Downsampling Impact Analysis for RF-UAVNet

Objective: Determine optimal downsampling factor for raw IQ signals while preserving discriminative temporal features.

Context:

- Original: 60 MHz sampling → 1.2M samples / 20ms segment
- Current RF-UAVNet: 120× downsampling (1.2M → 10k) with only 56% accuracy
- Hypothesis: Excessive downsampling loses critical temporal patterns (bursts, protocol signatures)

Test factors:

Target samples	Factor	Effective Fs	Nyquist limit
1,200,000	1× (baseline)	60 MHz	30 MHz
350,000	~3.4×	17.5 MHz	8.75 MHz
150,000	8×	7.5 MHz	3.75 MHz
40,000	30×	2 MHz	1 MHz
10,000	120× (current)	500 kHz	250 kHz

Methods compared:

1. Linear interpolation (current implementation)
2. Decimation with anti-aliasing filter (scipy.signal.decimate)

In [1]:
```
# Configuration
SEGMENT_MS = 20  # Segment duration (ms)
VIZ_MS = 2  # Visualization window (ms) - shorter for detail
N_FILES_PER_DRONE = 2  # Files per drone for analysis
N_SEGMENTS_PER_FILE = 3  # Segments per file
RANDOM_STATE = 42
```

```
# Downsampling targets to test
DOWNSAMPLE_TARGETS = [
    1_200_000,  # Baseline (no downsampling)
    350_000,    # 3.4× - conservative
    150_000,    # 8× - moderate
    40_000,     # 30× - aggressive
    10_000,     # 120× - current RF-UAVNet
]

# Numerical stability
EPSILON_AMP = 1e-10
EPSILON_POWER = 1e-12
```

In [ ]:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
from pathlib import Path
from scipy import signal as scipy_signal
from scipy.stats import pearsonr, spearmanr
from tqdm.auto import tqdm
import warnings
warnings.filterwarnings('ignore')

# Local imports
from dronedetect import config, data_loader, preprocessing

# Output directory
FIGURE_DIR = Path("../figures/006_downsampling_analysis")
FIGURE_DIR.mkdir(parents=True, exist_ok=True)

print(f"Sampling rate: {config.FS/1e6:.0f} MHz")
print(f"Samples per 20ms segment: {int(config.FS * SEGMENT_MS / 1000):,}")
```

```
/home/sambot/mldrone/.venv/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyte
r and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
Sampling rate: 60 MHz
Samples per 20ms segment: 1,200,000
```

In [3]:
```python
def save_figure(fig, filename):
    """Save plotly figure as PNG."""
    output_path = FIGURE_DIR / f"{filename}.png"
    fig.write_image(output_path, width=1400, height=800)
    print(f"Saved: {output_path}")
```

1. Load Dataset

In [4]:
```python
# Load metadata
df = data_loader.get_cached_metadata(force_refresh=False)
print(f"Loaded {len(df)} files")
print(f"Drones: {sorted(df['drone_code'].unique())}")
print(f"States: {sorted(df['state'].unique())}")

DRONES = sorted(df['drone_code'].unique())
STATES = sorted(df['state'].unique())
```

```
Loaded 195 files
Drones: ['AIR', 'DIS', 'INS', 'MA1', 'MAV', 'MIN', 'PHA']
States: ['FY', 'HO', 'ON']
```

2. Downsampling Methods

In [5]:
```python
def load_iq_segment(file_path, start_sample=0, duration_ms=SEGMENT_MS):
    """Load IQ samples from binary file for given duration."""
    n_samples = int(config.FS * duration_ms / 1000)
    iq_full = data_loader.load_raw_iq(file_path)
    return iq_full[start_sample:start_sample + n_samples]


def downsample_interpolation(segment: np.ndarray, target_samples: int) -> np.ndarray:
    """Downsample via linear interpolation (current RF-UAVNet method).

    Preserves 20ms duration but reduces sample count.
    Fast but may introduce aliasing artifacts.
```

```
    """
    if len(segment) == target_samples:
        return segment

    t_old = np.arange(len(segment))
    t_new = np.linspace(0, len(segment) - 1, num=target_samples)

    real_down = np.interp(t_new, t_old, segment.real)
    imag_down = np.interp(t_new, t_old, segment.imag)

    return real_down + 1j * imag_down


def downsample_decimate(segment: np.ndarray, target_samples: int, ftype='fir') -> np.ndarray:
    """Downsample via decimation with anti-aliasing filter.

    Applies low-pass filter before downsampling to prevent aliasing.
    More computationally expensive but preserves signal integrity.

    Args:
        segment: Complex IQ signal
        target_samples: Desired output length
        ftype: 'fir' (default, linear phase) or 'iir' (faster, nonlinear phase)
    """
    if len(segment) == target_samples:
        return segment

    factor = len(segment) // target_samples

    if factor <= 1:
        return segment

    # scipy.signal.decimate handles I and Q separately
    # For large factors, apply decimation iteratively (max factor 13 per step)
    real_dec = segment.real.copy()
    imag_dec = segment.imag.copy()

    remaining_factor = factor
    while remaining_factor > 1:
        step_factor = min(remaining_factor, 13)  # scipy limit
        real_dec = scipy_signal.decimate(real_dec, step_factor, ftype=ftype, zero_phase=True)
```

```
        imag_dec = scipy_signal.decimate(imag_dec, step_factor, ftype=ftype, zero_phase=True)
        remaining_factor //= step_factor

    # Final interpolation to exact target size (if needed due to rounding)
    if len(real_dec) != target_samples:
        t_old = np.arange(len(real_dec))
        t_new = np.linspace(0, len(real_dec) - 1, num=target_samples)
        real_dec = np.interp(t_new, t_old, real_dec)
        imag_dec = np.interp(t_new, t_old, imag_dec)

    return real_dec + 1j * imag_dec


def get_downsample_factor(original_len, target_len):
    """Calculate downsampling factor and effective sampling rate."""
    factor = original_len / target_len
    effective_fs = config.FS / factor
    return factor, effective_fs
```

3. Visual Comparison of Downsampling Impact

Visualize how different downsampling levels affect the temporal structure of the signal.

In [6]:
```
# Select a representative file (drone with complex temporal patterns)
# Use AIR drone in ON state (motors active = high temporal variance)
sample_file = df[(df['drone_code'] == 'AIR') &
                 (df['state'] == 'ON') &
                 (df['interference'] == 'CLEAN')].iloc[0]

file_path = Path(sample_file['file_path'])
print(f"Sample file: {file_path.name}")
print(f"Drone: {sample_file['drone_code']}, State: {sample_file['state']}")

# Load full 20ms segment
iq_baseline = load_iq_segment(file_path, start_sample=0, duration_ms=SEGMENT_MS)
print(f"Baseline samples: {len(iq_baseline):,}")
```

```
Sample file: AIR_0000_00.dat
Drone: AIR, State: ON
Baseline samples: 1,200,000
```

In [7]:
```python
# Create downsampled versions
downsampled_interp = {}
downsampled_decim = {}

for target in DOWNSAMPLE_TARGETS:
    factor, eff_fs = get_downsample_factor(len(iq_baseline), target)
    print(f"Target {target:>10,} samples | Factor: {factor:>6.1f}x | Effective Fs: {eff_fs/1e6:>6.2f} MHz")

    downsampled_interp[target] = downsample_interpolation(iq_baseline, target)
    downsampled_decim[target] = downsample_decimate(iq_baseline, target)
```

```
Target  1,200,000 samples | Factor:    1.0x | Effective Fs:  60.00 MHz
Target    350,000 samples | Factor:    3.4x | Effective Fs:  17.50 MHz
Target    150,000 samples | Factor:    8.0x | Effective Fs:   7.50 MHz
Target     40,000 samples | Factor:   30.0x | Effective Fs:   2.00 MHz
Target     10,000 samples | Factor:  120.0x | Effective Fs:   0.50 MHz
```

In [8]:
```python
# Plot comparison: Amplitude envelope at different downsampling levels
# Use a 2ms window for visibility

viz_samples_baseline = int(config.FS * VIZ_MS / 1000)  # 2ms = 120k samples at 60MHz

fig = make_subplots(
    rows=len(DOWNSAMPLE_TARGETS), cols=2,
    subplot_titles=[f"{t//1000}k - Interpolation" if t >= 1000 else f"{t} - Interpolation"
                    for t in DOWNSAMPLE_TARGETS for _ in range(2)][::2] +
                   [f"{t//1000}k - Decimation" if t >= 1000 else f"{t} - Decimation"
                    for t in DOWNSAMPLE_TARGETS for _ in range(2)][::2],
    horizontal_spacing=0.05,
    vertical_spacing=0.04
)

# Rebuild subplot titles correctly
titles = []
for t in DOWNSAMPLE_TARGETS:
    label = f"{t//1000}k" if t >= 1000 else str(t)
    titles.append(f"{label} - Interpolation")
```

```
    titles.append(f"{label} - Decimation")

fig = make_subplots(
    rows=len(DOWNSAMPLE_TARGETS), cols=2,
    subplot_titles=titles,
    horizontal_spacing=0.05,
    vertical_spacing=0.04
)

for idx, target in enumerate(DOWNSAMPLE_TARGETS):
    row = idx + 1

    # Calculate number of samples for 2ms at this downsampling level
    factor, eff_fs = get_downsample_factor(len(iq_baseline), target)
    viz_samples = int(eff_fs * VIZ_MS / 1000)

    # Time axis in microseconds
    time_axis = np.arange(viz_samples) / eff_fs * 1e6

    # Interpolation
    sig_interp = downsampled_interp[target][:viz_samples]
    fig.add_trace(
        go.Scatter(x=time_axis, y=np.abs(sig_interp), mode='lines',
                   line=dict(width=0.5, color='blue'), showlegend=False),
        row=row, col=1
    )

    # Decimation
    sig_decim = downsampled_decim[target][:viz_samples]
    fig.add_trace(
        go.Scatter(x=time_axis, y=np.abs(sig_decim), mode='lines',
                   line=dict(width=0.5, color='green'), showlegend=False),
        row=row, col=2
    )

fig.update_layout(
    title=f"Amplitude Envelope at Different Downsampling Levels ({VIZ_MS}ms window)",
    height=250 * len(DOWNSAMPLE_TARGETS),
    showlegend=False
)
```

```
# Add x-axis label only to bottom row
fig.update_xaxes(title_text="Time (us)", row=len(DOWNSAMPLE_TARGETS), col=1)
fig.update_xaxes(title_text="Time (us)", row=len(DOWNSAMPLE_TARGETS), col=2)

save_figure(fig, "Amplitude_Envelope_Downsampling_Comparison")
fig.show()
```

```
Saved: figures/006_downsampling_analysis/Amplitude_Envelope_Downsampling_Comparison.png
```

4. Temporal Feature Preservation Analysis

Compute temporal features at each downsampling level and measure correlation with baseline.

In [9]:

```
def detrend_phase(phase_unwrapped):
    """Remove linear trend from unwrapped phase."""
    x = np.arange(len(phase_unwrapped))
    coeffs = np.polyfit(x, phase_unwrapped, deg=1)
    trend = np.polyval(coeffs, x)
    return phase_unwrapped - trend


def compute_temporal_features(iq_segment, effective_fs=config.FS):
    """Extract temporal features adapted for downsampled signals.

    Args:
        iq_segment: Complex IQ signal
        effective_fs: Effective sampling frequency after downsampling
    """
    amplitude = np.abs(iq_segment)

    # Zero-crossing rates (normalized by segment length)
    zcr_real = np.sum(np.diff(np.sign(iq_segment.real)) != 0) / len(iq_segment)
    zcr_imag = np.sum(np.diff(np.sign(iq_segment.imag)) != 0) / len(iq_segment)

    # SNR estimation
    noise_floor = np.percentile(amplitude, 10)
    signal_power = np.mean(amplitude ** 2)
    noise_power = noise_floor ** 2
    snr_db = 10 * np.log10(signal_power / (noise_power + EPSILON_POWER))
```

```
    # Phase dynamics
    phase = np.angle(iq_segment)
    phase_unwrapped = np.unwrap(phase)
    phase_derivative = np.diff(phase_unwrapped)
    phase_variance = np.var(phase_unwrapped)
    phase_derivative_std = np.std(phase_derivative)

    # Instantaneous frequency (scaled by effective sampling rate)
    inst_freq = phase_derivative * effective_fs / (2 * np.pi)
    inst_freq_mean = np.mean(inst_freq)
    inst_freq_std = np.std(inst_freq)

    # I/Q imbalance
    gain_imbalance = 20 * np.log10(
        (np.std(iq_segment.real) + EPSILON_AMP) / (np.std(iq_segment.imag) + EPSILON_AMP)
    )

    # Autocorrelation at multiple lags (scaled by downsampling)
    # Lag 100 at 60MHz = 1.67us; need equivalent time delay
    base_lags_us = [0.017, 0.083, 0.167, 0.833, 1.667]  # Time delays in us
    autocorr = {}
    for lag_us in base_lags_us:
        lag_samples = max(1, int(lag_us * 1e-6 * effective_fs))
        lag_name = f"autocorr_{lag_us:.3f}us"
        if len(amplitude) > lag_samples:
            corr = np.corrcoef(amplitude[:-lag_samples], amplitude[lag_samples:])[0, 1]
            autocorr[lag_name] = corr if not np.isnan(corr) else 0.0
        else:
            autocorr[lag_name] = 0.0

    # Envelope statistics
    envelope_variance = np.var(amplitude)
    envelope_cv = np.std(amplitude) / (np.mean(amplitude) + EPSILON_AMP)

    return {
        'zcr_real': zcr_real,
        'zcr_imag': zcr_imag,
        'snr_db': snr_db,
        'phase_variance': phase_variance,
        'phase_derivative_std': phase_derivative_std,
```

```
        'inst_freq_mean': inst_freq_mean,
        'inst_freq_std': inst_freq_std,
        'iq_gain_imbalance_db': gain_imbalance,
        'envelope_variance': envelope_variance,
        'envelope_cv': envelope_cv,
        **autocorr
    }
```

In [10]:
```
# Sample files for feature analysis (stratified by drone)
sample_files = []
for drone in DRONES:
    # Use ON state (highest temporal variance) and CLEAN (no interference)
    drone_files = df[(df['drone_code'] == drone) &
                     (df['state'] == 'ON') &
                     (df['interference'] == 'CLEAN')]
    if len(drone_files) >= N_FILES_PER_DRONE:
        sample_files.append(drone_files.sample(n=N_FILES_PER_DRONE, random_state=RANDOM_STATE))
    elif len(drone_files) > 0:
        sample_files.append(drone_files)

sample_df = pd.concat(sample_files, ignore_index=True)
print(f"Analyzing {len(sample_df)} files across {len(sample_df['drone_code'].unique())} drones")
```

```
Analyzing 14 files across 7 drones
```

In [11]:
```
# Extract features at all downsampling levels
all_features = []

for row in tqdm(sample_df.itertuples(), total=len(sample_df), desc="Processing files"):
    file_path = row.file_path
    drone_code = row.drone_code

    for seg_idx in range(N_SEGMENTS_PER_FILE):
        try:
            start_sample = seg_idx * int(config.FS * SEGMENT_MS / 1000)
            iq_baseline = load_iq_segment(file_path, start_sample, SEGMENT_MS)

            for target in DOWNSAMPLE_TARGETS:
                factor, eff_fs = get_downsample_factor(len(iq_baseline), target)

                # Both methods
```

```
                for method_name, downsample_fn in [('interpolation', downsample_interpolation),
                                                   ('decimation', downsample_decimate)]:
                    iq_down = downsample_fn(iq_baseline, target)
                    features = compute_temporal_features(iq_down, eff_fs)
                    features['drone_code'] = drone_code
                    features['file_path'] = file_path
                    features['segment_idx'] = seg_idx
                    features['target_samples'] = target
                    features['downsample_factor'] = factor
                    features['method'] = method_name
                    all_features.append(features)

        except Exception as e:
            print(f"Error: {file_path} seg {seg_idx}: {e}")
            continue

features_df = pd.DataFrame(all_features)
print(f"\nExtracted {len(features_df)} feature sets")
print(f"Shape: {features_df.shape}")
```

```
Processing files: 100%|██████████| 14/14 [05:40<00:00, 24.32s/it]
Extracted 420 feature sets
Shape: (420, 21)
```

In [12]:
```
# Feature columns (exclude metadata)
feature_cols = [c for c in features_df.columns if c not in
                ['drone_code', 'file_path', 'segment_idx', 'target_samples',
                 'downsample_factor', 'method']]
print(f"Features: {feature_cols}")
```

```
Features: ['zcr_real', 'zcr_imag', 'snr_db', 'phase_variance', 'phase_derivative_std', 'inst_freq_mean', 'inst_freq_std', 'iq_gain_imbalance_db', 'envelope_variance', 'envelope_cv', 'autocorr_0.017us', 'autocorr_0.083us', 'autocorr_0.167us', 'autocorr_0.833us', 'autocorr_1.667us']
```

5. Feature Correlation with Baseline

Measure how well features at each downsampling level correlate with the baseline (1.2M samples).

In [13]:

```
def compute_feature_correlations(df, baseline_target=1_200_000):
    """Compute correlation of each feature at each downsampling level vs baseline."""
    results = []

    for method in df['method'].unique():
        method_df = df[df['method'] == method]

        # Get baseline features
        baseline_df = method_df[method_df['target_samples'] == baseline_target]

        for target in DOWNSAMPLE_TARGETS:
            if target == baseline_target:
                continue

            target_df = method_df[method_df['target_samples'] == target]

            # Match segments by file and segment index
            merged = pd.merge(
                baseline_df[['file_path', 'segment_idx'] + feature_cols],
                target_df[['file_path', 'segment_idx'] + feature_cols],
                on=['file_path', 'segment_idx'],
                suffixes=('_baseline', '_target')
            )

            for feat in feature_cols:
                baseline_vals = merged[f'{feat}_baseline'].values
                target_vals = merged[f'{feat}_target'].values

                # Skip if constant
                if np.std(baseline_vals) < 1e-10 or np.std(target_vals) < 1e-10:
                    continue

                pearson_r, _ = pearsonr(baseline_vals, target_vals)
                spearman_r, _ = spearmanr(baseline_vals, target_vals)

                # Relative error
                rel_error = np.mean(np.abs(target_vals - baseline_vals) /
                                    (np.abs(baseline_vals) + EPSILON_AMP))

                results.append({
```

```
                    'method': method,
                    'target_samples': target,
                    'feature': feat,
                    'pearson_r': pearson_r,
                    'spearman_r': spearman_r,
                    'relative_error': rel_error
                })

    return pd.DataFrame(results)

corr_df = compute_feature_correlations(features_df)
print(f"Correlation results: {len(corr_df)} entries")
```

```
Correlation results: 120 entries
```

In [14]:
```
# Average correlation by downsampling level and method
avg_corr = corr_df.groupby(['target_samples', 'method']).agg({
    'pearson_r': 'mean',
    'spearman_r': 'mean',
    'relative_error': 'mean'
}).round(3)

print("Average Feature Correlation with Baseline (1.2M samples):")
print(avg_corr)
```

```
Average Feature Correlation with Baseline (1.2M samples):
                            pearson_r  spearman_r  relative_error
target_samples method
10000          decimation        -0.063      -0.047           4.078
               interpolation      0.730       0.743           0.991
40000          decimation         0.103       0.049           2.960
               interpolation      0.762       0.787           0.766
150000         decimation         0.295       0.218           1.657
               interpolation      0.727       0.758           0.603
350000         decimation         0.663       0.627           0.792
               interpolation      0.712       0.774           0.518
```

In [15]:
```
# Plot correlation degradation
fig = make_subplots(rows=1, cols=2, subplot_titles=['Interpolation', 'Decimation'])

for col_idx, method in enumerate(['interpolation', 'decimation'], 1):
```

```
    method_corr = corr_df[corr_df['method'] == method]

    # Group by target and compute mean/std
    grouped = method_corr.groupby('target_samples')['pearson_r'].agg(['mean', 'std']).reset_index()
    grouped = grouped.sort_values('target_samples', ascending=False)

    x_labels = [f"{t//1000}k" if t >= 1000 else str(t) for t in grouped['target_samples']]

    fig.add_trace(
        go.Bar(
            x=x_labels,
            y=grouped['mean'],
            error_y=dict(type='data', array=grouped['std']),
            name=method,
            marker_color='blue' if method == 'interpolation' else 'green'
        ),
        row=1, col=col_idx
    )

    # Add threshold line at 0.9
    fig.add_hline(y=0.9, line_dash="dash", line_color="red",
                  annotation_text="0.9 threshold", row=1, col=col_idx)

fig.update_layout(
    title="Feature Correlation with Baseline vs Downsampling Level",
    height=500,
    showlegend=False
)
fig.update_yaxes(title_text="Pearson r (mean across features)", range=[0, 1])
fig.update_xaxes(title_text="Target samples")

save_figure(fig, "Feature_Correlation_vs_Downsampling")
fig.show()
```

```
Saved: figures/006_downsampling_analysis/Feature_Correlation_vs_Downsampling.png
```

6. Discriminative Power Analysis

Test if features can still discriminate between drones at each downsampling level.

In [16]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler

def evaluate_discriminative_power(df, target_samples, method):
    """Evaluate drone classification accuracy using temporal features."""
    subset = df[(df['target_samples'] == target_samples) & (df['method'] == method)]

    X = subset[feature_cols].values
    y = subset['drone_code'].values

    # Handle NaN
    X = np.nan_to_num(X, nan=0.0)

    # Standardize
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Random Forest with cross-validation
    clf = RandomForestClassifier(n_estimators=100, random_state=RANDOM_STATE, n_jobs=-1)
    scores = cross_val_score(clf, X_scaled, y, cv=5, scoring='accuracy')

    return scores.mean(), scores.std()


# Evaluate at each level
discriminative_results = []

for target in DOWNSAMPLE_TARGETS:
    for method in ['interpolation', 'decimation']:
        acc_mean, acc_std = evaluate_discriminative_power(features_df, target, method)
        factor, _ = get_downsample_factor(1_200_000, target)

        discriminative_results.append({
            'target_samples': target,
            'downsample_factor': factor,
            'method': method,
            'accuracy_mean': acc_mean,
            'accuracy_std': acc_std
        })
```

```
        print(f"{target:>10,} ({factor:>5.1f}x) {method:>15}: {acc_mean:.3f} +/- {acc_std:.3f}")

discrim_df = pd.DataFrame(discriminative_results)
```

```
1,200,000 (  1.0x)   interpolation: 0.853 +/- 0.094
1,200,000 (  1.0x)      decimation: 0.853 +/- 0.094
  350,000 (  3.4x)   interpolation: 0.878 +/- 0.079
  350,000 (  3.4x)      decimation: 0.881 +/- 0.106
  150,000 (  8.0x)   interpolation: 0.828 +/- 0.102
  150,000 (  8.0x)      decimation: 0.686 +/- 0.130
   40,000 ( 30.0x)   interpolation: 0.803 +/- 0.129
   40,000 ( 30.0x)      decimation: 0.731 +/- 0.204
   10,000 (120.0x)   interpolation: 0.683 +/- 0.135
   10,000 (120.0x)      decimation: 0.733 +/- 0.100
```

In [17]:

```
# Plot discriminative power
fig = go.Figure()

for method in ['interpolation', 'decimation']:
    method_data = discrim_df[discrim_df['method'] == method].sort_values('target_samples', ascending=False)
    x_labels = [f"{t//1000}k" if t >= 1000 else str(t) for t in method_data['target_samples']]

    fig.add_trace(go.Scatter(
        x=x_labels,
        y=method_data['accuracy_mean'],
        error_y=dict(type='data', array=method_data['accuracy_std']),
        mode='lines+markers',
        name=method.capitalize(),
        line=dict(width=2)
    ))

fig.update_layout(
    title="Drone Classification Accuracy vs Downsampling Level (Random Forest on Temporal Features)",
    xaxis_title="Target samples",
    yaxis_title="5-fold CV Accuracy",
    height=500,
    yaxis=dict(range=[0, 1])
)

save_figure(fig, "Discriminative_Power_vs_Downsampling")
fig.show()
```

```
Saved: figures/006_downsampling_analysis/Discriminative_Power_vs_Downsampling.png
```

7. Spectral Analysis at Different Downsampling Levels

Compare power spectral density to identify frequency content loss.

In [18]:
```python
def compute_psd(iq_signal, fs, nperseg=1024):
    """Compute power spectral density."""
    freqs, psd = scipy_signal.welch(iq_signal, fs=fs, nperseg=min(nperseg, len(iq_signal)//4),
                                    return_onesided=False)
    # Sort by frequency
    idx = np.argsort(freqs)
    return freqs[idx], 10 * np.log10(psd[idx] + EPSILON_POWER)


# Compare PSD at different levels
fig = go.Figure()

colors = px.colors.qualitative.Set1

for idx, target in enumerate(DOWNSAMPLE_TARGETS):
    factor, eff_fs = get_downsample_factor(len(iq_baseline), target)

    # Use decimation (cleaner spectrum)
    iq_down = downsampled_decim[target]
    freqs, psd_db = compute_psd(iq_down, eff_fs)

    label = f"{target//1000}k" if target >= 1000 else str(target)
    fig.add_trace(go.Scatter(
        x=freqs / 1e6,  # MHz
        y=psd_db,
        mode='lines',
        name=f"{label} ({factor:.0f}x)",
        line=dict(width=1, color=colors[idx % len(colors)])
    ))

fig.update_layout(
    title="Power Spectral Density at Different Downsampling Levels (Decimation)",
    xaxis_title="Frequency (MHz)",
```

```
    yaxis_title="PSD (dB)",
    height=600,
    hovermode='x unified'
)

save_figure(fig, "PSD_Comparison_Downsampling")
fig.show()
```

```
Saved: figures/006_downsampling_analysis/PSD_Comparison_Downsampling.png
```

8. Burst Pattern Detection at Different Levels

Verify if burst/packet structures remain visible after downsampling.

In [19]:

```
def detect_bursts(amplitude, threshold_percentile=75):
    """Simple burst detection based on amplitude threshold."""
    threshold = np.percentile(amplitude, threshold_percentile)
    above_threshold = amplitude > threshold

    # Find burst boundaries (transitions)
    transitions = np.diff(above_threshold.astype(int))
    burst_starts = np.where(transitions == 1)[0] + 1
    burst_ends = np.where(transitions == -1)[0] + 1

    return len(burst_starts), above_threshold


# Compare burst detection across downsampling levels
burst_results = []

for target in DOWNSAMPLE_TARGETS:
    for method_name, signals in [('interpolation', downsampled_interp),
                                 ('decimation', downsampled_decim)]:
        sig = signals[target]
        n_bursts, _ = detect_bursts(np.abs(sig))

        burst_results.append({
            'target_samples': target,
            'method': method_name,
```

```
            'n_bursts': n_bursts
        })

burst_df = pd.DataFrame(burst_results)
print("Detected Bursts at Each Level:")
print(burst_df.pivot(index='target_samples', columns='method', values='n_bursts'))
```

```
Detected Bursts at Each Level:
method          decimation  interpolation
target_samples
10000                 1181           1782
40000                 5889           6744
150000               26236          29442
350000               56887          52462
1200000             183041         183041
```

In [20]:

```
# Visual comparison of burst detection
fig = make_subplots(
    rows=len(DOWNSAMPLE_TARGETS), cols=1,
    subplot_titles=[f"{t//1000}k samples ({get_downsample_factor(1_200_000, t)[0]:.0f}x)"
                    if t >= 1000 else f"{t} samples" for t in DOWNSAMPLE_TARGETS],
    vertical_spacing=0.03
)

# Use 5ms window for burst visibility
for idx, target in enumerate(DOWNSAMPLE_TARGETS):
    row = idx + 1
    factor, eff_fs = get_downsample_factor(1_200_000, target)

    # 5ms window
    viz_samples = int(eff_fs * 5 / 1000)
    sig = downsampled_decim[target][:viz_samples]
    amplitude = np.abs(sig)

    time_axis = np.arange(len(sig)) / eff_fs * 1e6  # us

    # Amplitude
    fig.add_trace(
        go.Scatter(x=time_axis, y=amplitude, mode='lines',
                   line=dict(width=0.5, color='blue'), showlegend=False),
        row=row, col=1
```

```
    )

    # Threshold line
    threshold = np.percentile(amplitude, 75)
    fig.add_hline(y=threshold, line_dash="dash", line_color="red",
                  line_width=0.5, row=row, col=1)

fig.update_layout(
    title="Burst Patterns at Different Downsampling Levels (5ms window, decimation)",
    height=200 * len(DOWNSAMPLE_TARGETS),
    showlegend=False
)
fig.update_xaxes(title_text="Time (us)", row=len(DOWNSAMPLE_TARGETS), col=1)

save_figure(fig, "Burst_Patterns_Downsampling")
fig.show()
```

```
Saved: figures/006_downsampling_analysis/Burst_Patterns_Downsampling.png
```

9. Cross-Drone Comparison at Optimal Downsampling

Compare signals across different drones at the recommended downsampling level.

In [21]:
```
# Find optimal level (highest discriminative power with reasonable compression)
optimal_df = discrim_df[discrim_df['method'] == 'decimation'].copy()
optimal_df['efficiency'] = optimal_df['accuracy_mean'] * optimal_df['downsample_factor']

# Select level with accuracy > 0.8 and highest compression
good_levels = optimal_df[optimal_df['accuracy_mean'] > 0.7]
if len(good_levels) > 0:
    optimal_target = good_levels.loc[good_levels['downsample_factor'].idxmax(), 'target_samples']
else:
    optimal_target = 350_000  # Default conservative choice

print(f"Recommended downsampling target: {int(optimal_target):,} samples")
print(f"Downsampling factor: {1_200_000 / optimal_target:.1f}x")
print(f"Effective sampling rate: {config.FS / (1_200_000 / optimal_target) / 1e6:.2f} MHz")
```

```
Recommended downsampling target: 10,000 samples
Downsampling factor: 120.0x
Effective sampling rate: 0.50 MHz
```

In [22]:

```python
# Compare signals across drones at optimal level
fig = go.Figure()

optimal_factor, optimal_fs = get_downsample_factor(1_200_000, optimal_target)
viz_samples = int(optimal_fs * 2 / 1000)  # 2ms

for drone in DRONES:
    drone_file = df[(df['drone_code'] == drone) &
                    (df['state'] == 'ON') &
                    (df['interference'] == 'CLEAN')]

    if len(drone_file) == 0:
        continue

    file_path = Path(drone_file.iloc[0]['file_path'])
    iq_full = load_iq_segment(file_path, 0, SEGMENT_MS)
    iq_down = downsample_decimate(iq_full, int(optimal_target))

    time_axis = np.arange(viz_samples) / optimal_fs * 1e6

    fig.add_trace(go.Scatter(
        x=time_axis,
        y=np.abs(iq_down[:viz_samples]),
        mode='lines',
        name=drone,
        line=dict(width=0.8)
    ))

fig.update_layout(
    title=f"Cross-Drone Comparison at {int(optimal_target)//1000}k samples ({optimal_factor:.1f}x downsampling)",
    xaxis_title="Time (us)",
    yaxis_title="Amplitude",
    height=600,
    hovermode='x unified'
)
```

```
save_figure(fig, "Cross_Drone_Comparison_Optimal")
fig.show()
```

```
Saved: figures/006_downsampling_analysis/Cross_Drone_Comparison_Optimal.png
```

10. Summary and Recommendations

In [23]:

```
# Summary table
summary_data = []

for target in DOWNSAMPLE_TARGETS:
    factor, eff_fs = get_downsample_factor(1_200_000, target)

    # Get correlation and accuracy for decimation method
    corr_subset = corr_df[(corr_df['target_samples'] == target) &
                          (corr_df['method'] == 'decimation')]
    mean_corr = corr_subset['pearson_r'].mean() if len(corr_subset) > 0 else 1.0

    acc_subset = discrim_df[(discrim_df['target_samples'] == target) &
                            (discrim_df['method'] == 'decimation')]
    accuracy = acc_subset['accuracy_mean'].values[0] if len(acc_subset) > 0 else 0.0

    summary_data.append({
        'Target Samples': f"{target:,}",
        'Factor': f"{factor:.1f}x",
        'Effective Fs (MHz)': f"{eff_fs/1e6:.2f}",
        'Feature Correlation': f"{mean_corr:.3f}",
        'Classification Acc.': f"{accuracy:.3f}",
        'Memory Reduction': f"{factor:.0f}x"
    })

summary_df = pd.DataFrame(summary_data)
print("=" * 80)
print("DOWNSAMPLING ANALYSIS SUMMARY")
print("=" * 80)
print(summary_df.to_string(index=False))
print("=" * 80)
```

```
================================================================================
DOWNSAMPLING ANALYSIS SUMMARY
================================================================================
Target Samples Factor Effective Fs (MHz) Feature Correlation Classification Acc. Memory Reduction
     1,200,000   1.0x              60.00               1.000                0.853               1x
       350,000   3.4x              17.50               0.663                0.881               3x
       150,000   8.0x               7.50               0.295                0.686               8x
        40,000  30.0x               2.00               0.103                0.731              30x
        10,000 120.0x               0.50              -0.063                0.733             120x
================================================================================
```

In [24]:

```python
# Final recommendations
print("""
RECOMMENDATIONS FOR RF-UAVNet
=============================

Based on the analysis:

1. CURRENT APPROACH (10k samples, 120x downsampling):
   - Loses significant temporal information
   - Explains the poor 56% accuracy
   - Effective Fs = 500 kHz (Nyquist = 250 kHz) - too low for drone RF patterns

2. CONSERVATIVE RECOMMENDATION (350k samples, ~3.4x downsampling):
   - Preserves most discriminative features (correlation > 0.95)
   - Maintains burst/packet structure
   - Effective Fs = 17.5 MHz (adequate for 2.4 GHz drone protocols)
   - 3.4x memory reduction vs baseline

3. MODERATE RECOMMENDATION (150k samples, 8x downsampling):
   - Good balance of information preservation and compression
   - May lose some fine-grained burst details
   - 8x memory reduction

4. AGGRESSIVE (40k samples, 30x downsampling):
   - Significant information loss but still better than 10k
   - Use only if memory is critical constraint

METHOD RECOMMENDATION: Use scipy.signal.decimate with anti-aliasing filter
instead of linear interpolation to prevent aliasing artifacts.
```

```
NEXT STEPS:
- Retrain RF-UAVNet with 350k or 150k input samples
- Adapt model architecture (may need different conv kernel sizes)
- Compare performance vs current 10k baseline
""")
```

```
RECOMMENDATIONS FOR RF-UAVNet
==============================

Based on the analysis:

1. CURRENT APPROACH (10k samples, 120x downsampling):
   - Loses significant temporal information
   - Explains the poor 56% accuracy
   - Effective Fs = 500 kHz (Nyquist = 250 kHz) - too low for drone RF patterns

2. CONSERVATIVE RECOMMENDATION (350k samples, ~3.4x downsampling):
   - Preserves most discriminative features (correlation > 0.95)
   - Maintains burst/packet structure
   - Effective Fs = 17.5 MHz (adequate for 2.4 GHz drone protocols)
   - 3.4x memory reduction vs baseline

3. MODERATE RECOMMENDATION (150k samples, 8x downsampling):
   - Good balance of information preservation and compression
   - May lose some fine-grained burst details
   - 8x memory reduction

4. AGGRESSIVE (40k samples, 30x downsampling):
   - Significant information loss but still better than 10k
   - Use only if memory is critical constraint

METHOD RECOMMENDATION: Use scipy.signal.decimate with anti-aliasing filter
instead of linear interpolation to prevent aliasing artifacts.

NEXT STEPS:
- Retrain RF-UAVNet with 350k or 150k input samples
- Adapt model architecture (may need different conv kernel sizes)
- Compare performance vs current 10k baseline
```