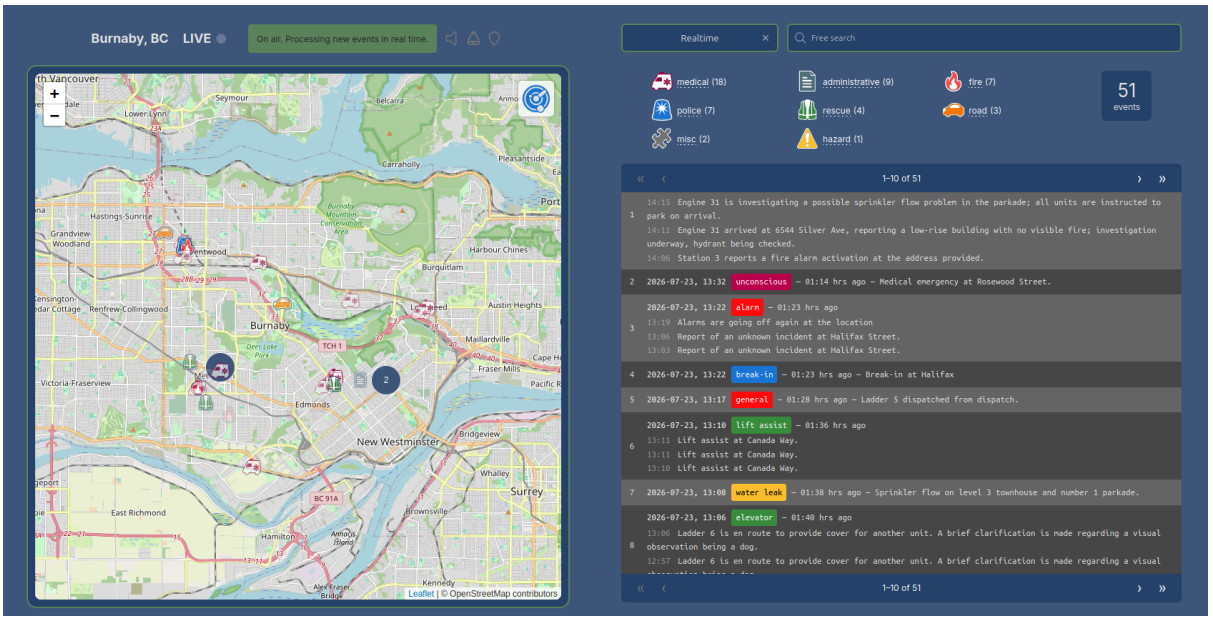
Building an Incident Map with Deep SDR

Like many others, I like listening to my local EMS/Fire Dispatch radio talk. In a perplexing way, it gives me a sense of awareness and attachment to the community. Then one day I thought that I wanted to visualize and analyze it, which would add a historical 4D angle (time) into the equation.

If there are repeating vehicle collisions at the same spot, there may be an issue with road design. Fires happening in the same place tell a story. Many overdoses happening near a place I am looking to rent would raise my eyebrows.

I built the incident map to track those patterns, and since what's nowadays known as AI (ML, LLMs) has been used, it may be appropriate to call it Deep SDR. I believe that this data accumulation is extremely useful not just personally, but to local communities at large. I would like to walk you through the key components. At a glance it's a pipeline that looks like this: capture > transcribe > structure > group > visualize.

The image below shows the map view that displays individual incidents.



Audio Capture.

We can tune in to the local radio comms using an SDR dongle, but there's a very efficient way to bypass creating a stream with something like IMBE-ASR (https://github.com/trunk-reporter/imbe-asr). In plain language, it doesn't make an audio feed, and allows us to transcribe digital bitstream to text directly. It's a great option if you want to optimize resources, don't want to listen to the radio, and are focused only on transcription.

However, for my preferences, it's not optimal. I love listening to the radio, and I would like to be able to process external audio streams, not just the one I host. This is why I needed to reconstruct the audio first, which I did using GNU Radio. Then I opened a feed with Icecast,

which provides a Broadcastify-like player with audio pre-buffering, and simple HTML play/pause controls.

Transcription

Once we get the audio stream going, we can transcribe it using open source transcription models. I tried Whisper and Moonshine, and Whisper works really great while Moonshine didn't give me any good transcription. I tried several Whisper versions: tiny, base, small, medium, large-v3, faster_whisper. For cleaner voice they were all quite accurate. For less clear voice, large-v3 seemed to make fewer mistakes, but small wasn't too bad either. Depending on the resources you have, you can run Whisper with or without a GPU, and the results would still be decent.

LLMs

Before querying an LLM, you need to look through your data to design your categories. For example, I use the following categories: medical, administrative, fire, rescue, road, police, investigation, hazard, hazmat, animal, marine and misc, as seen in the image below.

I ask an LLM to give me a structured output (JSON) that summarizes the incident, assigns a category, etc. Here you also can prompt the LLM to redact addresses and names for privacy.

LLMs can be hosted locally. It won't be those gigantic GPT models comparable to Grok, ChatGPT or Claude, but for summarization tasks LLMs work very well. I genuinely don't believe that LLMs are realistically ready to replace too many engineering jobs, not in their current form anyway, but organizing natural speech into structured output is one of the most useful and realistic use cases for modern LLMs.

Grouping Connected Incidents

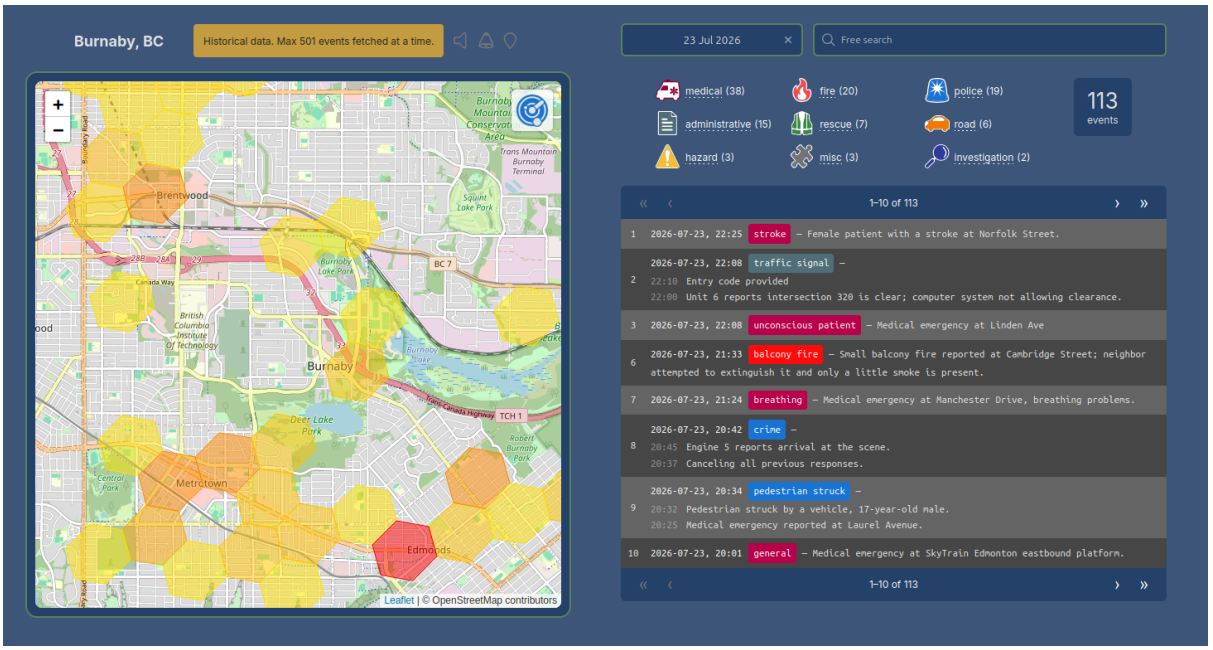
Incidents spread over time. It may take an hour or more for an event to extend. EMS can talk about an event, then not mention it for five minutes, and talk about it again. Without previous context, an LLM would treat it as a new event. You can include context in your prompt, but it will also make the prompt longer and more demanding for your LLM. Once upon a time programmers counted bytes of RAM, and programs were highly optimized. We're probably in a similar stage with LLM prompts right now as we may want to be economical. Probably some years from now it won't be a concern for this sort of tasks anymore.

If you don’t include context in your prompt, you’d need to rely on incident attributes, such as address, time proximity, etc. I do a combination of prompting and attributes to group connected incidents. I am not entirely satisfied with my grouping, so it’s one of those things that need extra thinking.

Hexagonal Grouping In Front-End

We don’t always benefit from displaying results as points on a map. Albeit I find it very engaging to see where exactly each incident happened, analytically it may be more useful to group events into hexagons. This is also more privacy oriented. We’d assign each incident to precalculated hexagons that cover the entire map, and let Leaflet show hexagons. Unfortunately this also means that displaying categories directly on the map will need to go away. We can still see incident icons on hover, but the default user facing view will be incident density.

The image below confirms how incidents tend to congregate around subway stations.



I had people reaching out to me with ideas about how this can be used, including improving response to natural disasters and optimizing EMS resources. The system needs fine-tuning for each case, but if you want to implement it for your area or you are ready to contribute to the development, please reach out to me directly using this form: https://deep-sdr.github.io/