Isolation Forest (IF) and **Robust Random Cut Forest (RRCF)** are both tree-based, unsupervised models for anomaly detection, but they differ in how they build and score their trees:

1. Tree Construction

- **Isolation Forest**
  - Builds an ensemble of **binary trees**.
  - At each node, it randomly selects one feature and then randomly picks a split value between the feature's minimum and maximum in the current subset.
  - Recursively partitions until each instance is isolated in its own leaf or a maximum tree depth is reached.
  - Because anomalies tend to be "few and different," they require **fewer splits** to isolate, resulting in shorter path lengths in the tree.
- **RRCF**
  - Builds an ensemble of **cut trees** (a variant of random trees).
  - At each cut, it selects a **random dimension** and then cuts at a **random location** within the bounding box of the points in that node—effectively slicing the data space into two.
  - Continues cutting until each point is isolated or a stopping criterion is met.
  - The process preserves more global structure in the tree's "mass" or "cardinality" at each node.

2. Anomaly Scoring

- **Isolation Forest**
  - Score is based on the **average path length** from the root to the leaf across all trees.
  - **Shorter average path lengths** ⇒ more anomalous.
  - Scores are normalized to lie roughly in [0, 1].
- **RRCF**
  - Uses the concept of **"co-dispersion"** or **"collusive displacement."**
  - For each point, it measures the **change in tree structure** (specifically, the average subtree "mass" or branch cardinality) when that point is removed.
  - **Larger displacement** ⇒ more anomalous, because the point's removal drastically alters the tree.

3. Handling of Streaming Data

- **Isolation Forest**
  - Originally designed for static datasets; can be adapted to streams by periodically retraining or using sliding windows.
  - Doesn't inherently support online updates without retraining.
- **RRCF**

- Designed with **streaming in mind**: supports **incremental updates**—you can **insert** and **delete** points in cut trees in O
(
log

n
)

per operation.
- Makes it well suited for concept-drifting data and real-time anomaly scoring.

4. Practical Implications

Aspect	Isolation Forest	RRCF
Model updates	Retrain on new data (batch)	Online insert/delete operations
Score interpretation	Short path ⇒ anomaly	Large tree-structure displacement ⇒ anomaly
Complexity per tree	O (n log n) to build, O (log n) per score	O (n log n) to build, O (log n) per update
Use case	Static or slowly evolving datasets	High-throughput streams with concept drift

In summary:

- **Isolation Forest** isolates anomalies by how quickly random splits separate them—great for static data.
- **RRCF** measures how much each point "holds together" the forest structure, allowing efficient online updates—ideal for streaming anomaly detection with evolving distributions.

Hierarchical Temporal Memory (HTM) is a biologically inspired framework for sequence learning and anomaly detection, developed by Numenta. At its core, HTM attempts to mimic key structural and algorithmic properties of the neocortex, notably: sparse representations, temporal sequence memory, and continuous online learning. Here's how it works and why it's used in NAB:

1. Key Components

1. **Encoding**
   - Raw inputs (e.g. a time-series value) are first passed through an **encoder** that produces a Sparse Distributed Representation (SDR): a high-dimensional, binary vector with a small percentage of bits active.
   - Encoders preserve semantic similarity: similar inputs yield overlapping SDRs.
2. **Spatial Pooler**
   - Learns a set of stable, sparse patterns (columns) that consistently represent the input SDRs.
   - Given a new SDR, the Spatial Pooler "activates" a subset of its columns that best match the input, enforcing fixed sparsity and boosting rarely used columns.
3. **Temporal Memory**
   - Models sequences by connecting active columns over time.
   - Each spatial column contains multiple "cells." When an SDR is fed in:
     - **Predictive cells** (those that were correctly anticipating the current input based on prior context) become active.
     - **Bursting** occurs when there's no strong prediction: all cells in a column activate, signaling surprise.
   - Synaptic permanence values are adjusted online to reinforce transitions that correctly predict the next input.
4. **Anomaly Scoring**
   - At each time step, HTM calculates an **anomaly score**:
     score
     t

     =
     1
     −
     ∣
     predicted_active_cells
     t

     ∣

|
active_cells
t

|

.

- A score near 0 means “well-predicted” (normal); near 1 means “highly surprising” (potential anomaly).
- NAB wraps this with an **anomaly likelihood** calculation that smooths scores over time and estimates how unusual a given score is relative to its recent history.

2. Strengths

- **Fully Online & Unsupervised.** Learns continually with no batch retraining.
- **Context-Aware.** Captures both short- and long-term temporal dependencies via its dendritic prediction mechanism.
- **Robust to Noise.** Sparse representations and bursting help distinguish true novelty from minor fluctuations.
- **No Explicit Distributional Assumptions.** Unlike statistical methods, HTM does not assume any form for the data’s distribution.

3. Limitations

- **Hyperparameter Sensitivity.** Performance depends on encoder settings, column count, permanence thresholds, etc.
- **Computational Overhead.** Maintaining predictive states and permanence updates can be heavier than simple statistical thresholds—especially in very high-frequency streams.
- **Opaque Mechanics.** While grounded in neuroscience, the internal representations are less interpretable than, say, EVT’s quantile thresholds.

4. HTM in the NAB Context

- **Native Implementation.** NAB includes a reference HTM detector (nupic library) tuned to maximize NAB’s early-detection score.
- **Scoring Metric Alignment.** NAB rewards detectors for raising alarms shortly after true anomalies (but penalizes false positives), which aligns well with HTM’s anomaly likelihood smoothing.
- **Benchmark Diversity.** HTM has shown strong performance on certain

classes of temporal patterns (seasonal, trending, repeated signals), making it a valuable baseline alongside more geometric or statistical rivals.

Why Include HTM in Your Comparison?

- It represents a **sequence-prediction** paradigm, contrasting with SPOT/DSPOT's **extreme-value, tail-modeling**approach.
- Its continuous, adaptive learning highlights whether modeling temporal context yields advantages over purely threshold-based detectors.
- Comparing against HTM under the same NAB scoring reveals strengths and weaknesses in early-warning capability across different anomaly types (point spikes vs. contextual drifts).

By evaluating SPOT/DSPOT alongside HTM, RRCF, and ADWIN+IF, you'll give a **comprehensive view** of how statistical tail models fare against both predictive-model and tree-based anomaly paradigms in streaming settings.

That's a sound and well-motivated baseline suite—it covers three complementary paradigms in streaming anomaly detection:

1. **RRCF (Robust Random Cut Forest)**
    - **Why it's important:** A pure streaming-native tree model with built-in online insert/delete, so it represents the state of the art for real-time, nonparametric anomaly scoring.
    - **What you'll learn:** How SPOT/DSPOT stack up against a method that "remembers" the full data geometry in an efficient, incremental way.
2. **HTM (Hierarchical Temporal Memory), the NAB "native" method**
    - **Why it's important:** NAB's own reference implementation, tuned for early detection on their benchmarks. It leverages sequence modeling and temporal prediction to flag deviations.
    - **What you'll learn:** Whether an EVT-based tail model outperforms a predictive, biologically inspired approach on the same data and scoring rules.
3. **ADWIN + Isolation Forest**
    - **Why it's important:** Combines an online change detector (ADWIN) to segment the stream with a classic static anomaly scorer (Isolation Forest) retrained periodically.
    - **What you'll learn:** How much of the benefit of streaming methods comes simply from "windowing" static detectors versus from true online updates.

Strengths of this lineup

- **Diverse mechanisms:** You cover density-based tree isolation (RRCF), temporal sequence modeling (HTM), and sliding-window adaptation of an established static method (ADWIN+IF), giving a panoramic view of contemporary approaches.
- **Fair apples-to-apples:** All three can be evaluated under NAB's real-time scoring, so you're comparing their true streaming performance—latency, false-positive control, and early-warning scores.
- **Insight into adaptation costs:** By including ADWIN+Isolation Forest, you'll quantify how much overhead (retraining cost, detection delay) static methods pay to become "streaming-capable."

Recommendations for execution

1. **Hyperparameter tuning protocol**
    - Use the same "warm-up" period on each NAB time series.
    - Tune forest sizes (# trees, tree depth) and window parameters (ADWIN's δ, IF's subsample size) via held-out segments.
2. **Compute & timing measurements**
    - Measure not just detection accuracy but also per-point update latency and memory footprint—streaming methods live or die by efficiency.
3. **Ablation on ADWIN trigger**
    - Try different retraining frequencies (e.g. every change detected vs. fixed-interval retrain) to show how sensitive the static+window approach is to that design choice.
4. **Visual comparisons**
    - Plot example segments where each method raises an alarm: this concretely illustrates the "early vs. late" trade-offs NAB scores.

Bottom line:
Your three-method comparison is both rigorous and revealing. It will demonstrate whether EVT-based SPOT/DSPOT truly outperforms (a) a geometric, online tree (RRCF), (b) a predictive sequence model (HTM), and (c) a "patched" static detector (ADWIN+IF). As long as you control for tuning and computational context, this approach will make a compelling, balanced experimental chapter.