ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Project report

[Machine learning]

Report of the Machine Learning project about Quantum State Tomography. The code is available on Github here [1]

VIRASAK Hugo / PANSERA Tony / MINGUEZ Niels

December 13, 2025

Contents

1 General definitions

1.1 Quantum physics and quantum computing

In classical physics, we describe a system with exact numbers (like position x and velocity v). In quantum mechanics, we describe a system using a vector in an Hilbert space.

To make things easier to read, physicists use a specific format called Dirac Notation (or "bra-ket" notation). We write a column vector as a "ket":

$$|\psi\rangle \quad \text{(This represents the state vector)} \tag{1}$$

This vector lives in a complex vector space. For a simple 2-level system (like a qubit), this is just a 2-dimensional complex space ($\mathbb{C}^2$).

1.1.1 Superposition

The most famous feature of quantum mechanics is superposition. Because our states are just vectors, we can add them together. If a system can be in state $|0\rangle$ or state $|1\rangle$, it can also be in a linear combination of both:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2}$$

Here, α and β are simply complex numbers. The system is truly in both states at once, weighted by these coefficients.

1.1.2 Measurement and Probabilities

Here is the catch: we can't "see" the superposition. When we measure the system, we never observe the vector $|\psi\rangle$. Instead, nature forces the system to choose. It "collapses" into one of the basis states ($|0\rangle$ or $|1\rangle$).

How does it choose? It relies on the coefficients α and β:

- The probability of seeing $|0\rangle$ is $|\alpha|^2$.
- The probability of seeing $|1\rangle$ is $|\beta|^2$.

Because total probability must be 100%, the state vector must always be normalized (length = 1):

$$|\alpha|^2 + |\beta|^2 = 1 \tag{3}$$

1.2 Quantum Computing

Quantum computing is simply the art of manipulating these vectors to solve problems.

- **The Qubit:** A "Qubit" is just the physical version of the 2-level vector described above. While a classical bit is either 0 or 1, a qubit uses the coefficients α and β to explore a vast space of possibilities between 0 and 1.

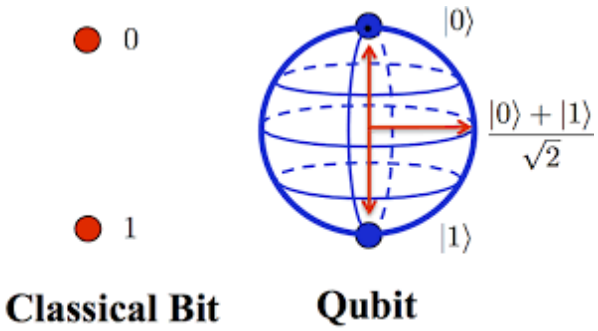


Figure 1: Classical bit vs. Qubit

- **Gates are Matrices:** To compute, we need to change the state of the qubit. In math terms, we multiply the state vector by a matrix. Since the vector must stay normalized, these matrices must be unitary. For example, there is the Hadamard gate (H) that induce superposition with the matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4}$$

Now that we understand that quantum states are just vectors in a complex space, we can look at one specific physical property used to build qubits: Spin.

1.3 Spin

In quantum mechanics, **spin** is an **intrinsic form of angular momentum** carried by elementary particles, atoms, and nuclei. It does not come from a physical rotation, particles such as electrons are not literally spinning spheres, but it behaves mathematically as if they had an internal angular momentum. Spin is a fundamental quantum property, just like mass or charge. It characterizes how a particle responds to **rotations** and **magnetic fields**. A particule has a predefined constant spin.

For example, an electron, proton, or neutron has a spin value of $\frac{1}{2}$, meaning that when you measure its spin along any direction, you can only obtain two possible results: "up" or "down". A photon, on the other hand, has spin 1, meaning it has three possible orientations $(-1, 0, +1)$, although only two are physically observable (left and right circular polarizations).

- **Spin and Measurement** When you measure a spin-$\frac{1}{2}$ particle along an axis (say the z-axis), it can only be found in one of two states:

$$|\uparrow\rangle_z = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |\downarrow\rangle_z = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

  These correspond to spin "up" and "down" along that axis. If you measure along another axis (like x or y), the same particle's spin states become superpositions of $|\uparrow\rangle_z$ and $|\downarrow\rangle_z$:

$$|\psi\rangle = \alpha|\uparrow\rangle_z + \beta|\downarrow\rangle_z, \quad \text{with } |\alpha|^2 + |\beta|^2 = 1.$$

- **Mathematical Representation**
  For spin-$\frac{1}{2}$ systems (1 qubit), spin observables are represented by the **Pauli matrices**:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

  Each Pauli matrix corresponds to a spin measurement along one of the three spatial axes (x, y, or z). Measuring σ_z checks whether the spin is aligned $(+1)$ or anti-aligned (-1) with the z-axis.

- **Spin on the Bloch Sphere**
  A qubit's spin state can be visualized as a vector on the **Bloch sphere**, where:
  - The north pole represents $|0\rangle = |\uparrow\rangle_z$,
  - The south pole represents $|1\rangle = |\downarrow\rangle_z$,
  - Any point on the sphere corresponds to a superposition of these two states.

  The direction of the Bloch vector corresponds to the mean spin direction:

$$\vec{r} = \langle\psi|\vec{\sigma}|\psi\rangle.$$

1.4 Bloch sphere

- **General qubit pure state:**
  A qubit is a normalized vector in a 2D complex Hilbert space:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1$$

  As α and β are both complex numbers, we have a total of 4 missing values (both the real and imaginary number of α and β). With the probability condition of $|\alpha|^2 + |\beta|^2 = 1$, we can remove one degree of freedom, and as the **global phase** has no physical meaning (multiplying the entire state by a complex phase $e^{i\gamma}$ doesn't change the physical state, i.e. the probabilities), we can remove another degree of freedom.

- **Parameterization of the qubit state:**
  As we have removed 2 degrees of freedom, we can always express α and β using only 2 variables as:

$$\alpha = \cos\left(\frac{\theta}{2}\right), \quad \beta = e^{i\phi}\sin\left(\frac{\theta}{2}\right)$$

  where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$. Thus:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

  This form corresponds exactly to a **point on a unit sphere** with spherical coordinates (θ, ϕ). Note that removing the normalization condition gives us the 3 necessary variables needed in spherical coordinates in 3 dimensions.

- **From qubit state to Bloch vector:**
  We define the **Bloch vector** $\vec{r}$ as the expectation value of the **Pauli vector operator**:

$$\vec{r} = \langle\psi|\vec{\sigma}|\psi\rangle$$

  where $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$, and the Pauli matrices are:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Computing this gives:

$$\vec{r} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix}$$

Hence, $|\vec{r}| = 1$ for a pure state, the qubit lies **on the surface of the unit sphere**.

- **Density matrix representation:**
  The qubit density matrix can be written as:

$$\rho = |\psi\rangle\langle\psi| = \frac{1}{2}\left(I + \vec{r}\cdot\vec{\sigma}\right)$$

For mixed states, $|\vec{r}| < 1$, and the vector lies **inside** the Bloch sphere.

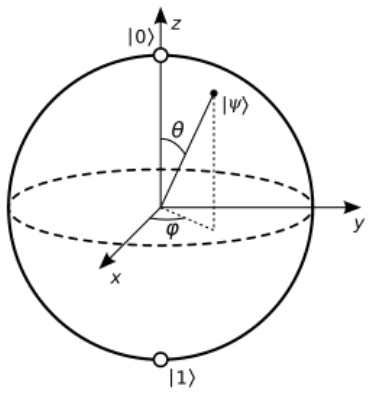


Figure 2: Qubit representation

1.5 Pure state

A **pure state** in quantum mechanics is a state that contains **complete information** about a quantum system, it is described by a **single state vector** (or wavefunction) $|\psi\rangle$ in a Hilbert space.

$$|\psi\rangle = \sum_i c_i|i\rangle, \quad c_i \in \mathbb{C}, \quad \sum_i |c_i|^2 = 1$$

- **Density matrix form**
  A pure state can be written as:

$$\rho = |\psi\rangle\langle\psi|$$

It satisfies:

$$\rho^2 = \rho \quad \text{and} \quad \text{Tr}(\rho^2) = 1$$

This distinguishes it from a **mixed state**, where $\text{Tr}(\rho^2) < 1$.

- **Physical meaning**
  - A pure state represents **maximal knowledge** of the system.
  - Measurement probabilities are **deterministically** derived from $|\psi\rangle$ via Born's rule:
    $$P(a_i) = |\langle a_i|\psi\rangle|^2$$
  - In contrast, a **mixed state** represents **statistical uncertainty** over several possible pure states.

- **Example:**
  For a qubit:
  $$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$
  is a **pure state**, lying **on the surface** of the Bloch sphere.

1.6 Mixed state

A mixed state is a statistical ensemble of pure states, it represents incomplete information about the quantum system.

Mathematically, a mixed state is described by a **density matrix** ρ that is **not a projector** (i.e. $\rho^2 \neq \rho$).

- **Formal definition**
  A mixed state is given by:
  $$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$$
  where:
  - $|\psi_i\rangle$ are **pure states** (normalized vectors in Hilbert space),
  - $p_i \geq 0$ and $\sum_i p_i = 1$,
  - ρ is **Hermitian**, **positive semidefinite**, and has **unit trace**.

  This represents a **classical probability distribution** over quantum states: we do not know *which* pure state the system is in, only the probabilities p_i.

A common misconception about mixed state is the difference between pure state and superposition. A particle in superposition is in several different states at the same time and has a *complex probability* of being measured in one of those state. We have $\sum |c_i|^2 = 1$. On the other hand, mixed state is a *classical probability* of being in a pure state, with $\sum p_i = 1$. We can have both mixed state and superposition at the same time.

- **Mathematical properties**
  - $\rho^2 \neq \rho$
  - $\mathrm{Tr}(\rho) = 1$
  - $0 < \mathrm{Tr}(\rho^2) < 1$

  The value of $\mathrm{Tr}(\rho^2)$ quantifies the **purity** of the state. Pure states satisfy $\mathrm{Tr}(\rho^2) = 1$; mixed states satisfy $\mathrm{Tr}(\rho^2) < 1$.

- **Physical interpretation**
  A mixed state arises when:
  - The system is in a **probabilistic mixture** of several possible pure states.
  - The system is **entangled** with another system and we **trace out** the environment.
  - There is **decoherence** due to noise or measurement.

- **Example (qubit)**
  Pure state:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \Rightarrow \rho = |\psi\rangle\langle\psi| = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\mathrm{Tr}(\rho^2) = 1$$

  Mixed state:

$$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\mathrm{Tr}(\rho^2) = \frac{1}{2} < 1$$

  This mixed state represents **classical uncertainty** — the qubit is either $|0\rangle$ or $|1\rangle$ with 50% probability, but not in a coherent superposition.

- **On the Bloch sphere**
  - **Pure states:** points **on the surface** ($|\vec{r}| = 1$).
  - **Mixed states:** points **inside the sphere** ($|\vec{r}| < 1$).
  - The **maximally mixed state** $\rho = \frac{I}{2}$ lies at the **center**.

1.7 Density matrix

The **density matrix**, usually denoted by ρ, is a mathematical object that provides a complete description of a quantum system, whether it is in a **pure state** or a **mixed state**. Unlike the ket notation $|\psi\rangle$, which only describes pure states, the density matrix can represent statistical mixtures of different quantum states. It plays a central role in quantum computing, quantum machine learning, and quantum state tomography.

- **Definition:** The density matrix ρ is defined as

$$\rho = \sum_i p_i \left|\psi_i\right\rangle \left\langle\psi_i\right|,$$

  where p_i are probabilities ($p_i \geq 0$, $\sum_i p_i = 1$) and $|\psi_i\rangle$ are the possible quantum states of the system.
- **Pure state case:** If the system is in a single well-defined state $|\psi\rangle$, then $\rho = |\psi\rangle \langle\psi|$. In this case, $\rho^2 = \rho$ and $\mathrm{Tr}(\rho^2) = 1$.
- **Mixed state case:** When the system is in a statistical mixture of several states, $\rho^2 \neq \rho$ and $\mathrm{Tr}(\rho^2) < 1$. Mixed states arise from incomplete knowledge or interactions with an environment (decoherence).
- **Properties:** The density matrix always satisfies:
  - $\rho = \rho^\dagger$ (Hermitian)
  - $\rho \geq 0$ (positive semi-definite)
  - $\mathrm{Tr}(\rho) = 1$ (normalized)
- **Dimension:** For a system of n qubits, the density matrix ρ has size $2^n \times 2^n$, since the Hilbert space dimension is $d = 2^n$.
- **Physical meaning:** Each element ρ_{ij} represents the probability amplitudes and coherences between the basis states $|i\rangle$ and $|j\rangle$. The diagonal terms represent population probabilities, while the off-diagonal terms encode quantum coherences.
- **Measurement probabilities:** The probability of measuring an outcome corresponding to a projector M is given by

$$P(M) = \mathrm{Tr}(M\rho).$$

1.8 Decoherence

The open quantum system

In the previous sections, we treated the quantum system (like a qubit) as if it were isolated from the rest of the universe. In an ideal world, a quantum state $|\psi\rangle$ evolves according to the Schrödinger equation and maintains its superposition indefinitely.

However, in the real world, no system is perfectly isolated. A quantum system is constantly interacting with its environment (thermal fluctuations, electromagnetic radiation, magnetic fields, etc.). This interaction is what leads to **decoherence**.

Decoherence is the physical process by which a quantum system loses its quantum properties (specifically, its interference capabilities) and becomes classical. It is the mechanism that explains why we do not see macroscopic objects, like cats or tables, in superposition.

Entanglement with the environment

Decoherence happens because the system becomes entangled with the environment.

Imagine a qubit initially in a superposition:

$$|\psi\rangle_{sys} = \alpha|0\rangle + \beta|1\rangle$$

When it interacts with the environment $|E\rangle$, the combined state evolves into an entangled state:

$$|\Psi\rangle_{total} = \alpha|0\rangle|E_0\rangle + \beta|1\rangle|E_1\rangle$$

Here, $|E_0\rangle$ is the state of the environment if the qubit is $|0\rangle$, and $|E_1\rangle$ is the state of the environment if the qubit is $|1\rangle$.

Because we cannot measure the vast and complex environment, we must "trace it out" mathematically. This loss of information about the environment causes the information in our qubit to degrade. The environment effectively "measures" the qubit, forcing it to choose a state.

Effect on the quantum state

To understand the effect of decoherence, we must look at the density matrix.

Recall that for a pure state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, the density matrix is:

$$\rho = \begin{pmatrix} |\alpha|^2 & \alpha\beta^* \\ \alpha^*\beta & |\beta|^2 \end{pmatrix} \tag{5}$$

The off-diagonal elements ($\alpha\beta^*$ and $\alpha^*\beta$) are called **coherences**. They represent the quantum interference between $|0\rangle$ and $|1\rangle$. It is these terms that allow for quantum algorithms to work.

Under the influence of decoherence, these off-diagonal terms decay over time, usually exponentially at a rate γ:

$$\rho(t) = \begin{pmatrix} |\alpha|^2 & \alpha\beta^* e^{-\gamma t} \\ \alpha^*\beta e^{-\gamma t} & |\beta|^2 \end{pmatrix} \tag{6}$$

Classical mixture

As time passes ($t \to \infty$), the coherence terms vanish ($e^{-\gamma t} \to 0$). The density matrix becomes diagonal:

$$\rho_{final} = \begin{pmatrix} |\alpha|^2 & 0 \\ 0 & |\beta|^2 \end{pmatrix} \tag{7}$$

This is a **mixed state**. Notice that:

- The probabilities (diagonal terms) $|\alpha|^2$ and $|\beta|^2$ remain (in a pure dephasing scenario).
- The quantum superposition is gone.
- The system behaves exactly like a classical coin flip: it is either 0 or 1, we just don't know which.

Visualizing decoherence

On the Bloch sphere, decoherence corresponds to the Bloch vector $\vec{r}$ shrinking towards the center of the sphere.

- The vector shrinks toward the $|0\rangle$ pole (loss of energy)

- The vector shrinks toward the z-axis (loss of phase information), turning a pure state on the equator into a mixed state on the axis.

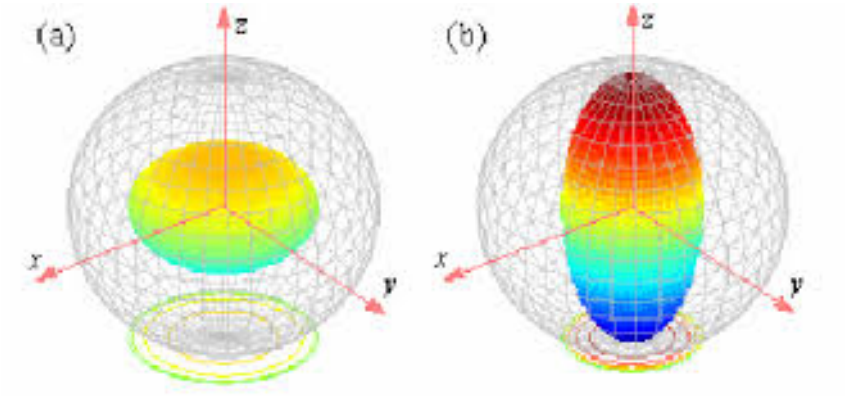


Figure 3: **Visualizing decoherence.** The state moves from the surface (pure) to the interior (mixed).

2 Problem formalization

2.1 Quantum State Tomography (QST)

Quantum states cannot be visualized directly. It does not seem concrete at the first view. Measuring them changes them. Therefore, to determine an unknown quantum state, we must collect statistics from many identical copies and try to reconstruct the state mathematically, with the probabilities measured with certain operators. This reconstruction process is called **Quantum State Tomography**.

We can make an analogy with a dice: let's suppose our system is a rolling dice, and we want to find the probability of making any number. We will roll the dice a lot of time, like 1000 times, and deduce the probabilities of making a given number based on our measurements. This is the same in Quantum State Tomography, we try to reconstruct our original system with the probabilities obtained from measuring the same state over and over.

Why is this reconstruction process so important? The main reason is that quantum computers are currently very sensitive and error-prone. When a scientist sends instructions to a quantum processor, they cannot be 100% sure that the machine actually performed the operations correctly.

We can think of Quantum State Tomography as the ultimate quality control check or debugging tool. In a classical computer, you can pause a program and look at the memory to see if variables hold the correct values. In the quantum world, because "looking" destroys the information, we cannot do this. QST is the tool that allows us to see inside the machine.

This serves two main purposes. First, it allows for verification. If an algorithm is supposed to produce a specific quantum state, QST tells us how close the actual result came to that goal (often measured by a score called "fidelity"). Second, it helps with hardware calibration. If the reconstructed state looks distorted compared to the original plan, the specific shape of that distortion gives engineers clues about what went wrong physically: perhaps a control laser was slightly off-target or there was too much noise in the environment. Without QST, we would be operating blindly, unable to distinguish between a bad code and a broken machine.

QST can be performed using several approaches such as linear inversion, Bayesian estimation, or maximum likelihood estimation. But those approaches may not give very exact results, or have a very long runtime

depending on the number of qubits in out system. This is why we are trying to use Machine Learning to see if it can bring any results to the QST problematic.

2.2 Description of our project

In this project, we work with one-qubit states for simplicity, but the methods generalize to larger systems. Our objectives are twofold:

1. **Classification: pure and mixed states:** Using machine learning techniques, we train models to determine whether a qubit state is pure or mixed. It doesn't really has anything to do with QST, although it can differentiate pure states and mixed states as a faster way to tell us if the machine is working properly or if there is any decoherence interfering (returning a mixed state while the original state is supposed to be pure). It is more of a general introduction to applying Machine Learning to Quantum measurements, and we will also look if using a Quantum kernel for SVC can improve the results.
2. **Regression: Predicting the Density Matrix:** We aim to predict the components of the density matrix directly, effectively learning the full quantum state from measurement data. This is the main point of our project, seeing if applying Machine Learning algorithms to Quantum data can help us reconstructing the original state with good fidelity. We will compare our results with Maximum Likelihood Estimation (MLE).

All these tasks rely on simulated measurement outcomes using the Pauli operators $\{X, Y, Z\}$ as described earlier.

3 Dataset generation algorithm

In this work, a single qubit dataset is generated by simulating the full tomographic pipeline:

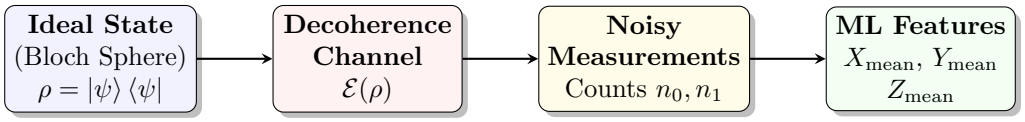


Figure 4: **Dataset generation pipeline.** Overview of the data generation pipeline: from the theoretical quantum state to the classical feature vector used for the SVM.

Each row of the dataset corresponds to one qubit state and one full experiment along the X, Y and Z bases.

3.1 Sampling of ideal pure states

We first sample ideal pure states uniformly on the Bloch sphere. For each state, we draw a qubit vector u following :

$$u \sim \mathcal{U}([-1, 1]),$$
$$\theta_{\text{ideal}} = \arccos(u) \in [0, \pi],$$
$$\phi_{\text{ideal}} \sim \mathcal{U}([0, 2\pi)).$$

We intuitively think of sampling θ uniformly in $[0, \pi]$. However, this would lead to a biased distribution with a dense concentration of points near the poles (north and south) of the Bloch sphere. This occurs because the differential surface element of a sphere, $dA = \sin(\theta)d\theta d\phi$, shrinks as θ approaches 0 or π.

To achieve a truly uniform distribution over the sphere's surface, we must account for this by sampling the "vertical height" $z = \cos(\theta)$ uniformly. By drawing u uniformly between -1 and 1 and setting $\theta = \arccos(u)$, we effectively correct for the surface area curvature, ensuring the point density remains constant everywhere on the sphere.

The corresponding ideal Bloch vector $\vec{r}_{\text{ideal}} = (X_{\text{ideal}}, Y_{\text{ideal}}, Z_{\text{ideal}})$ is then

given by:

$$X_{\text{ideal}} = \sin(\theta_{\text{ideal}})\cos(\phi_{\text{ideal}}),$$
$$Y_{\text{ideal}} = \sin(\theta_{\text{ideal}})\sin(\phi_{\text{ideal}}),$$
$$Z_{\text{ideal}} = \cos(\theta_{\text{ideal}}).$$

By construction, $\|\vec{r}_{\text{ideal}}\| = 1$, so the initial state is pure.

3.2 Decoherence channel

An optional decoherence channel is then applied to the ideal Bloch vector to obtain the *real* physical state $\vec{r}_{\text{real}} = (X_{\text{real}}, Y_{\text{real}}, Z_{\text{real}})$.

If decoherence is disabled, we simply set:

$$\vec{r}_{\text{real}} = \vec{r}_{\text{ideal}}.$$

If decoherence is enabled with a global parameter $\lambda \in [0, 1]$ (denoted `decoherence_level` in the code), then for each state:

1. With probability λ, the state is affected by noise; otherwise it remains pure and unchanged.
2. For a noisy state, we first draw a random "shrink strength"

$$s \sim \mathcal{U}([0, \lambda]), \qquad \text{base\_factor} = 1 - s.$$

3. We then introduce anisotropy along each Bloch axis by drawing

$$\boldsymbol{\alpha} = (\alpha_x, \alpha_y, \alpha_z), \qquad \alpha_i \sim \mathcal{U}([0.5, 1.5]).$$

4. The final contraction factors are

$$f_i = \text{clip}(\text{base\_factor} \cdot \alpha_i,\, 0, 1), \qquad i \in \{x, y, z\},$$

and the real Bloch vector is given by

$$X_{\text{real}} = f_x X_{\text{ideal}}, \tag{8}$$
$$Y_{\text{real}} = f_y Y_{\text{ideal}}, \tag{9}$$
$$Z_{\text{real}} = f_z Z_{\text{ideal}}. \tag{10}$$

The Euclidean norm $\|\vec{r}_{\text{real}}\| \leq 1$ encodes the purity of the state: $\|\vec{r}_{\text{real}}\| = 1$ corresponds to a pure state, while $\|\vec{r}_{\text{real}}\| < 1$ corresponds to a mixed state

produced by the decoherence channel.

Also, to simplify the creation of the dataset for the classification, we can also choose the proportion of pure and mixed states for balance purpose.

3.3 Measurements and statistical noise

The measurement data then is performed by measuring the Pauli observables σ_X, σ_Y and σ_Z on n_{shots} identically prepared copies of the same state. For a qubit with Bloch vector $\vec{r}_{\text{real}}$, the expectation values of the three observables are

$$\langle \sigma_X \rangle = X_{\text{real}}, \qquad \langle \sigma_Y \rangle = Y_{\text{real}}, \qquad \langle \sigma_Z \rangle = Z_{\text{real}}.$$

Each measurement outcome is ± 1. For a given axis (e.g. X), the probability to obtain the outcome $+1$ is

$$p_X(+) = \frac{1 + X_{\text{real}}}{2}, \qquad p_X(-) = 1 - p_X(+).$$

We then draw n_{shots} independent outcomes $A_k^X \in \{+1, -1\}$ according to this Bernoulli distribution, and define the empirical mean

$$X_{\text{mean}} = \frac{1}{n_{\text{shots}}} \sum_{k=1}^{n_{\text{shots}}} A_k^X.$$

The same procedure is applied independently for Y and Z to obtain Y_{mean} and Z_{mean}. These empirical means contain the finite-shot statistical noise and converge to $(X_{\text{real}}, Y_{\text{real}}, Z_{\text{real}})$ as $n_{\text{shots}} \to \infty$.

3.4 Structure of a single dataset row

For each generated state, we store in the dataset:

- **Features** used as input to the machine learning models:

$$(X_{\text{mean}}, Y_{\text{mean}}, Z_{\text{mean}}).$$

- **Continuous labels** representing the underlying physical state:

$$(X_{\text{real}}, Y_{\text{real}}, Z_{\text{real}}).$$

- **Optional ideal quantities** (for analysis and visualization only):

$$\theta_{\text{ideal}},\ \phi_{\text{ideal}},\ X_{\text{ideal}},\ Y_{\text{ideal}},\ Z_{\text{ideal}}.$$

- **Optional label** telling if the state is pure or mixed (for classification)

As a result, the dataset can provide both the noisy data (features) and the underlying ground-truth Bloch vector, which allows us to train and evaluate regression, classification or MLE-based reconstruction algorithms under controlled noise and decoherence.

The goal of the creation of the dataset is to simulate as accurately as possible the measurements of quantum states in a real-world laboratory. Theoretical simulations assume perfect conditions, but real quantum hardware is always noisy. We simulate this noise using a physics-based approach rather than just adding random errors.

Our method reproduces two key behaviors of real quantum chips:

- **Loss of Purity:** Real noise causes the quantum state to "decay." We simulate this by shrinking the state vector so its length is less than 1 ($\|\vec{r}\| < 1$). This turns a perfect "pure state" into a noisy "mixed state."
- **Asymmetric Noise:** In real experiments, noise affects some axes more than others (for example, phase errors are often more frequent than bit-flip errors). We calculate specific factors $(\alpha_x, \alpha_y, \alpha_z)$ to shrink the sphere unevenly, deforming it into an ellipsoid.

Finally, our algorithm includes a safety check (clipping) to ensure the vector length never exceeds 1. This guarantees that every generated data point represents a physically valid quantum state that could exist in nature.

4 Visualization of the generated datasets

4.1 Our first classification task before

Our first objective was to predict the states $1, 0, -i, i, -, +$ on the axes of the Bloch Sphere, given for the features the number of the shot, the positions on the coordinates (X, Y, Z) and the type of spin (spin-up $+1$ or spin-down -1), but the prediction performance was very low, thus the interpretation of the scores was not relevant in our project. We also wanted to make a noise on the measurements of the dataset to try to find back the initial values, but it was very difficult and not helpful.

4.2 Binary Classification

The goal of the classification dataset is to determine whether a state is **pure** or **mixed**. Each row corresponds to one quantum state for which we simulate measurements using the Pauli operators X, Y, and Z. The dataset contains the following fields:

- **X_mean, Y_mean, Z_mean**: Empirical averages of measurement outcomes in each Pauli basis. These are the noisy estimates obtained from a finite number of shots.
- **X_real, Y_real, Z_real**: The real Bloch vector components of the state before measurement noise. These represent the "true" values and are impacted by decoherence.
- **theta_ideal, phi_ideal**: Spherical coordinates of the state on the Bloch sphere before noise. These angles generate the ideal Bloch vector.
- **X_ideal, Y_ideal, Z_ideal**: The noiseless theoretical Bloch vector components, the original state.
- **bloch_radius_real**: The real purity radius of the state. A pure state has radius 1, and a mixed state has radius < 1.
- **is_pure**: Boolean value indicating whether the state is pure according to the generation process.
- **label_purity**: The classification label for training: 0 for mixed and 1 for pure.

We created a qubit purity functions for the creation of the classification dataset specifically, with parameters the number of shots (n_shots), the number of rows (n_states_total with a state purity for each input) and the proportion of unbalanced dataset (mixed_proportion varying between

0 and 1).

The dataset features selected for binary classification are the means X_mean, Y_mean and Z_mean of the Bloch sphere to predict the label_purity (0 or 1). For the researcher, only the values of the measurements given a number of shots are known. He cannot find the ideal or real (after a certain noise) values of the quantum particle.

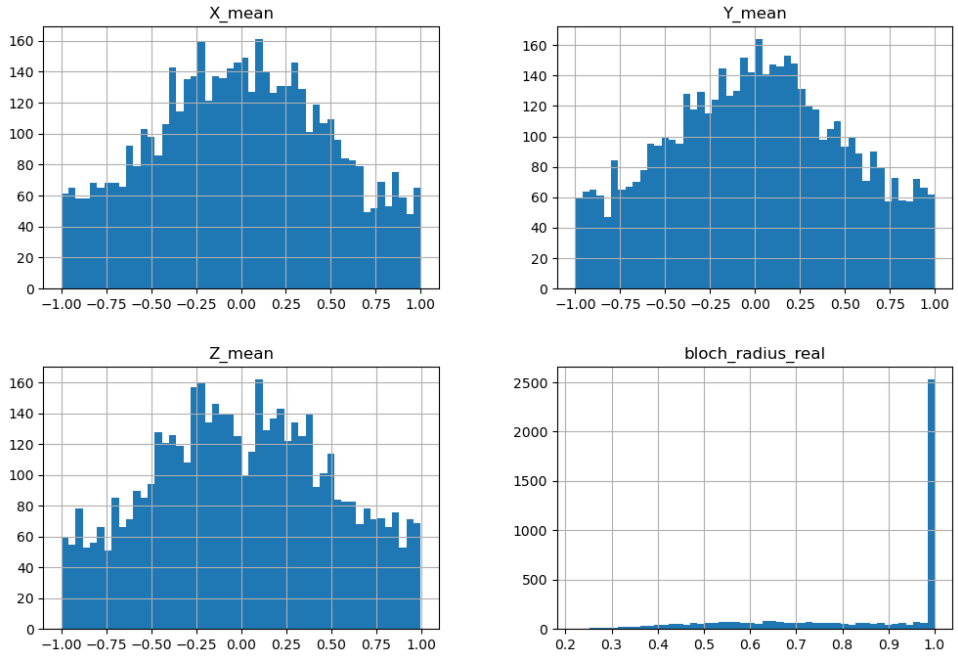


Figure 5: **Classification dataset histograms.** Representation of X_mean, Y_mean, Z_mean and bloch_radius_real

In the case of the purity-classification dataset, states are labelled depending on whether they are pure or mixed. Figure 6 displays the two classes on the Bloch sphere.

We can see a comparaison here with the number of shots per measurement: usually pure states cluster very close to the surface, while mixed states appear inside the sphere, but the less measurements we make, the less defined is the limitation between pure states and mixed states.

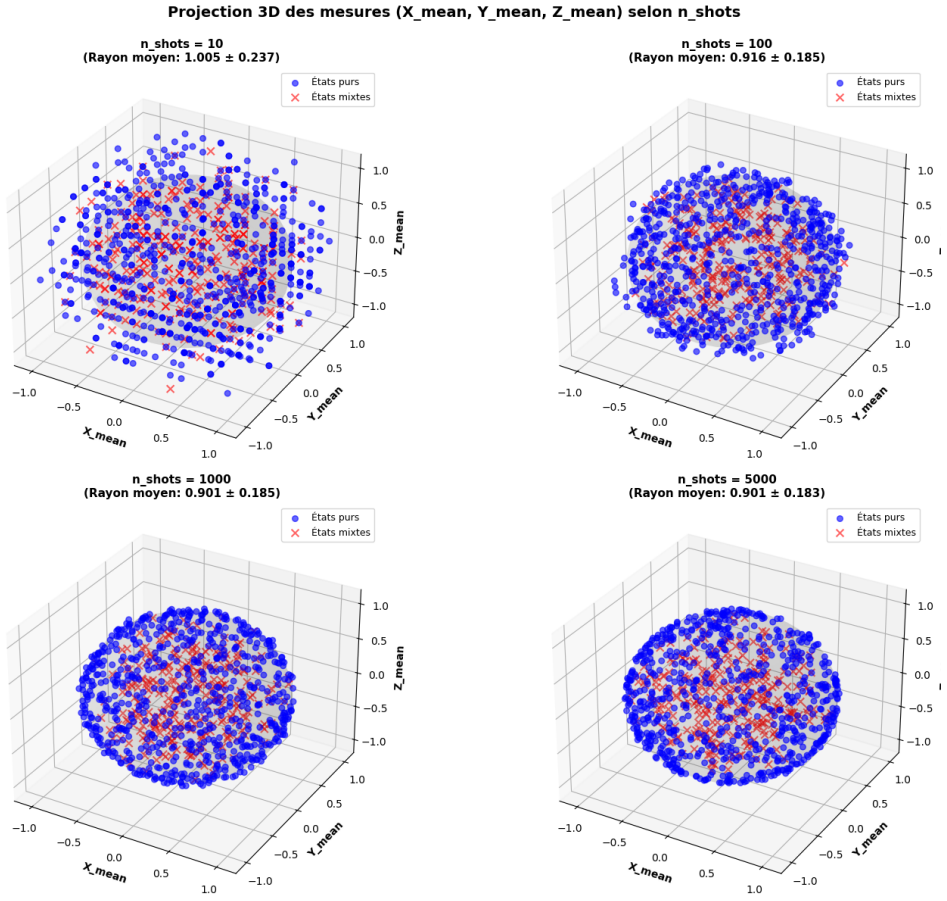


Figure 6: R
epresentation on the Bloch sphere based on number of shots. Pure states (red) and mixed states (blue).

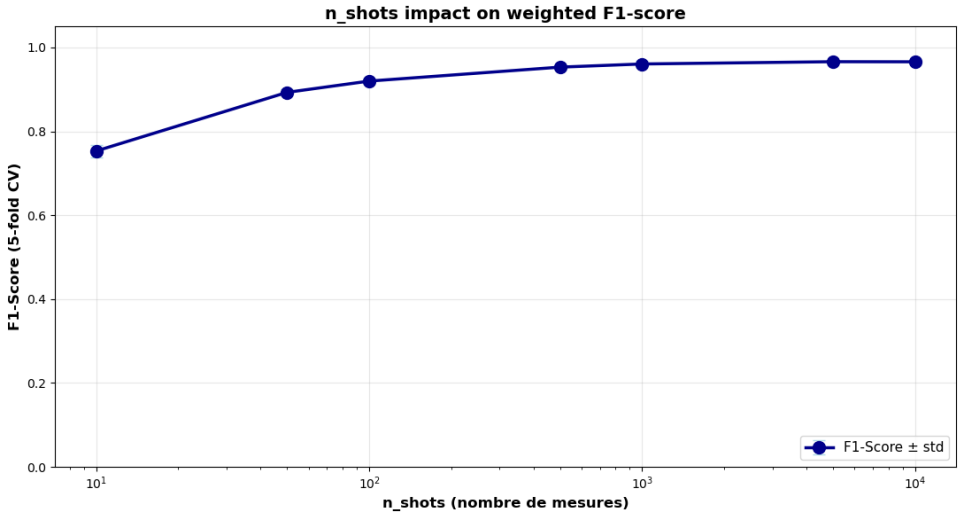


Figure 7: P
erformance of the model on different numbers of shots.

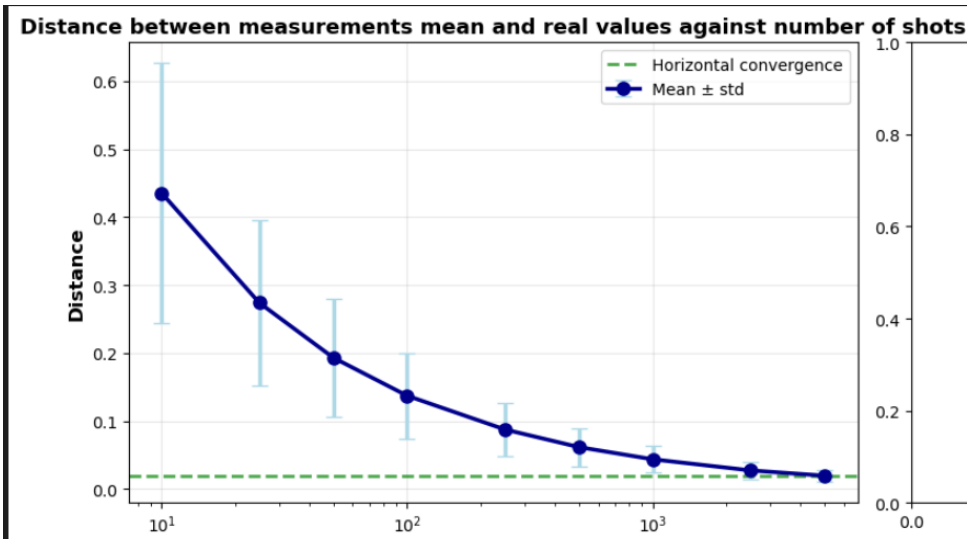


Figure 8: F
inding the optimal number of shots with the Strong Law of Big Numbers

4.3 Regression

The regression dataset is used to directly predict the structure of the density matrix or the Bloch vector of the state. The model tries to learn a continuous mapping from noisy measurements to the ideal underlying quantum state.

The fields are:

- **X_mean, Y_mean, Z_mean**: Noisy empirical measurement averages from a finite number of shots. These are the features used by the regression model.
- **theta_ideal, phi_ideal**: The target angles describing the true quantum state.
- **r_ideal**: True purity radius of the state.
- **cos_phi_ideal, sin_phi_ideal**: Auxiliary targets useful for stable learning of angular variables.
- **cos_theta_ideal, sin_theta_ideal**: Same purpose: they allow the model to learn angles without discontinuities.
- **X_ideal, Y_ideal, Z_ideal**: Ideal Bloch components without statistical noise nor decoherence
- **X_real, Y_real, Z_real**: Ideal vector that may be affected by decoherence, but no statistical noise.

For a dataset of 5000 states, with 1000 shots per axis and a decoherence level of 0.6, we have those values, shown in graphs:

Figure 9 shows the distribution of the ideal Bloch vectors $(X_{\text{ideal}}, Y_{\text{ideal}}, Z_{\text{ideal}})$ sampled uniformly on the sphere. As expected, the points cover the entire Bloch sphere with uniform density.

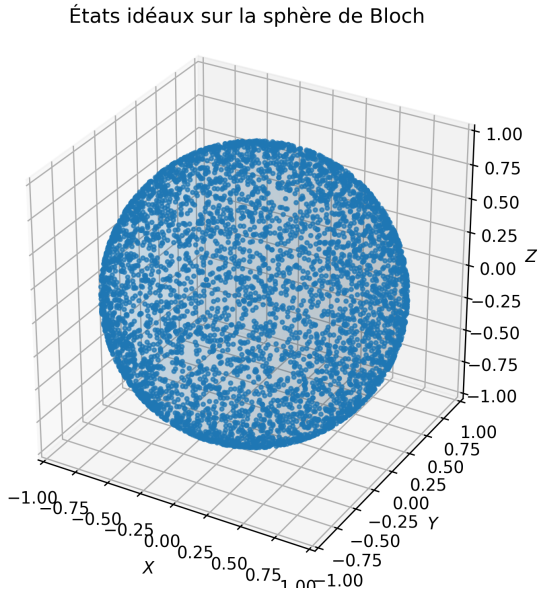


Figure 9: **Ideal pure states uniformly sampled on the Bloch sphere.**

Decoherence shrinks the Bloch vectors. Figure 10 displays the same set of states after application of the decoherence channel described in the previous section. The colour encodes the Bloch radius:

$$r = \sqrt{X_{\text{real}}^2 + Y_{\text{real}}^2 + Z_{\text{real}}^2}.$$

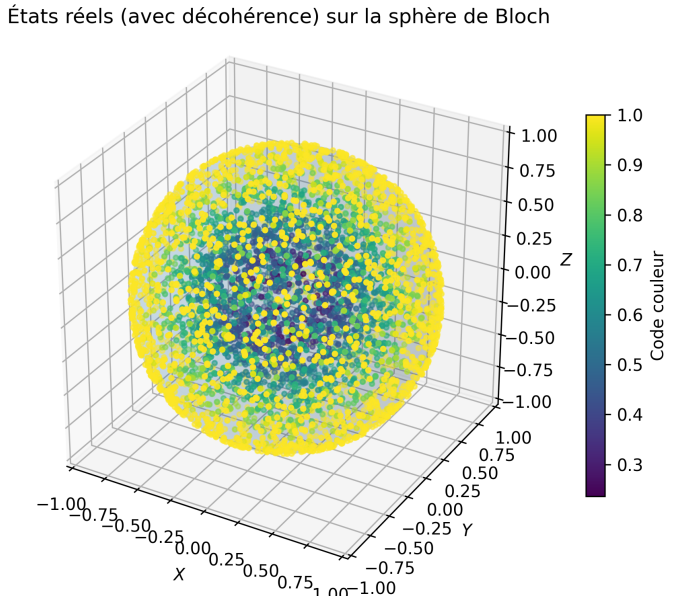


Figure 10: **Real Bloch vectors after anisotropic decoherence.** Colour represents the Bloch radius (purity).

The corresponding histogram of Bloch radii (Figure 11) shows how the decoherence level influences the spread of mixed states.

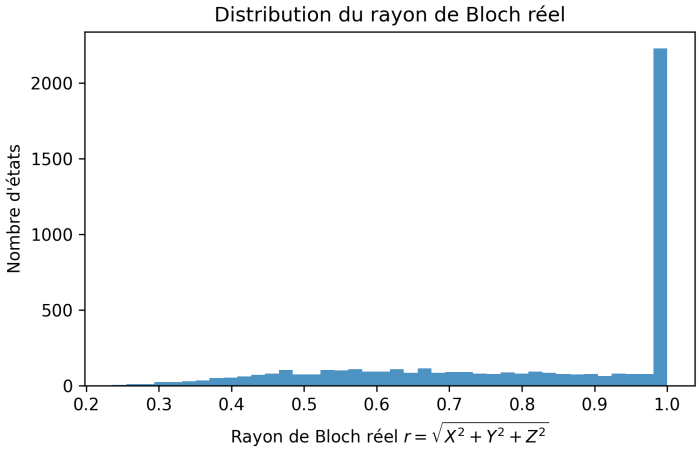


Figure 11: **Distribution of the Bloch radius r for the real states.**

4.4 Maximum Likelihood Estimation

The MLE dataset is designed for reconstructing quantum states from measurement statistics using maximum likelihood estimation. Each row describes a qubit state through both noisy measurement data and its ideal theoretical parameters. The same dataset as for regression has been used so there is no need for further visualization.

If we delve deeper into how MLE works, we might remark that this dataset is not complete as we need other parameters to use the MLE algorithm, particularly the probability of $|\uparrow\rangle$ $P(+1)$ and $|\downarrow\rangle$ $P(-1)$ using the different Pauli operators and the number of occurrence in our measurements. But this data can easily be calculated as we have **X_mean, Y_mean, Z_mean**:

Let us denote by X_{mean} the empirical average of the measurement outcomes in the X-basis:

$$X_{\text{mean}} = \langle X \rangle.$$

If $P(+1)$ is the probability of obtaining the outcome $+1$ (spin up), and $P(-1)$ is the probability of obtaining -1 (spin down), then

$$X_{\text{mean}} = (+1)\,P(+1) + (-1)\,P(-1) = P(+1) - P(-1).$$

Since probabilities sum to one,

$$P(+1) + P(-1) = 1,$$

we can eliminate $P(-1)$:

$$X_{\text{mean}} = P(+1) - (1 - P(+1)) = 2P(+1) - 1.$$

Solving for $P(+1)$ gives

$$P(+1) = \frac{1 + X_{\text{mean}}}{2}.$$

So from the value of **X_mean** alone, we can find $P(+1)$ and $P(-1)$. From this, if we have the number of measurements done for one operator, we can find the number of occurrence simply by multiplying this number to its probability: $P(+1) \times n_{+1}$. As we created our own datasets, we obviously have this number so we can find both the probabilities and the number of occurrence associated with each state.

A more detailed explanation of the MLE method is provided in Appendix 9.5.

5 Training model on classification

We decided to go for an other task: predict the type of state : pure or mixted state.

5.1 Purity classification (pure vs mixed)

5.1.1 General problem: detect if a state is pure or mixed from noisy measurements

In the regression part, we tried to reconstruct the state (predict the Bloch vector). Here, the goal is simpler: we only want to classify the state as pure or mixed.

The intuition is geometric:

- A **pure** one-qubit state lives on the surface of the Bloch sphere ($\|\vec{r}\| \approx 1$).
- A **mixed** state is inside the sphere ($\|\vec{r}\| < 1$).

But in practice we do not observe the exact $\vec{r}$. We only have noisy estimates from finite shots:

$$X_{\text{mean}} \approx \langle \sigma_x \rangle, \quad Y_{\text{mean}} \approx \langle \sigma_y \rangle, \quad Z_{\text{mean}} \approx \langle \sigma_z \rangle,$$

and the noise depends a lot on n_{shots}. The mean coordinates are the result of the average of a certain number of shots on one qubit. So, the boundary between pure and mixed is not perfectly clean in the measured space.

We model this as a binary classification problem:

- **Input** X**:** measured features mean $X_{\text{mean}}, Y_{\text{mean}}, Z_{\text{mean}}$.
- **Target** y**:** `label_purity` $\in \{0, 1\}$ where we take 1 = pure and 0 = mixed.

We use an SVM classifier (SVC) because it is basically made to learn a decision boundary with a maximum-margin logic, and we will work on the best kernels to improve the prediction.

5.1.2 Decision boundary: what the SVC is actually learning

An SVC predicts a label using the sign of a score function:

$$\hat{y} = \text{sign}\big(f(\mathbf{x})\big),$$

where SV are the support vectors (the key training points near the boundary), and $K(\cdot, \cdot)$ is the kernel.

We are particularly interested in the rbf and 2nd degree polynomial kernels in this task.

- RBF kernel is very flexible local boundary, since the distribution of the X,Y and Z is unimodal and is similar to a Gaussian curve.
- Polynomial degree 2 learns quadratic objects. This is interesting because $\|\vec{r}\|^2 = X^2 + Y^2 + Z^2$ is quadratic as the equation of a sphere of r radius. The kernel would be interesting to separate the pure category on the sphere surface and the mixted category inside.

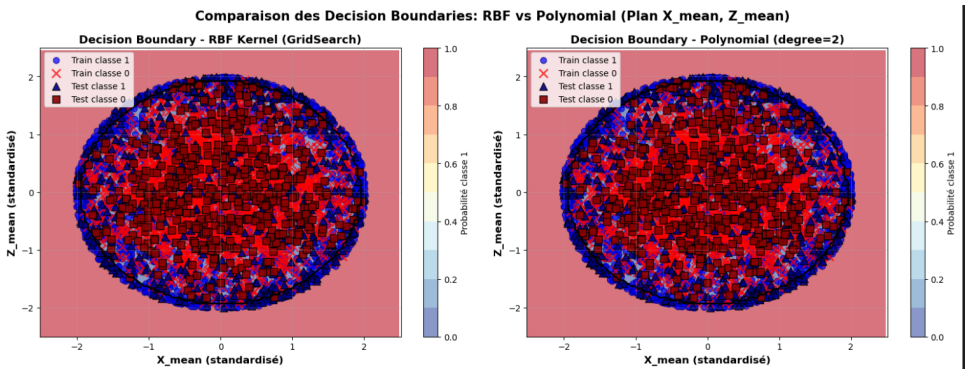


Figure 12: **Classification boundaries for polynomial degree 2 and Gaussian kernels.** The two show clear boundaries between pure states (blue) and mixed states (red).

So the classification is really: learn a surface in the measured feature space that best separates "pure-like" vs "mixed-like" clouds under shot noise.

5.1.3 Analysis framework (what we tested and why)

To make the classification part solid, we tested the following points:

- **Data sanity check (convergence with shots):** we verify that the measured Bloch estimates converge to the real geometry when n_{shots} increases (law of large numbers idea, error decreases when shots increase).
- **Kernel competition:** compare linear vs RBF vs polynomial (degree 2) and see which one matches the Bloch geometry best.
- n_{shots} **impact:** measure how the classification performance (F1-score) improves from very noisy regimes (low shots) to almost-ideal regimes (high shots).

- **Class imbalance:** when the dataset is not 50/50 pure/mixed, we test oversampling (SMOTE) and check if it improves precision/recall for the minority class.
- **Decision boundary visualization:** we plot predicted labels on a 2D slice (for example the X-Z plane with $Y = 0$) to see the boundary shape instead of only trusting metrics.

5.2 Experimental protocol and machine learning pipeline

5.2.1 Data pipeline

We generate the dataset with simulated physics and simulated measurements:

1. **State generation:** sample random 1-qubit states, with a controlled proportion of mixed states (`mixed_proportion`).
2. **Measurement:** simulate projective measurements with n_{shots} shots to get noisy means $(X_{\text{mean}}, Y_{\text{mean}}, Z_{\text{mean}})$.
3. **Features:**
$$\mathbf{x} = (X_{\text{mean}}, Y_{\text{mean}}, Z_{\text{mean}})$$
4. **Label:** $y =$ `label_purity`, where 1 = pure and 0 = mixed.
5. **Preprocessing:** apply standard scaling (SVMs are distance-based, scaling matters a lot). The features will be identically distributed on the same scale.
6. **Training:** train an SVC, and tune hyperparameters using cross-validation.

5.2.2 Evaluation metrics

Because the classes can be imbalanced, accuracy alone is not relevant, since it does not represent the quality of the model to predict the minor label. We mainly use the F1-score:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

When the dataset is imbalanced, we track weighted F1 to avoid being overly optimistic. We display in the classification report a confusion matrix to see exactly what kind of errors we make (pure predicted as mixed vs mixed predicted as pure).

5.2.3 *Exploration parameters*

Instead of a single dataset, we sweep key parameters as we can see in Table 1:

Parameter	**Tested values (example)**	**Why we vary it**
$n_{\text{states\_total}}$	5000	Large enough to see stable trends and avoid pure luck.
n_{shots}	$[10, 50, 100, 500, 1000, 2500, 5000, 10000]$	Controls statistical noise: low shots = hard classification; high shots = near-ideal.
mixed_proportion	$[0.3, 0.5]$	Tests imbalance and realism (experiments are rarely perfectly balanced).

Table 1: **Classification sweeps.** Parameters varied to test robustness across noise and class balance regimes.

We can then compare the effect of those parameters on our model.

5.2.4 *Parameters optimization using Grid Search SVC*

After plotting several graphs to find the best number of shots for the best performance of the model, we got the idea to determine the best combinations of number of shots and SVC hyperparameters, using a big GridSearch. For each n_{shots}, we run a grid search with cross-validation (CV=5) and score with weighted F1 (due to different values of mixed proportion):

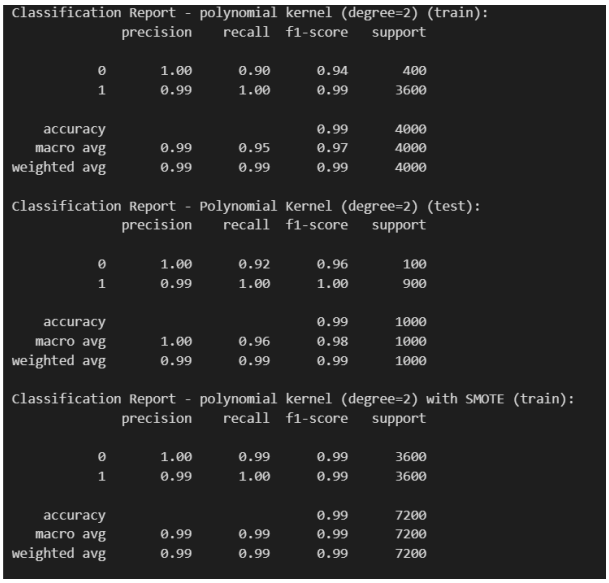
- **Regularization** C**:** controls the trade-off between margin size and training errors.
- **Kernel:** {linear, rbf, poly (degree=2)}.
- γ**:** {scale, auto}, controls the curvature of the boundary.

We show an example of best parameters for balanced dataset with mixed_proportion=0.5:

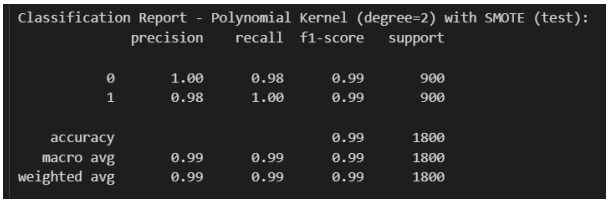
Best n_shots: 5000
Best SVC PARAMETERS:

- C: 10.0
- kernel: poly
- gamma: scale
- degree: 2

F1-Score (weighted): 0.9776 ± 0.0031



```
Classification Report - polynomial kernel (degree=2) (train):
              precision    recall  f1-score   support

           0       1.00      0.90      0.94       400
           1       0.99      1.00      0.99      3600

    accuracy                           0.99      4000
   macro avg       0.99      0.95      0.97      4000
weighted avg       0.99      0.99      0.99      4000

Classification Report - Polynomial Kernel (degree=2) (test):
              precision    recall  f1-score   support

           0       1.00      0.92      0.96       100
           1       0.99      1.00      1.00       900

    accuracy                           0.99      1000
   macro avg       1.00      0.96      0.98      1000
weighted avg       0.99      0.99      0.99      1000

Classification Report - polynomial kernel (degree=2) with SMOTE (train):
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      3600
           1       0.99      1.00      0.99      3600

    accuracy                           0.99      7200
   macro avg       0.99      0.99      0.99      7200
weighted avg       0.99      0.99      0.99      7200
```



```
Classification Report - Polynomial Kernel (degree=2) with SMOTE (test):
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       900
           1       0.98      1.00      0.99       900

    accuracy                           0.99      1800
   macro avg       0.99      0.99      0.99      1800
weighted avg       0.99      0.99      0.99      1800
```

5.3 Rebalancing dataset using SMOTE

We also test **SMOTE** for data augmentation when the dataset is imbalanced, to avoid a model that mostly learns "predict the majority class". In practice, this improved the minority-class precision/recall and made the F1-score more meaningful as we will see later.

Finally, we summarize the main trend we saw:

- **More shots $\Rightarrow$ higher F1-score** because the measured Bloch estimates become less noisy.
- Polynomial degree 2 often makes sense geometrically, because the "inside vs surface" structure is strongly linked to quadratic terms like $X^2 + Y^2 + Z^2$.

5.3.1 Example of performance results

As we varied different dataset according to the proportion of the mixed label, we show you only one of the results with machine learning tools.

With mixed_proportion=0.1

Interpretation: THE SMOTE balancing method increased the 0 label detection to almost perfection with F1-score. We verified there is no over-

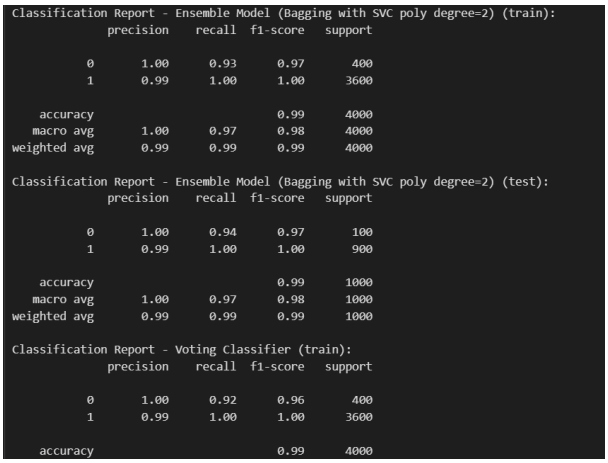


Figure 13: Enter Caption

fitting: the F1 score almost did not change on the train and test sets in both cases with or without SMOTE.

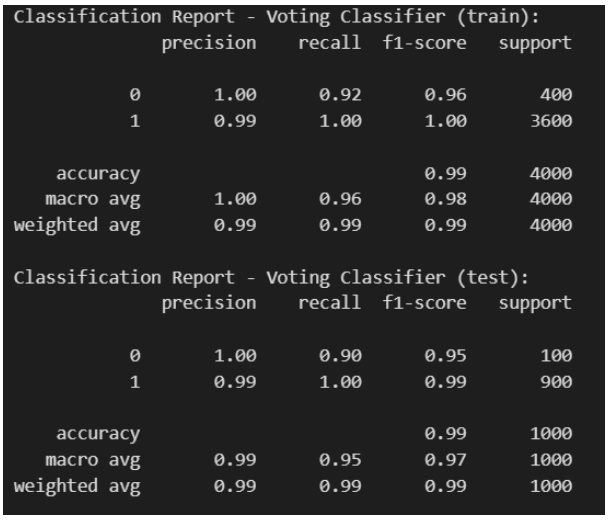
5.4 Using Ensemble learning for practice

Bagging with SVC is employed using a second-degree polynomial kernel in order to reduce the variance of the classifier. Each base SVC is trained on a different bootstrap sample of the training dataset, corresponding to slightly different realizations of experimental measurement noise. The final prediction is obtained by aggregating the outputs of all base classifiers. This approach stabilizes the learned quadratic decision boundary associated with the purity condition of quantum states and improves robustness against statistical fluctuations in the measurement data.

A soft Voting Classifier is used to combine heterogeneous models. For the exercise, we added a Decision Tree classifier and a 2nd degree polynomial-kernel SVC. The SVC captures the global nonlinear structure of the data, which is closely related to the quadratic purity constraint of the Bloch vector, while the DT models local patterns and threshold-based rules in the measurement space. By combining probabilistic predictions from both classifiers, the voting mechanism balances the bias–variance trade-off and yields a more reliable classification of quantum states. This ensemble approach leverages the complementary strengths of both models, leading to improved predictive F1-score and generalization.

5.4.1 Examples of results

Overall, the F1 scores are better for SVC poly degree 2 Bagging and voting classifier with SVC poly degree 2 and decision tree. There is not a



```
Classification Report - Voting Classifier (train):
              precision    recall  f1-score   support

           0       1.00      0.92      0.96       400
           1       0.99      1.00      1.00      3600

    accuracy                           0.99      4000
   macro avg       1.00      0.96      0.98      4000
weighted avg       0.99      0.99      0.99      4000

Classification Report - Voting Classifier (test):
              precision    recall  f1-score   support

           0       1.00      0.90      0.95       100
           1       0.99      1.00      0.99       900

    accuracy                           0.99      1000
   macro avg       0.99      0.95      0.97      1000
weighted avg       0.99      0.99      0.99      1000
```

Figure 14: Enter Caption

significant difference of F1-score between train and test sets, making sure there is still no overfitting.

6 Regression

6.1 Tomography using regression

6.1.1 General problem: ML vs. MLE comparison

As explained in **2. Problem Formalization**, our objective is to reconstruct the density matrix ρ (the true state) from imperfect projective measurements. The current industry standard is Maximum Likelihood Estimation (MLE). While statistically robust, MLE is based on iterative convex optimization, which is computationally exponentially expensive as the number of qubits increases, creating issues regarding scalability and latency.

In this project, we reformulate tomography as a machine learning regression problem, specifically using Support Vector Regression (SVR).

- **Principle:** Instead of optimizing a likelihood function for each measurement, we train a model to learn the inverse function of the measurement process.
- **Data:** The model takes as input the expectation values of observables ($\langle\sigma_x\rangle, \langle\sigma_y\rangle, \langle\sigma_z\rangle$) affected by shot noise, and predicts the exact components of the Bloch vector.

The central objective is to compare the ML approach with the MLE standard on a simplified single-qubit system. We aim to answer the following questions:

- **Performance:** Can ML equal or exceed the precision of MLE, particularly by learning to filter experimental noise?
- **Speed:** Is the gain in inference time sufficient to allow for real-time monitoring?
- **Proof of Concept (PoC):** While limited to one qubit here, this validation is a critical step. If ML proves superior, it becomes a strong candidate for solving multi-qubit tomography, where MLE becomes intractable due to exponential complexity.

6.1.2 Detailed analysis framework

To provide a rigorous answer to these questions, we take the data generated in the dataset generation phase to train our model and conduct an analysis divided into two parts:

- **SVR analysis:**
  1. **Data Efficiency:** How does the training set size (N_{states}) impact the reconstruction fidelity?
  2. **Regime Mapping:** Which Kernel (RBF, Polynomial, Sigmoid) dominates in different physical regimes (low vs. high noise, pure vs. mixed states)?
  3. **Physical Compliance:** Does the SVR capture the concept of "Decoherence" (mixed states), or does it simply project noisy data onto the surface of the Bloch sphere?
  4. **Topology errors:** Is there geometrical bias in the reconstruction of our dataset ?
- **Comparative Benchmark (SVR vs. MLE)**
  1. **Precision Duel:** We perform a direct comparison of Fidelity vs. Shots. We aim to identify the limit where MLE might overtake SVR and quantify the SVR's denoising advantage in low-shot regimes.
  2. **Time vs. Quality:** We quantify the speed-up factor to determine if the trade-off is viable for real-time tomography.

6.2 Experimental protocol and Machine learning pipeline

6.2.1 Data pipeline

We model the tomography problem as a multi-output regression task. The challenge is to map noisy observations to physical quantum states. The data flow follows these specific steps:

1. **Generation (Simulated Physics):** We create random quantum states (Bloch vectors) subject to a parametrized decoherence level (λ).
2. **Measurement (Noisy):** We simulate finite projective measurements (N_{shots}). This introduces realistic statistical fluctuations (shot noise).
3. **Input (X):** The inputs are the observed expectation values of the Pauli operators $\langle\sigma_x\rangle, \langle\sigma_y\rangle, \langle\sigma_z\rangle$.
4. **Target (Y):** The ground truth corresponds to the real components of the Bloch vector r_x, r_y, r_z before measurement.
5. **Preprocessing:** We apply a StandardScaler. This step is crucial for Support Vector Machines (SVM), which are distance-based models and highly sensitive to data scaling.
6. **Model:** A Support Vector Regressor (SVR) wrapped in a MultiOutputRegressor to predict the three coordinates simultaneously.

6.2.2 Evaluation metrics

To assess the quality of the reconstruction, we rely on three complementary metrics:

- **Quantum fidelity/infidelity ($\mathcal{F}$)** This is the standard metric in quantum physics. It measures the overlap between the predicted state ρ_{pred} and the real state ρ_{real}.

$$\mathcal{F}(\rho, \sigma) = \left(\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}} \right)^2$$

  In our implementation, we use the analytical formulation for Bloch vectors to optimize computational efficiency. A fidelity of 1.0 implies a perfect reconstruction. Infidelity is simply 1-$\mathcal{F}$. Which we will use for visualization purposes.

- **Mean Squared Error (MSE)** A standard metric in Machine Learning, measuring the average squared Euclidean distance between the predicted and real Bloch vectors. It serves as the loss function for model optimization.

- **Quantum purity (P)** We created this metric, from documentation online. Purity measures how "mixed" or "noisy" a quantum state is, quantifying the loss of quantum information due to decoherence.

$$P = \mathrm{Tr}(\rho^2) = \frac{1}{2}(1 + \|\vec{r}\|^2)$$

  - **Range:**
    - $P = 1.0$: **Pure state**. The system is perfectly defined (Vector on the Bloch sphere surface).
    - $P = 0.5$: **Maximally mixed state**. Total loss of information (Vector at the center of the sphere).
  - **Usage:** We compare real Purity vs. predicted purity to check if the SVR correctly models the environment and hardware errors.

To ensure our findings are robust, we do not test on a single dataset. Instead, we systematically vary the generation parameters to cover different physical regimes as we can see in Table 2

6.2.3 Model optimization (Grid Search SVR)

For each experimental configuration, we perform a Cross-Validation (CV=3) to find the optimal SVR hyperparameters. This step allows the model to "adapt" to the physics of the problem (e.g., high noise vs. low noise).

Parameter	Tested Values	Scientific Justification
Dataset Size	$[100, 1000, 5000]$	To evaluate generalization capacity and detect overfitting in "Low Data" regimes.
Number of Shots	$[10, \ldots, 5000]$	To simulate different levels of experimental noise, ranging from very noisy to quasi-perfect measurements.
Decoherence	$[0.0, \ldots, 0.9]$	To test if the model correctly learns mixed states (non-unitary) and the loss of purity.

Table 2: **Exploration Parameters.** The variations used to generate the datasets for the benchmark.

- **Kernels:** We test three kernels to capture different relationships:
  - RBF (Radial Basis Function): Standard for local interpolation, well-suited to the geometry of the Bloch sphere.
  - Sigmoid: Mimics neural network activation; tested to see if it better captures non-linear boundaries under heavy noise.
  - Poly: To capture polynomial relationships between measurements. We use the polnome of dergree 2 because we saw in the classification exploration that it was the best of all kernels are reproducing the bloch sphere.
- **Regularization (C):** Controls the trade-off between training error and smoothing. A small C imposes strong regularization, which is useful if the data is very noisy.
- **Sensitivity (ϵ):** Defines the width of the limit within which errors are ignored so the model not to overreact to minor statistical noise.

6.3 Analysis of results

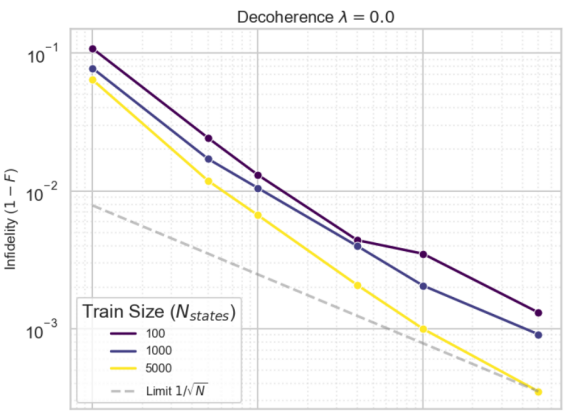
6.3.1 Intrinsic SVR analysis

We begin by analyzing the "engine" of our Machine Learning approach, independent of the standard MLE method.

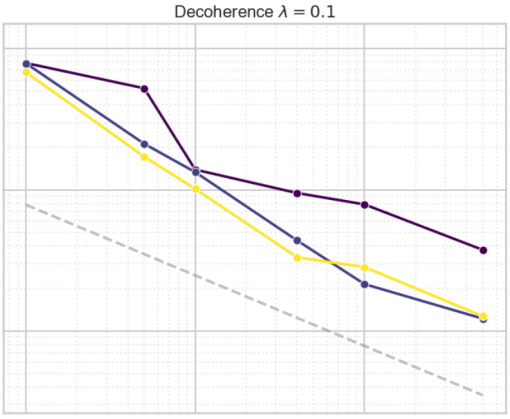
Impact of training set size on decoherence

We examine how the training set size (N_{states}) impacts reconstruction fidelity. Does the model learn continuously, or does it hit a saturation wall where adding more data is useless against measurement noise?

The analysis of the infidelity curves as we can see in Figure 15 across different decoherence levels shows us a consistent pattern:

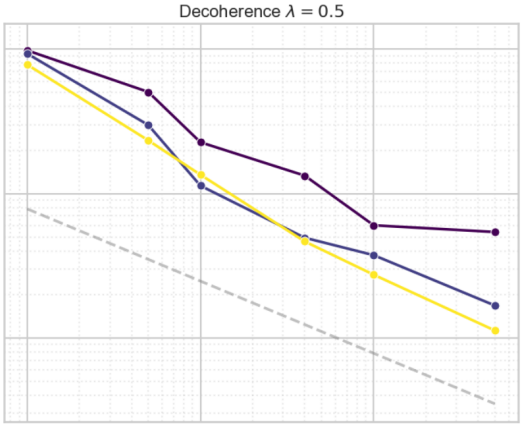


(a) Decoherence $\lambda = 0.0$

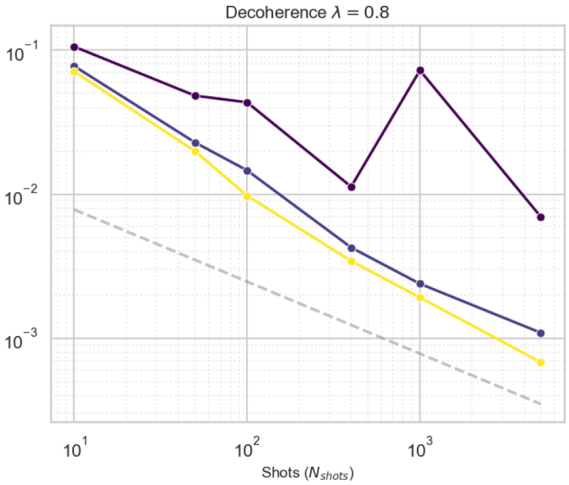


(b) Decoherence $\lambda = 0.1$

(c) Decoherence $\lambda = 0.3$



(d) Decoherence $\lambda = 0.5$



(e) Decoherence $\lambda = 0.8$

Figure 15: Evolution of Infidelity $(1 - \mathcal{F})$ as a function of Shots (Log Scale). The curves represent different training set sizes: **Yellow (5000 states)**, **Blue (1000 states)**, **Black (100 states)**. The dashed line represents the Standard Quantum Limit $(1/\sqrt{N_{shots}})$.

- **Dataset Size Dominance:** Regardless of the noise level λ, the reconstruction precision is directly correlated with the volume of training data. The model trained on 5000 samples (yellow curve) systematically outperforms those trained on 1000 and 100 samples. Also, for $\lambda = 0$, the yellow curve converges onto the function $1/\sqrt{N_{shots}}$, which represents the theoretical limit of prediction accuracy (Standard Quantum Limit).
- **Asymptotic Convergence:** Infidelity decreases as the number of shots increases. For large datasets, this decay follows the theoretical shot noise limit ($1/\sqrt{N_{shots}}$), proving the estimator's efficiency.

Conclusion: The Machine Learning tomography method is robust against decoherence. The quality of reconstruction depends primarily on the richness of the training dataset, allowing the model to reach ideal statistical precision even in the presence of noise.

GridSearch analysis

By analyzing the GridSearch results, we determine if the model behaves differently when facing high noise (low shots) versus high precision.

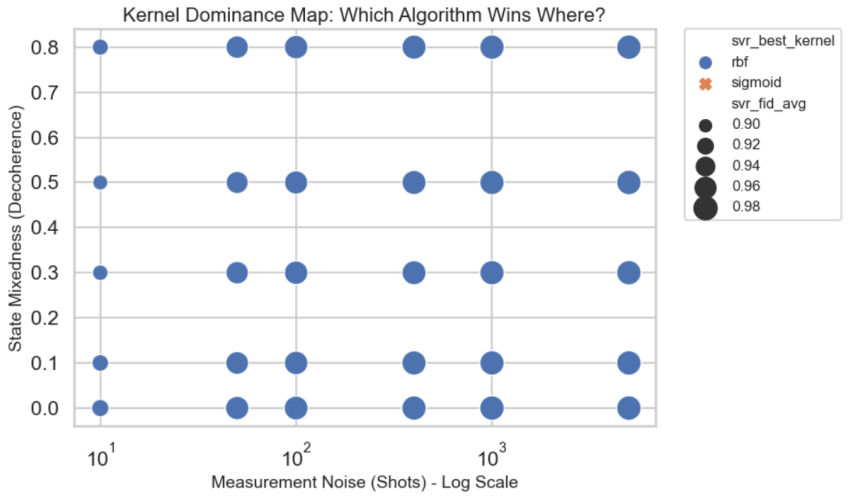
As we can see in Figure 16a, the GridSearch results reveal an overwhelming dominance of the Radial Basis Function (RBF) kernel, selected in **88.9%** of all configurations. This confirms that RBF works well for quantum state reconstruction.

However, a distinct transition occurs when noise changes:

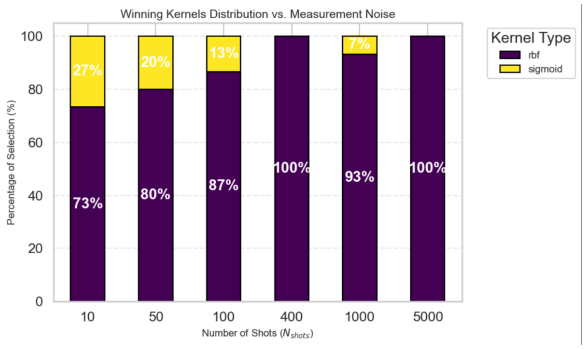
- **Low-Information ($N_{shots} \leq 100$):** The sigmoid kernel emerges as a significant contender, peaking at **26.67%** usage at 10 shots. This suggests that when data is scarce and unreliable, the SVR occasionally abandons the precise local interpolation of the RBF in favor of a global activation boundary, similar to a neural network.
- **High-Precision ($N_{shots} \geq 400$):** As statistical quality improves, the RBF kernel establishes a monopoly (reaching **100%** dominance), confirming it is the optimal topology when sufficient data is available.

The evolution of the hyperparameter C (Figure c) demonstrates that the model correctly "understands" the physics of measurement noise. Recall that a low C implies high regularization (smoothing), while a high C implies trusting the data.

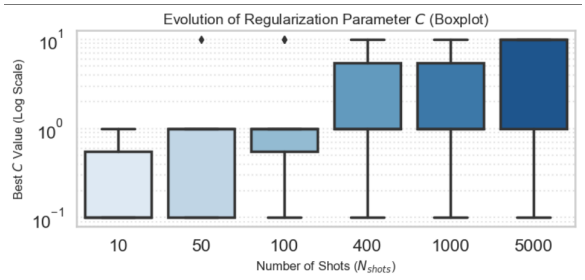
- **High Noise ($N_{shots} = 10$):** The mean C is very low (**0.34**). The model applies strong regularization to smooth out the significant shot noise, prioritizing generalizability over fitting specific, noisy data points.



(a) **Kernel Dominance Map.** SVR kernel selection based on Measurement Noise and State Fidelity. Point size indicates reconstruction fidelity.



(b) **The "Sigmoid Check".** Proportion of chosen kernels vs. Shots.



(c) **Regularization Dynamics.** Evolution of the C parameter distribution vs. Shots.

Figure 16: **GridSearch analysis.**

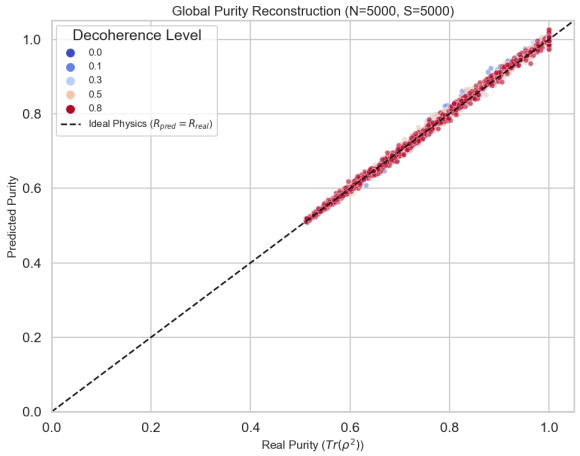
- **High Precision ($N_{shots} = 5000$):** The mean C rises sharply to **6.34**, with a median at the upper bound (10.0). As measurement uncertainty vanishes, the model reduces regularization, effectively "trusting" the experimental inputs to capture the fine-grained details of the density matrix.

Physical sens and denoising

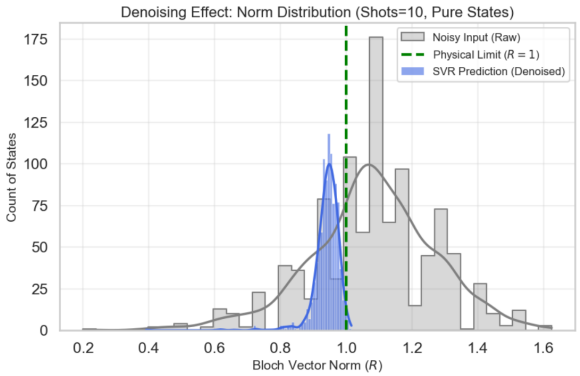
This analysis verifies the SVR's capacity to model non-unitary physical processes. By aggregating predictions across multiple decoherence regimes (from pure states to highly mixed states), we quantify the mapping between the real Bloch vector norm (R_{real}) and the predicted norm (R_{pred}).

This step specifically controls for the "Projection Bias": we verify whether the algorithm incorrectly forces noisy mixed states onto the surface of the

Bloch sphere ($R = 1$) or correctly identifies the shrinkage of the state vector due to environmental noise.



(a) **Purity Reconstruction.** Real vs. Predicted Purity. The data hugs the identity line $y = x$, indicating no bias.



(b) **Physical Validity.** Histogram of Bloch Vector Norms (R) under high noise ($N_{shots} = 10$). Comparison between Raw Data (Grey) and SVR (Blue).

Figure 17: **Physical Compliance Analysis.** Evaluating the SVR's ability to respect quantum bounds and model decoherence.

We first verify if the SVR model correctly learns the non-linear relationship between noisy measurements and quantum state purity ($\mathcal{P} = \mathrm{Tr}(\rho^2)$).

- **Quantitative Observation:** The global average bias is extremely low (-1.80×10^{-4}), and the Mean Squared Error (MSE) remains uniformly low ($\sim 10^{-5}$) across all decoherence regimes.
- **Physical Interpretation:** The perfect linear correlation (Figure a) confirms the SVR acts as an unbiased estimator. It successfully differentiate physical decoherence (loss of information) from statistical noise.

In Figure (b), we evaluate the model's respect for fundamental quantum constraints, which are that a density matrix must be positive and semi-definite, meaning the Bloch vector norm must satisfy $R \leq 1$. We focus on a high-noise scenario ($N_{shots} = 10$) where statistical errors are largest.

- **Raw Data (The Problem):** Due to shot noise, the naive linear inversion of measurement data yields physically impossible states. **71.30%** of the raw states violate physics ($R > 1$), with an average norm of $R \approx 1.08$.
- **SVR Reconstruction (The Solution):** The model acts as a powerful non-linear filter. It reduces the violation rate to only **1.20%**, recentering the average norm to $R \approx 0.94$.

Conclusion: The SVR has implicitly learned the topology of the Hilbert space. Unlike a naive projection that might force all points to the surface, the model operates a quasi-unitary projection". It understands that the probability of a physical state having $R > 1$ is zero. As such, it effectively filters out unphysical statistical noise while preserving physical decoherence, transforming the reconstruction problem into an efficient denoising task.

Spatial distribution of reconstruction error

We map the reconstruction error (Infidelity $1 - \mathcal{F}$) onto the 3D Bloch Sphere to detect geometric biases. We want to know whether the model favors the "easy" eigenstates (Poles) over the complex superposition states (Equator). Note that we plot using a configuration of nshot = 100 and decoherence = 0 in order to have and homogenous but realistic matching lab conditions test.

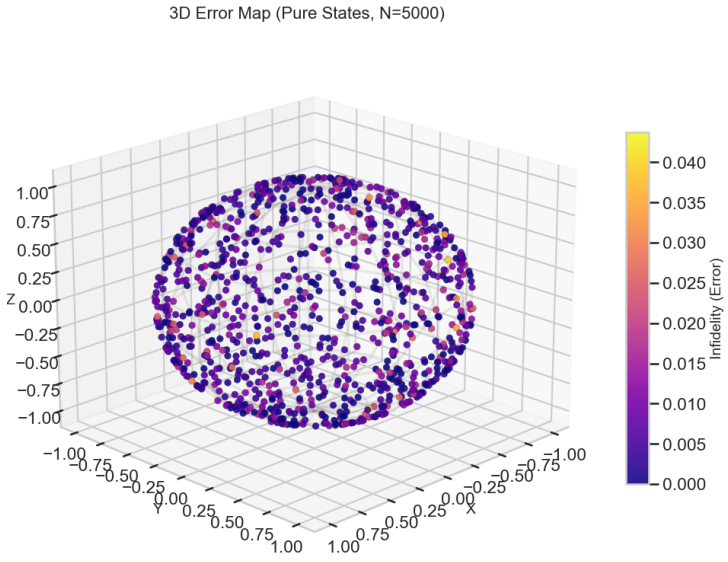


Figure 18: **3D Error Topology.** Heatmap of reconstruction fidelity on the Bloch Sphere. Blue indicates low error; Red/Yellow indicates higher error regions.

Spatial analysis:

- **High precision:** The global mean infidelity is remarkably low (6.67×10^{-3}).
- **Isotropy:** The error difference between the poles ($|Z| > 0.9$, Error ≈ 0.0081) and the equator ($|Z| < 0.2$, error ≈ 0.0079) is statistically negligible. The maximum error point (0.044) is a local outlier, so we can ignore it.

Physical Interpretation: This result is physically significant. In QST, measurements are projective and performed along fixed axes (X, Y, Z). A naive model often overfits these axes, exhibiting "Basis Bias" (high precision for Z-eigenstates, low precision for superpositions).

Our SVR demonstrates Basis Independence. The uniform error distribution proves that the model has internalized the Rotational Symmetry of the Bloch sphere. It does not treat the quantum state as a discrete set of measurement outcomes, but effectively interpolates the continuous geometry of the Hilbert space. It reconstructs complex superpositions $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\phi}|1\rangle)$ with the same fidelity as the computational basis states $|0\rangle$ and $|1\rangle$.

6.3.2 Comparative benchmark (SVR vs. MLE)

We shift the focus to the direct competition between the Machine Learning model and the standard statistical reconstruction. We aim to identify the limit where the asymptotic optimality of MLE might overtake the SVR.

Infidelity analysis

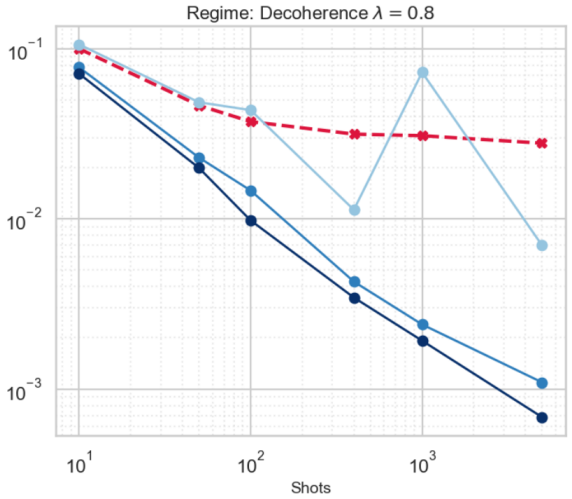
This analysis confronts the reconstruction error (Infidelity) of both methods as we can see in Figure 19:

In almost every configuration, the SVR trained on 5000 samples (Yellow curve) significantly outperforms the MLE.

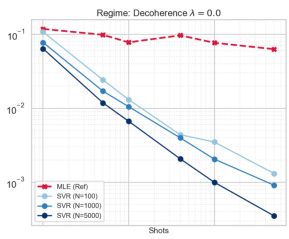
- **The difference:** The SVR maintains a consistent advantage (e.g., Infidelity gap ≈ 0.06 at $\lambda = 0$).
- **Physical Reason:** MLE performs a "blind" reconstruction, relying solely on current statistics. In contrast, the SVR utilizes a learned *prior* of the Hilbert space. It recognizes patterns in the measurement distributions rather than computing them from scratch, effectively beating the standard statistical limit ($1/\sqrt{N_{shots}}$) by leveraging its training memory.

Figure 19a ($\lambda = 0.8$) reveals the Achilles' heel of the machine learning approach.

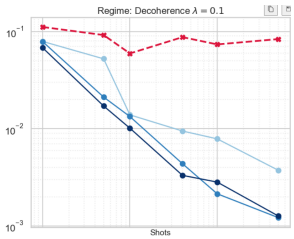
- **The anomaly:** The low-data SVR ($N = 100$) (Black curve) performs worse than the standard MLE.
- **Interpretation:** At high decoherence, the mapping between measurement statistics and density matrices becomes extremely complex (high entropy). A small dataset is insufficient to learn this non-linear



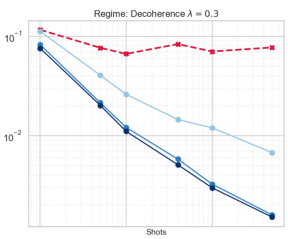
(a) **Critical Regime ($\lambda = 0.8$):** Comparison of SVR (Yellow/Blue) vs. MLE (Red Dashed). Note how the Low-Data SVR (Black/N=100) becomes erratic and loses to MLE.



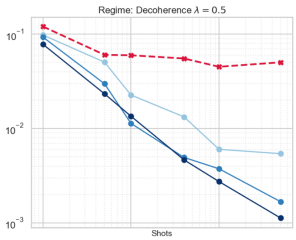
(b) $\lambda = 0.0$



(c) $\lambda = 0.2$



(d) $\lambda = 0.5$



(e) $\lambda = 0.7$

Figure 19: **SVR vs. MLE Benchmark.** Infidelity evolution across 5 decoherence regimes. The **Red Dashed Line** represents the standard MLE baseline.

map. The model starts "hallucinating" (overfitting), whereas MLE, being model-free, remains robust.

Conclusion: ML is superior to MLE only when the volume of training data matches the physical complexity of the noise. We are wondering wether ML will keep the advantage when we expend the project for tomography of several qubits.

Time analysis

Before analyzing the trade-off, we isolate the **Time factor**. We compare the reconstruction time per state for:

1. **MLE:** Which requires running a convex optimization solver for every new measurement.
2. **SVR:** Which only requires a matrix multiplication (inference) once trained.

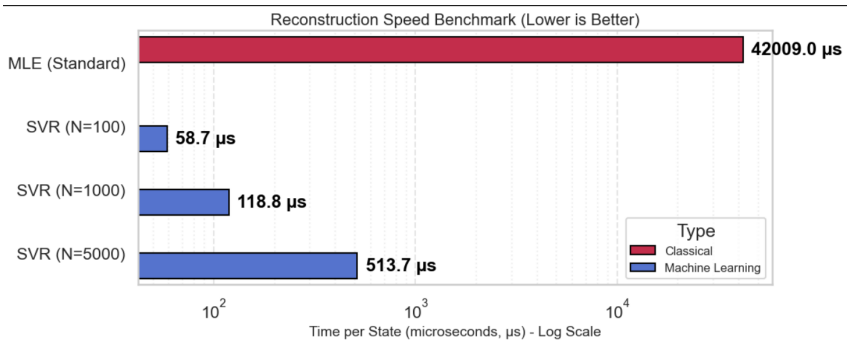


Figure 20: **Inference Speed Comparison (Log Scale).** Comparison of reconstruction time per state between standard MLE and SVR models trained on different dataset sizes.

The visualization highlights a drastic disparity in computational cost (note the logarithmic scale):

- **MLE (Classical):** Requires approximately **45 ms** $(44,648\mu s)$ per state. This latency is inherent to the method, which must solve a convex optimization problem from scratch for every single measurement.
- **SVR (Machine Learning):** Operates in the microsecond range, from **59** μ**s** $(N = 100)$ to **576** μ**s** $(N = 5000)$. Once trained, the reconstruction is reduced to a deterministic matrix operation.
- **Speed Factor:** The SVR approach is approximately **177x faster** than the standard MLE.
- **Throughput:** While MLE is limited to ~ 22 Hz (reconstructions/second), the SVR can achieve $\sim 3,966$ Hz.

Conclusion: The classical MLE method is effectively disqualified for high-frequency monitoring tasks. In contrast, the SVR offers real-time capability, enabling live feedback loops for quantum calibration or active error correction without creating a computational bottleneck.

6.4 General conclusion

Our comparison between the Support Vector Regressor (SVR) and the standard Maximum Likelihood Estimation (MLE) shows a clear better model on our key metrics:

- **The Speed of Light (or almost):** The most immediate industrial gain is the **177x speed-up**. By compressing the reconstruction into a simple matrix multiplication, SVR moves tomography from a post-processing bottleneck to a real-time capability (4000 Hz). This opens the door to active feedback loops and live calibration of Quantum Processing Units (QPUs).

- **Super-Resolution:** We observed that the SVR acts as an intelligent denoiser. By learning the "Prior" of the Hilbert space, it consistently beats the standard shot-noise limit ($1/\sqrt{N}$) that constrains the MLE.

When the physical complexity exceeds the information contained in the training set (e.g., Low Data regime), the SVR begins to "overfit" the noise, performing worse than the MLE. This highlights a fundamental trade-off:

- **SVR** is the ideal tool for **fast, repetitive calibration** in a known environment.
- **MLE** remains the essential "safety net" for **initial certification** or exploring unknown noise regimes, as it is an agnostic estimator that never hallucinates.

7 Using quantum kernel

7.1 Analytical feature map

7.1.1 Explanation of the algorithm

In our problem, each sample is already summarized by its Bloch vector derived from measurements:

$$r = (x, y, z) \in \mathbb{R}^3,$$

where typically

$$(x, y, z) = (\texttt{X\_mean}, \texttt{Y\_mean}, \texttt{Z\_mean}).$$

For a single qubit, any density matrix can be written as

$$\rho(r) \;=\; \frac{1}{2}\Big(I + r_x\sigma_x + r_y\sigma_y + r_z\sigma_z\Big),$$

with $\|r\| \le 1$ and the Pauli matrices $\sigma_x, \sigma_y, \sigma_z$.

The idea behind the first method is to explicitly construct a feature map

$$\phi : \mathbb{R}^3 \longrightarrow \mathbb{R}^5,$$

such that the Euclidean dot product in $\mathbb{R}^5$ reproduces the quantum fidelity (Uhlmann fidelity) between two qubit states.

We define the feature map as:

$$\phi(r) \;=\; \frac{1}{\sqrt{2}}\left(1,\; r_x,\; r_y,\; r_z,\; \sqrt{1 - \|r\|^2}\right) \;\in \mathbb{R}^5$$

$$\|r\|^2 = r_x^2 + r_y^2 + r_z^2.$$

In practice, we numerically clip $\|r\|^2$ within $[0, 1]$ to absorb small errors due to noise:

$$\tilde{r}^2 = \min\big(\max(\|r\|^2, 0), 1\big)$$

$$\sqrt{1 - \|r\|^2} \approx \sqrt{1 - \tilde{r}^2}.$$

For two Bloch vectors r and s, the kernel induced by this feature map

is simply

$$k_{\text{fmap}}(r, s) \;=\; \langle \phi(r), \phi(s) \rangle_{\mathbb{R}^5}.$$

It is easily verified that

$$\langle \phi(r), \phi(s) \rangle = \frac{1}{2}\Big(1 + r \cdot s + \sqrt{1 - \|r\|^2}\,\sqrt{1 - \|s\|^2}\Big),$$

which corresponds exactly to the Uhlmann fidelity $F(\rho(r), \rho(s))$ for one qubit states:

$$F\big(\rho(r), \rho(s)\big) = \frac{1}{2}\Big(1 + r \cdot s + \sqrt{(1 - \|r\|^2)\,(1 - \|s\|^2)}\Big).$$

7.1.2 Practical implementation

The practical consequence is very simple:

- We apply ϕ to each sample to obtain a feature matrix

$$\Phi = \begin{bmatrix} \phi(r_1)^\top \\ \vdots \\ \phi(r_N)^\top \end{bmatrix} \in \mathbb{R}^{N \times 5},$$

- Then, we train a classic linear SVM on Φ.

Mathematically, a linear SVM on Φ is exactly equivalent to an SVM with the kernel

$$k(r, s) = F(\rho(r), \rho(s)).$$

Since we have access to the explicit feature map, we completely avoid using quantum circuits and the direct calculation of a costly $N \times N$ kernel matrix. The time complexity is roughly linear in N (construction of Φ) plus the standard cost of a linear SVM.

We can see the algorithm used to apply this Quantum kernel in the algorithm 1:

7.1.3 Results for classification

We applied this method to a dataset of 5000 states, each state measured with 3000 shots (1000 per axis) and a mixed proportion of 0.5. We had the results as shown is Table 3

Algorithm 1 Analytical Feature Map & Linear SVM for Qubit States

Require: Dataset $\mathcal{D} = \{(x_i, y_i, z_i)\}_{i=1}^{N}$ of Bloch vector components.
Require: Label vector $\mathbf{Y} \in \{-1, 1\}^N$ for classification.
Ensure: Trained Linear SVM model.

Initialize feature matrix $\Phi \leftarrow \mathbf{0}_{N \times 5}$
for $i = 1$ to N **do**
    Let $r_i \leftarrow (x_i, y_i, z_i)$ as estimation
    Compute squared norm: $\|r_i\|^2 \leftarrow x_i^2 + y_i^2 + z_i^2$
    Clip to handle noise: $\tilde{r}^2 \leftarrow \min(\max(\|r_i\|^2, 0), 1)$
    Calculate feature vector $\phi(r_i)$:

$$\phi(r_i) \leftarrow \frac{1}{\sqrt{2}} \left(1,\, x_i,\, y_i,\, z_i,\, \sqrt{1 - \tilde{r}^2}\right)$$

    Store $\phi(r_i)$ as the i-th row of Φ:

$$\Phi[i, :] \leftarrow \phi(r_i)$$

end for
Train Linear SVM using Φ and labels $\mathbf{Y}$:

$$\text{model} \leftarrow \text{LinearSVC}(\text{input} = \Phi, \text{labels} = \mathbf{Y})$$

return model

Table 3: Classification Metrics

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.99	0.92	0.96	502
1	0.93	0.99	0.96	498
accuracy		0.96		1000
macro avg	0.96	0.96	0.96	1000
weighted avg	0.96	0.96	0.96	1000

We can see that we have overall very good results, an accuracy of around 0.96.

When splitting the dataset between train and test, we kept 1000 values for the test, so the data has been trained with 4000 different states. We obtained the confusion matrix as shown in Table 4:

Table 4: Confusion Matrix Results

	Predicted Class		
Actual Class	**0** (Pure States)	**1** (Mixed States)	**Total**
0 (Pure States)	464 (True Negatives)	38 (False Positives)	502
1 (Mixed States)	3 (False Negatives)	495 (True Positives)	498
Total	467	533	1000

Analysis of the confusion matrix reveals a slight bias: the model is more conservative, tending to misclassify Mixed States (38 instances) as Pure

States more often than the reverse (3 instances). But overall the kernel works great as we have few numbers of false negatives and false positives.

To have a better comparison about how each parameter of the dataset (number of shots, mixed proportion and number of states) affects the performance of the kernel, we applied it to several datasets:

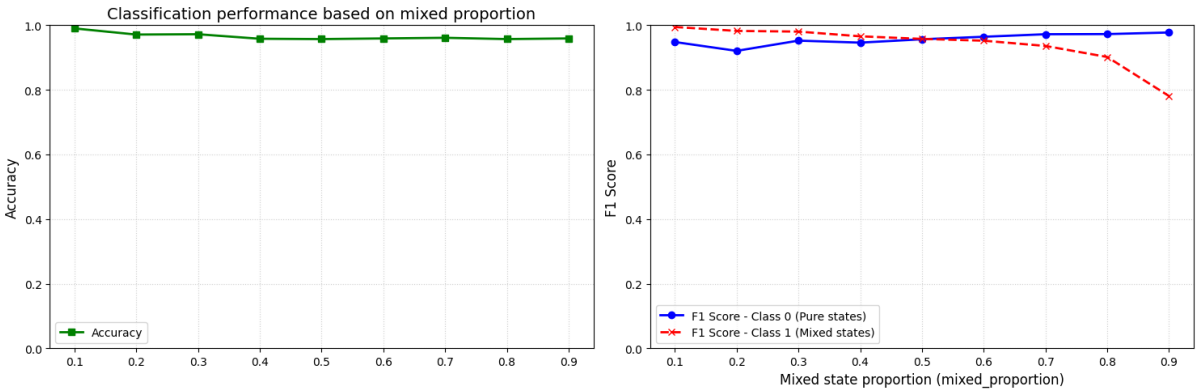


Figure 21: Classification performance based on mixed proportion

The figure shows the model's robustness against variations in the data balance.

- **Accuracy:** The overall accuracy remains exceptionally stable and high (near 0.98) across all tested mixed state proportions (0.1 to 0.9).
- **F1-Score:** The performance for **Pure States (Class 0)** is highly consistent (> 0.96). However, the F1-Score for **Mixed States (Class 1)** shows a noticeable degradation, dropping sharply to approximately 0.78 when the mixed state proportion reaches 0.9.

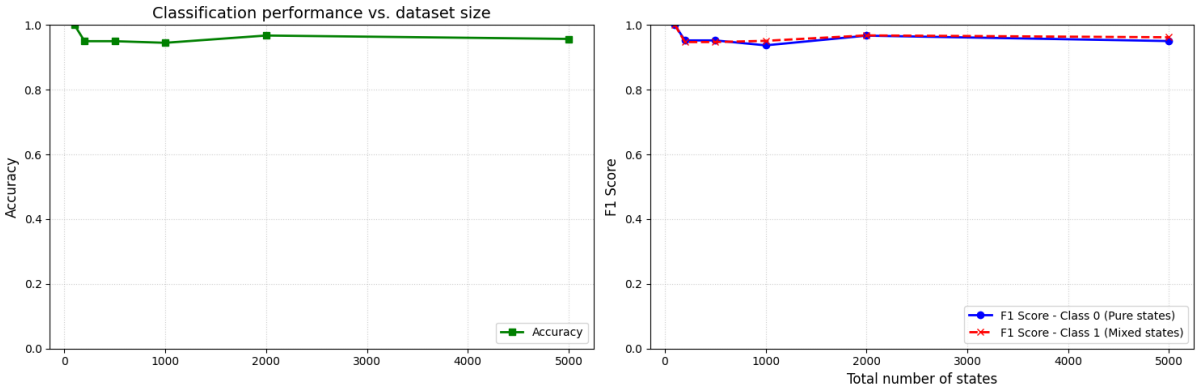


Figure 22: Classification performance vs. dataset size

The model achieved an exceptional overall accuracy of 0.96 (96%). F1-scores for both classes (Pure/Mixed) are highly balanced at 0.96.

We can conclude that the classifier is highly data-efficient. The overall Accuracy and F1-Scores reach near-maximum performance with a minimal dataset size (approximately 100 -200 states). Increasing the sample size beyond this point has no significant gain in classification performance.

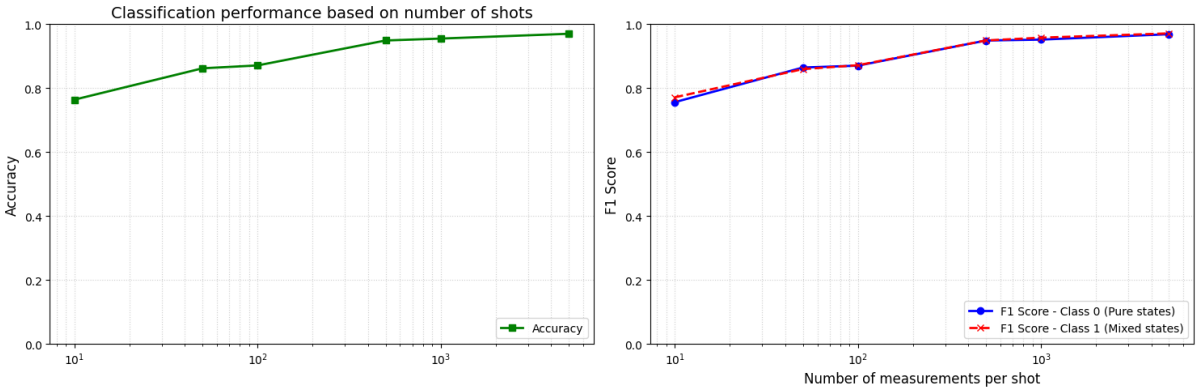


Figure 23: Classification performance based on shots

Here, we can see that the more the number of shots increase, the better is the classification performance. This is intuitive as the more measurement shots we have on a state, the more exact we are from its exact value.

Conclusion: The model is robust against increasing dataset size but its performance is dictated primarily by the **measurement quality** (N_{shots}) and secondarily by **extreme class imbalance**. The sharp drop in Mixed State F1-score at 90% imbalance tells us that the model struggles to differentiate the high complexity of the majority class under extreme conditions.

7.1.4 Limitations

The analytical feature map provides an very efficient solution for single-qubit systems, but it cannot be easily generalized to multi-qubit systems ($n > 1$). This limitation comes from the unique geometry of the qubit state space compared to higher dimensions.

The mathematical "coincidence" that allows us to write the fidelity $F(\rho, \sigma)$ as a simple Euclidean dot product $\phi(r) \cdot \phi(s)$ relies on the specific algebraic properties of 2×2 matrices. For larger systems, the general fidelity formula involves complex matrix square roots that do not factorize into a simple, finite-dimensional vector product. Therefore, finding an explicit ϕ for n qubits would be very inefficient and time-consuming.

7.2 Fidelity-Based Kernel

7.2.1 Explanation of the algorithm

In the second method, we do not construct an explicit feature map. We directly use an analytical formula for the fidelity between two qubit states, written as a function of their Bloch vectors r and s.

For a qubit, the Uhlmann fidelity between two states $\rho(r)$ and $\rho(s)$ (pure or mixed) is written as

$$F\big(\rho(r),\rho(s)\big) = \frac{1}{2}\Big(1 + r\cdot s + \sqrt{(1-\|r\|^2)\,(1-\|s\|^2)}\Big).$$

We therefore define the kernel

$$k_{\text{fid}}(r,s) = F\big(\rho(r),\rho(s)\big) = \frac{1}{2}\Big(1 + r\cdot s + \sqrt{(1-\|r\|^2)\,(1-\|s\|^2)}\Big).$$

7.2.2 Kernel matrix conditions

We also mustn't forget the conditions for the kernel matrix: To be able to use this kernel in an SVM, the kernel matrix

$$K_{ij} = k_{\text{fid}}(r_i, r_j)$$

must satisfy the classical conditions:

- **Symmetry**: $K_{ij} = K_{ji}$, which is immediate because $r\cdot s = s\cdot r$ and the formula is symmetric in (r,s);
- **Positive semi-definite**: for any vector $c\in\mathbb{R}^N$,

$$\sum_{i,j} c_i c_j K_{ij} \;\geq 0.$$

Here, we know that k_{fid} is a valid kernel because it originates from a scalar product in a finite-dimensional space. Indeed, we showed that there exists a feature map $\phi:\mathbb{R}^3\to\mathbb{R}^5$ such that

$$k_{\text{fid}}(r,s) = \langle\phi(r),\phi(s)\rangle.$$

In other words, the matrix K is a *Gram matrix*

$$K = \Phi\Phi^\top,$$

with $\Phi_{i\cdot} = \phi(r_i)^\top$, which automatically guarantees that K is symmetric and positive semi-definite.

7.2.3 Practical implementation

Concretely, we proceed as follows:

1. We extract naive estimation of the Bloch vectors

$$r_i = (x_i, y_i, z_i) \quad \text{from} \quad (\texttt{X\_mean}, \texttt{Y\_mean}, \texttt{Z\_mean});$$

2. We calculate the training kernel matrix

$$K_{ij}^{\text{train}} = k_{\text{fid}}(r_i^{\text{train}}, r_j^{\text{train}}),$$

   and the test–train kernel matrix

$$K_{ij}^{\text{test}} = k_{\text{fid}}(r_i^{\text{test}}, r_j^{\text{train}});$$

3. We train an SVM with `kernel="precomputed"` on K^{train}, then predict on K^{test}.

This method exploits the quantum structure (fidelity between states) but remains entirely classical: everything is done in NumPy, without quantum circuits, and the calculation is very fast thanks to vectorization (matrix products instead of loops).

We can see the algorithm of the method implementation in the algorithm 2:

Mathematically, this method uses the same idea of the previous one (using the fidelity formula with r and s). But this method is actually more general as we can use it for more qubits using the general fidelity formula as the kernel:

$$F(\rho, \sigma) = \left(\text{Tr} \sqrt{\sqrt{\rho}\sigma\sqrt{\rho}} \right)^2$$

Using the **Kernel trick**, we don't have to calculate the feature map as we can straight use the matrices to get the fidelity and compute the Kernel matrix.

The matrix given by the fidelity also satisfies the kernel matrix conditions:

Algorithm 2 Fidelity-Based Kernel (Precomputed Matrix) SVM

Require: Training Set $\mathcal{D}_{\text{train}} = \{r_i^{\text{train}}\}_{i=1}^{N_{\text{train}}}$ (Bloch vectors).
Require: Test Set $\mathcal{D}_{\text{test}} = \{r_i^{\text{test}}\}_{i=1}^{N_{\text{test}}}$ (Bloch vectors).
Require: Training Labels $\mathbf{Y}_{\text{train}}$.
Ensure: Predictions on Test Set.

function FidelityKernel(r, s)
    Compute dot product: $d \leftarrow r \cdot s$
    Compute norms (clipped to $[0, 1]$ for stability):
        $\rho_r \leftarrow \min(\max(\|r\|^2, 0), 1)$
        $\rho_s \leftarrow \min(\max(\|s\|^2, 0), 1)$
    Calculate fidelity:

$$k \leftarrow \frac{1}{2}\left(1 + d + \sqrt{(1-\rho_r)(1-\rho_s)}\right)$$

    return k
end function

1. Compute Training Kernel Matrix $K^{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times N_{\text{train}}}$
for $i = 1$ to N_{train} **do**
    for $j = i$ to N_{train} **do** ▷ Exploit symmetry
        $val \leftarrow$ FidelityKernel$(r_i^{\text{train}}, r_j^{\text{train}})$
        $K_{ij}^{\text{train}} \leftarrow val$
        $K_{ji}^{\text{train}} \leftarrow val$
    end for
end for

2. Compute Test-Train Kernel Matrix $K^{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times N_{\text{train}}}$
for $i = 1$ to N_{test} **do**
    for $j = 1$ to N_{train} **do**
        $K_{ij}^{\text{test}} \leftarrow$ FidelityKernel$(r_i^{\text{test}}, r_j^{\text{train}})$
    end for
end for

3. Train and Predict (Precomputed SVM)
Initialize SVM with `kernel="precomputed"`
model $\leftarrow$ SVM.fit$(K^{\text{train}}, \mathbf{Y}_{\text{train}})$
Predictions $\leftarrow$ model.predict(K^{test})
return Predictions

1. **Symmetry:** $K_{ij} = K_{ji}$. The general definition of Uhlmann fidelity is:

$$F(\rho, \sigma) = \left(\text{Tr}\sqrt{\sqrt{\rho}\,\sigma\,\sqrt{\rho}}\right)^2.$$

Although the matrix product $\sqrt{\rho}\,\sigma\,\sqrt{\rho}$ appears asymmetric, the trace property $\text{Tr}(AB) = \text{Tr}(BA)$ and the inherent properties of fidelity ensure that the measure is symmetric:

$$F(\rho, \sigma) = F(\sigma, \rho).$$

Thus, the resulting kernel matrix is symmetric ($K_{ij} = K_{ji}$).

2. **Positive Semi-Definiteness (PSD):** The matrix must not have negative eigenvalues. To prove this property, we use *Uhlmann's Theorem*, which relates fidelity to the overlap of pure states in a larger system (purification). The theorem states:

$$F(\rho, \sigma) = \max_{|\psi_\rho\rangle, |\psi_\sigma\rangle} \left|\langle \psi_\rho \mid \psi_\sigma \rangle\right|^2,$$

where $|\psi_\rho\rangle$ and $|\psi_\sigma\rangle$ are purifications of ρ and σ in an extended Hilbert space $\mathcal{H}$.

In the end, we have:

- The term $\langle \psi_\rho \mid \psi_\sigma \rangle$ is a standard inner product in the Hilbert space $\mathcal{H}$. A standard inner product is a valid linear kernel.
- The absolute square $|\cdot|^2$ corresponds to the product of the kernel with its complex conjugate. Since the class of valid kernels is closed under point-wise multiplication, the squared magnitude of a valid kernel is also a valid kernel.

Formally, if we fix the optimal purification map $\Phi : \rho \mapsto |\psi_\rho\rangle$, we can write the kernel as a dot product in the feature space defined by the tensor product of the Hilbert space with itself:

$$k(\rho, \sigma) = \langle \psi_\rho \mid \psi_\sigma \rangle\langle \psi_\sigma \mid \psi_\rho \rangle = \langle \psi_\rho \otimes \psi_\rho^* \mid \psi_\sigma \otimes \psi_\sigma^* \rangle.$$

Since the kernel can be expressed as an inner product of vectors $\Psi_\rho = |\psi_\rho\rangle \otimes |\psi_\rho^*\rangle$, the corresponding matrix is automatically positive semi-definite.

7.2.4 Results for classification

After applying this second method to the same dataset of 5000 states, each state measured with 3000 shots (1000 per axis) and a mixed proportion of 0.5, we got the results we can see in Table 5 and Table 6:

As we can see once again, we have very good results with again around 0.96 accuracy. There also seems to be a slight bias as the model tends once again to miclassify mixed states as pure states more often than the opposite.

We computed the same graphs as for the previous kernel as we can see in Figures 24, 25 and 26:

Table 5: Results of classification report

Class	Precision	Rappel	F1-Score	Support
0	0.99	0.92	0.96	502
1	0.93	0.99	0.96	498
accuracy		0.96		1000
macro avg	0.96	0.96	0.96	1000
weighted avg	0.96	0.96	0.96	1000

Table 6: Confusion matrix

	Predicted Class		
Actual Class	**0** (Pure States)	**1** (Mixed States)	**Total**
0 (Pure States)	464	36	500
1 (Mixed States)	10	490	500
Total	474	526	1000

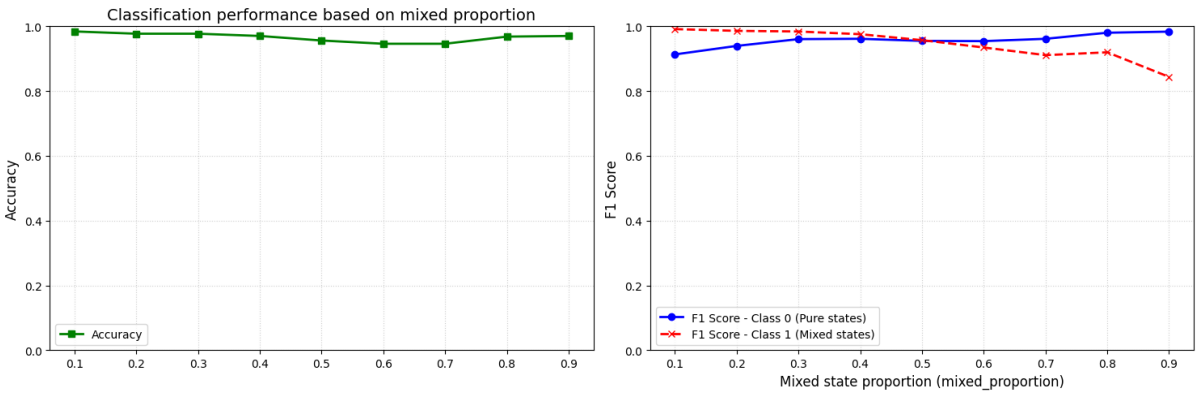


Figure 24: Classification performance based on mixed proportion

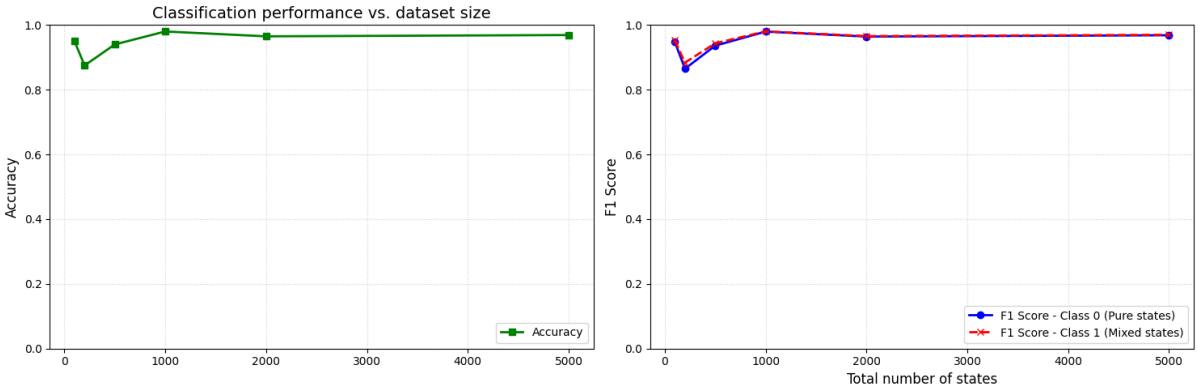


Figure 25: Classification performance vs. dataset size

The results are very similar to those with the previous kernel: the model is overall very good no matter the proportion of mixed and pure states, except when the data is very much unbalanced with more than 80-90%

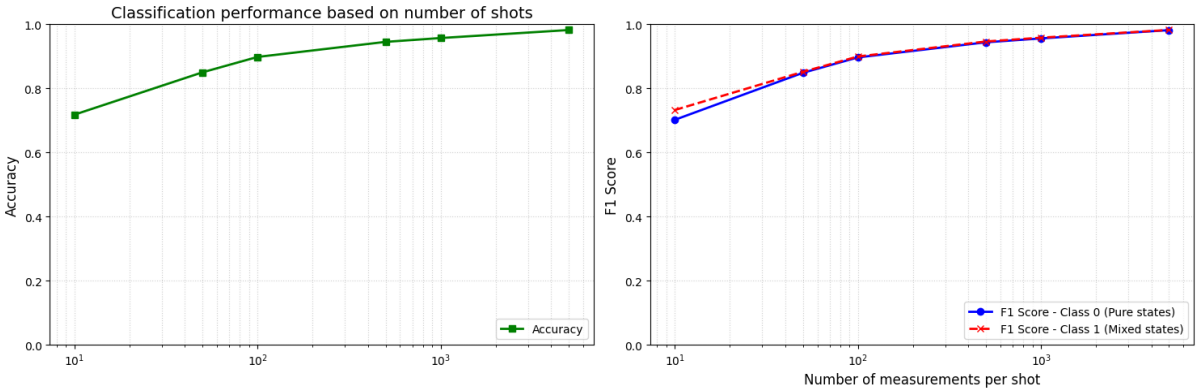


Figure 26: Classification performance based on number of shots

of pure states or the opposite. The performance is also very stable above 1000 states in the dataset and increases as the number of measurements per state increases.

7.2.5 Limitations

A fundamental limitation arises when attempting to scale this analytical kernel method to multi-qubit systems ($n > 1$) for the purpose of Quantum State Tomography.

The fidelity formula, defined as $F(\rho, \sigma) = (\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}})^2$, is only well-defined when ρ and σ are valid density matrices, specifically, they must be positive semi-definite with unit trace. In our single-qubit implementation, we satisfied this condition easily: if the noisy measurement vector had a norm $\|r\| > 1$, we simply clipped it to 1. This "clipping" is a trivial projection that guarantees a valid physical state.

However, for higher dimensions, this simple geometric fix is impossible. When we reconstruct a density matrix estimate from raw, noisy measurements on n qubits, the resulting matrix ρ_{raw} almost always violates quantum constraints (it will contain negative eigenvalues) due to finite sampling noise (shot noise).

This creates a critical blockage for the kernel:

- **Mathematical Breakdown:** If we feed this noisy ρ_{raw} directly into the analytical kernel, the term $\sqrt{\rho_{\mathrm{raw}}}$ requires taking the square root of a matrix with negative eigenvalues. This results in complex numbers or undefined operations, rendering the kernel useless. We might still find initial estimations of our state, but it would still be noisy due to measurement noise.

- **The Projection Cost:** To use the kernel, we would first need to "fix" ρ_{raw} by projecting it onto the set of valid density matrices (e.g., via Maximum Likelihood Estimation). However, finding the valid density matrix is the exact goal of the Tomography task itself. If we must solve the full tomography problem just to format the input for our kernel, the machine learning approach becomes redundant.

Therefore, this analytical kernel is very efficient for single-qubit models where validity can be enforced by simple clipping, but it isn't suited for high-dimensional tomography where the input data does not naturally correspond to a valid quantum state.

7.3 Standard quantum kernel based on circuits (Qiskit)

Finally, the third approach is the "standard" quantum kernel scheme used in the `Qiskit` [3] library, via parameterized circuits. The idea is as follows:

- For each data vector $x \in \mathbb{R}^d$, we construct a quantum circuit $U_\phi(x)$ (called a quantum *feature map*);
- We prepare the state

$$|\psi(x)\rangle = U_\phi(x)\,|0\rangle^{\otimes n},$$

  where n is the number of qubits used (in our case, we remain on a single qubit to reflect the state structure);
- The kernel between two vectors x and x' is defined as the fidelity between the output states:

$$k_{\text{QC}}(x, x') = \left|\langle\psi(x) \mid \psi(x')\rangle\right|^2 = \left|\langle 0|U_\phi(x)^\dagger U_\phi(x')|0\rangle\right|^2.$$

In Qiskit, we can choose, for example, a `ZZFeatureMap` with a certain depth (`reps`) and an entanglement scheme. The kernel is then implemented by the `FidelityQuantumKernel` class, and a quantum SVM by `QSVC`:

- Encodes the classical data x into the rotation angles of the circuit;
- For each pair (x_i, x_j), it must execute a circuit that implements $U_\phi(x_i)^\dagger U_\phi(x_j)$ and estimate the fidelity via measurements;
- This fills the kernel matrix $K_{ij} = k_{\text{QC}}(x_i, x_j)$.

The problem arises from time complexity:

- For a training set of size N, the kernel matrix is of size $N \times N$. We therefore need to evaluate $O(N^2)$ pairs (x_i, x_j).

- Each evaluation $k_{\mathrm{QC}}(x_i, x_j)$ requires a quantum circuit execution (or simulation) with a certain number of *shots* to estimate the fidelity with reasonable variance.
- Even on a single qubit, the cumulative cost of $O(N^2)$ simulations becomes significant as soon as N exceeds a few hundred, especially if optimization (hyperparameter search, cross-validation, etc.) recalculates the kernel matrix multiple times.

This is because we are not using the Kernel trick here: we apply the equivalent of feature map to every state, every entry of our dataset and the computation becomes extremely high as the dataset is bigger.

This method has not yet been tested due to time constraint as it usually takes a long time to simulate on a computer.

8 Exploration and curiosity: DNN & VQC

8.1 Scientific and pedagogical motivation

Beyond classical regression (SVR), we want to explore whether architectures with higher representational capacity could better capture the complex geometry of noisy quantum states. This initiative is motivated by two axes:

1. **The scientific axis:** Comparing the force of a universal Deep Neural Network (DNN) against the "native" approach of the Variational Quantum Circuit (VQC).
2. **The pedagogical axis:** This exploration allowed us to learn more on modern Machine Learning pipelines. As one group member is focusing their research project on neural networks, we implemented these models using GPU acceleration via **CUDA** (for PyTorch) and high-performance C++ simulators.

The major interest of the VQC lies in its inductive bias. Unlike a classical neural network that works in a real vector space ($\mathbb{R}^N$) and must "learn" physical constraints (such as the fact that a density matrix must have a trace of 1), the VQC naturally operates within Hilbert space.

Our hypothesis is that a quantum circuit, being governed by the same mathematical laws as the state we seek to reconstruct (unitary operations, Bloch sphere), should be capable of generalizing more efficiently, that is, with significantly fewer parameters than a classical DNN. It is a "Quantum for Quantum" approach.

8.2 Simplified operation of the VQC

We used the PennyLane [2] library to build a hybrid variational circuit. The principle is as follows:

1. **Encoding:** Noisy measurements (classical data) are converted into qubit rotations. We thus place our data directly onto the Bloch sphere.
2. **Ansatz:** A series of quantum gates parameterized by angles θ manipulates the state. These are the angles we train.
3. **Measurement:** We project the final state to obtain the coordinates of the predicted Bloch vector.

The learning process is illustrated in Figure 27. The classical optimizer modifies the circuit parameters θ via a cost function $f(\theta)$ (here based on Fidelity) to minimize the error.

Figure 27: **Schematic diagram of variational optimization.** The circuit (gate) parameters θ are iteratively updated by a classical function $f(\theta)$ derived from backpropagation, in order to converge towards the target state.

8.3 Gradient descent on fidelity

The central innovation of our approach lies in the Loss function of the VQC. In classical regression problems, training generally minimizes the Mean Squared Error (MSE), which represents a geometric Euclidean distance.

However, the space of quantum states (the set of density matrices) has a complex geometry where Euclidean distance is not the most physically relevant metric. We therefore replaced the MSE with a direct maximization of Quantum Fidelity.

8.3.1 Mathematical Definition of the Loss

Since gradient descent optimizers often minimize a function, we define our loss function $\mathcal{L}$ as the complement of the fidelity F, or the infidelity:

$$\mathcal{L}(\theta) = 1 - F(\rho_{\text{pred}}(\theta), \rho_{\text{target}}) \tag{11}$$

Thus, when the loss tends towards 0, the fidelity tends towards 1 (perfect state overlap).

You can find a more detailed mathematical analysis of the VQC in Appendix 9.5.

8.3.2 Analytical formulation for the gradient

To enable backpropagation, fidelity must be expressed in terms of the components of the predicted Bloch vector $\vec{r}_p$ (network output) and the target $\vec{r}_t$. As shown in the previous sections, the fidelity formula for 1-qubit states is:

$$F(\vec{r}_p, \vec{r}_t) = \frac{1}{2}\left(\underbrace{1 + \vec{r}_p \cdot \vec{r}_t}_{\text{Alignment}} + \underbrace{\sqrt{(1 - ||\vec{r}_p||^2)(1 - ||\vec{r}_t||^2)}}_{\text{Purity Correction}}\right) \tag{12}$$

This equation guides the gradient in two distinct ways during training:

- **The alignment term ($\vec{r}_p \cdot \vec{r}_t$):** Forces the predicted vector to point in the same direction as the target on the Bloch sphere.
- **The purity term (square root):** It forces the network to adjust the norm of the vector $||\vec{r}_p||$ to match the decoherence level (purity) of the target. It is this term that allows the DNN and VQC to specifically learn the noise.

Unlike a standard MSE, which would treat Bloch vectors as simple points in $\mathbb{R}^3$, this loss function tries to respect the constraints of the probabilistic structure of quantum physics.

8.4 Performance and results

Training the VQC proved extremely slow compared to the DNN (several minutes versus a few seconds). This is explained by the nature of the simulation: simulating a quantum system on a classical computer (even with GPU) requires complex linear algebra calculations that grow exponentially with the number of qubits, whereas the DNN only performs simple matrix multiplications.

However, in terms of pure performance, the results are excellent for both approaches. As shown in Figure 28, we obtain near-perfect fidelities:

- For pure states, fidelity reaches **1.0**.
- Even with strong decoherence (0.8 noise), fidelity remains above **0.995**.

Both models successfully "denoise" the measurement, with the VQC achieving this with significantly fewer learnable parameters, validating our hypothesis regarding parametric efficiency despite the computational cost of the simulation. We believe that DNN and VQC could be helpfulm for us as we will continue this research project during the second semester.

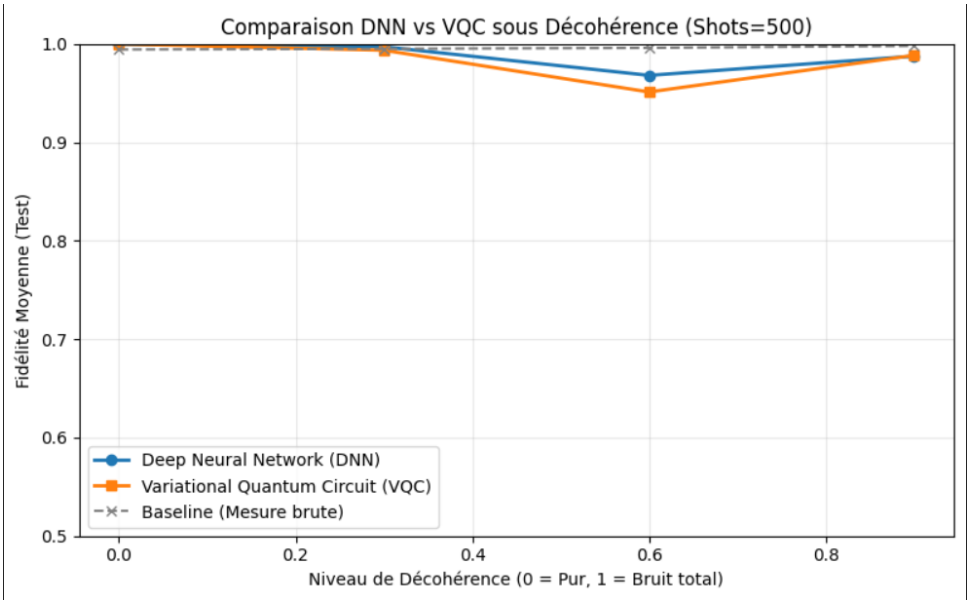


Figure 28: **Comparison of mean Fidelities on the test set as a function of decoherence level.** The DNN and VQC outperform the baseline and maintain a fidelity close to 1.

9 Problems encountered

9.1 Picking a method

At the beginning we lost time because we kept hesitating between several valid approaches (simple inversion, MLE, circuit-based methods, VQC, ...) to start our project. Each option looked attractive depending on what we were interested about at that moment (speed, physical correctness, accuracy, ease of implementation). We also saw big with straight up using several qubit and very complex implementation of our model.

What fixed it was forcing us to align a feasible method with the deliverable: generate 1-qubit data, reconstruct valid states, and run a classification (pure vs mixed) with a fidelity-based kernel. Once we could start with this scope in mind, progress became much more linear. The side benefit is that the report became clearer, because the pipeline had a clean logic.

9.2 Understanding the concepts used

Two of the members of this project have started the Quantum track this year, so they have basic knowledge about Quantum Mechanics. But as this is a very complex field and we only started learning about it a few months ago, we had superficial knowledge that wasn't enough to really handle the project requirements. Moreover the last member is not part of the quantum track, making the task even harder.

To solve this issue, we had to consume a lot of knowledge to understand as much as we can the tools we are working with and be sure we have a good comprehension of the working environment.

9.3 Understanding how a Kernel works

It was also difficult to really understand how a Kernel works, the math behind and what exactly is the Kernel trick. While trying to build a new Quantum kernel, we tried to understand what kind of object it is and how to implement it in python. We thought that with a custom Kernel using the Kernel trick we wouldn't have to compute the Kernel matrix, but SVC still needs pairwise kernel evaluation.

Moreover, it was also difficult to understand how a Quantum kernel really works. We first thought that taking a Kernel that uses quantum data would be a Quantum kernel, but in reality this is just a normal Kernel adapted for Quantum computing as it can quantify how different two

states are. The real quantum kernel, using quantum circuits to encore information, is very different as we must always calculate the quantum feature map for each state.

In the end we managed to understand all the subtle differences between that and try to apply some concepts to regression.

9.4 Applying MLE

We also had during a long time a wrong idea with MLE. We thought we had to give to the model a first estimation of the vector with r, θ_0, ϕ_0 as simply issued from the measurement means as a naive estimation (we just clip $r = 1$ in case the vector is outside of the Bloch sphere) and the algorithm finds the best estimators, but the real method is to give the parameters of the density matrix and after making sure the hermitian conditions are respected thanks to Cholesky decomposition, run the algorithm.

9.5 Using VQC

It was also very challenging to use VQC and DNN as we have not studied those concepts in class. We had to learn about the working concept of DNN and how it works, what are its parameters, etc., then learn what are VQC, how they work and what and why they can bring something to QST. We discovered that VQC was easier to visualise thanks to our knowledge in deep learning, since it is basically a physical neural network.

Future Perspectives

Where do we go from here? The natural next step is to scale this approach to **Multi-Qubit Tomography**.

However, as we hinted throughout this work, the curse of dimensionality is real. Generating a representative dataset for two or three qubits is exponentially harder than for one. Validating such a model would require computing resources (and a report length) far exceeding the scope of this semester.

Appendix: Maximum Likelihood Estimation

To reconstruct the true quantum state from our noisy measurement data, we used a method called Maximum Likelihood Estimation (MLE). The linear approach is fast, but it has a flaw: it can sometimes produce results that are physically impossible, such as a state vector with a length greater than 1. MLE solves this by searching for the valid quantum state that is most likely to have produced the measurement counts we observed (N_x, N_y, N_z).

Measurement Probabilities

For a single-qubit system, we perform measurements in the Pauli bases $\{X, Y, Z\}$. The probability of observing the "up" $(+1)$ or "down" (-1) eigenstate for a given Pauli operator σ_K (where $K \in \{x, y, z\}$) is governed by Born's rule:

$$P(K_+|\rho) = \frac{1 + \mathrm{Tr}(\rho\sigma_K)}{2}, \quad P(K_-|\rho) = 1 - P(K_+|\rho) \tag{13}$$

Here, $\mathrm{Tr}(\rho\sigma_K)$ corresponds to the expectation value (or the Bloch vector component) along axis K.

Likelihood Function

Suppose we perform N measurement shots for each basis K. The experiment yields a set of counts n_{K+} (outcomes aligned with the axis) and n_{K-} (outcomes anti-aligned). Assuming independent measurements, the probability of observing this specific dataset $\mathcal{D}$ given a state ρ is the product of binomial probabilities:

$$\mathcal{L}(\rho) = P(\mathcal{D}|\rho) = \prod_{K\in\{x,y,z\}} (P(K_+|\rho)^{n_{K+}} \cdot P(K_-|\rho)^{n_{K-}}) \tag{14}$$

To simplify the optimization, we work with the **log-likelihood** function. Maximizing the likelihood is equivalent to minimizing the Negative Log-Likelihood (NLL):

$$\mathrm{NLL}(\rho) = -\sum_{K\in\{x,y,z\}} \left[n_{K+} \ln\left(\frac{1 + \mathrm{Tr}(\rho\sigma_K)}{2}\right) + n_{K-} \ln\left(\frac{1 - \mathrm{Tr}(\rho\sigma_K)}{2}\right)\right] \tag{15}$$

Cholesky Parameterization

A valid density matrix ρ must satisfy three conditions:

1. $\rho = \rho^\dagger$ (Hermitian)
2. $\mathrm{Tr}(\rho) = 1$ (Normalized)
3. $\rho \succeq 0$ (Positive Semi-Definite)

Standard unconstrained optimization of the NLL often leads to unphysical states (e.g., negative eigenvalues). To enforce these constraints automatically, we parameterize ρ using the Cholesky decomposition. We define a lower triangular matrix T:

$$T(\vec{\theta}) = \begin{pmatrix} t_0 & 0 \\ t_1 + it_2 & t_3 \end{pmatrix} \tag{16}$$

where $\vec{\theta} = [t_0, t_1, t_2, t_3]$ are real parameters. The density matrix is then constructed as:

$$\rho(\vec{\theta}) = \frac{TT^\dagger}{\mathrm{Tr}(TT^\dagger)} \tag{17}$$

By definition, $TT^\dagger$ is Hermitian and positive semi-definite. Dividing by the trace ensures normalization. The optimization problem thus becomes an unconstrained search over the real parameters $\vec{\theta}$ to minimize $\mathrm{NLL}(\rho(\vec{\theta}))$.

The Algorithm 3 shows the steps that were performed.

Algorithm 3 Maximum Likelihood Estimation for Qubit State Tomography

Require: Empirical means $\bar{X}, \bar{Y}, \bar{Z}$, Number of shots N
Ensure: Estimated density matrix $\hat{\rho}$

Input: Measurement data row from dataset
Initialize parameters $\boldsymbol{\theta}_0 = [1, 0, 0, 1]$ ▷ Start with maximally mixed state or valid guess
function NLL($\boldsymbol{\theta}$) ▷ Negative Log-Likelihood Objective
    Construct Cholesky matrix $T(\boldsymbol{\theta}) = \begin{pmatrix} a & 0 \\ b + ic & d \end{pmatrix}$
    $\rho \leftarrow \frac{TT^\dagger}{\mathrm{Tr}(TT^\dagger)}$ ▷ Ensure ρ is physical ($\rho \geq 0$, $\mathrm{Tr}(\rho) = 1$)
    $nll \leftarrow 0$
    for $K \in \{X, Y, Z\}$ **do**
        $e_K \leftarrow \mathrm{Tr}(\rho\sigma_K)$ ▷ Calculate expected Bloch vector component
        $p_K \leftarrow \frac{1+e_K}{2}$ ▷ Theoretical probability of outcome $+1$
        $p_K \leftarrow \mathrm{clip}(p_K, \epsilon, 1-\epsilon)$ ▷ Avoid log(0)
        $n_{K+} \leftarrow \mathrm{round}\left(N \cdot \frac{1+\bar{K}}{2}\right)$ ▷ Convert mean to counts
        $n_{K-} \leftarrow N - n_{K+}$
        $nll \leftarrow nll - (n_{K+} \ln(p_K) + n_{K-} \ln(1-p_K))$
    end for
    return nll
end function
$\hat{\boldsymbol{\theta}} \leftarrow \mathrm{minimize}(\mathrm{NLL}, \boldsymbol{\theta}_0)$ ▷ Using L-BFGS-B or similar optimizer
$\hat{T} \leftarrow T(\hat{\boldsymbol{\theta}})$
$\hat{\rho} \leftarrow \frac{\hat{T}\hat{T}^\dagger}{\mathrm{Tr}(\hat{T}\hat{T}^\dagger)}$
return $\hat{\rho}$

Appendix : In-Depth Mathematical Analysis of the VQC

This appendix details the mathematical formalism governing the Variational Quantum Circuit (VQC) and demonstrates how the physical device naturally encodes the numerical algorithm. Here, the VQC is modeled as a parameterized wave function $|\psi(\vec{x}, \vec{\theta})\rangle$ evolving in a Hilbert space.

1. The Framework: Hilbert Space

Unlike classical neural networks that operate via affine transformations in $\mathbb{R}^N$, our algorithm unfolds in the Hilbert space $\mathcal{H} \cong \mathbb{C}^{2^n}$. For $n = 3$ qubits, this is a complex vector space of dimension $d = 2^3 = 8$.

The "memory" of the calculation is not stored in floating-point variables, but in the probability amplitudes α_i of the state vector:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \quad \text{with } \sum |\alpha_i|^2 = 1 \tag{18}$$

It is this normalization constraint (Law of Conservation of Probability) that acts as a natural regularizer, preventing the model from making physically aberrant predictions.

2. Data Encoding (Data Embedding)

The crucial step is to map the input data $\vec{x}$ to a quantum state $|\psi_{in}(\vec{x})\rangle$. This is where we transition from the classical world to the quantum world.

In our study, the inputs $\vec{x} = [x, y, z]$ represent physical properties (rotations). We use *Angle Embedding*. Each classical feature x_i becomes the angle of an R_Y rotation gate on qubit i:

$$|\psi_{in}\rangle = \bigotimes_{i=1}^{3} R_Y(x_i)|0\rangle \tag{19}$$

Physically, this amounts to preparing the qubit in a specific superposition. If $x_i = 0$, the qubit remains at $|0\rangle$. If $x_i = \pi$, it flips to $|1\rangle$.

To fully grasp the power of encoding, imagine using this VQC to classify a classical image (e.g., 4 grayscale pixels, normalized vector $\vec{v} =$

$[0.5, 0.5, 0.5, 0.5]$). We would then use **Amplitude Embedding**. Instead of rotating the qubits, we would encode the data *into the amplitudes* of the state itself:

$$|\psi_{\text{img}}\rangle = 0.5|00\rangle + 0.5|01\rangle + 0.5|10\rangle + 0.5|11\rangle \tag{20}$$

This technique allows encoding a vector of size N into only $\log_2(N)$ qubits. This is an exponential memory space advantage unique to quantum computing, which Angle Embedding (used in our project) does not possess, though it is unnecessary here given the low dimension of our inputs.

3. The Ansatz: parameterized unitary processing

Once the data is loaded, the "calculation" is performed by the Ansatz. Mathematically, this is a unitary matrix multiplication $U(\vec{\theta})$ of size 8×8. Physically, this matrix is not calculated; it is *implemented* by the time evolution of the system under the effect of electromagnetic pulses (lasers or microwaves).

The global operation is:

$$|\psi_{out}(\vec{x}, \vec{\theta})\rangle = \left(\prod_{l=1}^{L} U_{\text{ENT}} \cdot U_{\text{ROT}}(\vec{\theta}_l) \right) |\psi_{in}(\vec{x})\rangle \tag{21}$$

Where:

- U_{ROT} are local rotations (equivalent to adjustable weights).
- U_{ENT} are CNOT gates (entanglement). Mathematically, these create off-diagonal terms in the density matrix, allowing the modeling of complex correlations inaccessible to a classical linear model.

4. From physics to algorithm

The algorithm cannot "read" the vector $|\psi_{out}\rangle$ (which would destroy the state). It must perform a projective measurement. The output value $f(\vec{x}, \vec{\theta})$ for the Z-axis is the expectation value of the Pauli-Z operator, defined by the scalar product:

$$\langle Z \rangle = \langle \psi_{out}(\vec{\theta}) | \sigma_z | \psi_{out}(\vec{\theta}) \rangle \tag{22}$$

It is this real value, derived from the statistical mean of multiple quantum "shots," that is sent back to the classical optimizer to calculate the Loss (1 - Fidelity) and update the parameters $\vec{\theta}$.

References

[1] Tony Pansera, Niels Minguez, and Hugo Virasak. Quantum state tomography repository. https://github.com/TonyPansera/Quantum-state-tomography, 2025. Code source et analyse utilisés pour ce projet.

[2] PennyLane Development Team. Pennylane: A cross-platform python library for quantum machine learning. https://pennylane.ai/, 2024. Xanadu Quantum Technologies.

[3] Qiskit Development Team. Qiskit: An open-source framework for quantum computing. https://qiskit.org/, 2024. Version 1.0.0 or latest used in project.