Rust ↔️ JS Glue

wasm_bindgen, and why you might want it.

Jon Hardie

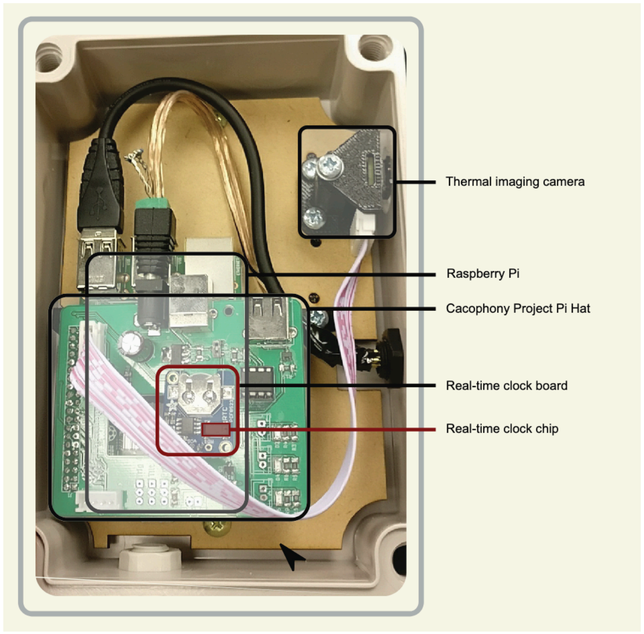
Background

The Cacophony Project: cacophony.org.nz

- We're an open source hardware/software non-profit, started around the beginning of 2016.
- Initially created a prototype of a "bird monitor" to listen to and index bird song levels in an area.
- Predator free 2050 was announced mid 2017.
- New Zealand is sparsely populated, predator control is labour intensive. Can we automate?
- First, can we correlate birdsong levels with levels of predator activity?
  - Traditional monitoring solutions: tracking tunnels and chew cards. Cheap, low tech, not that comprehensive, and require labour to check and collate data.
  - Commercially available infrared trail-cams trialed, but found not to be sensitive to small/fast moving predators, missing a lot of activity. Mostly designed for hunting and larger animals such as deer.

First thermal camera prototype

- A raspberry pi based prototype of a thermal camera using a CNN for recognition of predators.



Trivia: there's a picture of this early device in "Rust in Action" by Tim McNamara, in a section talking about RTCs. This predates the use of any Rust code on the camera though.

- Thermal camera uses a FLIR Lepton 3.5 sensor (160x120 resolution)
- Sweet spot with regard to resolution vs price
- Records in a temperature scale with around 14bits of resolution per pixel

- We wanted to retain as much of that raw data as possible, just in case the AI can make use of it.
- Created a simple file format “CPTV” packing delta encoded pixel data into a gzip file, along with some frame metadata.

~$160USD

The big problem

No tagged dataset available for thermal imagery of NZ predator species.

- No way to train a computer vision model.

Our solution

- A web portal was created to gather thermal recordings and provide an interface for users to review and tag animals, so that we can begin to train and improve a model.

Background

Thermal recording processing pipeline

- Originally uploaded the CPTV recording encoded in a Go program, decoded on the server side by a python program, and then encoded to colourised MP4 using ffmpeg, for playing on the web.
- This worked, but meant more server resources were required, and there was a delay depending on the length of the queue. We also have to save both CPTV and MP4 files to object storage, which costs us more. MP4 files were often bigger than the CPTV files they were derived from.
- This is where I joined the project, initially as an open-source contributor early 2019 (later as an employee in March 2020)
- Idea: just serve the CPTV file to the web and render it to a <canvas>?

- Wrote a Rust CPTV decoder, used wasm_bindgen, drew it to a <canvas>, job done?

- Buuuut, could have just done that in JS using typed arrays. Bit twiddling is a bit painful in JS, but it works.
- Maybe it would be useful to have just one reasonably fast implementation of the CPTV codec, and share that code around?
- Existing impls unzip the gzip container to an intermediate buffer, but gzip is a streaming format - do we really need to use all that extra memory?
- Made a streaming gzip decoder in Rust. That would be quite hard to do in JS land. (Not sure any implementation existed prior to 2021. Multiple high quality implementations in Rust.)
- Aside: CPTV format to this day still doesn’t allow arbitrary seeking in the stream, but as most clips we deal with are quite short, we’ve not seen fit to address that yet.

- Replaced python implementation of CPTV decoder with Rust implementation wrapped with py03. Used in our ML pipeline.
- We use the wasm implementation in the browser, as well as in our NodeJS backend (to generate thumbnails as tracks are automatically classified)

Demo time

JON'S BACKYARD

Activity

Visits

Date range

Last week

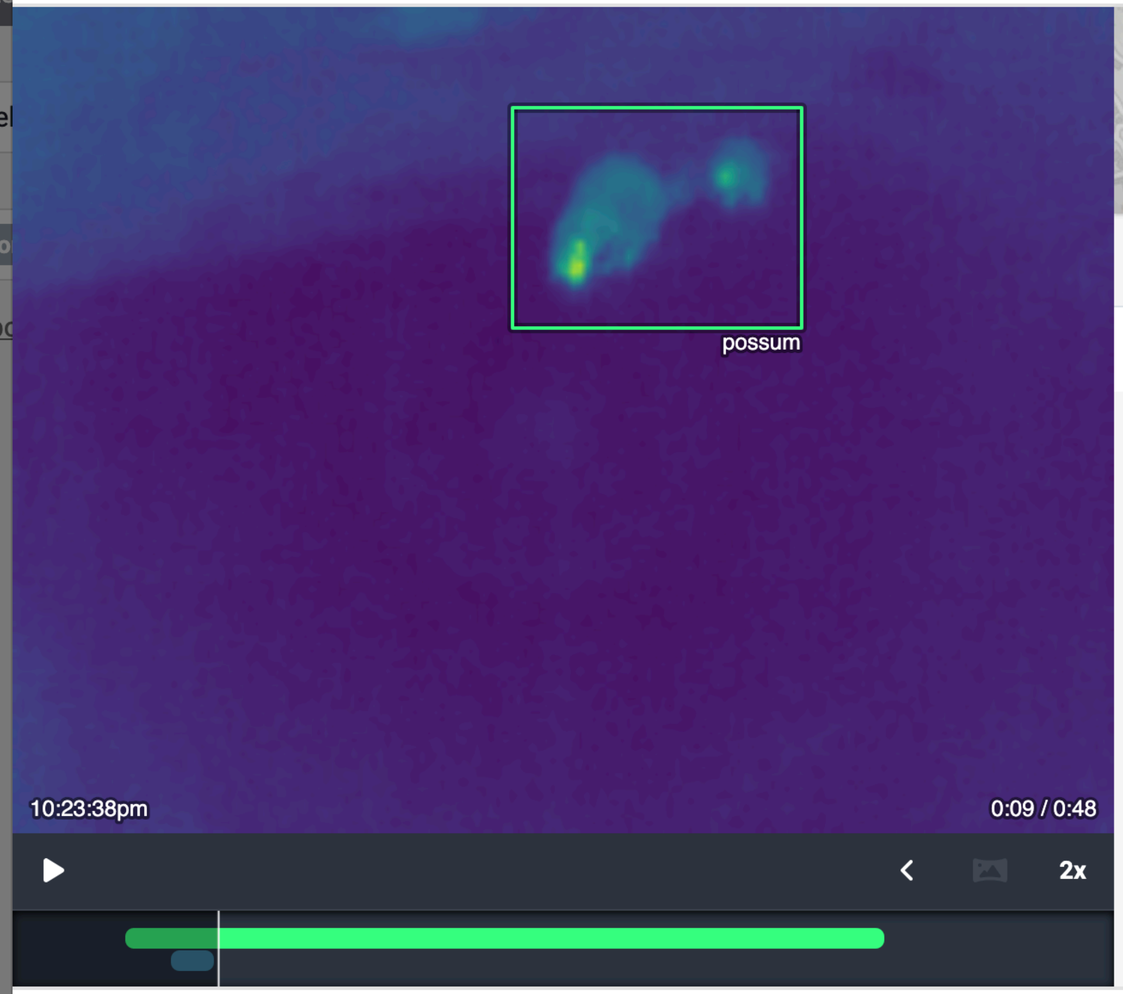
Locations

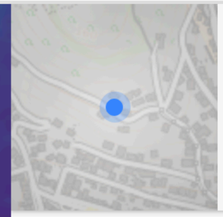
All locations

Export

THERMAL RECORDING

10:23–10:24pm (48 seconds)





jon-dev

Lyttelton Heights

15/05/2024

10:23:28pm

Tracks | Labels (2) | Notes (0)

1 MANUAL ID
Possum

2 MANUAL ID
Possum

JON'S BACKYARD

Activity

Visits

Date range

Last week

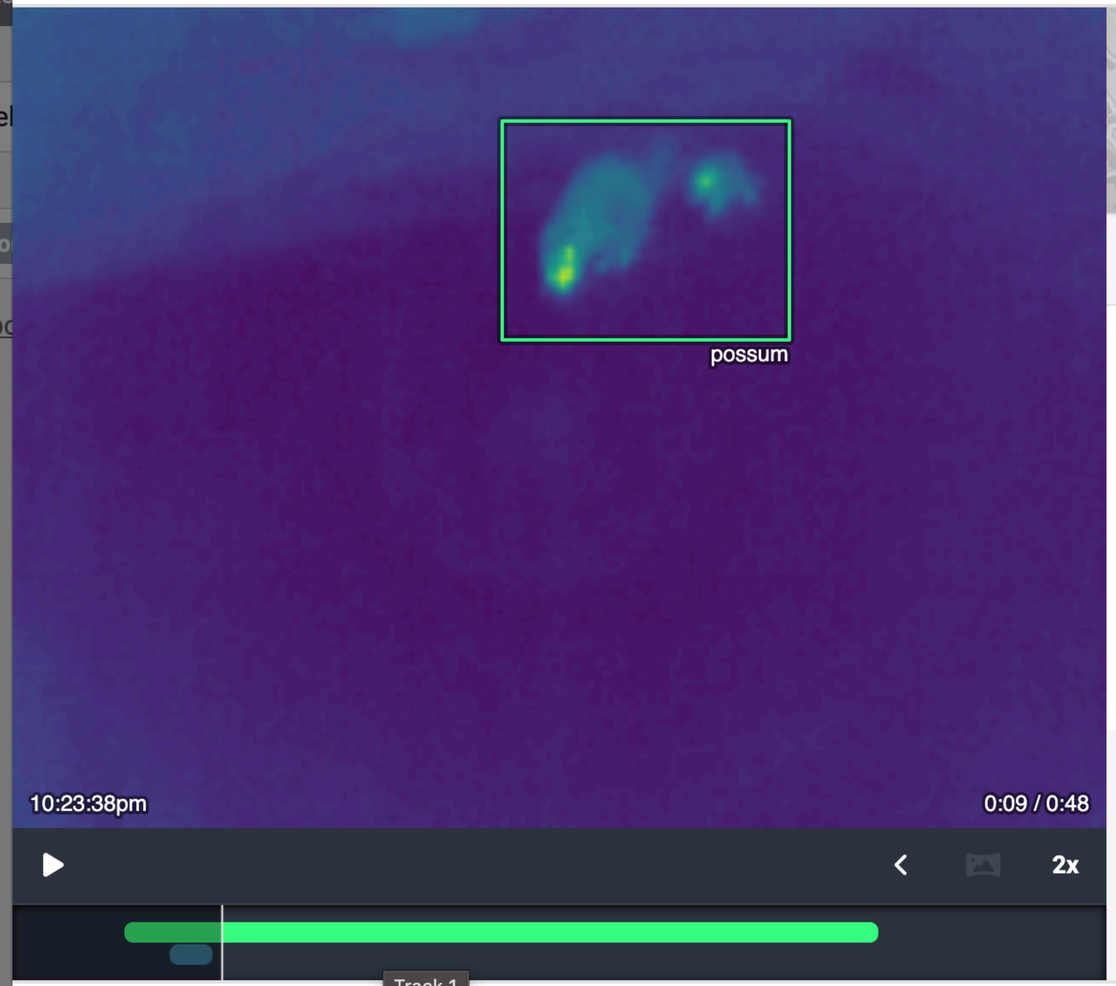
Locations

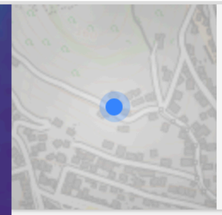
All locations

Export

THERMAL RECORDING

10:23–10:24pm (48 seconds)





jon-dev

Lyttelton Heights

15/05/2024

10:23:28pm

Tracks | Labels (2) | Notes (0)

1 MANUAL ID

Possum

Possum | Rodent | Hedgehog | Cat | Bird

False Trigger | Not Identifiable | Human | +

View details

2 MANUAL ID

Possum

What is WASM anyway?

- WebAssembly is a bytecode format to run code written in non-GC languages at near-native speeds on the web. Sandboxed and safe.
- First release in 2017
- Predated by asm.js in 2010, which established that this was something people were excited about.
- Initially shipped without support for nice things like SIMD, but this was added a few years ago
- Still only supports a 32bit address space, but 64bit is coming, along with support for GC’d languages

WASM in Rust - wasm32-unknown-unknown

- Added robust support with the Rust 2018 Edition, with the addition of wasm_bindgen and wasm_pack

```toml
[package]
name = "cptv-decoder"
version = "0.1.0"
edition = "2024"

[lib]
crate-type = ["cdylib", "lib"]

[dependencies]
codec = { path = "../cptv-codec-rs" }
js-sys = "0.3.77"
wasm-bindgen = "0.2.100"
serde-wasm-bindgen = "0.6.5"
# Routing messages to the JS error console
console_log = "1.0.0"
console_error_panic_hook = "0.1.7"
log = "0.4.27"
web-sys = { version = "0.3.77", features = ["ReadableStreamDefaultReader"] }
# Futures backed by JS Promises
wasm-bindgen-futures = "0.4.50"
```

- Building as a lib, not a binary
- We need to annotate the Rust functions we want to expose to JS.
- wasm_bindgen is responsible for this.
- wasm-pack is a build tool that runs wasm_bindgen with the options we want to generate JS wrapper files appropriate to the target environment, typically the browser or NodeJS. Also debug vs release etc.
- The generated JS is pretty ugly, but it seems to get the job done.

```
import loadWasm, {CptvDecoderContext} from "../pkg/cptv_decoder.js";
import fs from "fs";

(async function main() {
    await loadWasm(fs.readFileSync("../pkg/cptv_decoder_bg.wasm"));
    const buffer = fs.readFileSync("../../cptv-codec-rs/tests/fixtures/2025-06-11--15-28-18.cptv");
    const reader = new StreamReader(buffer, 100000);
    const decoderContext = CptvDecoderContext.newWithReadableStream(reader);
    const header = await decoderContext.getHeader();
    console.log(header);
    let frame;
    let num = 0;
    while ((frame = await decoderContext.nextFrame())) {
        // Do stuff with the frame here.
        console.log(frame);
        num++;
    }
    console.log(`Total frames: ${num}`);
    decoderContext.free()
}());
```

Top reasons to reach for wasm

- You just don’t want to write JS
  - Any kind of manipulation of binary formats
  - Dealing with slices of bytes is just more ergonomic in Rust IMO
- Code reuse across different systems by keeping core logic in Rust.
- Speed (Especially with SIMD?)
  - Wasm can hit up to ~90% of native speed, but YMMV.
- *Deterministic* performance characteristics (avoiding GC, and JIT ramp up)
- Use high quality libs from Rust ecosystem.
- In NodeJS, you don’t have to deal with node-gyp etc for compiling native modules, for only a small speed penalty. You also get cross-arch, cross platform support from a single module.

Caveats and limitations

- After you've finished using your wasm module, you need to manually .free() it in your JS code.
  - If you don't you'll leak memory.
  - If your rust/wasm module panics you won't be able to call .free() !!!
    - So don't panic!
    - Or, maybe run it in a WebWorker, and just rely on the WebWorker cleanup to free the memory?
- Code size - you might want panic = "abort", but then you need to deal with not having stack traces.
- No multithreading out of the box. You can share your .wasm file among Workers, but coordination is up to you. You can't just .par_iter() with rayon.
    - It's possible with nightly features (+atomics), and using SharedArrayBuffer in a "secure context"
    - But no drop-in pthreads compatible API yet.
    - https://github.com/RReverser/wasm-bindgen-rayon might help, I haven't used it.

Memory ownership

- We’re using Rust for speed, so we typically want to avoid copying large bits of memory around.
- You can pass in references to JS memory using js_sys types.
- If a Rust function takes a buf: &[u8], and you pass a Uint8Array from JS, the array will be copied into wasm owned linear memory.
- You can reference the JS owned Uint8Array by making your function take that as it’s arg.
- You can use unsafe { Uint8Array::view(my_u8_slice) } to pass a reference to wasm owned memory to JS, but you need to take care around lifetimes of the underlying buffer backing the TypedArray.

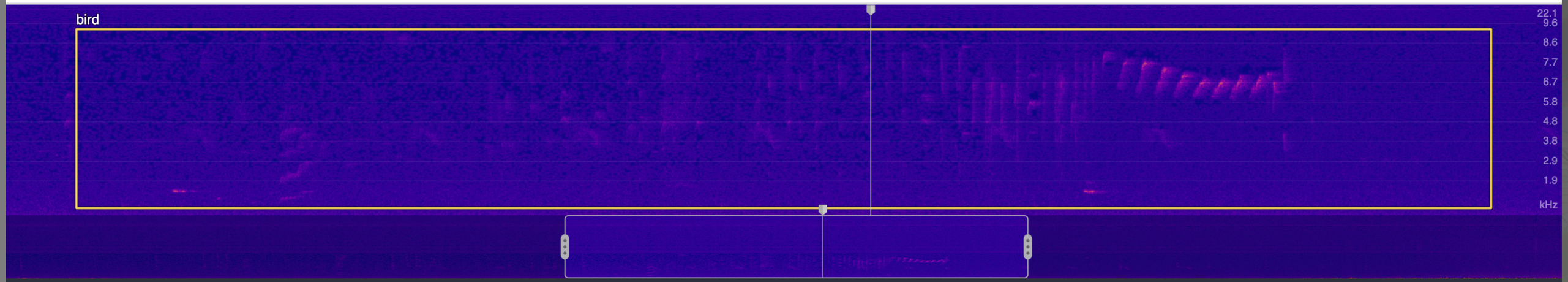
Interactive spectrogram tagging

Training bird classification

- Thermal predator recognition is mostly working pretty nicely.
  - We have nearly 1.7 million thermal recordings in our database, with around 170,000 user-tagged predator tracks.
- **We developed a new lower power thermal camera for mass production, with a built-in bird monitor.** Embedded firmware in Rust.
- We want to do recognition of bird species, not just “noise levels” which we were previously using as a proxy for bird song.
  - There’s not a great datasource of tagged NZ bird calls to train a model…


B197
B253
Li-ion BATTERY
Warning
A307

AUDIO RECORDING



0:31 / 1:00

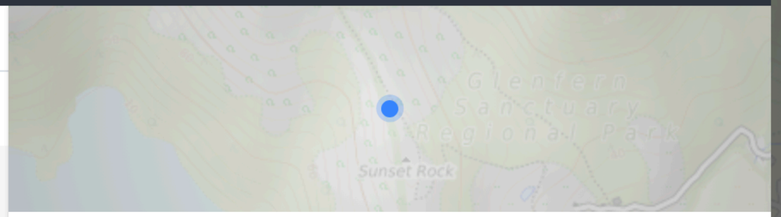
add track

resize

Tracks | Labels (0) | Notes (0)

Show 2 hidden noise tracks

1	AI CLASSIFICATION Fantail		
2	AI CLASSIFICATION Fantail		
3	AI CLASSIFICATION Bird	Confirm	
4	AI CLASSIFICATION Bird		



A132

New station for A048_20...

10/11/2025

02:19:00pm

Next recording
14/15

Audio tagging spectrogram

- Uses the excellent RustFFT library.
- You can already do FFT using the WebAudio API. An earlier implementation did use this.
- But it’s not multithreaded, and maybe not as fast as RustFFT using wasm with SIMD.
- We use the WebAudio API to decode our audio file (.m4a/AAC) into a Float32Array buffer. This part is sadly the bottleneck, it’s single threaded.
- Pass overlapping slices of the audio buffer to WebWorkers, run FFT in wasm.
- Also pass in disjoint slices of the target Float32Array buffer to Rust/wasm to write out the FFT results to.
  - Can even avoid copies if using SharedArrayBuffer!
- When all Worker threads complete, pass off the target buffer to WebGL to render in a glsl shader.

```rust
#[wasm_bindgen]
pub struct FftContext {
    fft: Arc<dyn Fft<f32>>,
    filter: [f32; CHUNK_LEN],
    scratch: [Complex<f32>; CHUNK_LEN],
    filtered: [Complex<f32>; CHUNK_LEN],
}

#[wasm_bindgen]
impl FftContext {
    pub fn new() -> FftContext {
        init_console();
        let mut planner = FftPlanner::new();
        let fft = planner.plan_fft_forward(CHUNK_LEN);
        FftContext {
            fft,
            filter: init_blackman_harris(),
            scratch: [Complex{re: 0.0f32, im: 0.0f32}; CHUNK_LEN],
            filtered: [Complex{re: 0.0f32, im: 0.0f32}; CHUNK_LEN],
        }
    }

    #[wasm_bindgen(js_name = processAudio)]
    pub fn process_audio(&mut self, prelude: &[f32], data: &[f32], output: &mut [f32]) {
      // Do the work
    }
```

```
import {initSync, FftContext} from "./pkg/spectastiq.js";
let fftContext;
let output;
self.onmessage = async ({data}) => {
  if (data && data.type === "Process") {
    const outputLength = data.offsets.outEnd - data.offsets.outStart;
    if (!output || output && output.length !== outputLength) {
      output = new Float32Array(outputLength);
    }
    data.output = output;
    fftContext.processAudio(data.prelude, data.data, data.output);
    self.postMessage({
      id: data.id,
      n: self.name,
      offsets: data.offsets,
      output: data.output
    });
  } else if (data && data.type === "Init") {
    initSync({ module: data.wasm });
    self.name = data.name;
    fftContext = FftContext.new();
    self.postMessage({
      id: data.id,
      m: "Inited",
      n: data.name
    });
  }
}
```

So, can we correlate bird-song with predator activity?

- We’re still working on it!
- Science types at Lincoln university are helping out.
- We could position as a “set and forget” biodiversity monitor, with automated reporting. Farmers are interested in a solution like this.
- Our manufacturing partner 2040 Ltd is selling more cameras, and word of mouth seems to be spreading. Still small-volume manufacturing.
    - The hope is for the project to benefit from money donated back from the proceeds of these sales and eventually become self-funded.
- Overseas ecology efforts look to NZ for “best practices”, and we’re seeing more uptake from offshore (time to fix those subtle timezone bugs!)

Conservation

‘There were stoats in kitchen cupboards’: AI deployed to help save Orkney’s birds

Stoats have been an existential threat to Orkney’s rare birds but technology is helping to eradicate them

Severin Carrell *Scotland editor*

Sat 18 Oct 2025 13.00 BST

Bonus slide

Other work

- An ongoing question - can we use thermal monitoring data to better inform the design and development of more effective traps?
- Some ideas that dovetail in with AI assisted thermal monitoring: Can we make a smart trap that only activates when target species are detected? Design of current traps is constrained by the need to reduce bycatch of non-target species. Often small apertures are used to discourage birds from entering. Predators also don’t like entering small enclosed spaces though.
- Development of an “open-architecture” trap, supported by AI thermal vision.
- People hate lugging batteries (and they’re hard to ship), working towards a solar solution appropriate for our kit.

Questions?