TorchSig: A GNU Radio Block and New Spectrogram Tools for Augmenting ML Training

Phil Vallance, LTS
Ere Oh, LTS
Justin Mullins, LTS
Manbir Gulati, Omega ML
Jared Hoffman, Applied Insight
Matt Carrick, Peraton Labs

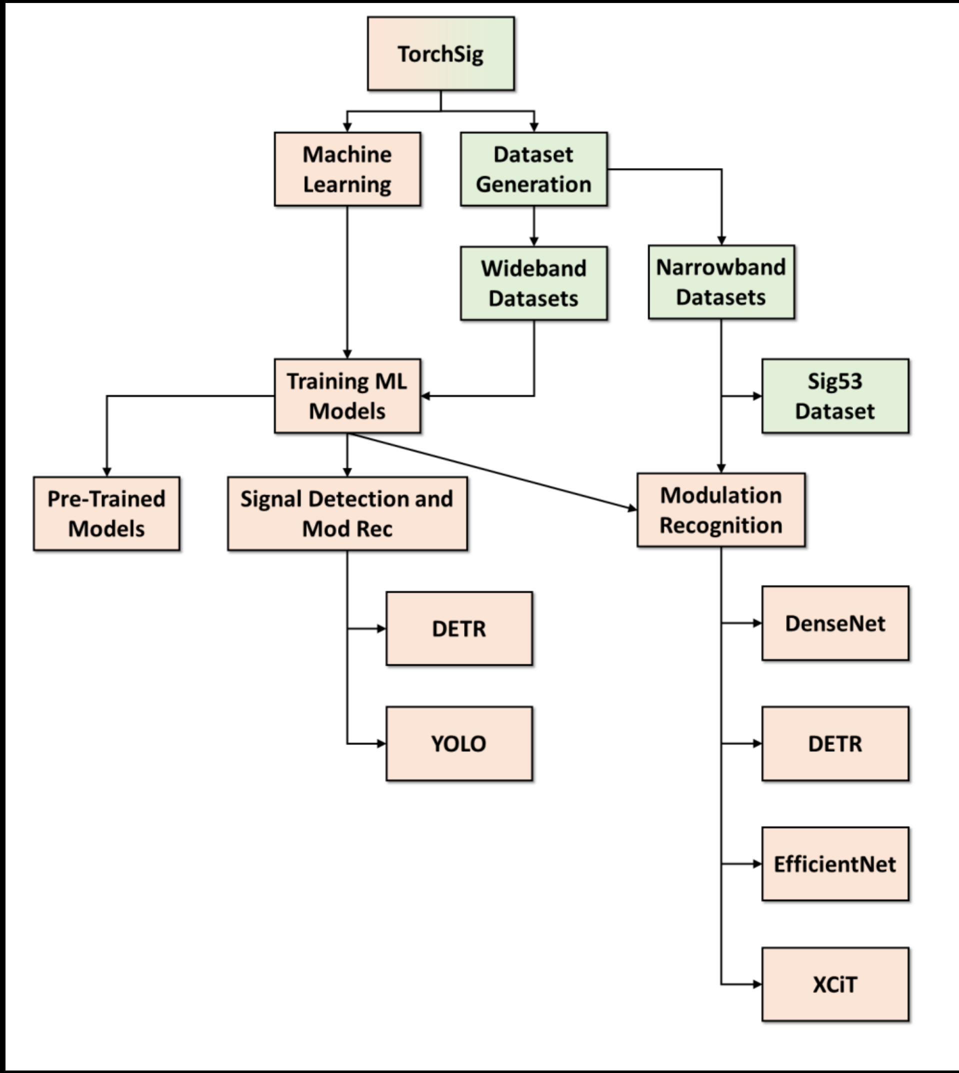
TORCHSIG

GRCon 2024

September 20, 2024

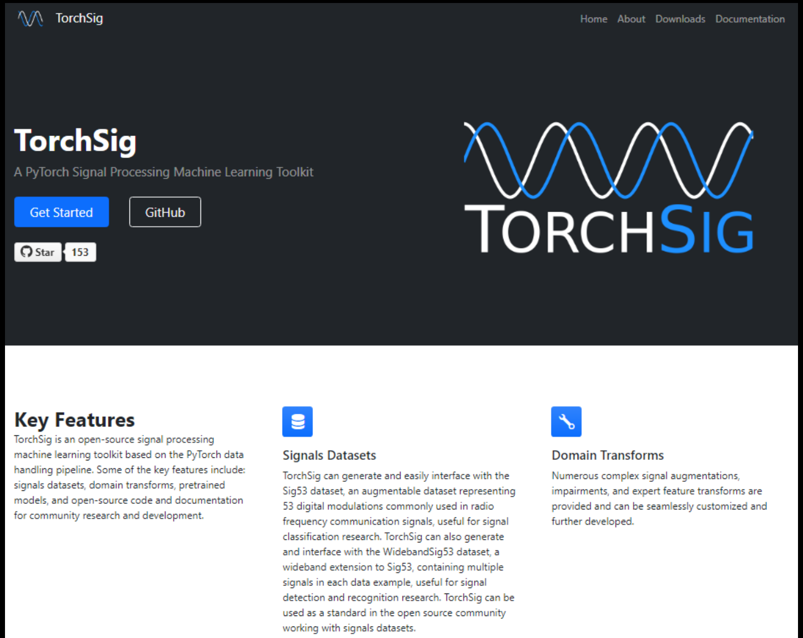
What is TorchSig?

- A system for RF machine learning
  - Training new ML models
  - Models for energy detection and mod rec
  - Narrowband and wideband dataset generation
  - Pretrained, downloadable models

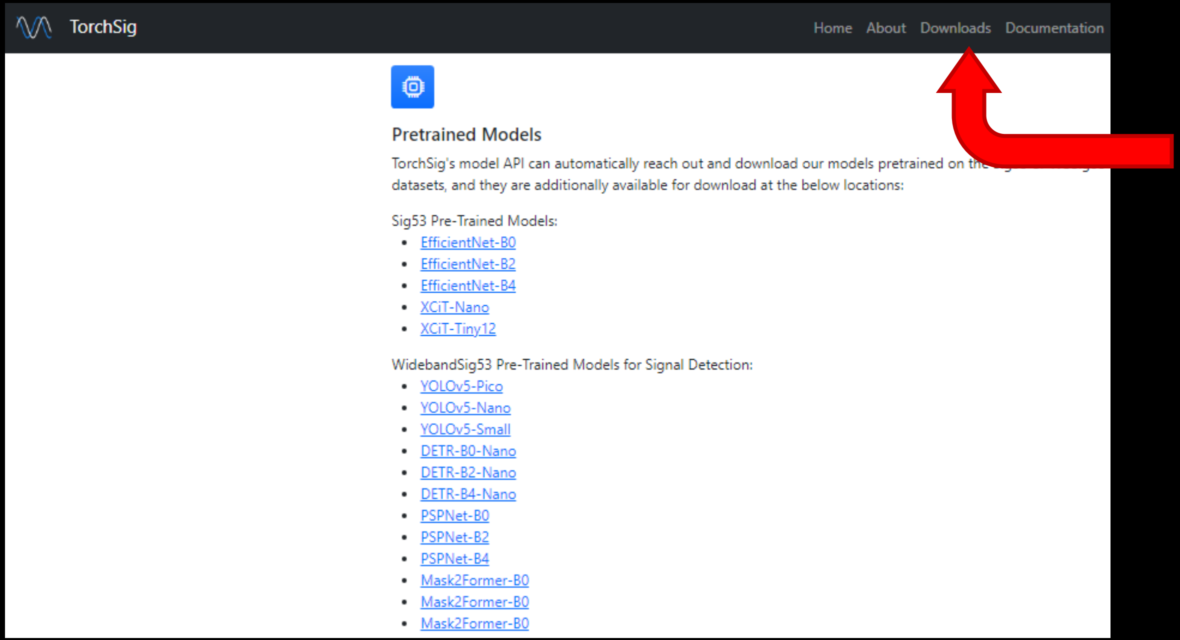


TorchSig.com

What is TorchSig? (cont.)



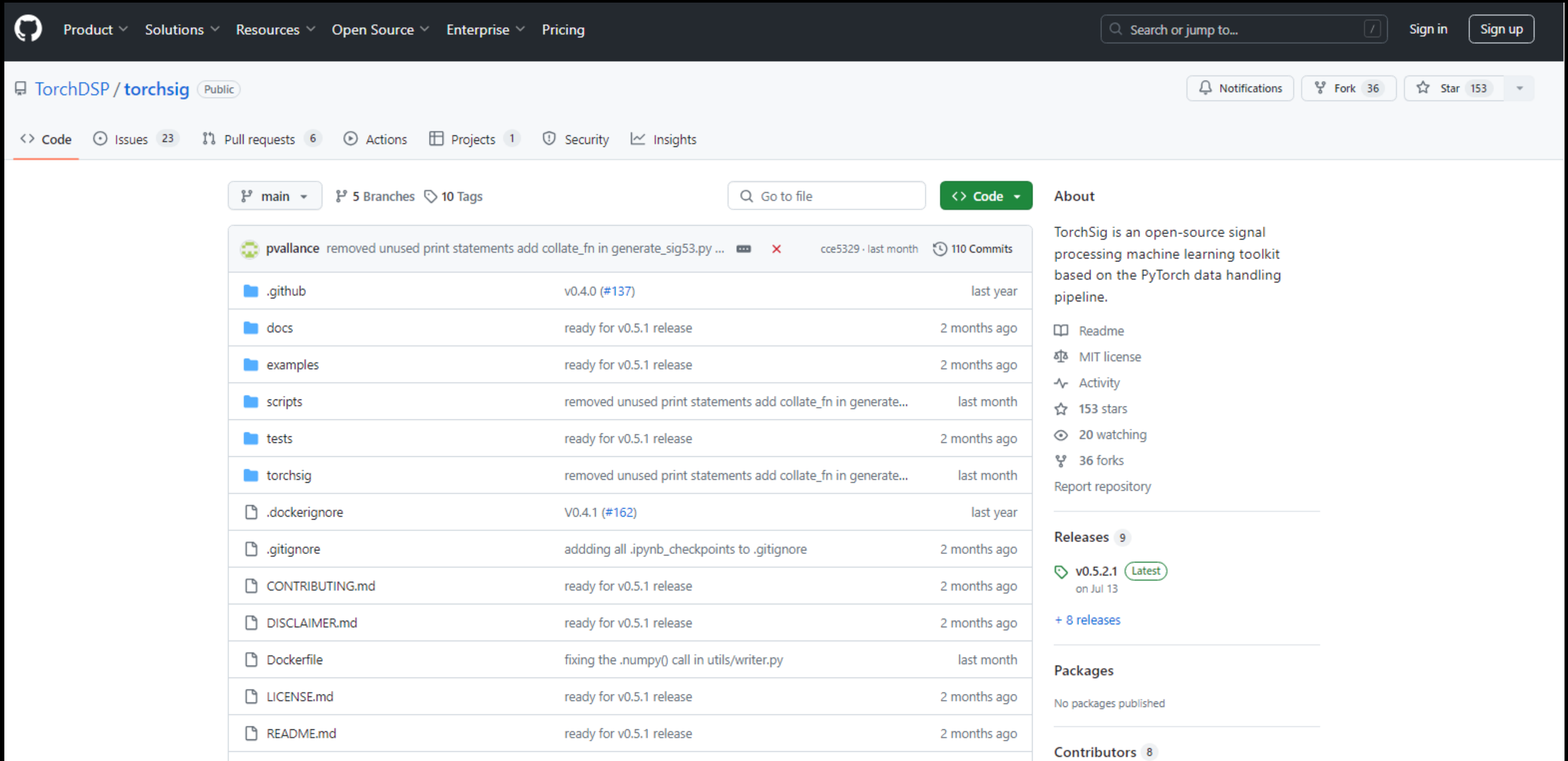
TorchSig.com



Papers & Pretrained Models (models under development and likely to be replaced & reworked soon)

What is TorchSig? (cont.)

- Code available github.com/torchdsp/torchsig



Machine Learning Models

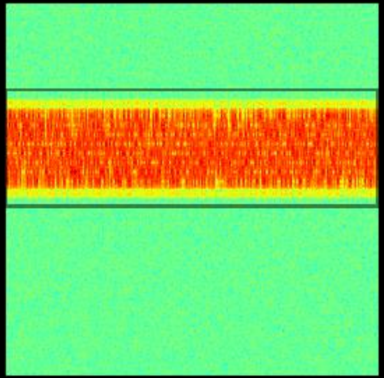
- TorchSig provides two types of models:
- “Narrowband” or modulation recognition model
  - Operates on IQ samples
  - Assumes signal is already channelized, basebanded and roughly time-aligned by an energy detector
- “Wideband” model
  - Operates on spectrograms
  - Able to locate multiple signals in time and frequency
  - Also has modulation recognition feature

Dataset Generation and Sig53

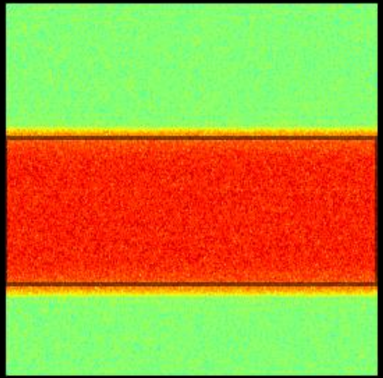
- Signal modulators:
  - Frequency Shift Keying (FSK and GFSK)
  - Minimum Shift Keying (MSK and GMSK)
  - Quadrature Amplitude Modulation (QAM)
  - Phase Shift Keying (PSK)
  - Pulse Amplitude Modulation (PAM)
  - On-Off Keying (OOK)
  - Orthogonal Frequency Division Multiplexing (OFDM)
- Narrowband datasets: single signal channelized and time-aligned at complex baseband
- Wideband datasets: multiple constant-wave (continuous) or bursty signals across a wide bandwidth

Dataset Generation and Sig53 (cont.)

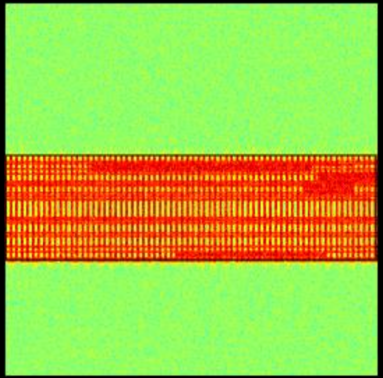
- Examples of wideband signals:



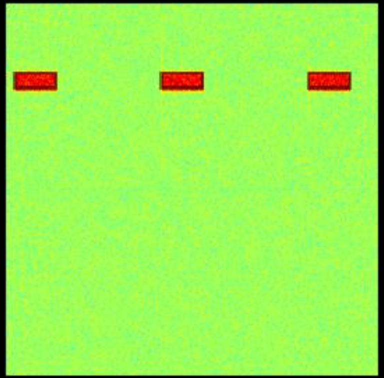
8-FSK



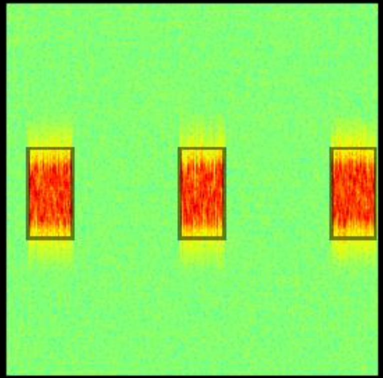
128-QAM



OFDM w/ 256 Subcarriers

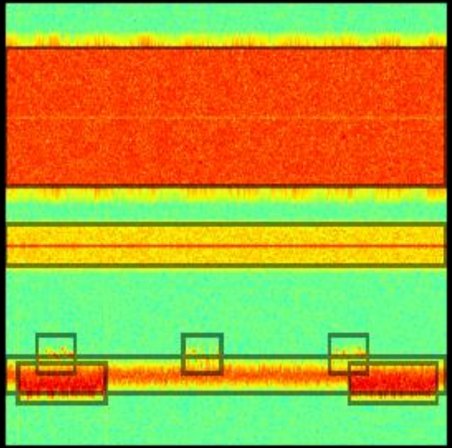


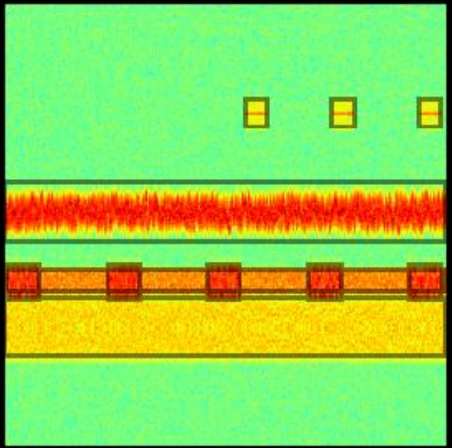
16-QAM



16-MSK

32-PAM, 2-GFSK, 4-GFSK, 16-GMSK, OFDM w/ 72 Subcarriers





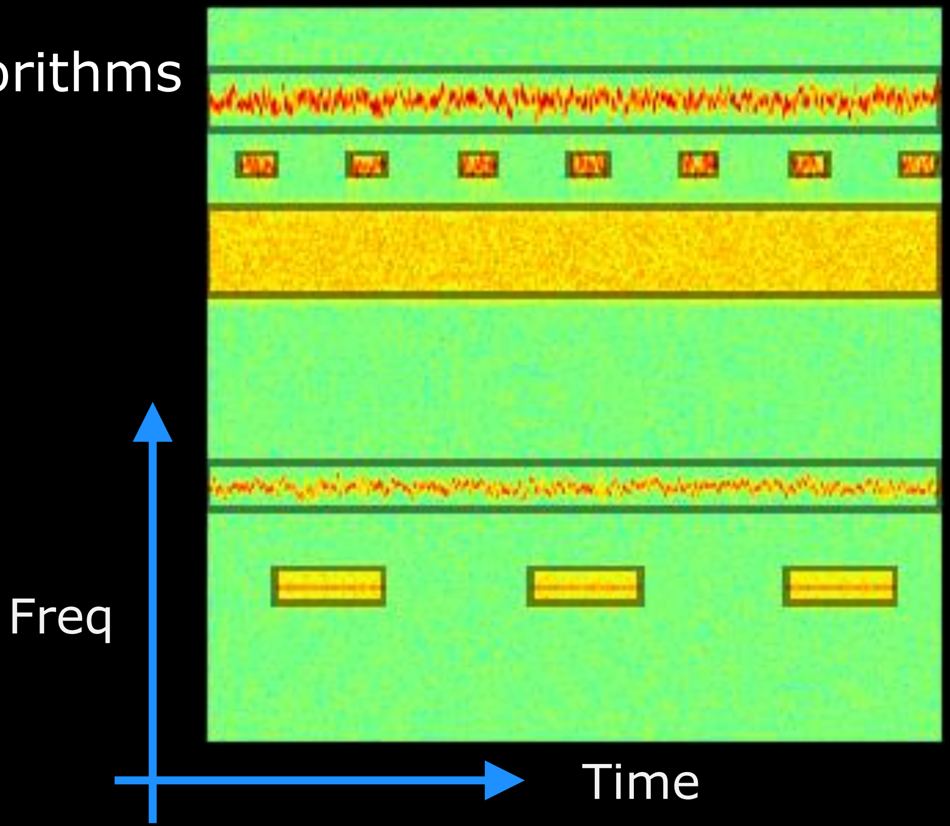
BPSK, 8-PAM, 32-QAM, 4-FSK, 4-GMSK

Recent Updates

- Three releases in 2024: 0.5.1, 0.5.2, 0.5.2.1, with more planned
- New features, improved DSP algorithms, faster speed, reduced memory, fixed bugs
  - Image-only spectrogram-based dataset tools: create, transform, extract
  - 10x speed improvement when using more than 32 workers
  - Reduced file size stored to disk by using different storage datatype
  - Improved randomization by fixing bug that caused some identical signals to be generated
  - Tighter bounding boxes for FSK and MSK
  - Reduced the sidelobes in resampling filters from -60 dB to -90 dB
  - Improved anti-aliasing filtering, minimize energy wrapping around the –fs/2 and +fs/2 boundary for multiple cases and transforms

Spectrograms

- Spectrograms display time-varying frequency content
- Also referred as “waterfalls”
- Easy for signals to be identified visually
- Foundational tool for training TorchSig ML algorithms
- ML to locate in both time and frequency
- “Bounding boxes”

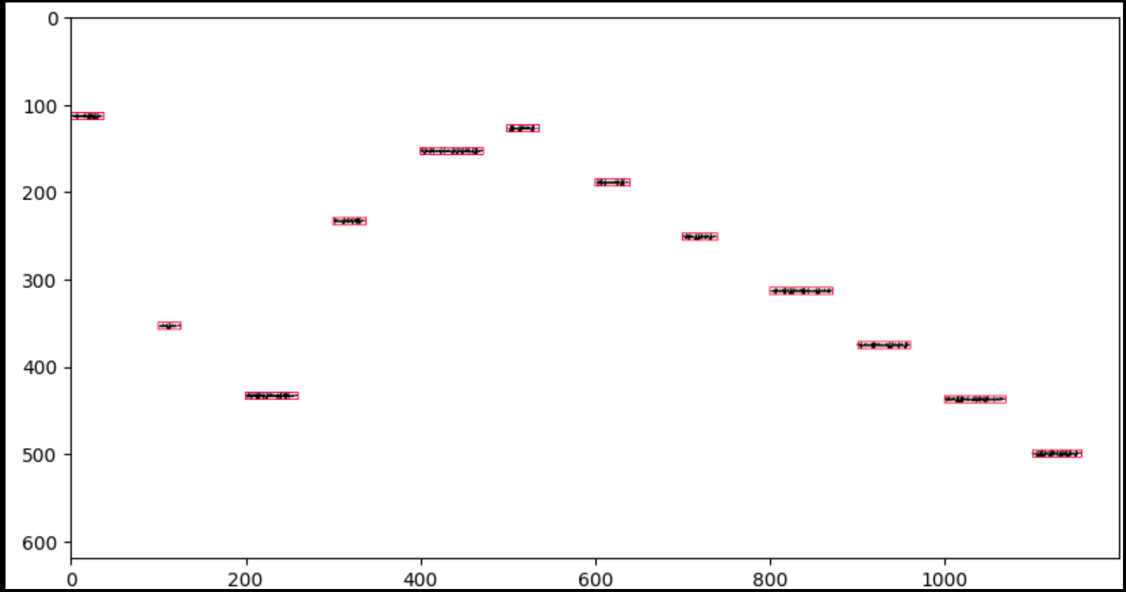


Synthetic Spectrograms

- Simulating spectrograms require large memory and computational effort
  - Creation and modulation of underlying signal(s)
  - Noise and channel impairments
  - Computation of spectrogram itself
- Can result in large databases (Many GBs, sometimes TBs)
- Slow to compute, therefore training must be done offline and in non-real time
- Synthetic Spectrograms are created directly, avoiding memory and computational burden
- Training can be done in near-real time, one spectrogram at a time
- Avoids large database problem

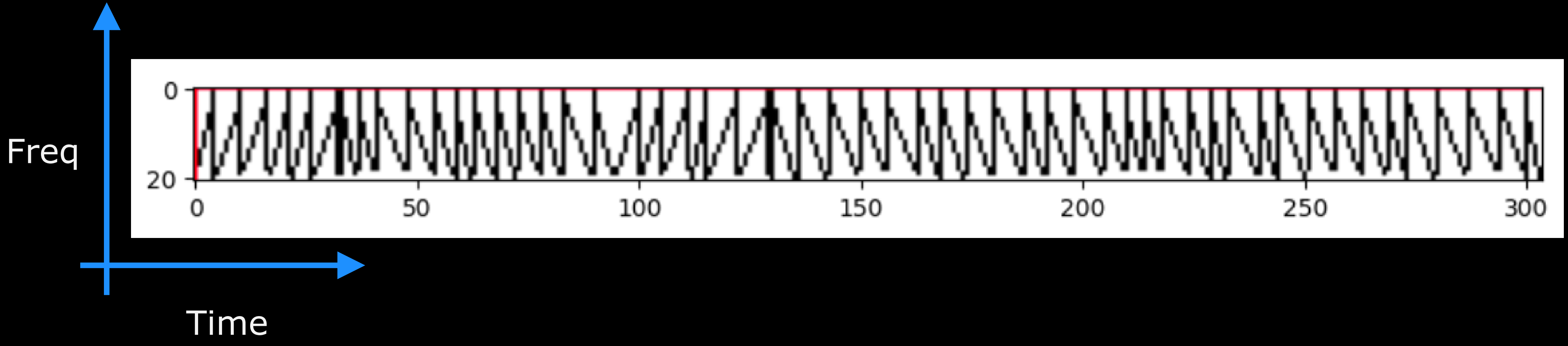
Recycling and Reusing Spectrograms

- When training ML models, more data is better
- But what if limited data is available?
- Few-shot ML techniques can be used to expand dataset
- Impairments and transforms on original data augment the dataset
- Example: "recycling" narrowband spectrograms and building a frequency hopper dataset



Direct Creation of Synthetic Spectrograms

- Some modulations allow for direct creation of synthetic spectrograms
- Chirp-based waveforms (LoRa-like) can be simulated this way
- A rising chirp and a falling chirp are defined and assembled



Direct Creation of Synthetic Spectrograms (cont.)

- Synthetic spectrograms assembled using Context Free Grammar (CFG)
- Establishes rules for how spectrograms are to be created and combined
- Example:
  - Multiple chirps = a data symbol
  - Header + multiple data symbols = data packet

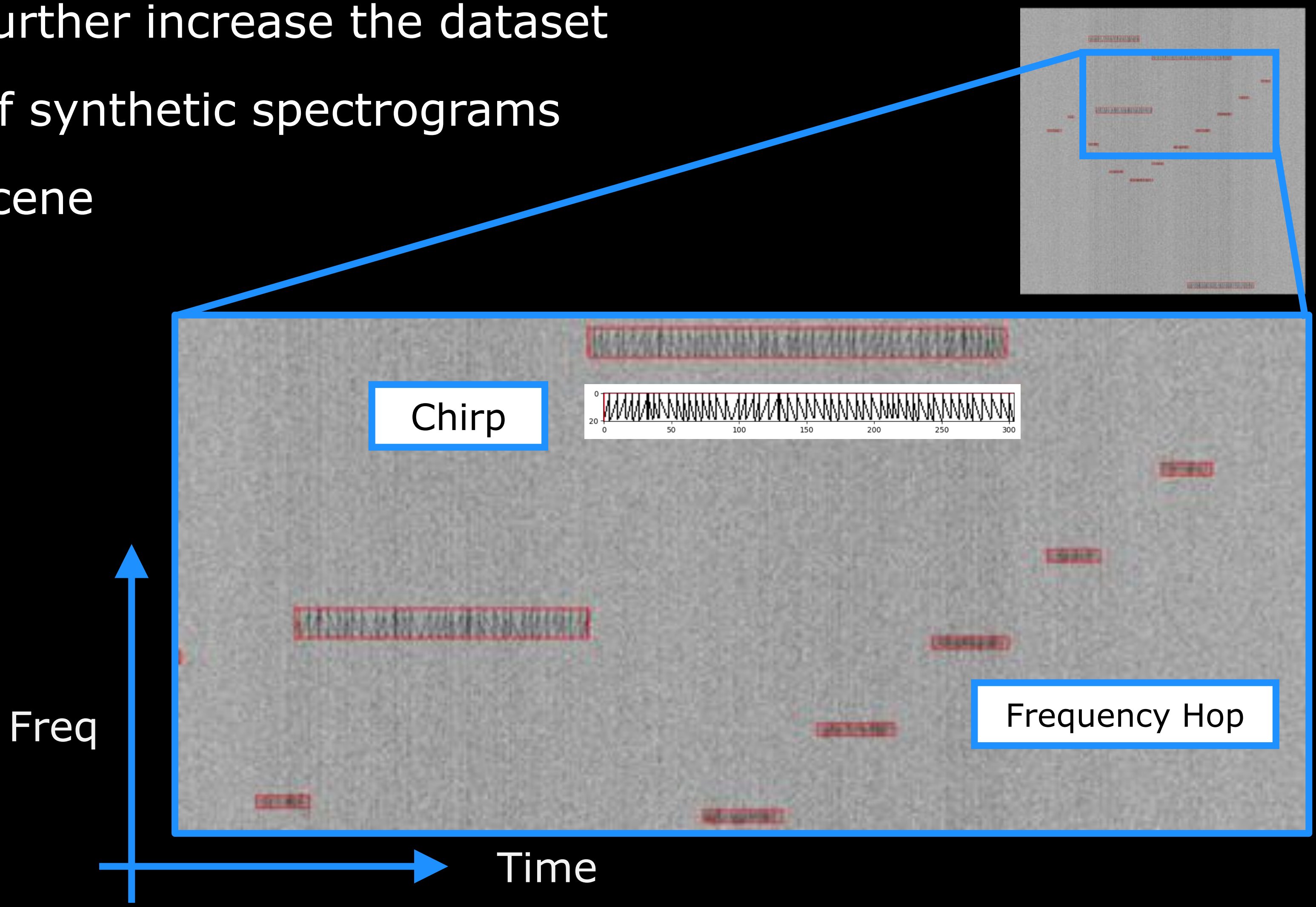
```
rising_chirp = GeneratorFunctionDataset(
    chirp_generator_function(1, 20, 4, random_height_scale = [0.9,1.1], random_width_scale = [1,2]),
    transforms=add_falling_edge)
falling_chirp = GeneratorFunctionDataset(
    chirp_generator_function(1, 20, 4, random_height_scale = [0.9,1.1], random_width_scale = [1,2]),
    transforms=[add_falling_edge, lambda x: x.flip(-1)])

chirp_stream_ds = CFGSignalProtocolDataset('cfg_signal')
chirp_stream_ds.add_rule('cfg_signal', ['rising_or_falling_stream'] + ['rising_falling_or_null']*12)
chirp_stream_ds.add_rule('rising_falling_or_null', 'rising_or_falling_stream', 1)
chirp_stream_ds.add_rule('rising_falling_or_null', 'null', 1)
chirp_stream_ds.add_rule('null', None)
chirp_stream_ds.add_rule('rising_or_falling_stream', 'rising_stream')
chirp_stream_ds.add_rule('rising_or_falling_stream', 'falling_stream')
chirp_stream_ds.add_rule('rising_stream', ['rising_segment'] + ['rising_segment_or_null']*2)
chirp_stream_ds.add_rule('rising_segment_or_null', 'rising_segment')
chirp_stream_ds.add_rule('rising_segment_or_null', 'null')
chirp_stream_ds.add_rule('rising_segment', ['rising_chirp']*3)
chirp_stream_ds.add_rule('rising_chirp', rising_chirp)
chirp_stream_ds.add_rule('falling_stream', ['falling_segment'] + ['falling_segment_or_null']*2)
chirp_stream_ds.add_rule('falling_segment_or_null', 'falling_segment')
chirp_stream_ds.add_rule('falling_segment_or_null', 'null')
chirp_stream_ds.add_rule('falling_segment', ['falling_chirp']*3)
chirp_stream_ds.add_rule('falling_chirp', falling_chirp)

yolo_chirp_stream_ds = YOLODatasetAdapter(chirp_stream_ds, class_id=0)
```

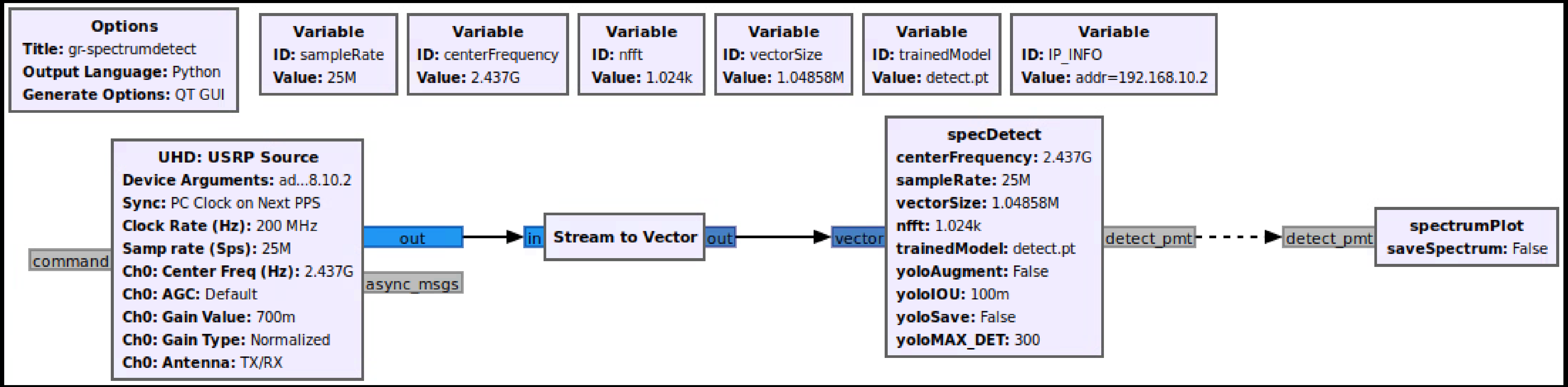
Composite Spectrograms

- Composite spectrograms further increase the dataset
- Combines multiple types of synthetic spectrograms
- Creates a more complex scene



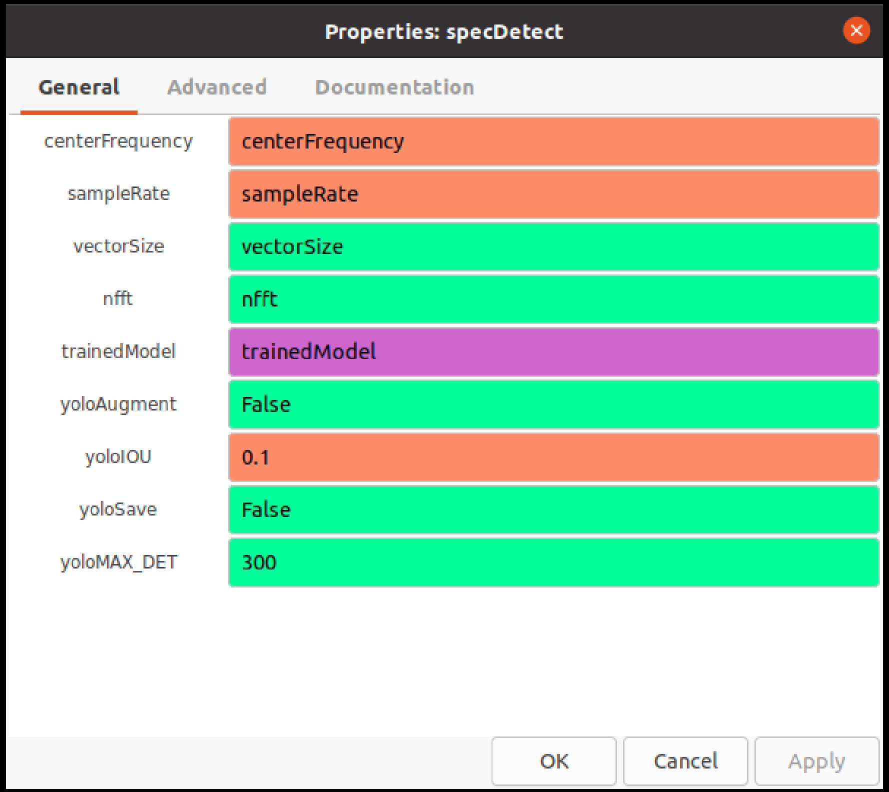
GNU Radio Block for Energy Detection

- OOT block gr-spectrumdetect performs energy detection with ML model
- Pretrained detect.pt YOLOv8x model against ISM band
- Uses a 1024x1024 spectrogram images
- specDetect block detects energy, spectrumPlot labels & displays spectrogram
- example.grc flowgraph:



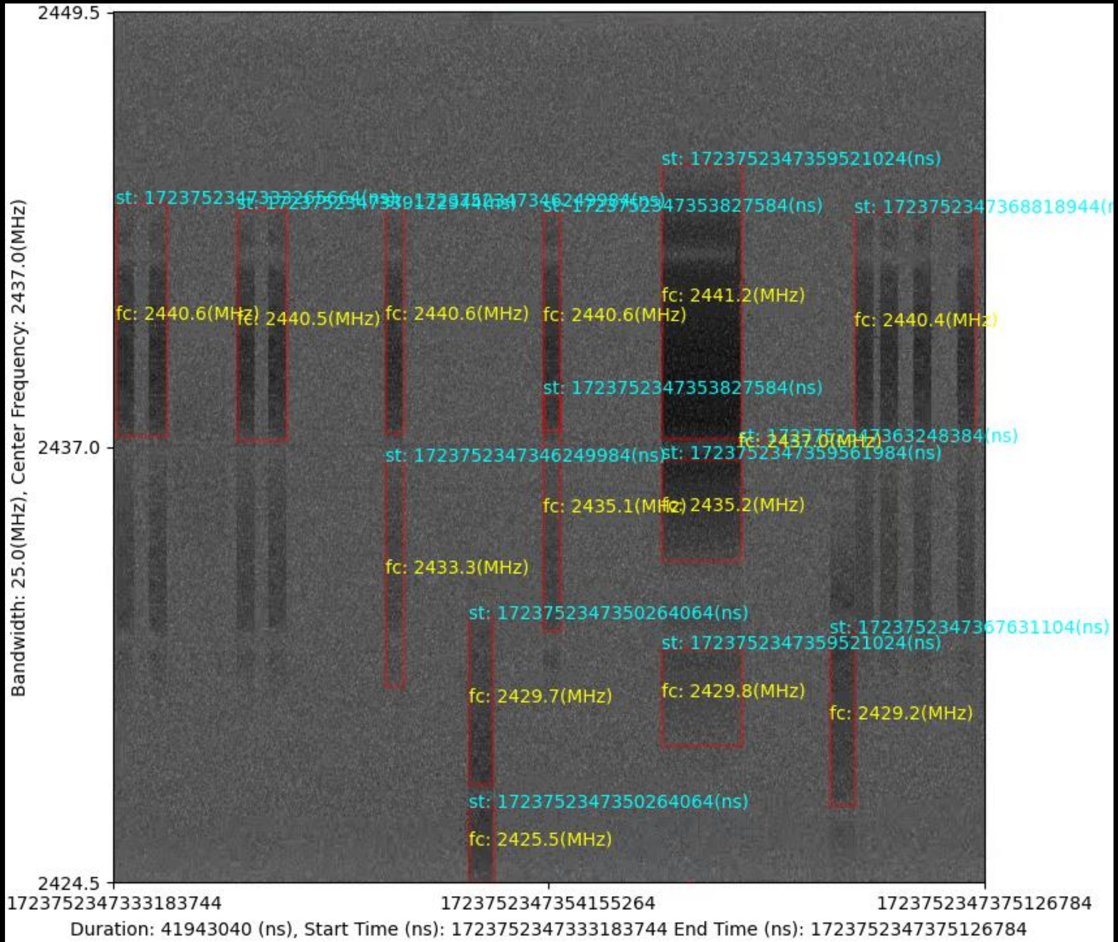
GNU Radio Block for Energy Detection (cont.)

- specDetect parameters must match those the model is trained for
- It's parametrizable, but not every feature is reconfigurable



GNU Radio Block for Energy Detection (cont.)

- spectrumPlot displays bounding boxes for detected energy



Future Work

- New tools and methods for integrating and labeling custom data sets, use of Label Studio
- Added support for HuggingFace and PyTorch, enable broader use and accessibility of TorchSig models
- Expansion of synthetic spectrogram generation feature, direct texturing and impairments to spectrograms
- Rework and expansion of analog modulations (AM, FM) and their variants (SSB, DSB, etc.)

Questions?

TorchSig