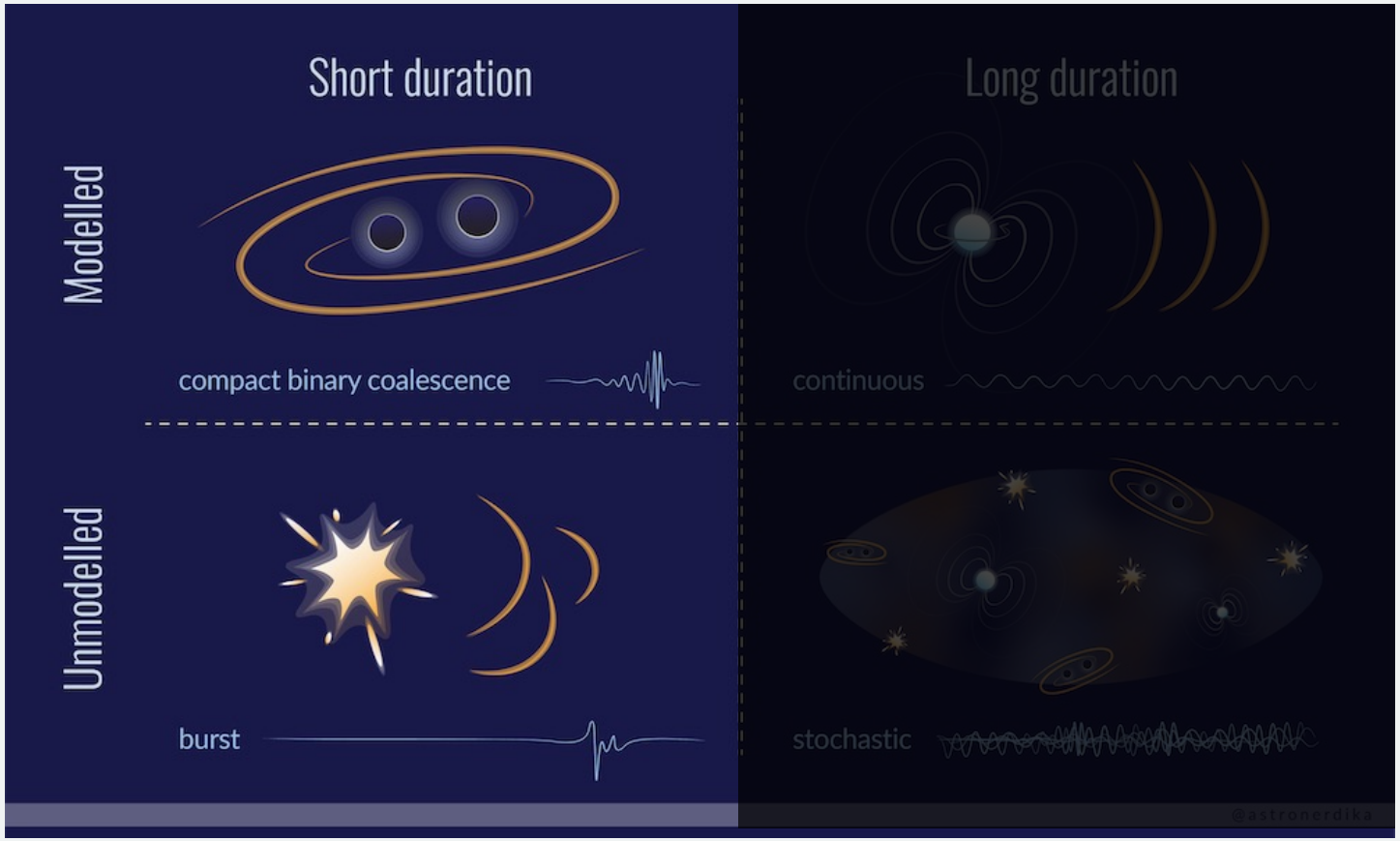
Towards the next generation of transient gravitational wave searches

Melissa Lopez

g2Net September 2022

Utrecht University

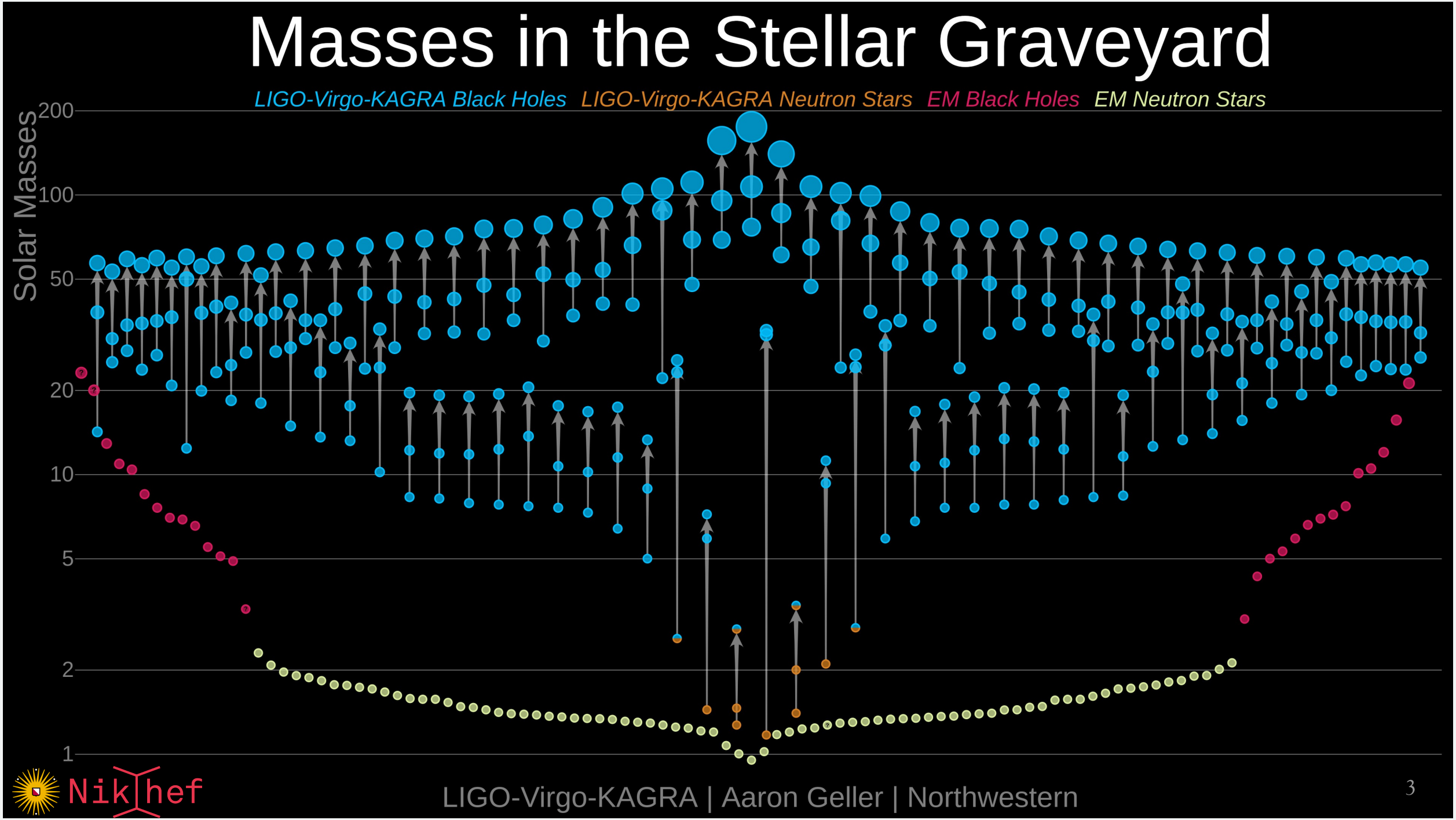
Nikhef



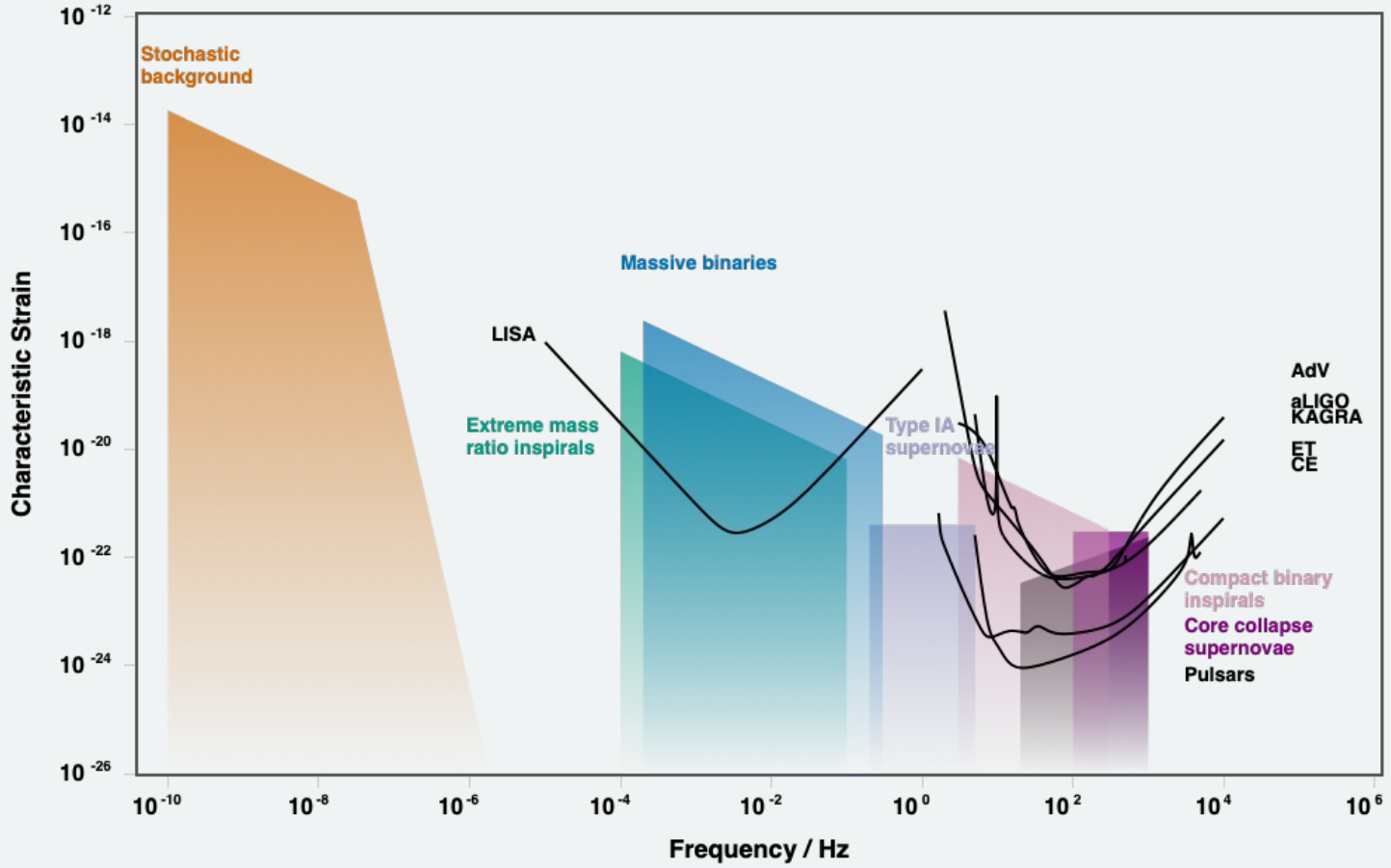
Credits: Shanika Galaudage

Gravitational wave detection and its sources

Masses in the Stellar Graveyard



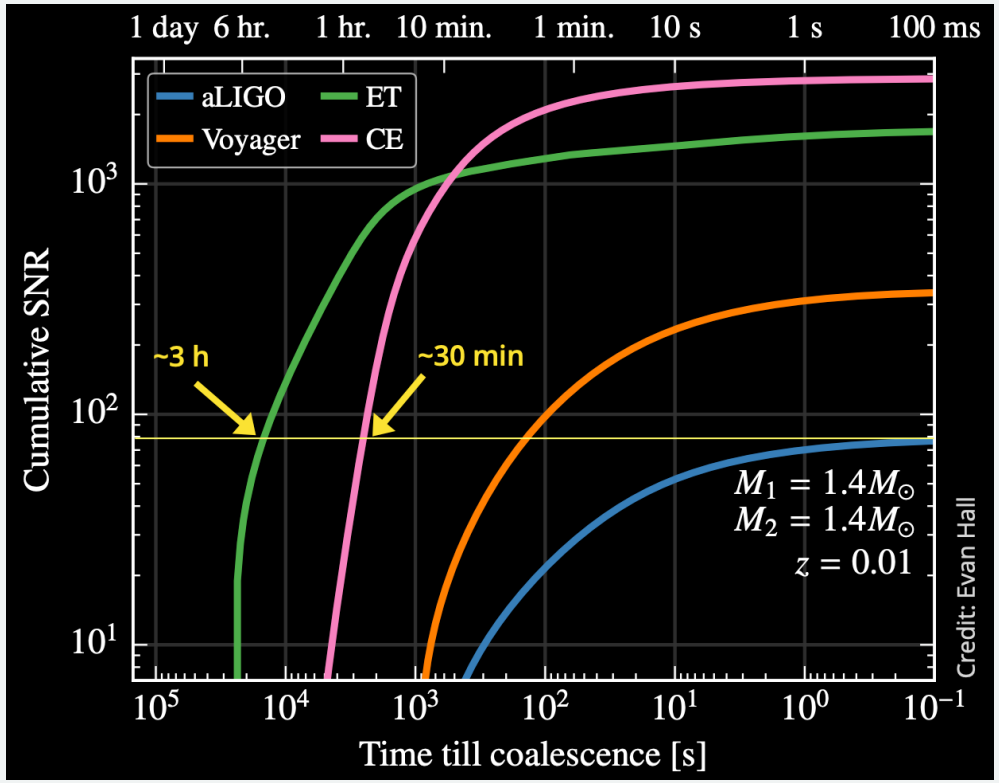
Towards future gravitational wave observatories



Source: gwplotter.com

The next generation of challenges: Einstein Telescope

Credits: Andreas Freise

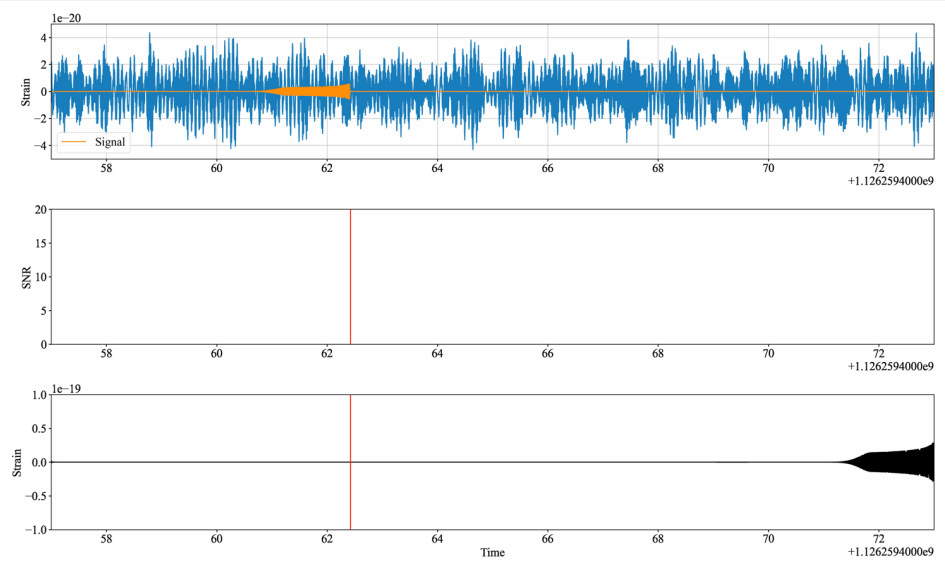


We will see *more* signals for *longer times*

Matched filtering (MF) for CBC searches

Several pipelines for short modelled searches: GstLAL, PyCBC, SPIIR, …

MF := matching models (templates) to unknown signals



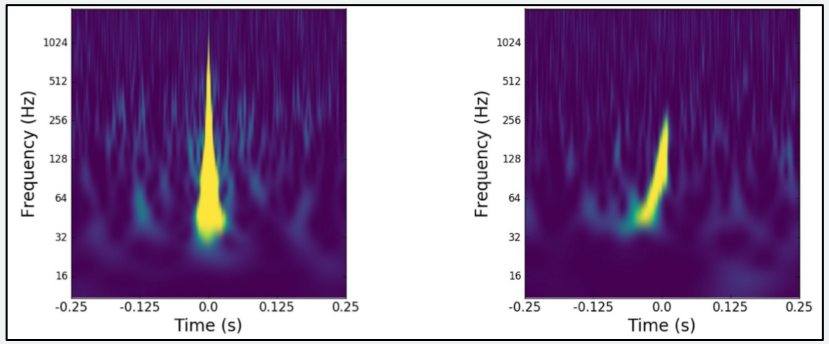
Unknown signal * Template Bank (simulated GW) → Triggers

Seeing *more* signals for *longer times*, implies *longer templates* and *larger banks*

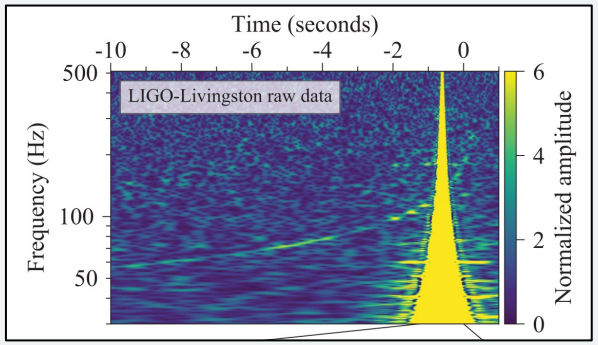
Overlapping signals!

Old and new challenges: transient noise burst (glitches)

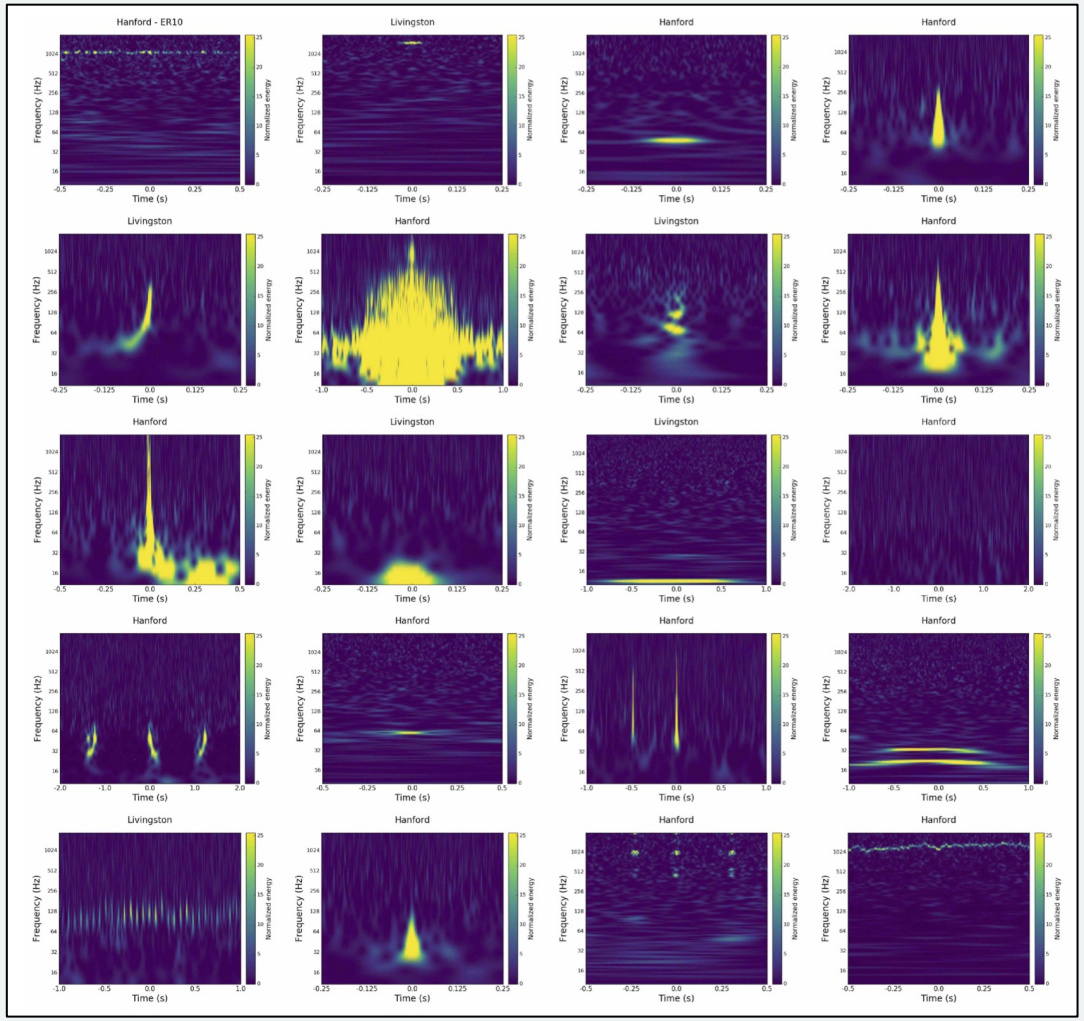
- Caused by instruments or environment (known or unknown)
- Diminish scientific data available
- Hinder transient GW searches (mask and/or mimic)
- Novel glitches in future runs



Example of a blip glitch (left)
and a intermediate-mass BBH (right)



GW170817 masked by a glitch



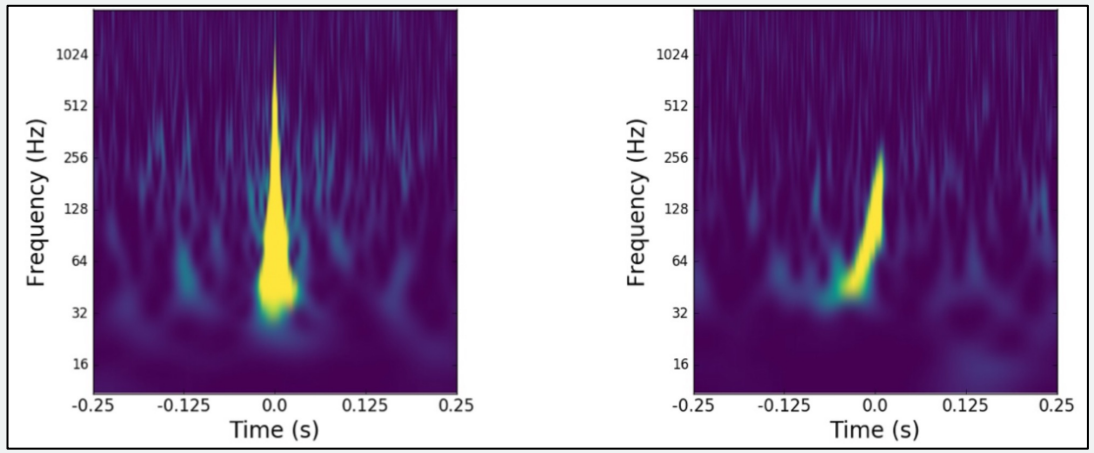
S. Bahaadini. Inf. Sci. 2018

Towards the next generation of transient gravitational wave searches

Can Machine Learning help us with the next challenges?

We focus in two problems:

1. Understand populations of glitches → better inclusion and modeling
2. Beyond current searches: extract more information



Example of a blip glitch (left) and a IMBH (right)

Blips from L1 and H1 (O2) are abundant and simple

IMBH signals are similar to other glitches

1. Understanding populations of glitches

What is the aim of this work?

Simulate transient noise burst (glitches) from LIGO detectors with Machine Learning

Why more? We have so many!

Create an open-source interface for production:

- mock data challenges: Einstein Telescope
- improve classifiers: controled dataset → robust algorithms

and more!

How are we going to do it?

Generate glitches in time domain with Generative Adversarial Networks

Data and pre-processing

Arxiv: 1611.04596

Gravity Spy (selection: ML classifier)

Arxiv: 1410.3835

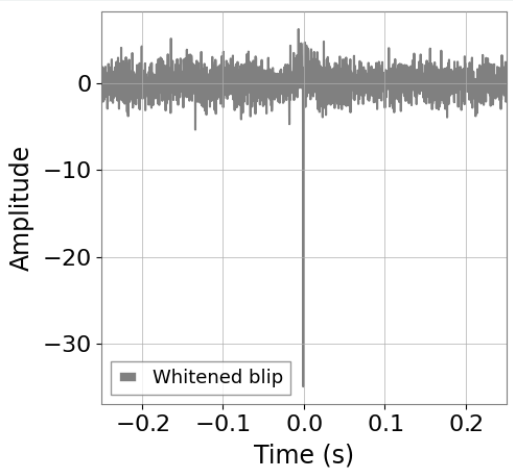
BayesWaves (reconstruction : wavelet modeling)

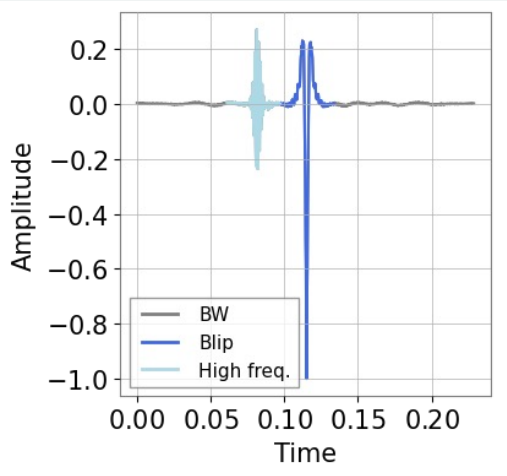
Arxiv: 1806.07329

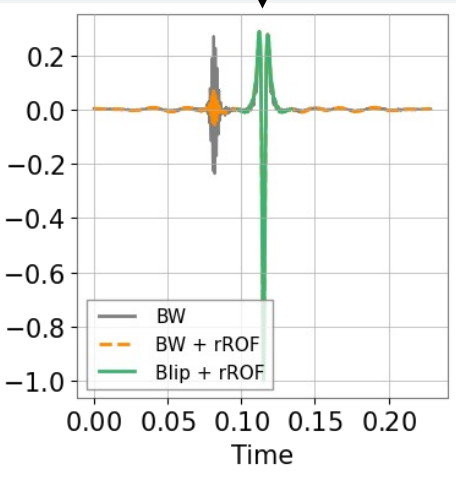
rROF (denoising: variational method)

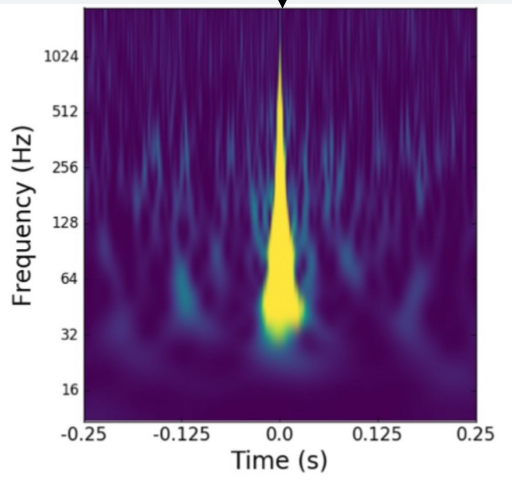
Gravity Spy (re-evaluation)

GAN input

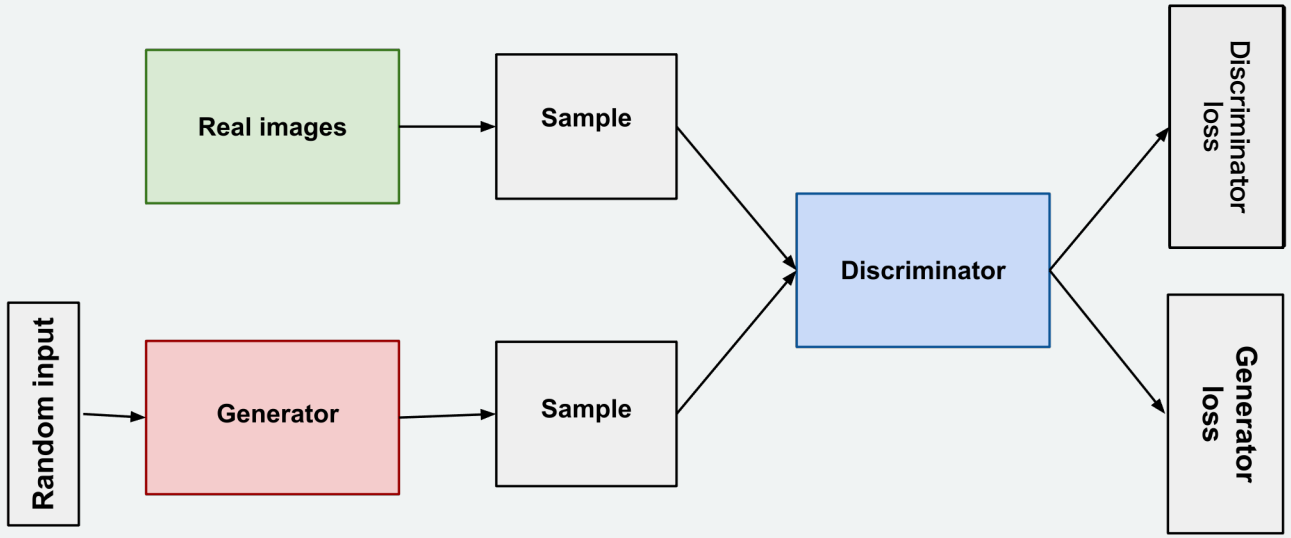








Generative Adversarial Networks



- Used to learn the underlying distribution of the data
- Convolutional neural networks (CNN)
- Inspired by Game Theory: game with 2 networks
- Use Wasserstein loss: discriminator till optimality
- Very unstable process
- Penalize the network to stabilize it

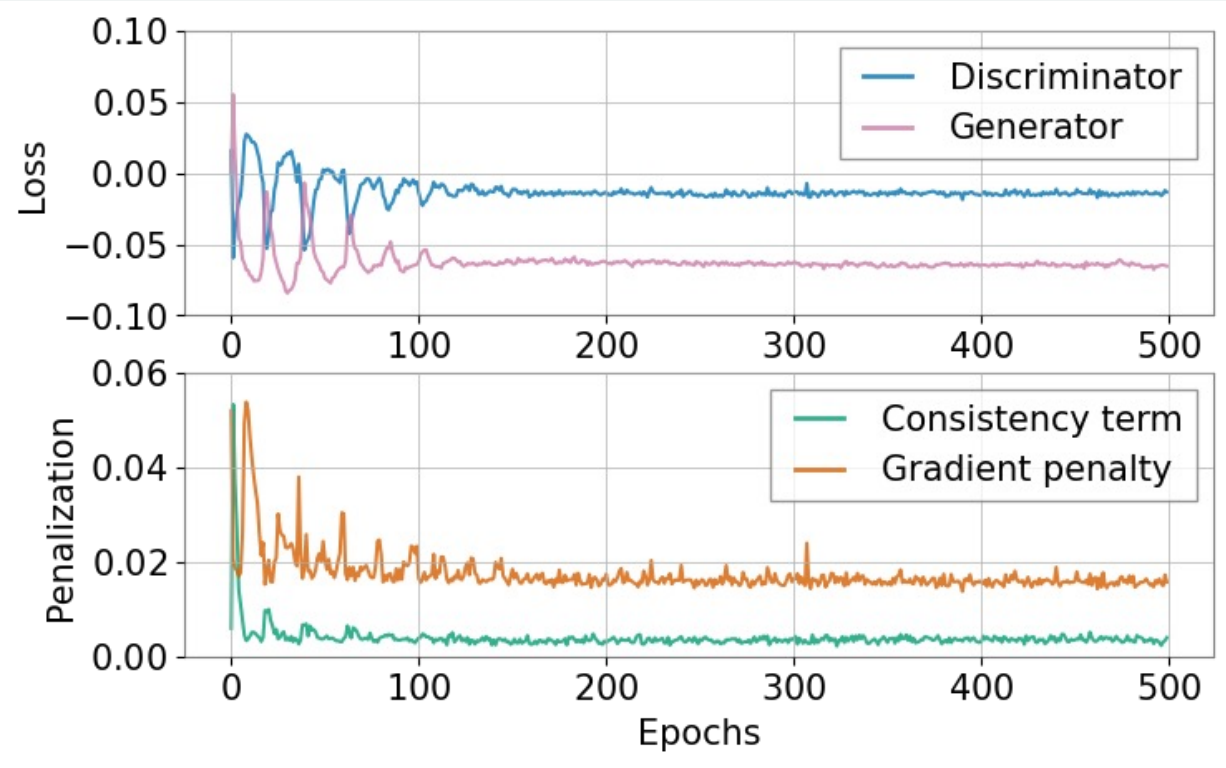
Network employed: CT-GAN (Wei, ICLR 2018)

CT-GAN: GP + CT with Dropout

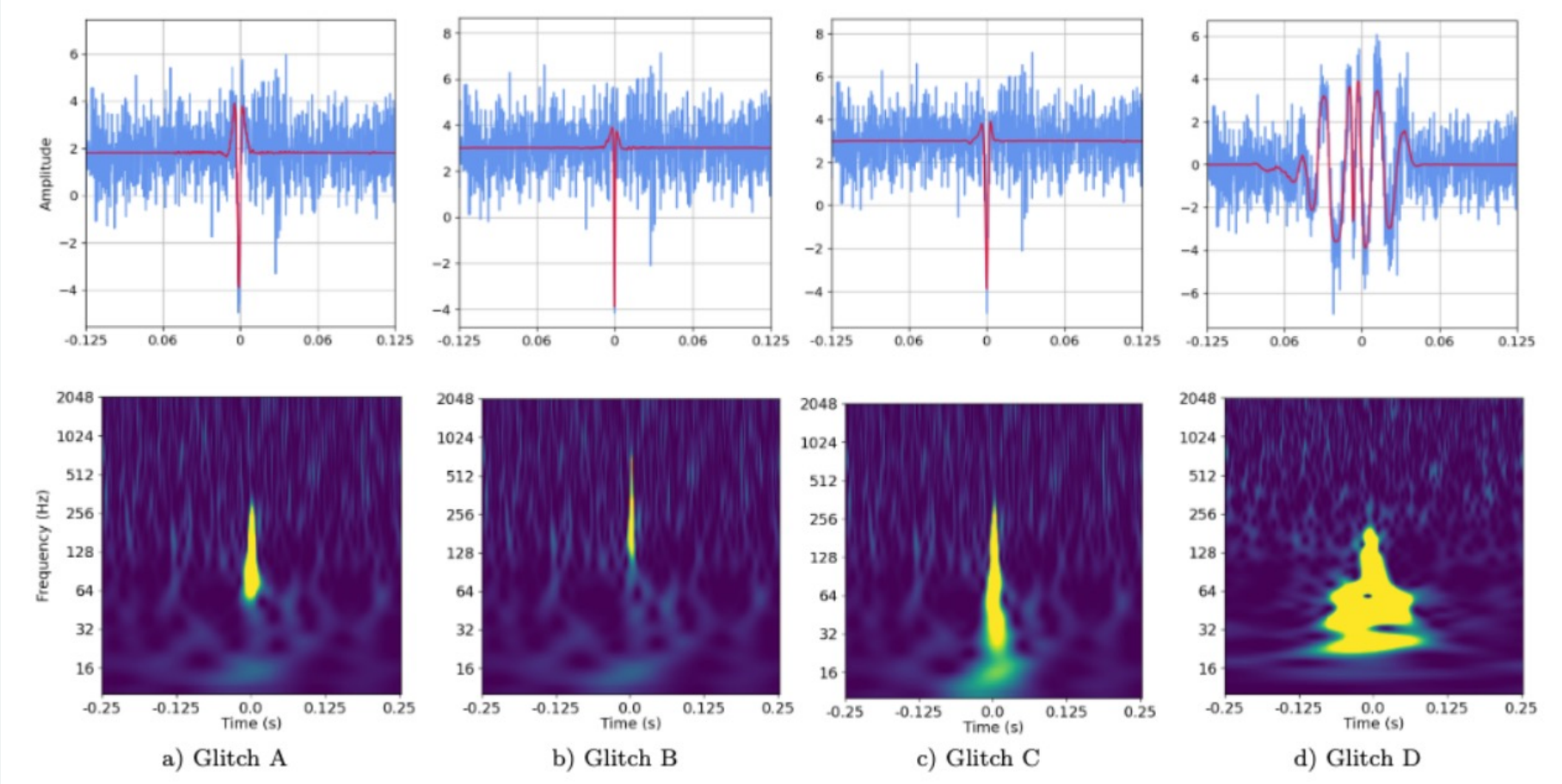
Some intuition from the experiments:

- Gradient Penalty (GP): balances the loss of the discriminator and generator
- Consistency term (CT): regularizes the generator.
- Dropout: regularizes the discriminator.

Both terms tend to zero when the network is stable.



Building a fake population of blips



Metrics and hypothesis to avoid misgenerations

Define metric m

$m(b_i, b_j)$:= similarity between two signals b_i and b_j.

$m(B, b_j) := \mu(M_j) \pm \varepsilon(M_j)$ where $M_j := \{m(b_i, b_j) \; \forall \; b_i \in B\}$

What metrics?

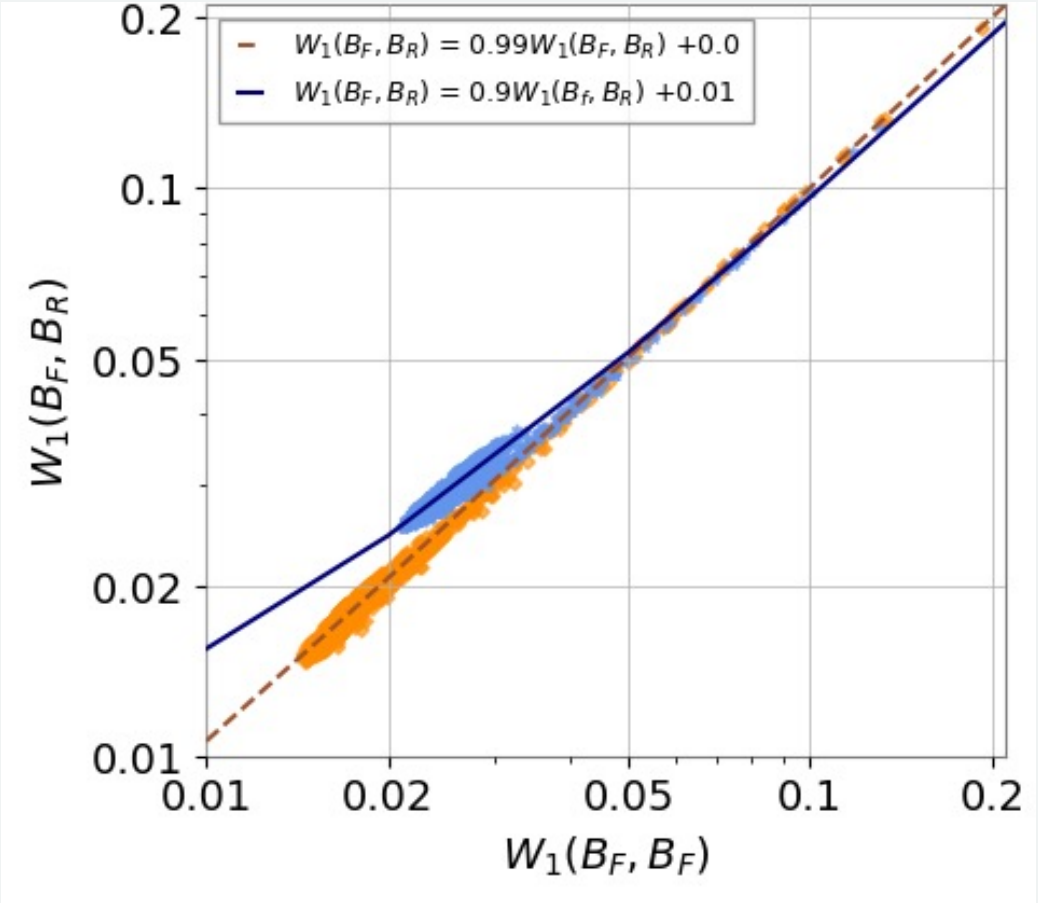
Similarity measures → Wasserstein distance (W_1), Match function (M_f), Normalized cross-covariance (k)

Assumption: CT-GAN learnt the underlying population except certain anomalies

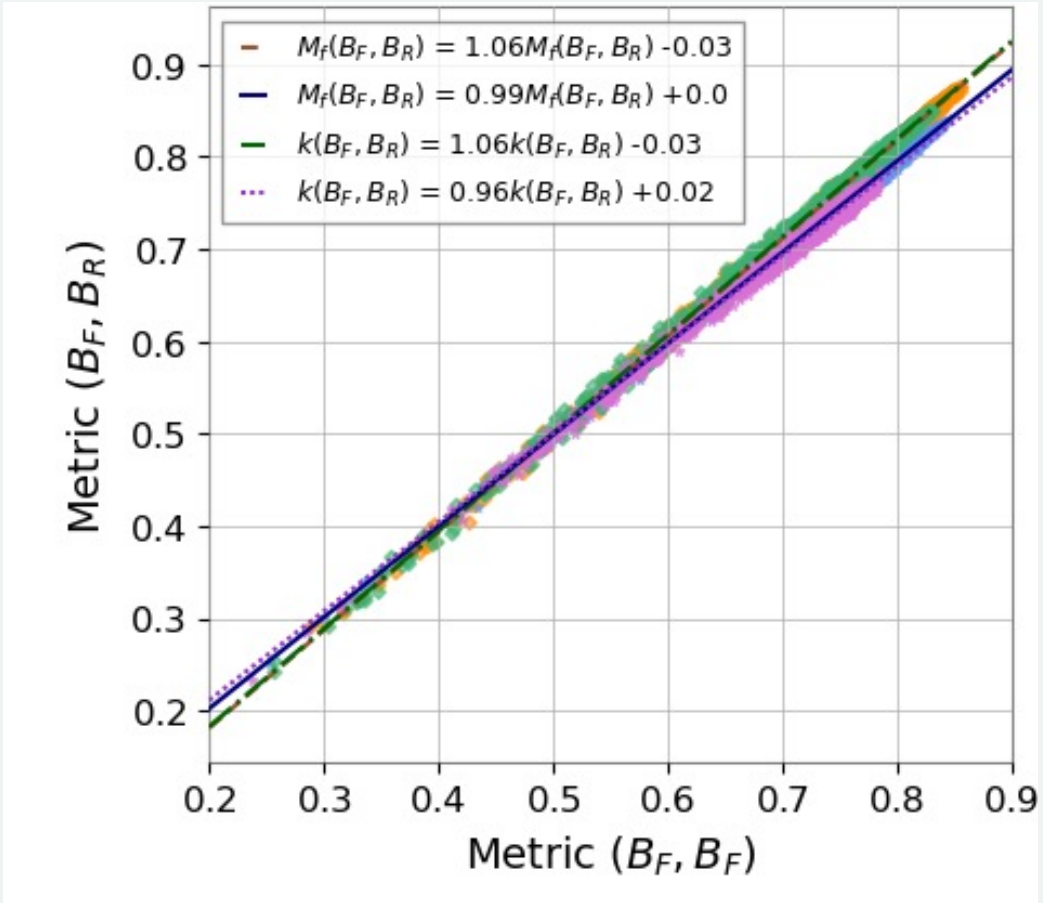
- If b_j is reliable blip, it will represent both real and fake populations → $m(B_{real}, b_j) \approx m(B_{fake}, b_j) \approx 1.0$
- If b_j is anomalous blip, it will not represent both real and fake populations → $m(B_{real}, b_j) \approx m(B_{fake}, b_j) \approx 0$

Hypothesis: $m(B_{real}, b_j)$ and $m(B_{fake}, b_j)$ are linearly correlated.

Results



SCAN ME

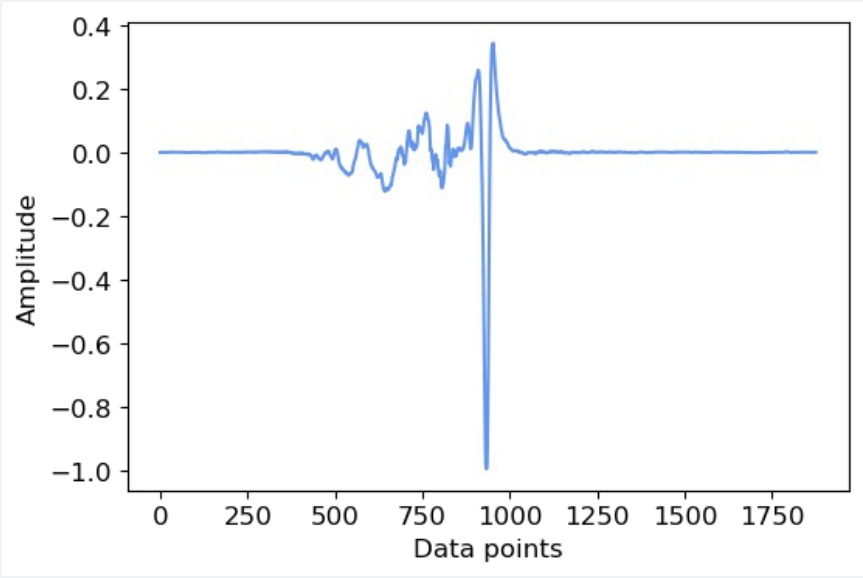


A practical example with gengli

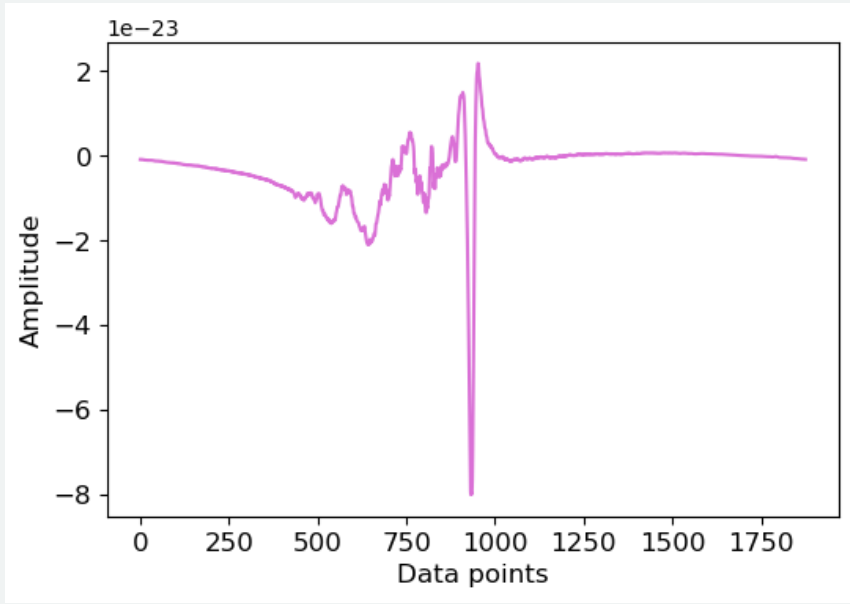
Mantainer: Stefano Schmidt

```
import gengli

g = gengli.glitch_generator('L1')

g_whithened = g.get_raw_glitch()

g_coloured_ET = g.get_glitch(4,
    srate = 16384,
    psd = 'EinsteinTelescopeP1600143',
    SNR = 10)
```

Full example: plot_glitch.py



Whitened

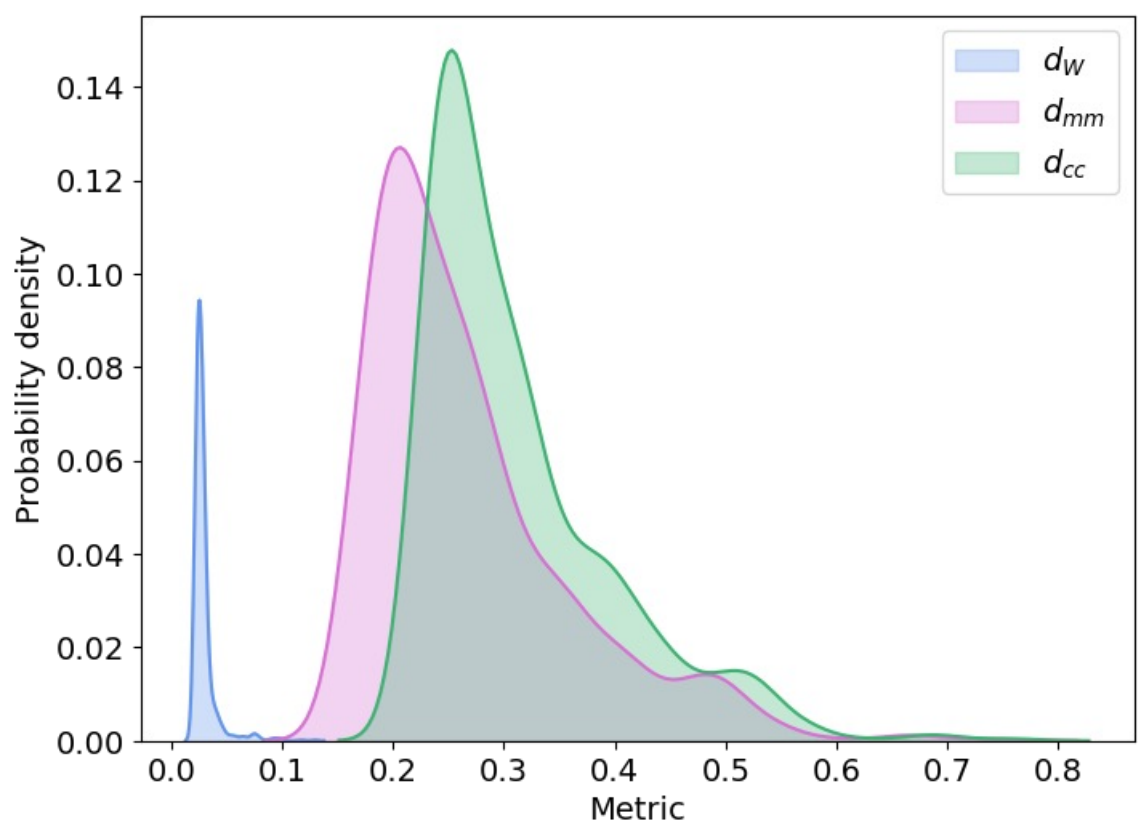


Einstein Telescope design

GitHub repository: https://git.ligo.org/melissa.lopez/gengli

Selecting reliable generations

Build initial data set (100 samples) to compute the confidence of the generated glitch



d_w : Wasserstein distance
d_{mm} : Mis-match (1- match)
d_{cc} : Cross covariance (1 – k)

Percentile $p \in [0.0, 1.0]$
If the generated glitch is in the percentile region it is accepted. Otherwise, it is dropped.

Credits: James Webb

2. Beyond current searches

What is the aim of this work?

Improve the robustness of GW searches by analysing trigger pipelines with ML

Why?

Intermediate-mass binary black holes are hard to detect

CBC searches generate triggers → "free" information that we can use

Similar ideas with cWB: Gayathri et al. (2020), Lopez et al. (2021)

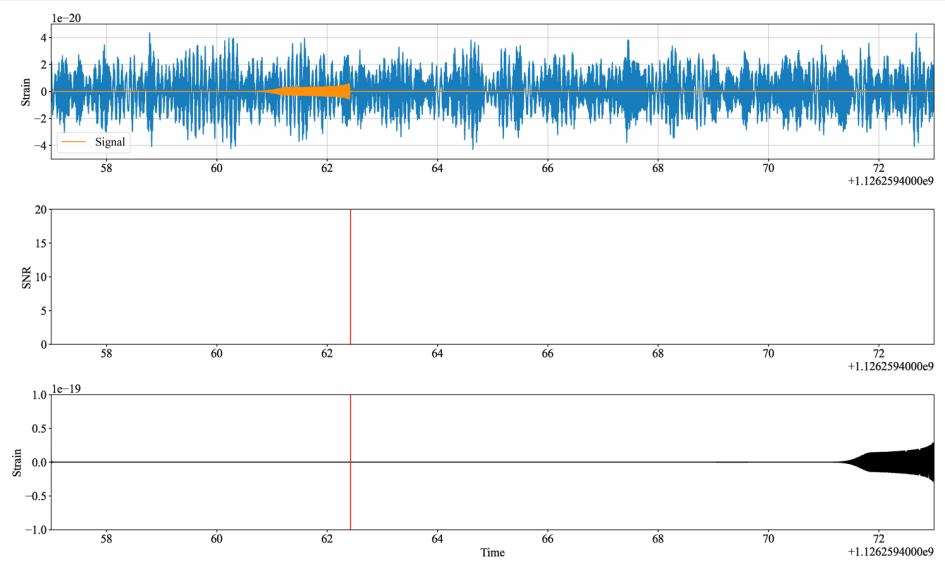
How are we going to do it?

Distinguish IMBH signals from glitches with GstLAL triggers from a *truncated* search

Matched filtering (MF) for CBC searches

Several pipelines for short modelled searches: **GstLAL**, PyCBC, SPIIR, …

MF := matching models (templates) to unknown signals

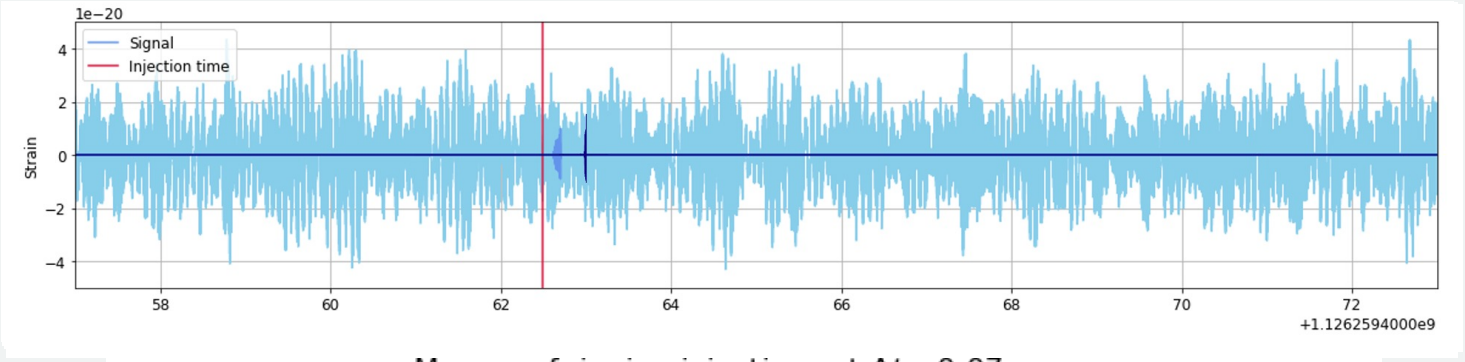


Unknown signal * Template Bank (simulated GW) → Triggers

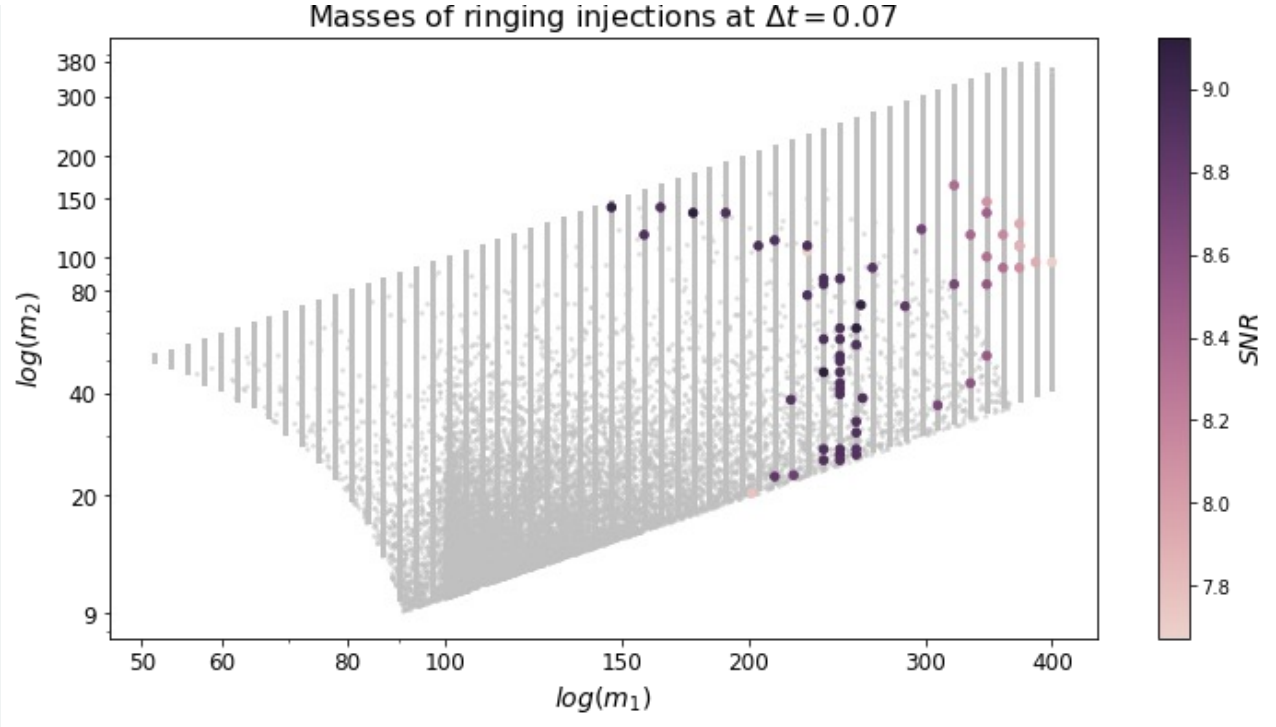
Idea: unknown signals generate *multiple* triggers. Can we detect *patterns*?

A simulated GW through a CBC pipeline

Δt: time where trigger happened – time where GW signal is present in the noise

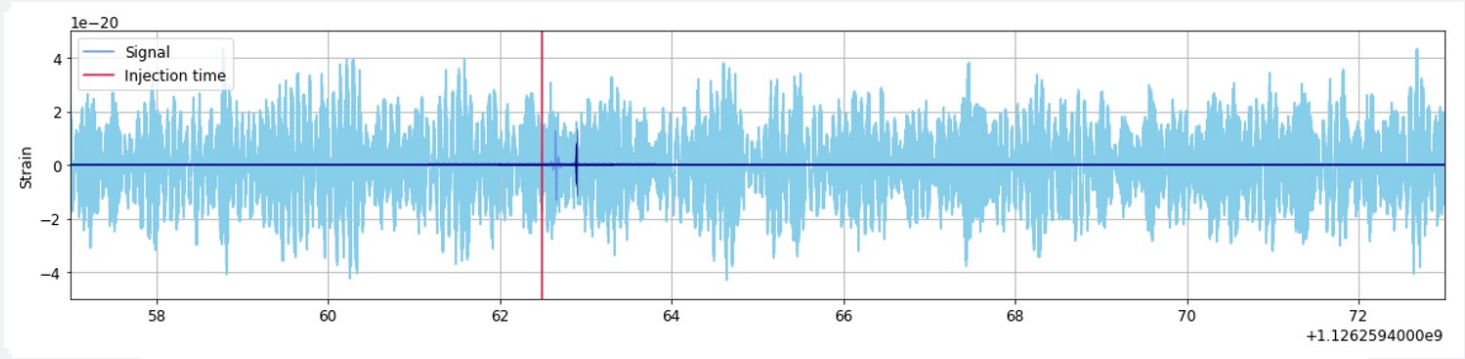


Trigger → template in template bank

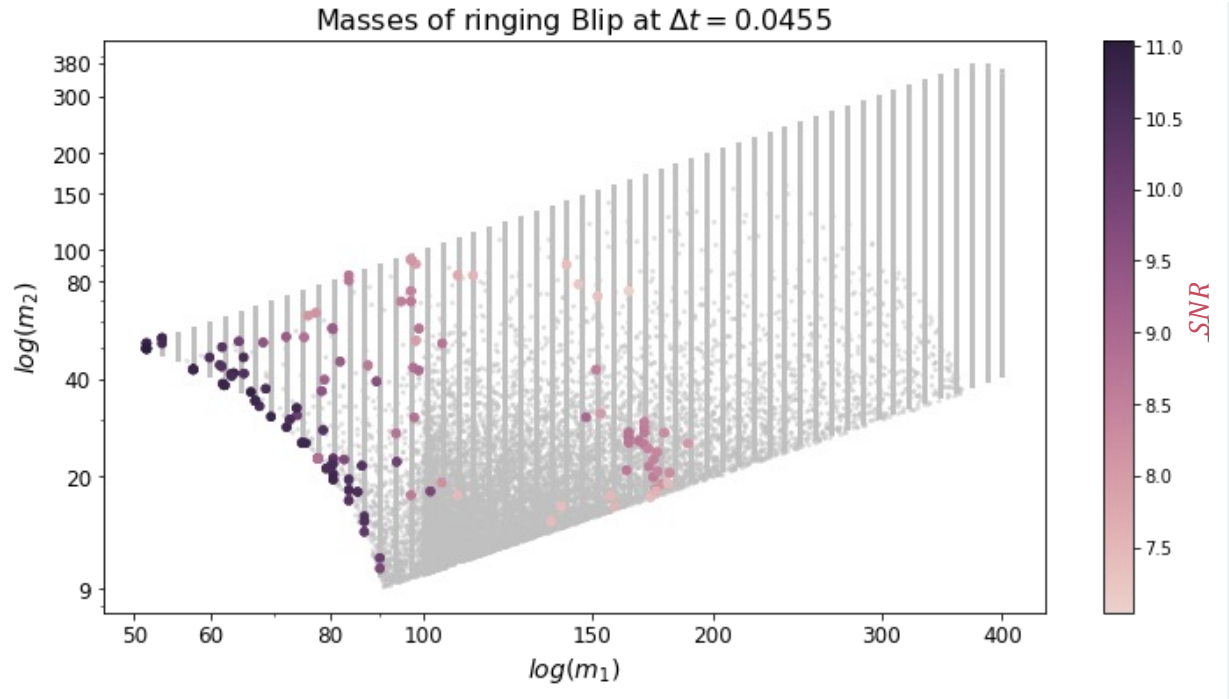


A blip through a CBC pipeline

Δt: time where trigger happened – time where glitch happened

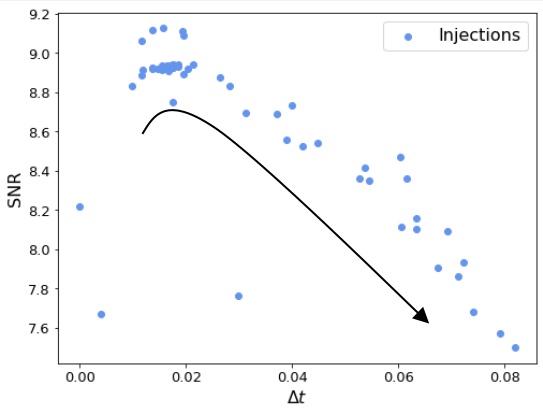


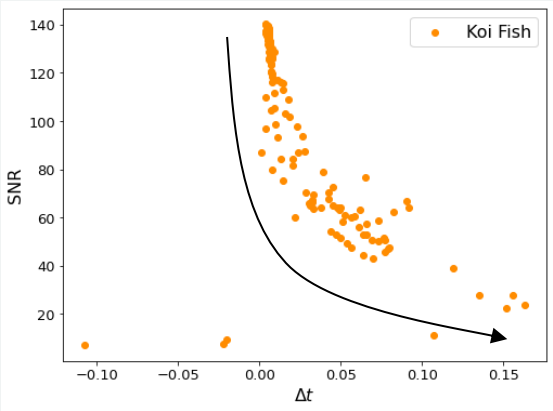
Trigger → template in template bank

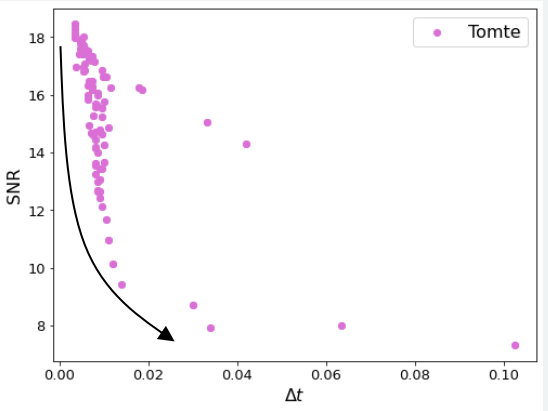


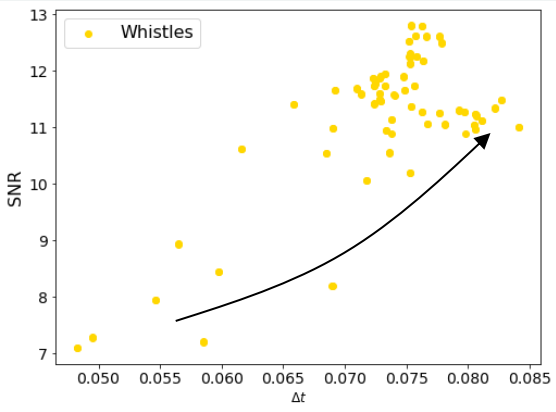
What does the pattern look like of SNR against time?

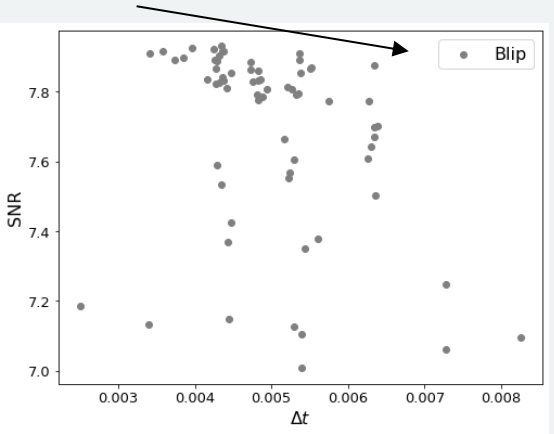
Taking time interval: -1s < event time < 1s











Not all glitches produce many injections/triggers

Simplest problem: binary classification with Gaussian Processes

Task: binary classification problem → GW signal (IMBH) vs glitch class

To simplify the problem, we discard time component.

For a single event we have multiple triggers:

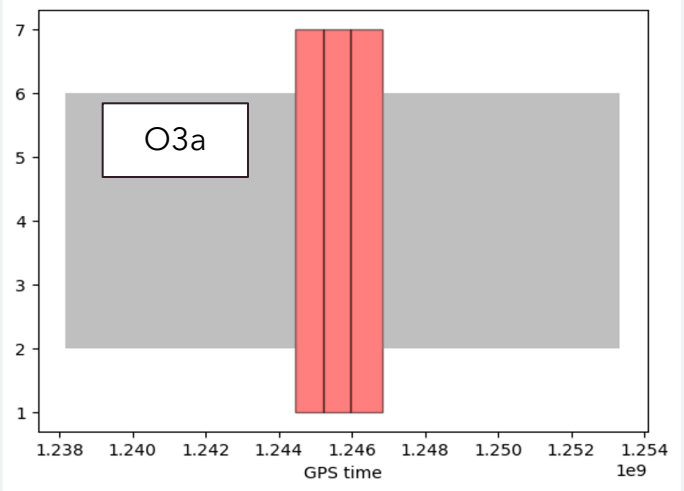
$(m_1, m_2, s_{1z}, s_{2z}, \chi^2, SNR)_i$ where i ∈ $triggers$ → μ$(m_1, m_2, s_{1z}, s_{2z}, \chi^2, SNR)$

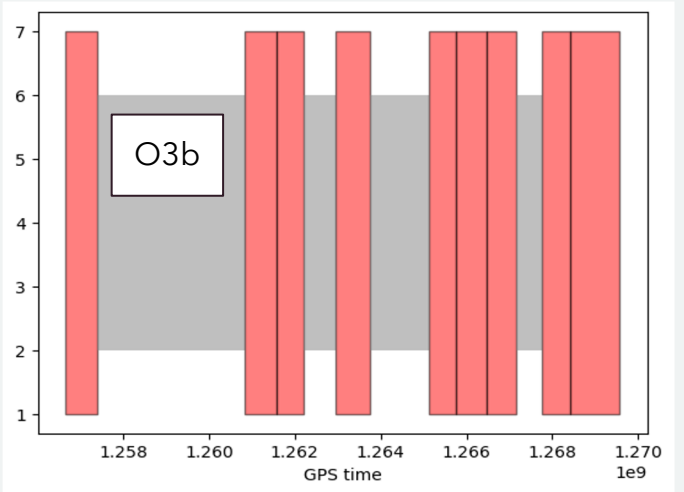
where χ^2 → signal consistency check of time-frequency evolution of the signal

SNR → match between signal and template

To simplify the problem, we balance the data set with undersampling.

Number of samples glitch class = number of samples IMBH

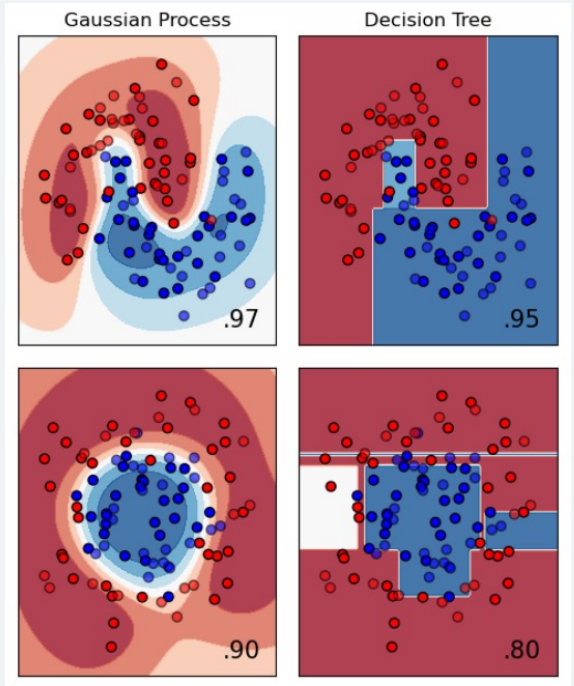




Simplest problem: binary classification with Gaussian Processes

Proof-of-concept:

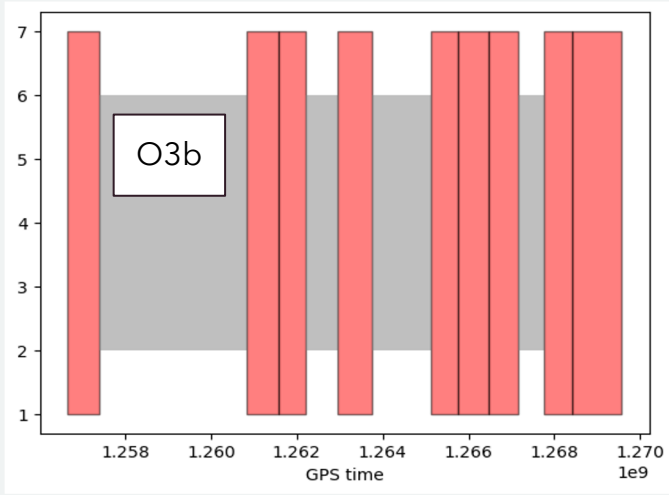
- *Feature vector:* $[\mu(SNR), \mu(\chi^2), \mu(m_1), \mu(m_2), \mu(s_{1z}), \mu(s_{2z})]$
- *Task:* binary classification problem with undersampling → IMBH vs glitch class
- *Algorithm:* Gaussian Process classifier with default values
- *Package: Scikit-learn*
- *Output: probability of being a glitch*



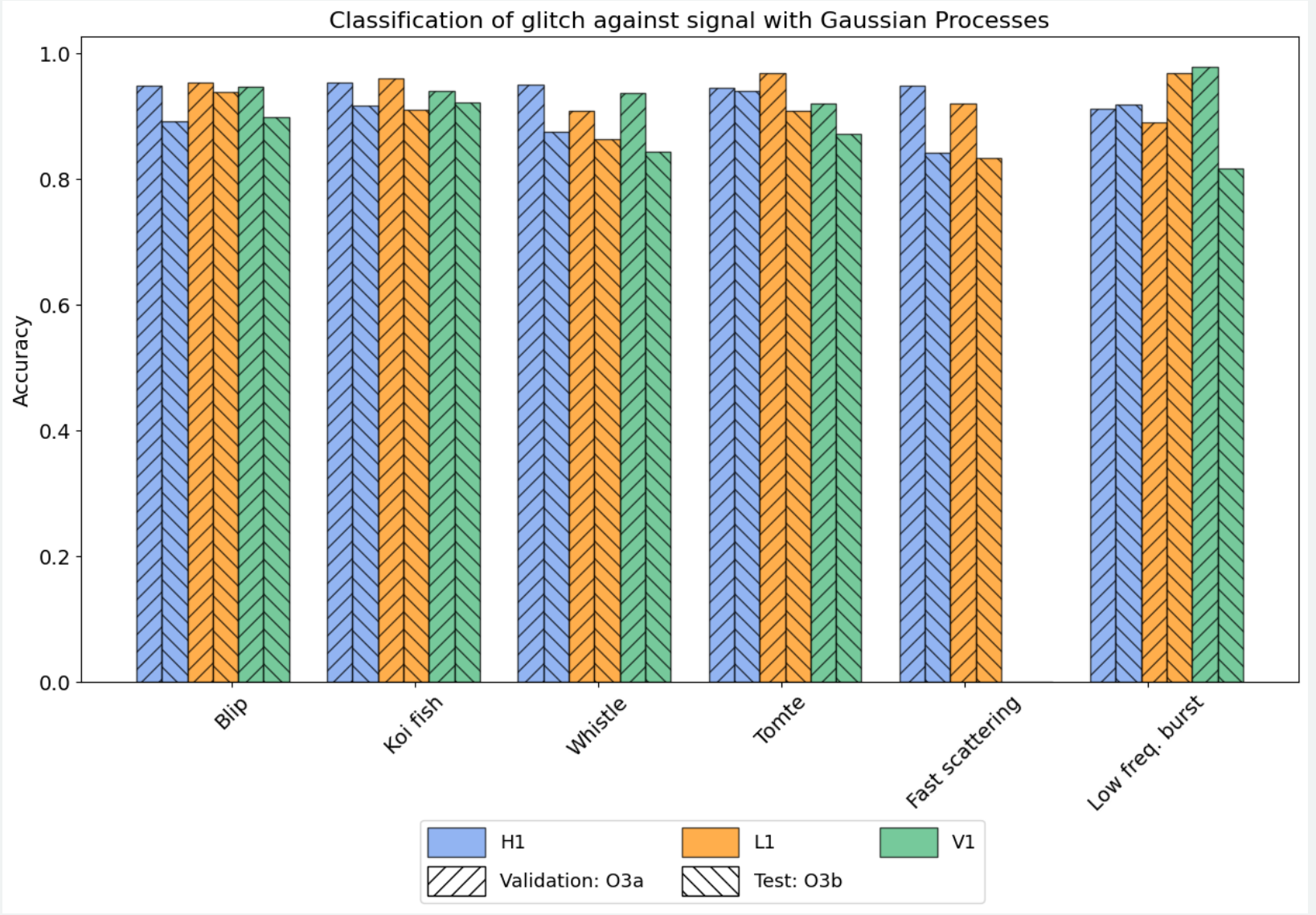
Proof-of-concept:

O3a

- *Feature vector:* $[\mu(SNR), \mu(\chi^2), \mu(m_1), \mu(m_2), \mu(s_{1z}), \mu(s_{2z})]$
- *Task:* binary classification problem with undersampling → IMBH vs glitch class
- *Algorithm:* Gaussian Process classifier with default values
- *Package: Scikit-learn*
- *Output: probability of being a glitch*



Preliminary results



Conclusions & future work

Challenge 1

- We can generate blip glitches.
- Generated blips represent the real blip population.
- Construct a full pipeline for glitch generation.
- Generalize to other types of glitches.
- Application of artificial data set.

https://git.ligo.org/melissa.lopez/gengli

SCAN ME

Challenge 2

- Proof-of-concept binary classification
- At least a testing accuracy > 0.8
- Get more data!
- Include time component: ordering might be relevant


Thank you for listening!
Questions?