

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

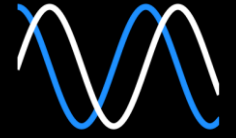
Matt Carrick, Peraton Labs



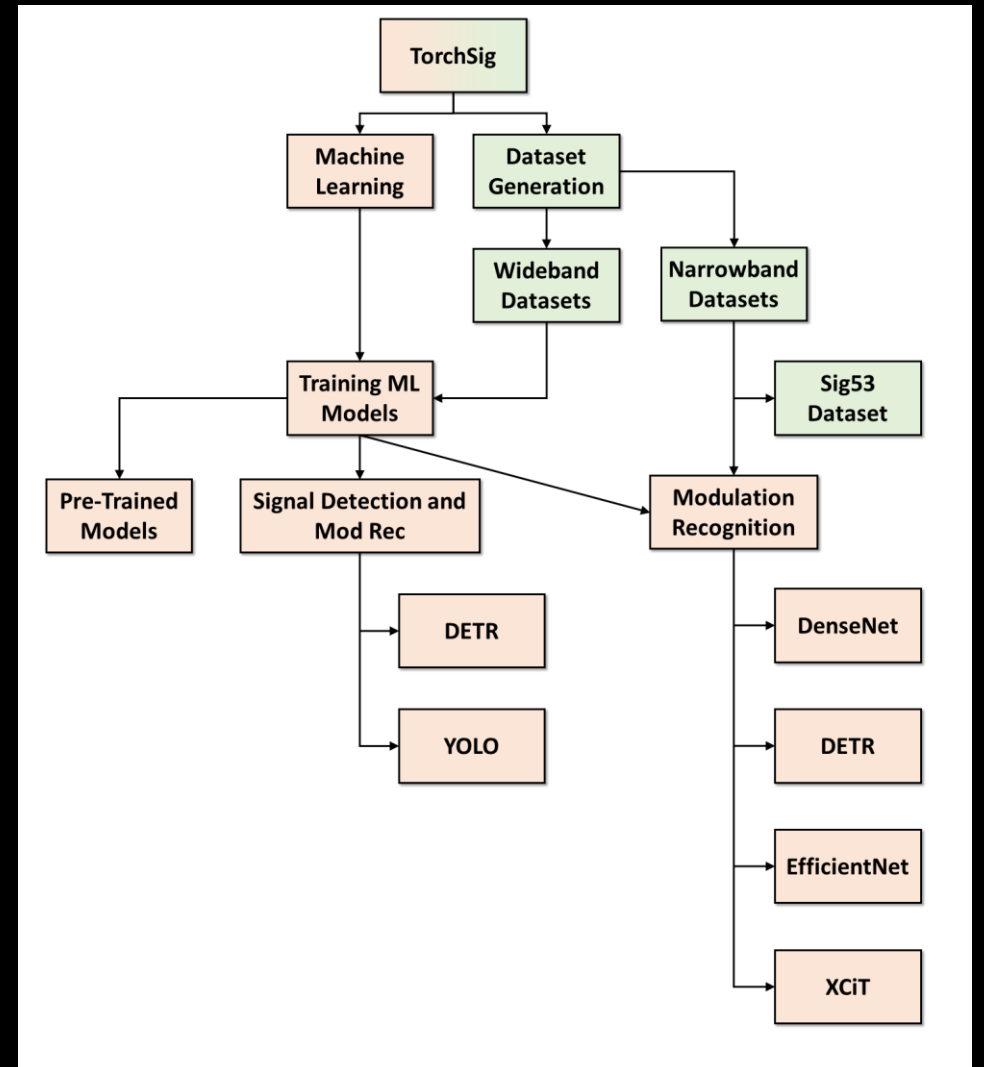
GRCOn 2024

September 20, 2024

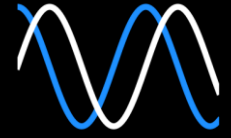
What is TorchSig?



- 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



What is TorchSig? (cont.)



The screenshot shows the TorchSig homepage. At the top, there is a navigation bar with links for Home, About, Downloads, and Documentation. The main header features the TorchSig logo and the tagline "A PyTorch Signal Processing Machine Learning Toolkit". Below this, there are buttons for "Get Started" and "GitHub", and a GitHub star count of 153. The "Key Features" section is divided into three columns: "Signals Datasets", "Domain Transforms", and "Pretrained Models".

Key Features

- Signals Datasets**: TorchSig can generate and easily interface with the Sig53 dataset, an augmentable dataset representing 53 digital modulations commonly used in radio frequency communication signals, useful for signal classification research. TorchSig can also generate and interface with the WidebandSig53 dataset, a wideband extension to Sig53, containing multiple signals in each data example, useful for signal detection and recognition research. TorchSig can be used as a standard in the open source community working with signals datasets.
- Domain Transforms**: Numerous complex signal augmentations, impairments, and expert feature transforms are provided and can be seamlessly customized and further developed.

TorchSig.com

The screenshot shows the "Pretrained Models" page on TorchSig.com. It features a navigation bar at the top with links for Home, About, Downloads, and Documentation. A red arrow points to the "Downloads" link. The main content area is titled "Pretrained Models" and includes a sub-section for "Sig53 Pre-Trained Models" with a list of model names: EfficientNet-B0, EfficientNet-B2, EfficientNet-B4, XCiT-Nano, and XCiT-Tiny12. Below this is a sub-section for "WidebandSig53 Pre-Trained Models for Signal Detection" with a list of model names: YOLOv5-Pico, YOLOv5-Nano, YOLOv5-Small, DETR-B0-Nano, DETR-B2-Nano, DETR-B4-Nano, PSPNet-B0, PSPNet-B2, PSPNet-B4, Mask2Former-B0, and Mask2Former-B0.

Pretrained Models

TorchSig's model API can automatically reach out and download our models pretrained on the Sig53 datasets, and they are additionally available for download at the below locations:

Sig53 Pre-Trained Models:

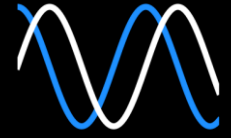
- [EfficientNet-B0](#)
- [EfficientNet-B2](#)
- [EfficientNet-B4](#)
- [XCiT-Nano](#)
- [XCiT-Tiny12](#)

WidebandSig53 Pre-Trained Models for Signal Detection:

- [YOLOv5-Pico](#)
- [YOLOv5-Nano](#)
- [YOLOv5-Small](#)
- [DETR-B0-Nano](#)
- [DETR-B2-Nano](#)
- [DETR-B4-Nano](#)
- [PSPNet-B0](#)
- [PSPNet-B2](#)
- [PSPNet-B4](#)
- [Mask2Former-B0](#)
- [Mask2Former-B0](#)

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

What is TorchSig? (cont.)

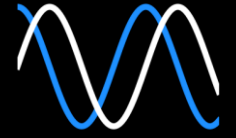


- Code available github.com/torchdsp/torchsig

The screenshot shows the GitHub repository page for TorchDSP/torchsig. The repository is public and has 153 stars, 36 forks, and 23 issues. The main branch is 'main' with 5 branches and 10 tags. The repository contains a file tree with folders like .github, docs, examples, scripts, tests, and torchsig, and files like .dockerignore, .gitignore, CONTRIBUTING.md, DISCLAIMER.md, Dockerfile, LICENSE.md, and README.md. The 'About' section describes TorchSig as an open-source signal processing machine learning toolkit based on the PyTorch data handling pipeline. The 'Releases' section shows the latest release is v0.5.2.1, published on Jul 13. The 'Packages' section shows no packages published. The 'Contributors' section shows 8 contributors.

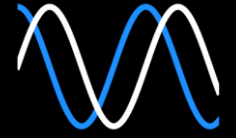
File/Folder	Commit Message	Last Commit
.github	v0.4.0 (#137)	last year
docs	ready for v0.5.1 release	2 months ago
examples	ready for v0.5.1 release	2 months ago
scripts	removed unused print statements add collate_fn in generate...	last month
tests	ready for v0.5.1 release	2 months ago
torchsig	removed unused print statements add collate_fn in generate...	last month
.dockerignore	V0.4.1 (#162)	last year
.gitignore	adding all .ipynb_checkpoints to .gitignore	2 months ago
CONTRIBUTING.md	ready for v0.5.1 release	2 months ago
DISCLAIMER.md	ready for v0.5.1 release	2 months ago
Dockerfile	fixing the .numpy() call in utils/writer.py	last month
LICENSE.md	ready for v0.5.1 release	2 months ago
README.md	ready for v0.5.1 release	2 months ago

Machine Learning Models



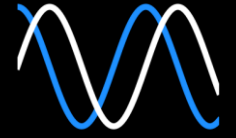
- TorchSig provides two types of models:
- “Narrowband” or modulation recognition model
 - Operates on IQ samples
 - Assumes signal is already channelized, basebanded and roughly time-aligned by an energy detector
- “Wideband” model
 - Operates on spectrograms
 - Able to locate multiple signals in time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

Dataset Generation and Sig53

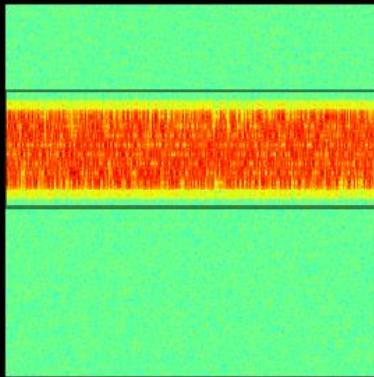


- Signal modulators:
 - Frequency Shift Keying (FSK and GFSK)
 - Minimum Shift Keying (MSK and GMSK)
 - Quadrature Amplitude Modulation (QAM)
 - Phase Shift Keying (PSK)
 - Pulse Amplitude Modulation (PAM)
 - On-Off Keying (OOK)
 - Orthogonal Frequency Division Multiplexing (OFDM)
- Narrowband datasets: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

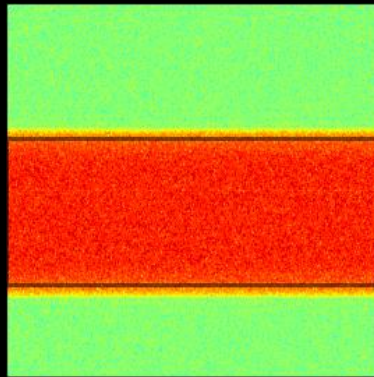
Dataset Generation and Sig53 (cont.)



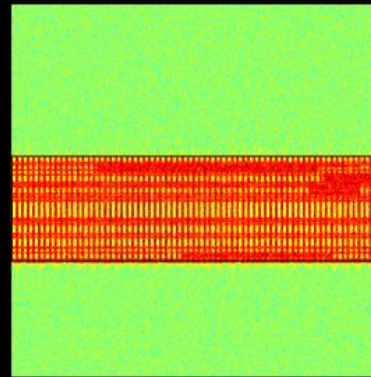
- Examples of wideband signals:



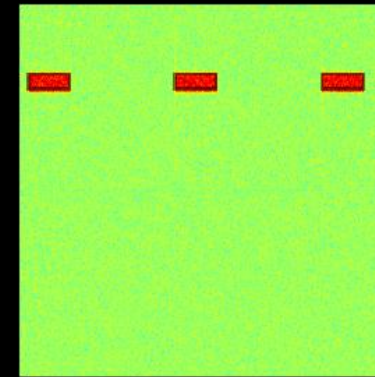
8-FSK



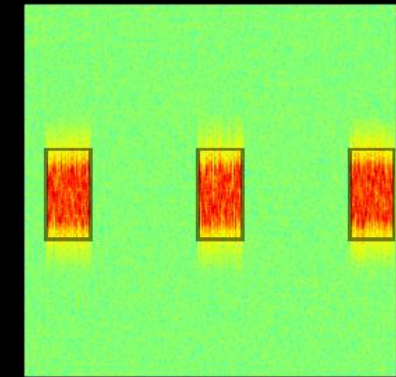
128-QAM



OFDM w/ 256
Subcarriers

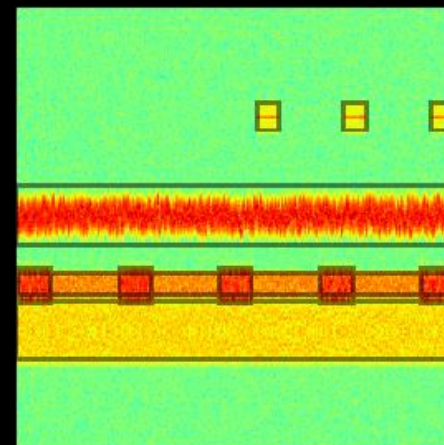
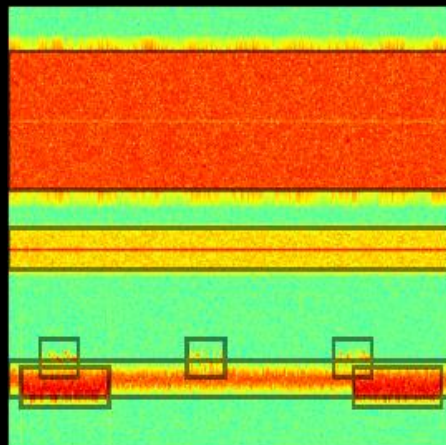


16-QAM



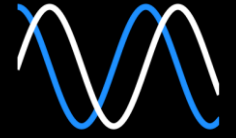
16-MSK

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



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

Recent Updates

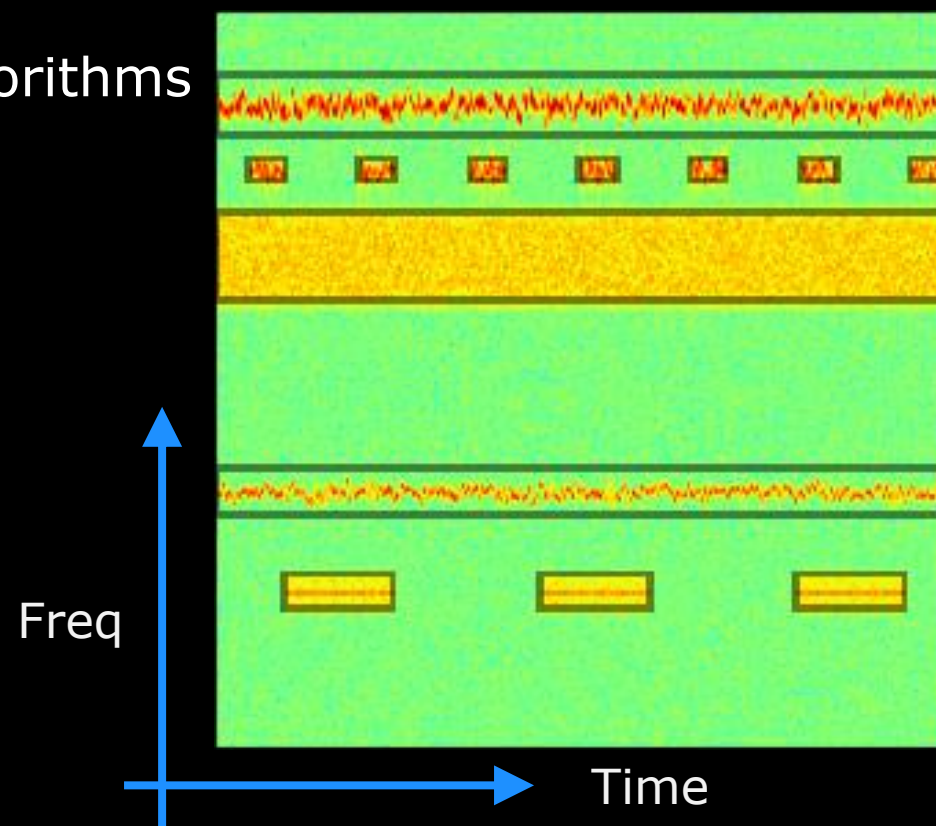


- Three releases in 2024: 0.5.1, 0.5.2, 0.5.2.1, with more planned
- New features, improved DSP algorithms, faster speed, reduced memory, fixed bugs
 - Image-only spectrogram-based dataset tools: create, transform, extract
 - 10x speed improvement when using more than 32 workers
 - Reduced file size 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

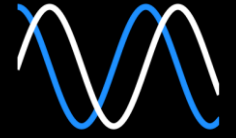
Spectrograms



- 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”

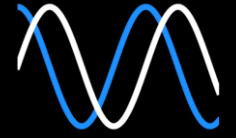


Synthetic Spectrograms

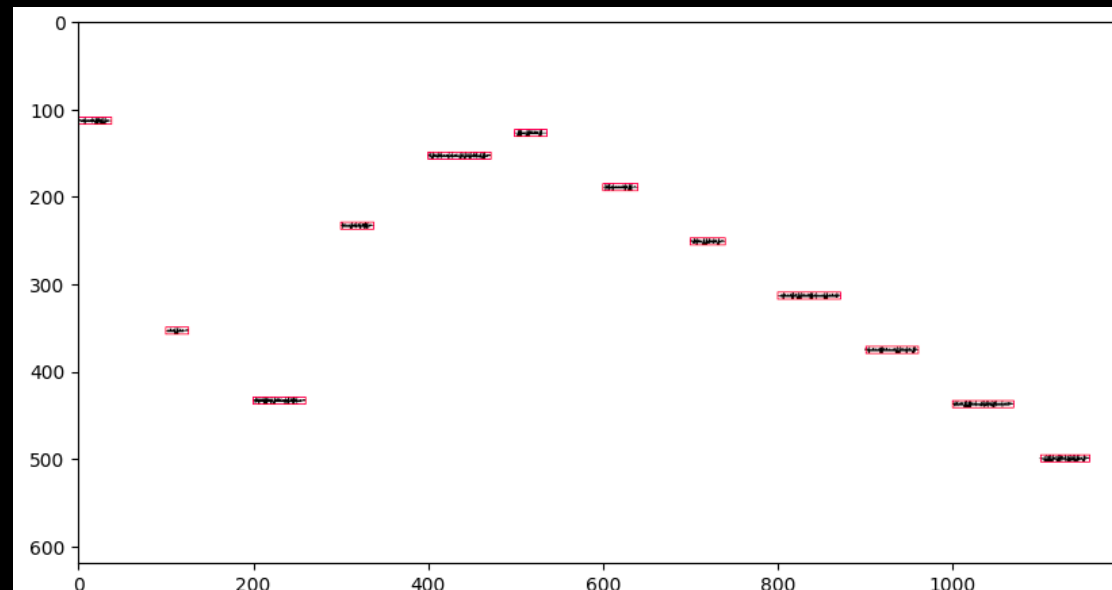


- Simulating spectrograms require large memory and computational effort
 - Creation and modulation of underlying signal(s)
 - Noise and channel impairments
 - Computation of spectrogram itself
- Can result in large databases (Many GBs, sometimes TBs)
- Slow to compute, therefore training must be done 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

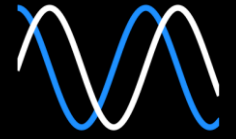
Recycling and Reusing Spectrograms



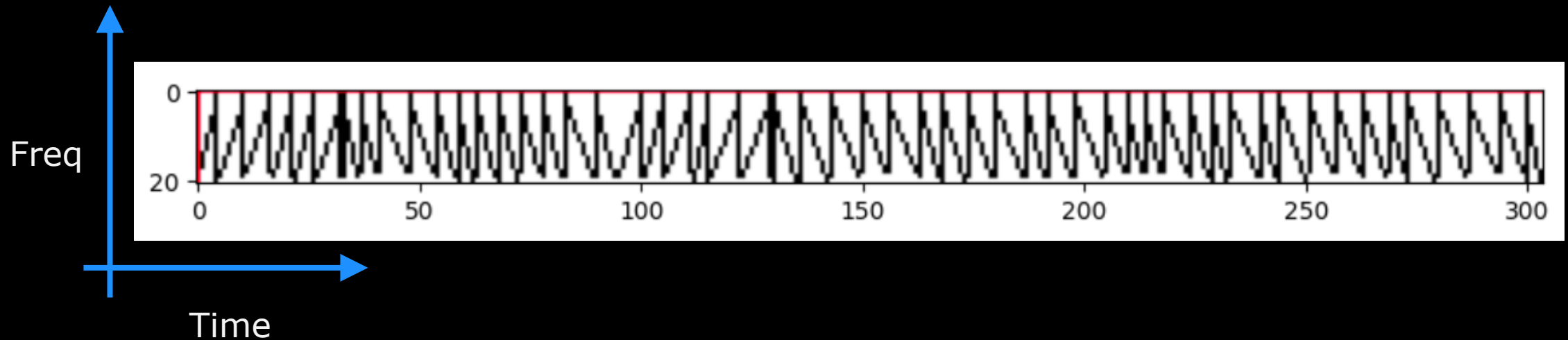
- When training ML models, more data is better
- But what if limited data is available?
- Few-shot ML techniques can be used to expand dataset
- Impairments and transforms on original data augment the dataset
- Example: “recycling” narrowband spectrograms and building a frequency hopper dataset



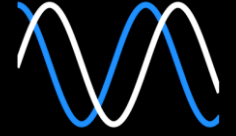
Direct Creation of Synthetic Spectrograms



- 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



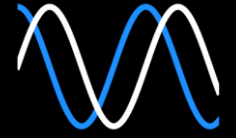
Direct Creation of Synthetic Spectrograms (cont.)



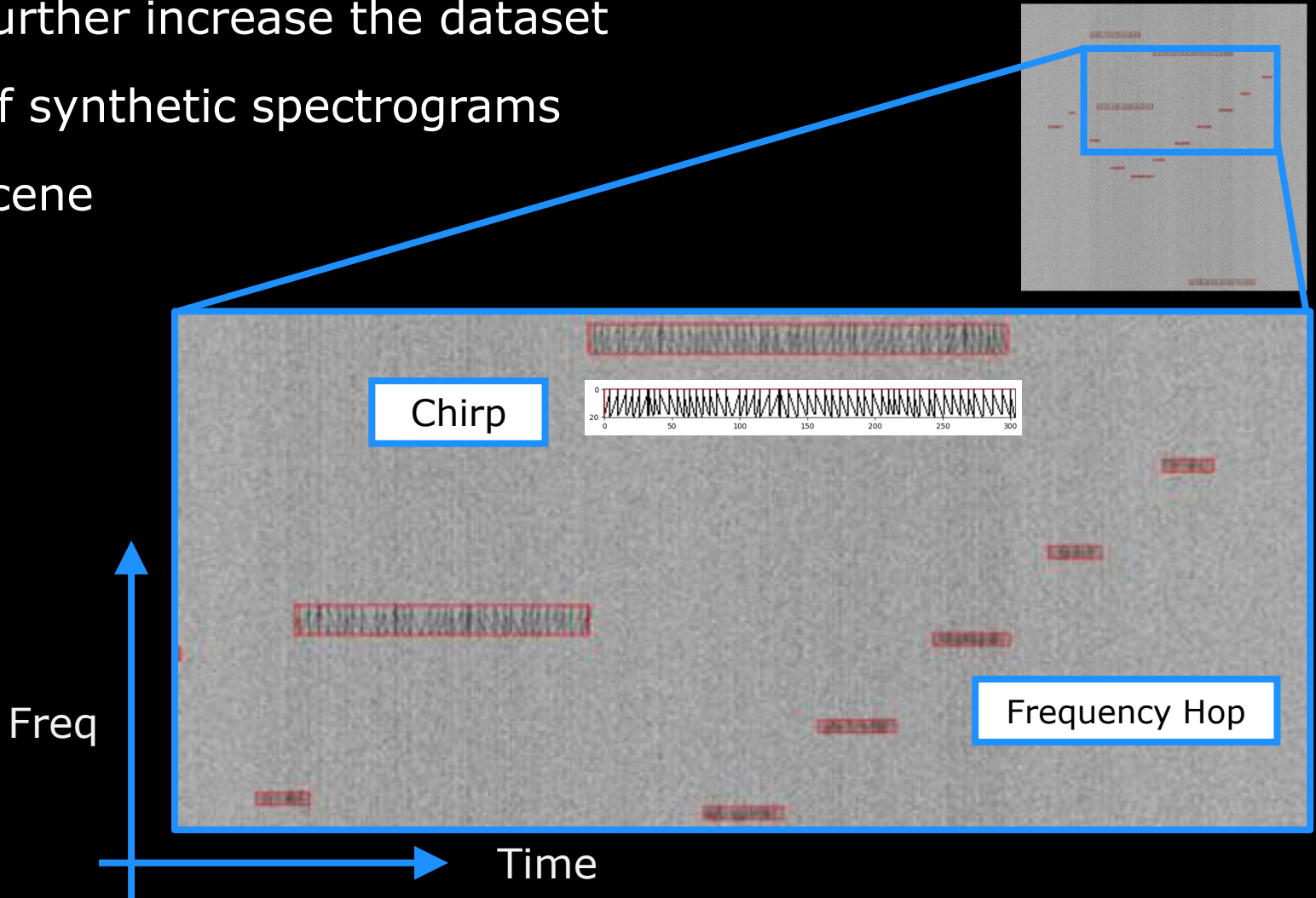
- Synthetic spectrograms assembled using Context Free Grammar (CFG)
- Establishes rules for how spectrograms are to be created and combined
- Example:
 - Multiple chirps = a data symbol
 - Header + multiple data symbols = data packet

```
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)
```

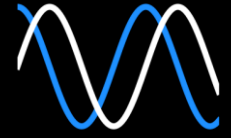
Composite Spectrograms



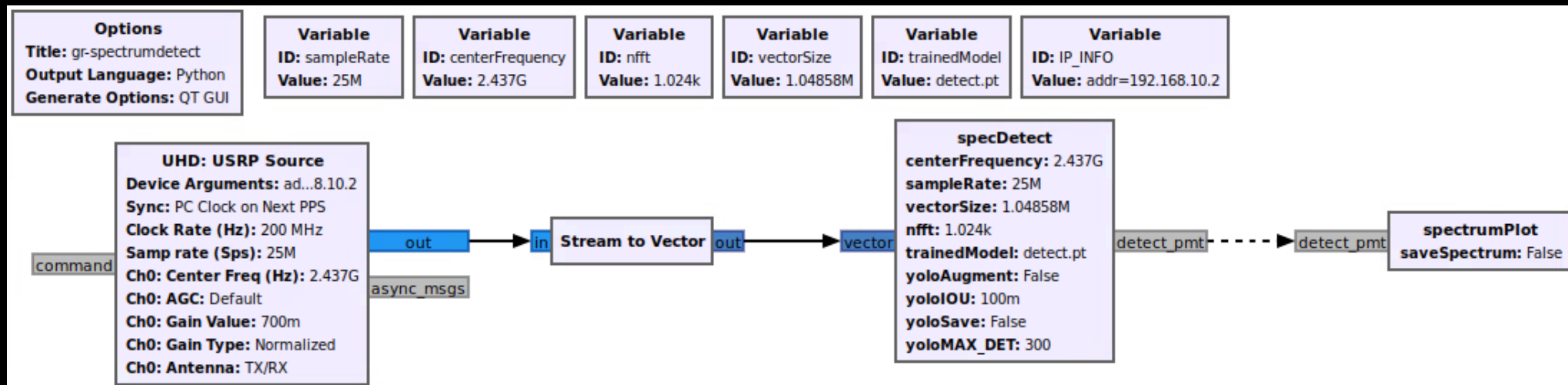
- Composite spectrograms further increase the dataset
- Combines multiple types of synthetic spectrograms
- Creates a more complex scene



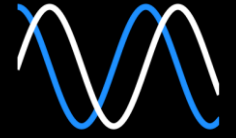
GNU Radio Block for Energy Detection



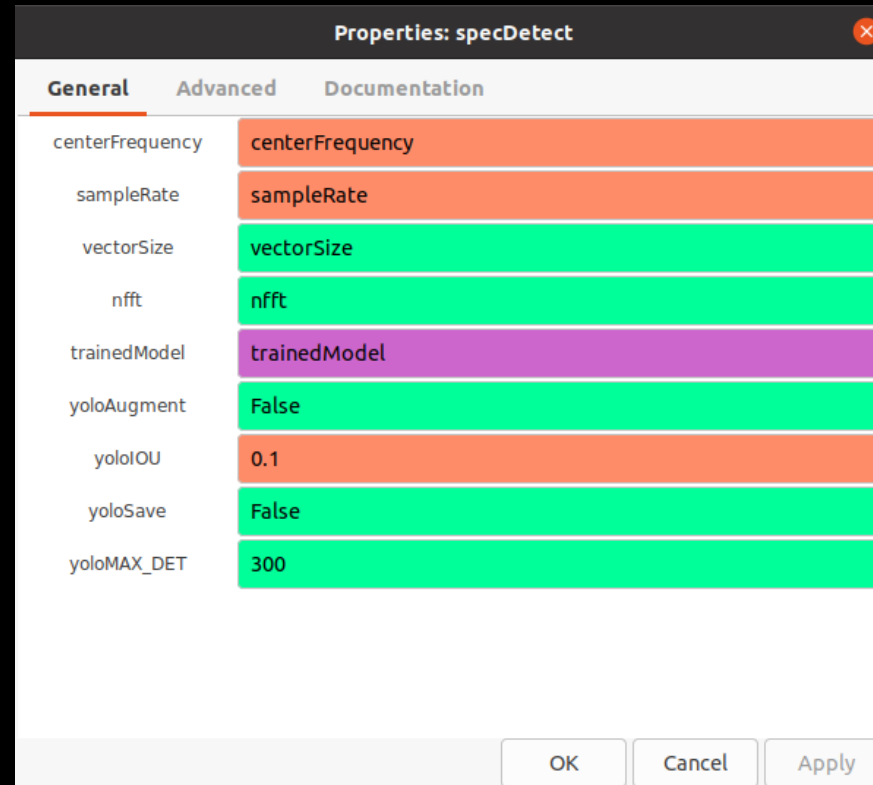
- OOT block gr-spectrumdetect performs energy detection with ML model
- Pretrained detect.pt YOLOv8x model against ISM band
- Uses a 1024x1024 spectrogram images
- specDetect block detects energy, spectrumPlot labels & displays spectrogram
- example.grc flowgraph:



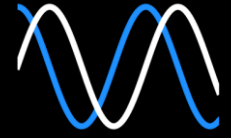
GNU Radio Block for Energy Detection (cont.)



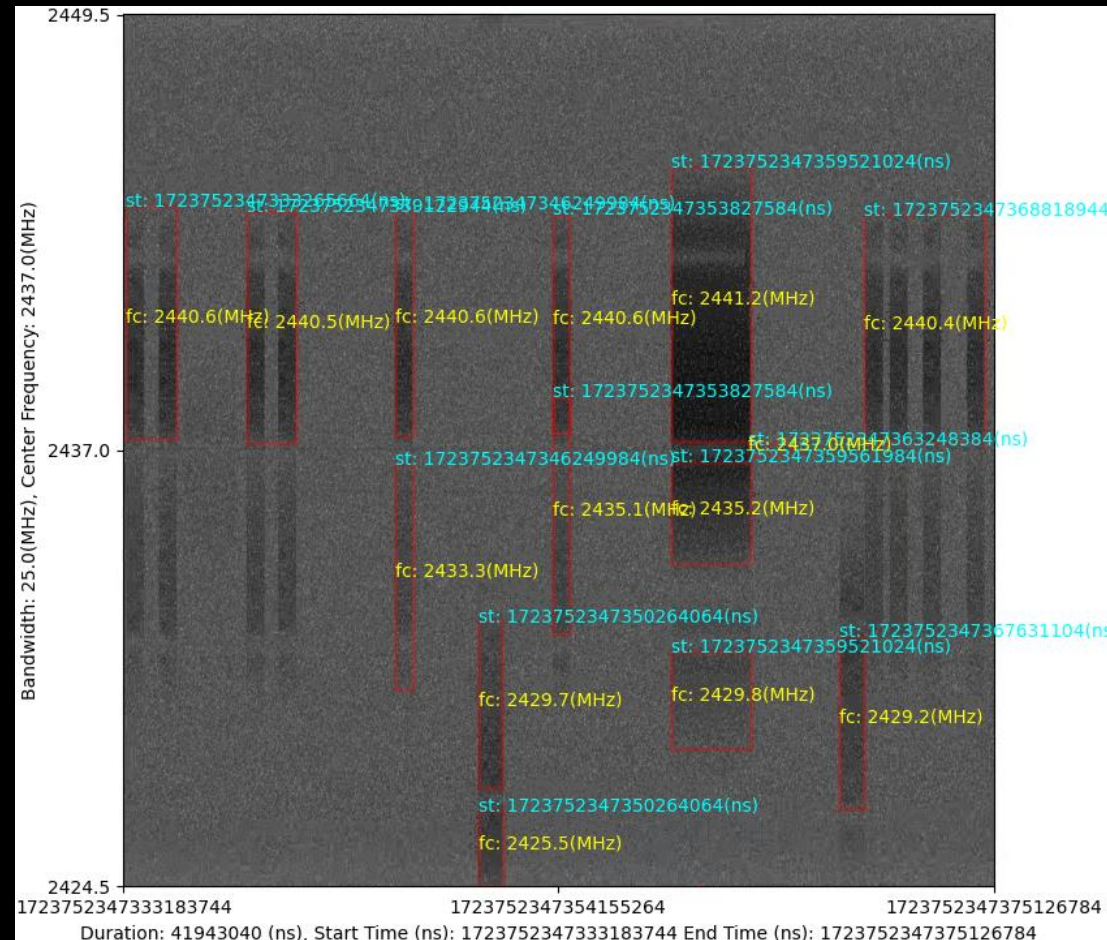
- specDetect parameters must match those the model is trained for
- It's parametrizable, but not every feature is reconfigurable



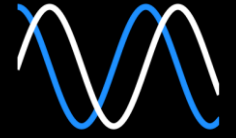
GNU Radio Block for Energy Detection (cont.)



- 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?

