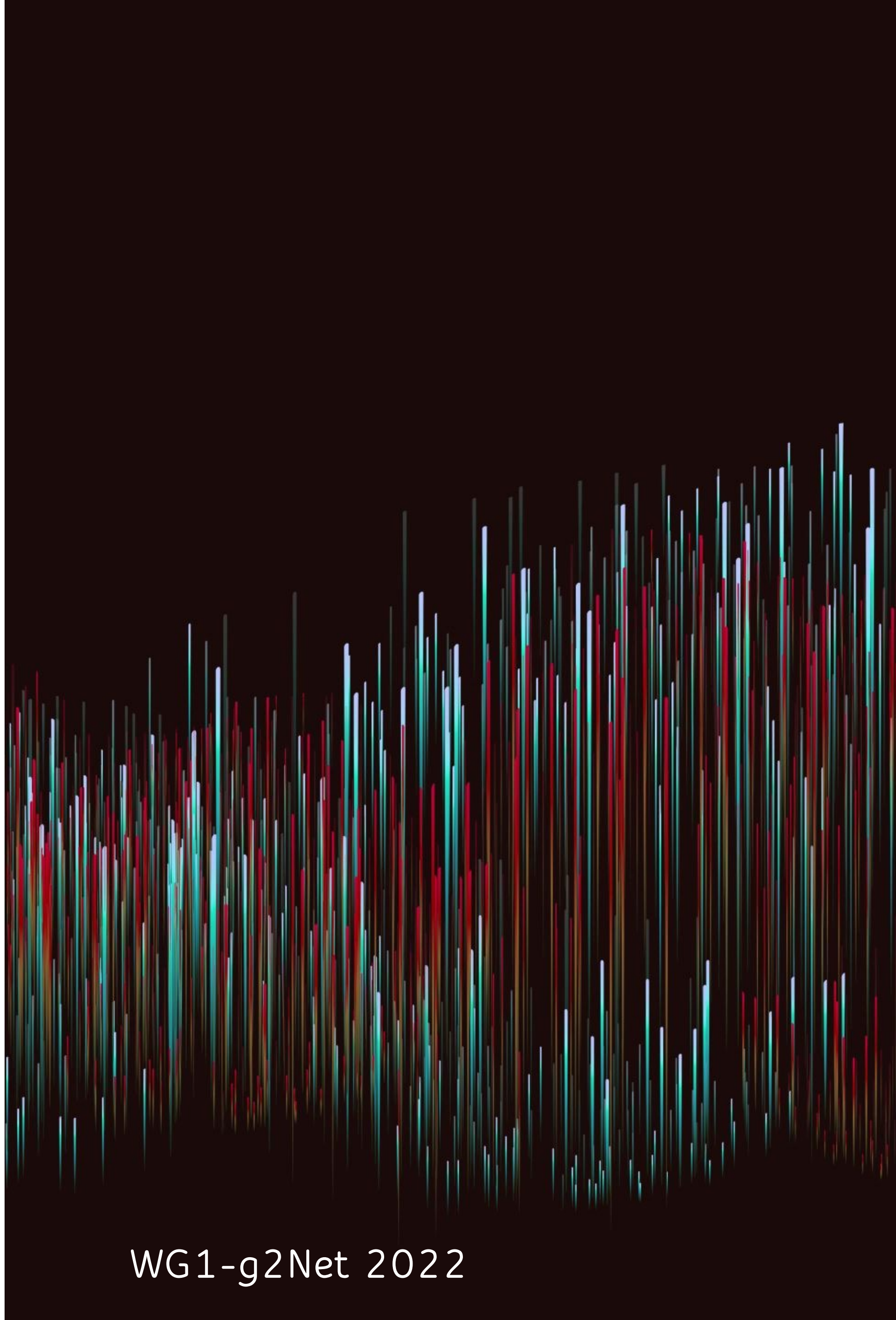
Simulating transient noise bursts in LIGO

WITH GENERATIVE ADVERSARIAL NETWORKS

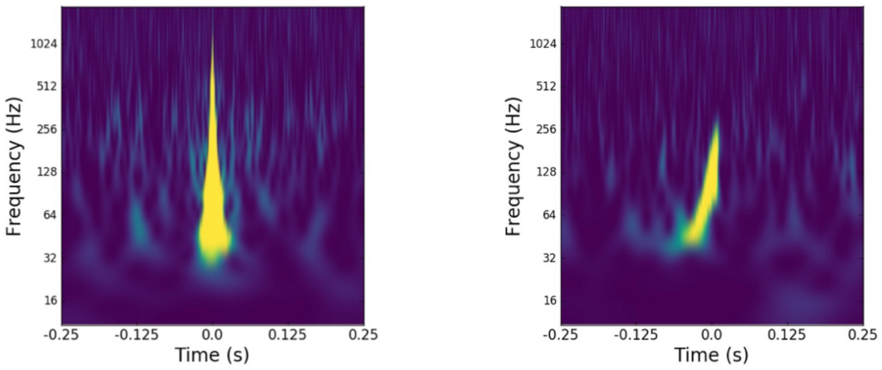
Melissa Lopez[1,2], Vincent Boudart[3], Kerwin Buijsman[2], Amit Reza[1,2], Sarah Caudill[1,2]

1. Institute for Gravitationaland Subatomic Physics (GRASP), Utrecht University, the Netherlands.
2. Nikhef, Amsterdam, the Netherlands.
3. Institut de Physique, Université de Liège, Belgium.

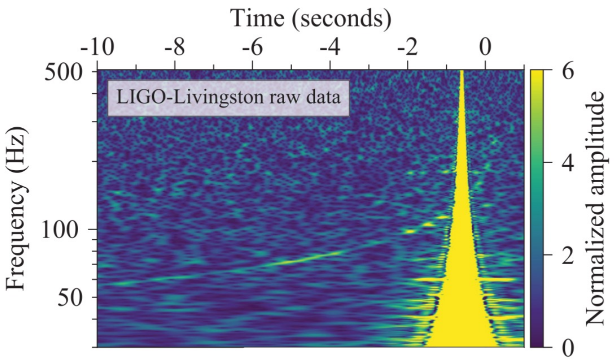


Transient noise in LIGO (glitches)

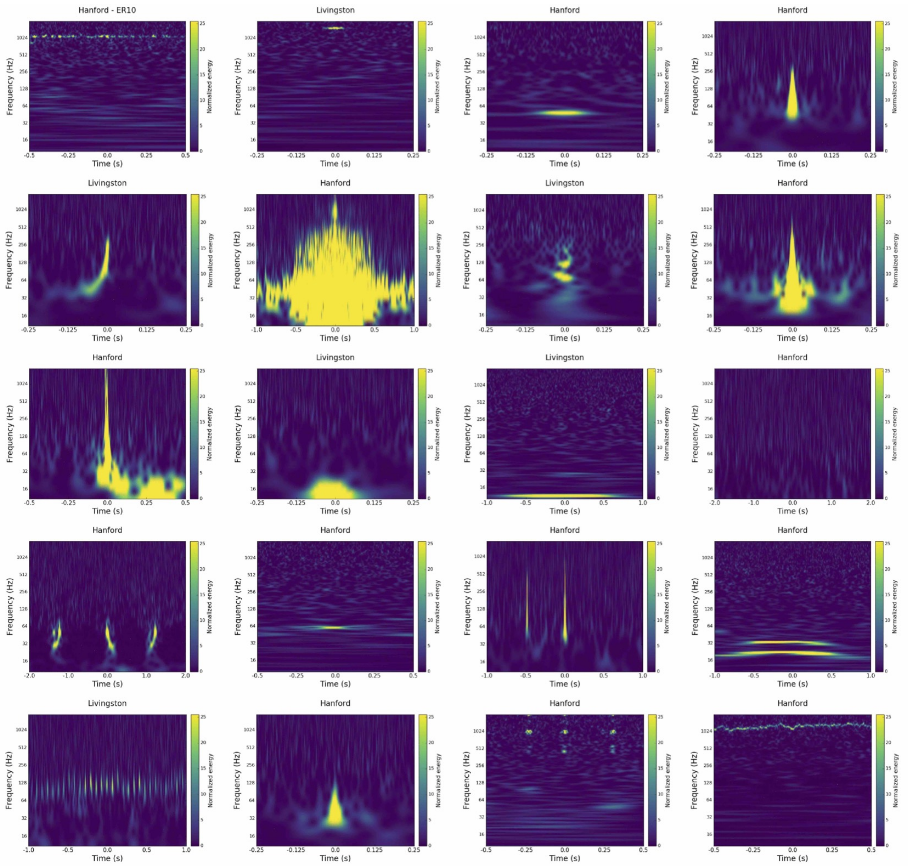
- Caused by instruments or environment (known or unknown)
- Diminish scientific data available
- Hinder GW detection (mask and/or mimic)



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



GW masked by glitch (GW170817)



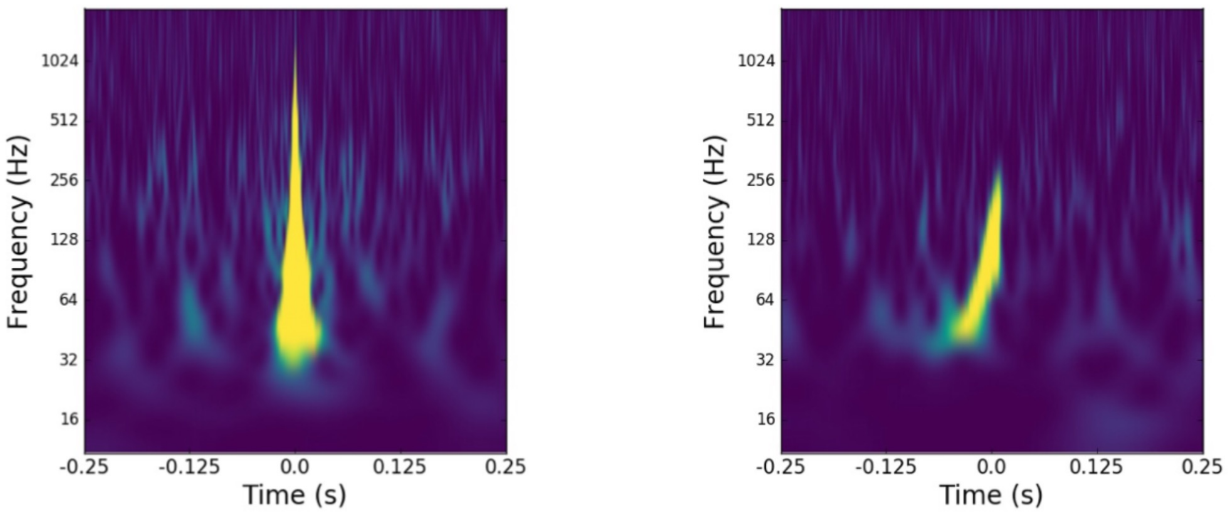
Examples of glitches.

S. Bahaadini. Inf. Sci. 2018

Motivation

- Create an open-source interface for producing time-domain glitch “waveforms”
- Generate glitches in time domain with GANs
- To use in different applications

Data set



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

We focus on blips from L1 and H1, O2

Simple morphology and abundant

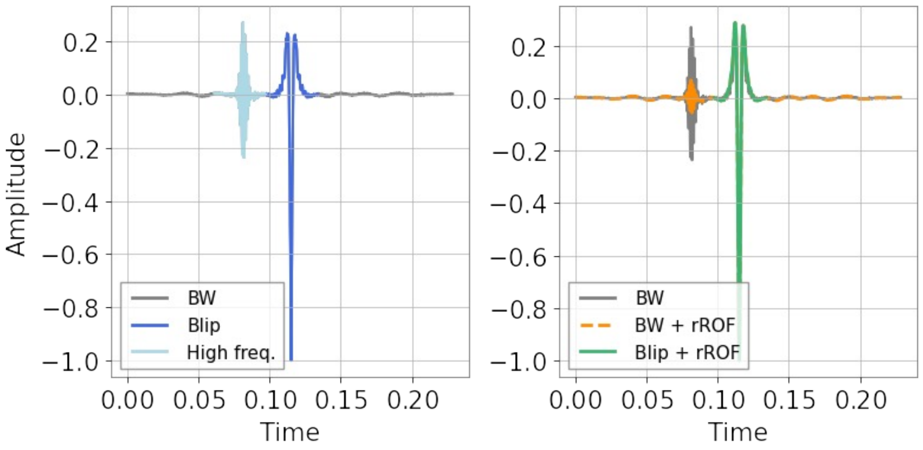
Similar to other GWs

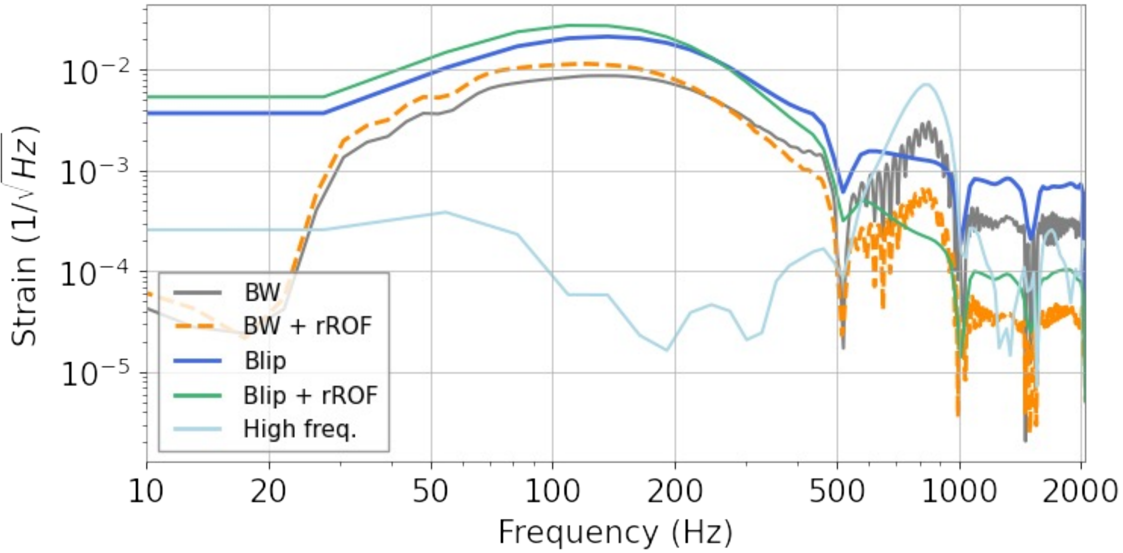
[30, 250] Hz

The noise will hinder our Machine Learning algorithm.
Can we separate the glitch from the noise?

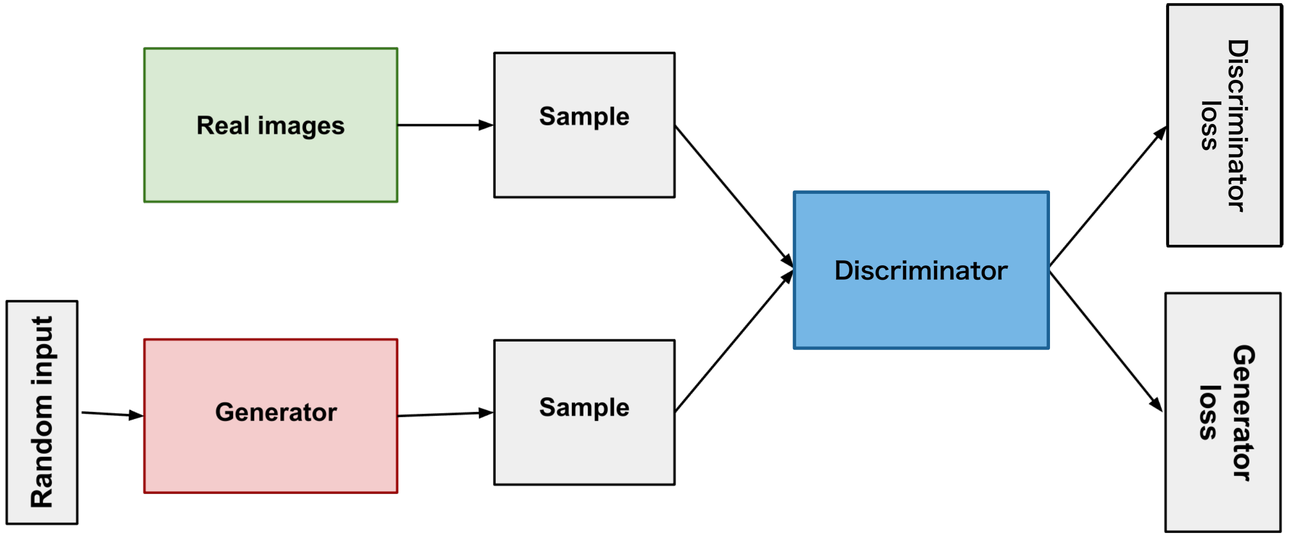
Pre-processing

Gravity Spy (selection) → BayesWaves (reconstruction) → rROF (denoising) → Gravity Spy (re-evaluation) → GAN input





Generative Adversarial Networks



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

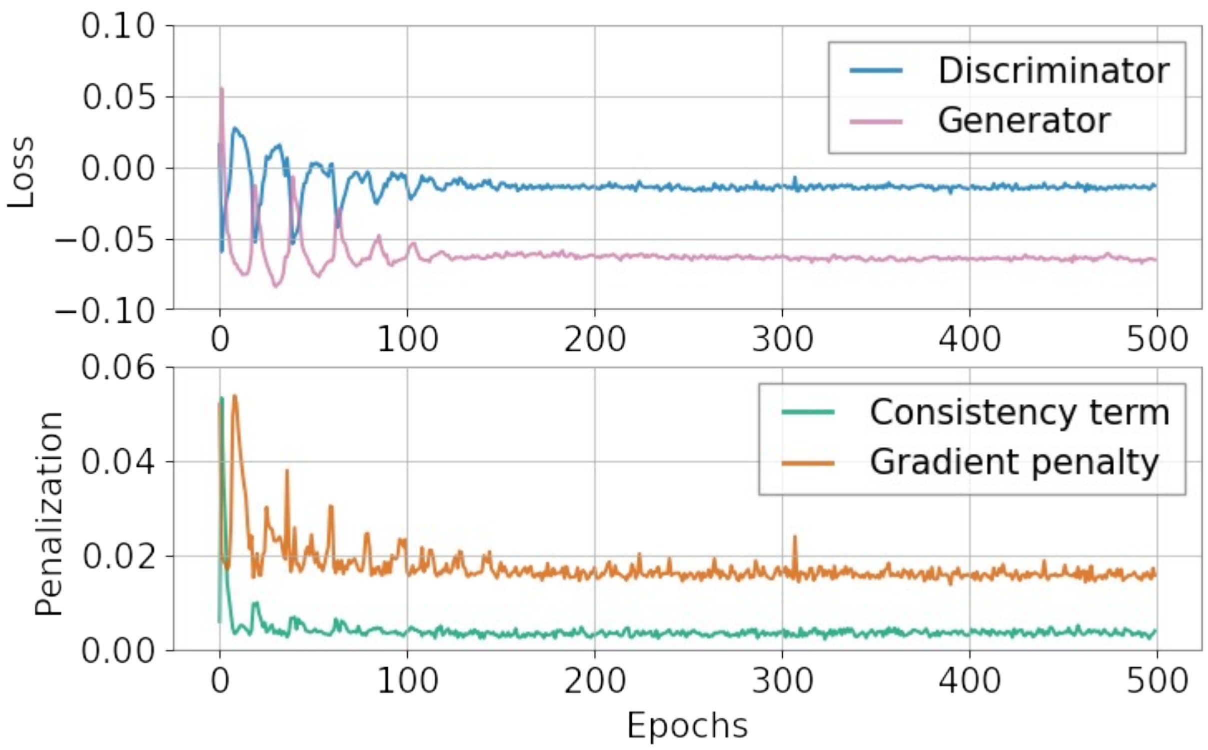
Network employed: CT-GAN (Wei, ICLR 2018)

CT-GAN: GP + CT with Dropout

Some intuition from the experiments:

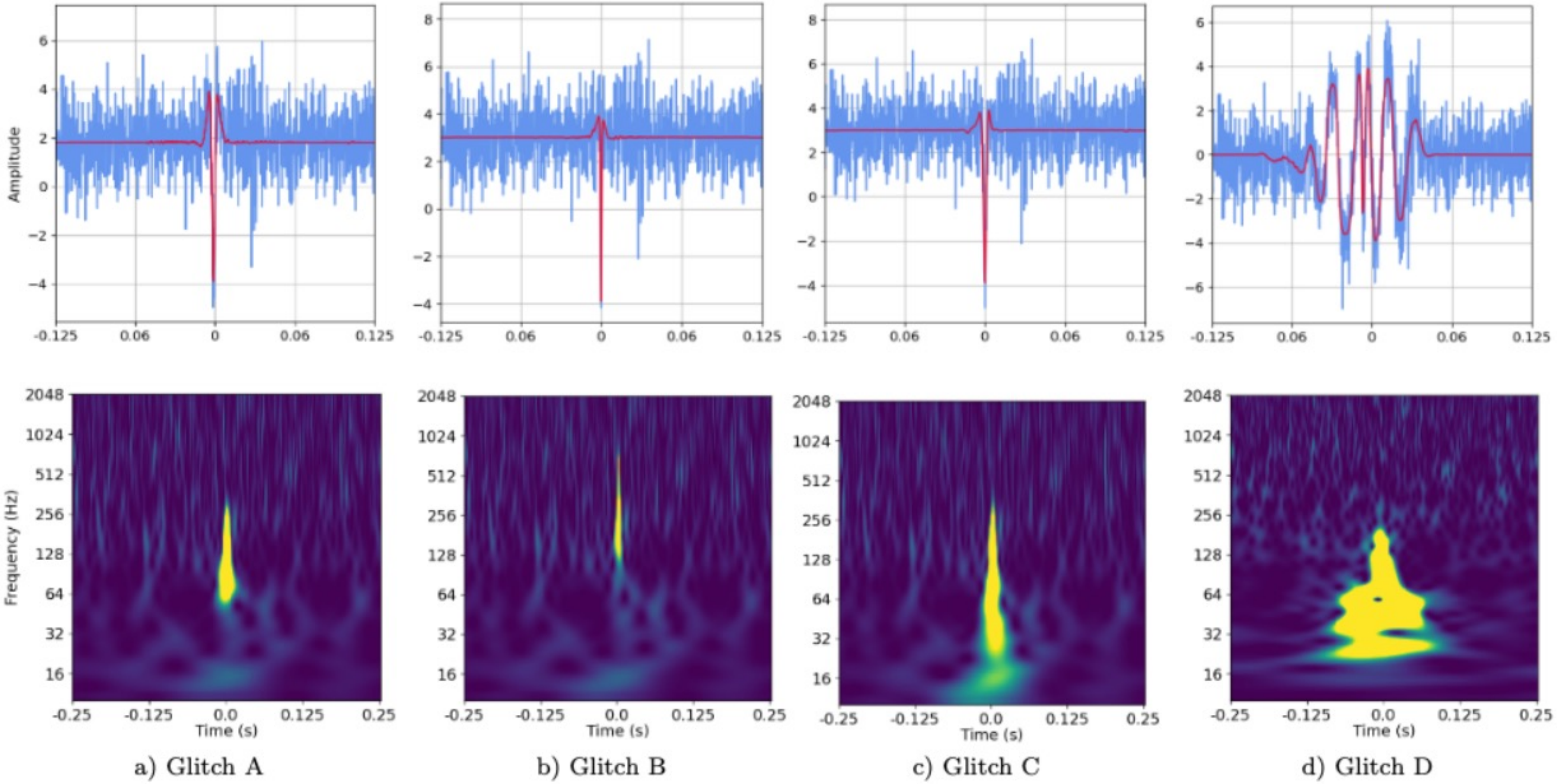
- Gradient Penalty (GP): balances the loss of the discriminator and generator
- Consistency term (CT): regularizes the generator.
- Dropout: regularizes the discriminator.

Both terms tend to zero when the network is stable.



Building a fake population of blips

CT-GAN learnt to generate reliable blips but also anomalies, since our real data set is not perfect.



Simulated glitches in real whitened noise from H1. (Top) timeseries representation. (Bottom) Q-scan representation.

Metrics 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 $M_j := \{m(b_i, b_j)\ \forall\ b_i \in B\}$

Wasserstein distance (W_1): or Earth's mover distance estimator computes similarities between two distributions.

Match function (M_f): inner product between two normalized signals maximized over time t and phase ϕ.

$$M_f(a, b) := \max_{t,\phi} \langle \hat{a}, \hat{b} \rangle.$$

Normalized cross-covariance (k): assuming two random processes X and Y,

$$k = \max\left(\frac{K_{X,Y}(t_1, t_2)}{\sigma_X \sigma_Y}\right) \quad \text{where} \quad K_{X,Y}(t_1, t_2) \equiv E[(X_{t_1} - \mu(X_{t_1}))\overline{(Y_{t_1} - \mu(Y_{t_1}))}]$$

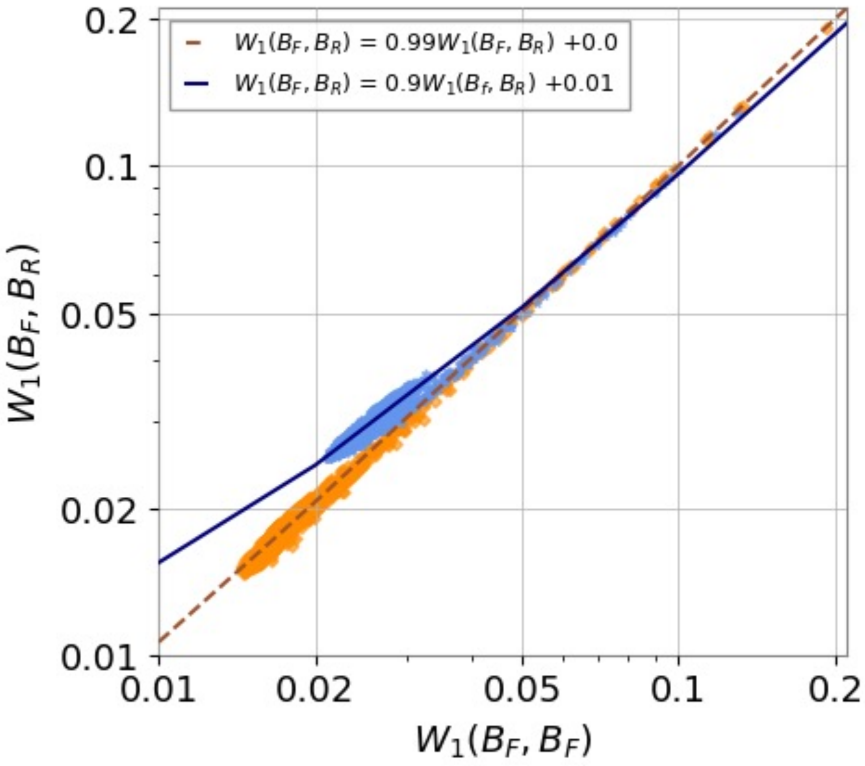
Hypothesis description

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

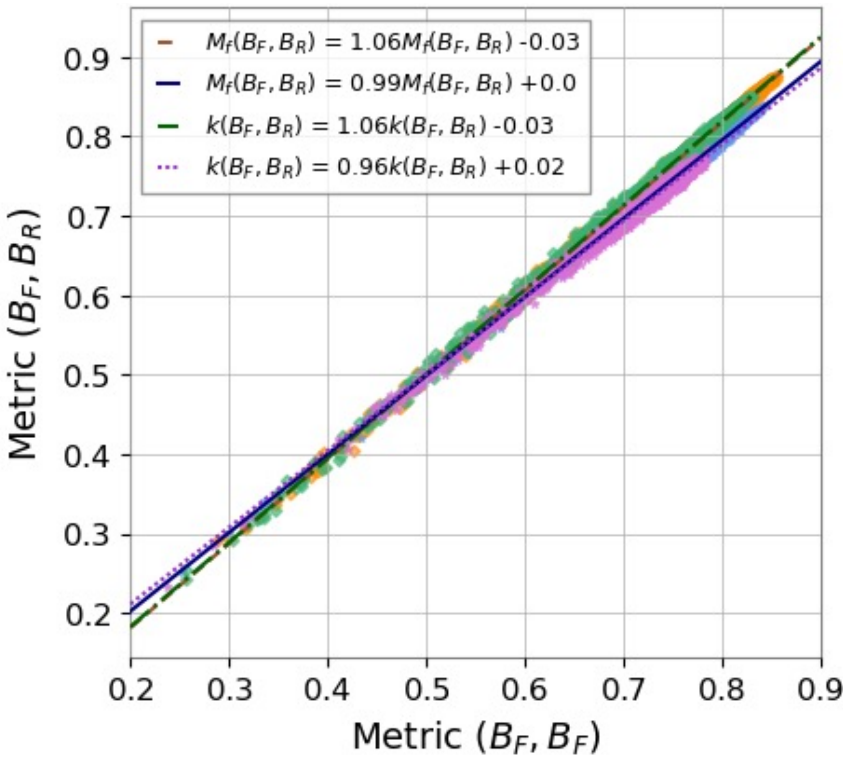
- If bj is reliable blip, it will represent both real and fake populations → $m(B_{real}, b_j) \approx m(B_{fake}, b_j) \approx 1.0$
- If bj is anomalous blip, it will not represent both real and fake populations → $m(B_{real}, b_j) \approx m(B_{fake}, b_j) \approx 0$

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

Results



$W_1(B_F, B_R) = 0.99W_1(B_F, B_R)$ +0.0
$W_1(B_F, B_R) = 0.9W_1(B_f, B_R)$ +0.01
0.2
0.1
0.05
0.02
0.01
0.01 0.02 0.05 0.1 0.2
$W_1(B_F, B_R)$
$W_1(B_F, B_F)$



$M_f(B_F, B_R) = 1.06M_f(B_F, B_R)$ -0.03
$M_f(B_F, B_R) = 0.99M_f(B_F, B_R)$ +0.0
$k(B_F, B_R) = 1.06k(B_F, B_R)$ -0.03
$k(B_F, B_R) = 0.96k(B_F, B_R)$ +0.02
0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2
0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
Metric (B_F, B_R)
Metric (B_F, B_F)

Applications

a) Glitch population statistics

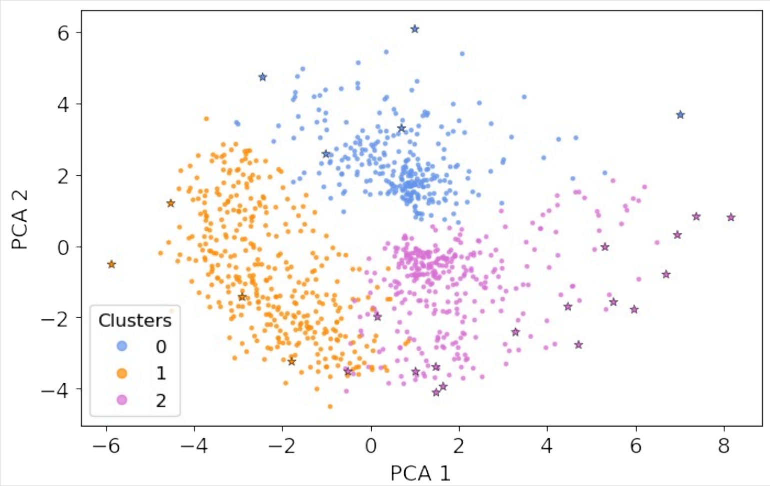
b) Glitch template bank

c) Mock data challenges.

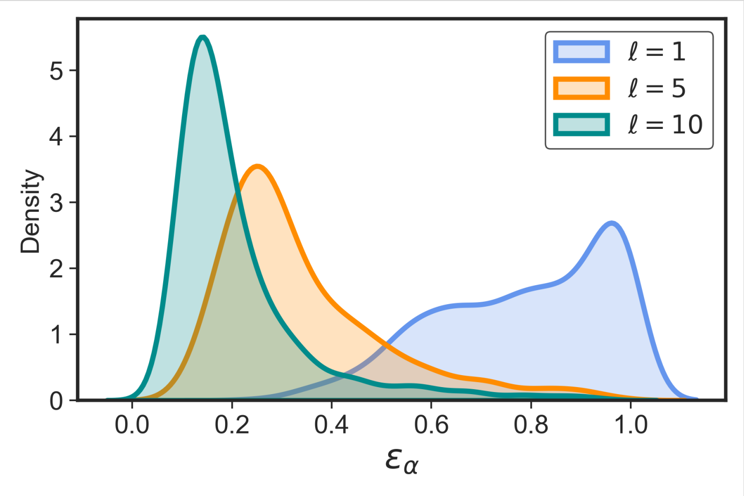
d) New glitch detection

e) Improving glitch classification

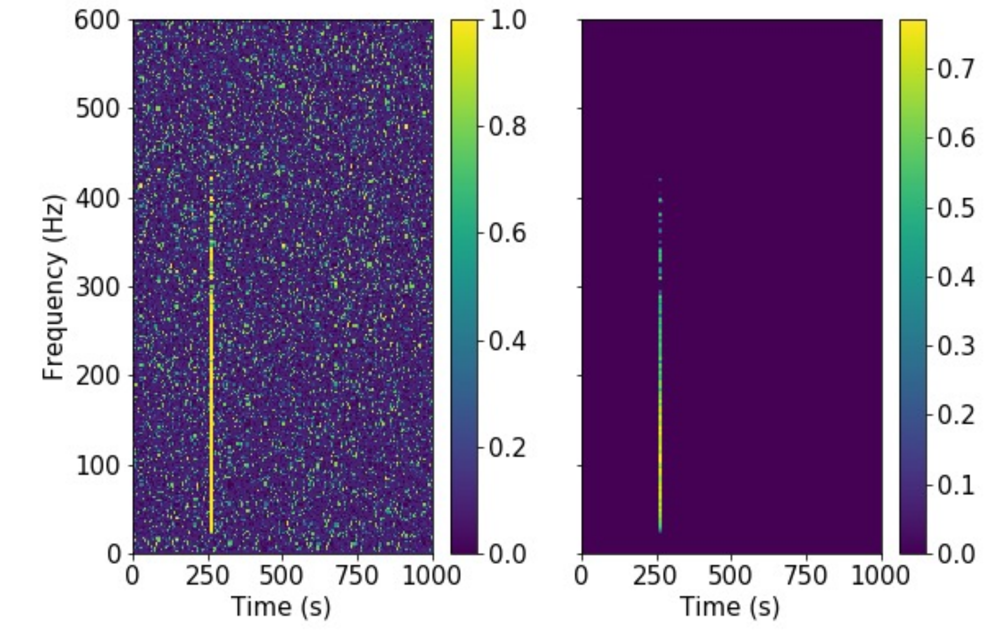
a)



b)



c)



https://arxiv.org/abs/2201.08727

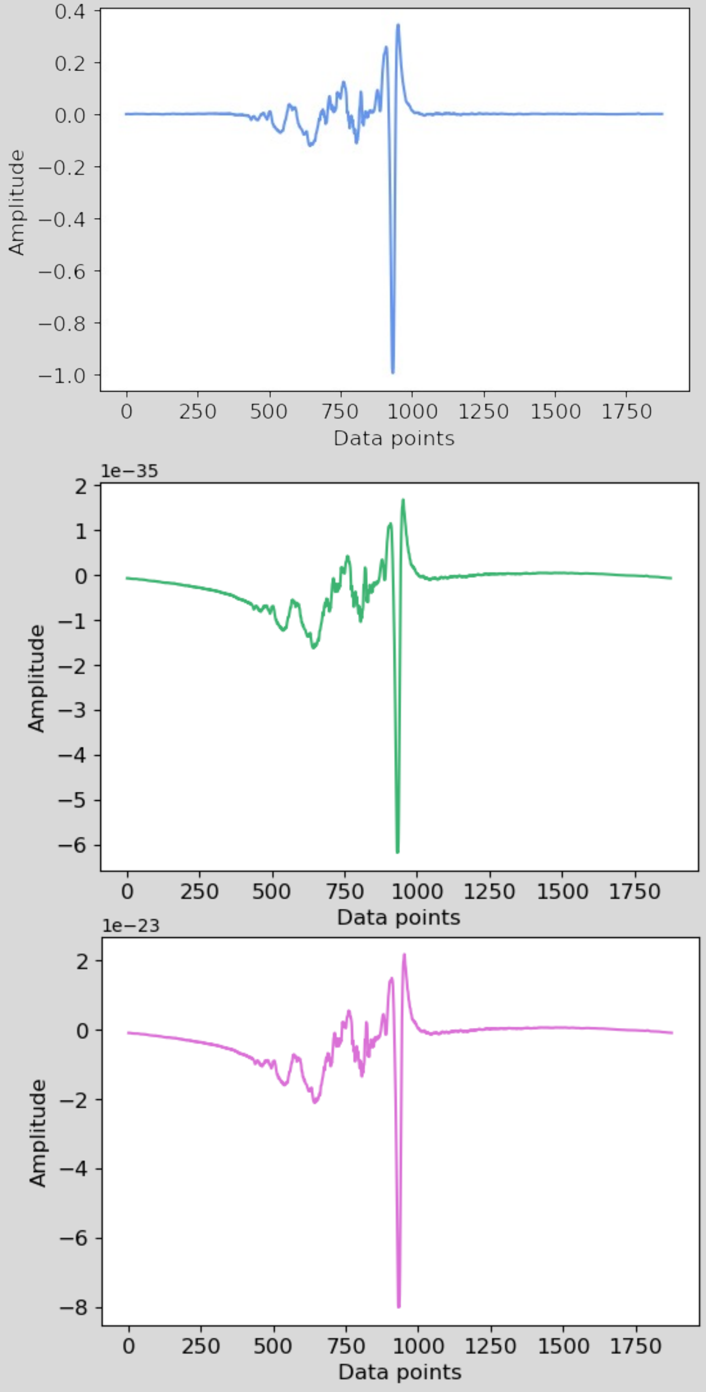
Colouring, resampling and rescaling

CT-GAN $\longrightarrow g_w \longrightarrow g_c = ifft(\sqrt{PSD}\,\widetilde{g_w})$

CT-GAN $\longrightarrow g_{4k\ Hz} \longrightarrow g_{8kHz} = resample(\,g_{4kHz})$

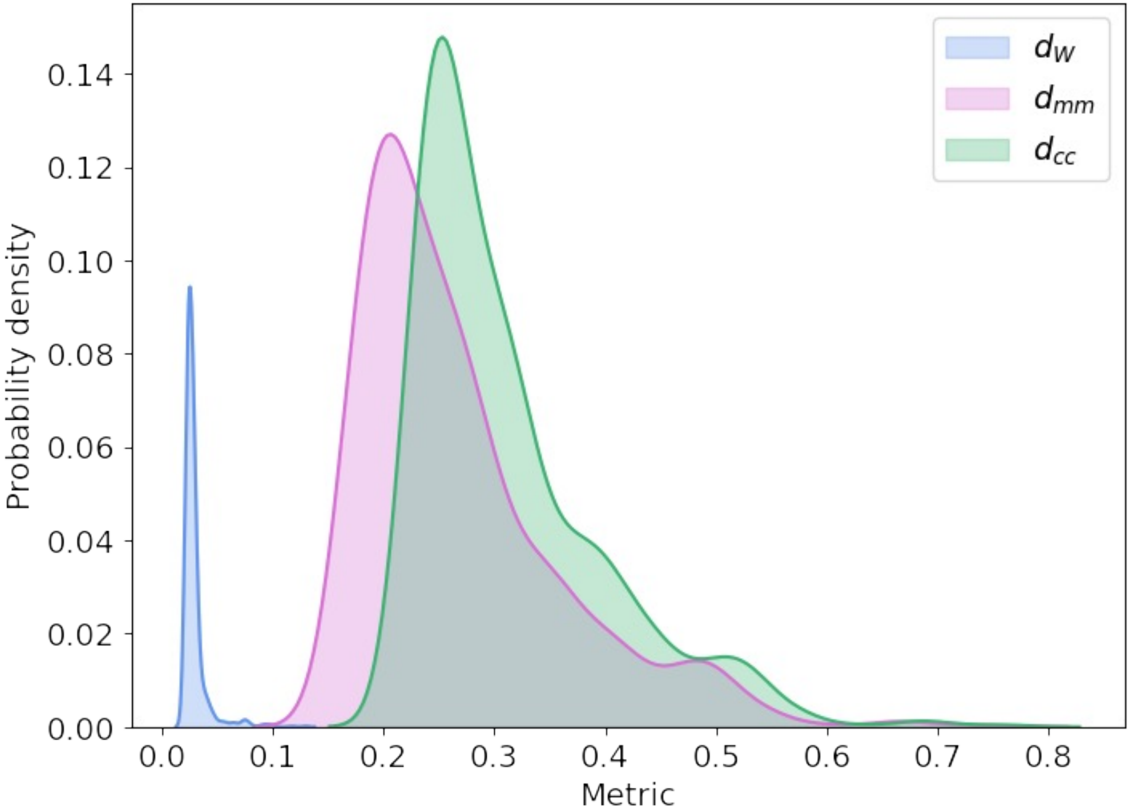
CT-GAN $\longrightarrow g_\rho \longrightarrow g_{\rho_{opt}} = 4\alpha^2 \int_{f_{min}}^{f_{max}} \frac{|\tilde{g}(f)|^2}{PSD(f)} df = \alpha^2 g_\rho$

$$\boxed{g_{c,\rho} + n_c = s_c \longrightarrow s_w}$$



Selecting reliable generations

Build initial data set (1000 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.

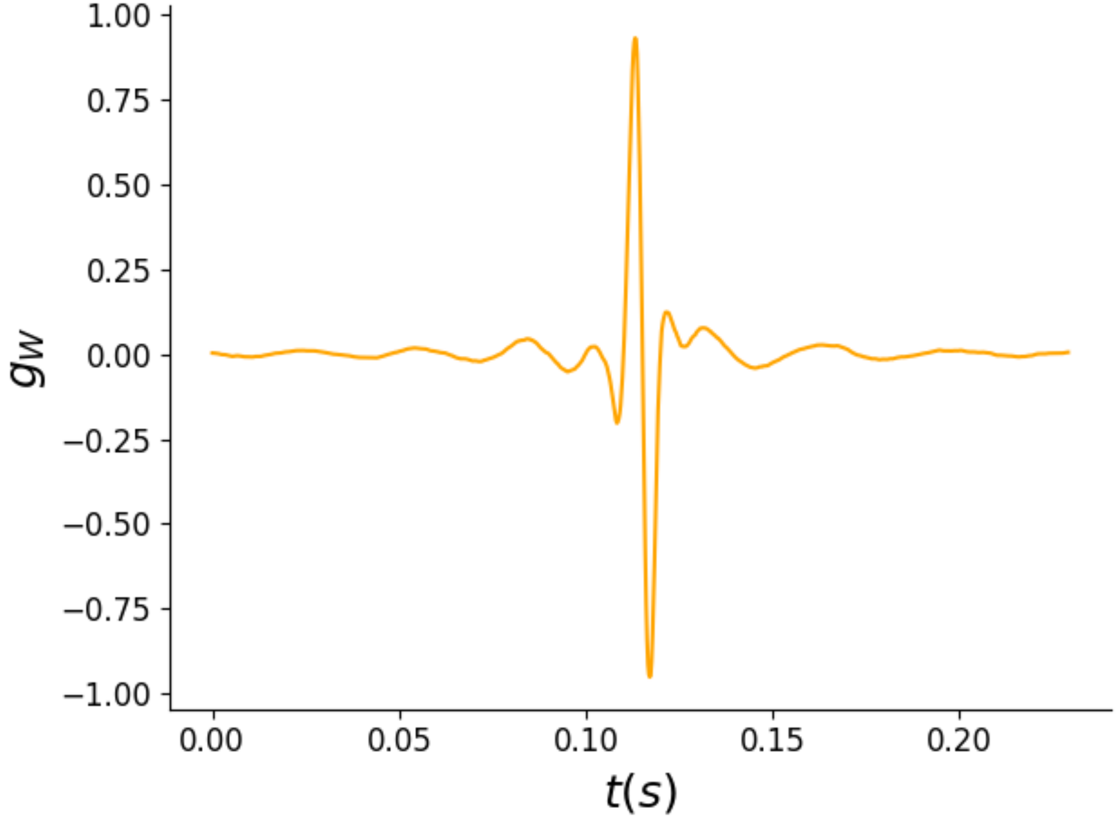
A practical example with *gengli*

GitHub repository (work in progress): https://git.ligo.org/melissa.lopez/gengli

```
import gengli

g = gengli.glitch_generator('L1')
g_whithened = g.get_glitch()
g_coloured = g.get_glitch(10,
        srate = 16384,
        psd = 'EinsteinTelescopeP1600143',
        SNR = 10)
```

Full example: plot_glitch.py



Conclusion and future work

- 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://arxiv.org/abs/2203.06494

https://dcc.ligo.org/P2200115

Thank you for listening!

Special thanks to: Chris Messenger, Siddharth Soni, Jess McIver, Marco Cavaglia, Alejandro Torres-Forne and Harsh Narola.