EAI Endorsed Transactions
on Tourism, Technology and Intelligence
Research Article EAI.EU

PennyLane–Qiskit Plugin: A Protocol for Integrating Noisy, Fake, and Real Quantum Backends

Matthieu Hubert[1], Quang Nhat Le[1,*], and Emmanuel Haven[1]

[1]Memorial University, St. John's, NL A1B3X5, Canada

Abstract

Quantum computing frameworks like PennyLane, Qiskit, and others offer powerful tools for developing quantum algorithms. However, integrating their devices can be challenging for newcomers, especially given the frequent updates in frameworks like Qiskit, which can create significant barriers to learning and thus slow down the growth of quantum computing and quantum machine learning. The PennyLane–Qiskit plugin is a particularly interesting framework that allows researchers and practitioners interested in quantum computing and quantum machine learning to implement and simulate realistic models in a simple and efficient way. This work provides a practical method to use the PennyLane–Qiskit plugin, focusing on three main device types: the Aer simulator with noise models, IBM fake backends (with `FakeManilaV2` as a representative example), and real IBM Quantum hardware accessed remotely (with `ibm_torino` used in this study). We demonstrate how to set up and use each of these devices within PennyLane using a simple example of a quantum machine learning regression task on simulated stock price data. Our goal is to simplify the process for all interested users, especially (but not only) beginners by clearly explaining and demonstrating how to use each of these device options while highlighting best practices for effectively leveraging PennyLane and Qiskit together via Pennylane–Qiskit plugin. This work aims to provide a turnkey solution enabling users with only minimal foundational knowledge to quickly start running meaningful simulations. The three devices implementation will be explained with the code here. The accompanying full code will be provided to ensure that all results can be easily reproduced, including with users' own datasets.

Received on 18 March 2026; accepted on 07 June 2026; published on 11 June 2026

Keywords: Hybrid quantum–classical computing, noisy quantum simulation, PennyLane–Qiskit plugin, quantum hardware backends, quantum machine learning

doi:10.4108/eettti.12288

1. Introduction

Quantum computing has emerged as a promising paradigm for solving complex problems, with potential applications in finance [1], chemistry [2], healthcare, medicine [3], cryptography [4], meteorology, and climate modeling [5]. Over the past decade, several open-source quantum computing frameworks have been developed to facilitate the design, simulation, and execution of quantum algorithms on both simulators and real quantum hardware. Table 1 summarizes some of the major quantum programming languages along with the quantum hardware platforms they support.

Among the most widely used frameworks, *Qiskit* has become a reference platform for interacting with IBM Quantum hardware, offering a rich ecosystem. *Cirq*, developed by Google, is also a well-known quantum computing framework, e.g., which allows us to integrate naturally with *TensorFlow Quantum*, enabling the implementation of hybrid quantum–classical machine learning models [6]. In parallel, *PennyLane* has positioned itself as a cross-platform framework specifically designed for hybrid quantum–classical workflows, with a strong interest on quantum machine learning, automatic differentiation, and integration with classical optimization libraries. Additionally, *Q#* was developed by Microsoft and is integrated with the Azure Quantum ecosystem [7].

*Corresponding author is Quang Nhat Le, e-mail: qnle@mun.ca

Table 1. Major quantum programming languages and their supported hardware platforms

Language	Supported Hardware
Qiskit	IBM Quantum
PennyLane	Multiple
Cirq	Google Quantum
Q#	Azure Quantum

While these frameworks provide powerful and complementary capabilities, the growing diversity of tools and application programming interfaces has also introduced significant complexity, particularly in the management and integration of quantum devices. Frequent updates, evolving interfaces, and hardware-specific constraints especially in frameworks such as Qiskit can create substantial barriers for newcomers, slowing down experimentation and so the adoption of quantum computing and quantum machine learning techniques.

In this context, the PennyLane–Qiskit plugin offers an effective solution by providing a unified and high-level interface for accessing Qiskit-based devices directly within the PennyLane framework. This integration allows users to easily switch between different interesting execution environments, ranging from noisy simulators to realistic fake backends and real quantum hardware, without modifying the core of the code.

The aim of this work is to provide a practical method and a concrete introduction to the PennyLane–Qiskit plugin, with a particular focus on three device categories that are most relevant for realistic experimentation: (i) the Aer simulator with a noisy model, (ii) IBM fake backends, and (iii) IBM real backends. With a simple quantum machine learning regression task on simulated stock price data, we aim to demonstrate how each device can be configured and used in practice. This work is intended as a turnkey resource for researchers, practitioners, and newcomers interested in quantum computing and quantum machine learning, enabling them to quickly perform meaningful quantum simulations while following best practices for effectively combining PennyLane and Qiskit. The reduction of the practical barrier to access and use realistic quantum devices is a central motivation of this work, as it is essential for the sustainable development and adoption of quantum computing technologies.

The outline of this paper is as follows. Section 2 introduces the main frameworks used in this work. Section 3 introduces the three devices cited previously and shows their practical usage through the simple quantum machine learning example. Finally, Section 4 concludes the paper and discusses future perspectives.

2. Background

2.1. PennyLane and Qiskit

PennyLane is an open-source quantum computing framework designed to build and train hybrid quantum–classical models. It was first released in 2018 by Xanadu with the explicit goal of bridging quantum computing and modern machine learning. PennyLane is particularly well suited for quantum machine learning applications where parameterized quantum circuits are combined with classical optimization algorithms. One of its main strengths is its seamless integration with classical machine learning libraries such as PyTorch, TensorFlow, and JAX, thus enabling automatic differentiation through quantum circuits. PennyLane provides a high-level interface that abstracts many hardware-specific details to enable users to focus on model design rather than quantum operations [8, 9].

Qiskit is a widely used quantum computing framework developed by IBM and first released in 2017. It provides both low-level and high-level tools to build, simulate, and execute quantum circuits. Qiskit supports the full workflow of quantum algorithm development, from circuit construction and compilation to execution and result analysis [10]. A key feature of Qiskit is its close integration with IBM Quantum services, which gives users access to a variety of execution environments including ideal simulators, noisy simulators, and fake backends that mimic real devices and actual quantum hardware. Over time, Qiskit has evolved into a modular ecosystem, with specialized packages targeting different application domains such as finance and chemistry.

The Qiskit Machine Learning module, introduced later as part of this ecosystem, extends Qiskit's capabilities to quantum machine learning tasks [11]. It provides tools for building variational quantum circuits, quantum kernels, and hybrid learning pipelines, thus making Qiskit a comprehensive framework for both quantum algorithm research and applied experimentation.

In Qiskit, the execution target is referred to as a *backend*. Each backend is characterized by hardware-specific properties such as the number of qubits, supported gate sets, qubit connectivity, and noise characteristics. This explicit representation of hardware

constraints gives users fine-grained control and reflects the realities of near-term quantum devices. However, while this level of control is powerful, it can also introduce complexity, especially for users new to quantum computing or primarily interested in higher-level applications such as quantum machine learning. Additionally, the necessary evolution of Qiskit versions, i.e., notably the transition from v1.x to v2.x, has introduced changes in application programming interfaces (APIs) and programming paradigms, which can further increase the learning curve for practitioners.

2.2. PennyLane–Qiskit Plugin

The PennyLane–Qiskit plugin connects PennyLane to Qiskit by allowing Qiskit backends to be used as PennyLane devices. Its goal is to make it easier to combine the strengths of both frameworks: the flexibility and hardware access provided by Qiskit and the simple high-level programming style of PennyLane. With this plugin, users can run PennyLane circuits on a wide range of Qiskit backends including simulators and fake and real quantum hardware. Most of the technical configuration is handled automatically, which helps users focus on building and testing their quantum models. In this work, we use the PennyLane–Qiskit plugin to show how quantum experiments can be run with minimal effort.

While the three backends share the same interface, they differ in execution and noise characteristics. The Aer simulator runs locally and allows for user-defined noise models. Fake backends also run locally but mimic specific machines using past fixed snapshots. Real hardware provides real-time physical noise but requires remote access via an API Token and is subject to queue times. These differences are summarized in Table 2.

3. Methodology

The following executions are performed on Google Colab. We need to install the `pennylane`, `pennylane-qiskit`, and `qiskit-ibm-runtime` libraries via the `!pip` command. To ensure reproducibility, all simulations were performed using Python 3.12.13, PennyLane v0.45.0, the PennyLane–Qiskit plugin v0.45.0, Qiskit v2.3.0, and `qiskit-ibm-runtime` v0.45.1.

3.1. Data: A Classic Financial Forecasting Task

First, in order to demonstrate a simple example of quantum machine learning, we need some data to work with. For this purpose, we generate synthetic data simulated in a simplified manner aiming to capture the essential characteristics of stock market behavior. This approach allows us to create controlled data that reflects key patterns without relying on the complexities of real markets. Other types of personal financial datasets or even non-financial datasets can be used as alternatives depending on the specific application or analysis goals. The following is the implementation to generate our simulated financial data:

```
np.random.seed(42)

S0 = 100
mu = 0.002
sigma = 0.01
steps = 100

prices = [S0]
dt = 1
for _ in range(steps - 1):
    dS = prices[-1] * (mu * dt + sigma * np.random
        .normal() * np.sqrt(dt))
    prices.append(prices[-1] + dS)

df = pd.DataFrame({'Close': prices})
```

This above code simulates a stock price over time (100 steps) using a basic financial model. Starting from an initial price, the value is updated iteratively by adding a small random variation that reflects both a positive average trend (drift) and random fluctuations (volatility), as commonly modeled in the classical financial literature [12, 13]. To ensure reproducibility, the random seed is fixed at the beginning of the simulation. The resulting financial time series mimics the evolution of a real stock price over time, as shown in Fig. 1.

Figure 1. Evolution of the stock price over time

3.2. Simple Used Quantum Pipeline

The regression pipeline is structured as follows. The input feature X consists of the closing price at time t, and the target y is the closing price at time $t+1$, forming a one-step-ahead prediction task. The dataset is split chronologically into a training set (80%) and a test set (20%), with no shuffling to pr eserve th e temporal ordering of the data. Model performance is evaluated on the test set using the root mean squared error (RMSE).

We use a simple circuit consisting of only R_Y rotations on a single qubit and a measurement in

Table 2. Comparison of the three backend types available via the PennyLane–Qiskit plugin

Backend	Execution	Noise Model	Access
Aer Simulator	Local	User-defined	Offline
Fake Backend	Local	Past fixed snapshots	Offline
Real Hardware	Remote	Real-time	Online (API Token)

the Pauli-Z basis. This choice aims to demonstrate the feasibility of basic quantum regression without unnecessary complexity. The optimization focuses on a single parameter, the rotation angle θ, applied to the parameterized gate $R_Y(\theta)$ of the circuit. This single parameter is adjusted over only 5 epochs, i.e., a very limited number but sufficient for a fast and simple convergence, which is well suited to the simplicity of the used quantum model. Thus, the training remains efficient and straightforward, appropriate for the experimental framework.

In all experiments, the number of shots is set to 128 per circuit execution. Each result reported in this work corresponds to a single run without averaging over multiple executions. The primary objective of this study is to provide a practical and accessible protocol for using the PennyLane–Qiskit plugin across different backend types, illustrated through a concrete quantum machine learning example. Accordingly, the focus is not on optimizing predictive performance or conducting a rigorous benchmarking of the backends, but rather on demonstrating the feasibility and ease of use of each execution environment.

3.3. Noisy Simulation with Aer Device

The PennyLane–Qiskit plugin makes it easy to access and implement various quantum backends and noise models provided by Qiskit Aer directly within PennyLane. This integration simplifies experimentation by allowing users to switch between different devices and simulate realistic noisy quantum circuits without complex backend management. To explore the available simulators (backends) in Qiskit Aer, we can run the following codes:

```
from qiskit_aer import Aer
print(Aer.backends())
```

Then, a list of available Aer backends is shown. The following are some of them:

- `aer_simulator`
- `aer_simulator_statevector`
- `aer_simulator_density_matrix`
- `qasm_simulator`
- `statevector_simulator`
- `unitary_simulator`

`aer_simulator` serves as the general-purpose entry point supporting various execution methods. For example `aer_simulator_statevector` calculates the ideal wave function evolution, while `aer_simulator_density_matrix` is utilized for noise-aware simulations as it explicitly accounts for mixed state operations and decoherence effects.

Similarly, in order to see different noise models provided by Qiskit Aer, we can list them as follows:

```
import qiskit_aer.noise as noise

noise_list = [name for name in dir(noise) if not
    name.startswith('_')]
for n in noise_list:
    print(n)
```

Then, a list of available noisy models is shown. Below are some of them:

- `depolarizing_error`
- `amplitude_damping_error`
- `phase_damping_error`
- `PauliError`
- `thermal_relaxation_error`

For example, `depolarizing_error` simulates the loss of quantum information by applying random Pauli operators. `amplitude_damping_error` models energy dissipation, corresponding to relaxation from the excited state $|1\rangle$ to the ground state $|0\rangle$. `thermal_relaxation_error` incorporates physical T_1 and T_2 relaxation times to capture decoherence effects in superconducting quantum hardware.

Once we have selected one of the previous noisy models, we can easily incorporate it into a PennyLane device using the PennyLane—Qiskit plugin. For example, the codes for constructing a noisy single-qubit device by incorporating a depolarizing error into the qubit on the RY gate, and for defining a simple quantum circuit that can be trained on the simulated data are described as follows:

```
import pennylane as qml
from qiskit_aer import AerSimulator, noise

# Here we define our selected depolarizing noise
prob_1 = 0.002
error_1 = noise.depolarizing_error(prob_1, 1)

noise_model = noise.NoiseModel()
noise_model.add_all_qubit_quantum_error(error_1,
    ['ry'])

backend = AerSimulator()

# Here PennyLane use AerSimulator noisy model
dev = qml.device(
    "qiskit.aer",
    wires=1,
    backend=backend,
    noise_model=noise_model,
    shots=128
)

# Here we define our simple quantum circuit
@qml.qnode(dev, interface="autograd")
def circuit(x, weight):
    qml.RY(x[0], wires=0)
    qml.RY(weight, wires=0)
    return qml.expval(qml.PauliZ(0))
```

By displaying the error probabilities for the used qubit, we can easily verify with just a few lines of code that the depolarizing noise model has been correctly applied to our quantum circuit, as shown in Fig. 2.

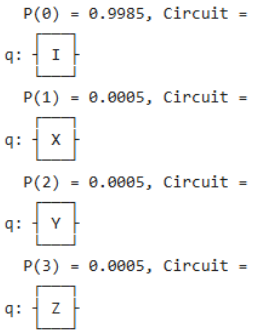

Figure 2. Visualization of the depolarizing noise model confirming that our AerSimulator noise model was successfully used

This setup and example demonstrate how to simulate noisy quantum circuits using the Aer Simulator, which is essential for developing and testing quantum machine learning models under realistic noise conditions.

3.4. Fake Backend Simulation

The PennyLane–Qiskit plugin supports IBM's fake backends, which simulate realistic noise and device constraints. The following is how to list the available fake backends:

```
import qiskit_ibm_runtime.fake_provider as
    fake_provider

fake_backend_classes = [name for name in dir(
    fake_provider) if isinstance(getattr(
    fake_provider, name), type)]

for fb in fake_backend_classes:
    print(fb)
```

It then also shows us a long list of available Fake backends. Below are some of them:

- `FakeManilaV2`
- `FakeAthensV2`
- `FakeArmonkV2`
- `FakeMelbourneV2`
- `FakeTorontoV2`

Once we have chosen a fake backend to simulate a real quantum device with realistic hardware constraints, we can easily use it with the PennyLane–Qiskit plugin. How to do it with the `FakeManilaV2` fake backend applied to the same simple quantum circuit and our simulated data is illustrated as follows:

```
from qiskit_ibm_runtime.fake_provider import
    FakeManilaV2

backend = FakeManilaV2()

# Here PennyLane use our selected Fake Backend
dev = qml.device("qiskit.remote", wires=1, backend
    =backend, shots=128)

# Here we define our simple quantum circuit
@qml.qnode(dev, interface="autograd")
def circuit(x, weight):
    qml.RY(x[0], wires=0)
    qml.RY(weight, wires=0)
    return qml.expval(qml.PauliZ(0))
```

By displaying the properties of the fake backend (FakeManilaV2) for the used qubit, we can easily verify with just a few lines of code that the backend has been correctly applied in our quantum circuit, as shown in Fig. 3.

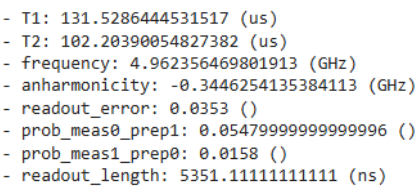

Figure 3. Visualization of the FakeManilaV2 characteristics confirming that this backend was successfully used, showing realistic noise features such as T1/T2 relaxation times, readout errors, etc., that mimic the behavior of a real IBM device

This setup demonstrates how to simulate quantum circuits with realistic constraints using a fake backend, thus enabling the development and testing of quantum machine learning models under practical conditions.

We have seen how to implement the noisy simulation as well as the fake backend. We now move to the final backend implementation, i.e., the real quantum backend.

3.5. Real Backend via Remote Device

To run quantum circuits on actual IBM Quantum hardware, we first need to connect to the IBM Quantum Runtime service using our API token. This allows access to real quantum devices available on the IBM Quantum platform:

```
from qiskit_ibm_runtime import
    QiskitRuntimeService

api_token = "YOUR_API_TOKEN"

# Here we initialize the Qiskit Runtime service
service = QiskitRuntimeService(
    token=api_token,
    channel='ibm_quantum_platform'
)

# We can list available backends on IBM Quantum
print(service.backends())
```

This above command outputs available real quantum devices, allowing us to select a backend for our experiments:

- `ibm_fez`
- `ibm_marrakesh`
- `ibm_torino`

To optimize for shorter queue times, we can select the least busy real backend with the following method:

```
backend = service.least_busy(operational=True,
    simulator=False, min_num_qubits=1)
```

In this study, the `least_busy()` method selected `ibm_torino` as the backend at the time of execution.

Once the backend is selected, we can create a PennyLane device connected to this real quantum backend using the PennyLane—Qiskit plugin. How to define a simple single-qubit quantum circuit and train it on simulated financial data using this real backend is shown as follows:

```
import pennylane as qml
from qiskit_ibm_runtime import
    QiskitRuntimeService

# Here we define our IBM Quantum Runtime setup
api_token = "API_TOKEN"
service = QiskitRuntimeService(
    token=api_token,
    channel='ibm_quantum_platform',
    instance='open-instance'
)
backend = service.least_busy(operational=True,
    simulator=False, min_num_qubits=1)

# Here PennyLane use our selected real Backend
dev = qml.device("qiskit.remote", wires=1, backend
    =backend, shots=128)

# Here we define our simple quantum circuit
@qml.qnode(dev, interface="autograd")
def circuit(x, weight):
    qml.RY(x[0], wires=0)
    qml.RY(weight, wires=0)
    return qml.expval(qml.PauliZ(0))
```

By displaying the properties of the selected real backend, we can easily verify with just a few lines of code that the backend has been correctly applied in our quantum circuit, as illustrated in Fig. 4.

```
- T1: 227.48171537346406 (us)
- T2: 254.72744286580593 (us)
- readout_error: 0.1473388671875 ()
- prob_meas0_prep1: 0.151123046875 ()
- prob_meas1_prep0: 0.1435546875 ()
- readout_length: 1560 (ns)
```

Figure 4. Visualization of the ibm torino backend characteristics confirming that this backend was successfully used, showing realistic noise features such as T1/T2 relaxation times, readout errors, etc., that reflect the behavior of a real IBM device

This setup demonstrates how to run quantum circuits on real IBM Quantum hardware with realistic constraints, enabling the development and testing of quantum machine learning models under practical conditions.

The test RMSE obtained across the three backends are 1.75 for the Aer simulator, 1.79 for the fake backend (FakeManilaV2), and 1.94 for the real hardware (ibm_torino). These values are reported for illustrative purposes only, as the primary objective of this work is to demonstrate the accessibility and correct functioning of the PennyLane–Qiskit plugin across different execution environments, rather than to optimize predictive performance.

4. Conclusion and Discussion

The objective of this work was to provide a practical and hands-on introduction to the PennyLane—Qiskit plugin focusing specifically on three key device categories that are most relevant for realistic quantum

experimentation: noisy simulators, fake backends, and real quantum hardware. A central motivation was to reduce the practical barrier to access and use realistic quantum devices, i.e., an essential step for the sustainable development and broader adoption of quantum computing technologies.

In this work, we successfully demonstrated how to incorporate realistic noise models using the Aer simulator enabling controlled experiments under noisy conditions. We also showed how to use fake backends which reproduce hardware constraints to facilitate benchmarking without requiring direct hardware access. Finally, we connected to real IBM Quantum devices remotely and run quantum circuits on physical hardware to experience authentic device limitations and noise. This multi-level methodology facilitates robust development and testing of quantum machine learning models. The PennyLane—Qiskit plugin is proven to be an easy and accessible interface, thus significantly easing the transition from theory to application. By lowering the entry barrier, it empowers both newcomers and experts to experiment and innovate efficiently within the quantum computing ecosystem.

In conclusion, it is essential to keep quantum computing tools simple and easy to use. As quantum hardware gets better and more powerful, tools like PennyLane, Qiskit, and others help to make quantum technology available to more people. Along with these tools, creating simple and up-to-date tutorials is essential to help people use them as efficiently as possible. By making it easier for researchers and developers from all kinds of backgrounds to try out and build with quantum computers, we can speed up progress and find new useful applications. This wider access is key to ensure the growth of quantum computing in a way that includes everyone and leads to real benefits in many areas. In the end, making these tools simple and open helps more people use and benefit from the power of quantum technology.

References

[1] D. Herman, C. Googin, X. Liu, Y. Sun, A. Galda, I. Safro, M. Pistoia, and Y. Alexeev, "Quantum computing for finance," *Nature Reviews Physics*, vol. 5, pp. 450–465, Jul. 2023.

[2] S. Lee, J. Lee, H. Zhai, Y. Tong, A. M. Dalzell, A. Kumar, P. Helms, J. Gray, Z.-H. Cui, W. Liu, M. Kastoryano, R. Babbush, J. Preskill, D. R. Reichman, E. T. Campbell, E. F. Valeev, L. Lin, and G. K.-L. Chan, "Evaluating the evidence for exponential quantum advantage in ground-state quantum chemistry," *Nature Communications*, vol. 14, p. 1952, Apr. 2023.

[3] R. Ur Rasool, H. F. Ahmad, W. Rafique, A. Qayyum, J. Qadir, and Z. Anwar, "Quantum computing for healthcare: a review," *Future Internet*, vol. 15, no. 3, p. 94, Feb. 2023.

[4] S. K. Subramani, M. Selvi, and S. Svn, "Review of security methods based on classical cryptography and quantum cryptography," *Cybernetics and Systems*, pp. 302–320, Jan. 2023.

[5] F. Tennie and T. N. Palmer, "Quantum computers for weather and climate prediction: the good, the bad, and the noisy," *Bulletin of the American Meteorological Society*, vol. 104, no. 2, pp. 488–500, Feb. 2023.

[6] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, E. Peters, O. Lockwood, A. Skolik, S. Jerbi, V. Dunjko, M. Leib, M. Streif, D. Von Dollen, H. Chen, S. Cao, R. Wiersema, H.-Y. Huang, J. R. McClean, R. Babbush, S. Boixo, D. Bacon, A. K. Ho, H. Neven, and M. Mohseni, "Tensorflow quantum: A software framework for quantum machine learning," *arXiv preprint arXiv:2003.02989*, Mar. 2020. [Online]. Available: https://arxiv.org/abs/2003.02989

[7] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, "Q#: Enabling scalable quantum computing and development with a high-level dsl," in *Proceedings of the Real World Domain Specific Languages Workshop (RWDSL)*, no. 7. ACM, Feb. 2018, pp. 1–10.

[8] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, N. Killoran, and C. Weedbrook, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, Nov. 2018. [Online]. Available: https://arxiv.org/abs/1811.04968

[9] S. Shapiro, "Hybrid quantum-classical machine learning with pennylane: A comprehensive guide for computational research," *arXiv preprint arXiv:2511.14786*, Nov. 2023. [Online]. Available: https://arxiv.org/abs/2511.14786

[10] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with qiskit," *arXiv preprint arXiv:2405.08810*, May 2024. [Online]. Available: https://arxiv.org/abs/2405.08810

[11] M. E. Sahin, E. Altamura, O. Wallis, S. P. Wood, A. Dekusar, D. A. Millar, T. Imamichi, A. Matsuo, and S. Mensa, "Qiskit machine learning: an open-source library for quantum machine learning tasks at scale on quantum hardware and classical simulators," *arXiv preprint arXiv:2505.17756*, May 2025. [Online]. Available: https://arxiv.org/abs/2505.17756

[12] J. C. Hull, *Options, Futures, and Other Derivatives*, 10th ed. Pearson, 2018.

[13] P. Wilmott, *Paul Wilmott on Quantitative Finance*. John Wiley & Sons, 2006.