🚀 NanoEdge™ Outlier Detection Demo Walkthrough

This is the full development and deployment cycle—from signal capture to embedded anomaly detection—optimized for tight SRAM and clean reproducibility.

🔹 Step 1: Raw DSP Signal Logging

- Run firmware with `SIGNAL_FORMAT` **undefined**
- Watch real-time DSP output over UART (FFT bins, temp, jitter, etc.)
- Validate that your pipeline is transforming raw hardware signals correctly

🔸 Step 2: Data Collection for Model Training

- Define `SIGNAL_FORMAT` in `dsp_test.h`
- Outputs one space-delimited signal vector per row (128 floats per signal window)
- Post-process as needed (e.g. limit rows with `head`, format for upload)

⚙️ Step 3: Train Outlier Model in NanoEdge AI Studio

- Create a new **Outlier Detection** project
- Upload your signal CSV/TXT, select active dimensions
- Benchmark various libraries
- Use **Validation tab** to preview results & choose the best inference engine
- Deploy and extract the generated ZIP archive

📦 Step 4: Integrate Inference Library

- Copy headers (`*.h`) and static library (`*.a`) into STM32CubeIDE project
- Update linker settings:
    - Add `lib/` path to include directories
    - Add `-lNanoEdgeAI` (or appropriate library name) to libraries list
- Define `INFERENCE_MODE` to enable inference behavior

🔍 Step 5: Inference on Real Data

- Run the firmware with `TExaSdisplay`, PuTTY, or CubeIDE terminal
- The device will:
    - Loop over the entire `const float inference_data[495][128]`
    - Copy one row at a time to a RAM buffer (`inf_call[]`) to strip `const`
    - Run `neai_oneclass()` and print detection results
- Only 512 bytes of RAM used during active inference

♻️ Step 6: Highlight Reusable Buffer Strategy

- Swap static arrays for a custom `ru_vec` abstraction
- Confirm that inference still works with `memcpy()` to buffer.pbuf
- Keep inference input lean and prevent duplicate allocations