Introduction à PennyLane

PENNYLANE

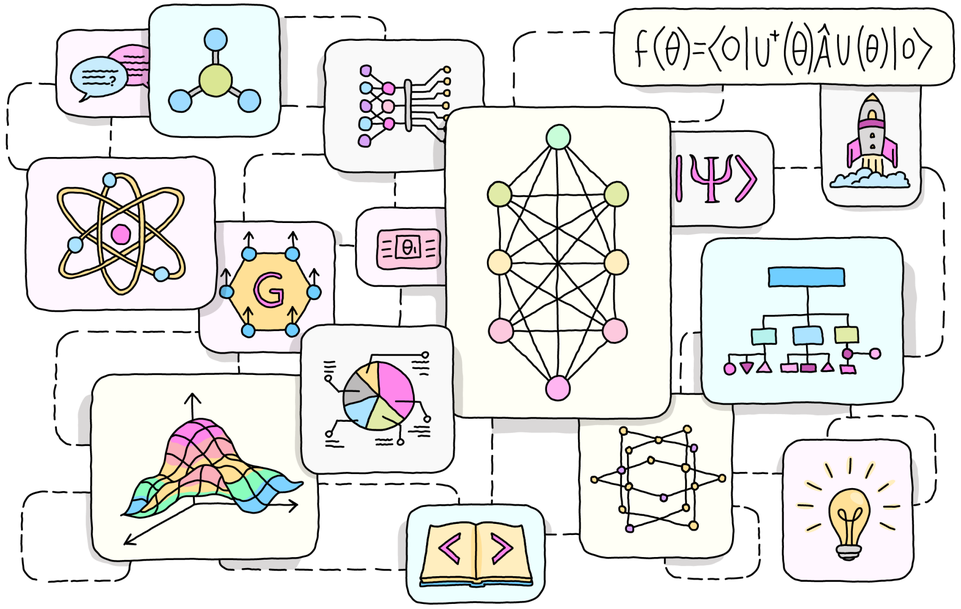
Ibrahim chegrane
08 Mai 2025

UNIVERSITÉ DE SHERBROOKE

PennyLane

Qu’est-ce que PennyLane ?

- Framework Python **open-source** pour la **programmation quantique**
- Supporte :
  - **Calcul quantique**
  - **Apprentissage automatique quantique**
  - **Chimie quantique**
- Compatible avec **TensorFlow**, **PyTorch**, et **JAX**



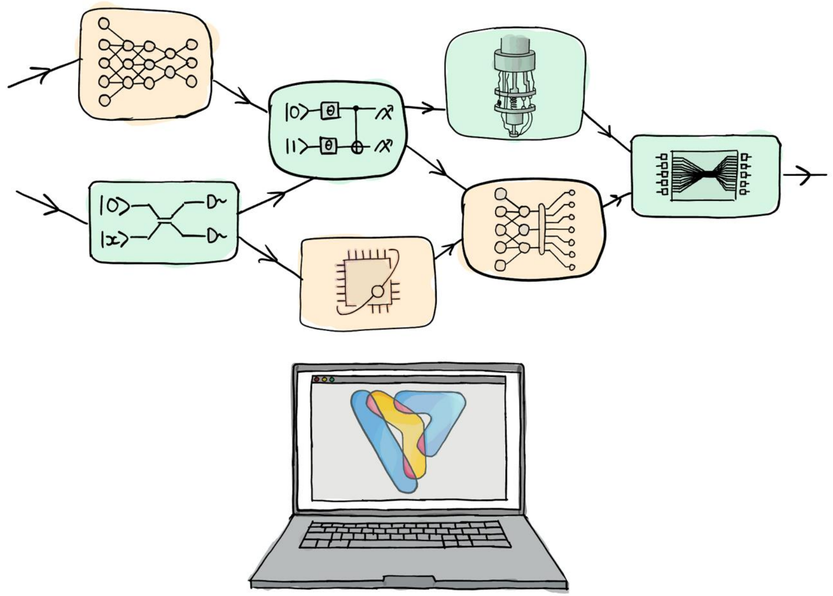
PENNYLANE

XANADU

PennyLane

Fonctionnalités clés qui le distinguent

- Compatibilité multiplateforme :
  - Un seul code → plusieurs appareils
  - Agnosticisme matériel



- Différentiation automatique:
  - Calculez automatiquement les gradients des circuits quantiques.
  - L'optimisation des circuits quantiques via des approches basées sur le gradient

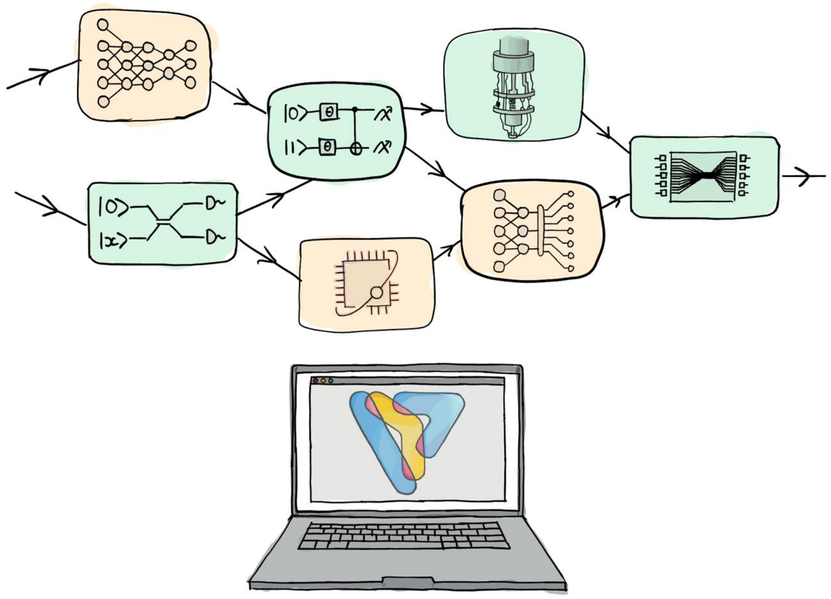
PENNYLANE

Le Cœur de PennyLane : Calcul Hybride

Le Graphe computationnel hybride

- Vision fondamentale : Les circuits quantiques ne sont pas des entités isolées, mais des composants intégrables dans des flux de calculs classiques plus larges.

- Le Qnode:
  - Des entrées classiques (features, poids, hyperparamètres),
  - Effectue les opérations quantiques,
  - Renvoie des sorties classiques (espérances, probabilités, échantillons), prêtes à être consommées par d’autres nœuds



Le Cœur de PennyLane : Calcul Hybride

Un graphe hybride simple

La figure 1 montre un exemple qui illustre l'idée centrale du framework.

```python
import pennylane as qml

# Create backend to run the quantum node
dev = qml.device('default.qubit', wires=1)

# Define quantum node
@qml.qnode(dev)
def circuit(x):
    qml.RX(x, wires=0)
    return qml.expval(qml.PauliZ(0))

# Define classical node/cost
def cost(weights):
    return weights[0] * circuit(weights[1])

# Update initial variable
opt = qml.GradientDescentOptimizer()
init_weights = [0.1, 0.2]
new_weights = opt.step(cost, init_weights)
```

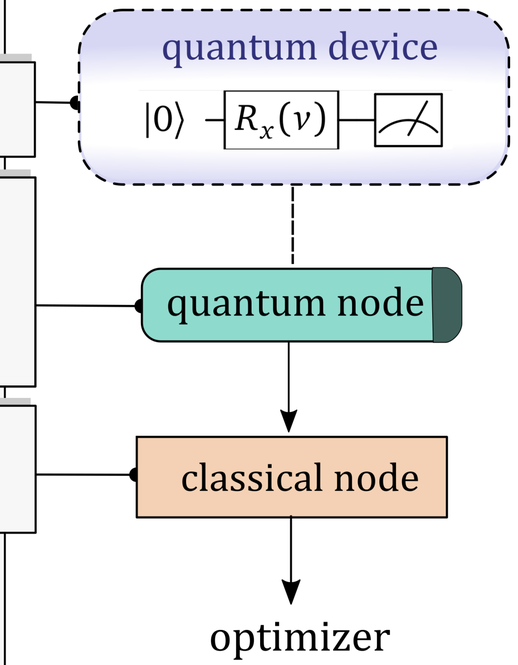


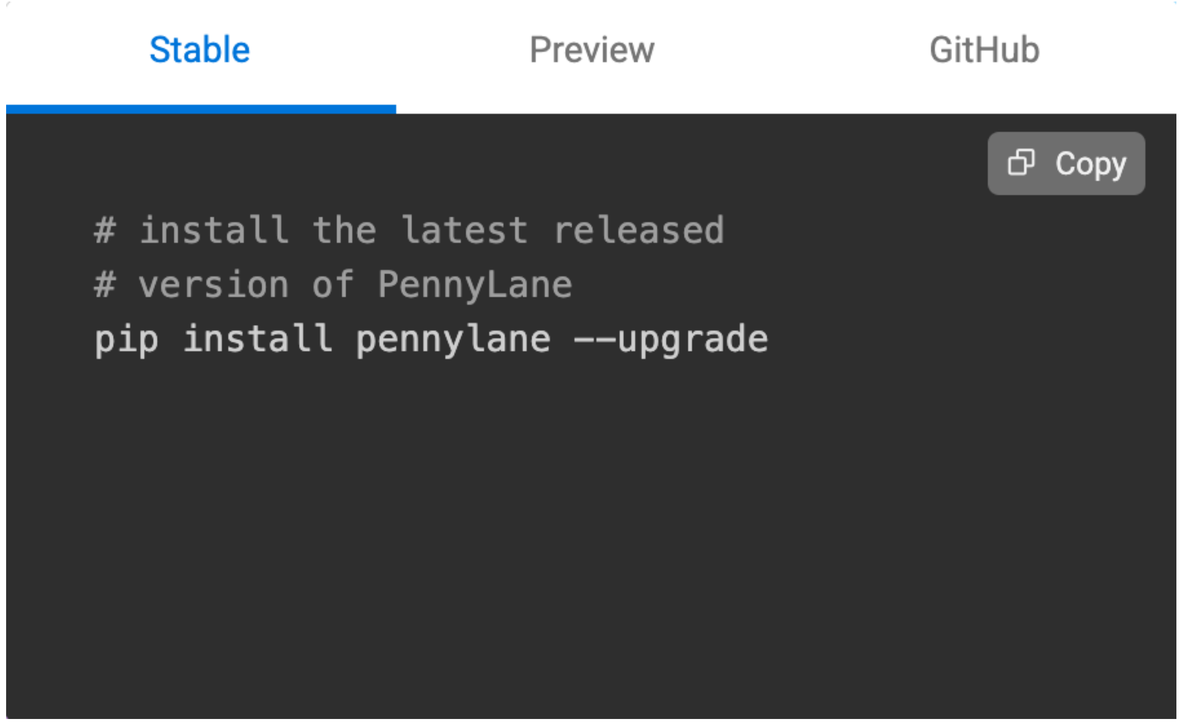
FIG. 1 : Exemple basique d'un programme PennyLane composé d'un nœud quantique suivi d'un nœud classique. La sortie du nœud classique constitue l'objectif de l'optimisation.

PennyLane: Automatic differentiation of hybrid quantumclassical computations: https://arxiv.org/pdf/1811.04968

Installer PennyLane

Avec pip

- Le seul prérequis pour installer PennyLane est d'avoir Python.



https://pennylane.ai/install

Mise en Place : Les Imports

import pennylane as qml

- Avant de commencer, voici les bibliothèques que nous importons généralement :

```
# Import principal de PennyLane
import pennylane as qml

# NumPy (PennyLane fournit une version compatible avec l'autodiff)
from pennylane import numpy as np
# Note : Préférez pennylane.numpy pour la différentiation automatique

# Optionnel - pour la visualisation
import matplotlib.pyplot as plt

# Optionnel - pour l'intégration avec le Machine Learning classique
# import torch
# import tensorflow as tf
# import jax
# from jax import numpy as jnp
```

https://pennylane.ai/install

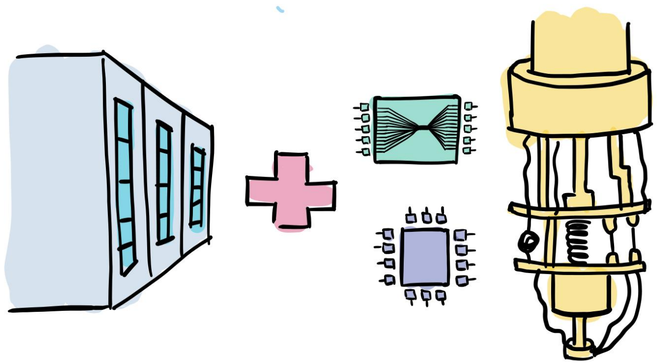
Devices Quantiques

qml.device

- Un *device* représente le **backend** d'exécution des circuits quantiques dans PennyLane :
  - Simulateur cpu, gpu ou processeur quantique réel.
- Dispositifs supplémentaires via plugins:
  - IBM Quantum, Amazon Braket, Google Quantum AI

```
dev = qml.device(name, wires=...)
```

- default.qubit
- lightning.qubit (en C++)
- default.mixed
- qiskit.remote
- monarq.default
- …



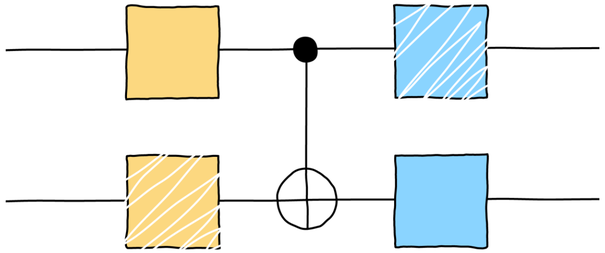
https://pennylane.ai/devices

Fonction Quantique

Définir le Circuit

Le cœur d'un programme PennyLane est une fonction python spéciale qui définit le circuit quantique :

- Prend des entrées classiques (ex: paramètres x, y).
- Contient la séquence d’opérations quantiques (qml.RZ, qml.CNOT...)
- Peut contenir des structures classiques (for, if, etc.)
- Retourne une ou plusieurs mesures quantiques (qml.expval, qml.probs...).



```python
import pennylane as qml

# Exemple de fonction quantique
def ma_fonction_quantique(x, y):
    qml.RZ(x, wires=0)          # Rotation Z sur qubit 0
    qml.CNOT(wires=[0, 1])      # Porte CNOT entre qubit 0 et 1
    qml.RY(y, wires=1)          # Rotation Y sur qubit 1
    return qml.expval(qml.PauliZ(1)) # Mesure: valeur moyenne de Z sur qubit 1
```

Fonction Quantique

Construire des circuits complexes en combinant plusieurs fonctions

- Une fonction quantique peut être définie sans instruction return.
- Elle sert alors de sous-circuit (ou "layer", "subroutine") : un bloc d'opérations réutilisable que vous pouvez appeler depuis d'autres fonctions quantiques.

Avantages :

- Clarté du code.
- Réutilisabilité des blocs
- Meilleure structuration des circuits complexes

```
def encoding_layer(params):
    qml.RX(params[0], wires=0)
    qml.RY(params[1], wires=1)

def entangling_layer():
    qml.CNOT(wires=[0, 1])

def my_full_circuit(params):
    encoding_layer(params)
    entangling_layer()
    return qml.expval(qml.PauliZ(0))
```

Exécution via QNode

qml.Qnode

- Le QNode est le nœud d’exécution qui relie un circuit quantique à un device (simulateur ou matériel réel).

Instanciation explicite

```
qnode = qml.QNode(my_quantum_function, dev)
```

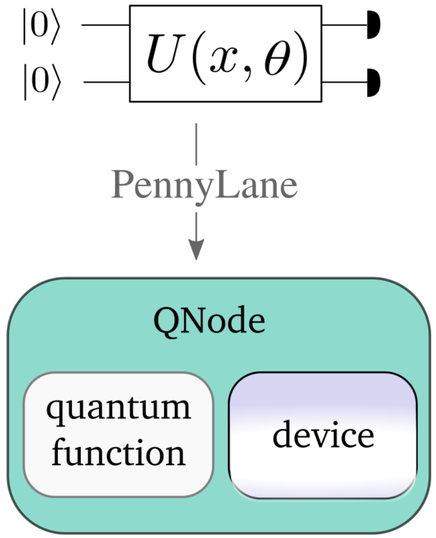
Flexibilité (créer dynamiquement des QNodes)
Plus de contrôle

Création de QNode

Décorateur `@qml.qnode`

```
@qml.qnode(dev)
def my_quantum_function(...):
    ...
```

Méthode simple, concise et "Pythonique"
Syntaxe claire, directe

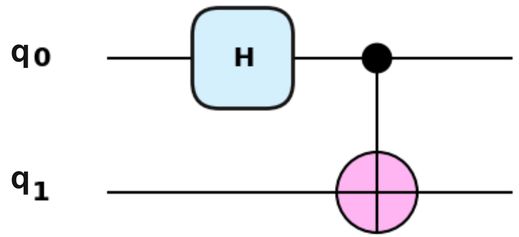


Un **circuit n’est exécutable que s’il est encapsulé dans un QNode**.

PennyLane : Ordre des Qubits

Big Endian

- PennyLane utilise l'ordre **big endian** pour les registres de qubits.
  - Le qubit 0 (wire 0) correspond au **bit le plus à gauche** dans la représentation binaire.



$q_0\ q_1$

$q_1\ q_0$

|01⟩ : wire 0 = 0 (gauche), wire 1 = 1 (droite)

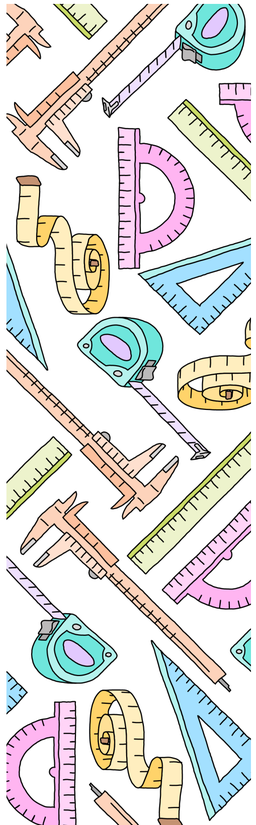
https://discuss.pennylane.ai/t/endian-ness-of-circuit-simulator/1797
https://en.wikipedia.org/wiki/Endianness

Mesures dans PennyLane

Vue d’ensemble

Deux grands types de mesures :

- Analytique (shots=None, simulateur) : Résultats exacts, calculés à partir de l’état quantique.
- Statistique (shots > 0, matériel réel ou simulateur avec shots) : Résultats estimés à partir d’échantillons (shot noise).



Fonctions principales :

- qml.sample() / qml.counts() : Résultats bruts ou comptages des mesures individuelles.
- qml.probs() : Probabilités des états mesurés ou des valeurs d’un observable.
- qml.expval() : Valeur moyenne (espérance) d’un observable.

Mesures dans PennyLane

Exemples

- La mesure est représentée par la ou les valeurs retournées par la fonction du circuit (return ...).
- On peut mesurer un ou plusieurs qubits
- Avec ou sans des observables (sauf pour qml.expval()).

```
return qml.sample(wires=0)
```

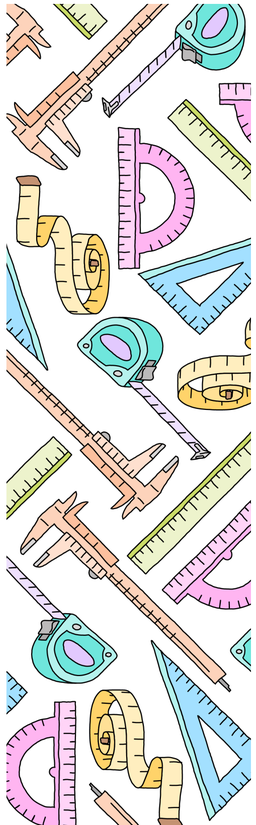
```
return qml.counts(), qml.sample()
```

```
# On demande les comptes pour les valeurs propres de PauliZ(0)
return qml.counts(op=qml.PauliZ(0))
```

```
return qml.probs(wires=[0, 1])
```

```
return qml.expval(qml.PauliZ(0))
```

```
# On demande la valeur moyenne du produit tensoriel :
# PauliZ agissant sur le qubit 0 ET PauliX agissant sur le qubit 1
observable = qml.PauliZ(0) @ qml.PauliX(1)
return qml.expval(observable)
```



Compilation et Transpilation

Compilation (qml.compile) : suite de transformations qui réécrit un circuit pour le :

- Simplifier : fusion de rotations simples
- Optimiser : annulation de portes inverses, simplification d'observables
- Adapter : substitution de types de portes selon des contraintes

Transpilation (qml.transforms.transpile): adapte un circuit à une topologie matérielle :

- Tient compte de la connectivité et du jeu de portes
- Ajoute des SWAPs, remappe les qubits pour respecter le hardware

Différentiation et Gradients

- PennyLane calcule automatiquement les **gradients** des circuits quantiques.
- Essentiel pour l'optimisation et le machine learning quantique.
- Plusieurs méthodes disponibles : analytique, numérique, backpropagation.

```python
@qml.qnode(dev)
def circuit(x):
    qml.RX(x, wires=0)
    return qml.expval(qml.Z(0))

x = np.array(0.5, requires_grad=True)
grad_fn = qml.grad(circuit)
```