IA for online gravitational wave studies

A.Boudon, G.Joubert, S.Viret

IP2I Lyon ***Virgo group***

Context
Example
What's next ?

→ ML and gravitational waves:

→ First paper in 2016, since then the field is following steadily the latest developments. Not a huge community but quite proactive (*more info:* https://iphysresearch.github.io/Survey4GWML/).

2016: CNN
(*eg:* https://arxiv.org/abs/1701.00008v3)

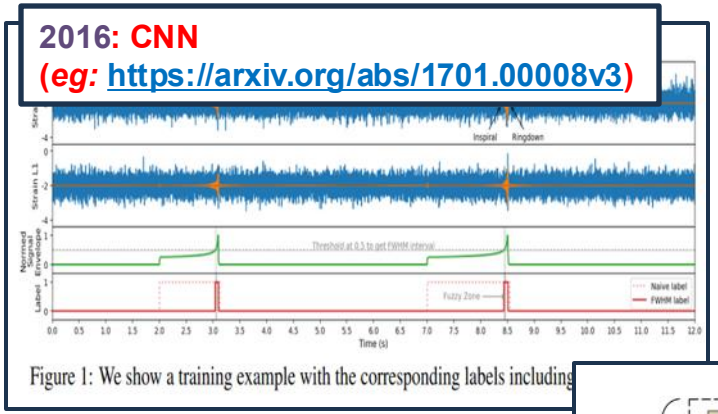
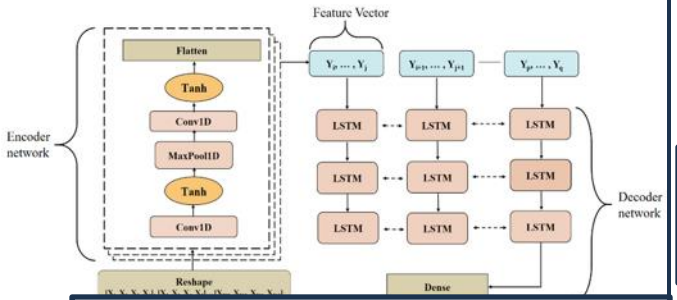
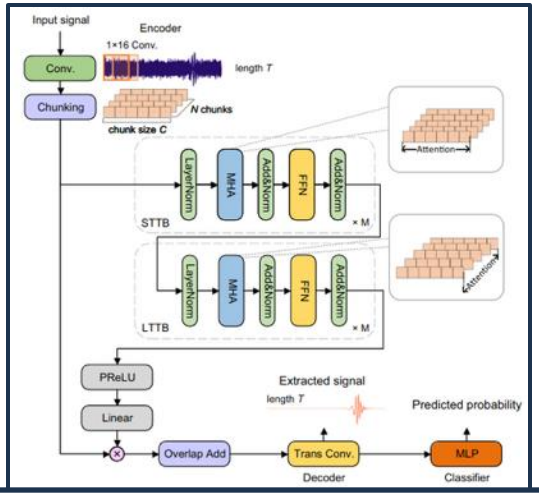


Figure 1: We show a training example with the corresponding labels including



2019: Smart denoising with LSTM
(https://arxiv.org/abs/1909.06367v2)



2024: Denoising and categorization with Transformers
(https://arxiv.org/abs/2406.06324)

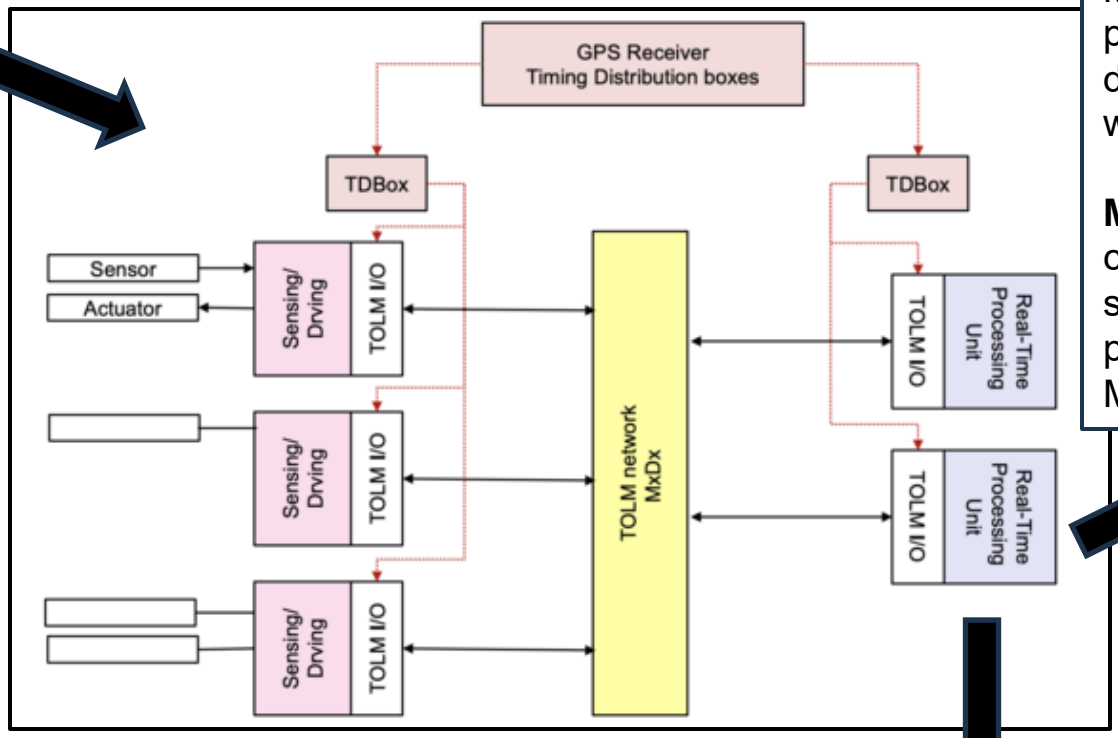
2016 **2019** **2024**

→ **Virgo DAQ ecosystem:**

1. Frontend (~μs)

Fast control loops, ADC/DAC, fast processing

Main constraints: low noise, low power (*passive thermal dissipation only for boards in vacuum*)



2. RTPC (~ms)

More complex processes, run on dedicated machines with RT OS (RTAI)

Main constraints: code need to be suitable for RT processing. If using ML here

3. Online data processing (~s)

On CPU farms for the moment. Possible to use GPUs at this level

Main constraints: robustness of the network as you don't want to retract public alerts

→ Machine learning online in VIRGO: current status

→ 1. Front-end level:

- **Potential use cases**: RT position control, photodiode signal shaping
- **Status:** some very first works on RT position control have been presented within the collaboration

→ 2 Real time PC level:

- **Potential use cases:** complex control loops, signal reconstruction
- **Status:** some first developments done, aiming for detector upgrades and 3rd gen interferometers (*eg.* ***Seismic noise reduction with reinforcement learning in LIGO***).

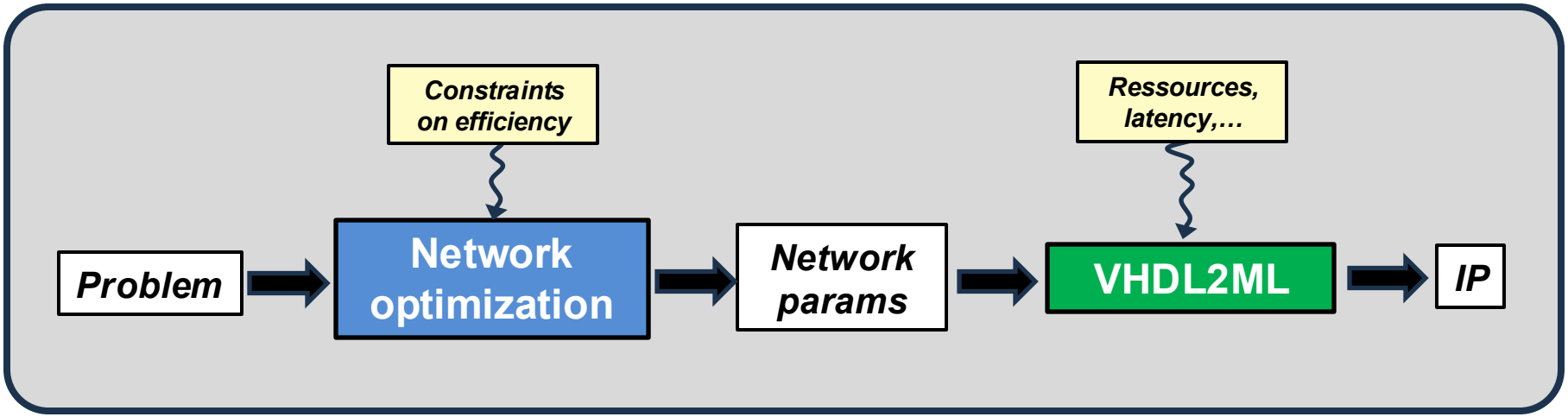
→ 3 Online processing level:

- **Potential use cases:** detection algorithms, possibly parameters estimation but more challenging
- **Status:** considering latency available here, most of the developments can be applied here. Already one ML detection pipeline has been deployed online (***https://arxiv.org/abs/2505.21261***)

→ Latency/power constraints are clearly different for all those points, but in all cases we need to be **very robust,** so we should aim for the simplest architecture, and if possible for the smallest possible network

→ **Our strategy:**

→ Try to develop a simple approach to implement complex neural network architecture on small FPGAs. Keep **low latency** and **good efficiency**, but also aim for **less power consumption**.



→ The first step is actually also relevant if you use a GPU. **Compact networks are much easier to understand and control.**

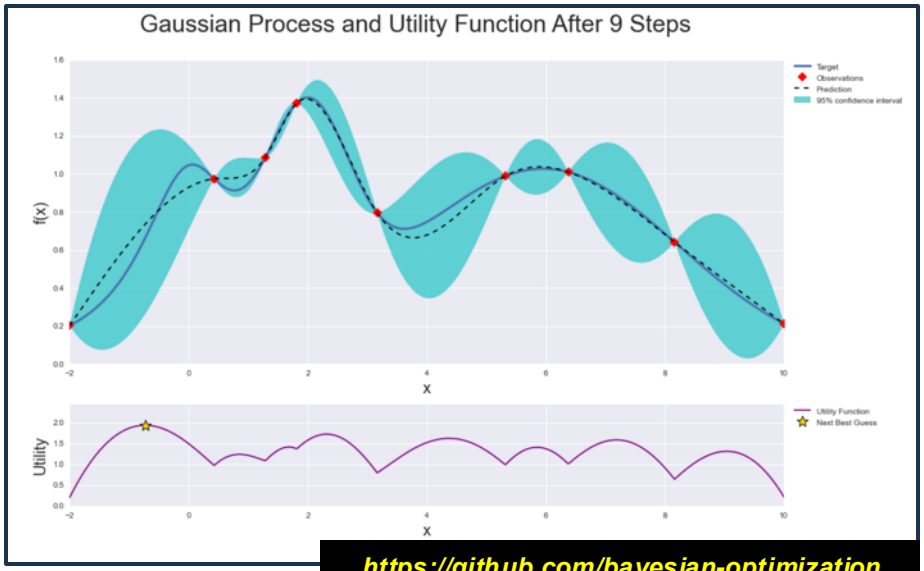
→ The second part is more related to FE sector, and was presented by **Geoffrey** yesterday

→ Simplifying network optimisation with bayesian approach

→ **Bayesian optimization** can help you to converge towards an optimal network topology

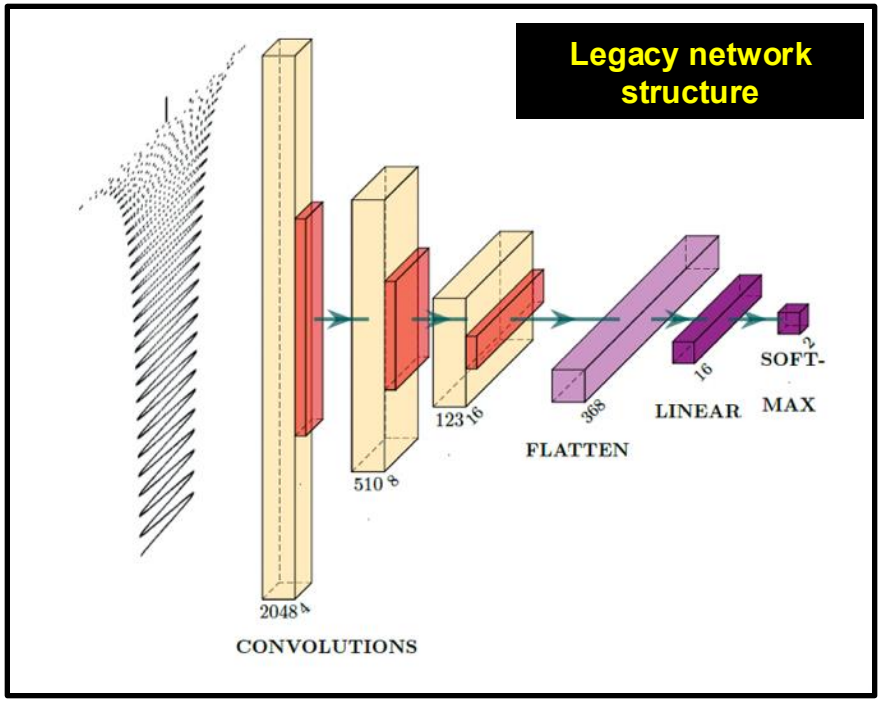
→ The optimization process tries to maximise a cost (*utility*) function. You can put the network efficiency there, but also sizing info.

→ You end up with a network combining **good efficiency**, but also optimized **inference complexity**



→ Applying this method to any type of network provides you with the **most online-friendly topology.** For an FPGA approach **it's mandatory** to have this done before going to the next stage. For a GPU-based, if you plan to go online, **it's worth the effort.**

→ Illustration on a simple use case:



→ Started from a legacy GW detection network (1701.00008v3).

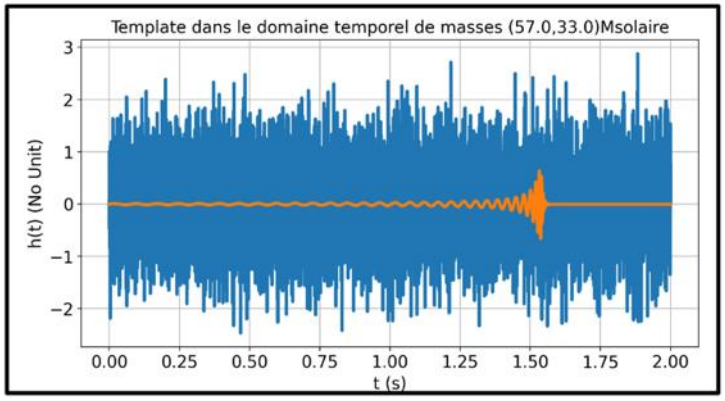
→ Very simple CNN encoder take 1s of the data strain as input (2048 points), a provides 2 values at the output:

- Compatibility of the data with signal
- Compatibility of the data with noise

→ There are much more elaborated architectures today, but this one is simple, efficient, and particularly attractive for FPGA implementation

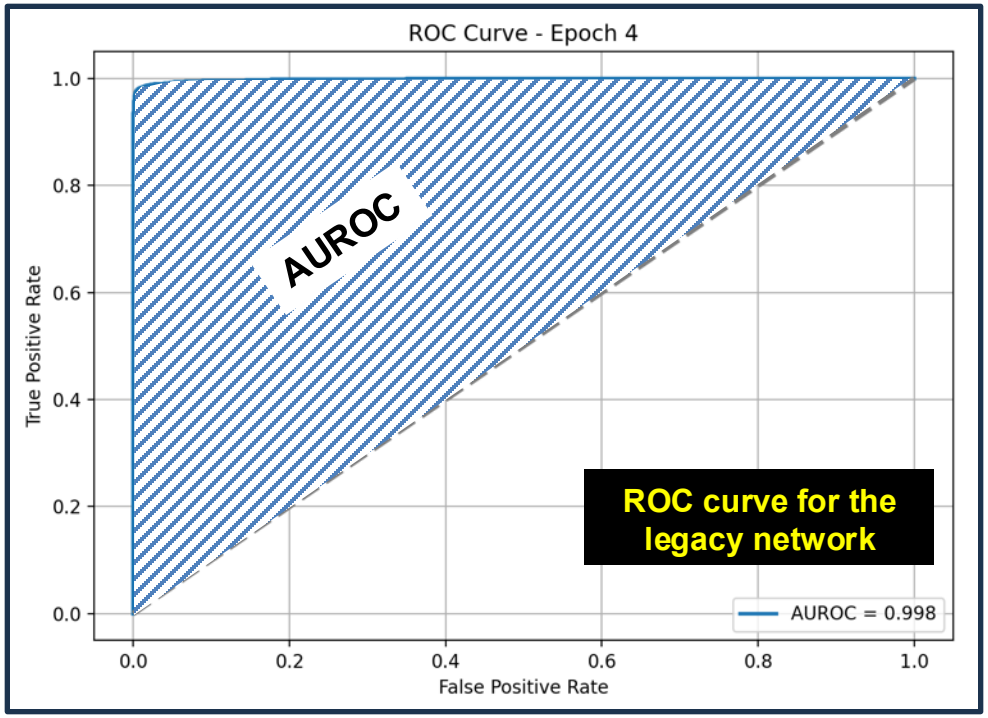
Number of parameters: 37094

Number of operations for 1 inference:
-> N additions : 1187600
-> N multiplications: 1209184



→ Make the network FPGA friendly:

→ Remove some **intensive steps**: batchnorm (*don't needed because data is whitened upstream*), use relu activations, remove the last softmax activation step (*signal cat over threshold is sufficient*)



→ Then try to adjust hyperparameters in order to maximise efficiency at low fake rate, and minimise of the number of operations and parameters (*relatively easy with a simple network*).

→ This work helps to refine hyperparams phase space, which can be fed to the bayesian optimizer to find optimal architectures

→ The optimizer gives a reduced set of good results, just have to pick up the best one

→ Finding the good cost function:

→ First looked at the normalized number of operations (*to a value we're aiming for, here 200000*) and AUROC value, and tries to find the best relative weighting between both

$$cost = \alpha \cdot log\left(\frac{n_{operations}}{200000}\right) + \beta \cdot AUROC$$

→ The choice of α and β coefficients depends on which param is the most important. Here we choose 1 and 2 respectively

→ Was working not too bad but networks found were not performing very well at very low fake rate (*which is important for us*). So instead of using AUROC, we used the efficiency at a false positive rate of 0.1%

$$cost = \alpha \cdot log\left(\frac{n_{operations}}{200000}\right) + \beta \cdot log(10 \cdot (1 - eff_{0.1\%}))$$

→ Efficiency criteria being very important for us, we used 1 and 4 for α and β coefficient values

→ FPGA-friendly encoder, structure and perf:

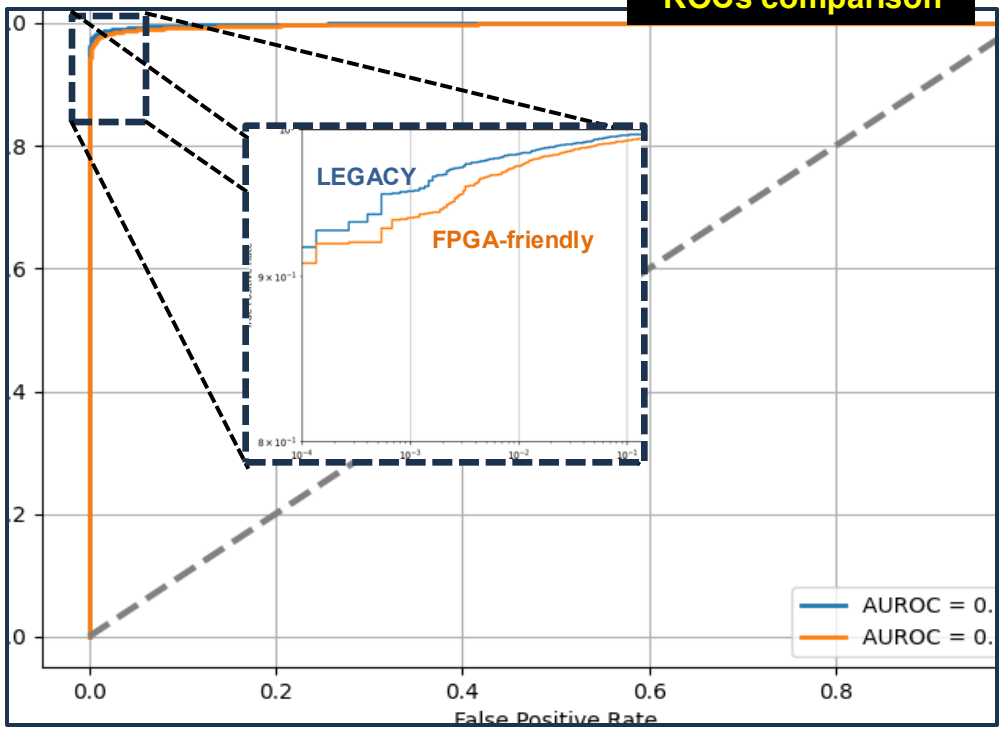
→ Best candidate from the optimizer:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 2048, 1)]	0
conv1d (Conv1D)	(None, 2041, 2)	18
max_pooling1d (MaxPooling1D)	(None, 510, 2)	0
activation (Activation)	(None, 510, 2)	0
conv1d_1 (Conv1D)	(None, 504, 7)	105
max_pooling1d_1 (MaxPooling1D)	(None, 126, 7)	0
activation_1 (Activation)	(None, 126, 7)	0
conv1d_2 (Conv1D)	(None, 122, 9)	324
max_pooling1d_2 (MaxPooling1D)	(None, 30, 9)	0
activation_2 (Activation)	(None, 30, 9)	0
conv1d_3 (Conv1D)	(None, 25, 8)	440
max_pooling1d_3 (MaxPooling1D)	(None, 6, 8)	0
activation_3 (Activation)	(None, 6, 8)	0
flatten (Flatten)	(None, 48)	0
dense (Dense)	(None, 2)	98

Total params: 985
Trainable params: 985
Non-trainable params: 0

Network structure

ROCs comparison



Number of parameters: 985 (37094)

Number of operations for 1 inference:
-> N additions : 131326 (1187600)
-> N multiplications: 131374 (1209184)

→ With the new cost function we end up with something better than the network found by hand

→ **Some personal toughts about bayesian optimization:**

→ Tools like **bayes_opt** are very simple to use

→ **Heuristic optimization beforehand is important to reduce the range of hyperparameters to be fed to the optimizer.** You should not find the best network here, but get an idea about where to look (*no free lunch*).

→ **Defining the cost function is, TMO, the most touchy step.** You need to order the requirements of the network you're looking for. So you need to know them properly.

→ Could become cumbersome for a complex network: parallelization could help a lot. Would be interesting to test Raphael's BO code to compare results and speed.

→ **ML opportunities in online GW ecosystem:**

→ **Real time suspension control** (*FPGA or RTPC*): eg use liquid network reinforcement learning (https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2019.00883/full). There is a clear need here, and some very promising works with more classic networks.

→ **FE signal fast denoising/shaping** (FPGA): need to evaluate what ML could bring here.

→ **Online signal processing** (RTPC): currently looking at transformers (*Gaspard & Quentin*). Here the FPGA approach is not necessary for the moment. But optimisation technique can certainly be exploited to improve networks robustness.

→ Conclusion

→ Machine learning can/will play a significant role in the next generation of interferometers online architecture. But one should keep things compact and robust. **Frugality is the key here.**

→ **A lot can/should be done to optimize the network upstream.** Bayesian optimization is an important ally there, but this will work only if you understand what you're doing (*eg to choose the right network*).

→ FPGA might be useful in some front end sectors. We need to investigate this more closely as we now have a tool to produce optimized RTL code. Liquid networks might finally come back into the game.