SecuritySpy Custom Model Plugin

Run your own CoreML models on camera frames and receive results via the web event stream.

1. Overview

SecuritySpy's custom model plugin system allows you to supply your own Apple CoreML classification model. SecuritySpy will run it on each camera's video frames at a configurable rate and deliver the raw model output values to connected clients in real time via the web server's event stream.

This enables use cases such as custom object detection, scene classification, anomaly scoring, or any image-based inference task, without modifying SecuritySpy itself.

The custom model uses the same region of interest as SecuritySpy's presence detection feature (the "presence rectangle" configured in the camera's motion detection settings). The image within this rectangle is scaled to the model's input dimensions before inference.

2. Supported Model Formats

Format	Extension	Description
Compiled CoreML bundle	`.mlmodelc`	Pre-compiled model package (a directory). **Recommended** — loads immediately with no compilation delay.
CoreML source model	`.mlmodel`	Uncompiled model file. SecuritySpy will compile it at first use via `MLModel.compileModel(at:)`. This incurs a one-time startup delay.

Recommendation

Use `.mlmodelc` for production. You can generate a compiled model bundle using Xcode or the `coremltools` Python package: `coremltools.utils.compile("model.mlmodel", ".")`

Any other file extension will be rejected with an error.

3. Installation

Place two items in SecuritySpy's home folder:

```
~/SecuritySpy/
├── models.json          ← manifest describing your model
└── YourModel.mlmodelc/  ← (or YourModel.mlmodel)
```

Restart Required

SecuritySpy checks for the existence of `models.json` once at launch. You must quit and restart SecuritySpy after adding or removing the manifest file.

4. The models.json Manifest

The manifest is a JSON array. Currently, only the first element is used (one custom model at a time).

Full Example

```
[
  {
    "fileName": "MyModel.mlmodelc",
    "inputWidth": 224,
    "inputHeight": 224,
    "inputPixelFormat": "BGRA",
    "outputType": "multiArray",
    "rate": 2.0
  }
]
```

Field Reference

Field	Type	Default	Required	Description
`fileName`	string	—	Yes	Model file name. Resolved relative to `~/SecuritySpy/`. Must end in `.mlmodelc` or `.mlmodel`.
`inputWidth`	integer	224	No	Width in pixels of the image fed to the model.
`inputHeight`	integer	224	No	Height in pixels of the image fed to the model.
`inputPixelFormat`	string	`"BGRA"`	No	Pixel format of the input image: `"BGRA"` or `"ARGB"`.
`outputType`	string	`"multiArray"`	No	Must be `"multiArray"`. Reserved for future expansion.
`rate`	float	1.0	No	Inference rate in Hz (executions per second). For example, `2.0` means the model runs twice per second per camera. Lower values reduce CPU/GPU load.

Supported Pixel Formats

Value	Bytes/Pixel	Channel Order
`"BGRA"`	4	Blue, Green, Red, Alpha
`"ARGB"`	4	Alpha, Red, Green, Blue

BGRA is the standard CoreML image format and is recommended for most models.

5. Model Input Specification

Your model must accept a single image input. SecuritySpy automatically reads the input feature name from the model's metadata (`MLModelDescription.inputDescriptionsByName`), so the name you used when creating the model is used automatically.

Property	Value
Input type	Image (`CVPixelBuffer`)
Dimensions	As specified by `inputWidth` × `inputHeight` (default 224×224)
Pixel format	As specified by `inputPixelFormat` (default BGRA)
Source region	The camera's presence detection rectangle, scaled to the model's input dimensions

The image is extracted from the camera's current video frame, cropped to the presence detection rectangle, and scaled to the model's input dimensions using pixel-accurate resampling.

6. Model Output Specification

Your model must produce a single output of type `MLMultiArray` containing floating-point values. SecuritySpy reads the output feature name from the model's metadata automatically.

Property	Value
Output type	`MLMultiArray` (float32)
Maximum elements	8
Value range	Any float, but typically 0.0–1.0 for classification probabilities
Precision delivered	Values are rounded to 2 decimal places when transmitted via the event stream

Output Size Limit

If your model outputs more than 8 values, only the first 8 are used. Design your model accordingly.

There is no built-in threshold or filtering applied to custom model results. All output values are delivered to clients as-is, regardless of magnitude. Your client application should implement any thresholding or interpretation logic.

7. Receiving Results via the Event Stream

Custom model results are delivered through SecuritySpy's web server event stream, the same mechanism used for motion detection events, classification results, and other real-time notifications.

Connecting to the Event Stream

Make an HTTP GET request to:

```
http://<address>:<port>/eventStream?version=3&format=multipart
```

Authentication is required (HTTP Basic Auth or session-based, depending on your SecuritySpy web server configuration).

Parameter	Value	Description
`version`	`3`	Event stream protocol version. Use version 3 or higher to receive custom model events.
`format`	`multipart`	Optional. Use `multipart` for `multipart/mixed` framing with boundaries, or omit for plain text with `\r`-delimited lines.

Event Format

Each event stream line has this general format:

```
<timestamp> <sequence> <camera> <event_type> [data]
```

For custom model results, the event type is `MODELO` :

```
20260305143022 14 3 MODELO 0.85,0.12,0.03
```

Field	Description
`20260305143022`	Timestamp: YYYYMMDDHHmmss (local time)
`14`	Sequence number (per-connection incrementing counter)
`3`	Camera number (0-based index)
`MODELO`	Event type identifier for custom model results
`0.85,0.12,0.03`	Comma-separated model output values (2 decimal places each)

The number of values in the comma-separated list matches the number of elements in your model's output `MLMultiArray` (up to 8).

Multipart Format

When using `format=multipart` , each event is wrapped in a MIME multipart boundary:

```
--eventStreamBoundary
Content-Type: text/plain
Content-Length: 42

20260305143022 14 3 MODELO 0.85,0.12,0.03
```

Plain Text Format

Without the `format=multipart` parameter, events are delivered as plain text lines terminated by `\r` (carriage return).

8. Complete Example

Step 1: Create or Obtain a CoreML Model

Using Python with `coremltools` , convert a trained model to CoreML format:

```
import coremltools as ct

# Example: convert a PyTorch/TensorFlow model
# The model should output a MultiArray (not a dictionary/class labels)
model = ct.convert(your_traced_model,
                   inputs=[ct.ImageType(shape=(1, 3, 224, 224))])

# Save and compile
model.save("FireDetector.mlmodel")
compiled = ct.utils.compile("FireDetector.mlmodel", ".")
# This creates FireDetector.mlmodelc/
```

Step 2: Create models.json

```
[
  {
    "fileName": "FireDetector.mlmodelc",
    "inputWidth": 224,
    "inputHeight": 224,
    "inputPixelFormat": "BGRA",
    "outputType": "multiArray",
    "rate": 1.0
  }
]
```

Step 3: Install

Copy both `models.json` and `FireDetector.mlmodelc/` to:

```
~/SecuritySpy/
```

Restart SecuritySpy.

Step 4: Consume Results

Connect to the event stream and parse `MODELO` events. Here is a minimal JavaScript example:

```
async function connectEventStream(host, port, username, password) {
  const auth = btoa(`${username}:${password}`);
  const url = `http://${host}:${port}/eventStream?version=3`;

  const response = await fetch(url, {
    headers: { "Authorization": `Basic ${auth}` }
  });

  const reader = response.body.getReader();
  const decoder = new TextDecoder();
  let buffer = "";

  while (true) {
    const { done, value } = await reader.read();
    if (done) break;

    buffer += decoder.decode(value, { stream: true });
    const lines = buffer.split("\r");
    buffer = lines.pop();

    for (const line of lines) {
      const trimmed = line.trim();
      if (!trimmed) continue;

      const match = trimmed.match(
        /^(\d{14}) (\d+) (\d+) MODELO (.+)$/
      );

      if (match) {
        const [, timestamp, seq, camera, valuesStr] = match;
        const values = valuesStr.split(",").map(Number);

        console.log(`Camera ${camera}: ${JSON.stringify(values)}`);

        // Apply your own thresholds:
        if (values[0] > 0.8) {
          console.log("High confidence detection on camera " + camera);
        }
      }
    }
  }
}
```

9. Limitations

- **One custom model at a time.** Only the first entry in the `models.json` array is loaded.
- **Maximum 8 output values.** If the model's output array has more than 8 elements, the excess is ignored.
- **Restart required.** Adding or removing `models.json` requires an application restart. The file is checked once at launch.
- **Presence rectangle required.** The custom model uses the camera's presence detection rectangle as its region of interest. If no presence rectangle is configured, custom model detection is disabled for that camera.
- **No built-in thresholding.** All output values are delivered to clients regardless of magnitude. Implement filtering in your client.
- **CoreML only.** Models must be in Apple CoreML format. Convert from other frameworks (PyTorch, TensorFlow, ONNX) using coremltools.
- **Model auto-release.** If no camera uses the custom model for 30 minutes, it is unloaded from memory. It will be reloaded automatically when next needed.
- **Output precision.** Values are transmitted with 2 decimal places (e.g. `0.85`), providing 0.01 resolution in the event stream.