08:43<22:07, 19.24s/it]
  Saved 200 samples to batch_028.npz
  Memory: 530 MB

Batch 29: files 58-60
Processing batches:  31%|███       | 30/98 [09:02<21:45, 19.20s/it]
  Saved 200 samples to batch_029.npz
  Memory: 531 MB

Batch 30: files 60-62
Processing batches:  32%|███▏      | 31/98 [09:21<21:29, 19.24s/it]
  Saved 200 samples to batch_030.npz
  Memory: 531 MB

Batch 31: files 62-64
Processing batches:  33%|███▎      | 32/98 [09:40<21:01, 19.11s/it]
  Saved 200 samples to batch_031.npz
  Memory: 531 MB

Batch 32: files 64-66
Processing batches:  34%|███▎      | 33/98 [09:57<20:08, 18.60s/it]
  Saved 200 samples to batch_032.npz
  Memory: 531 MB

Batch 33: files 66-68
Processing batches:  35%|███▍      | 34/98 [10:15<19:23, 18.18s/it]
  Saved 200 samples to batch_033.npz
  Memory: 533 MB

Batch 34: files 68-70
Processing batches:  36%|███▌      | 35/98 [10:32<18:44, 17.85s/it]
```

```
  Saved 200 samples to batch_034.npz
  Memory: 533 MB

Batch 35: files 70-72
Processing batches:  37%|███▋      | 36/98 [10:50<18:33, 17.95s/it]
  Saved 200 samples to batch_035.npz
  Memory: 531 MB

Batch 36: files 72-74
Processing batches:  38%|███▊      | 37/98 [11:09<18:44, 18.44s/it]
  Saved 200 samples to batch_036.npz
  Memory: 531 MB

Batch 37: files 74-76
Processing batches:  39%|███▉      | 38/98 [11:27<18:20, 18.34s/it]
  Saved 200 samples to batch_037.npz
  Memory: 531 MB

Batch 38: files 76-78
Processing batches:  40%|███▉      | 39/98 [11:47<18:28, 18.79s/it]
  Saved 200 samples to batch_038.npz
  Memory: 531 MB

Batch 39: files 78-80
Processing batches:  41%|████      | 40/98 [12:06<18:10, 18.80s/it]
  Saved 200 samples to batch_039.npz
  Memory: 531 MB

Batch 40: files 80-82
Processing batches:  42%|████      | 41/98 [12:25<17:44, 18.68s/it]
  Saved 200 samples to batch_040.npz
  Memory: 479 MB

Batch 41: files 82-84
Processing batches:  43%|████      | 42/98 [12:42<17:13, 18.45s/it]
  Saved 200 samples to batch_041.npz
  Memory: 479 MB

Batch 42: files 84-86
```

```
Processing batches:  44%|████▍      | 43/98 [13:00<16:31, 18.04s/it]
  Saved 200 samples to batch_042.npz
  Memory: 479 MB

Batch 43: files 86-88
Processing batches:  45%|████▌      | 44/98 [13:19<16:30, 18.34s/it]
  Saved 200 samples to batch_043.npz
  Memory: 534 MB

Batch 44: files 88-90
Processing batches:  46%|████▌      | 45/98 [13:51<19:55, 22.56s/it]
  Saved 200 samples to batch_044.npz
  Memory: 534 MB

Batch 45: files 90-92
Processing batches:  47%|████▋      | 46/98 [14:10<18:45, 21.64s/it]
  Saved 200 samples to batch_045.npz
  Memory: 479 MB

Batch 46: files 92-94
Processing batches:  48%|████▊      | 47/98 [14:30<17:48, 20.94s/it]
  Saved 200 samples to batch_046.npz
  Memory: 479 MB

Batch 47: files 94-96
Processing batches:  49%|████▉      | 48/98 [14:48<16:49, 20.19s/it]
  Saved 200 samples to batch_047.npz
  Memory: 479 MB

Batch 48: files 96-98
Processing batches:  50%|█████      | 49/98 [15:06<15:56, 19.51s/it]
  Saved 200 samples to batch_048.npz
  Memory: 479 MB

Batch 49: files 98-100
Processing batches:  51%|█████      | 50/98 [15:25<15:20, 19.17s/it]
```

```
  Saved 200 samples to batch_049.npz
  Memory: 479 MB

Batch 50: files 100-102
Processing batches:  52%|█████     | 51/98 [15:42<14:39, 18.71s/it]
  Saved 200 samples to batch_050.npz
  Memory: 479 MB

Batch 51: files 102-104
Processing batches:  53%|█████     | 52/98 [15:59<14:01, 18.28s/it]
  Saved 200 samples to batch_051.npz
  Memory: 530 MB

Batch 52: files 104-106
Processing batches:  54%|█████     | 53/98 [16:18<13:49, 18.43s/it]
  Saved 200 samples to batch_052.npz
  Memory: 479 MB

Batch 53: files 106-108
Processing batches:  55%|█████     | 54/98 [16:36<13:15, 18.09s/it]
  Saved 200 samples to batch_053.npz
  Memory: 532 MB

Batch 54: files 108-110
Processing batches:  56%|█████     | 55/98 [16:54<12:57, 18.07s/it]
  Saved 200 samples to batch_054.npz
  Memory: 528 MB

Batch 55: files 110-112
Processing batches:  57%|█████     | 56/98 [17:12<12:46, 18.26s/it]
  Saved 200 samples to batch_055.npz
  Memory: 533 MB

Batch 56: files 112-114
Processing batches:  58%|█████     | 57/98 [17:31<12:31, 18.32s/it]
  Saved 200 samples to batch_056.npz
  Memory: 530 MB

Batch 57: files 114-116
```

```
Processing batches:  59%|██████    | 58/98 [17:49<12:13, 18.34s/it]
  Saved 200 samples to batch_057.npz
  Memory: 479 MB

Batch 58: files 116-118
Processing batches:  60%|██████    | 59/98 [18:06<11:40, 17.95s/it]
  Saved 200 samples to batch_058.npz
  Memory: 533 MB

Batch 59: files 118-120
Processing batches:  61%|██████    | 60/98 [18:22<10:57, 17.31s/it]
  Saved 200 samples to batch_059.npz
  Memory: 533 MB

Batch 60: files 120-122
Processing batches:  62%|██████▏   | 61/98 [18:38<10:29, 17.02s/it]
  Saved 200 samples to batch_060.npz
  Memory: 533 MB

Batch 61: files 122-124
Processing batches:  63%|██████▏   | 62/98 [18:56<10:15, 17.10s/it]
  Saved 200 samples to batch_061.npz
  Memory: 530 MB

Batch 62: files 124-126
Processing batches:  64%|██████▍   | 63/98 [19:11<09:40, 16.59s/it]
  Saved 187 samples to batch_062.npz
  Memory: 477 MB

Batch 63: files 126-128
Processing batches:  65%|██████▌   | 64/98 [19:27<09:18, 16.42s/it]
  Saved 200 samples to batch_063.npz
  Memory: 530 MB

Batch 64: files 128-130
Processing batches:  66%|██████▋   | 65/98 [19:43<08:55, 16.23s/it]
```

```
  Saved 200 samples to batch_064.npz
  Memory: 531 MB

Batch 65: files 130-132
Processing batches:  67%|██████▋   | 66/98 [20:01<08:58, 16.82s/it]
  Saved 200 samples to batch_065.npz
  Memory: 529 MB

Batch 66: files 132-134
Processing batches:  68%|██████▊   | 67/98 [20:19<08:50, 17.10s/it]
  Saved 200 samples to batch_066.npz
  Memory: 530 MB

Batch 67: files 134-136
Processing batches:  69%|██████▉   | 68/98 [20:35<08:29, 16.99s/it]
  Saved 200 samples to batch_067.npz
  Memory: 533 MB

Batch 68: files 136-138
Processing batches:  70%|███████   | 69/98 [20:54<08:23, 17.38s/it]
  Saved 200 samples to batch_068.npz
  Memory: 479 MB

Batch 69: files 138-140
Processing batches:  71%|███████▏  | 70/98 [21:10<08:00, 17.15s/it]
  Saved 200 samples to batch_069.npz
  Memory: 479 MB

Batch 70: files 140-142
Processing batches:  72%|███████▏  | 71/98 [21:27<07:38, 16.97s/it]
  Saved 200 samples to batch_070.npz
  Memory: 530 MB

Batch 71: files 142-144
Processing batches:  73%|███████▎  | 72/98 [21:45<07:26, 17.18s/it]
  Saved 200 samples to batch_071.npz
  Memory: 531 MB

Batch 72: files 144-146
```

```
Processing batches:  74%|███████▍  | 73/98 [22:01<07:06, 17.07s/it]
  Saved 200 samples to batch_072.npz
  Memory: 530 MB

Batch 73: files 146-148
Processing batches:  76%|███████▌  | 74/98 [22:18<06:49, 17.06s/it]
  Saved 200 samples to batch_073.npz
  Memory: 530 MB

Batch 74: files 148-150
Processing batches:  77%|███████▋  | 75/98 [22:34<06:25, 16.75s/it]
  Saved 200 samples to batch_074.npz
  Memory: 530 MB

Batch 75: files 150-152
Processing batches:  78%|███████▊  | 76/98 [22:51<06:07, 16.69s/it]
  Saved 200 samples to batch_075.npz
  Memory: 531 MB

Batch 76: files 152-154
Processing batches:  79%|███████▉  | 77/98 [23:10<06:02, 17.26s/it]
  Saved 200 samples to batch_076.npz
  Memory: 531 MB

Batch 77: files 154-156
Processing batches:  80%|███████▉  | 78/98 [23:29<05:58, 17.94s/it]
  Saved 200 samples to batch_077.npz
  Memory: 531 MB

Batch 78: files 156-158
Processing batches:  81%|████████  | 79/98 [23:47<05:38, 17.84s/it]
  Saved 200 samples to batch_078.npz
  Memory: 531 MB

Batch 79: files 158-160
Processing batches:  82%|████████▏ | 80/98 [24:03<05:13, 17.44s/it]
```

```
  Saved 200 samples to batch_079.npz
  Memory: 529 MB

Batch 80: files 160-162
Processing batches:  83%|████████▎  | 81/98 [24:21<04:55, 17.41s/it]
  Saved 200 samples to batch_080.npz
  Memory: 530 MB

Batch 81: files 162-164
Processing batches:  84%|████████▍  | 82/98 [24:38<04:39, 17.48s/it]
  Saved 200 samples to batch_081.npz
  Memory: 530 MB

Batch 82: files 164-166
Processing batches:  85%|████████▍  | 83/98 [24:56<04:22, 17.49s/it]
  Saved 200 samples to batch_082.npz
  Memory: 533 MB

Batch 83: files 166-168
Processing batches:  86%|████████▌  | 84/98 [25:12<04:00, 17.20s/it]
  Saved 200 samples to batch_083.npz
  Memory: 478 MB

Batch 84: files 168-170
Processing batches:  87%|████████▋  | 85/98 [25:29<03:40, 16.99s/it]
  Saved 200 samples to batch_084.npz
  Memory: 478 MB

Batch 85: files 170-172
Processing batches:  88%|████████▊  | 86/98 [25:46<03:25, 17.10s/it]
  Saved 200 samples to batch_085.npz
  Memory: 530 MB

Batch 86: files 172-174
Processing batches:  89%|████████▉  | 87/98 [26:03<03:07, 17.00s/it]
  Saved 200 samples to batch_086.npz
  Memory: 530 MB

Batch 87: files 174-176
```

```
Processing batches:  90%|████████▉ | 88/98 [26:20<02:51, 17.15s/it]
  Saved 200 samples to batch_087.npz
  Memory: 530 MB

Batch 88: files 176-178
Processing batches:  91%|█████████ | 89/98 [26:37<02:33, 17.04s/it]
  Saved 200 samples to batch_088.npz
  Memory: 529 MB

Batch 89: files 178-180
Processing batches:  92%|█████████▏ | 90/98 [26:54<02:16, 17.11s/it]
  Saved 200 samples to batch_089.npz
  Memory: 529 MB

Batch 90: files 180-182
Processing batches:  93%|█████████▎ | 91/98 [27:11<01:58, 16.91s/it]
  Saved 200 samples to batch_090.npz
  Memory: 479 MB

Batch 91: files 182-184
Processing batches:  94%|█████████▍ | 92/98 [27:27<01:40, 16.81s/it]
  Saved 200 samples to batch_091.npz
  Memory: 479 MB

Batch 92: files 184-186
Processing batches:  95%|█████████▍ | 93/98 [27:45<01:25, 17.08s/it]
  Saved 200 samples to batch_092.npz
  Memory: 479 MB

Batch 93: files 186-188
Processing batches:  96%|█████████▌ | 94/98 [28:02<01:07, 16.91s/it]
  Saved 200 samples to batch_093.npz
  Memory: 534 MB

Batch 94: files 188-190
Processing batches:  97%|█████████▋ | 95/98 [28:19<00:51, 17.15s/it]
```

```
  Saved 200 samples to batch_094.npz
  Memory: 479 MB

Batch 95: files 190-192
Processing batches:  98%|█████████▊| 96/98 [28:36<00:33, 16.89s/it]
  Saved 200 samples to batch_095.npz
  Memory: 534 MB

Batch 96: files 192-194
Processing batches:  99%|█████████▉| 97/98 [28:53<00:16, 16.88s/it]
  Saved 200 samples to batch_096.npz
  Memory: 534 MB

Batch 97: files 194-195
Processing batches: 100%|██████████| 98/98 [29:01<00:00, 17.77s/it]
  Saved 100 samples to batch_097.npz
  Memory: 504 MB

Batch processing complete: 98 batches saved
Memory: 504 MB
```

In [ ]:

3. Merge Batches into Final Files

Combine all batch files into single feature files.

In [11]:

```python
# ============================================================================
# OPTIMIZED BATCH MERGING (Disk-backed, ZERO RAM accumulation)
# ============================================================================
print("Merging batches (disk-backed memmap)...")

MERGE_CHUNK_SIZE = 5  # Process N batches at a time


def merge_pipeline_chunked(pipeline_name, batch_files, chunk_size=5):
    """Merge batches for a single pipeline using disk-backed memory mapping.
```

```
    This approach prevents memory saturation by:
    1. Writing directly to disk via memmap (NO RAM accumulation)
    2. Processing one pipeline at a time
    3. Pre-allocating final array size
    4. Keeping memmap for direct npz save (no RAM load)

    Args:
        pipeline_name: Name of pipeline ('psd', 'spectrogram', 'iq')
        batch_files: List of batch file paths
        chunk_size: Number of batches to process simultaneously

    Returns:
        dict with memmap + metadata (labels stay in RAM - small)
    """
    print(f"\n[{pipeline_name}] Processing {len(batch_files)} batches (disk-backed)...")

    # STEP 1: Calculate total size by scanning batches
    print(f"[{pipeline_name}] Scanning batches to determine size...")
    total_samples = 0
    feature_shape = None

    for batch_file in batch_files:
        data = np.load(batch_file, allow_pickle=True)
        batch_features = data[pipeline_name]
        total_samples += len(batch_features)
        if feature_shape is None:
            feature_shape = batch_features.shape[1:]  # (1024,) or (224,224,3)
        del data

    print(f"[{pipeline_name}] Total samples: {total_samples}, feature shape: {feature_shape}")

    # STEP 2: Create memmap file (writes directly to disk)
    memmap_file = config.FEATURES_DIR / f'{pipeline_name}_features_X.mmap'
    final_shape = (total_samples,) + feature_shape

    print(f"[{pipeline_name}] Creating memmap: {final_shape} ({np.prod(final_shape)*4/1e9:.2f} GB)")
    X_memmap = np.memmap(memmap_file, dtype='float32', mode='w+', shape=final_shape)

    # Storage for labels (small, can stay in RAM)
    all_drone_labels = []
```

```
    all_interference_labels = []
    all_state_labels = []
    all_file_ids = []

    # STEP 3: Fill memmap progressively
    write_offset = 0

    for chunk_start in tqdm(range(0, len(batch_files), chunk_size), desc=f"{pipeline_name} chunks"):
        chunk_batch_files = batch_files[chunk_start:chunk_start + chunk_size]

        for batch_file in chunk_batch_files:
            data = np.load(batch_file, allow_pickle=True)

            # Write features directly to disk
            batch_features = data[pipeline_name]
            n_samples = len(batch_features)
            X_memmap[write_offset:write_offset + n_samples] = batch_features
            write_offset += n_samples

            # Collect labels (small)
            labels = data['labels']
            all_drone_labels.extend([label['drone'] for label in labels])
            all_interference_labels.extend([label['interference'] for label in labels])
            all_state_labels.extend([label['state'] for label in labels])
            all_file_ids.append(data['file_ids'])

            del data, batch_features

        # Flush to disk periodically
        X_memmap.flush()
        gc.collect()

        print(f"  Written {write_offset}/{total_samples} samples, RAM: {get_memory_mb():.0f} MB")

    # STEP 4: Final flush (keep memmap, do NOT load into RAM)
    X_memmap.flush()

    final_file_ids = np.concatenate(all_file_ids, axis=0)

    return {
        'X_memmap': X_memmap,  # Return memmap directly
```

```
        'memmap_file': memmap_file,
        'memmap_shape': final_shape,
        'y_drone': np.array(all_drone_labels, dtype=np.int32),
        'y_interference': np.array(all_interference_labels, dtype=np.int32),
        'y_state': np.array(all_state_labels, dtype=np.int32),
        'file_ids': final_file_ids
    }


# Process each pipeline separately (reduces peak memory ~3x)
final_data = {}
# Process single pipeline
print(f"\n{'='*60}")
final_data = merge_pipeline_chunked(
    "iq",
    batch_files,
    chunk_size=MERGE_CHUNK_SIZE
)
print(f"[iq] Shape: {final_data['memmap_shape']}")
print(f"Memory after iq: {get_memory_mb():.0f} MB")

print("\n" + "="*60)
print("Final shapes (memmap-backed):")
# Process single pipeline
print(f"  {pipeline.name}: {final_data['memmap_shape']}")
print(f"  File IDs: {final_data['file_ids'].shape} (unique: {len(np.unique(final_data['file_ids']))})")
print(f"Final memory: {get_memory_mb():.0f} MB")
```

```
Merging batches (disk-backed memmap)...

============================================================

[iq] Processing 98 batches (disk-backed)...
[iq] Scanning batches to determine size...
[iq] Total samples: 19478, feature shape: (2, 10000)
[iq] Creating memmap: (19478, 2, 10000) (1.56 GB)
iq chunks:   5%|▌         | 1/20 [00:00<00:14,  1.32it/s]
  Written 1000/19478 samples, RAM: 584 MB
iq chunks:  10%|█         | 2/20 [00:01<00:13,  1.33it/s]
  Written 2000/19478 samples, RAM: 660 MB
```

```
iq chunks:  15%|██         | 3/20 [00:02<00:13,  1.23it/s]
  Written 2991/19478 samples, RAM: 736 MB
iq chunks:  20%|██         | 4/20 [00:03<00:14,  1.14it/s]
  Written 3991/19478 samples, RAM: 812 MB
iq chunks:  25%|███        | 5/20 [00:04<00:13,  1.15it/s]
  Written 4991/19478 samples, RAM: 889 MB
iq chunks:  30%|███        | 6/20 [00:04<00:11,  1.21it/s]
  Written 5991/19478 samples, RAM: 965 MB
iq chunks:  35%|████       | 7/20 [00:05<00:10,  1.26it/s]
  Written 6991/19478 samples, RAM: 1041 MB
iq chunks:  40%|████       | 8/20 [00:06<00:09,  1.29it/s]
  Written 7991/19478 samples, RAM: 1118 MB
iq chunks:  45%|█████      | 9/20 [00:07<00:08,  1.26it/s]
  Written 8991/19478 samples, RAM: 1194 MB
iq chunks:  50%|█████      | 10/20 [00:08<00:09,  1.11it/s]
  Written 9991/19478 samples, RAM: 1271 MB
iq chunks:  55%|██████     | 11/20 [00:09<00:07,  1.14it/s]
  Written 10991/19478 samples, RAM: 1347 MB
iq chunks:  60%|██████     | 12/20 [00:10<00:06,  1.17it/s]
  Written 11991/19478 samples, RAM: 1423 MB
iq chunks:  65%|███████    | 13/20 [00:10<00:05,  1.23it/s]
  Written 12978/19478 samples, RAM: 1499 MB
iq chunks:  70%|███████    | 14/20 [00:11<00:04,  1.29it/s]
  Written 13978/19478 samples, RAM: 1575 MB
iq chunks:  75%|████████   | 15/20 [00:12<00:03,  1.34it/s]
  Written 14978/19478 samples, RAM: 1651 MB
iq chunks:  80%|████████   | 16/20 [00:13<00:03,  1.26it/s]
  Written 15978/19478 samples, RAM: 1728 MB
iq chunks:  85%|█████████  | 17/20 [00:14<00:02,  1.16it/s]
  Written 16978/19478 samples, RAM: 1801 MB
iq chunks:  90%|█████████  | 18/20 [00:14<00:01,  1.22it/s]
  Written 17978/19478 samples, RAM: 1877 MB
iq chunks:  95%|██████████ | 19/20 [00:16<00:00,  1.06it/s]
  Written 18978/19478 samples, RAM: 1926 MB
iq chunks: 100%|██████████| 20/20 [00:16<00:00,  1.20it/s]
```

```
  Written 19478/19478 samples, RAM: 1939 MB
[iq] Shape: (19478, 2, 10000)
Memory after iq: 1939 MB

============================================================
Final shapes (memmap-backed):
  iq: (19478, 2, 10000)
  File IDs: (19478,) (unique: 195)
Final memory: 1939 MB
```

4. Save Final Features to Disk

In [12]:

```python
# Save features for each pipeline (directly from memmap, no RAM load)
print("Saving features to npz (streaming from memmap)...")

# Save single pipeline
output_file = config.FEATURES_DIR / "iq_features.npz"

# Save directly from memmap (npz will read from disk as needed)
np.savez_compressed(
    output_file,
    X=final_data['X_memmap'],  # Memmap array
    y_drone=final_data['y_drone'],
    y_interference=final_data['y_interference'],
    y_state=final_data['y_state'],
    file_ids=final_data['file_ids'],
    drone_classes=drone_encoder.classes_,
    interference_classes=interference_encoder.classes_,
    state_classes=state_encoder.classes_,
    # Metadata for frequency conversion (baseband to absolute RF)
    fs=config.FS,
    center_freq=config.CENTER_FREQ,
    bandwidth=config.BANDWIDTH
)
print(f"Saved iq features to {output_file}")

# Clean up memmap
del final_data['X_memmap']
final_data['memmap_file'].unlink()
```

```
# Free remaining memory
del final_data
gc.collect()

print(f"\nAll features saved successfully! Final RAM: {get_memory_mb():.0f} MB")
```

```
Saving features to npz (streaming from memmap)...
Saved iq features to data/features/iq_features.npz

All features saved successfully! Final RAM: 454 MB
```

5. Verify Saved Files

In [13]:

```
import zipfile

with zipfile.ZipFile(config.FEATURES_DIR / 'iq_features.npz', 'r') as z:
    print("Keys in archive:", z.namelist())

    # Read X shape
    with z.open('X.npy') as f:
        version = np.lib.format.read_magic(f)
        shape, fortran, dtype = np.lib.format._read_array_header(f, version)
        print(f"  X shape: {shape}, dtype: {dtype}")

    # Read y_drone shape
    with z.open('y_drone.npy') as f:
        version = np.lib.format.read_magic(f)
        shape, fortran, dtype = np.lib.format._read_array_header(f, version)
        print(f"  y_drone shape: {shape}, dtype: {dtype}")

    # Classes can be loaded (small arrays)
    drone_classes = np.load(z.open('drone_classes.npy'), allow_pickle=True)
    interference_classes = np.load(z.open('interference_classes.npy'), allow_pickle=True)
    state_classes = np.load(z.open('state_classes.npy'), allow_pickle=True)

    print(f"  Drone classes: {drone_classes}")
    print(f"  Interference classes: {interference_classes}")
    print(f"  State classes: {state_classes}")
```

```
Keys in archive: ['X.npy', 'y_drone.npy', 'y_interference.npy', 'y_state.npy', 'file_ids.npy', 'drone_classes.npy', 'interferen
ce_classes.npy', 'state_classes.npy', 'fs.npy', 'center_freq.npy', 'bandwidth.npy']
  X shape: (19478, 2, 10000), dtype: float32
  y_drone shape: (19478,), dtype: int32
  Drone classes: ['AIR' 'DIS' 'INS' 'MA1' 'MAV' 'MIN' 'PHA']
  Interference classes: ['BOTH' 'CLEAN']
  State classes: ['FY' 'HO' 'ON']
```

Appendix: Stratified Train/Test Split with File-Level Grouping

Problem: Temporal Autocorrelation and Data Leakage

Each `.dat` file in DroneDetect V2 contains approximately 2 seconds of continuous RF signal sampled at 60 MHz. During preprocessing, this signal is segmented into ~100 overlapping windows of 20ms each.

Critical observation: Consecutive segments from the same recording exhibit strong temporal autocorrelation. The RF characteristics (carrier frequency drift, hardware imperfections, environmental noise) remain largely constant within a single acquisition.

If segments from the same source file appear in both training and test sets, the model may learn to recognize recording-specific artifacts rather than generalizable drone RF signatures. This constitutes **data leakage** and leads to overly optimistic performance estimates that fail to generalize to unseen recordings.

Solution: StratifiedGroupKFold

We implement a file-grouped stratified split using `sklearn.model_selection.StratifiedGroupKFold`:

1. **Grouping constraint**: All segments from a given `.dat` file are assigned to the same split (train OR test, never both)
2. **Stratification**: Splits maintain the drone class distribution to ensure balanced representation
3. **Validation**: An assertion verifies zero file overlap between splits

This approach ensures that test set performance reflects the model's ability to generalize to entirely new recordings, providing a realistic estimate of real-world deployment accuracy.

In [14]:
```
from sklearn.model_selection import StratifiedGroupKFold
```

```python
def get_stratified_file_split(X, y, file_ids, test_size=0.2, random_state=42):
    """
    Split data at FILE level to prevent data leakage.

    Segments from the same .dat file (~100 segments) will never appear
    in both train and test sets.

    Parameters
    ----------
    X : array-like
        Features (n_samples, ...)
    y : array-like
        Labels for stratification (n_samples,)
    file_ids : array-like
        Source file ID for each sample (n_samples,)
    test_size : float
        Approximate test set proportion (actual may vary due to file grouping)
    random_state : int
        Random seed for reproducibility

    Returns
    -------
    train_idx, test_idx : arrays
        Indices for train/test split
    """
    n_splits = int(1 / test_size)  # e.g., test_size=0.2 -> 5 splits -> 1 fold = 20%

    sgkf = StratifiedGroupKFold(n_splits=n_splits, shuffle=True, random_state=random_state)

    # Take first fold as train/test split
    train_idx, test_idx = next(sgkf.split(X, y, groups=file_ids))

    # Verify no file leakage
    train_files = set(file_ids[train_idx])
    test_files = set(file_ids[test_idx])
    assert len(train_files & test_files) == 0, "Data leakage detected: files in both splits"

    return train_idx, test_idx
```

```
# Usage example (run after loading features):
# psd_data = np.load(config.FEATURES_DIR / 'iq_features.npz')
# X, y, file_ids = psd_data['X'], psd_data['y_drone'], psd_data['file_ids']
# train_idx, test_idx = get_stratified_file_split(X, y, file_ids)
# X_train, X_test = X[train_idx], X[test_idx]
# y_train, y_test = y[train_idx], y[test_idx]

print("Split function defined: get_stratified_file_split(X, y, file_ids, test_size=0.2)")
```

```
Split function defined: get_stratified_file_split(X, y, file_ids, test_size=0.2)
```