# Quantum machine learning for Monte Carlo events

*Julien Baglio*

[in collaboration with C. Bravo-Prieto, M. Cè, A. Francis, D. Grabowska, S. Carrazza, arXiv:2110.06933]

QUANTUM TECHNOLOGY INITIATIVE

QCD Seminar, CERN TH, 19/11/2021

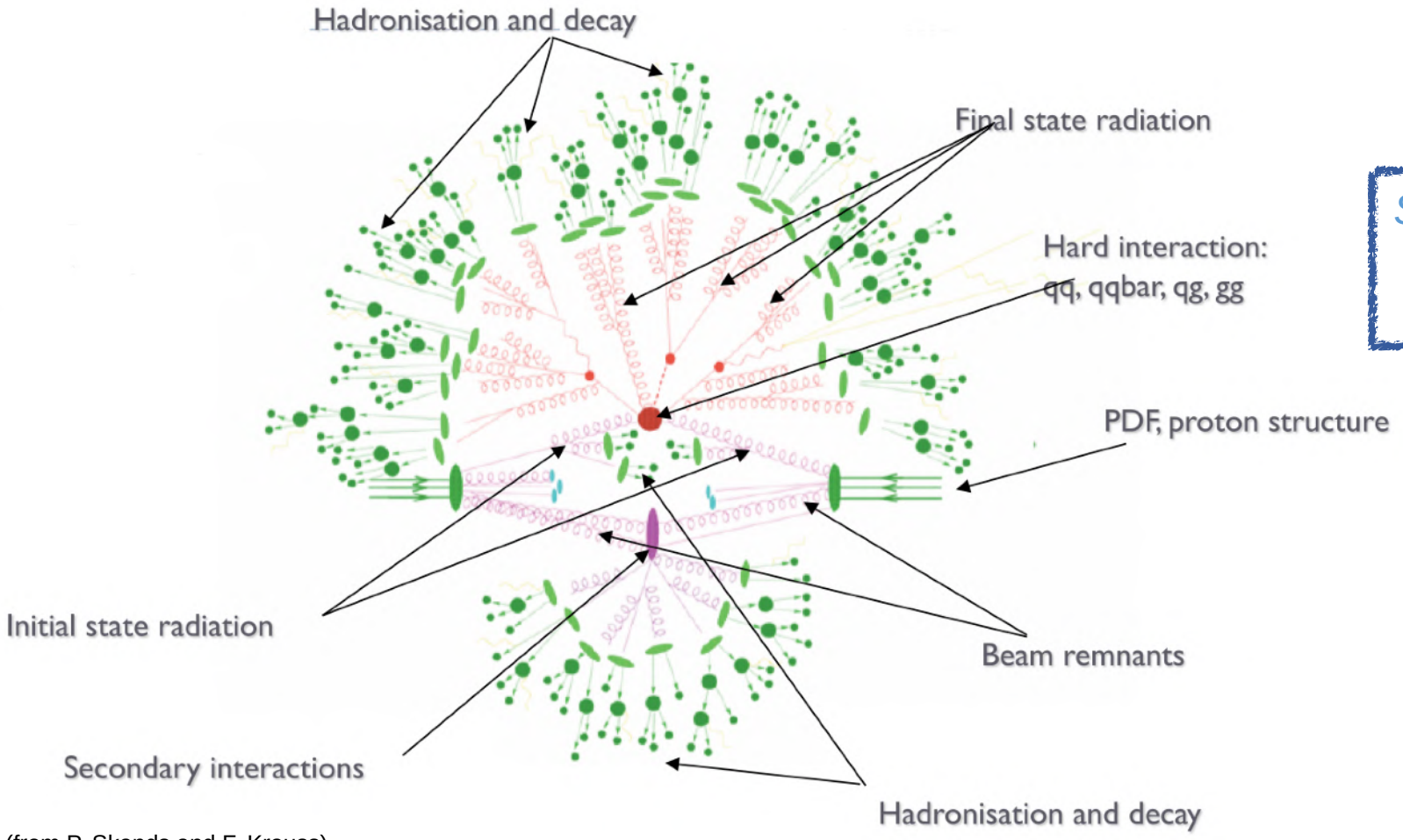
# Summary

1. Introduction
2. Quantum generator for generative adversarial networks
3. Quantum computing in a nutshell
4. Results for $pp \to t\bar{t}$ leading-order event generation
5. Conclusions

# Introduction

# Context: hadronic collisions at the Large Hadron Collider

**LHC produces $\mathcal{O}(10^9)$ proton collisions per second: huge complex environment!**




*Simulation of the full event is very intensive and requires lots of computing power*

(from P. Skands and F. Krauss)

# Toolchain for event generation

1. Hard matrix element generation (Les Houches event)
2. Parton shower to account for soft radiation
3. Hadronization effects, pile-up, etc.
4. Detector simulation (fast with Delphes, or full with dedicated tool e.g. GEANT4)
5. Extract useful physical observables from Root files produced after step 4

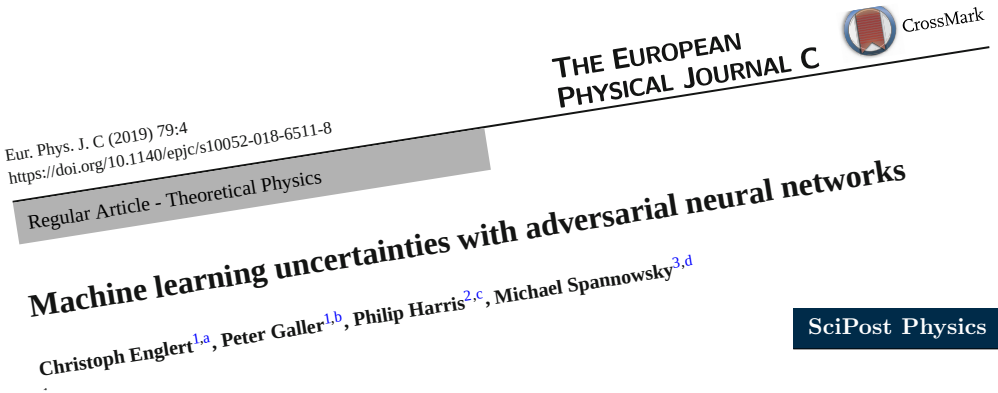
← ***With $\mathcal{O}(1M)$ events, a bottleneck***

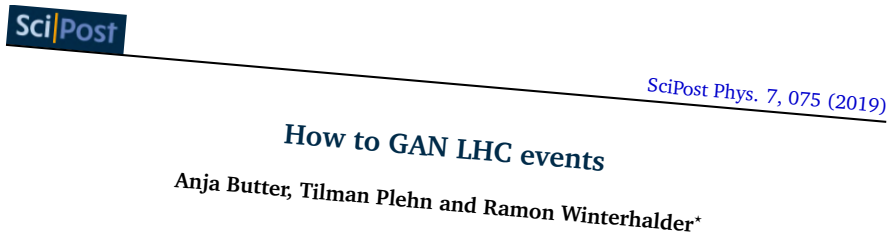
***All steps can be computer-intensive: how to accelerate the process?***

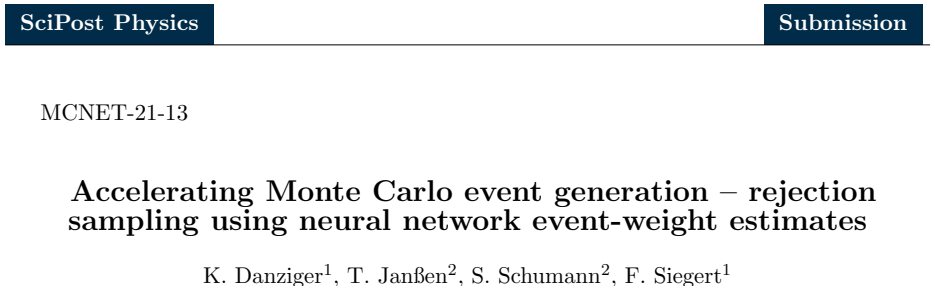
# The machine learning approach to event generation

Since 2018, many papers have approached event generation with machine learning

[see a review in A. Nutter, T. Plehn, arXiv:2008.08558]



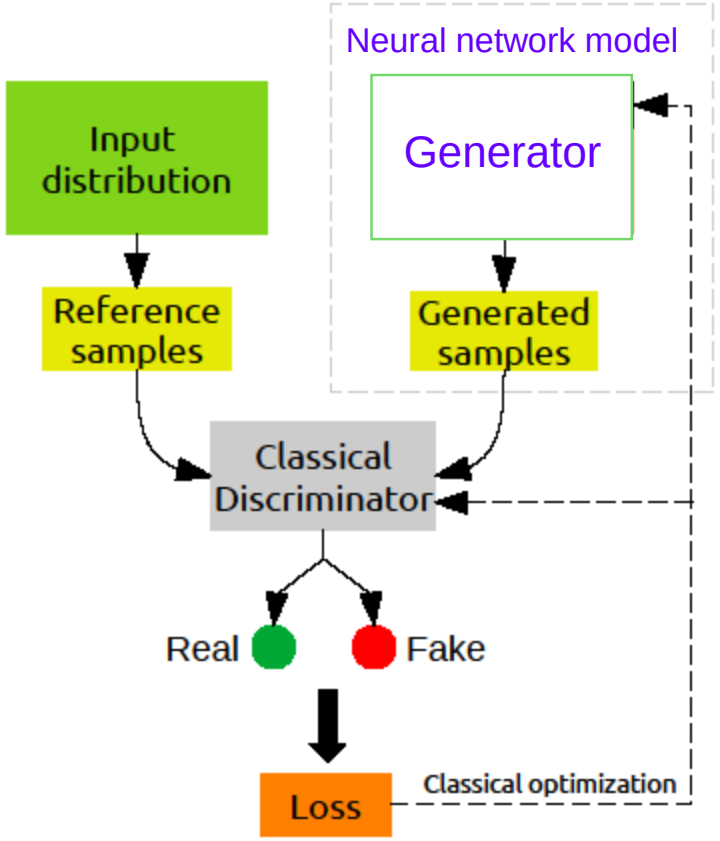




***Main idea: train with a small dataset, use machine learning networks to learn the underlying distribution and generate for free a much larger dataset***

# What is a generative adversarial network (GAN)?

**Two networks competing**: **generator** produces fake data, **discriminator** distinguishes between real (training) input data and fake (produced by the generator) data

⇒ *game theory where the generator learns the underlying (input) distribution*




***Art forger analogy***

**Generator (art forger):** Try creating fake paintings that look authentic

**Discriminator (art historian):** Check paintings and try to catch the forgery

**Training:** "Catch me if you can" game between the art forger and the art historian

**Success:** Painted forgeries are so good that the art historian has at most a 50% guess ratio ⇒ The forger creates new work of the same style

# Goal: proof-of-concept of quantum GAN

***Can we use quantum technologies to create an enhanced GAN?***

- Can a quantum GAN (qGAN) be faster than a classical GAN? ⇔ **quantum supremacy?**
- Would a qGAN have less parameters?
- Would energy consumption be reduced on a quantum architecture?

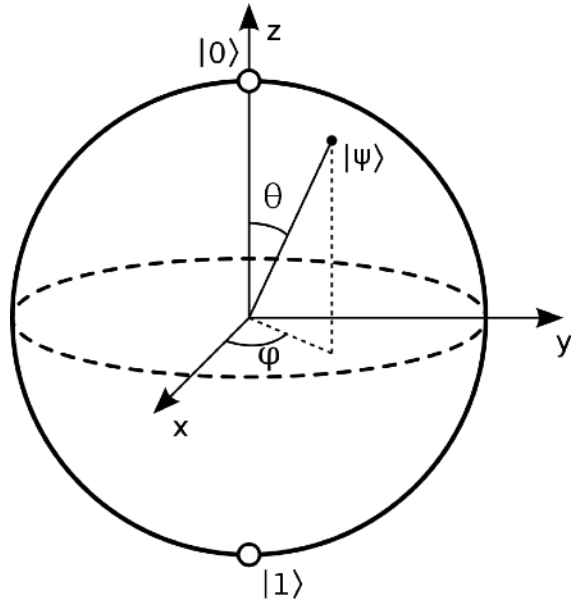
**Need first to assess whether we can create a qGAN for event generation: proof-of-concept!**

**Can we also improve the state-of-the-art qGAN that already exist for other purposes?**

# Quantum computing in a nutshell

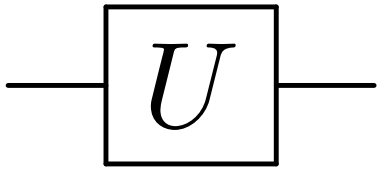
# Bits and qubits (digital quantum computer)

From 0 and 1 to $|0\rangle$ and $|1\rangle$: **one-qubit state** $|\psi\rangle$ as a **superposition**

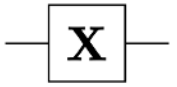



More qubits: e.g. bit 5 can be represented by a three-qubit state $|101\rangle$

**Boolean operator → Unitary matrices (gates)**




*Some commons one-qubit gates:*

Quantum NOT gate 


$$X|0\rangle = |1\rangle, X|1\rangle = |0\rangle$$

Pauli Z gate 

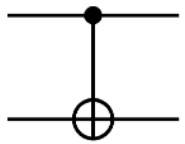

$$X|0\rangle = |0\rangle, X|1\rangle = -|1\rangle$$

**Hadamard gate** 


$$X|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$$

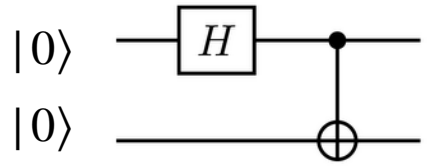
# Gates and circuits

*Important two-qubit gate: controlled-NOT (CNOT)*

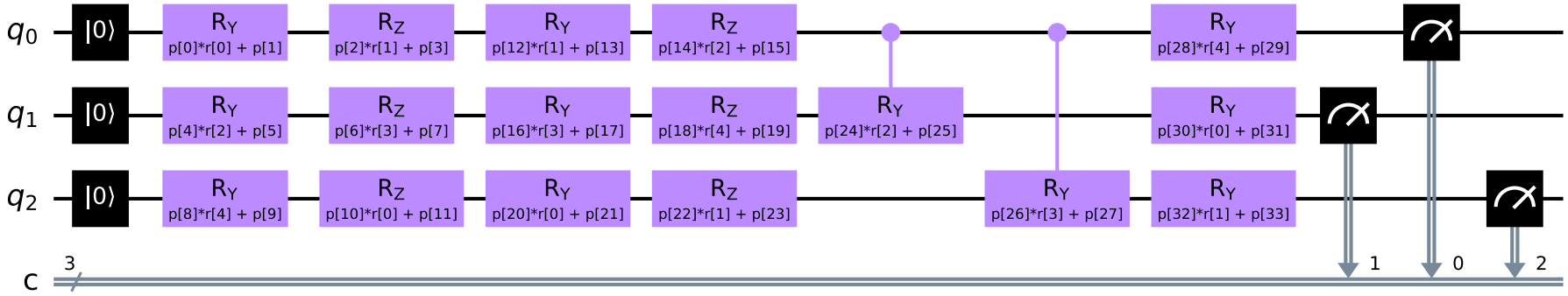


- Qubit 1 is $|0\rangle$: do nothing on qubit 2
- Qubit 1 is $|1\rangle$: apply one-qubit gate NOT on qubit 2

***Create entanglement***




$= \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$: Bell state

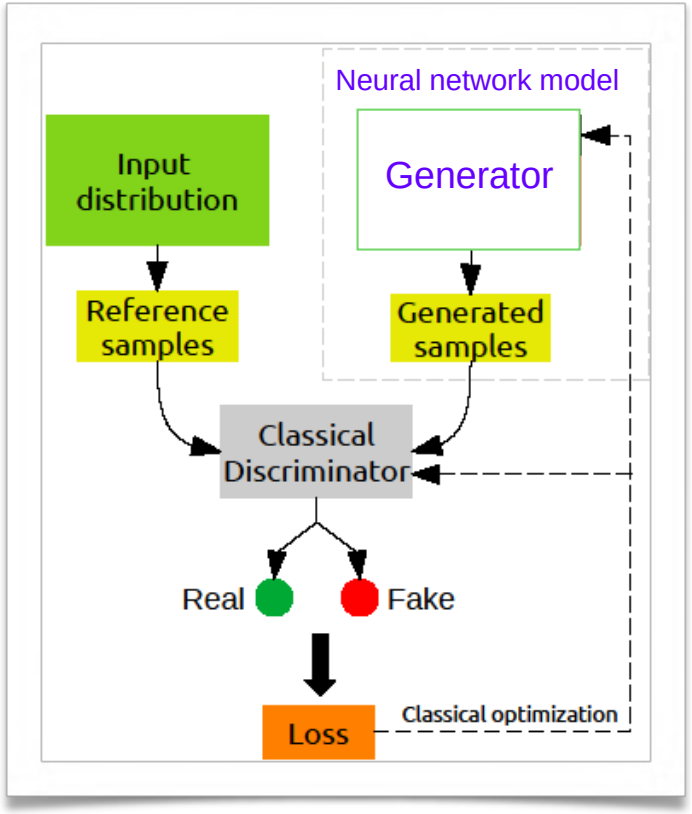



- Important remark: circuits are unitary ⇒ **quantum algorithms are invertible**
- At the end of a circuit, measurement is performed: **build expectation values**

# Quantum generator for generative adversarial networks

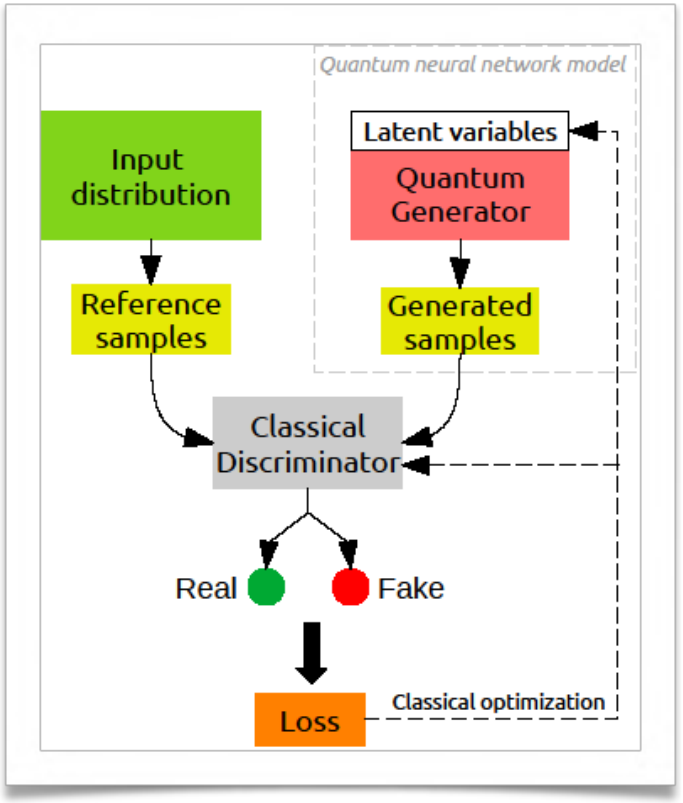
# Hybrid approach for a qGAN

**Classical setup:**



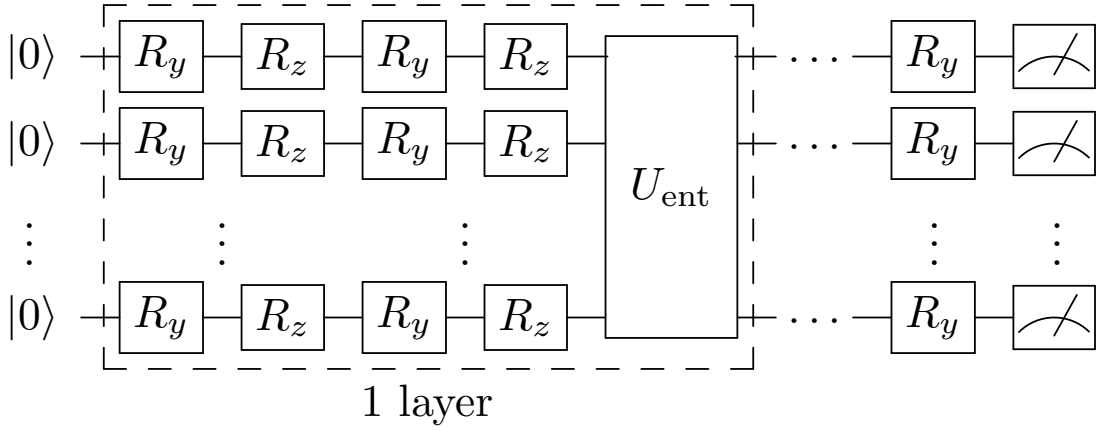

*Optimization of the calculation:*
*Only the generator becomes quantum*

**Hybrid quantum-classical setup:**



# Model for the quantum layer: styled qGAN

**Quantum network:** a series of quantum layers with rotation gates and entanglement operators




1 observable = 1 qubit

$$R_y = \exp\left(-i\frac{\theta}{2}\sigma_y\right),\ R_z = \exp\left(-i\frac{\theta}{2}\sigma_z\right)$$

$U_{\text{ent}}$ set of controlled rotations for entanglement

***Novelty of our network:***
the noise is inserted in **every gate and not only in the initial quantum state**

← *data reuploading*

Circuit implemented in Python with qibo [S. Efthymiou et al., arXiv:2009.01845] for quantum simulation and qiskit [G. Aleksandrowicz et al., code on Zenodo] for hardware deployment on IBM Q

# Latent dimension and encoding of the observables

**Latent dimension:** dimension of the noise vector $\xi^{(j)}$ inserted in the circuit

*a key **hyperparameter***

**Parameter tuning:** each gate with tuneable parameters $\phi_g^{(j)}$

*here comes the **training***

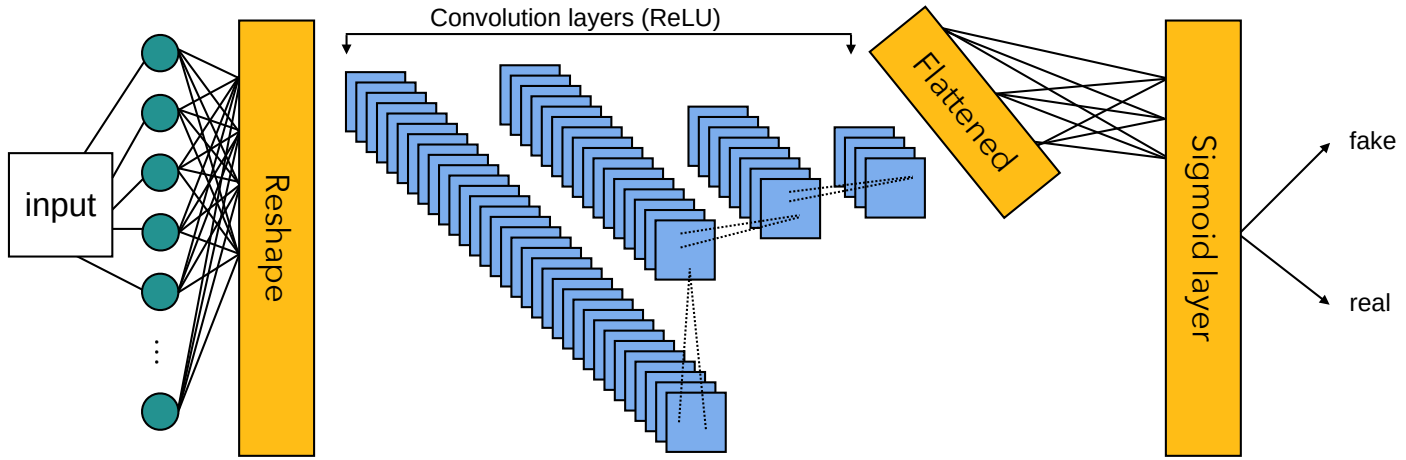
$R^i_{y,z}(\phi_g^{(i)}, \xi^{(j)}) = R_{y,z}(\phi_g^i \xi^j + \phi_g^{i+1})$, simple linear parameterization of angle $\theta$

**Measure expectation values:** perform N measurements (shots) of the final quantum state of the circuit, building $\overrightarrow{x}_{\text{fake}} = -[\langle\sigma_z^1\rangle, \langle\sigma_z^2\rangle, \ldots, \langle\sigma_n^1\rangle]$

# Classical discriminator network

*Use deep **convolutional neural network** with 7 main layers*

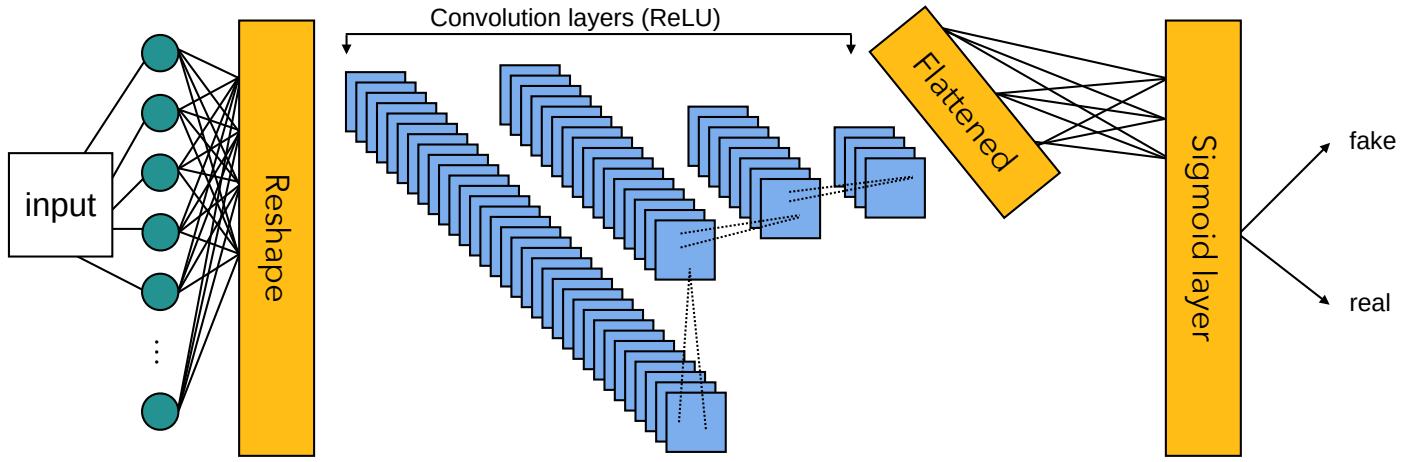
- **1 input** layer: Take *n* inputs (data), dense nodes fully connected to get an output of high dimensionality
- **4 convolution** layers with 64, 32, 16, 8 *filters* (convolution matrices), using a **ReLU activation function**
- **1 flatten** layer: One-dimensional fully connected layer of data
- **1 activation** layer: output the binary classification using a **sigmoid activation function**



# Classical discriminator network

*Use deep **convolutional neural network** with 7 main layers*

- **1 input** layer: Take *n* inputs (data), dense nodes fully connected to get an output of high dimensionality
- **4 convolution** layers with 64, 32, 16, 8 *filters* (convolution matrices), using a **ReLU activation function**
- **1 flatten** layer: One-dimensional fully connected layer of data
- **1 activation** layer: output the binary classification using a **sigmoid activation function**




$$\text{ReLU: } f(x) = \begin{cases} x \text{ if } x \geq 0 \\ \alpha x \text{ if } x < 0 \end{cases}, \alpha = 0.2 \qquad \text{Sigmoid: } f(x) = \frac{1}{1 + e^{-x}}$$

# Training procedure: Nash equilibrium

**Training:** Adapt alternatively the quantum generator $G(\phi_g, z)$ and the classical discriminator $D(\phi_d, x)$ [ $z$ and $x$ input samples for the two networks]

**Mathematical tool:** binary cross-entropy for the loss functions

- Generator loss function:
$$\mathscr{L}_G(\phi_g, \phi_d) = -\mathbb{E}_{z\sim p_{\text{noise}}(z)}[\log(D(\phi_d, G(\phi_g, z))]$$
- Discriminator loss function:
$$\mathscr{L}_D(\phi_g, \phi_d) = \mathbb{E}_{z\sim p_{\text{real}}(z)}[\log(D(\phi_d, z)] + \mathbb{E}_{z\sim p_{\text{noise}}(z)}[\log(1 - D(\phi_d, G(\phi_g, z))]$$

# Training procedure: Nash equilibrium

**Training:** Adapt alternatively the quantum generator $G(\phi_g, z)$ and the classical discriminator $D(\phi_d, x)$ [ $z$ and $x$ input samples for the two networks]

**Mathematical tool:** binary cross-entropy for the loss functions

- Generator loss function:
$$\mathscr{L}_G(\phi_g, \phi_d) = -\mathbb{E}_{z\sim p_{\text{noise}}(z)}[\log(D(\phi_d, G(\phi_g, z))]$$
- Discriminator loss function:
$$\mathscr{L}_D(\phi_g, \phi_d) = \mathbb{E}_{z\sim p_{\text{real}}(z)}[\log(D(\phi_d, z)] + \mathbb{E}_{z\sim p_{\text{noise}}(z)}[\log(1 - D(\phi_d, G(\phi_g, z))]$$

**Game theory:** min-max two-player game to reach Nash equilibrium

$$\min_{\phi_g} \mathscr{L}_G(\phi_g, \phi_d) \text{ and } \max_{\phi_d} \mathscr{L}(\phi_g, \phi_d)$$

# Training procedure: gradient descent

***Find the Nash equilibrium***: update the parameters $(\phi_g, \phi_d)$ with **gradient descent**

$$\phi^i_{g,d} \to \phi^i_{g,d} + \delta\phi^i_{g,d},\ \delta\phi^i_{g,d} = -\eta \mathrm{gr}_{\phi^i_{g,d}} \text{ with } \mathrm{gr}_{\phi^i_{g,d}} = \frac{\partial \mathscr{L}}{\partial \phi^i_{g,d}}$$

*learning rate: a key **hyperparameter***

For the training of the styled qGAN: **ADADELTA** stochastic gradient descent method with an initial learning rate 0.5 (generator) and 0.1 (discriminator) [M. Zeiler, arXiv:1212.5701]

# How to compare generated samples with real samples?

**Use the Kullback-Leibler divergence**: $D_{\mathrm{KL}}(P\,||\,Q) = \sum_x P(x)\log\left(\frac{P(x)}{Q(x)}\right)$

***Main idea:*** with $Q$ reference distribution, $D_{\mathrm{KL}} = 0 \Leftrightarrow P \equiv Q$

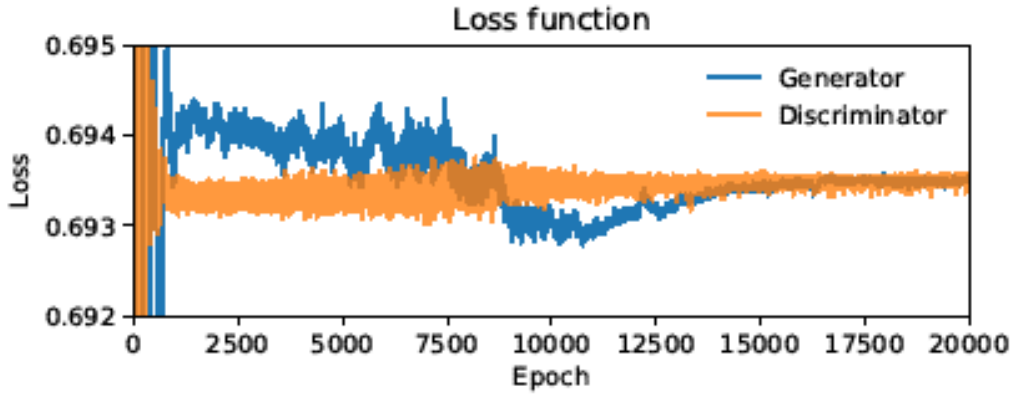
The KL divergence is the difference between the information entropy of $P$ and the cross entropy of $P$ with $Q$

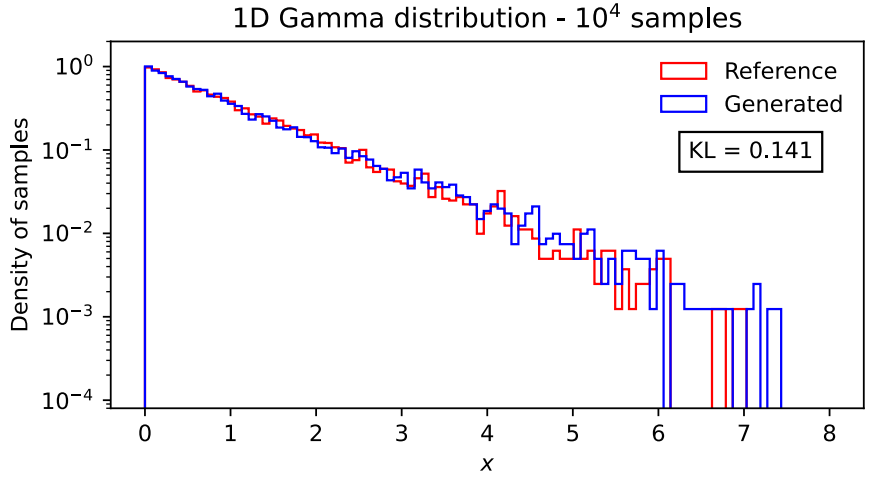
# styled qGAN validation: 1D and 3D-gaussian examples

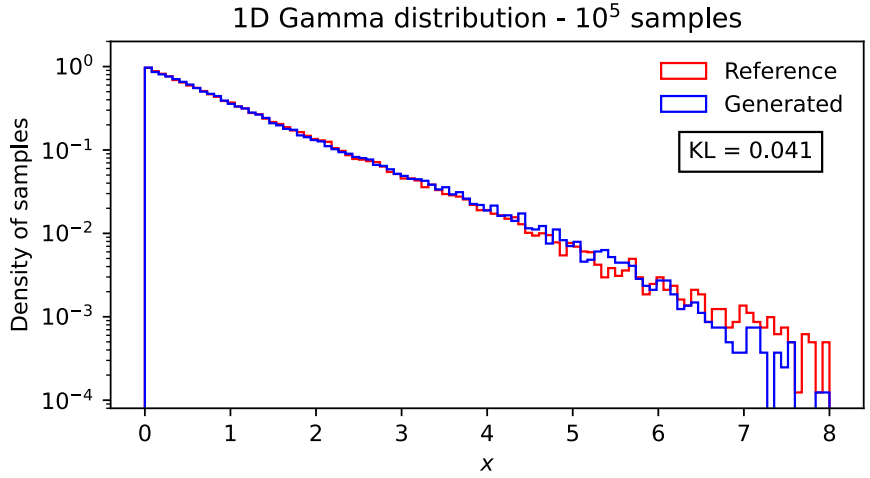
**Assessing the validity of the approach:** train and test on known distribution

With one qubit, **one layer**, using 100 bins: 1D Gamma function $p_\gamma(x,\alpha,\beta) = x^{\alpha-1}\frac{e^{-x/\beta}}{\beta^\alpha \Gamma(\alpha)}, \alpha = \beta = 1$

- Pre-processing of the data to fit samples in [-1;1]
- Train on $10^4$ samples until convergence is reached, perform hyperparameter optimization
- Use generator to generate $10^4$ and $10^5$ samples to demonstrate reproducibility and data augmentation



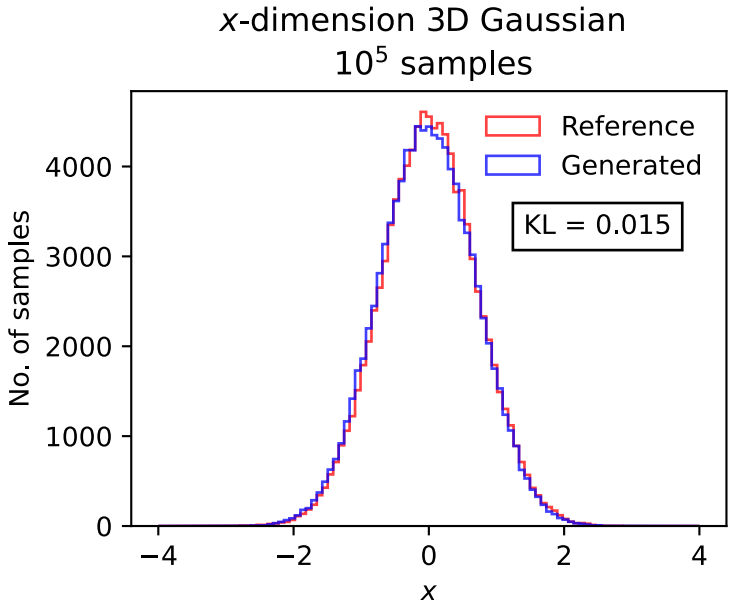



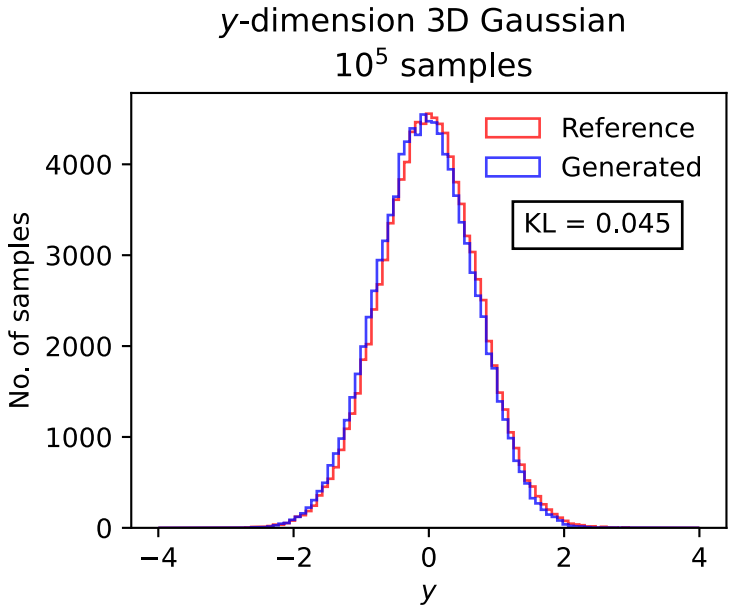
# styled qGAN validation: 1D and 3D-gaussian examples

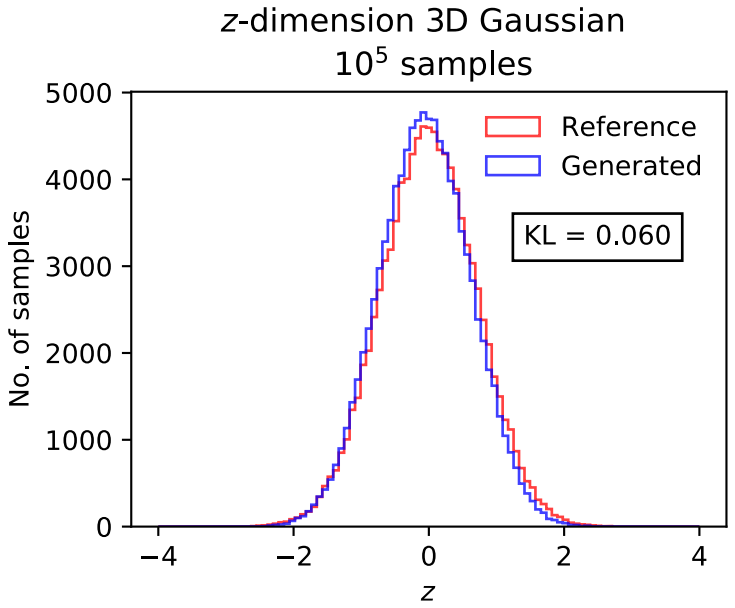
**Test whether the qGAN captures correlations:** train on 3D Gaussian function

$$p(\vec{x}) \propto \exp\left[-\frac{1}{2}(\vec{x}-\Sigma)^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right], \text{ with } \Sigma = \begin{pmatrix} 0.5 & 0.1 & 0.25 \\ 0.1 & 0.5 & 0.1 \\ 0.25 & 0.1 & 0.5 \end{pmatrix}, \vec{\mu} = (0,0,0)$$

## Now entanglement is necessary



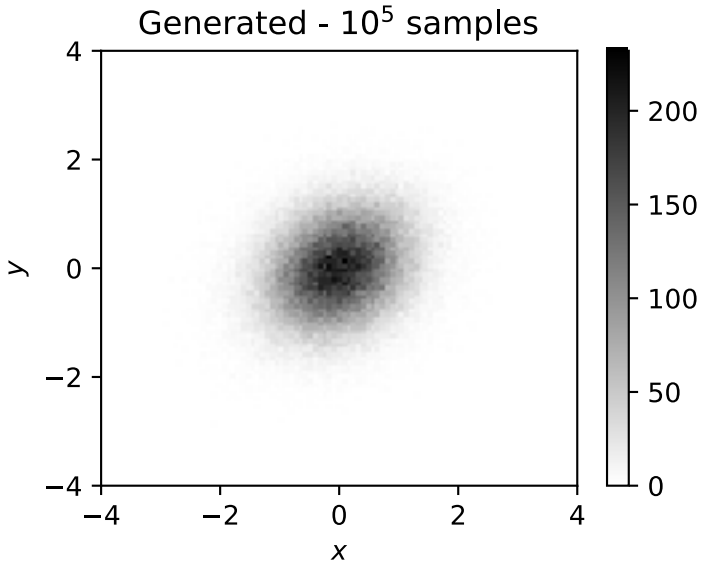



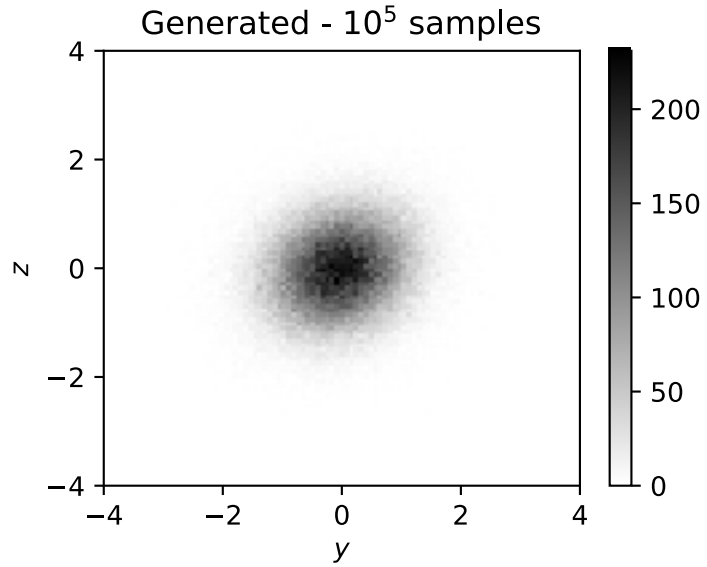
# styled qGAN validation: 1D and 3D-gaussian examples

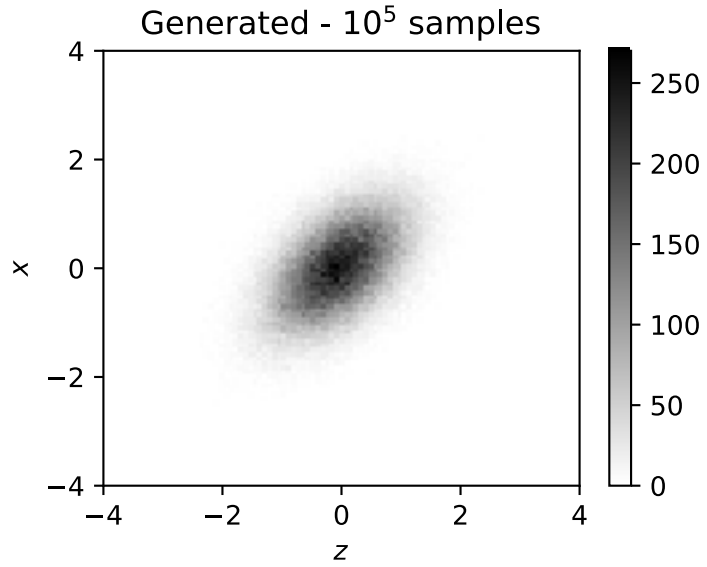
**Test whether the qGAN captures correlations:** train on 3D Gaussian function

$$p(\vec{x}) \propto \exp\left[-\frac{1}{2}(\vec{x}-\Sigma)^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right], \text{ with } \Sigma = \begin{pmatrix} 0.5 & 0.1 & 0.25 \\ 0.1 & 0.5 & 0.1 \\ 0.25 & 0.1 & 0.5 \end{pmatrix}, \vec{\mu} = (0,0,0)$$

## Now entanglement is necessary



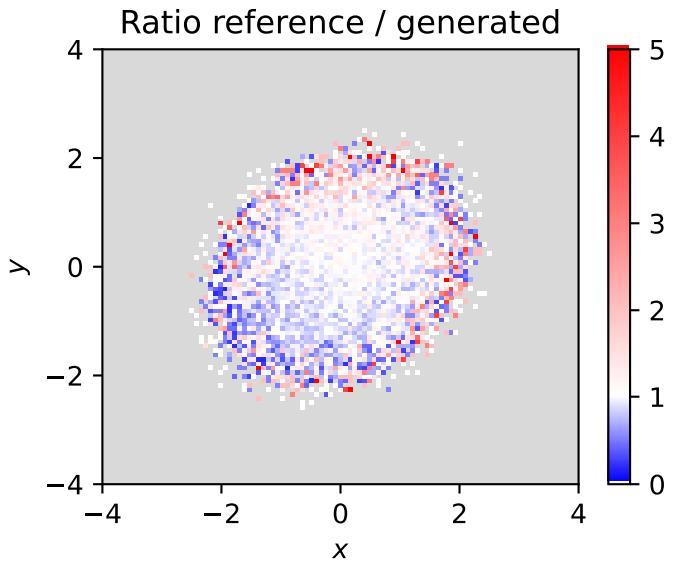



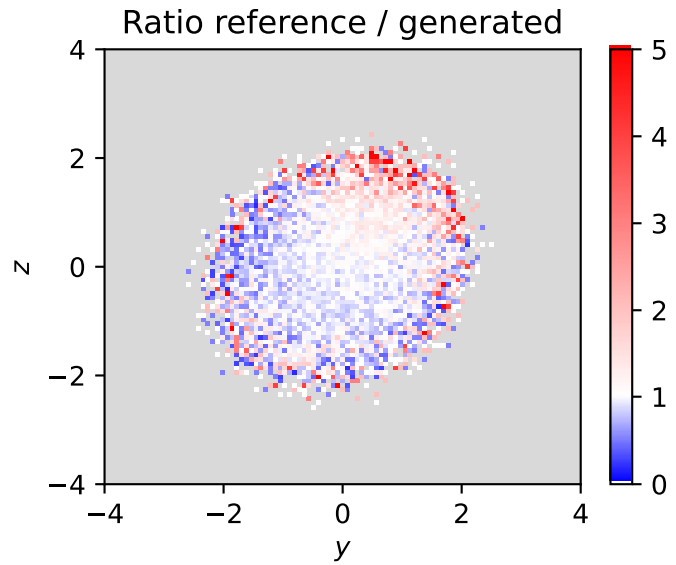
# styled qGAN validation: 1D and 3D-gaussian examples

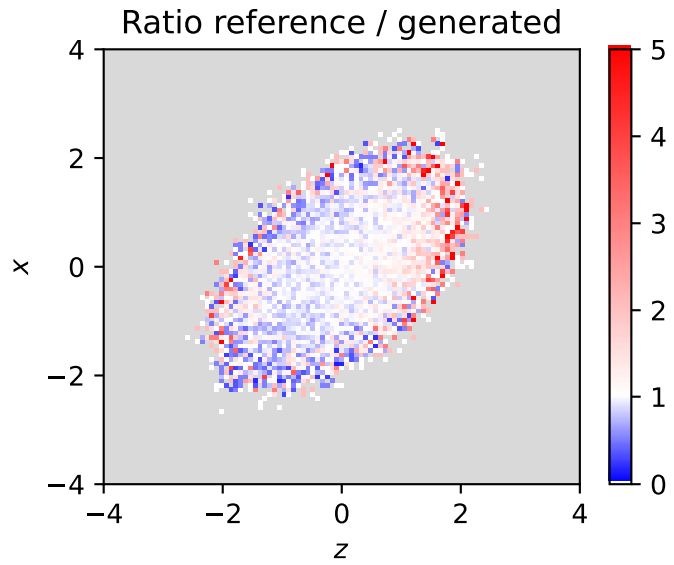
**Test whether the qGAN captures correlations:** train on 3D Gaussian function

$$p(\vec{x}) \propto \exp\left[-\frac{1}{2}(\vec{x}-\Sigma)^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right], \text{ with } \Sigma = \begin{pmatrix} 0.5 & 0.1 & 0.25 \\ 0.1 & 0.5 & 0.1 \\ 0.25 & 0.1 & 0.5 \end{pmatrix}, \vec{\mu} = (0,0,0)$$

## Now entanglement is necessary



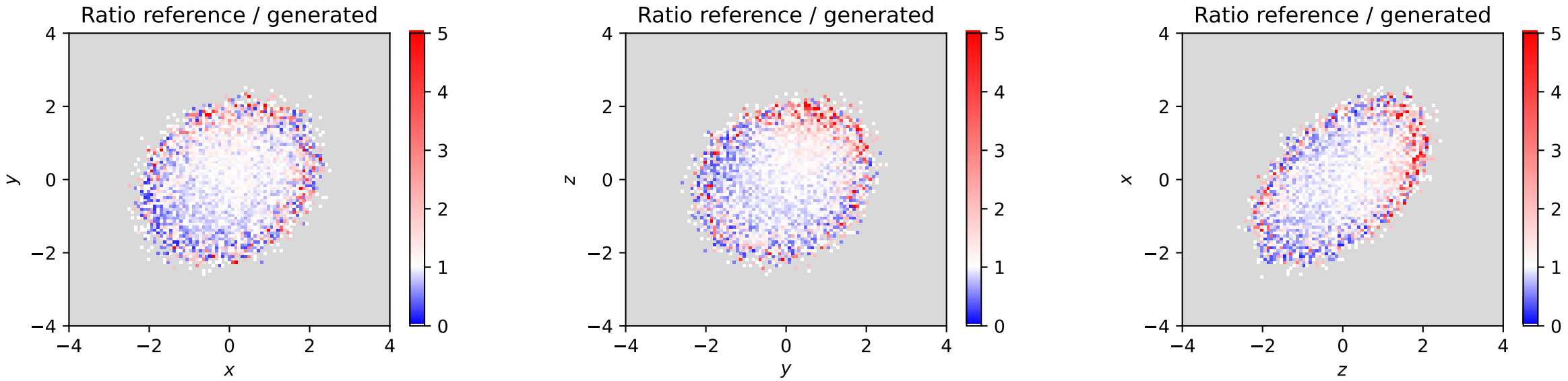



# styled qGAN validation: 1D and 3D-gaussian examples

**Test whether the qGAN captures correlations:** train on 3D Gaussian function

$$p(\vec{x}) \propto \exp\left[-\frac{1}{2}(\vec{x}-\Sigma)^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right], \text{ with } \Sigma = \begin{pmatrix} 0.5 & 0.1 & 0.25 \\ 0.1 & 0.5 & 0.1 \\ 0.25 & 0.1 & 0.5 \end{pmatrix}, \vec{\mu} = (0,0,0)$$

## Now entanglement is necessary

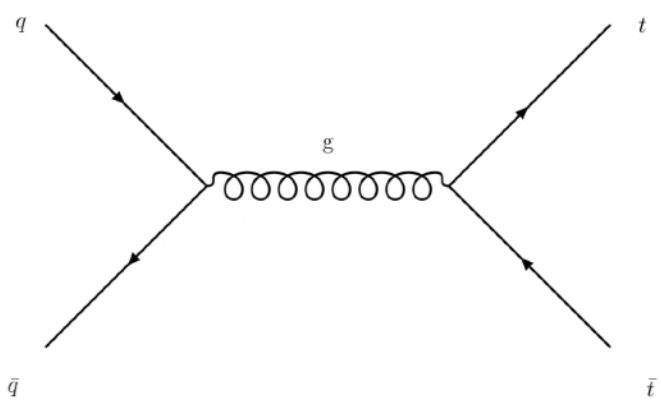


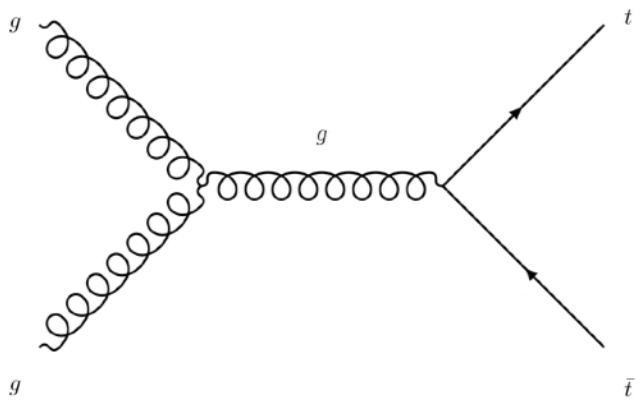

***Correlations are well captured!***

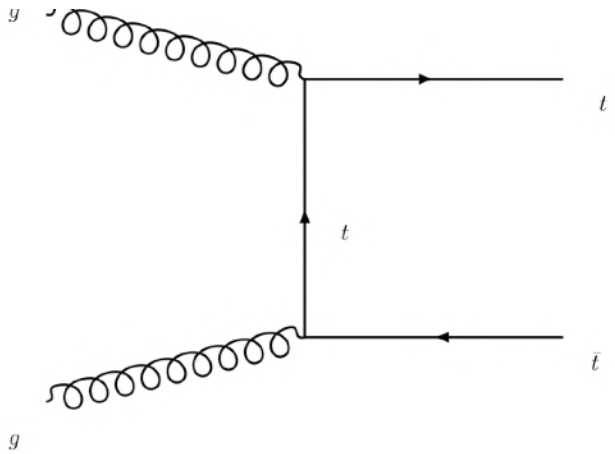
# Results for $pp \to t\bar{t}$ leading-order event generation

# Simulation with actual LHC data

**Testing the styled qGAN with real data: test-case with leading-order production $pp \to t\bar{t}$**



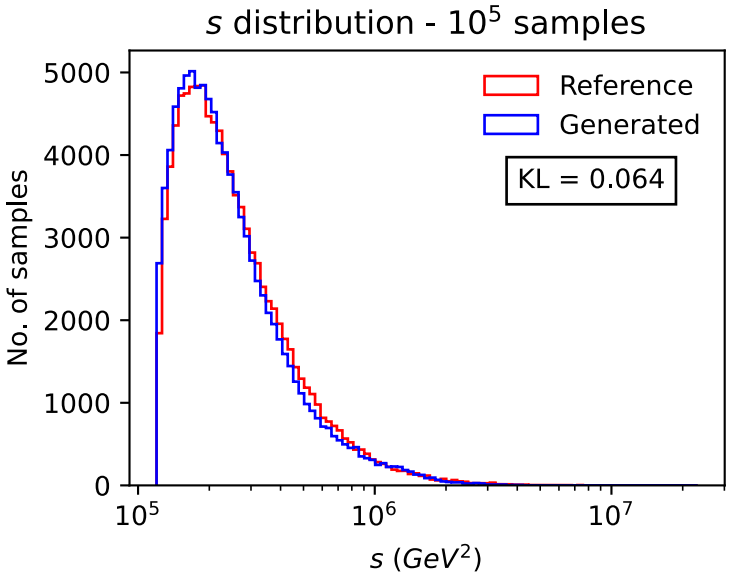

Training and reference samples generated with MadGraph5_aMC@NLO [Alwall et al., JHEP 07 (2014) 079]

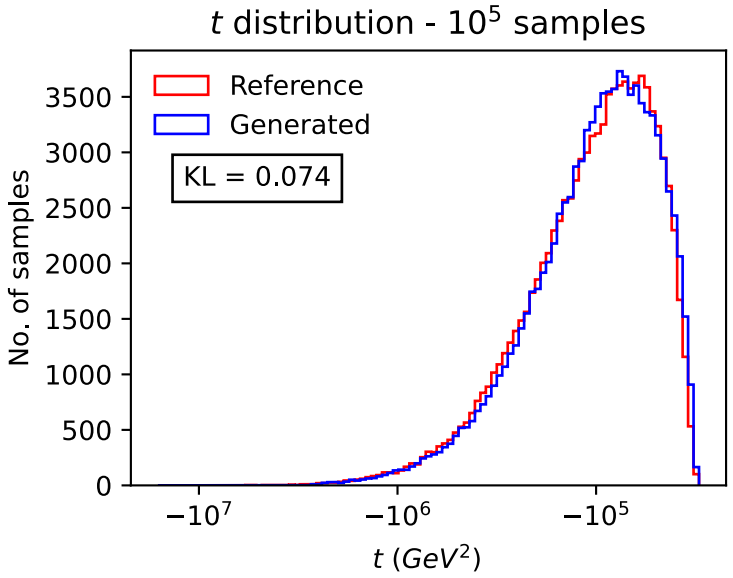
***LHC at 13 TeV set-up, training set of $10^4$ samples, Mandelstam variables $(s, t)$ and rapidity $y$***

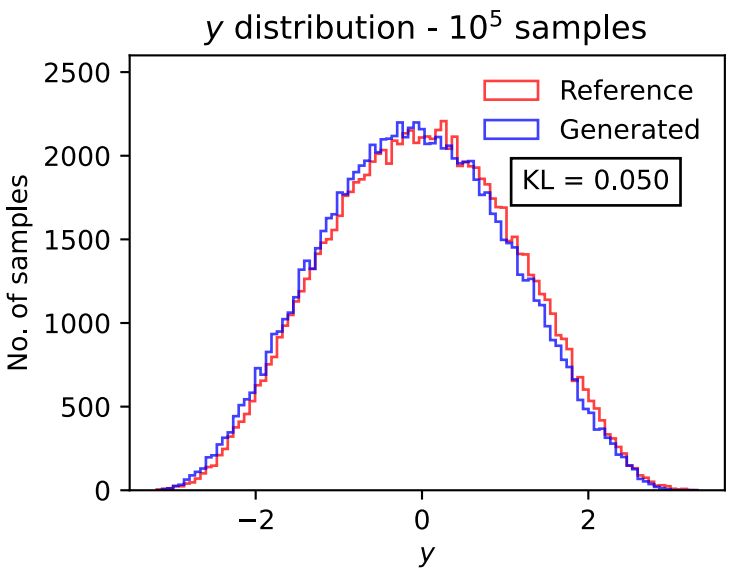
# Results of noiseless simulations

**After classical training, assessment of the potential performance with noiseless simulations**

*Simulation performed on a classical computer with a quantum simulator in the qibo framework*



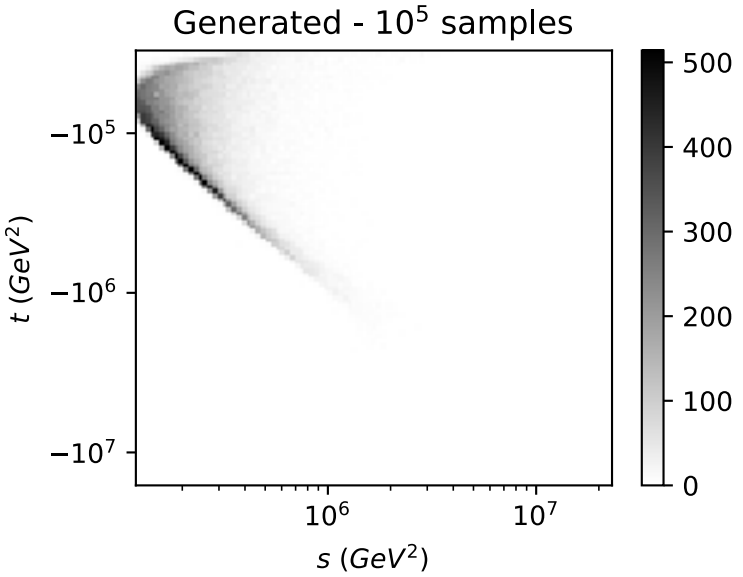



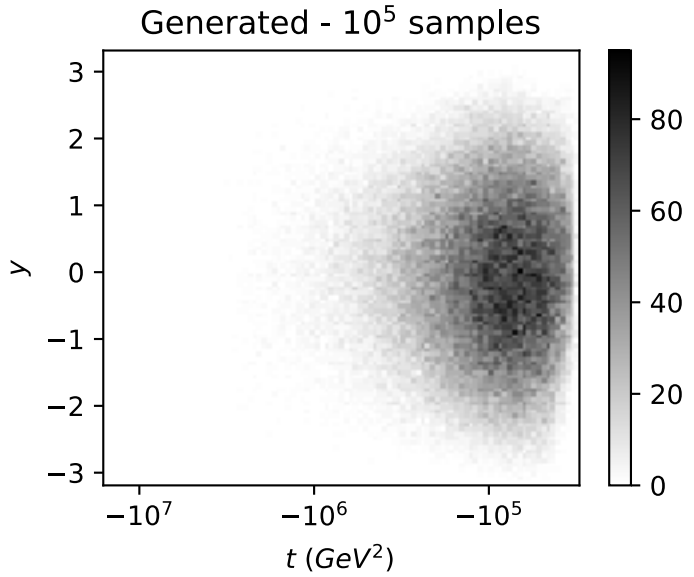



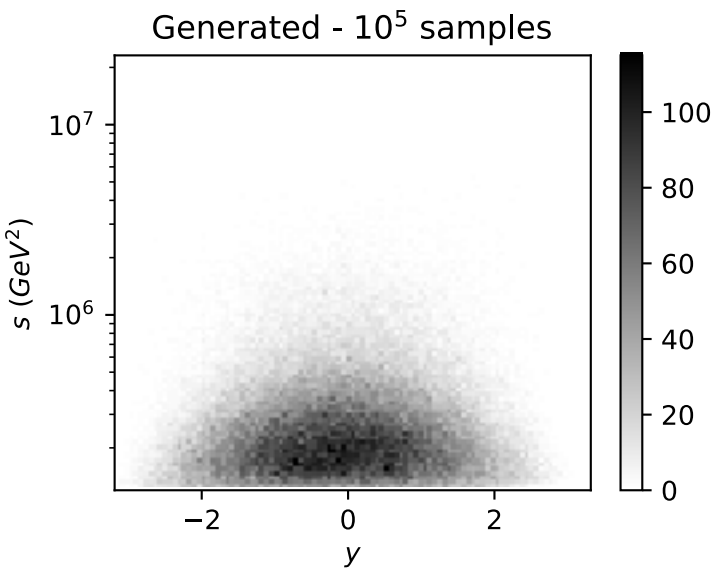
# Results of noiseless simulations

**After classical training, assessment of the potential performance with noiseless simulations**

*Simulation performed on a classical computer with a quantum simulator in the qibo framework*



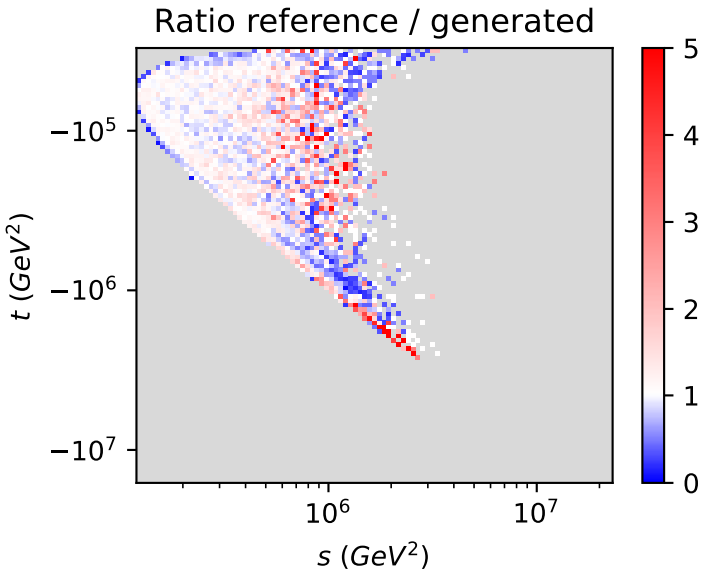



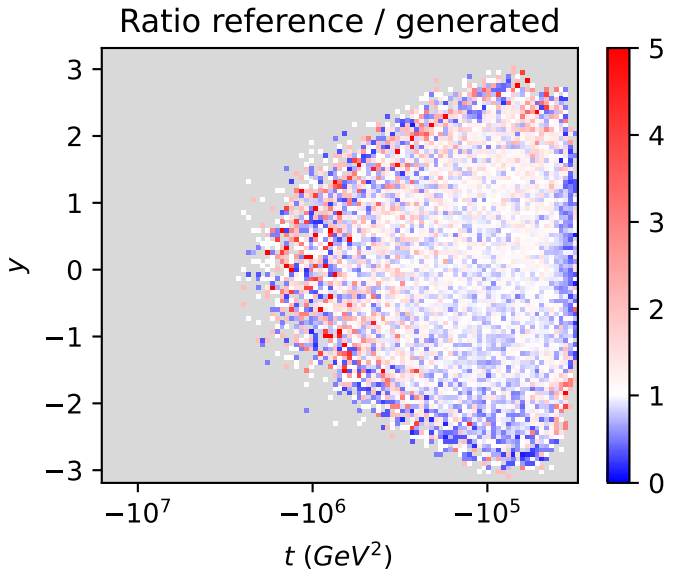



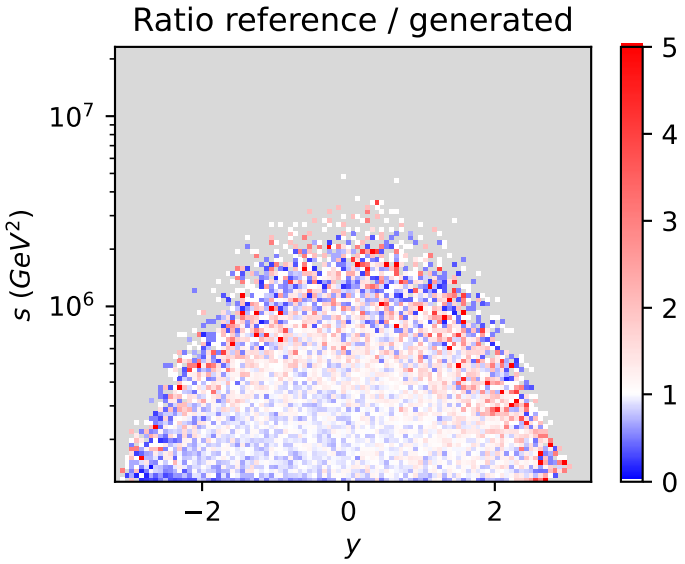
# Results of noiseless simulations

**After classical training, assessment of the potential performance with noiseless simulations**

*Simulation performed on a classical computer with a quantum simulator in the qibo framework*



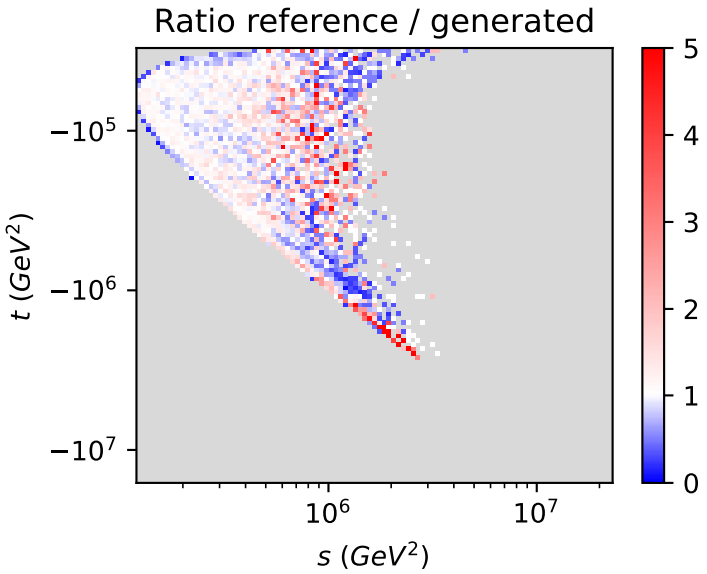



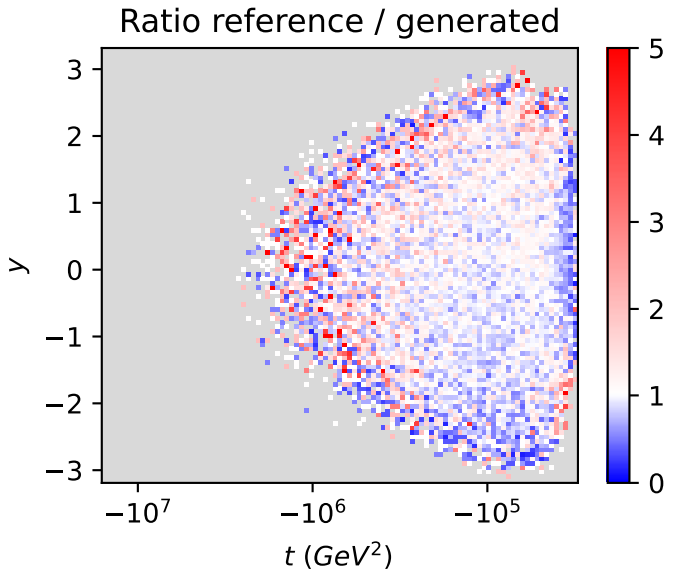



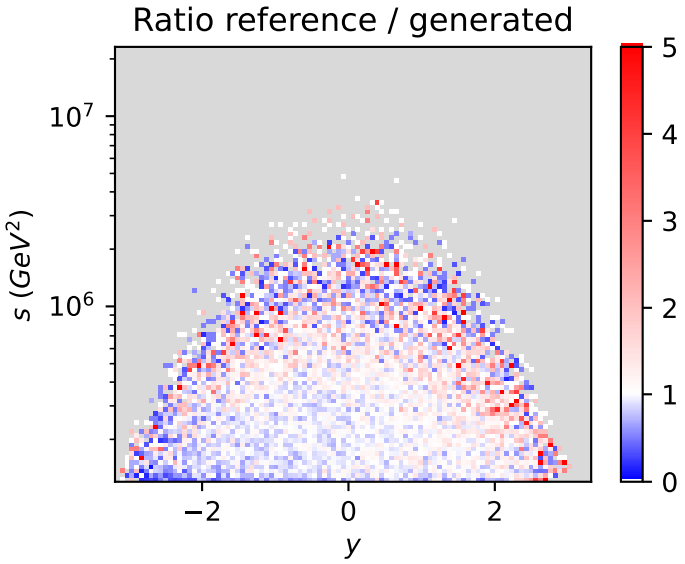
# Results of noiseless simulations

**After classical training, assessment of the potential performance with noiseless simulations**

*Simulation performed on a classical computer with a quantum simulator in the qibo framework*



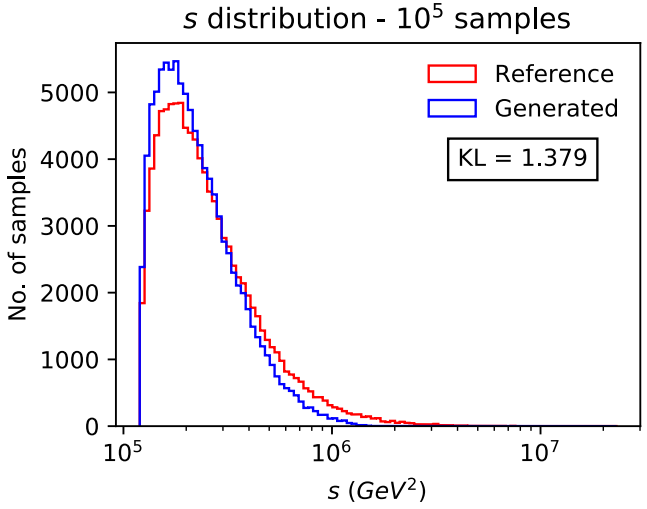

***Remarkable low KL divergences with data augmentation!***
***Are these nice results maintained on real hardware ?***

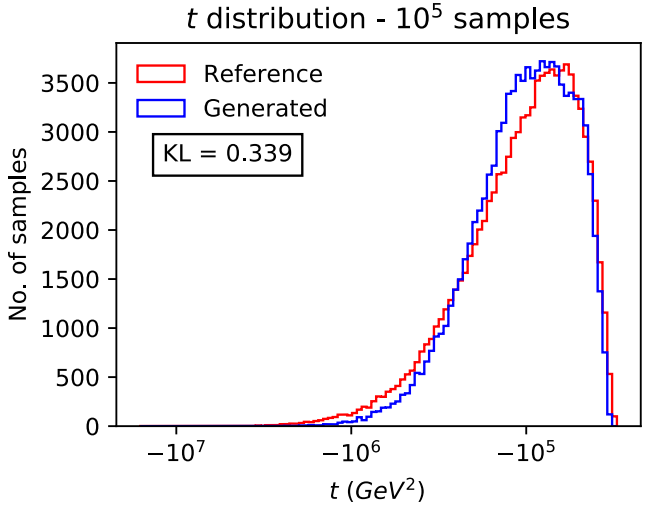
# Preparing actual runs: simulating the noise

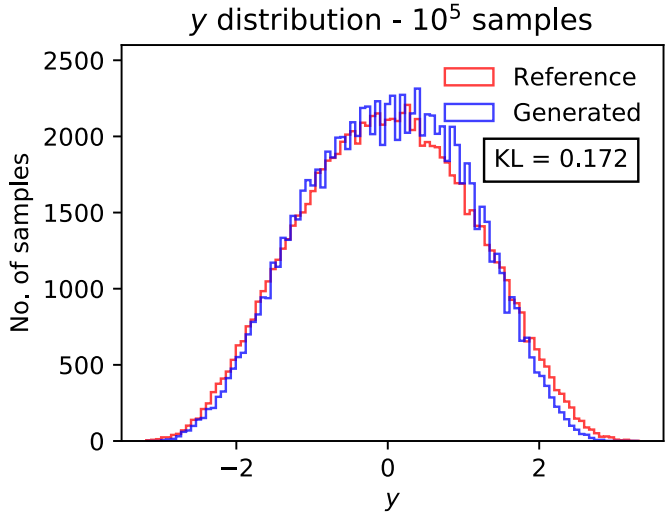
*Current hardware era: Noisy Intermediate-Scale Quantum (NISQ) devices*

- Read-out errors, decoherence time
- Probability of state $|0\rangle$ being actually in state $|1\rangle$ (and vice-versa)
- One-qubit gate errors, Two-qubit gate errors, etc.

**⇒ Before sending jobs on real hardware: estimate the impact of noise on the results**



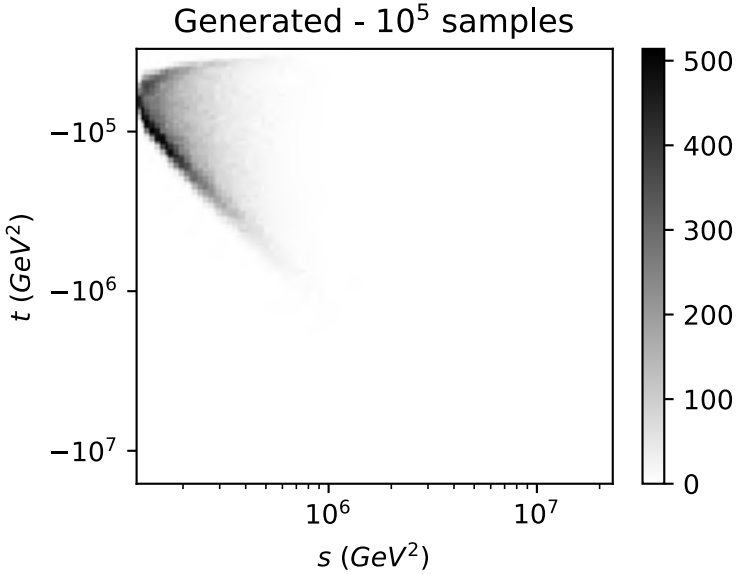



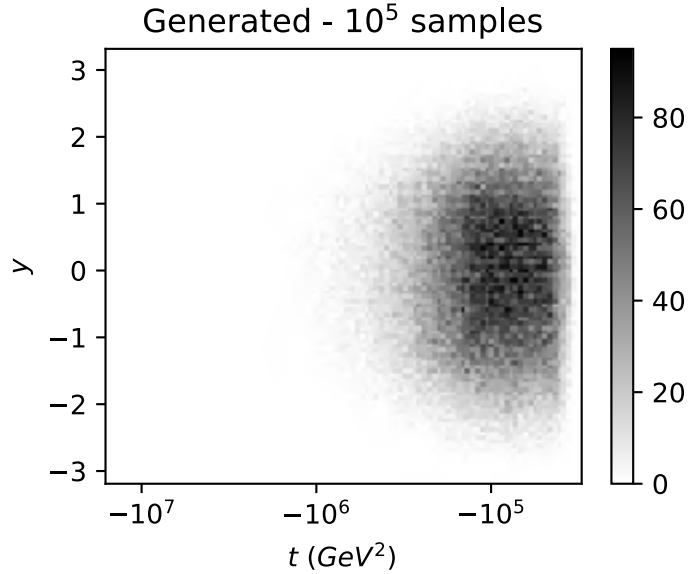
# Preparing actual runs: simulating the noise

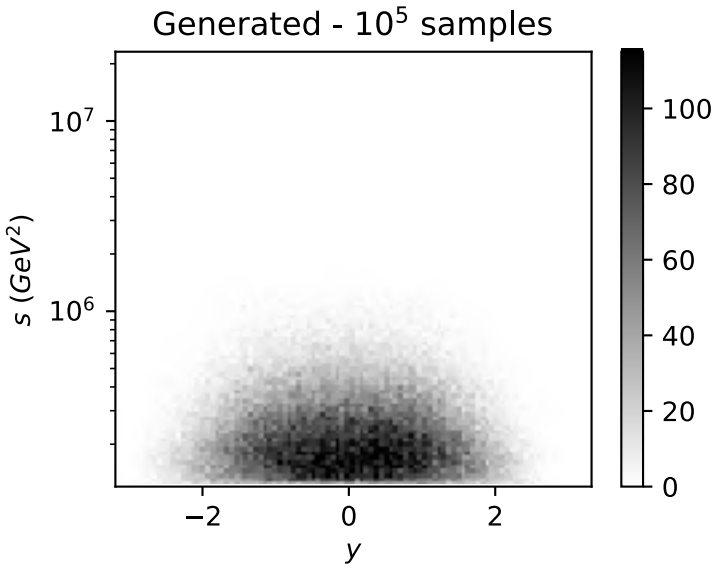
*Current hardware era: Noisy Intermediate-Scale Quantum (NISQ) devices*

- Read-out errors, decoherence time
- Probability of state $|0\rangle$ being actually in state $|1\rangle$ (and vice-versa)
- One-qubit gate errors, Two-qubit gate errors, etc.

⇒ **Before sending jobs on real hardware: estimate the impact of noise on the results**



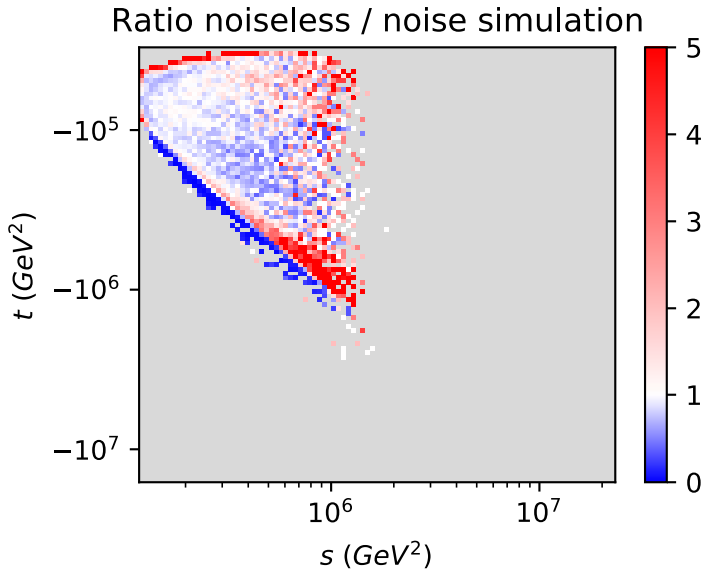



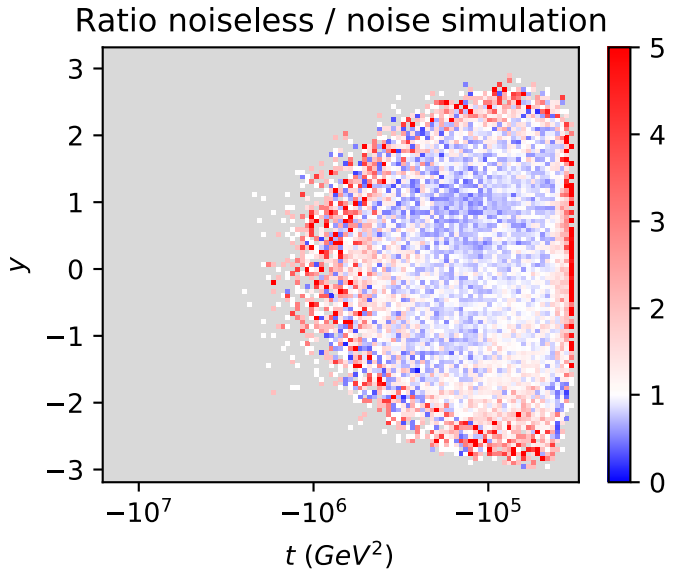
# Preparing actual runs: simulating the noise

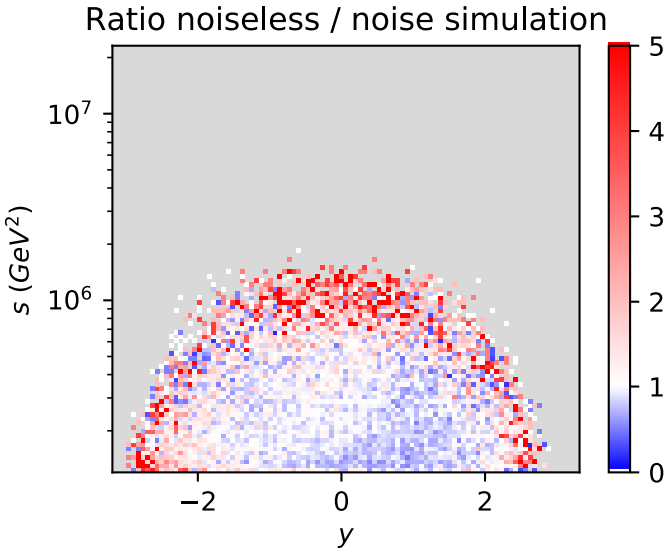
*Current hardware era: Noisy Intermediate-Scale Quantum (NISQ) devices*

- Read-out errors, decoherence time
- Probability of state $|0\rangle$ being actually in state $|1\rangle$ (and vice-versa)
- One-qubit gate errors, Two-qubit gate errors, etc.

**⇒ Before sending jobs on real hardware: estimate the impact of noise on the results**



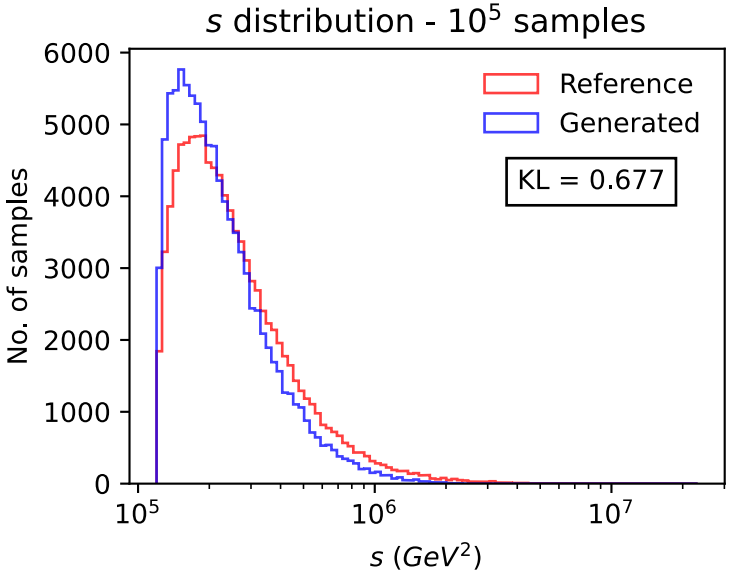



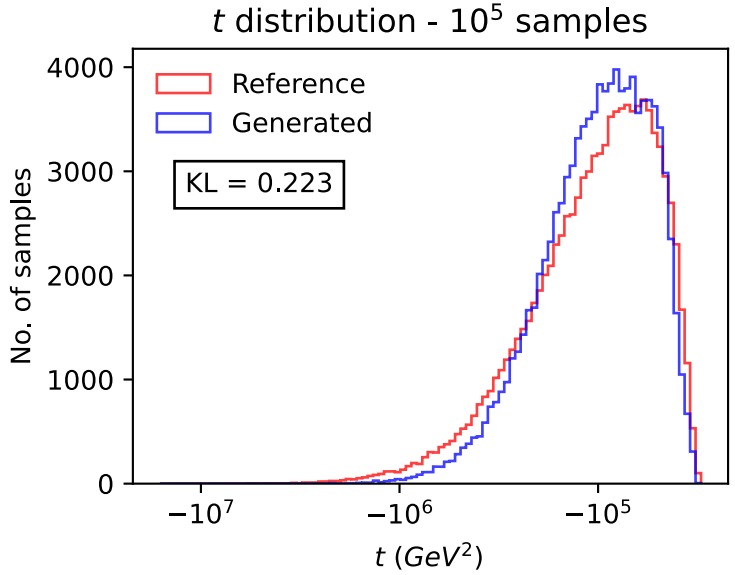
# Results on IBM Q hardware

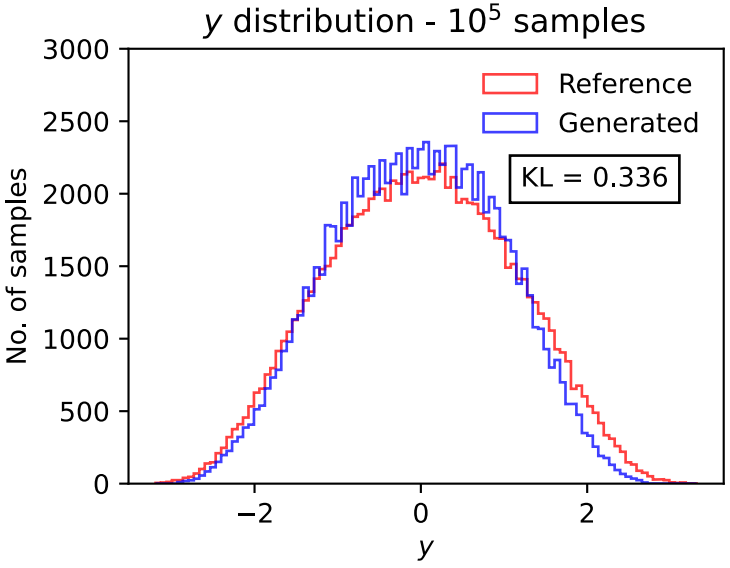
*Access to IBM quantum hardware via IBM Q cloud service*

Technology of superconducting transmon qubit: reduced sensitivity to charge noise

- Translation of qibo Python script to IBM qiskit Python script
- Run on **ibmq_santiago 5-qubit machine**

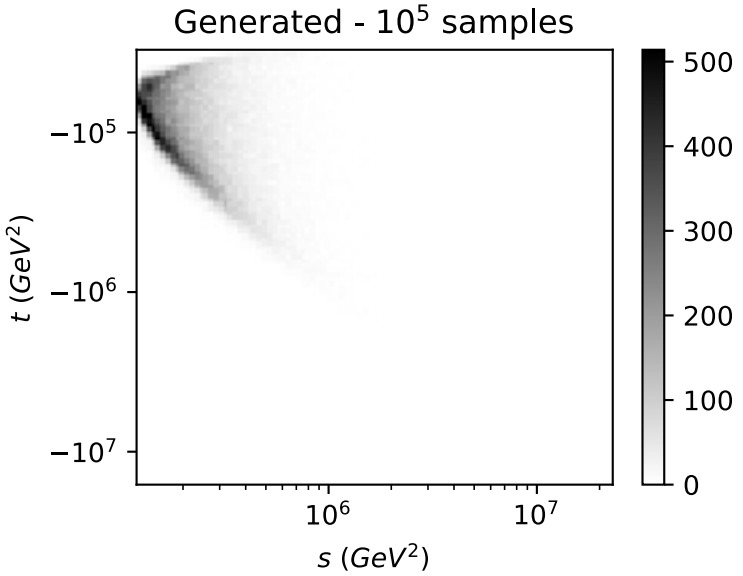



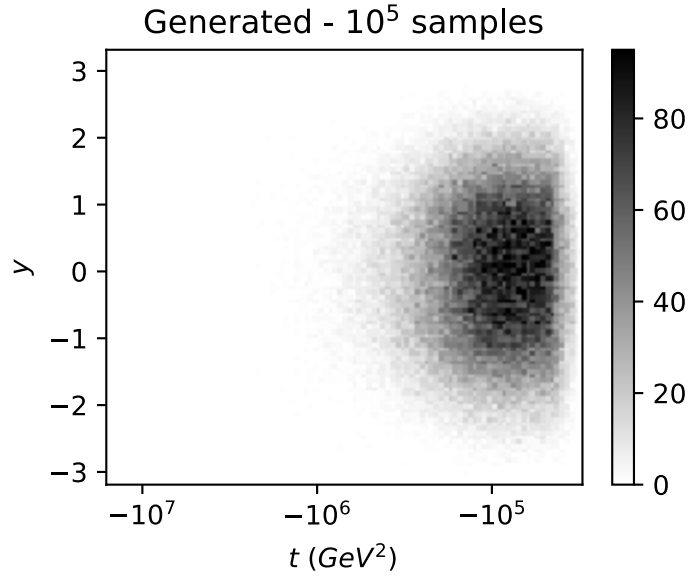
# Results on IBM Q hardware

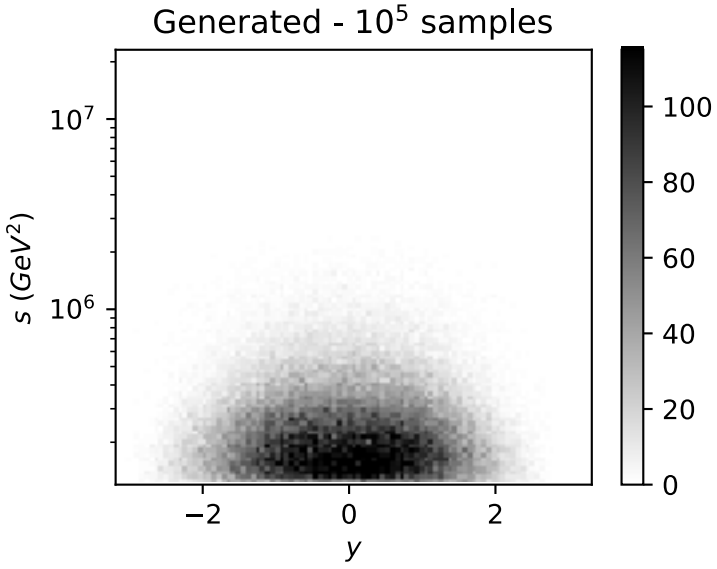
*Access to IBM quantum hardware via IBM Q cloud service*

Technology of superconducting transmon qubit: reduced sensitivity to charge noise

- Translation of qibo Python script to IBM qiskit Python script
- Run on **ibmq_santiago 5-qubit machine**

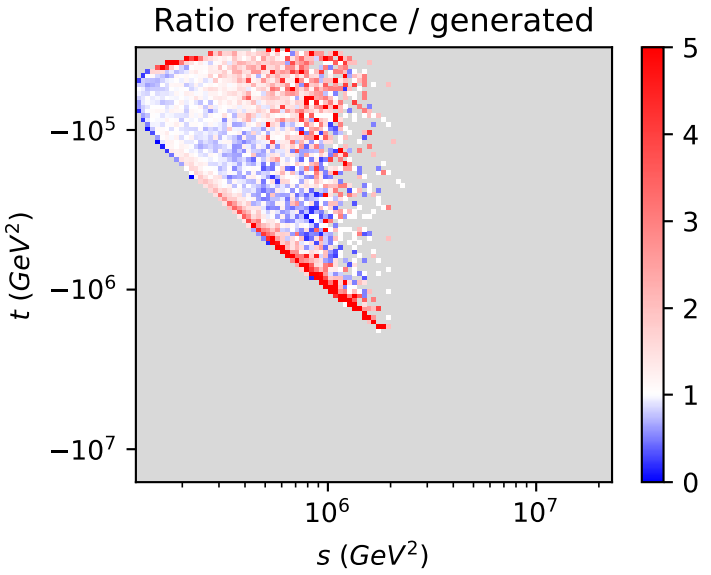



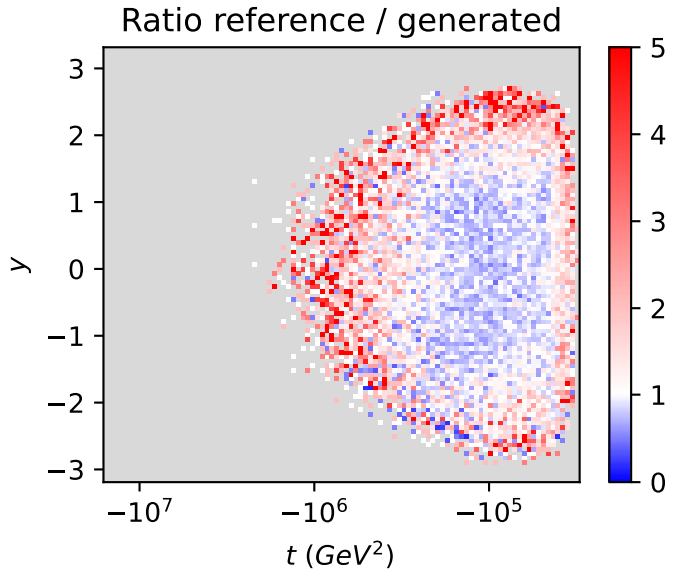
# Results on IBM Q hardware

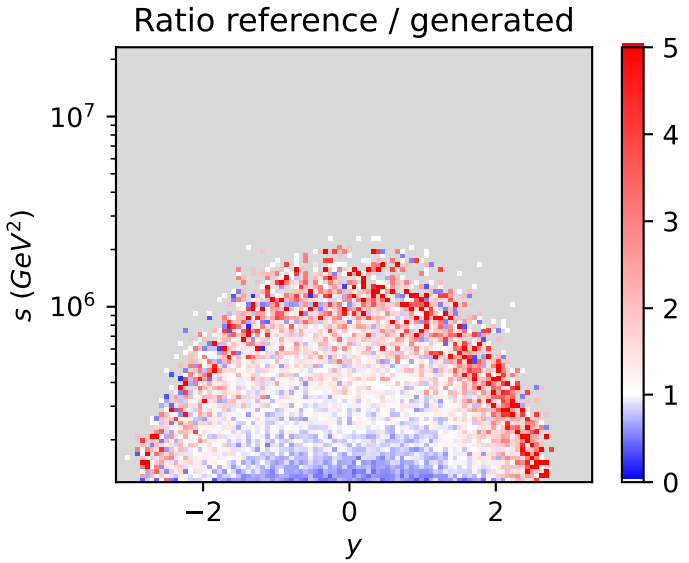
*Access to IBM quantum hardware via IBM Q cloud service*

Technology of superconducting transmon qubit: reduced sensitivity to charge noise

- Translation of qibo Python script to IBM qiskit Python script
- Run on **ibmq_santiago 5-qubit machine**

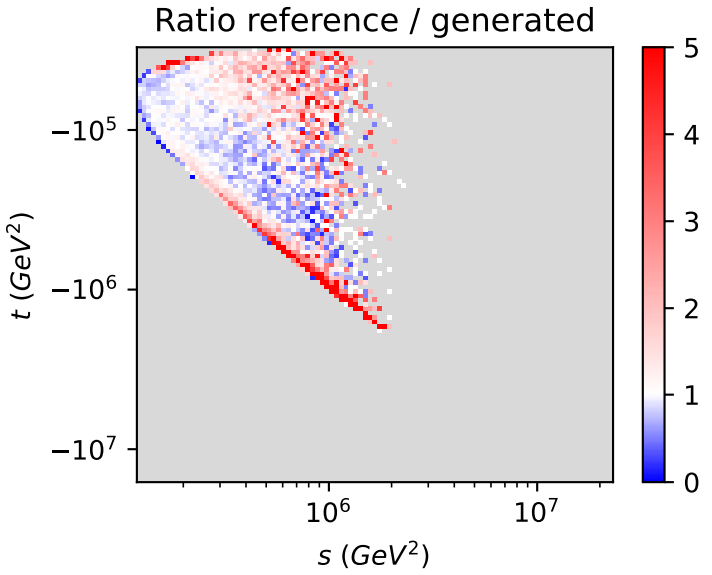
# Results on IBM Q hardware

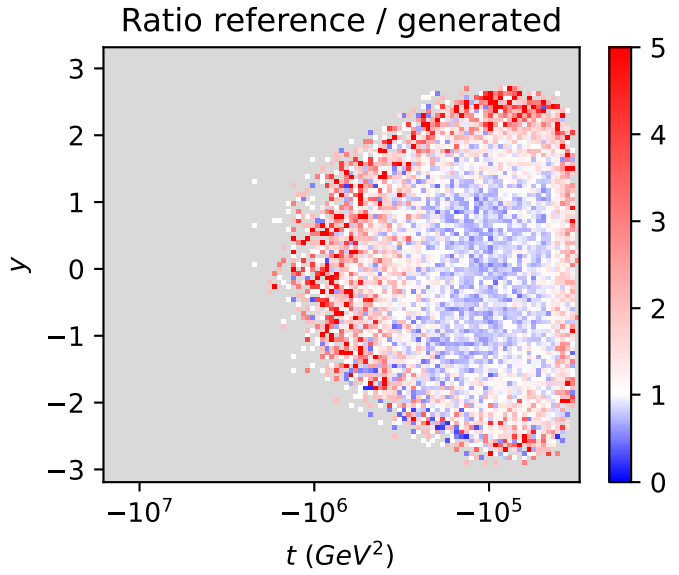
*Access to IBM quantum hardware via IBM Q cloud service*

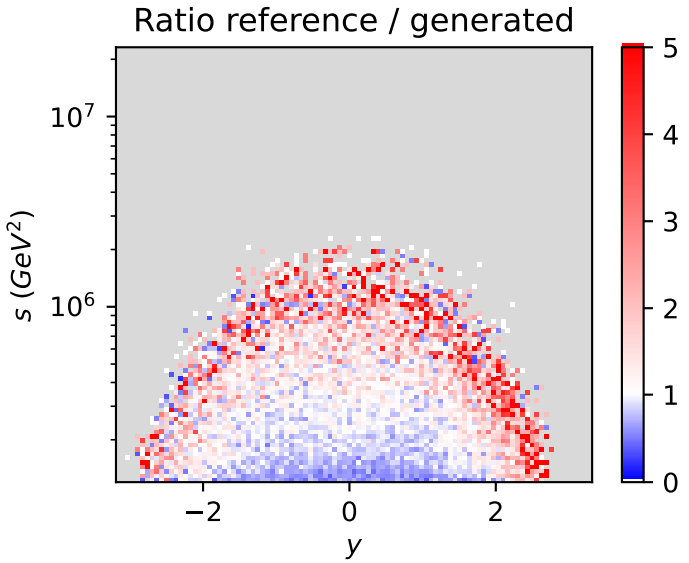
***Still good results with relatively low KL divergence!***

Technology of superconducting transmon qubit: reduced sensitivity to charge noise

- Translation of qibo Python script to IBM qiskit Python script
- Run on **ibmq_santiago 5-qubit machine**

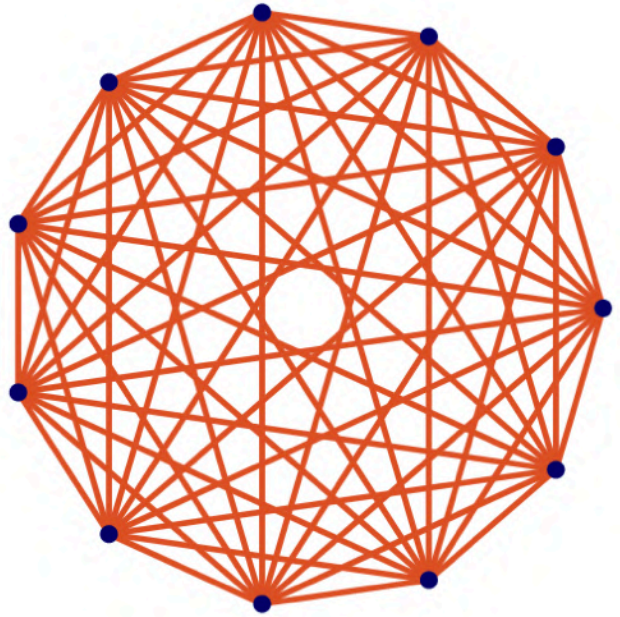



# Testing of different architectures

**Superconducting transmon qubits: *ibmq_santiago* with 2-neighbouring site connectivity**


Access via IBM Q cloud service

**Trapped ion technology: ionQ with all-to-all connectivity**

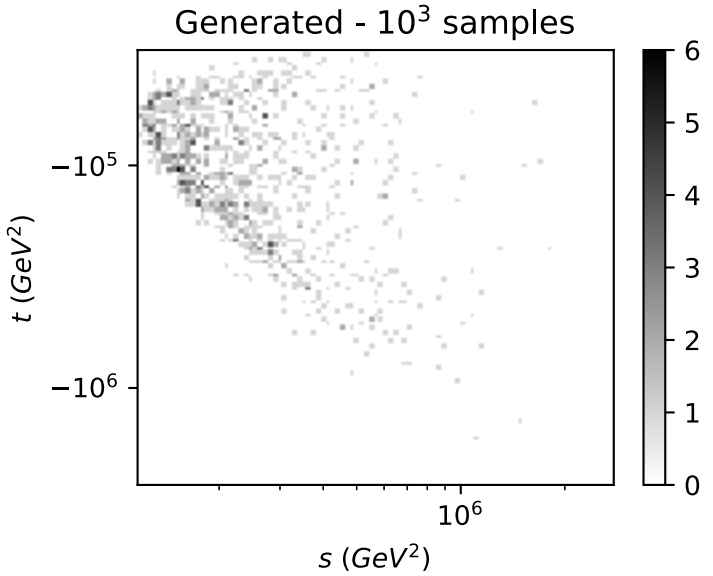


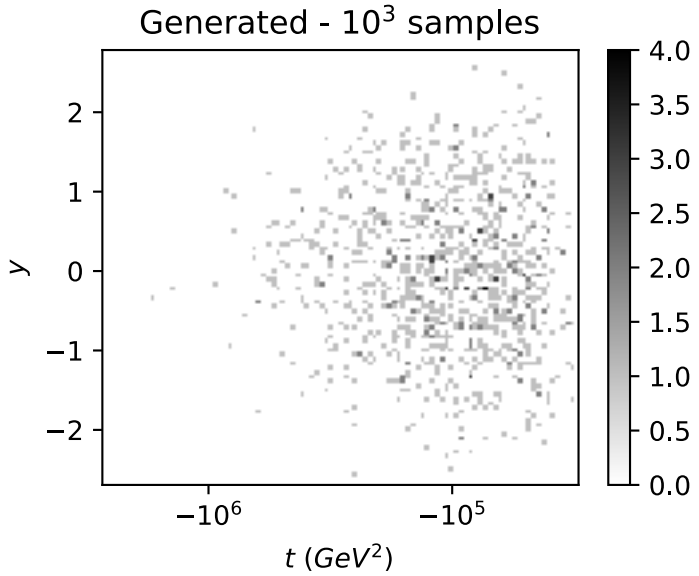
Access via Amazon Web Services

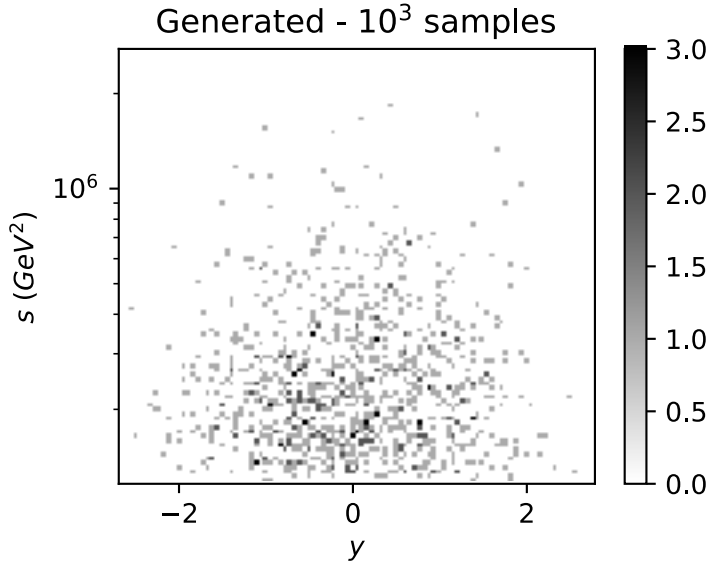
# Testing of different architectures: results

- Translation of qibo Python script to Amazon Web Services Braket script
- Access constraints to ionQ: test limited to 1k samples only

***ionQ samples:***



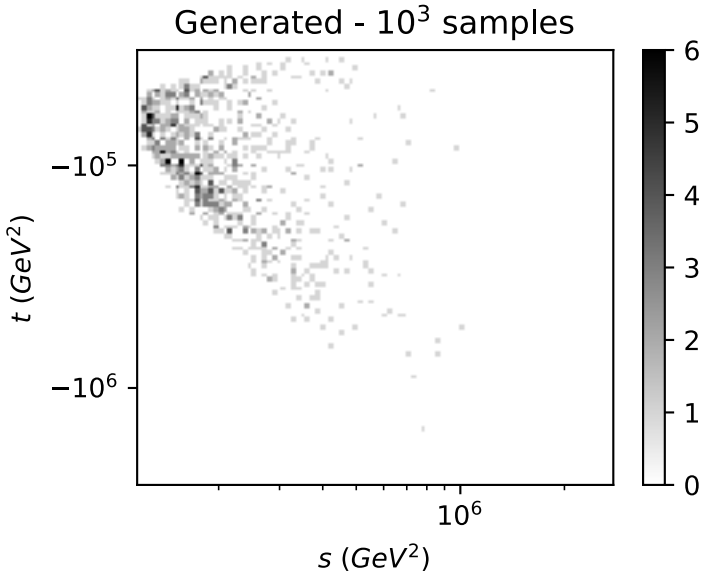



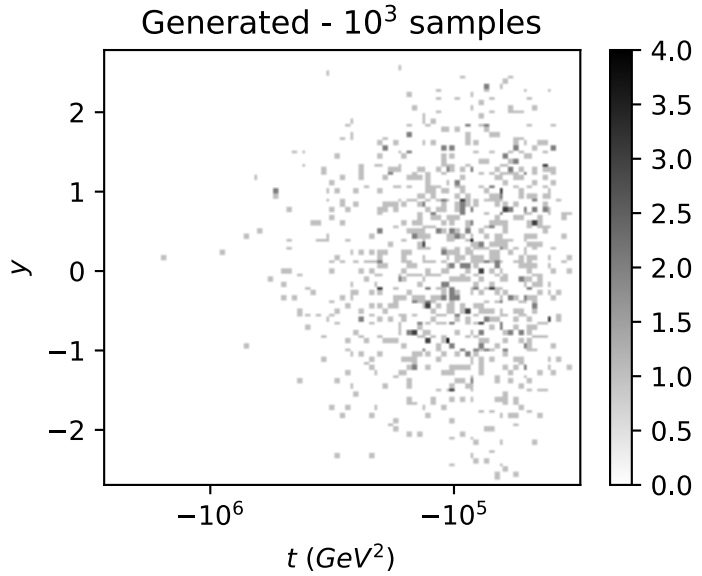



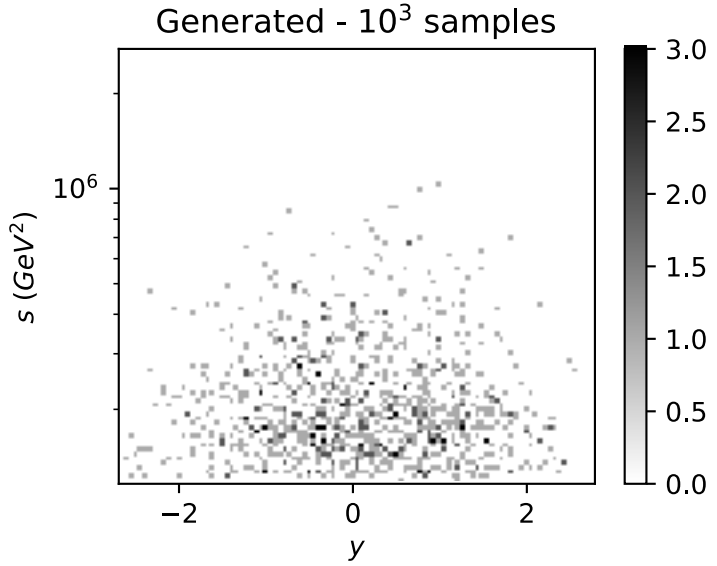
# Testing of different architectures: results

- Translation of qibo Python script to Amazon Web Services Braket script
- Access constraints to ionQ: test limited to 1k samples only

***IBM Q samples:***



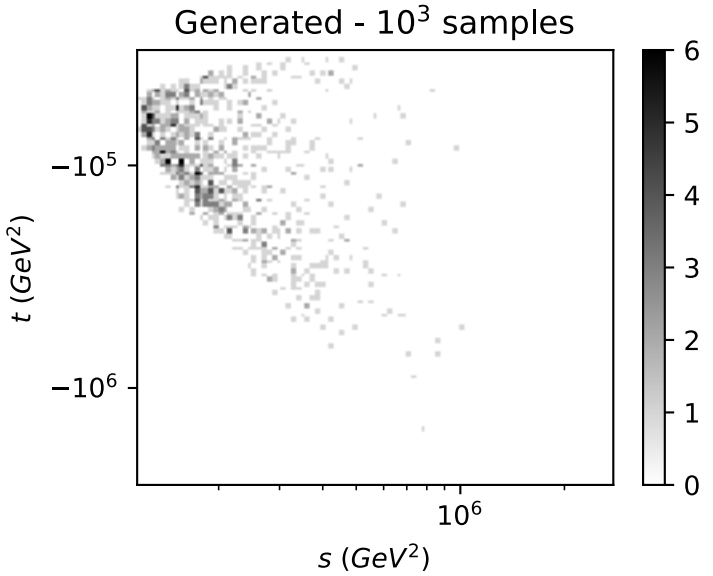



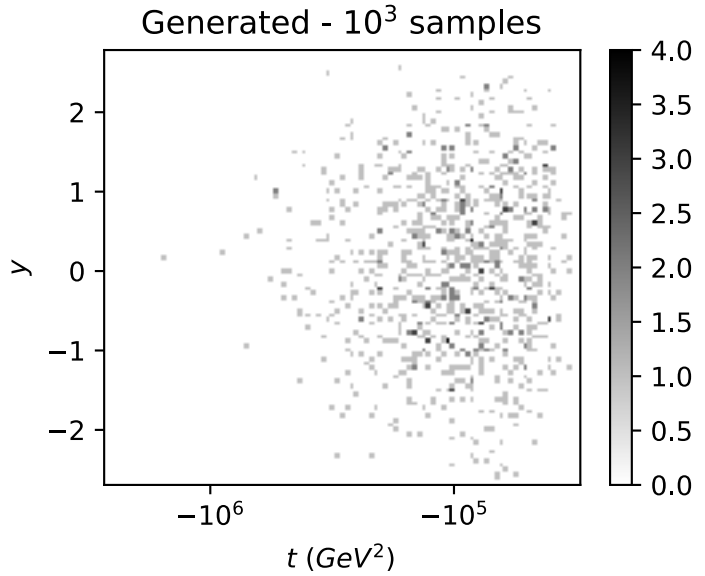
# Testing of different architectures: results

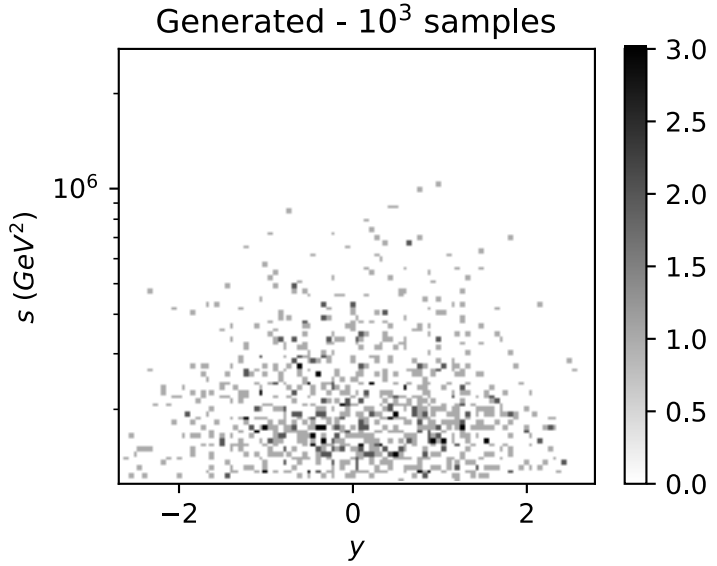
- Translation of qibo Python script to Amazon Web Services Braket script
- Access constraints to ionQ: test limited to 1k samples only

***Very similar results:***
***algorithm largely hardware-independent***

***IBM Q samples:***



# Conclusions: What's next?

# Summary of main points

- A proof-of-concept for a **quantum generator** (styled qGAN) has been presented
- A test-case with real Monte Carlo event has demonstrated **success**: the generator has learnt underlying $(s, t, y)$ distributions and correlations for $pp \to t\bar{t}$ production
  ⇒ **Demonstrated data augmentation from $10^4$ training data to $10^5$ generated data**
- The **quantum network** is **shallow**: great advantage in the current **NISQ era**
- Tested on **two different quantum architectures**: superconducting transmon qubits (IBM) and trapped ions (ionQ) with similar performances
  ⇒ **The quantum generator seems quite hardware-independent**

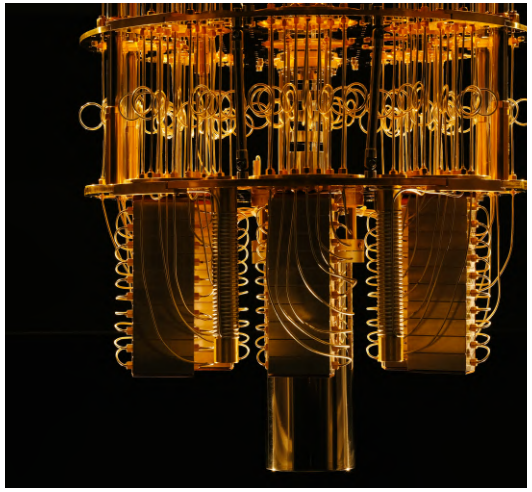
Code available at https://doi.org/10.5281/zenodo.5567077

# What's next?

Apply the styled qGAN to other problems (parton shower? Monte Carlo integration?)

Apply error mitigation techniques to increase the quality of the output?

Transition from hybrid approach to a full quantum GAN: discriminator+generator in one quantum network?



CERN | QUANTUM TECHNOLOGY INITIATIVE

***Thanks for your attention!***
***Questions?***