

SF2 SUITE RT 1.1.90

Waterpump Maker Edition

Build the complete SF2 chain on your computer

From a CSV signal to a real SCADA alarm

RAW DATA • UDTAT • UDM • UFP • STUDIO • ENGINEERT • SCADA

BUILD-ALONG TUTORIAL

You do not just want to look at SF2; you want to get the complete chain running yourself. In this Maker Edition, you build the Waterpump project step by step: from the CSV recording through data preparation, the semantic map, and fingerprints to a signed runtime package, an automatically discovered local Engine, and a real alarm in the SCADA Simulator.

The important idea is continuity. An SF2 project is a chain of versioned artifacts, not a collection of unrelated settings:

sensor_in.csv -> UDTAT -> UDM -> UFP -> Studio Status Library -> signed Engine package -> local EngineRT -> SCADA

Work in short loops: build, save, open, verify. Continue only when the new artifact exists, opens successfully, and genuinely belongs to the preceding files.

Before You Start

Allow about 60 to 90 minutes if you perform the training steps yourself. You need SF2 Suite RT, write access to the selected SF2-Suite-Data folder, and the packaged Waterpump project at SF2-Suite-Data/waterpump/project.sf2project.json. For the local runtime demo, you also put the Suite's File Stream Server and SCADA Simulator, plus one licensed local Engine slot, on your workbench.

Maker Goal: At the end, the complete chain is running on your computer: historical sensor data is prepared, translated into a semantic map and a fingerprint dictionary, exported as a runtime package, processed by a local Engine, and emitted to SCADA as FIRE/CLEARED events.

The screenshots were produced on Apple Silicon. File pickers and acceleration labels can differ on other platforms, but filenames and artifact contracts stay the same. Production Engine packages must be signed with a trusted Ed25519 key. Unsigned packages are suitable only for explicitly configured development environments.

What You Build and Where It Goes

Artifact	Created by	Why you need it
waterpump_norm_binning.json	UDTAT	Column roles, normalization, and binning
waterpump_numeric_data.csv	UDTAT	Continuous normalized UDM input
waterpump_binned_data.csv	UDTAT	Discrete UFP input
waterpump_F_r64-0.02_e100_g64x64_MAP.jsonl	UDM	Learned semantic map
waterpump_retina_64.jsonl	UFP	Runtime fingerprint dictionary
status_library.statuslib.json	Studio	Reference fingerprints and trigger policy

Artifact	Created by	Why you need it
runtime.sf2enginepkg	Studio	Signed portable EngineRT package

1. Start the Workbench: Open SF2 Suite RT

At startup, Studio asks where project data should be stored. Select the folder that contains the `SF2-Suite-Data` directory. The choice shown below resolves the packaged Waterpump project without changing its internal paths.

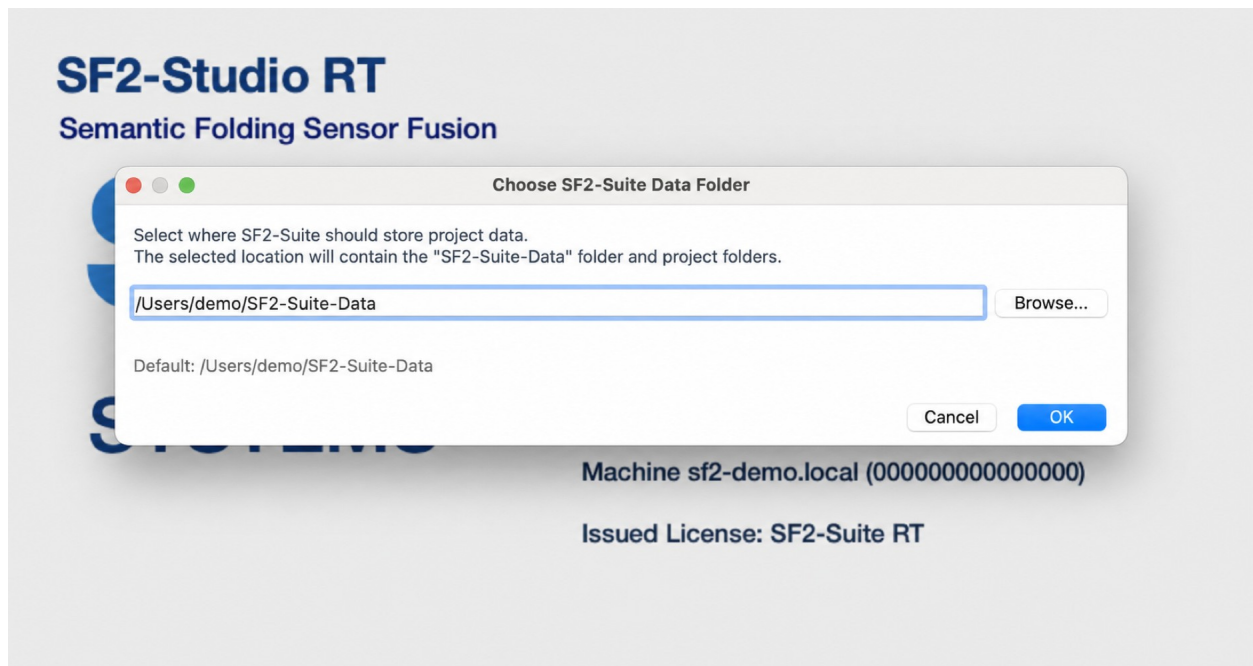


Figure 1. The splash screen and data-root chooser are the first visible states of the walkthrough.

Choose **OK**. Studio opens directly; you do not need to create an empty project first.

Maker Check: The selected data root is writable and contains a `waterpump` project folder.

2. Load the Waterpump Project

Choose **Open Project**, navigate to the `waterpump` folder, and select `project.sf2project.json`.

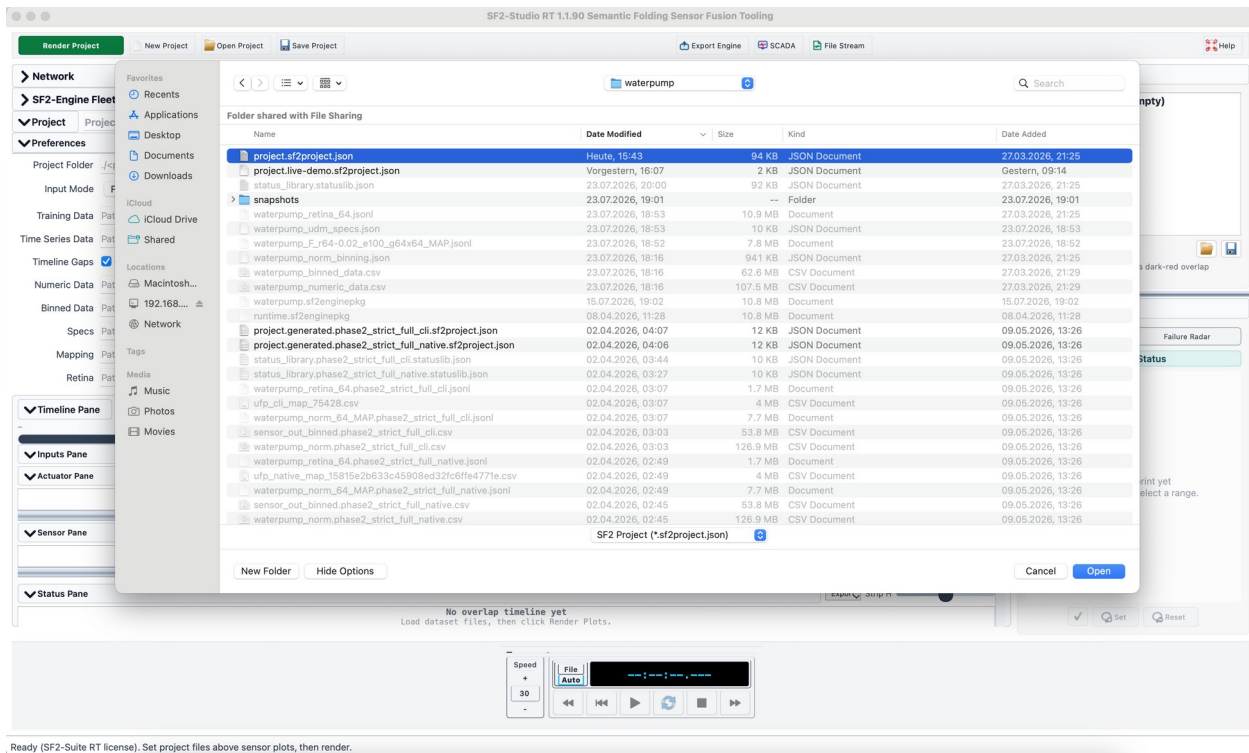


Figure 2. Open the packaged project rather than assembling its artifact paths manually.

Expand **Project** and **Preferences**. Confirm the complete artifact chain:

Field	Expected value
Input Mode	File CSV
Training Data	sensor_in.csv
Time Series Data	sensor_in.csv
Specs	waterpump_norm_binning.json
Numeric Data	waterpump_numeric_data.csv
Binned Data	waterpump_binned_data.csv
Mapping	waterpump_F_r64-0.02_e100_g64x64_MAP.jsonl
Retina	waterpump_retina_64.jsonl

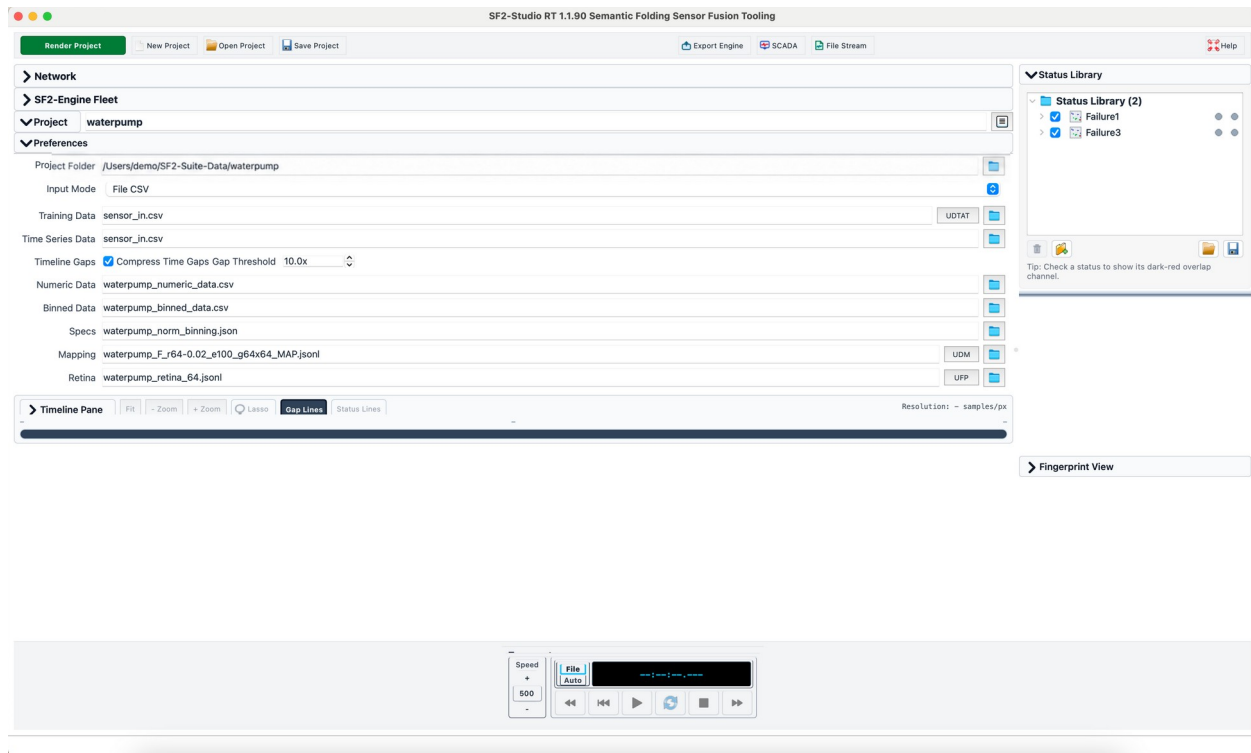


Figure 3. The loaded project ties the source, recipe, numeric and binned data, map, retina, and Status Library together.

The recording contains `id`, `timestamp`, `status`, and `sensor_00` through `sensor_51`: 52 raw sensor columns. The packaged recipe removes the unused `sensor_15`, so the trained model contains 51 sensors. The tutorial intentionally uses the same recording for training and playback so that the complete workflow is reproducible. In production, training and validation or replay data are normally separated.

Maker Check: No artifact field is blank, the project is named `waterpump`, and the supplied `Failure1` and `Failure3` references are present.

3. Target View: What the Finished Project Looks Like

Before rebuilding the project yourself, inspect the finished state. This target view will help you debug later: the Waterpump project is loaded, historical sensor and status data are visible, reference fingerprints are ready, and their similarity curves share the same timeline.

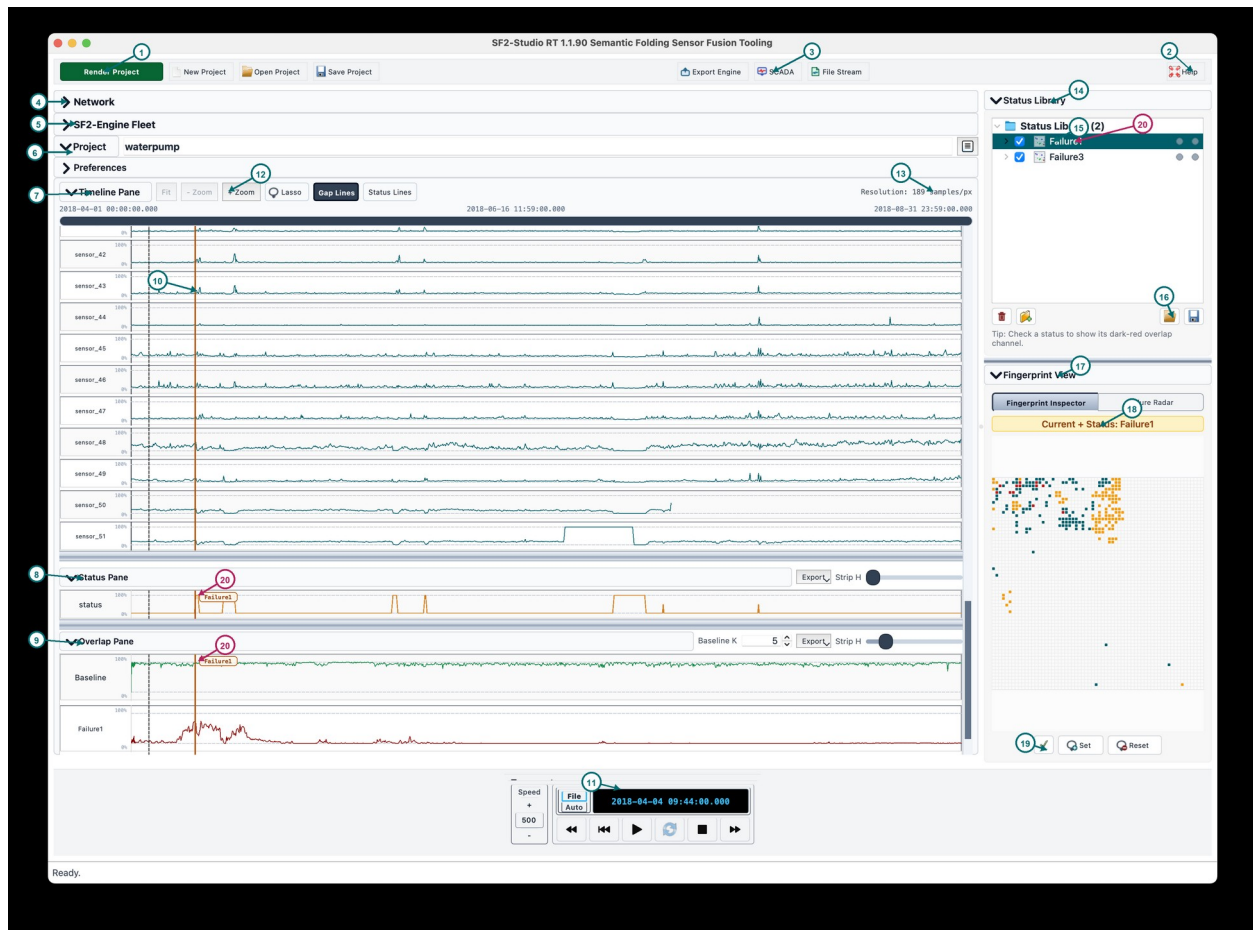


Figure 4. A configured Waterpump project. The numbered legend connects each control with its role in the workflow.

No.	Interface element	What it means in this tutorial
1	Render Project	Loads the configured artifacts and renders the historical timeline, status channels, fingerprints, and overlap curves.
2	Help	Opens contextual help for the active SF2 Studio version.
3	Export Engine / SCADA / File Stream	Opens deployment and simulation tools. These controls are inactive in the SE edition, but the exported artifacts remain part of the same workflow.
4	Network Pane	Configures discovery and alarm uplinks such as MQTT or REST.
5	SF2-Engine Fleet Pane	Discovers, adopts, and monitors EngineRT instances.
6	Project and Preferences Panes	Identify the project and expose its source, preprocessing recipe, numeric data, binned data, map, retina, and tool launch buttons.
7	Timeline Pane	Displays every selected sensor against one shared time axis.
8	Status Pane	Displays recorded operating-state labels, including the high phase used for <u>Failure1</u> .

No.	Interface element	What it means in this tutorial
9	Overlap Pane	Displays one similarity channel for each visible library fingerprint, plus the Baseline channel.
10	Cursor position	The vertical line selects one exact record and keeps sensors, status, overlap, and the current fingerprint synchronized.
11	Tape control	Navigates or replays file-mode data and shows the current recording timestamp.
12	Timeline controls	Fit shows the full recording; Zoom changes detail; Lasso selects a range; Gap Lines and Status Lines add context markers.
13	Resolution	Reports the number of samples represented by each horizontal pixel. Lower values show more temporal detail.
14	Status Library	Stores named, reusable reference states and their alarm settings.
15	Failure fingerprints	Failure1 and Failure3 are named numerical reference vectors captured from reviewed moments in the recording.
16	Load / save library	Reopens or persists the complete Status Library independently of the project file.
17	Fingerprint View	Compares the current or selected fingerprint with a library reference.
18	Current + Status fingerprint	Blue/teal positions belong to the current selection, yellow positions to the selected reference, and red positions to both.
19	Green check mark	Accepts the current cursor or lasso selection as a new library fingerprint.
20	The `Failure1` link	The library entry names the reference, the timeline marker shows where it was captured, and the Failure1 overlap strip shows similarity to it at every time step.

The repeated number 20 is the central relationship to remember. A Status Library item is not merely a label: it is a named fingerprint vector with an origin in the timeline and a corresponding overlap signal across the entire recording.

Maker Check: You can locate the artifact chain, timeline controls, Status Library, fingerprint comparison, and the three linked representations of **Failure1**.

4. Make the Raw Data Model-Ready: UDTAT

UDTAT converts the raw table into two synchronized model inputs. Always enter it through the project workflow: start **SF2 Suite RT**, open the Waterpump project, expand **Preferences**, and choose **UDTAT** beside the training-data field. This preserves the active project context and its configured paths.

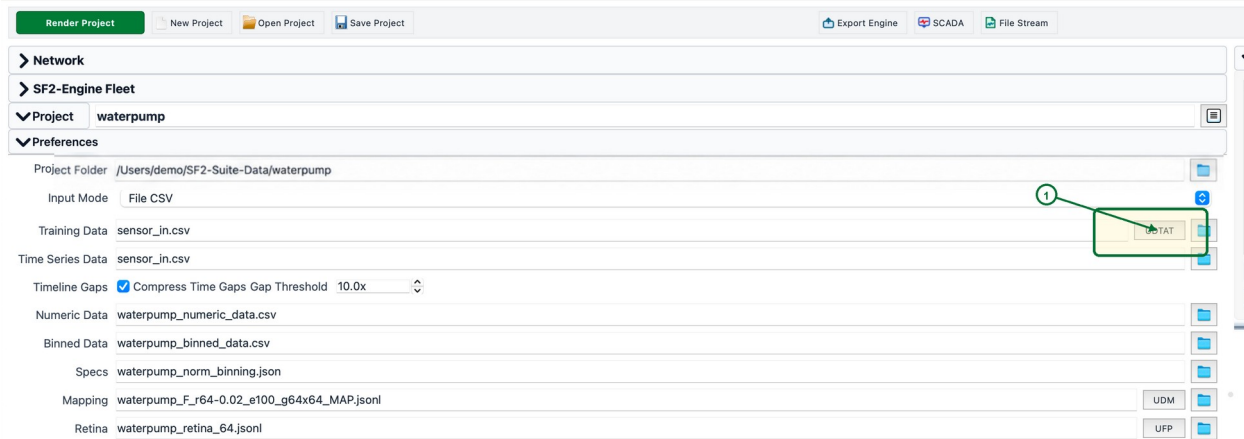


Figure 5. UDTAT is launched from the open project in SF2 Suite RT, not as a detached first step.

4.1 Understand the Role of Each Column

In UDTAT, open **Original** and verify 220,320 rows. Assign **id** and **timestamp** the role **Ignore**, **status** the role **Status**, and the sensor columns the role **Sensor**. Status is supervisory evidence; it must remain available for later interpretation but must not leak into the map-training feature set.

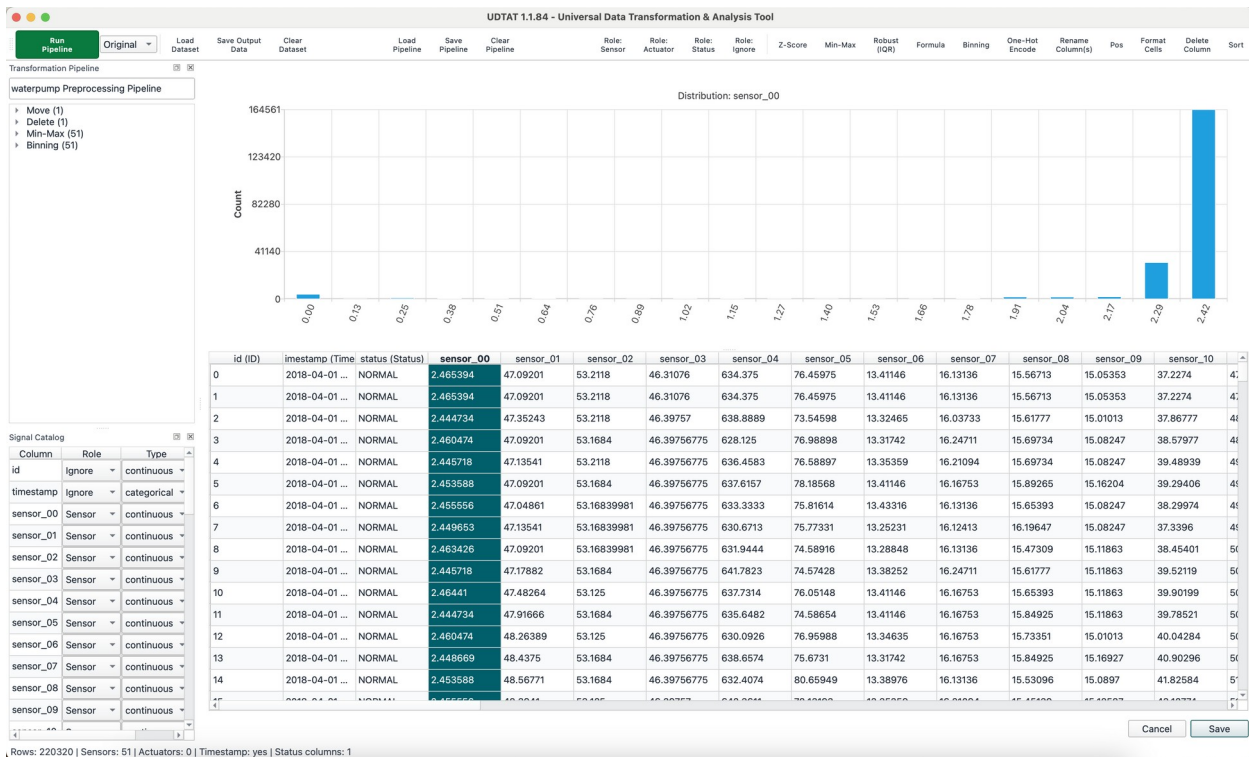


Figure 6. The source table and Signal Catalog establish which columns are features, metadata, and recorded state.

The packaged pipeline moves the status column into the intended output position, removes the unused **sensor_15**, Min-Max normalizes the 51 retained sensors, and assigns a semantic bin label to every

normalized value. Choose **Run Pipeline**, then switch between **Numeric** and **Binned** to verify both outputs.

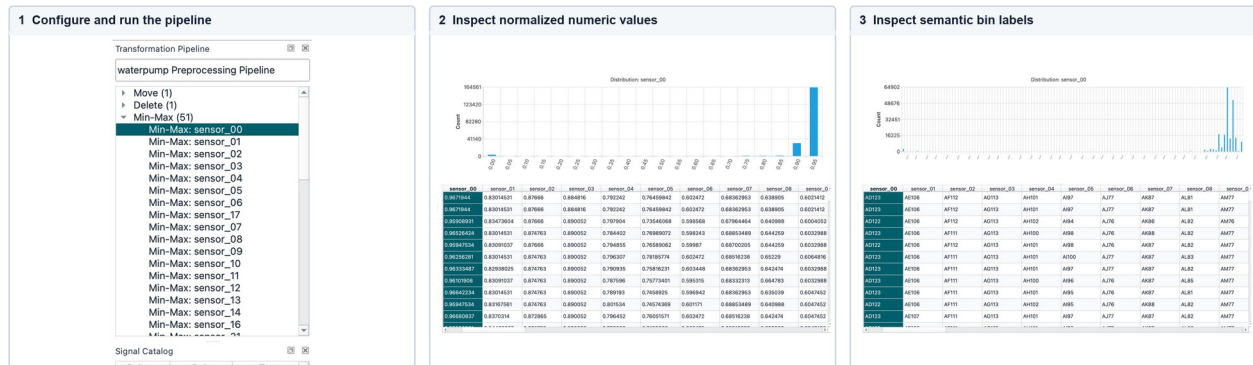


Figure 7. Three focused states replace repeated full-screen views: configure and run the pipeline, inspect normalized values, then inspect semantic bin labels.

Numeric and binned data are siblings created by the same recipe. The numeric table is the continuous UDM input; the binned table is the symbolic UFP input. Their rows must stay aligned because both describe the same physical records.

4.2 Open and Edit the Preprocessing Pipeline

Expand an operation group to inspect its steps. Double-click any entry to open **Edit Pipeline Step**, where the operation and target column can be reviewed or changed.

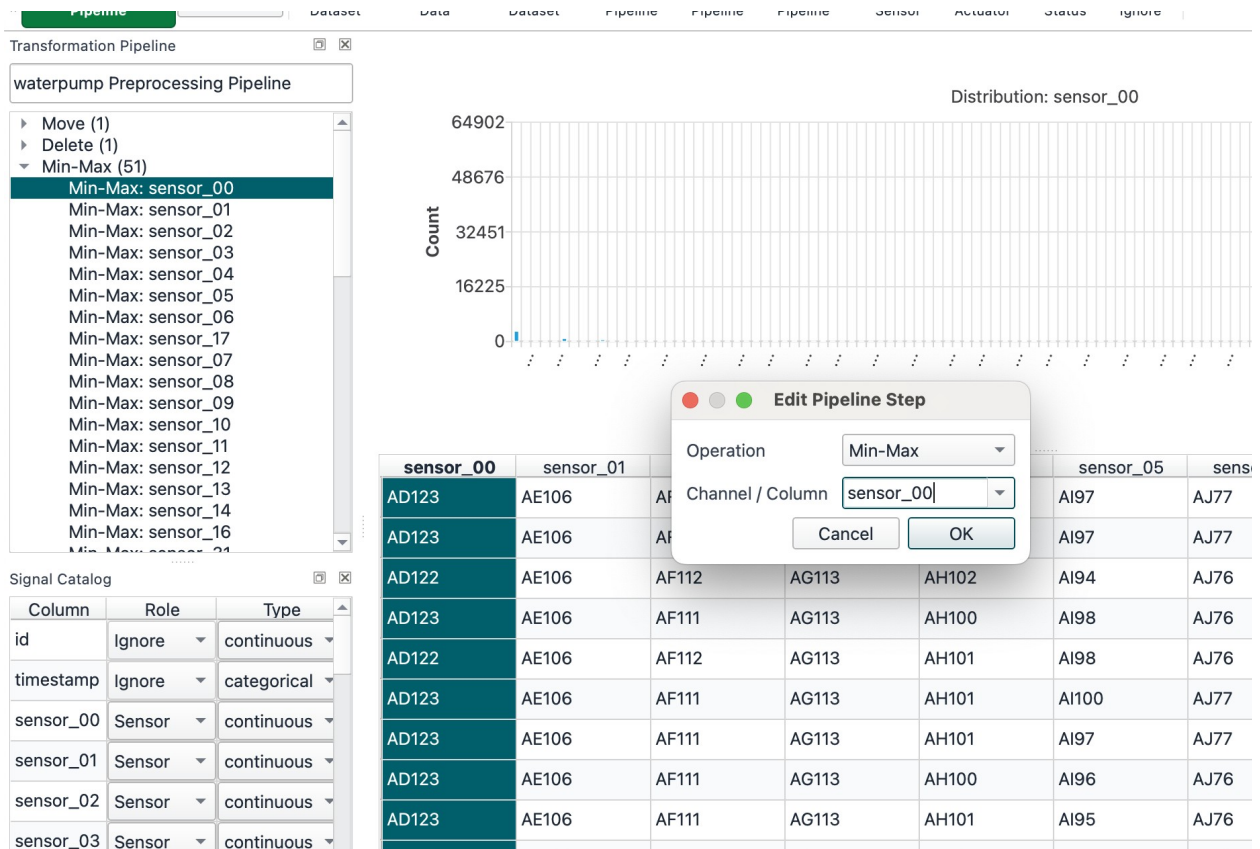


Figure 8. Double-clicking a pipeline entry opens the step editor; changes should be deliberate and followed by a complete pipeline run.

Save the recipe as `waterpump_norm_binning.json`. This file is a runtime contract, not disposable setup metadata. UDM, UFP, Studio export, and EngineRT all depend on exactly the same roles, normalization bounds, and binning rules. A map or retina created from a different recipe may still open, but it will no longer describe the same numerical space and cannot be trusted for correct processing.

4.3 Use the Bar Graph to See What Is in the Data

Selecting a pipeline entry or sensor column displays its value distribution. Each bar covers a value interval; its height is the number of records in that interval. A narrow high cluster indicates a comparatively stable operating band. A separated small cluster can reveal an alternate regime, shutdown, dropout, or fault-related population. A broad distribution means that the sensor explores much more of its range and will usually occupy more bins.

UDTAT distribution comparison on the original Waterpump sensor data

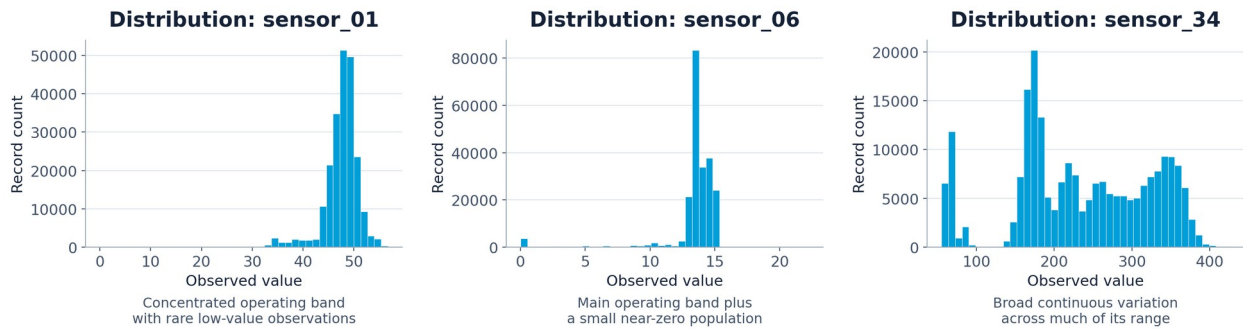


Figure 9. **sensor_01** has a concentrated operating band with rare low values; **sensor_06** adds a small near-zero population; **sensor_34** varies broadly across its range.

Use the graph as a data-quality and configuration check. Unexpected spikes, empty ranges, or a distribution unlike the source signal should be investigated before training. The graph describes frequency, not importance: a rare value may still be the most valuable indicator of a named condition.

Maker Check: UDTAT writes `waterpump_numeric_data.csv` and `waterpump_binned_data.csv` with 220,320 aligned rows, retains 51 modeled sensors, and saves the preprocessing contract used by every downstream stage.

5. Train the Semantic Map: UDM

UDM distributes the training records over a two-dimensional semantic map. Records with similar 51-sensor contexts are placed near one another, so neighboring map positions represent related operating situations.

Workshop Mental Model: Think of the map as a 64 x 64 pegboard. Each training record finds the location whose neighborhood best matches its 51-sensor context. Training means rearranging this grid until similar situations sit meaningfully close together.

Open only **Parameters Pane** and use the reproducible Waterpump baseline:

Parameter	Value
Grid Size	64
Map Topology	Flat Map
Initialization	Random Distribution
Epochs	100
Initial Radius	64.00
Final Radius	1.00

Parameter	Value
Radius Decay Strategy	Exponential (fast start, slow end)

Choose an available training implementation from the performance dropdown. The exact labels depend on the operating system and CPU/GPU combination; select a supported accelerated backend when one is offered, or use the compatible CPU implementation.

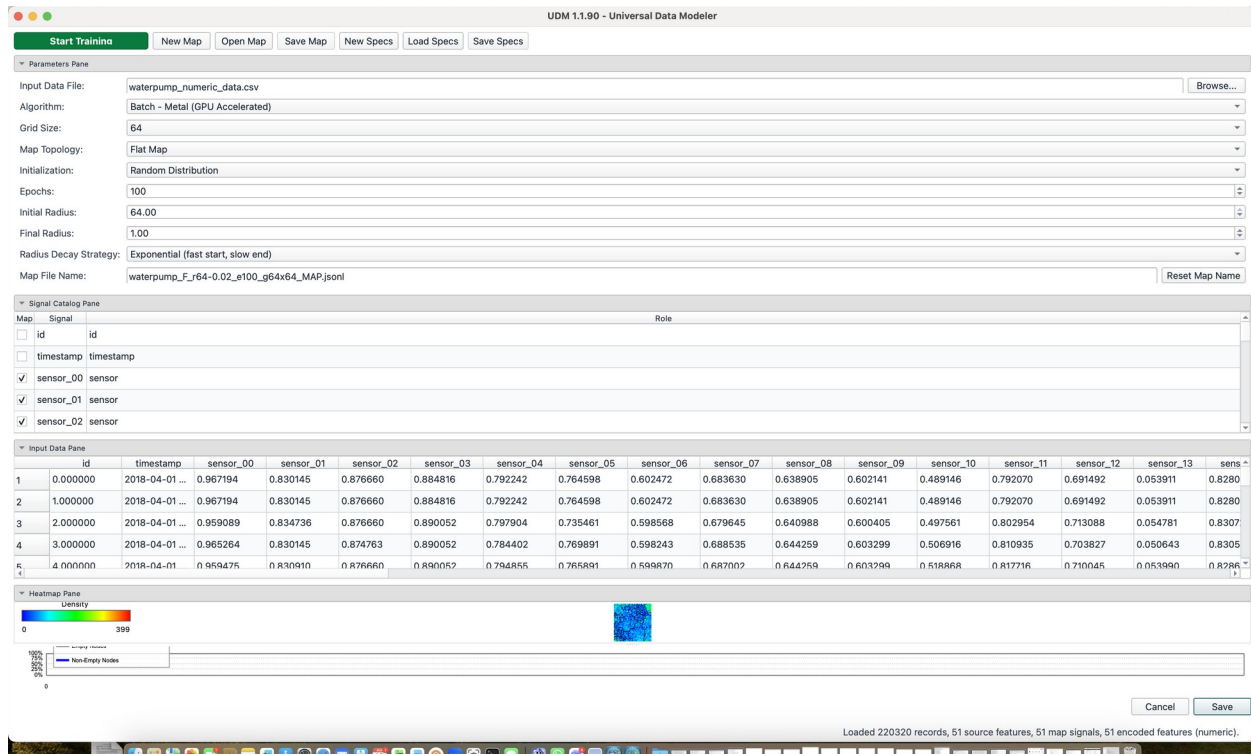


Figure 10. The baseline defines a reproducible 64 x 64 map while the performance selector exposes accelerators supported by the current system.

Close Parameters and open **Input Data Pane**. Confirm 220,320 records and 51 source, map, and encoded features. `id`, `timestamp`, and `status` must not be training features.

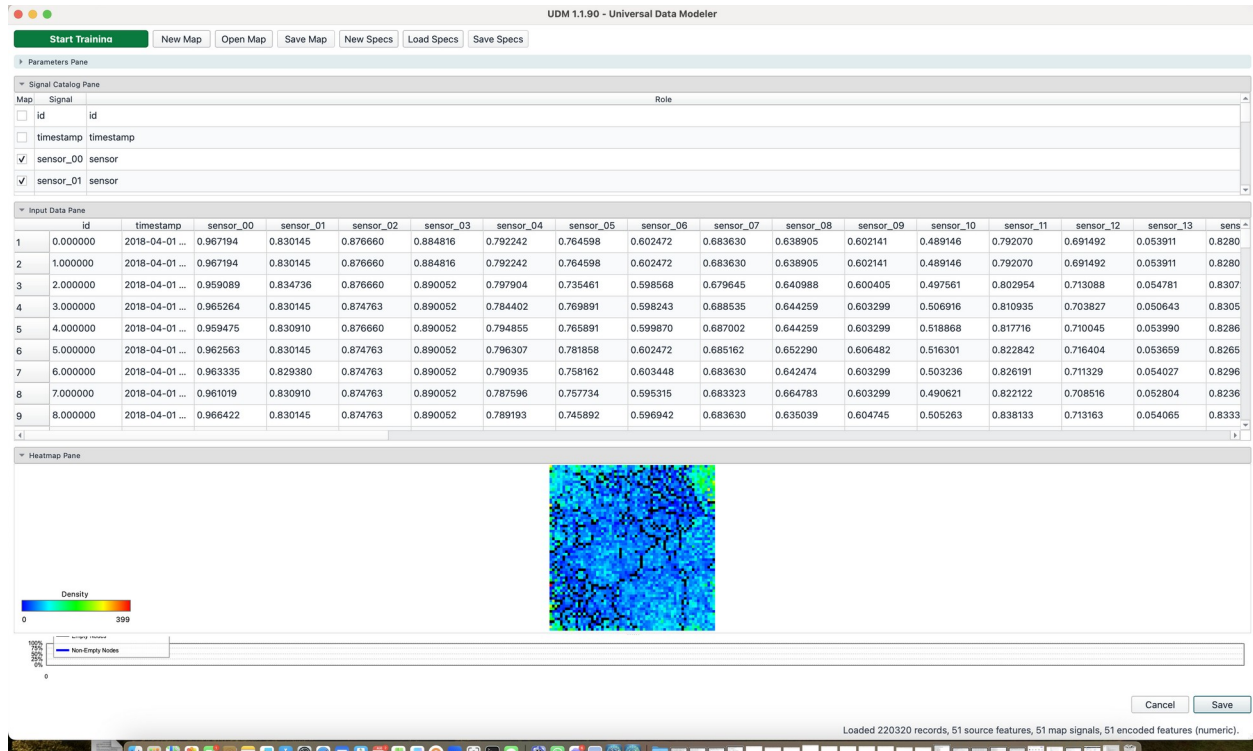


Figure 11. UDM reads the normalized numeric output created by the same UDTAT recipe.

Choose **Start Training**. When training finishes, close the secondary panes and make **Heatmap Pane** the primary view.

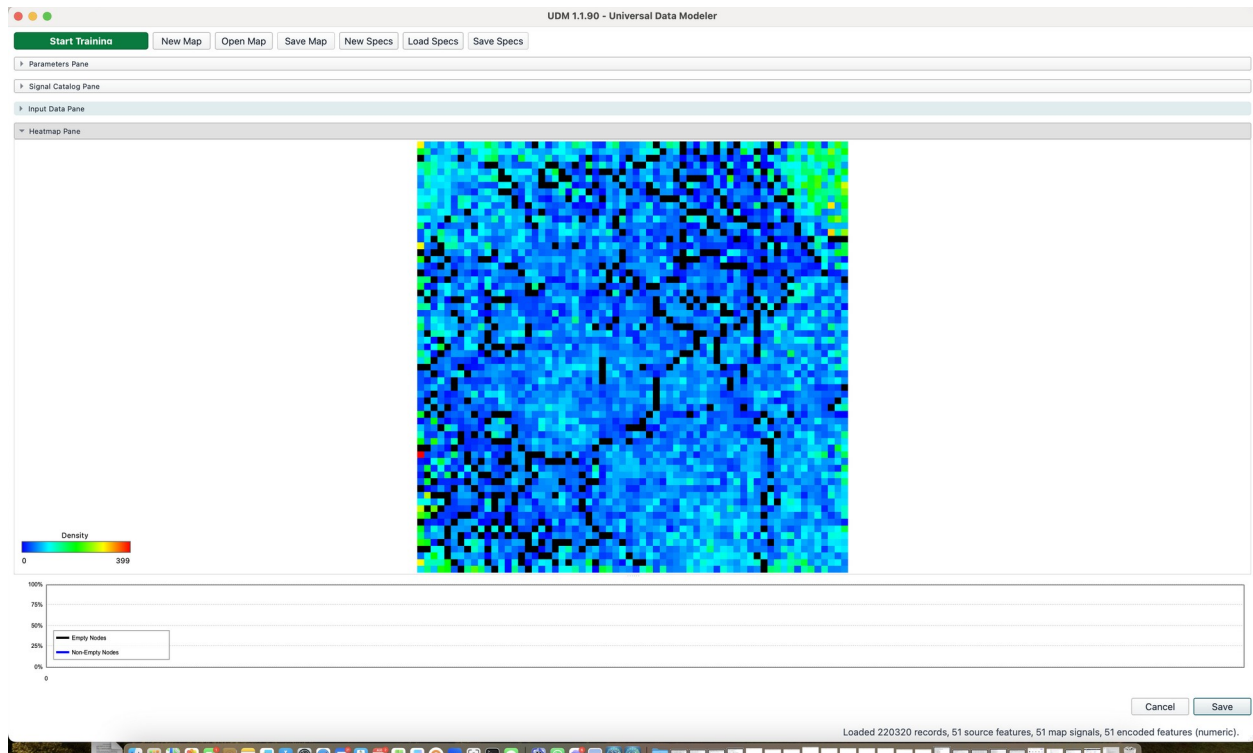


Figure 12. The heatmap is the main UDM result: color and occupancy show how the training records use the semantic map.

Interpret the result in four parts:

1. **Spatial grouping:** similar training records should form neighboring regions rather than being scattered without structure.
2. **Map-space use:** aim to minimize or eliminate unoccupied positions so that the available 64 x 64 space represents the data efficiently. Large empty regions indicate unused capacity or a training/configuration problem.
3. **Distribution graph and legend:** the graph summarizes how evenly records are assigned; the legend maps color to the number of records occupying a position.
4. **Hover inspection:** move the pointer over a calculated map cell to read its position and record count. Check both dense and lightly occupied regions instead of judging only the most saturated color.

A completed epoch counter proves only that computation ended. A useful result distributes the records across the map while preserving local similarity. A blank map, one overwhelmingly saturated cell, widespread empty space, or a feature-count mismatch should stop the workflow.

Save the map as `waterpump_F_r64-0.02_e100_g64x64_MAP.jsonl`.

Maker Check: The 64 x 64 heatmap is populated across the map, UDM reports 220,320 records and 51 modeled signals, and hover inspection returns plausible record counts.

6. Build the Fingerprint Dictionary: UFP

UFP combines the binned Waterpump values with the trained semantic map. For every bin label of every modeled sensor, it creates a 64 x 64 fingerprint. The complete collection of these bin fingerprints is the retina dictionary that Studio and EngineRT use to encode new records.

Workshop Mental Model: The map provides the coordinate system; UFP turns it into a lookup table. For every sensor-value bin, the dictionary records at which map positions and how often that value occurred during training.

Open **Data Sources** and confirm that all three inputs have the same lineage:

Input	Expected file
Binned CSV	waterpump_binned_data.csv
UDM Map File	waterpump_F_r64-0.02_e100_g64x64_MAP.jsonl
UDTAT Recipe	waterpump_norm_binning.json

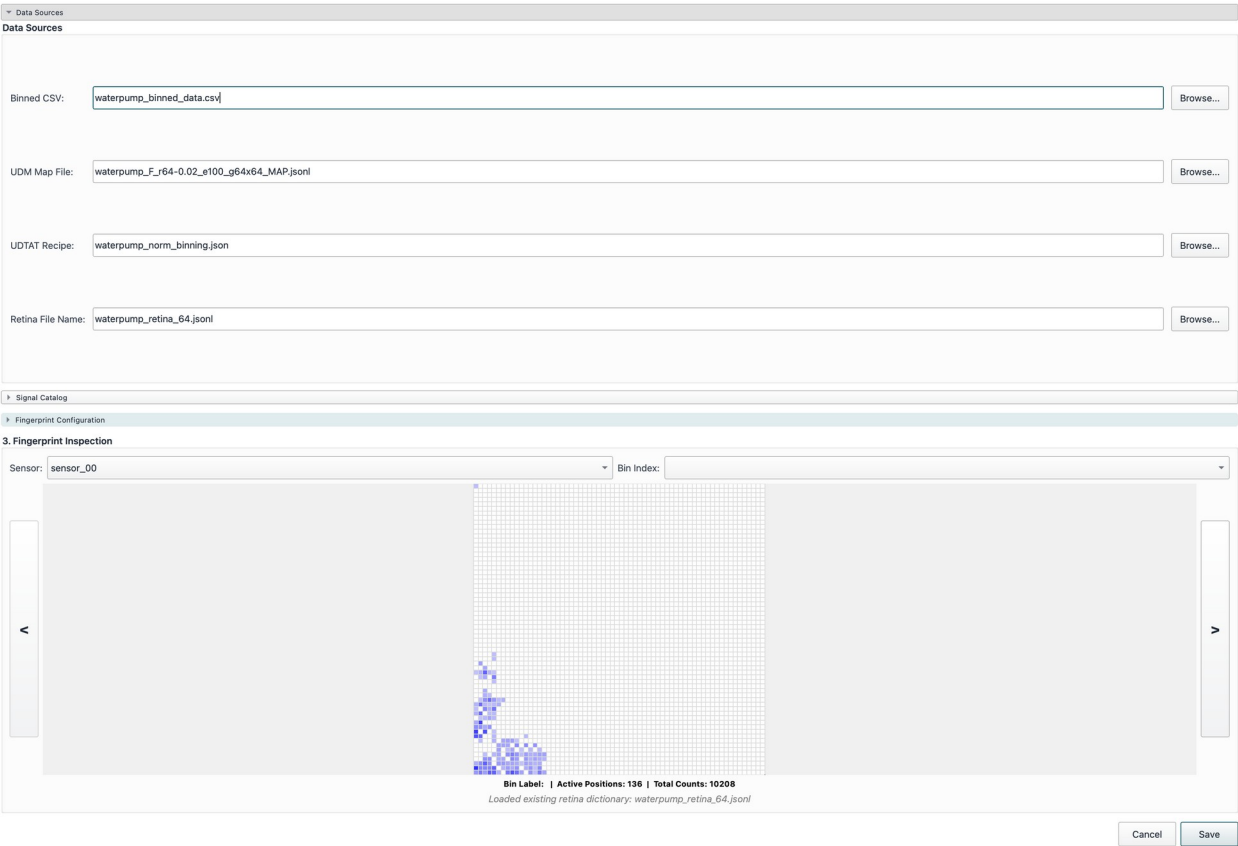


Figure 13. Binned data, map, and preprocessing recipe must belong to the same artifact chain.

In **Signal Catalog**, keep the packaged 51-sensor selection. In **Fingerprint Configuration**, keep the square grid at 64 px, matching the UDM map. Generate and save `waterpump_retina_64.jsonl`.

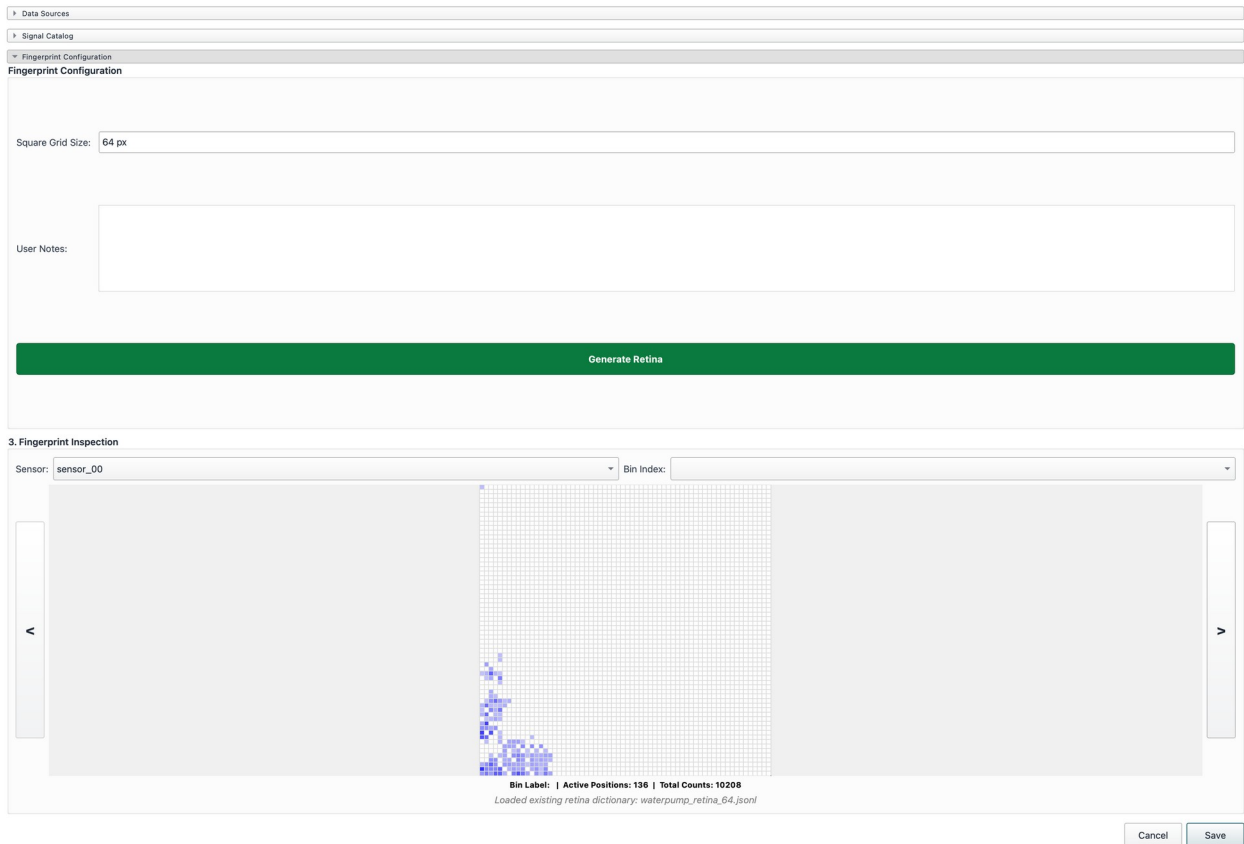


Figure 14. Generation completes the count-based fingerprint dictionary used at runtime.

6.1 From a Few Points to a Dense Surface

Give **Fingerprint Inspection** the full working area and examine bins from several sensors. A bin fingerprint can be extremely sparse or cover most of the map. A rare sensor value may have appeared at only a few training positions; a stable or frequently observed value can be distributed across many positions. Density is therefore a property of that sensor value in the training data, not a pass/fail criterion by itself.



Figure 15. Four bins from three sensors span 65 to 2,854 active positions and make the expected variability visible.

These are **count-based**, not binary, fingerprints. An occupied square records how often the bin occurred at that map position. Darker blue means a larger count; lighter blue means a smaller count. Two fingerprints can therefore occupy similar positions while still carrying different frequency information.

Do not reject a sparse fingerprint merely because much of the grid is empty, and do not assume a dense fingerprint is more important. Instead, confirm that the pattern is nonempty when the bin exists in the data, that the sensor/bin label is correct, and that its active-position and total-count values are plausible.

Maker Check: UFP saves a 64 x 64 retina, the dictionary contains fingerprints for all selected sensor bins, and inspected examples show plausible count shading from sparse to dense.

7. Read the Signals: Timeline in Studio

Return to Studio, save the project, and collapse the setup panes. Choose **Render Project**. If the sensor, status, or overlap curves do not appear, choose **Render Project** again.

Choose **Fit** first. This gives an overview of the complete recording before you investigate details. Scroll to the bottom of the Timeline Pane until the **Status Pane** and **Overlap Pane** are visible, and keep the **Status Library** and **Fingerprint View** visible at the right. Close panes that are not part of this inspection.



Figure 16. The complete inspection layout keeps sensor curves, recorded status, overlap channels, library references, and the fingerprint comparison in one synchronized view.

The three plot areas answer different questions:

Area	Question
Sensor timeline	What did the measured signals do?
Status Pane	Which operating state was recorded or annotated at that time?
Overlap Pane	How similar was the complete sensor context to each named reference?

Use **+ Zoom** and **- Zoom** to change temporal detail. Once zoomed in, drag the horizontal scrollbar to pan left or right. The **Resolution** value reports how many samples are represented by each pixel, so it is a useful indication of whether you are viewing months, days, or individual records. **Fit** restores the entire recording.

Enable **Lasso** when you want to inspect an interval rather than one cursor position. **Gap Lines** expose compressed time gaps; **Status Lines** carry status transitions through the sensor viewport. Every operation uses the same time axis, so a cursor position or lasso range remains aligned across sensor, status, and overlap strips.

Treat overlap as a similarity measure. High Baseline overlap means the current context resembles the baseline reference; high **Failure1** overlap means it resembles the stored **Failure1** reference. Similarity is evidence about system state, not by itself a guaranteed diagnosis or an exact remaining-life estimate.

Maker Check: Fit, zoom, horizontal pan, and lasso are understood; sensor, status, overlap, Status Library, and Fingerprint View remain visible together during interpretation.

8. Capture States: Fingerprint Library

The Status Library connects two levels of description. A name such as *Healthy - efficient operation*, *Failure1*, or *Maintenance - post service* is symbolic and understandable to an operator. The fingerprint stored behind that name is a numerical context vector that Studio and EngineRT can compare with every record.

Historical status channels are especially valuable here because they tell you when known operating states occurred. These named moments do not need to be failures. Periods in which the pump worked perfectly, commissioning stages, maintenance events, or accessory annotations can all become useful references when they describe a reviewed physical condition.

8.1 Capture a Distinctive Moment

Click a precise position in the timeline to use one time-series record. For *Failure1*, one valid method is to place the cursor exactly on the rising edge of a reviewed *status = high* phase.

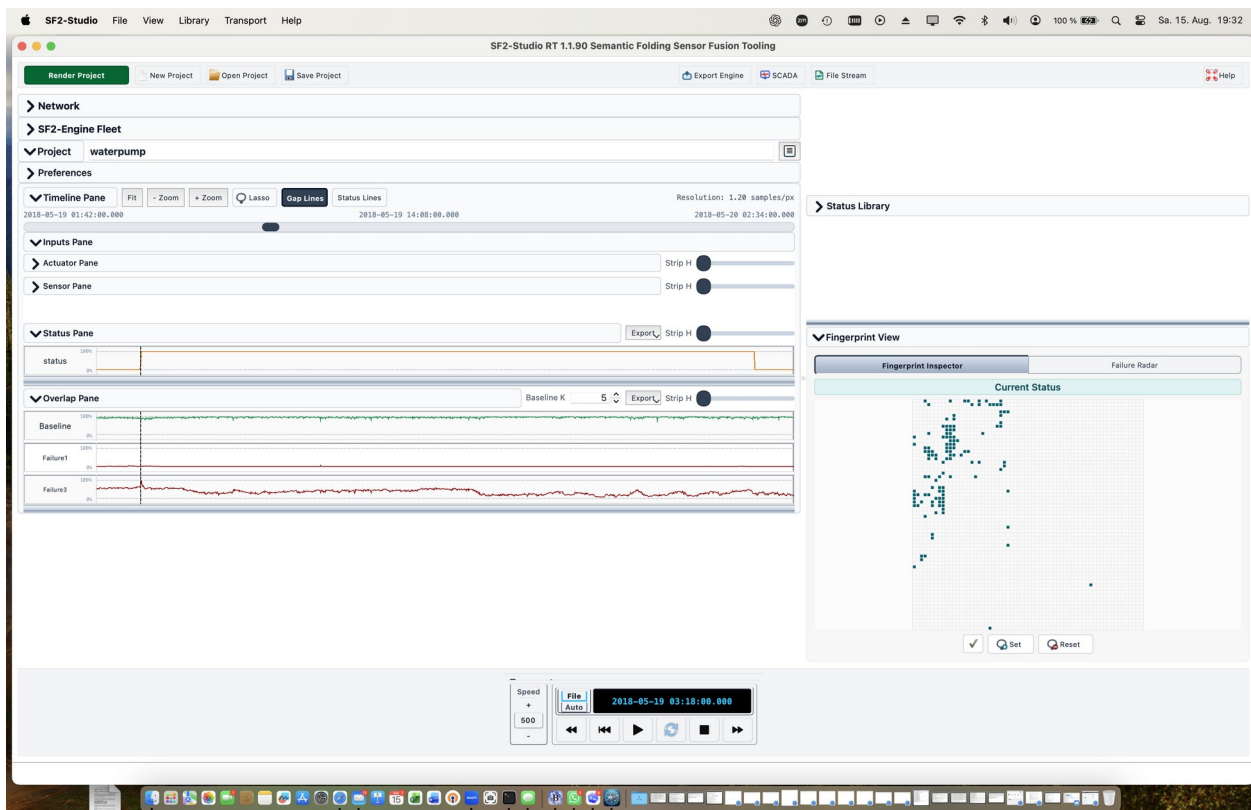


Figure 17. A single click synchronizes the cursor across the timeline and builds the current fingerprint from one record.

8.2 Average a Complete Phase

When one instant is too sensitive to noise, enable **Lasso** and drag across the complete contiguous high phase. Studio creates a fingerprint for every selected record and sums those fingerprints into one aggregate **Section Fingerprint**. Select only records that genuinely belong to the same named condition; do not include normal operation, gaps, or a different regime merely because they are nearby.

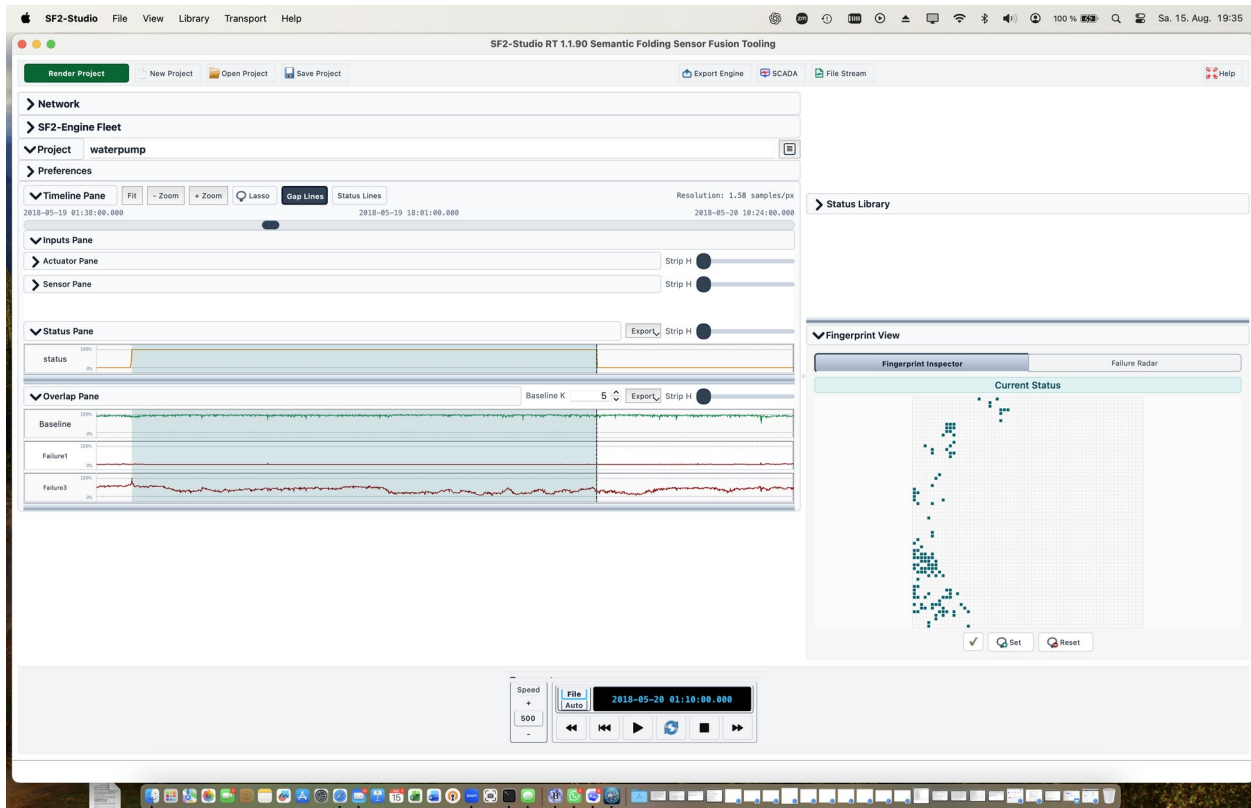


Figure 18. A lasso selection aggregates all records in one reviewed high phase into a more representative Section Fingerprint.

Give **Fingerprint View** enough width that the grid and buttons do not overlap. Whether the source is one record or a section, the result appears in the Fingerprint Inspector. Select the green check mark, whose action is **Accept current fingerprint into Status Library**.

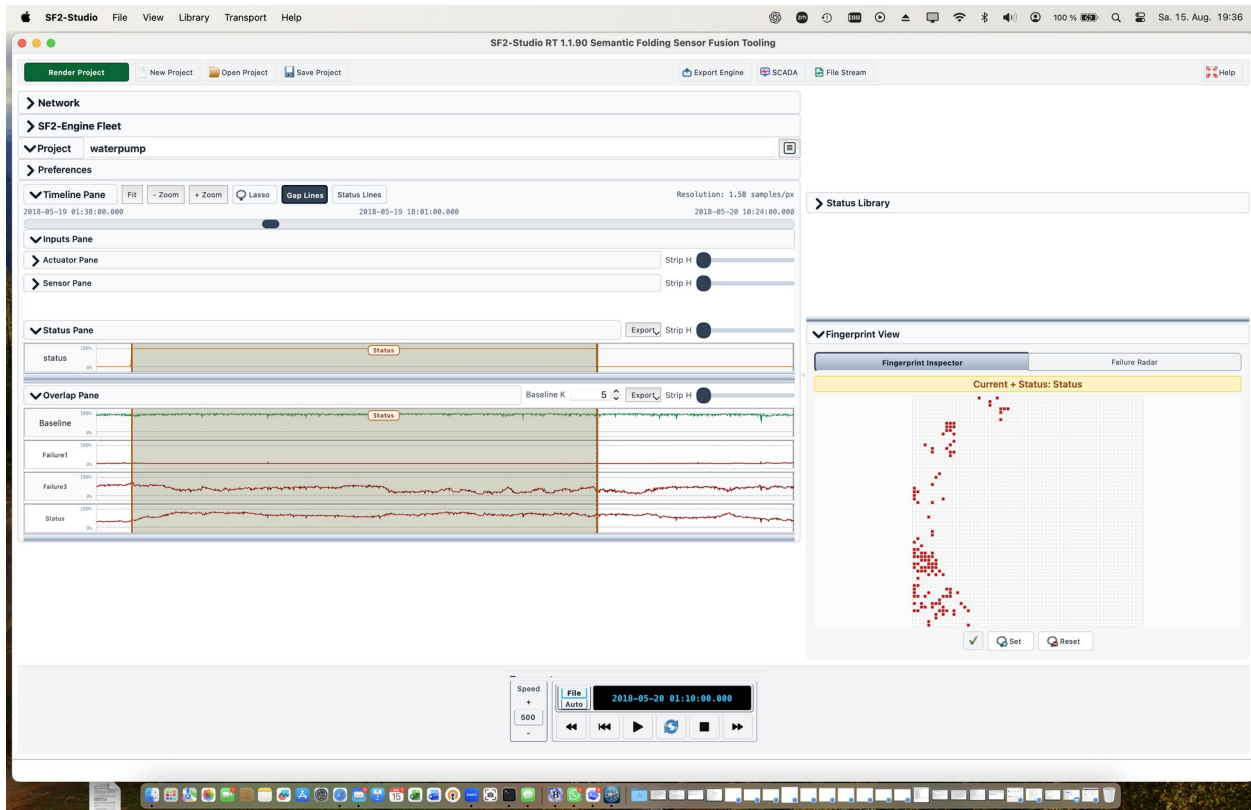


Figure 19. The green check transfers the numerical fingerprint into the Status Library.

Rename the new item immediately with a physical, reusable description. The tutorial uses **Failure - Pump degradation**; a production library may contain any number of named states.

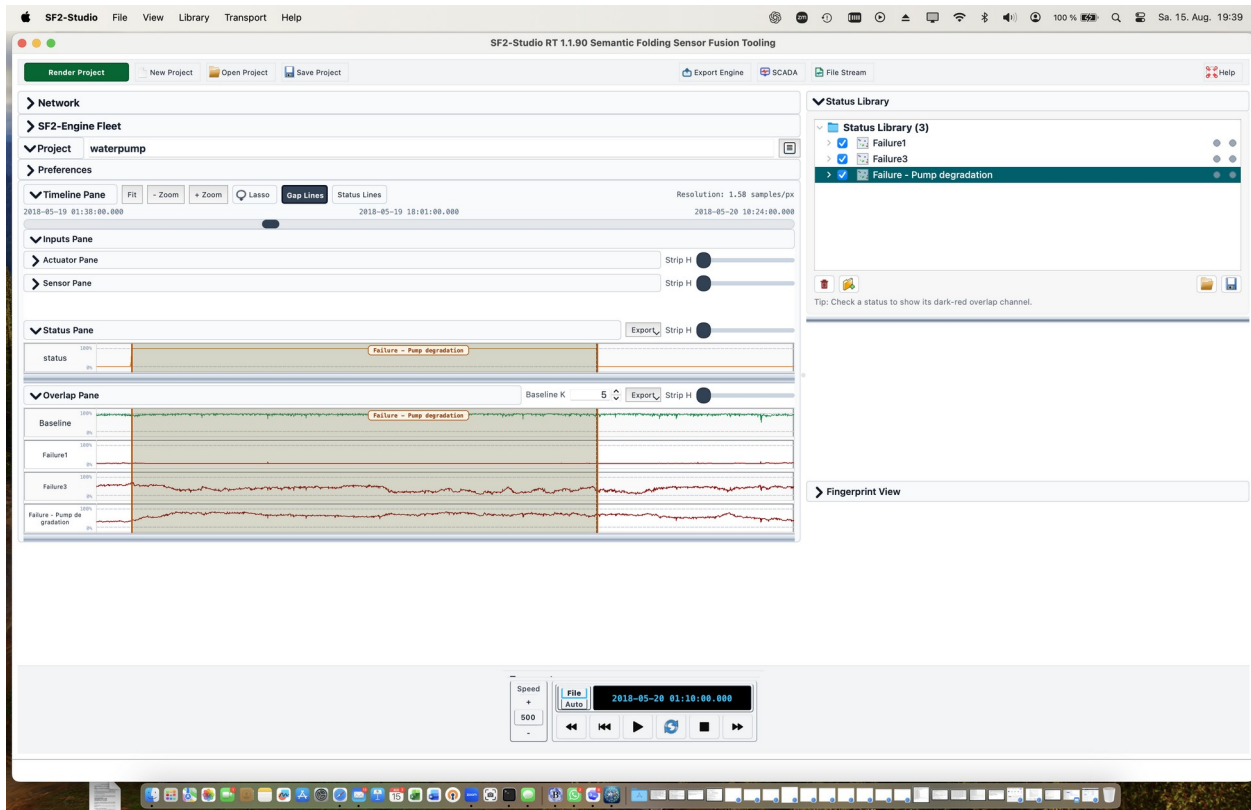


Figure 20. Naming completes the link between the symbolic operating state and its numerical fingerprint vector.

8.3 Read the Reference, Marker, and Overlap Together

Select a library fingerprint. The stored reference appears in yellow, the current cursor or section fingerprint in blue/teal, and positions occupied by both are red. This comparison makes the numerical overlap visible rather than reducing it to a single percentage.

At the same time, Studio shows the reference's source marker in the timeline and creates a named channel in the Overlap Pane. For `Failure1`, the three representations stay linked:

1. The **Status Library entry** stores and names the reference vector.
2. The **timeline marker** shows the record or section from which it was captured.
3. The **'Failure1' overlap strip** shows, at every time step, how strongly that context resembles the stored reference.

This means the curve can be read as a question repeated for every historical record: *How similar was the complete pump context at this moment to the context named 'Failure1'?*

The library-item checkbox controls overlap visibility; it does not arm an alarm. Use the disk button to save the Status Library and save the Studio project when the change should persist. **Set** and **Reset** support expert edits inside the Fingerprint Inspector; manual changes should always be reviewed and documented.

Maker Check: You can create a one-record or Section Fingerprint, accept and name it, interpret the yellow/blue/red comparison, and follow the reference from library entry to source marker to overlap curve.

9. From Pattern to Early Warning

Select **Failure1** in the Status Library. Zoom and pan so the viewport begins at the start of the recording and ends shortly after the first **Failure1** high phase, before the next fault period. This removes unrelated later events and makes the development toward the first failure readable.



Figure 21. The selected **Failure1** reference, its source marker, the recorded status transition, and the earlier overlap rise are visible in one focused window.

At the source marker, the selected context is the reference itself and therefore represents 100% self-overlap. The important observation is what happens before that marker. The **Failure1** overlap curve begins to rise more than four days before the recorded total-failure state. Its detailed shape is dynamic because operating conditions continue to vary, but the overall movement is persistent: the pump context becomes increasingly similar to **Failure1** as the failure approaches.

Aha Moment: The 100% at the marker is only the calibration point: there, you are comparing the fingerprint with itself. The actual finding is the rise to its left. It shows that the multivariate pump context begins drifting toward **Failure1** days before the recorded failure.

This is the conceptual core of SF2 predictive maintenance. Many industrial failures are not spontaneous. An initial event, such as one ball breaking inside a shaft bearing, changes the system context. Secondary damage and compensating behavior then accumulate until the final breakdown is recorded. A fingerprint captured at the final condition allows SF2 to search backward through the complete multivariate history and reveal when the system first began moving toward that context.

The curve does not claim that every increase is a failure or directly calculate remaining useful life. It provides an interpretable similarity trajectory. Domain review, repeated examples, and operating context determine which part of that trajectory is early enough to act on and specific enough to trust.

9.1 Build an Alarm from a Reviewed Curve

Expand the selected fingerprint in the Status Library. Each reference has its own alarm controls.

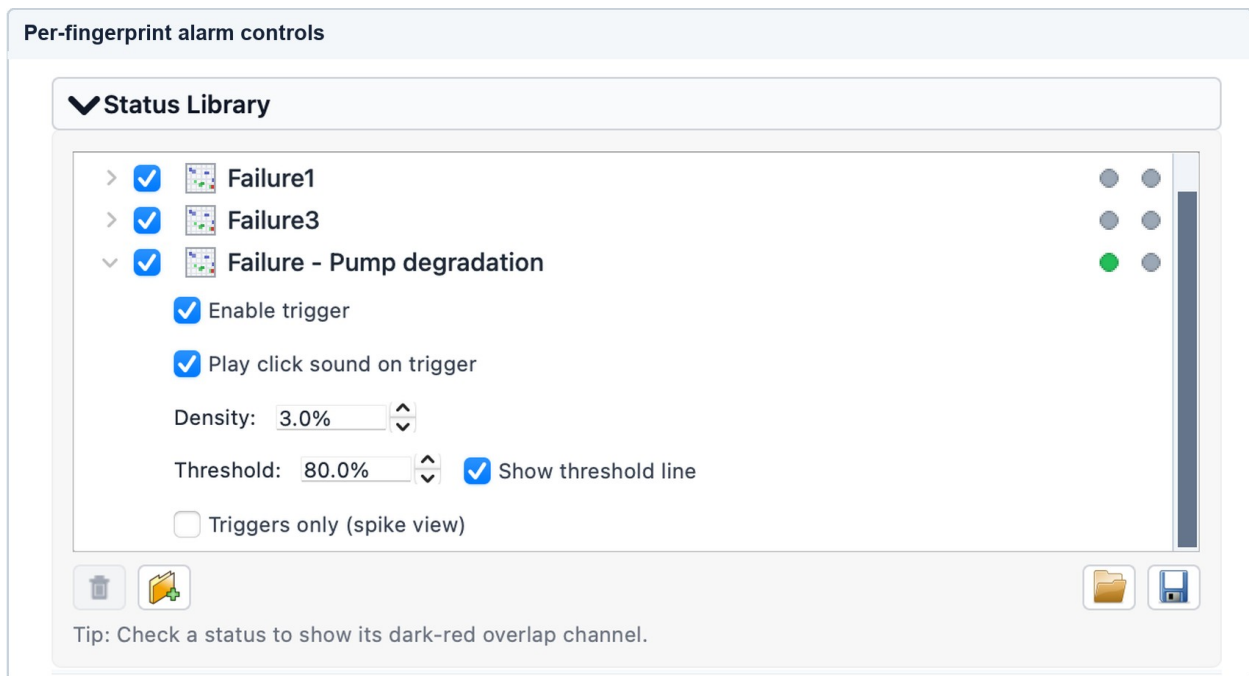


Figure 22. The expanded library item exposes trigger arming, density, overlap threshold, threshold-line display, local sound, and trigger-only visualization.

Control	Purpose
Library-item checkbox	Shows or hides the overlap curve; it never arms the trigger.
Enable trigger	Arms FIRE on upward threshold crossing and CLEARED on downward crossing.
Play click sound on trigger	Provides local Studio feedback.
Density	Defines the sparsity used for the stored reference and should be reviewed before threshold tuning.
Threshold	Sets the overlap percentage at which the named condition becomes an event.

Control	Purpose
Show threshold line	Draws the configured threshold directly in the overlap strip for visual tuning.
Triggers only (spike view)	Replaces the continuous curve with event markers when only crossings are needed.

Determine the threshold from representative historical timelines. A value that is too low can create excessive false alarms; a value that is too high can miss a real event or detect it too late. The goal is not the largest possible lead time, but the earliest threshold that still separates actionable degradation from normal variation. Keep **Show threshold line** enabled while comparing candidate values with known status periods.

Some systems repeatedly approach a fault-like context without failing. Seasonal water volume, for example, may raise similarity every year, while an actual failure occurs only when low water coincides with an ambient temperature above a limit for several hours. Such examples must be represented in threshold validation. A useful alarm catches every reviewed failure early enough while remaining stable across non-failing near approaches.

Per-fingerprint controls decide *when* an event exists. Global settings under **Network > Uplink** decide *where* it goes:

Alarm outcome	Configuration
Visual analysis only	Show the overlap and threshold line; leave Enable trigger off.
Local operator feedback	Enable the trigger and click sound; leave external publishing off.
SCADA through REST	Enable the trigger and REST webhook publishing.
SCADA or event processing through MQTT	Enable the trigger and MQTT publishing.
Redundant destinations	Enable both REST and MQTT.

MQTT provides host, port, client ID, topic, QoS, credentials, retry queue, overflow policy, and spool controls. REST provides the webhook URL and self-test. **MQTT Self-Test** and **REST Self-Test** prove connectivity; **Send Trigger Test** proves payload routing. These tests do not prove that the selected fingerprint or threshold is operationally correct.

Maker Check: The `Failure1` curve is understood as an early similarity trajectory, the threshold is chosen from real timelines rather than guessed, and trigger arming is kept distinct from overlap visibility and transport configuration.

10. Build and Test the Runtime Package

Choose **Export Engine**. Keep the portable **Broadcast** deployment capabilities so source and destination endpoints can be resolved when the local Engine is adopted.

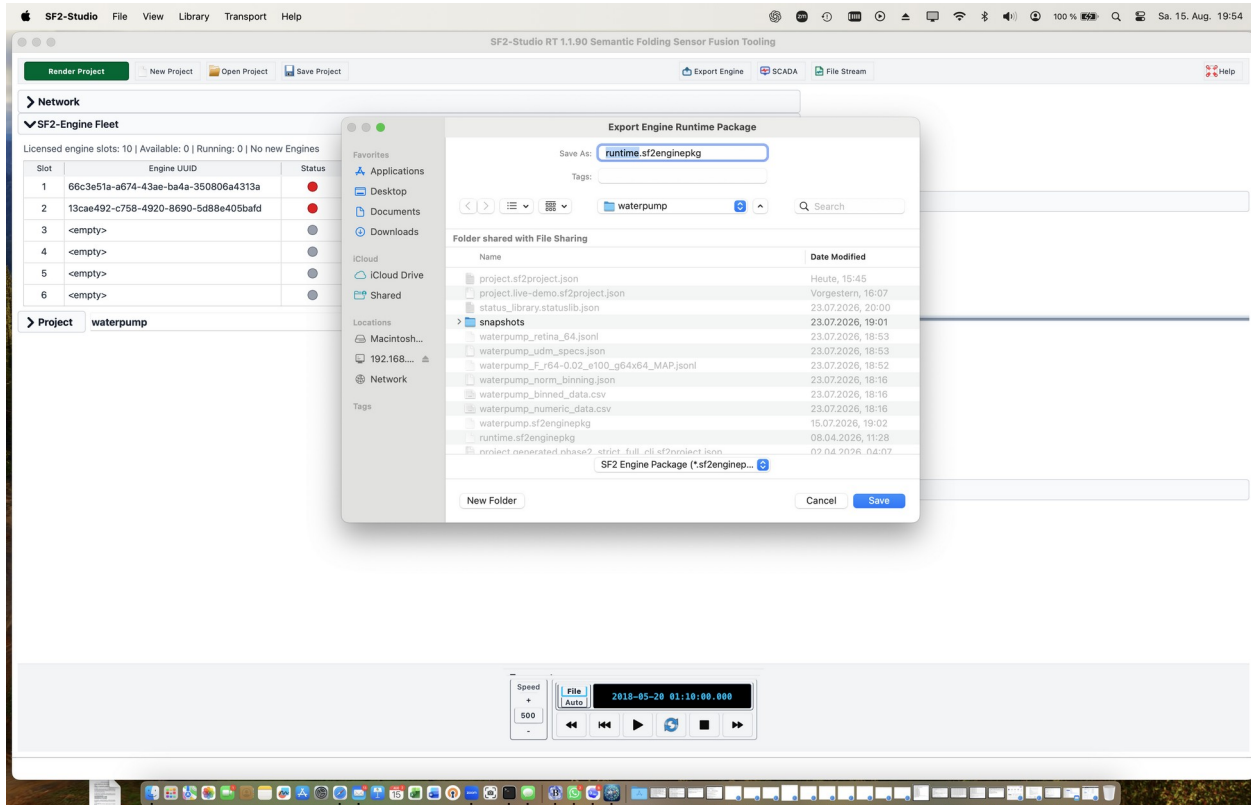


Figure 23. Studio assembles the retina, Status Library, trigger policy, and portable runtime capabilities.

Export `runtime.sf2enginepkg`. The tutorial package is Ed25519-signed and contains the required map, retina, recipe, and Status Library blobs. EngineRT verifies the signature and hashes before execution; production Engine trust must include the signing key.

Describe the package without starting it:

```
SF2-EngineRT \
--package /absolute/path/runtime.sf2enginepkg \
--describe-only
```

The current Waterpump export reports a present Ed25519 signature, four required artifact blobs, Broadcast capabilities, and three configured triggers. Then run the deterministic smoke test:

```
SF2-EngineRT --headless \
--package /absolute/path/runtime.sf2enginepkg \
--input-csv /absolute/path/sensor_in.csv \
--run-smoke \
--output-json /absolute/path/waterpump-engine-smoke.json
```

The verified tutorial run completed with `ok: true`, 220,320 input rows, and valid required hashes and signature evidence. Do not proceed if the package is unsigned, a blob hash differs, or the row count is unexpected.

Maker Check: `runtime.sf2enginepkg` is signed, describe-only validation succeeds, and the smoke report accepts all 220,320 rows.

11. Commission a Local Engine with Plug-and-Play

This tutorial demonstrates **only a local Engine instance**. Keep Studio and the Waterpump project open. Restarting the Suite is neither required nor desirable; reset only the Engine instance when repeating the sequence.

11.1 Make Room First: An Empty Fleet

Stop and release previous tutorial Engines until the Fleet has no assigned or discovered instance.

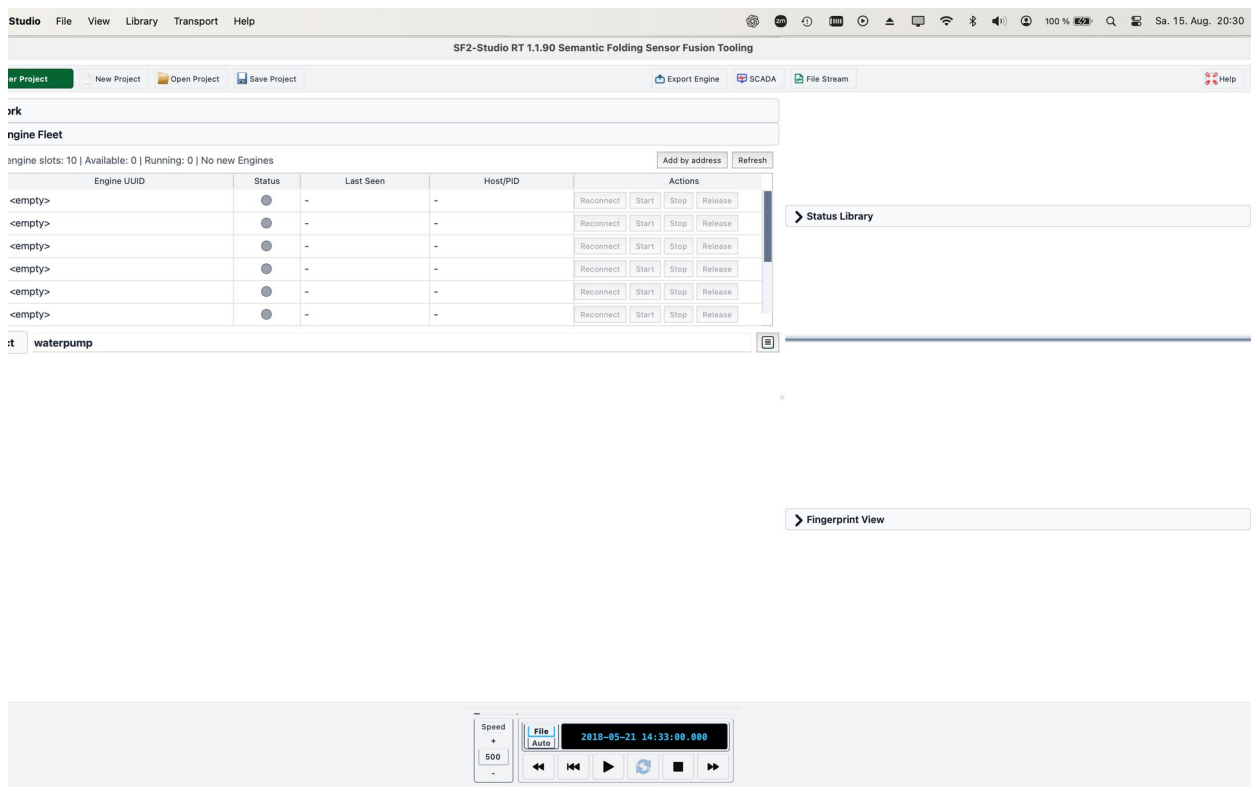


Figure 24. The sequence begins with an empty Fleet and all licensed slots free.

11.2 Start EngineRT Locally

Start one local EngineRT instance with its web control on `127.0.0.1:10032`. The web page confirms the service is alive before it is attached to the project.



Figure 25. EngineRT is healthy locally and waiting to be commissioned.

Do not use **Add by address** in this walkthrough. The plug-and-play path is automatic LAN discovery. Wait briefly or choose **Refresh** until Studio announces **1 engine ready to adopt**.

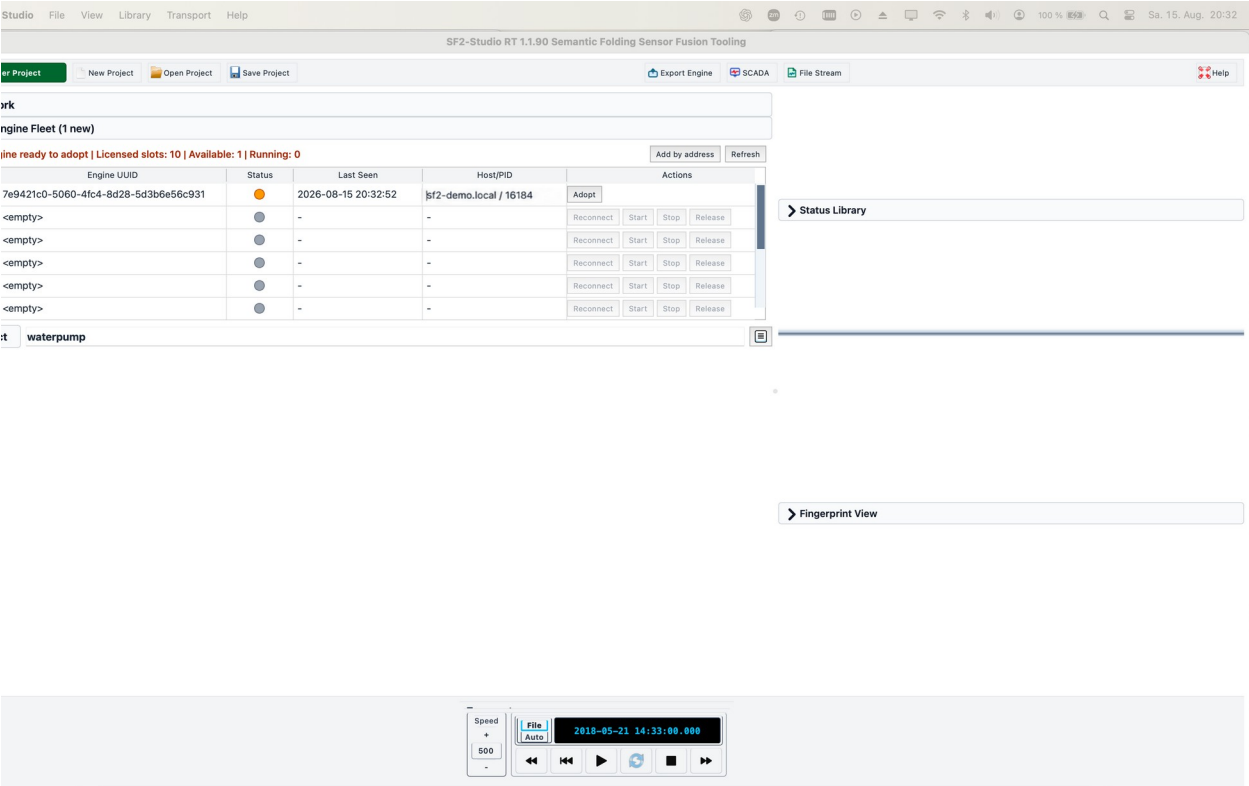


Figure 26. The local Engine UUID and host/PID appear automatically with an **Adopt** action.

11.3 Discover, Adopt, and Start

Select **Adopt**. Adoption claims one licensed slot, binds this Engine UUID to the current Suite gatekeeper, and resolves the package's portable capabilities without rewriting the signed package.

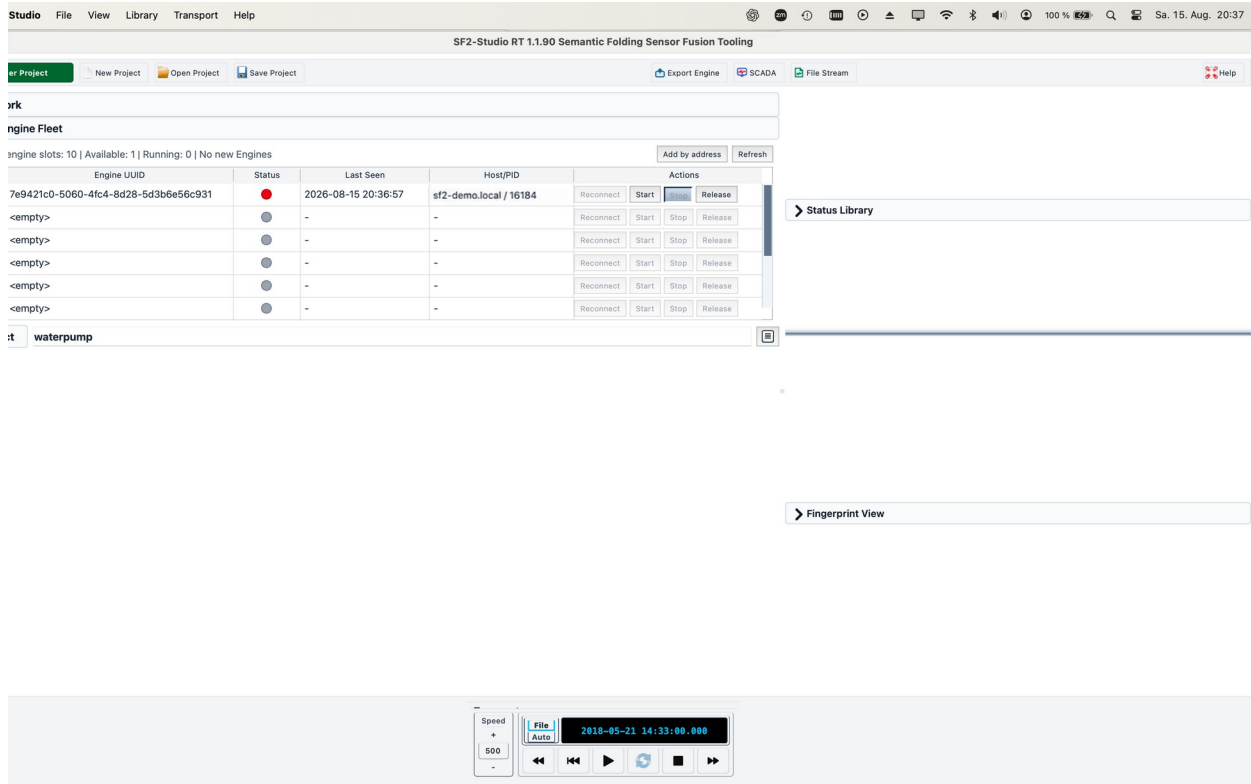


Figure 27. The discovered Engine is now assigned to the Suite.

Wait for the status indicator to turn green and for the Fleet summary to show **Available: 1** and **Running: 1**.

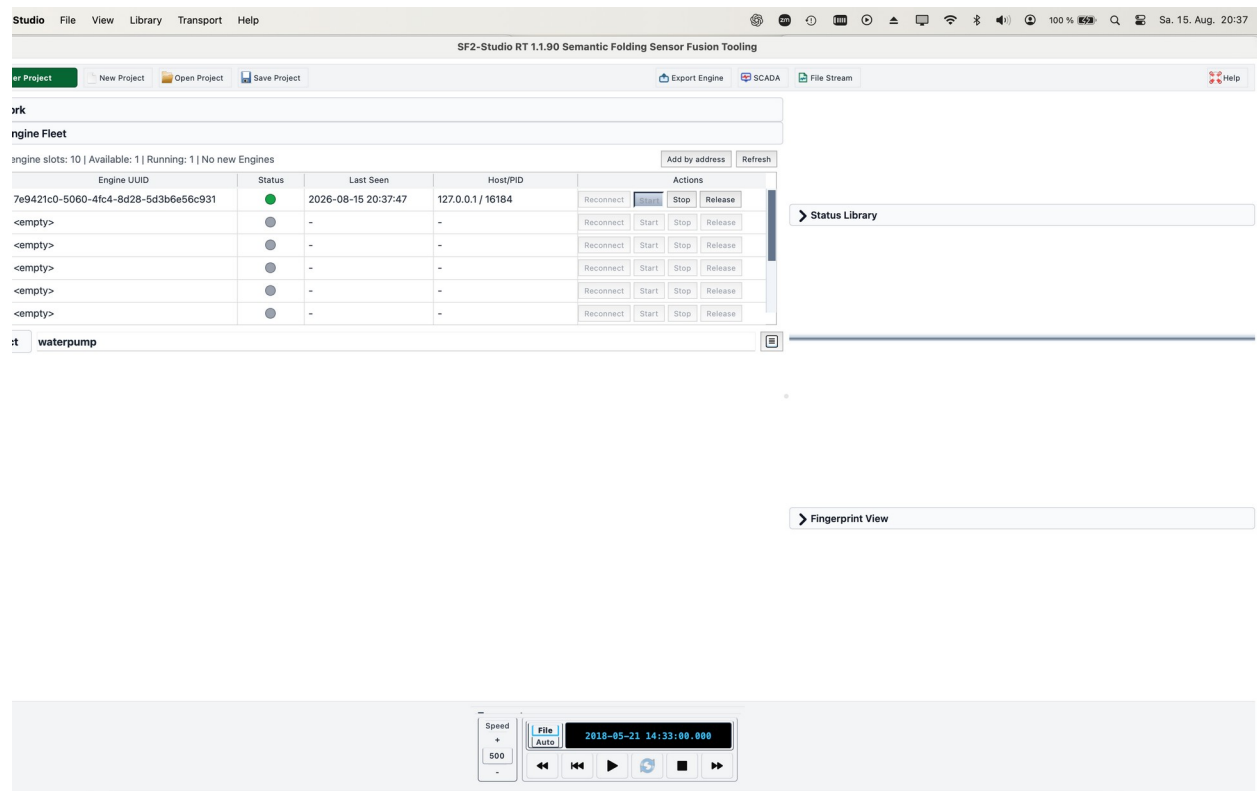


Figure 28. The adopted local Engine reaches Running in the Fleet.

EngineRT can be Running while waiting for its selected live source. Before File Stream starts, the control page shows a valid lease and package together with the waiting TCP state.



Figure 29. Commissioning is complete; the next contract is live input on TCP.

Fleet state	Correct response
Ready to adopt	Select Adopt for the discovered Engine
Waiting	Check binding resolution, gatekeeper lease, or source readiness
Running	Verify frames and trigger evaluation
Offline (license reserved)	Restore the Engine process or route and use Reconnect

Release requires a reachable Engine. EngineRT stops processing and removes its persistent activation token before Suite frees the licensed assignment. It is not a reconnect action.

Maker Check: One automatically discovered local Engine is adopted, the Fleet shows one Running instance, and no address was entered manually.

12. End-to-End: Stream Data and See the Alarm

The runtime proof uses a simple local chain:

File Stream TCP 127.0.0.1:19 001 -> EngineRT -> MQTT 127.0.0.1:1883 -> SCADA Simulator

12.1 Configure File Stream

Open **File Stream** from Studio. In **Data Source**, select `sensor_in.csv` and `stream_config.json`; close the other panes.

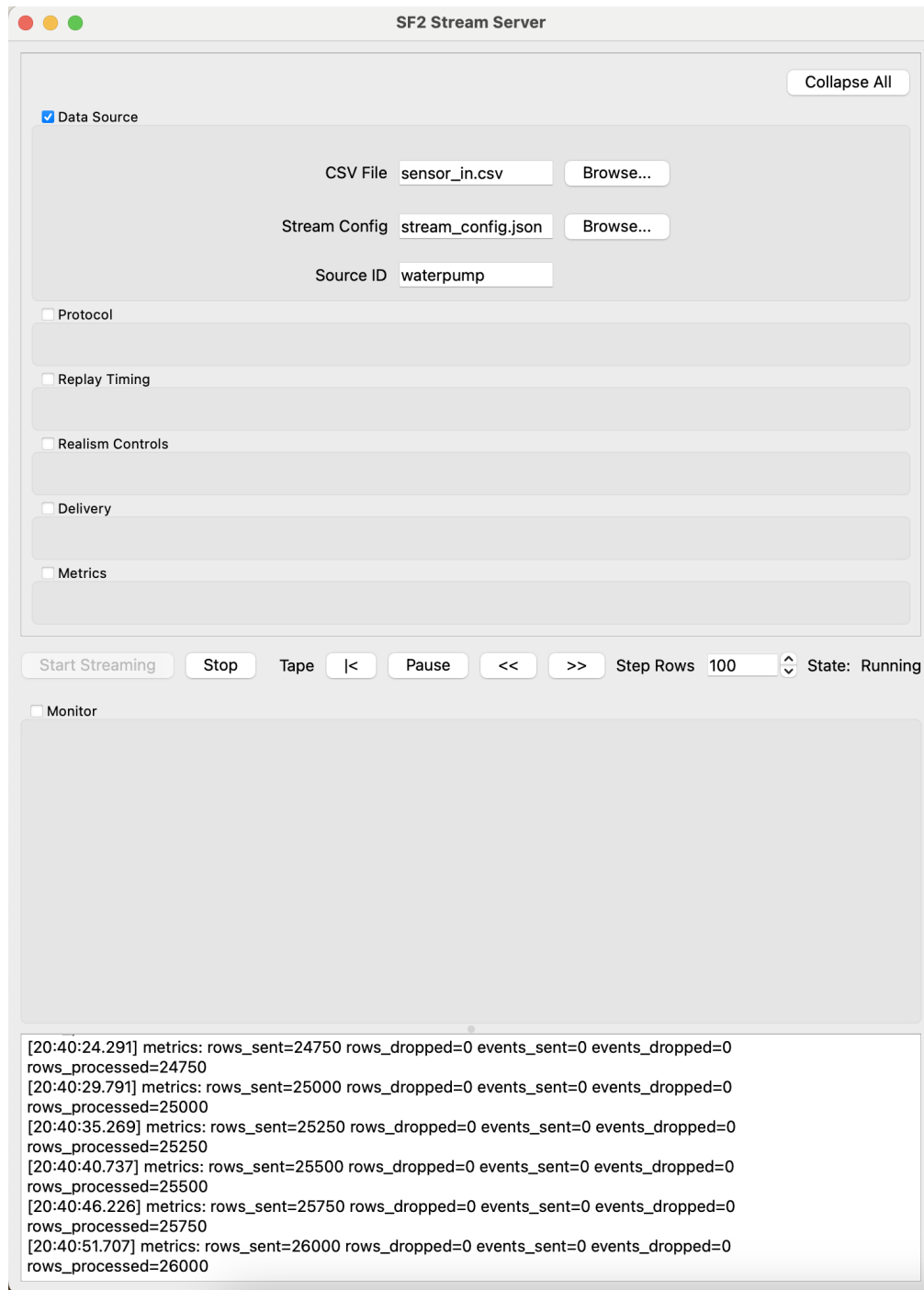


Figure 30. File Stream loads the Waterpump recording and its stream configuration.

Open only **Protocol & Endpoint**, select TCP, and use [127.0.0.1:19001](#).

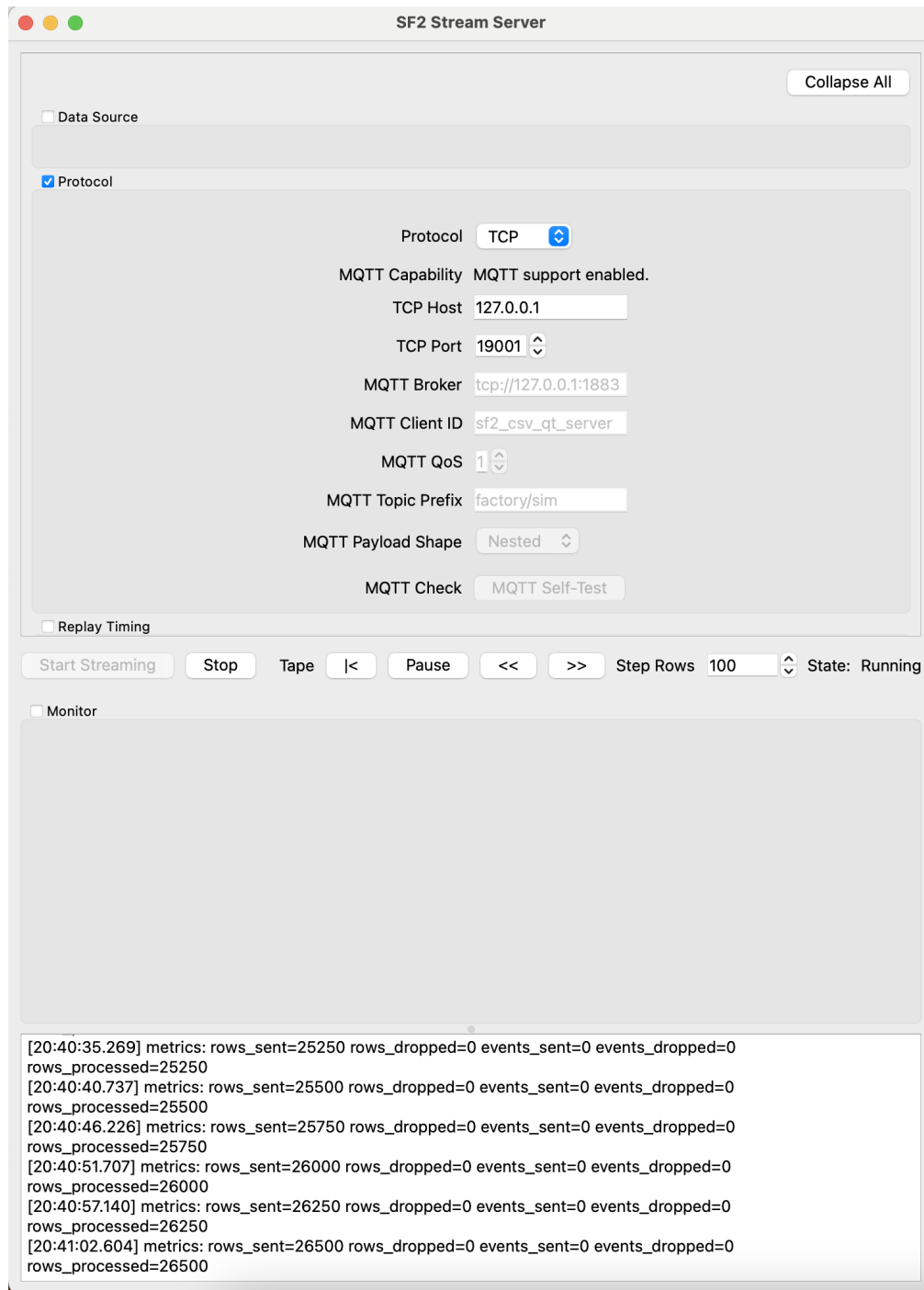


Figure 31. The deterministic local source matches the capability resolved during adoption.

Start the stream, then leave only **Monitor** open. The source contains 220,320 rows and 52 raw sensor fields; EngineRT applies the packaged 51-sensor model contract.

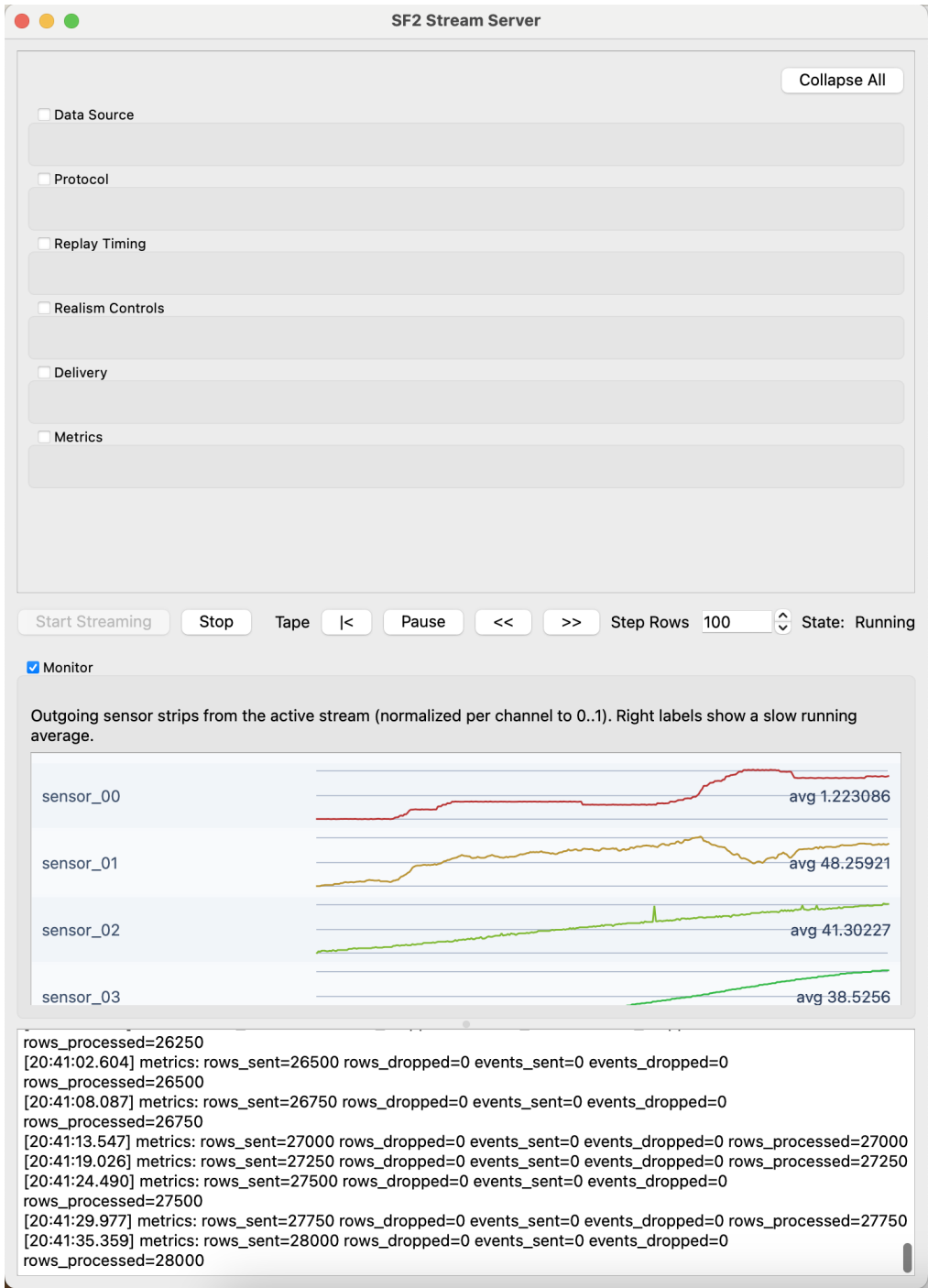


Figure 32. Running confirms that frames are being emitted to the local Engine.

To reach the labeled failure region promptly, use a large row step such as 40000 rather than changing the alarm threshold.



Figure 33. Fast-forward preserves the configured detector while moving through the recorded scenario.

Continue until the stream is near row 71,847, where the demonstrated failure transition occurs.

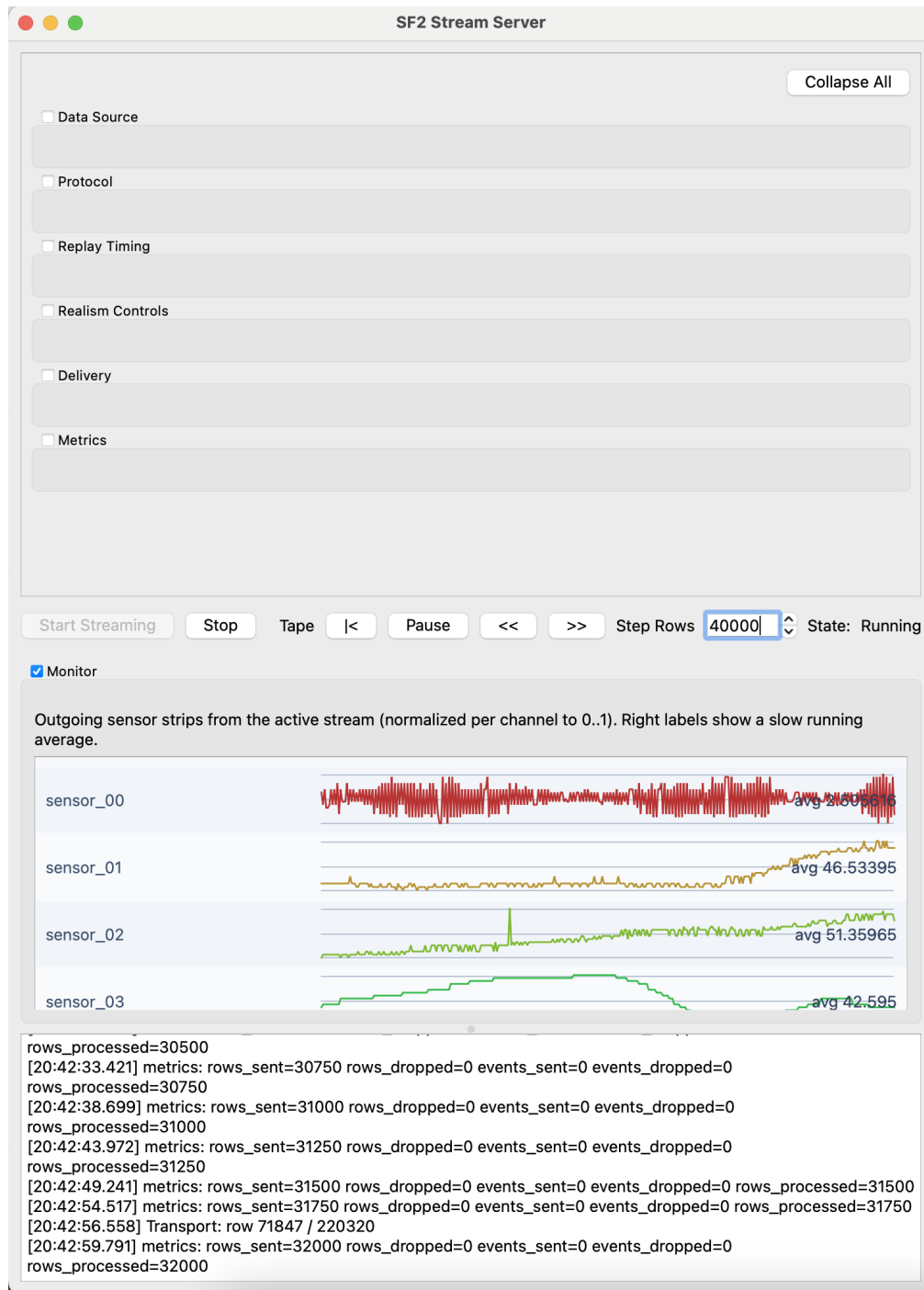


Figure 34. The replay is now in the part of the recording used for the runtime alarm proof.

12.2 Start the local SCADA receiver

Open **SCADA** from Studio. Start the managed MQTT broker on `127.0.0.1:1883`, subscribe the receiver to `sf2/uplink/alerts`, and keep only the active configuration pane open while setting it up.

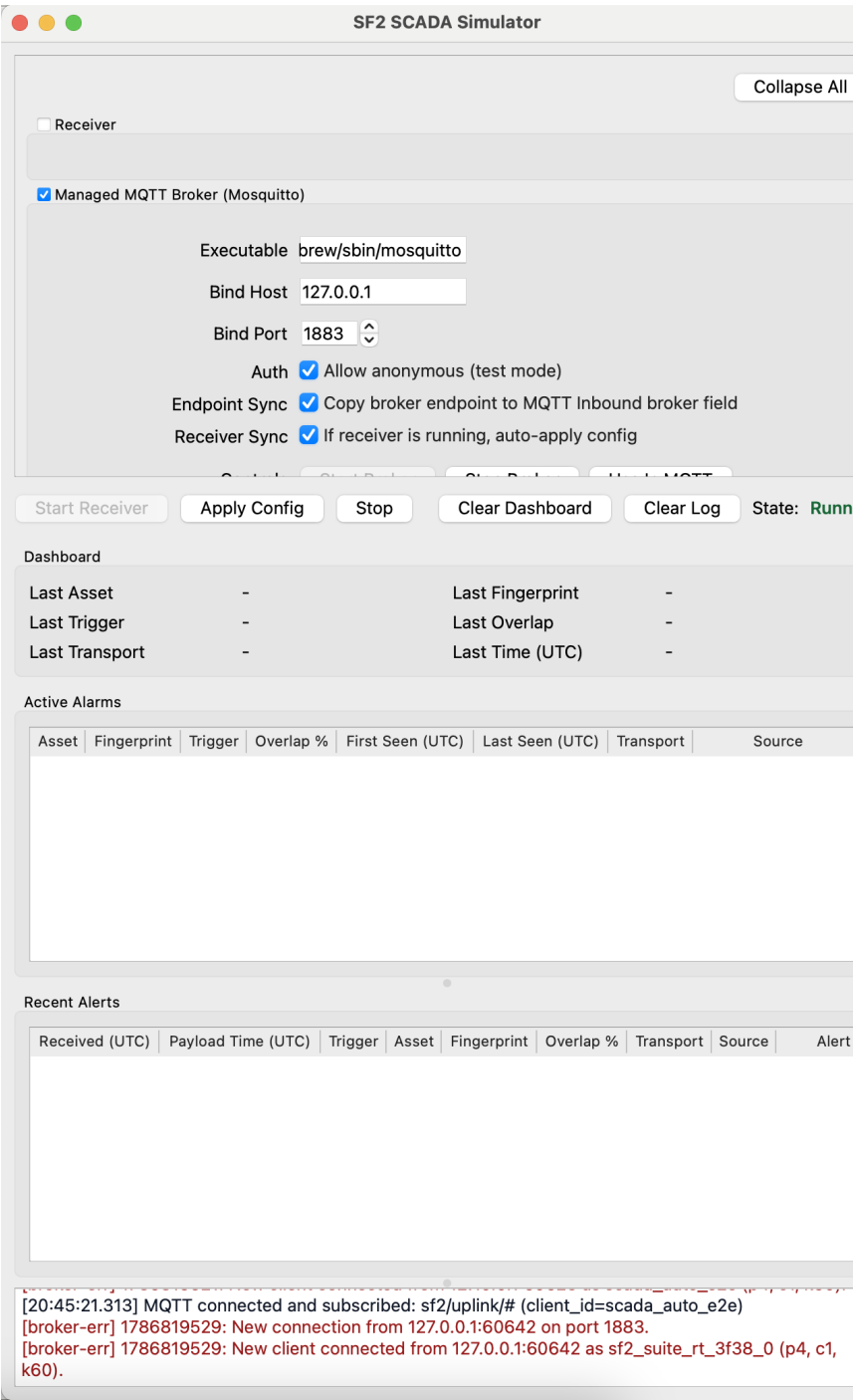


Figure 35. The local broker and inbound subscription provide the Engine's alarm destination.

12.3 Observe the real alarm

As File Stream crosses the failure condition, EngineRT emits the trigger transitions. SCADA receives a FIRE followed by CLEARED for Failure - Pump degradation on asset asset-001 over MQTT.

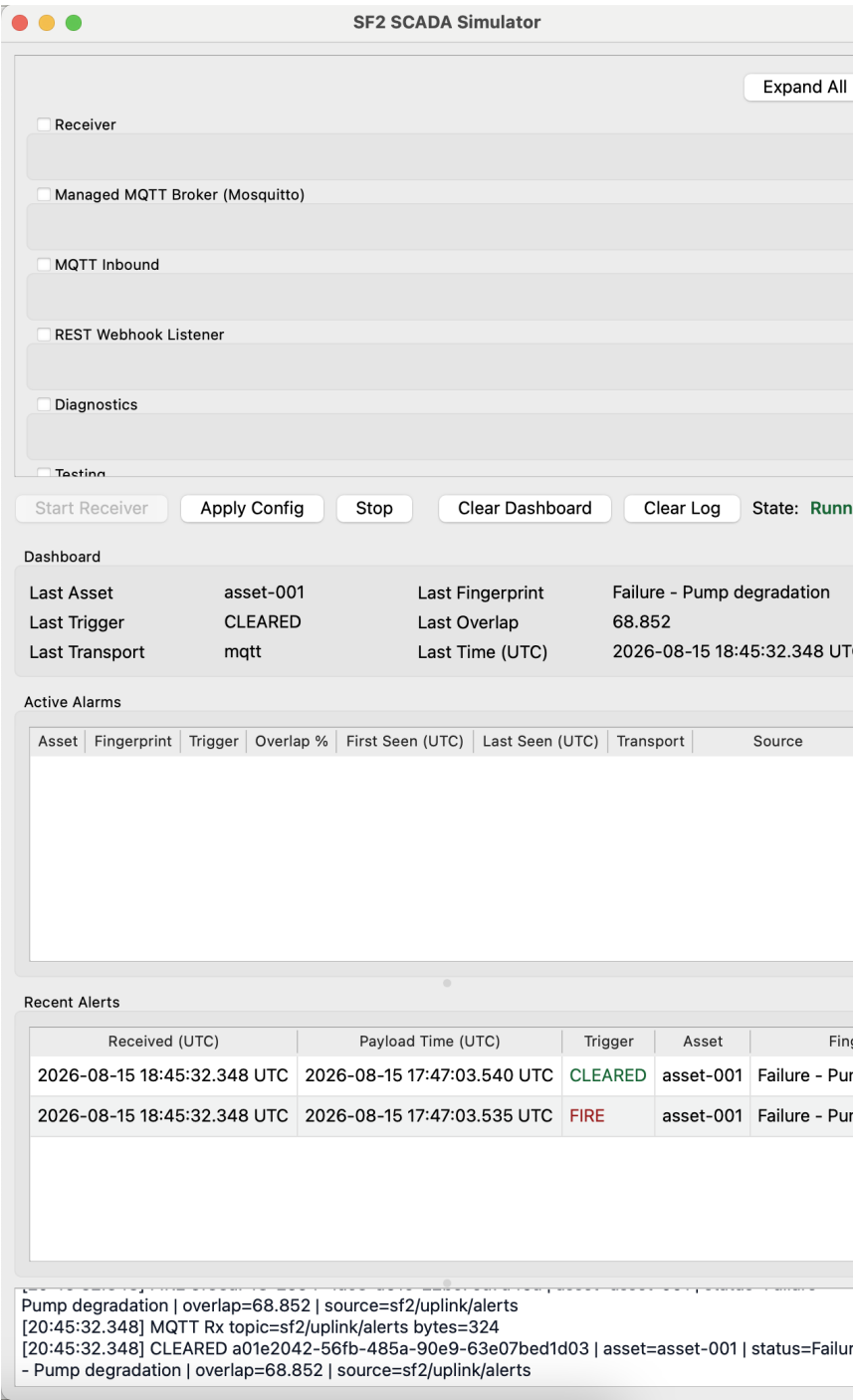


Figure 36. The dashboard contains the real FIRE/CLEARED pair received from the local Engine.

The last displayed overlap, 68.852, belongs to the CLEARED transition after overlap has fallen again. That is why it can be below the 80% firing threshold.

Finally, inspect EngineRT. frames_observed and trigger-edge counters must increase, inbound TCP must be connected, and the gatekeeper lease must remain active.



Figure 37. EngineRT shows the active package, inbound stream, advancing frames, and evaluated trigger edges.

Maker Check: File Stream is Running, EngineRT observes frames from TCP, and SCADA displays the matching MQTT FIRE/CLEARED events.

When Something Gets Stuck: Find the First Broken Contract

Start with the first stage that cannot prove its output. Threshold changes cannot repair a mismatched retina, and restarting Studio cannot repair a stopped source.

Symptom	Inspect first
UDM input is empty or has the wrong columns	UDTAT roles, numeric output, and the deleted <code>sensor_15</code> contract
Heatmap is blank or collapsed	UDM feature selection, input row count, and training completion
Retina is empty or signals are missing	Binned data, map signal set, recipe, and 64 x 64 dimensions
First Studio render is empty	Choose Render Project again
Overlap traces are missing	Keep the Overlap Pane visible and choose Render Project again
Failure fingerprint is noisy or mixed	Select one reviewed record or one complete contiguous phase without mixed regimes
Overlap is visible but no alarm fires	Expand the item and verify Enable trigger , density, and threshold

Symptom	Inspect first
Local click occurs but SCADA receives nothing	Global MQTT/REST publishing, endpoint, receiver state, and transport self-test
Engine does not appear in Fleet	Confirm the local EngineRT process is running, then wait or choose Refresh
Engine stays Ready to adopt	Select Adopt ; do not add its local address manually
Engine is healthy but Waiting	Gatekeeper binding, selected package, and source endpoint
Engine is Offline (license reserved)	Restore the process and use Reconnect , not Release
Engine runs with zero frames	File Stream Running state, TCP 127.0.0.1:19001, and frame age
EngineRT rejects the package	Signing trust, package signature, and artifact hashes

Maker Acceptance: Does the Whole Chain Work?

1. The splash screen opens and the selected data root contains the packaged Waterpump project.
2. Studio opens `project.sf2project.json` with the complete artifact chain.
3. UDTAT is launched from the open project's Preferences Pane and saves the preprocessing recipe used by every downstream stage.
4. UDTAT excludes metadata and status from features, removes `sensor_15`, and writes aligned numeric and binned outputs.
5. UDM trains a populated 64 x 64 map from 220,320 rows and 51 sensor features, with plausible occupancy under hover inspection.
6. UFP generates a matching count-based 64 x 64 dictionary with plausible fingerprints from sparse to dense.
7. Studio renders the recording; **Fit**, zoom, pan, and lasso work while sensor, status, overlap, library, and fingerprint views remain visible.
8. A named reference can be captured from one reviewed record or aggregated from one complete contiguous section.
9. The selected reference is understood across its library entry, source marker, fingerprint colors, and overlap channel.
10. The `Failure1` overlap rise more than four days before the recorded failure is understood as an early similarity trajectory.
11. Threshold selection is validated against real failures and non-failing near approaches, and is kept distinct from transport configuration.
12. The Engine package passes signature, hash, artifact, and 220,320-row smoke validation.
13. The Engine Fleet begins empty and discovers one local Engine automatically.
14. The local Engine is commissioned with **Adopt**, without **Add by address**, and reaches Running.
15. File Stream supplies TCP frames on 127.0.0.1:19001.
16. The SCADA managed broker receives the Engine's MQTT alarm on 127.0.0.1:1883.
17. EngineRT frame and trigger-edge counters advance, and SCADA records the matching **FIRE** and **CLEARED** transitions.