```
In [1]: import os
        import pandas as pd
        import numpy as np
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score
        from scipy.stats import ks_2samp
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
        sns.set(style='whitegrid')
```

```
In [2]: # Load and preprocess AI4I like notebook 05
        path = os.path.join('..','data','ai4i','ai4i2020.csv')
        df = pd.read_csv(path)
        df.columns = df.columns.str.strip().str.lower().str.replace(r'[^0-9a-z]+','_', regex=True).str.strip('_')
        if 'machine_failure' not in df.columns:
            raise RuntimeError('machine_failure not found')
        df['machine_failure'] = df['machine_failure'].astype(int)
        order_col = 'udi' if 'udi' in df.columns else df.columns[0]
        data = df.select_dtypes(include=[np.number]).dropna(subset=['machine_failure']).reset_index(drop=True)
        data = data.sort_values(order_col).reset_index(drop=True)
        n = len(data); train_end = int(0.6 * n)
        train = data.iloc[:train_end].copy(); stream = data.iloc[train_end:].copy()
        X_train = train.drop(columns=['machine_failure', order_col]); y_train = train['machine_failure']
        X_stream = stream.drop(columns=['machine_failure', order_col]); y_stream = stream['machine_failure']
        # baseline model
        RANDOM_STATE = 42
        base_model = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=RANDOM_STATE, n_jobs=-1)
        base_model.fit(X_train, y_train)
        print('Baseline trained')
```

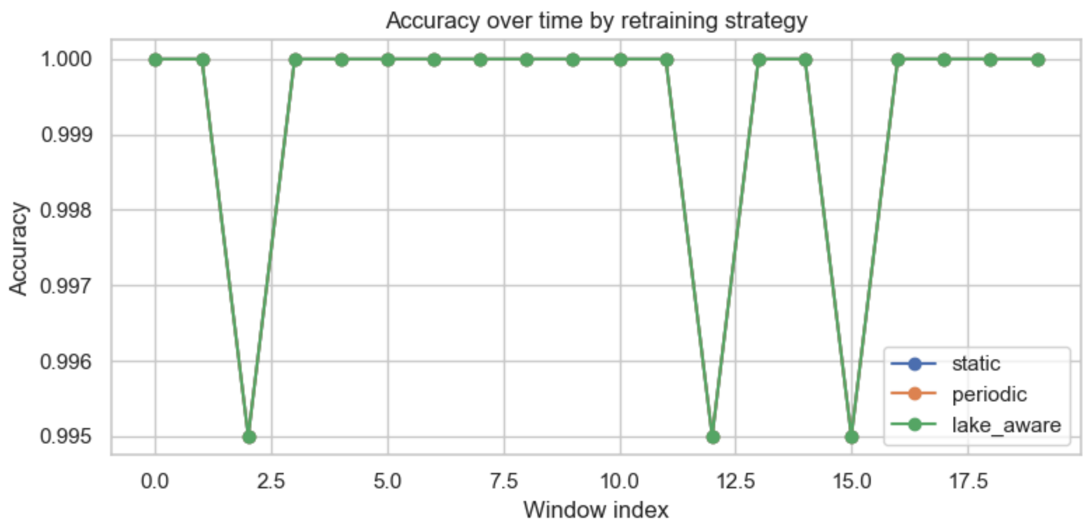
```
Baseline trained
```

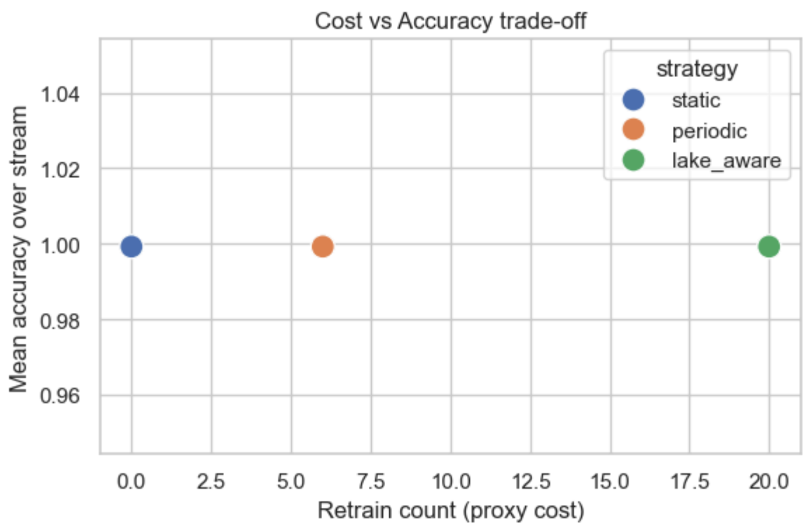
```
In [3]: # Helpers for simulated drift and KS-based detector
        drift_feats = [f for f in ['air_temperature_k','process_temperature_k'] if f in X_train.columns]
        def ks_score(batch):
            vals = []
            for f in drift_feats:
                vals.append(ks_2samp(X_train[f].values, batch[f].values).statistic)
            return np.mean(vals) if vals else 0.0

        def apply_gauss(X, feats, sigma_factor, rs=RANDOM_STATE):
            Xp = X.copy(); rng = np.random.RandomState(rs)
            for f in feats:
                sigma = Xp[f].std() * sigma_factor
                Xp[f] = Xp[f] + rng.normal(0, sigma, size=len(Xp))
            return Xp
```

```
In [4]: # Simulation parameters
        window_size = max(50, int(0.05 * len(X_stream)))
        threshold = 0.1
        n_windows = int(np.ceil(len(X_stream)/window_size))
        # strategies state holders
        results = {'static':[], 'periodic':[], 'lake_aware':[]}
        # prepare models for each strategy
        model_static = base_model
        model_periodic = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=RANDOM_STATE, n_jobs=-1)
        model_periodic.fit(X_train, y_train)
        model_lake = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=RANDOM_STATE, n_jobs=-1)
        model_lake.fit(X_train, y_train)
        # periodic retrain interval (windows)
        period = 3
        retrain_counts = {'static':0, 'periodic':0, 'lake_aware':0}
        # iterate windows and optionally inject simulated drift into certain windows
        for w in range(n_windows):
            start = w * window_size; end = min(len(X_stream), (w+1)*window_size)
            Xw = X_stream.iloc[start:end].copy(); yw = y_stream.iloc[start:end]
            # for this experiment inject moderate gaussian drift on window indices divisible by 4
            if w % 4 == 0:
                Xw_d = apply_gauss(Xw, drift_feats, sigma_factor=1.0, rs=RANDOM_STATE + w)
            else:
                Xw_d = Xw.copy()
            # static: evaluate without retraining
            ypred_s = model_static.predict(Xw_d)
            results['static'].append(accuracy_score(yw, ypred_s))
            # periodic: retrain every `period` windows using accumulated data
            if w % period == 0 and w>0:
```

```
        # retrain on all seen stream data (simple simulation)
        seen = X_stream.iloc[:end]; seen_y = y_stream.iloc[:end]
        model_periodic = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=RANDOM_STATE, n_jobs=-
        model_periodic.fit(pd.concat([X_train, seen]), pd.concat([y_train, seen_y]))
        retrain_counts['periodic'] += 1
    ypred_p = model_periodic.predict(Xw_d)
    results['periodic'].append(accuracy_score(yw, ypred_p))
    # lake-aware: compute KS and retrain only when above threshold
    score = ks_score(Xw_d)
    if score > threshold:
        seen = X_stream.iloc[:end]; seen_y = y_stream.iloc[:end]
        model_lake = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=RANDOM_STATE, n_jobs=-1)
        model_lake.fit(pd.concat([X_train, seen]), pd.concat([y_train, seen_y]))
        retrain_counts['lake_aware'] += 1
    ypred_l = model_lake.predict(Xw_d)
    results['lake_aware'].append(accuracy_score(yw, ypred_l))

# collect results into dataframe
time_idx = list(range(n_windows))
res_df = pd.DataFrame({'window':time_idx, 'static_acc':results['static'], 'periodic_acc':results['periodic'], 'lake_
print('Retrain counts:', retrain_counts)
os.makedirs(os.path.join('..','results','figures'), exist_ok=True)
res_df.to_csv(os.path.join('..','results','tables','08_retraining_accuracy_over_time.csv'), index=False)
# Plot accuracy over time for the three strategies
plt.figure(figsize=(8,4))
plt.plot(res_df['window'], res_df['static_acc'], marker='o', label='static')
plt.plot(res_df['window'], res_df['periodic_acc'], marker='o', label='periodic')
plt.plot(res_df['window'], res_df['lake_acc'], marker='o', label='lake_aware')
plt.xlabel('Window index')
plt.ylabel('Accuracy')
plt.title('Accuracy over time by retraining strategy')
plt.legend()
plt.tight_layout()
plt.savefig(os.path.join('..','results','figures','08_accuracy_over_time.png'), dpi=300)
plt.show()
# Cost vs accuracy trade-off: cost = number of retrains (proxy)
costs = {'static':0, 'periodic':retrain_counts['periodic'], 'lake_aware':retrain_counts['lake_aware']}
final_acc = {'static':res_df['static_acc'].mean(), 'periodic':res_df['periodic_acc'].mean(), 'lake_aware':res_df['l
cost_df = pd.DataFrame([{'strategy':k, 'retrain_count':v, 'mean_accuracy':final_acc[k]} for k,v in costs.items()])
cost_df.to_csv(os.path.join('..','results','tables','08_retrain_costs.csv'), index=False)
plt.figure(figsize=(6,4))
sns.scatterplot(data=cost_df, x='retrain_count', y='mean_accuracy', hue='strategy', s=150)
plt.xlabel('Retrain count (proxy cost)')
plt.ylabel('Mean accuracy over stream')
plt.title('Cost vs Accuracy trade-off')
plt.tight_layout()
plt.savefig(os.path.join('..','results','figures','08_cost_vs_accuracy.png'), dpi=300)
plt.show()
```

```
Retrain counts: {'static': 0, 'periodic': 6, 'lake_aware': 20}
```





Notes

- Lake-aware retraining triggers retraining only when distributional shift (KS) exceeds a threshold, reducing unnecessary retrains.
- Design choices (window size, threshold) are configurable and should be tuned for production.