POLITECNICO DI TORINO

Master's Degree in Computer engineering

Master's Degree Thesis

Evaluation of a Quantum Kernel for Graph Classification on Neutral Atoms Quantum Computer

Supervisors

Prof. Bartolomeo MONTRUCCHIO

Dr. Edoardo GIUSTO

Giacomo VITALI

Chiara VERCELLINO

Candidate

Gabriele IURLARO

11 April 2024

Summary

This thesis merges two cutting-edge fields of study, Machine Learning, and Quantum Computing, combined for solving the graph classification problem.

Machine learning is the study of models and algorithms that "learn" through experience to solve problems, mimicking human intelligence. In the **first chapter**, a distilled version of the acquired knowledge is presented, starting from the Artificial Intelligence definition, and proceeding towards arriving at machine learning. Meanwhile, the different components of the method are presented, such as the dataset representation, the models, and metrics, arriving to define the scheme and the pipeline commonly employed for validating ML algorithms.

In **Chapter 2** the *Quantum Computing* field is introduced and discussed. The topic is introduced starting from a historical introduction, that shows the need for quantum computation for simulating quantum mechanics, and continuing with the quantum mechanics principles that inspired the quantum information theory. After a discussion about the possibilities and peculiarities of Quantum Computation, such as the *superposition*, *entanglement* and the most common operation, the focus is turned on the physical implementation of quantum computers, describing the actual most prominent platforms. At the end of the chapter, the main topic of the thesis is revealed, and the 4 different paradigms of *Quantum Machine Learning* are shown.

Between the presented technologies for realizing quantum computers, Neutral Atoms seem a prominent platform. In **Chapter 3**, Aquila, the quantum computer used in this work, is presented, highlighting the peculiarities of the analog approach, and pointing out the differences with digital mode, discussed in the previous chapter. The analog approach, and in particular the possibilities of Aquila's qubits' arbitrary position in the register and the control of the parameters of the waveform allows embedding a variety of graph problems. Finally, the thesis reaches its core, with the presentation of the Quantum Evolution Kernel, a graph kernel function for making graph classification using a Support Vector Machine. The full process is described, starting from the definition of the quantum dynamics through a topology-based

Hamiltonian, and arriving at the sampling technique for defining a probability distribution for the graph, used later for computing the kernel based on a commonly employed probability distribution distance metric.

The experimental setup is fully described in **Chapter 4**. First, an analysis of the dataset is conducted, and the preprocessing step is described. It consists of a unit disk embedding, necessary for executing the quantum routine on a real machine. Once the preprocessing is described, the full evaluation approach is described, comprising the optimization of the waveform parameter and the hyperparameter tuning of the Support Vector Machine. The performance is evaluated first on a reduced dataset, emulated on a classical machine, and then on the full dataset, simulated on a real quantum computer. Both the result of the emulation and the simulation are compared with a classical graph kernel, the Shortest Path graph kernel.

At the end, **Chapter 5** summarizes the whole work and presents some possible future works.

Ringraziamenti

Prima di iniziare, vorrei ringraziare il prof. Bartolomeo Montrucchio per la possibilità di svolgere la tesi su un tema così interessante e pionieristico. Inoltre, vorrei ringraziare in poche parole i corelatori, partendo da Edoardo, per avermi fatto appassionare al tema e per avermi insegnato le gioie e i dolori della ricerca. Inoltre, ringrazio Giacomo e Chiara, per la loro infinita disponibilità nel darmi spiegazioni e per il loro tempo, e specialmente per la grossa possibilità offertami da LINKS.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

ML
Machine Learning

NLP
Natural Language Processing

EDA
Exploratory Data Analysis

ERM
Empirical Risk Minimization

SVM
Support Vector Machine

RBF
Radial Basis Function

TP
True Positives

FP
False Positives

TN
True Negatives

FN

False Negatives

QC

Quantum Computing

QML

Quantum Machine Learning

VEQ

Variational Quantum Eigensolver

QAOA

Quantum Approximated Optimization Algorithm

NISQ

Noisy Intermediate-Scale Quantum Computers

NMR

Nuclear Magnetic Resonance

LOQC

linear optics quantum computation

PQC

Photonics Quantum Computing

QDK

Quantum Development Kit

FTQC

Fault-Tolerant Quantum Computing

CNOT

Controlled NOT

QVM

Quantum Virtual Machine

DAG

Directed Acyclic Graph

QVSM

Quantum SUpport Vector Machine

FPQA

Field Programmable Qubit Array

AOM

acousto-optical modulators

MIS

Maximum Independent Set

MWIS

Maximum Weighted Independent Set

QPU

Quantum Process Unit

NP-H

Non Determinist Polynomial Time Hard

UD

Unit Disk

CUDG

Constrained Unit Disk Graph

DEN

Distance Encoder Network

ELF

Embedding Loss Function

ReLU

Rectified Linear Unit

AWS

Amazon Web Service

SPK

Shortest Path Kernel

GP

Gaussian Process

Chapter 1

Background

Artificial Intelligence (AI) has emerged as one of the most promising fields of study in the vast world of computer science. Thanks to the recent advancements in fields like computer vision and natural language processing, it has turned trend a trend topic. It is difficult to understand in these huge amounts of information what really AI is. *Andrew Ng* (cofounder at GoogleBrain, head of the most followed Deep Learning course, Stanford professor), one of the most influential people in the field, has expressed in simple words the essence of what artificial intelligence is:

> "[...] the ability of machines to perform tasks that would normally require human intelligence."

This definition is intentionally general because AI is not so simple to define. While this work has been written, Artificial Intelligence applications are various and dynamic, showing the generality of the methods. The most impactful applications today include:

- **Image Processing**: From classification to object detection, there are various and interesting tasks involving images, used in self-driving cars, but also in the medical field.
- **Natural Language Processing**: used to process text, with various applications, from text classification to sentiment analysis including SPAM filters.
- **Generative AI**: it is a common name for a set of methods and algorithms that tries to generate data starting from a *prompt*. This includes Image and video generation, and both text generation.

Although being one of the most discussed topics today, the history of artificial Intelligence is much longer, and started in 1950, when the first computer where theorized and built. In 1950, Alan Turing [1] asked in his paper "Can machines

think?". Later from 1957 on, the first algorithm and the word "Artificial Intelligence" were written for the first time. However at that time, computers could only execute instructions without memory, and the cost was too high. From 1980, computers were able to execute tasks, and machine learning algorithms were developed and used, to successfully obtain results in speech recognition and natural language processing (in tasks like sentiment analysis). The third boom of AI comes in the first decade of the millennium, when the computational power of the computers allows the training of deeper models, capable of extracting knowledge from the most variety of data (non-structured, text, images). It begins the era of big data: the availability of a huge amount of training data allows deeper models to successfully solve a high variety of data. A taxonomy of Artificial intelligence, as highlighted in this paragraph, can be found in figure 1.1.

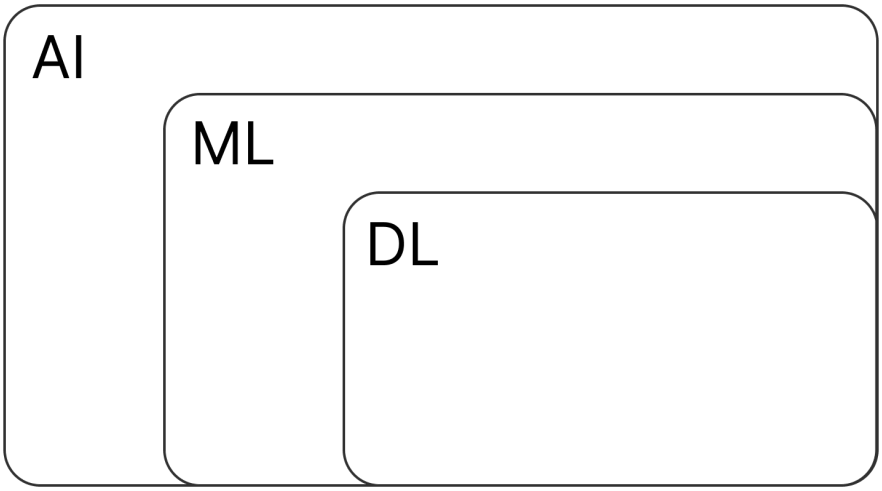


Figure 1.1: Difference between Artificial Intelligence, Machine Learning and Deep Learning

Under the artificial intelligence word, a lot of tasks and algorithms can be comprised. Algorithms that play games, like *DeepBlue* for the chess game, but also machine learning algorithms, that extract the knowledge directly from the data, without being programmed to do so (no explicit rules are required). Machine learning algorithms have another subfield: deep learning algorithms directly extract a representation of the data, without relying on the complex human-written preprocessing approaches, thanks to their deep structure, and thanks to the advent of Convolutional Neural Networks.

1.1 Machine Learning

Machine learning (ML) is a subset of the wide field of Artificial Intelligence, as described before. As AI, the Machine Learning objective is to build a machine, or algorithm, that intelligently solves problems. However, ML automatically learns how to solve the task, extracting a solution (or a strategy) directly from data, or from a representation of it. Tom Michael Mitchell, the father of Machine Learning, has given a definition of this described learning approach, in his book, considered the Bible of ML [2]:

> "A **computer program** is said to learn from **experience E** with respect to some class of **tasks T** and **performance measure P** if its performance at tasks in T, as measured by P, improves with experience E."

This definition gives a first intuition of which are the main differences between ML and the classical problem-solving approach. The classical way of solving a problem or solving a task $\mathcal{T}$, is to find an algorithm (as a sequence of instructions) that solves the problem for every input instance $\mathcal{D}$. Once the algorithm is found, one can apply the sequence of instructions on the specific instance of $\mathcal{T}$ in order to find the solution to the problem. Instead, the goal of Machine Learning is to automatically find a solution to a task extracting it directly from the data.

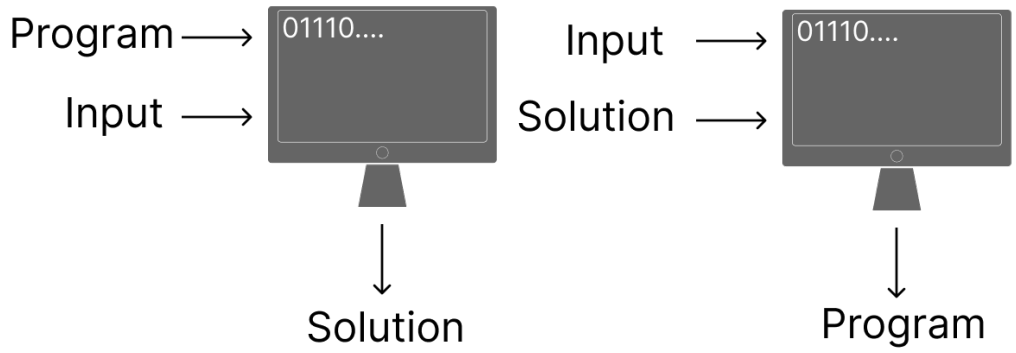


Figure 1.2: Differences between classical algorithmic problem solving and Machine Learning paradigm

Machine learning can be divided roughly into 3 main sub-fields, based on the learning algorithm and the type of data where they operate:

- **Supervised Learning**: the algorithm learns to solve a task T by learning from a set of pairs of data and solutions. Examples of Supervised Learning tasks are image classification and regression.

- **Unsupervised Learning**: instead of learning a mapping between data and labels, tries to infer information directly from data itself. Examples of Unsupervised Learning tasks are clustering and dimensionality reduction.
- **Reinforcement Learning**: Learn to accomplish a task by getting rewards or penalties while trying to solve it. It is usually used when the solution of the task corresponds to a series of choices, like in policy optimization (examples can be a Tic-Tac-Toe player or a control for a robotic arm).

Despite the differences between the paradigm, it is possible to define a set of components that are always present in a Machine Learning Task:

- A **set of data** (or **dataset**)$\mathcal{D}$. Examples of datasets are natural images paired with a content label, handwritten digits, proteins in the form of graphs, etc.
- A **Model** $\mathcal{M}_\theta$ of some parameter θ (called in some context weights) that describes the data in relation to the solutions.
- A **Risk function** (or **loss function**) $\mathcal{R}$, that specify of much the model $\mathcal{M}_\theta$ predictions differ from the real data $\mathcal{D}$. It can be interpreted in terms of model errors (i.e. predictions of the wrong label).
- An **Optimization Algorithm** $\mathcal{A}$ that starting from the dataset $\mathcal{D}$ minimizes the risk function $\mathcal{R}$ by tuning the model parameter θ. The process of learning (i.e. optimizing the parameter in order to minimize the risk function) is called **training**. In this context, the dataset takes the name of **training set**, or training data, and the parameters are called **learnable parameter**.

All the tasks and learning paradigms can be seen as minimizing the Risk function under the selected dataset by selecting a model and an Optimization algorithm that tunes the model parameter. In the following, the Supervised learning framework will be analyzed and detailed.

1.1.1 The mathematical framework

Consider a Supervised Learning task. In this case, the dataset $\mathcal{D}$ is defined as

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

of N paired instances, where every instances $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ is drawn from a distribution over $\mathcal{X} \times \mathcal{Y}$. $\mathcal{X}$ represent the *datapoint* space, while $\mathcal{Y}$ is the space of the label. Usually, the task $\mathcal{T}$ is expressed as a hidden mapping function $f : \mathcal{X} \to \mathcal{Y}$, such that

$$f(\mathbf{x}) = y \quad \forall\ (\mathbf{x}, y) \in \mathcal{D}$$

If the label space $\mathcal{Y}$ is categorical (i.e. the handwritten digits, the class of an image), the task takes the name of *classification*. Usually, it is common to have problems where the class is only 2. In this case, the task takes the name of *binary classification*, and one class is taken as reference and called *positive Hypothesis*, while the other one takes the name of *negative hypothesis*. If the label space is continuous, takes the name of *regression*.

In both cases, the model $\mathcal{M}_\theta$ can be seen as a parametric function of some parameter θ, $h(\mathbf{x};\theta) : \mathcal{X} \to \mathcal{Y}$, drawn from a class of function $\mathcal{H}$. The domain-specific knowledge, with an Exploratory Data Analysis (EDA) of the starting data, can give a hint on how to choose the class of function. The classifiers, or the class of functions, can be distinguished based on two main characteristics [3], generative or discriminative, probabilistic or not. The combination of these properties defines the different classes of classifiers:

- **Disciminant model**, in the original formulation of Bishop [3], is a model that directly constructs a mapping function $h(\mathbf{x}$ that maps the input feature vector directly to a label.

- **Discriminative non-probabilistic model**: Construct a function $h(\mathbf{x}$ that maps the input feature vector directly to a set of scores, one for each label.

- **Discriminative probabilistic model** Construct a function $h(\mathbf{x}$ that maps the input feature vector directly to a set of scores, one for each label, representing the *class posterior probabilities*:

  $$\mathbb{P}(y = y_k|\mathbf{x})$$

  representing the probabilities of the model belonging to a certain class label, conditional of the observation of the value $\mathbf{x}$.

- **Generative probabilistic model**: construct a function $h(x)$ by model the joint distribution of the feature vector $\mathbf{x}$ and the labels y $\mathbb{P}(\mathbf{x}, y)$. The posterior probabilities $\mathbb{P}(y = y_k|\mathbf{x})$ are computed according to the Bayes theorem:

  $$\mathbb{P}(C = c|\mathbf{X}) = \frac{\mathbb{P}(\mathbf{X} = \mathbf{x}|C = c)\mathbb{P}(c)}{\mathbb{P}(x)}$$

  Since $\mathbb{P}(c)$ and $\mathbb{P}(x)$ are assumed to be constant, and modeled as prior probabilities, the approach should learn only a statistical model that represents the probability of a sample given a certain label. This is the main reason because they are called generative models because one can sample a data point belonging to a certain class.

Classifiers, based on the form of the separation rule they create, can be divided into *linear classifiers* and *non-linear classifiers*. In general, since the data are highly non-linear, non-linear classifier performances are better, but the model complexity increases, raising the problem of *overfitting*.

The risk $\mathcal{R}$ can be expressed in several ways. In the following, we express it as a *loss function* $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, that numerically quantifies how much the model output $h(\mathbf{x}, \theta)$ is different from the expected output y:

$$\mathcal{R} = \mathbb{E}_{\mathcal{X}\times\mathcal{Y}}[\mathcal{L}(h(\mathbf{x};\theta), y]$$

Particularly relevant is the Bayes risk $\mathcal{R}^*$, defined as the infimum of all risk, all over the possible models and class models:

$$\mathcal{R} = \inf_h \mathcal{R}(h)$$

However, the risk defined as the expected value all over the data distribution cannot usually be analytically computed, it is often approximated with the *empirical risk*, defined as the loss function averaged all over the dataset. By minimizing the empirical risk with and optimization algorithm $\mathcal{A}$ one can obtain θ^*:

$$\theta^* = arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(h(\mathbf{x}_i;\theta), y_i)$$

The previously described framework takes the name of *Empirical Risk Minimization* (ERM) framework.

1.1.2 Support Vector Machines

Among the possible class of classifiers, Support Vector Machines (SVM) [4] are linear, discriminative, and non-probabilistic classifiers, that provide a geometric interpretation of the separation rule. In the simplest linear case, given an $m - dimensional$ input, the SVM algorithm finds $(m-1) - dimesional$ hyperplane that both separates the classes and provides the maximum separation margin.

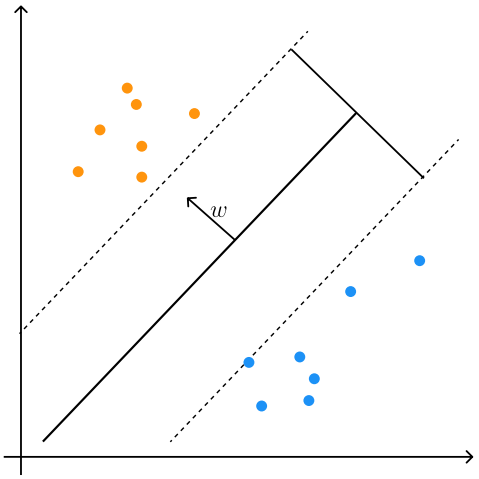


Figure 1.3: Support Vector Machine interpretation of the separation rule. The hyperplane is selected among the one that correctly separates the two classes with the maximum margin.

In the case of *linearly-separable* classes, SVM can provide the hyperplane that correctly classifies the samples with the max margin. However, it is not always possible to have a linear separation rule that correctly classifies all the samples. This problem can be overcome by including in the SVM objective function a penalty term, that counts the number of misclassified samples. This term can be interpreted as a *regularization term* (i.e. a term that controls how the model solution became big) and is usually weighted by a hyperparameter denoted with the letter C. The dual formulation of the Support Vector Machine problem is the following:

$$arg\max_{\alpha} \alpha^T \mathbf{1} - \frac{1}{2}\alpha^T \mathbf{H}\alpha \tag{1.1}$$

$$\text{s.t.} \quad 0 < \alpha_i < C \ \forall i$$
$$\sum_{i=1}^{n} \alpha_i z_i = 0$$

Since the prediction rule is based only on scalar product, it is possible to transform the feature space, increasing the dimensionality of the data and moving into spaces where the data characteristics are different. For example, consider a problem where data points are distributed as in figure 1.4.

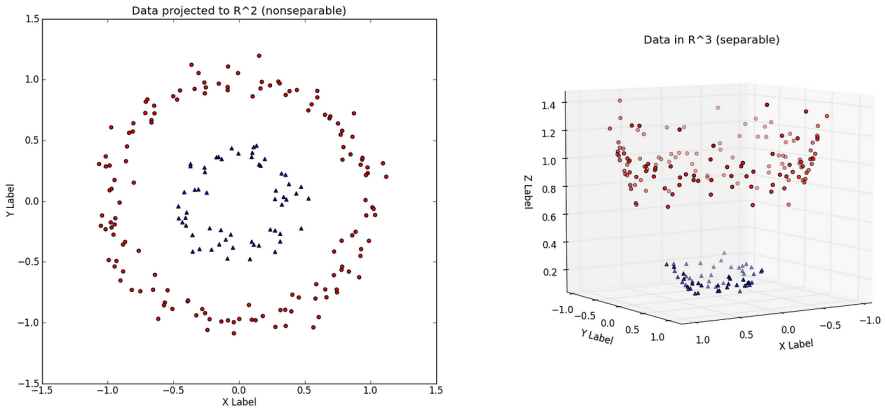


Figure 1.4: Non-linear separable data point and the corresponding transformed features, that are linearly separable.

In this case, there does not exist a liner hyperplane that correctly classifies the sample to each class. However, it is possible to imagine a separation rule, by looking at the distribution. By applying a non-linear transformation (in this case a transformation from $\mathbb{R}^2 \to \mathbb{R}^3$ where the z values depend on the distance of the point from the center), it is possible to find a linear separating rule. This approach can be applied to several methods, and consists of:

1. Starting from a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, a transformed dataset is built according to a transformation $\phi : \mathcal{X} \to \mathcal{X}'$, $\mathcal{D}' = \{(\phi(x_i), y_i)\}_{i=1}^N$.

2. The selected algorithm is applied in the transformed feature space $\mathcal{X}'$, inducing a non-linear separation rule.

In the case of Support Vector Machines, this procedure takes the name *kernel-trick*. Recalling the dual formulation of the SVM problem in 1.1, it is easy to notice that the prediction rule and the objective can be expressed in terms of scalar products of the datapoints features:

$$H_{ij} = z_i z_j \mathbf{x}_i^T \cdot \mathbf{x}_j$$

In the case of the protocol for making non-linear classification presented before, the matrix H (called **Kernel matrix**), has the following structure:

$$H_{ij} = z_i z_j \phi(\mathbf{x}_i)^T \cdot \phi(\mathbf{x}_j)$$

it is possible to define a function $k(\cdot, \cdot)$ that efficiently computes the scalar product into the transformed space, called **kernel function**:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \cdot \phi(\mathbf{x}_j)$$

it is possible to redefine the Kernel matrix H in terms of the kernel function between the datapoints features, obtaining a non-linear separation rule without explicitly computing the transformation.

$$H_{ij} = z_i z_j k(\mathbf{x}_i, \mathbf{x}_j)$$

Common kernel functions are:

- **Polinomial kernel** of degree d: $k(\mathbf{x}_i, \mathbf{x}_j) = (x_i^T x_j + 1)^d$
- **Gaussian RBF Kernel** $k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$, associated to an infinite dimensional transformed space.

The possibility of having a set of kernel functions for different transformations is one of the reasons why SVM is one of the most used shallow algorithms. Kernel functions allow faster computation since made in a single shot at the transformation step and the scalar product, which has a linear complexity in the transformed feature space. In addition, kernel function can be defined[1] for different objects rather than datapoints (a notable example is graph kernels). Another useful characteristic of SVM that comes directly from the problem definition is the possibility to incorporate different scores for the misclassification cost of each class. This is very useful in the case of class imbalance, and usually, the cost reflects the probability of obtaining the sample from the training distribution.

1.1.3 Metrics

The empirical risk, or specifically loss function cannot always directly quantify model performances in a human-readable fashion. For these reasons, different metrics have been developed in order to compare and choose models. In order to define a set of metrics, we consider a binary classification problem, with labels $\mathcal{H}_T$ corresponding to the true class (or class 0) and $\mathcal{H}_F$ corresponding to the false class (or class 1).

A commonly employed metric for classification is **accuracy** (or its complemented at 1, the **error rate**), defined as the number of correctly classified samples against the total number of samples. However, the accuracy, while being understandable, can be misleading in the case of class imbalance. Consider an example where we have 100 samples, 15 belonging to class 0 and 85 belonging to class 1. Let's consider a model that predicts according to this table, called **confusion matrix**:

[1]Notice that not every function can be a kernel. Mercer's condition provides a sufficient condition for $k(\cdot, \cdot)$ to be a kernel function.

	$\mathcal{H}_T$	$\mathcal{H}_F$
Predicted T	10	16
Predicted F	5	69

In this case, the accuracy of the model is given by the sum of the elements of the main diagonal divided by the total number of elements ($acc = 79\%$), which can be considered good. However, a dummy model that predicts always class 1, achieves an accuracy of 85%. This simple example shows how accuracy metrics do not well suit the case of imbalanced datasets. The confusion matrix shown in the example is a powerful tool for evaluating model performances since is a complete "dashboard" of the predicted values. The confusion matrix definition is the following:

Figure 1.5: Confusion matrix definition. The values on the row correspond to the predictions, while values on the columns correspond to the actual class labels.

- *True Positive* (TP): correctly predicted class 0
- *False Positive* (FP): predicted class 0 but belonging to class 1
- *True Negative* (TN): correctly predicted class 1
- *False Negative* (FN): predicted class 1 but belonging to class 0

Starting from the confusion matrix, a series of metrics can be defined:

- **Accuracy**:
$$acc = \frac{TP + TN}{TP + TN + FN + FP}$$
- **Precision**:
$$\text{precision} = \frac{TP}{TP + FP}$$
- **Recall**:
$$\text{recall} = \frac{TP}{TP + FN}$$

- **F-β score**

$$F_\beta = \frac{(1+\beta^2)TP}{(1+\beta^2)TP + FP + \beta^2 FN}$$

  The most commonly employed is with $\beta = 1$, and corresponds to the harmonic mean between the precision and the recall.

Each of these metrics reflects a different objective. For example, in the medical field (i.e. tumor classification) is important to minimize the false negative instead of the false positive. For this reason, having a high recall value is more important and valuable rather than having a high precision. Instead, if we imagine a hypothetical system that decides if a person goes to jail or not, minimizing the people that go to jail even if they didn't commit crimes has the priority. This problem highlights that having a higher precision is preferred to having a high recall.

1.1.4 Machine learning pipeline

The cores of the machine learning algorithm are the model $\mathcal{M}$ and the optimization algorithm $\mathcal{A}$ used for computing the optimal parameters θ. Despite being so important, the performances and the applicability of machine learning algorithms require a set of other components (some of them are already explained in previous sections) carefully connected in order to produce the **machine learning pipeline**. Details on a possible pipeline are presented in figure 1.6.

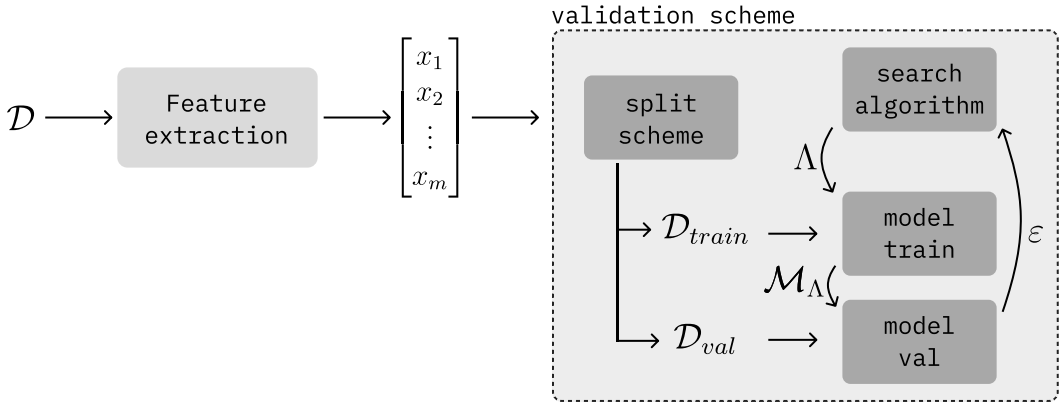


Figure 1.6: A full Machine Learning pipeline

In the supervised learning setting, the dataset consists of pairs of objects and an associated label. Data can be the most varied, representing images, or objects. The first step consists of the so-called *feature extraction*: a human-written

preprocessing routine that computes a fixed-size representation of the object, called *features*. Usually, the features are represented by continuous values (i.e. a feature $\mathbf{x}$ is a *datapoint* in an m-dimensional Euclidean space $\mathbb{R}^m$), while the labels are mapped to an integer belonging to $[0, l]$, where l is the number of different labels. Features are a middle and compressed representation of the data, and the feature extraction algorithms are designed to provide good separability and catch most of the characteristics of the object. Once the dataset is prepared as a list of data points, it enters into the validation scheme. The validation scheme is required for both validating the model performances, training the model, and choosing the parameters. Most models have 2 sets of parameters:

- *Model parameters* θ: optimized starting from the data based on an optimization algorithm.

- *Model Hyperparameters*: these are parameters that characterize the model but cannot be directly optimized using the optimization algorithm and need to be tuned.

The process of choosing the hyperparameter is called *hyperparameter tuning* or *model selection*. Often, this process is a trial and error, and several methods have been developed. The most used in the context where the hyperparameter is not many (maximum 3 or 4) is *grid search*. Consider a set of K hyperparameters $\Lambda = \{\lambda_1, \cdots, \lambda_K\}$, with each hyperparameter λ_i has a set of M_i possible values $\lambda_i \in [v_1^{\lambda_i}, \cdots, v_{M_i}^{\lambda_i}]$. The grid search consists of training one model for every set of hyperparameters belonging to the cartesian product $\{\lambda_1 \times \cdots \times \lambda_K\}$, and obtaining a score based on some of the previously defined metrics. The best model is the one associated with the hyperparameters that provide the best score.

This process can be very costly since requires training $\prod_{i=1}^{K} M_i$ models. Other methods consist of random sampling or heuristics search from the full grid search until a stopping criterion happens. Stopping criteria could be different, based on performance metrics or on execution time.

As seen before, each class of models is characterized by its interpretation of the data (that in general are called *model assumptions*) and by the interpretation of the output (i.e. probabilistic vs. non-probabilistic score). In addition, models are characterized by the separation rule, which directly reflects the model's complexity. More complex models (i.e. with several non-linearities) are capable of modeling more complex data.

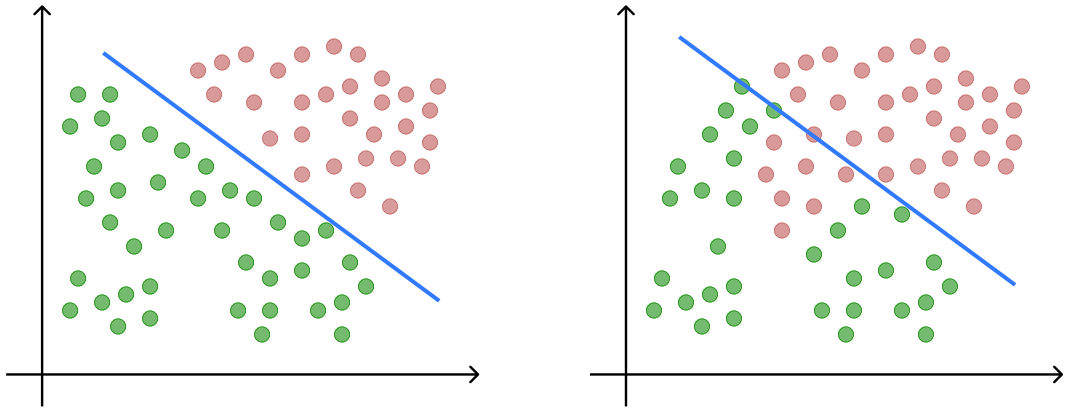


Figure 1.7: Complex data needs complex models

However, different problems can arise when the data assumptions are not met on the training data. In addition, the model performances cannot suffer from the incompatibility of the data assumptions, especially when the model is very complex, making it difficult to fully understand the origin of the problem. In order to overcome this problem, and fairly evaluate the model, is common to split the dataset $\mathcal{D}$ into 2 parts:

- **Training dataset** $\mathcal{D}_T$: used to train the model $\mathcal{M}_\theta$ and compute the parameters θ
- **Validation dataset** $\mathcal{D}_V$: used to evaluate model performances, and to tune the hyperparameters.

The validation data is not used to train the model, this allows complete independence of the model parameter on the performances. There are different strategies to split the dataset. As a practical consideration, valid for every method, is common to first shuffle the dataset, in order to avoid biases in the data collection process. The most used strategies are:

- **Train-val split**: starting from the original dataset $\mathcal{D}$, it is split into 2 partitions (based on a split ratio), one will be the training dataset, the other one the validation dataset. This method is well-suited when the data is not scarce, or when the model training process is long.
- **K-fold cross validation**: When the amount of data is scarce, splitting is not the best choice. Instead, It could be convenient to split the dataset into K splits (or folds). The procedure consists of K iteration. At each iteration, a model is trained on $K-1$ folds and evaluated on the remaining fold. In every iteration, the training set, and the validation set change. In the end,

the produced score can be aggregated or averaged to produce a score for the model. This method, although widely used, has the drawback of training the models K times.

- **Leave-one-out**: it is a variant of the K-fold cross-validation approach where K is set to the total number of samples. In this case, at each iteration, the validation set includes only a sample.

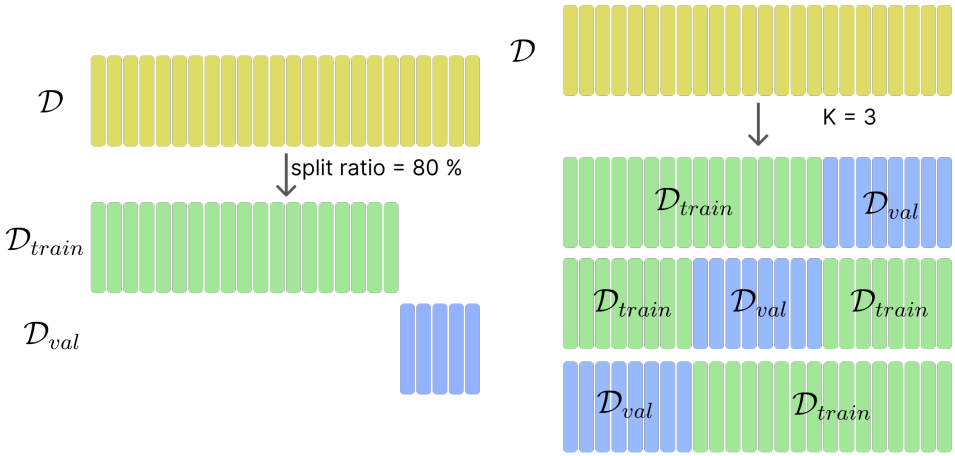


Figure 1.8: Differences between train-val split and K-fold cross validation

However, different problems can arise when model complexity does not fit data complexity. In particular, when the model complexity is too high, the model tends to *overfit* the training data. The term *overfitting* indicates a phenomenon that can arise for different reasons but is characterized by high performances on the training dataset, but poor performances on the validation dataset. The main cause can be the model complexity: if the data are simple to model (let's say, a linear model can approximate it), using a more complex model makes the model overspecialized on the training data. An example of what overfitting means can be found in figure 1.9.

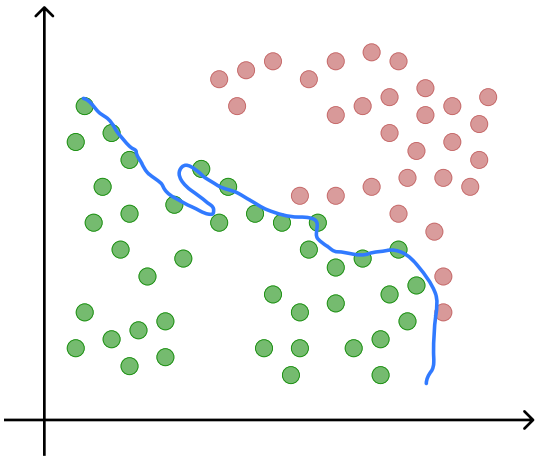


Figure 1.9: Overfitting due to model complexity

Overfitting is also described as a lack of generalization. The model, in fact, is not able to correctly perform the task when the data are slightly different (and unseen). On the other hand, when the model complexity is lower than the data complexity, there is a problem of *underfitting*. In this case, the classifiers are not able to capture the data relations, generating poor performances both on training and validation data.

In general, when performing hyperparameter tuning and model selection, it is important to take care of overfitting and underfitting, and consequently observe the model performances on both datasets. Sometimes, is better to choose the best model on the training set, since this model can lack generality, but considering the validation accuracy. As a general rule, it is better to choose the simplest model that correctly fits the data distributions, as measured by the performance metrics.

Chapter 2

Quantum computing

Two scientists were discussing a new quantum computer that could solve any problem instantly. One said, "Can I see it?" The other replied, "Sure, but only when you're not looking."
-Anectode circulating in the 80s [5]

2.1 An Historical introduction

While being a trending topic today, the origin of quantum computation is not so recent. Quantum mechanics started at the beginning of the last century, thanks to Max Plank, and since then, several theoretical physicists have investigated the law of nature at the tiniest scale, understanding that Newton's classical mechanics does not work anymore. It was 1984, when Richard Feynman, at the **Physics of Computation Conference**, uttered:

> *"Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy."*

Feynman, for the first time, highlighted the need for quantum computers as simulators of quantum systems. It was clear to him that simulating on classical machines could not be feasible, requiring an exponential number of resources. Only one year later, Paul Benioff describes the model of a Quantum Turing Machine [6], opening the possibility of creating a model of computation that uses quantum mechanics as driving principles.

In 1985, David Deutsch opened the era of *quantum algorithms*, publishing his model for a universal quantum computer [7], describing for the first time a problem that could be solved using a quantum computer. From now on, plenty of problems were defined as being solvable with quantum computers (like Deutsch, Deutsch-Jozsa [8], and Simon's algorithm [9]).

But the revolution came when Peter Shor, in 1994, published his algorithm, capable of solving a classically hard problem, integer prime number factorization. For the first time, someone demonstrated that Quantum Computing could be used to solve real-life problems, enabling an exponential speedup. In the same year, Grover publish his famous algorithm for searching, providing a polynomial speed-up. From then on, the interest in this technology grew, with companies starting to develop quantum computers. In 1996, David DiVincenzo [10] describes the criteria for building a usable quantum computer. Starting from the late 90s, and continuing in the first decade of 2000, different companies propose different technologies for building quantum computers. Among these, the biggest ones are IBM, Google, Rigetti, Intel, and many others, each one providing quantum computers with more and more qubits, and promoting the *quantum supremacy*. However, current devices belong to the *Noisy Intermediate-Scale Quantum Computer* (NISQ) era, with a modest number of qubits and short coherence time, that does not allow to apply quantum algorithms (like Shor algorithms). For this reason, hybrid quantum-classical algorithms have been developed, that take some advantages of quantum computing, but with the reliability of classical computing. Such algorithms include QAOA, VQE, and Quantum Machine Learning (QML). Following this wave, other companies started to build quantum computers, introducing the era of *cloud quantum computing*: from 2019 with IBM, it is possible to execute on real quantum hardware by submitting jobs. The race for quantum computers has just started: in the next years, we could expect quantum computers with more qubits, and error-correction codes will enable execution with high-fidelity quantum algorithms.

2.2 Quantum information theory

In the classical information theory of Claude Shannon, the littlest unit of information is the *bit*, a logical unit that can deterministically assume value 0 or 1. The base of Quantum Information Theory is the *qubit*, a Quantum Information unit representing a two-level (the two classical states, common to the bit, 0 and 1) quantum system, that by definition follows the principles of quantum mechanics. In order to understand the main differences between bits and qubits, can be useful to list and comment on the axioms of quantum mechanics, 5 experimentally proven principles [11]:

1. The properties of a quantum system are completely defined by the specification of its state vector ψ. The state vector is an element of a complex Hilbert space $\mathcal{H}$ called the space of states.

2. With every physical property $\mathcal{A}$ (energy, position, momentum, angular momentum, ...) there exists an associated linear, Hermitian operator A (usually

called observable), which acts in the space of states. The eigenvalues of the operator are the possible values of the physical properties.

3. (a) **Born rule**: If $|\psi\rangle$ is the vector representing the state of a system and if $|\phi\rangle$ represents another physical state, there exists a probability $p(|\psi\rangle, |\phi\rangle)$ of finding $|\phi\rangle$ in $|\psi\rangle$, which is given by the squared modulus of the scalar product on $\mathcal{H}$: $p(|\psi\rangle, |\phi\rangle) = |\langle\psi|\phi\rangle|^2$

   (b) **Wave function collapse** If $\mathcal{A}$ is an observable with eigenvalues a_k and eigenvectors $|k\rangle$ ($A|k\rangle = a_k|k\rangle$), given a system in the state $|\psi\rangle$, the probability of obtaining a_k as the outcome of the measurement of a is $p(a_k) = |\langle K|\psi\rangle|^2$. After the measurement, the system is left in the state projected on the subspace of the eigenvalue a_k.

4. **Time evolution** The evolution of a closed system is unitary. The state vector $|\psi(t)\rangle$ at the time t is derived from the state vector $|\psi(t_0)\rangle$ at the time t_0 by applying a unitary operator $U(t, t_0)$, called the *evolution operator*: $|\psi(t)\rangle = U(t, t_0)|\psi(t_0)\rangle$

As anticipated before, the qubits, short name for *Quantum bits* (the name was chosen by Schumacher in [12]) are mathematical objects that describe a 2-state quantum system. With respect to classical bits, qubits have some peculiarities typical of quantum mechanics.

- *Superposition*: the state of the qubit is a combination (i.e. a *superposition* of the basis state $|0\rangle$ and $|1\rangle$, called **computational basis**.
- *Entanglement*: the state of a single qubit cannot be represented without considering the whole system state.
- *Interference*: similar to optical interference, probability distributions determined by wavefunctions of quantum states are impacted by the constructive or destructive interference phenomenon.

2.2.1 Quantum bits

Quantum bits can be described by a state vector belonging to the complex Hilbert space $\mathbb{C}^2$, as stated by the first axiom of quantum mechanics. Each element (also called *amplitude*) of the state vector is a complex number, that represents the probability of the qubit of being measured in the corresponding state. A generic qubit state $|\psi\rangle$ is a linear combination (i.e., *superposition* of basis states $|0\rangle$ and $|1\rangle$):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

where α and β are complex numbers such that $||\alpha||^2 + ||\beta||^2 = 1$. The notation used is the *Dirac notation* [13], usually called "bra-ket". The Dirac notation is a shortcut for naming column and row vectors. Let's consider the binary alphabet $\sum = \{0, 1\}$, where the order of the elements is important. Then, the column vector named "ket-0" indicated with:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

is the column vector with a 1 in the position of the element. "bra-one" indicates the row vector with a 1 in the position of 1 in the alphabet:

$$\langle 1| = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

The Dirac notation can be used to indicate every possible state and star vector, by a linear combination of the basis vector, as in 2.1. These means that the associated *statevector* with $|\psi\rangle$ is:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Since the components of the statevector are complex numbers, each state is described by 4 real numbers. By exploiting $||\alpha||^2 + ||\beta||^2 = 1$, we can rewrite the state $|\psi\rangle$ as:

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

Where γ, θ, ϕ are real numbers. We can discard the term $e^{i\gamma}$ since the global phase has no observable effect (evidence of this can be found in the measurement section). By interpreting θ as the *polar angle* and ϕ as the *azimuthal angle*, a quantum state can be mapped to a point on the surface of a unit sphere in $\mathbb{R}^3$. This representation takes the name of *Bloch sphere*, by his ideator Felix Bloch [14], who creates it to represent the transformation of a 2-level quantum system in a much clearer way. Every quantum state belongs to the surface of the state and is called *pure state*. A state that does not belong to the Bloch sphere surface is called *mixed state*.

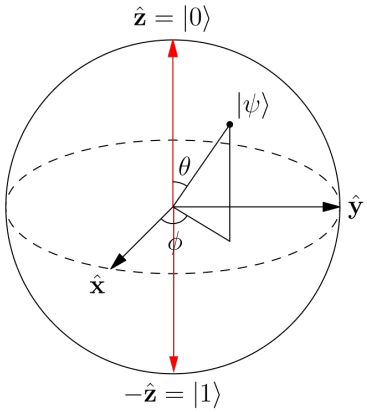


Figure 2.1: Bloch sphere [15] representing a generic state $|\psi\rangle$. The highlighted 2 antipodal state represent the computational basis vector $|0\rangle$ (northen pole) and $|1\rangle$ (south pole).

The Bloch sphere has at its pole the two states of the computational basis $|0\rangle$ and $|1\rangle$. Every state of a single qubit can be represented on the sphere. Bloch sphere can give a hint of what *superposition* and the *wave function collapse* principle means. Two important states that exhibit the superposition characteristics are:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.2}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.3}$$

Called respectively the *plus state*, and its antipodal respect to the X axis, the *minus state*.

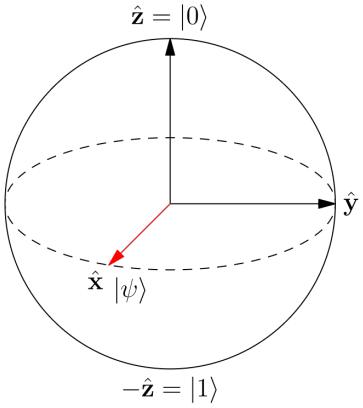


Figure 2.2: Plus state on the Bloch sphere, lying on the equator.

Both these states represent a state that is not 0 or 1, but a mixture of the 2 bases. Intuitively, it is possible to notice that the state on the sphere (represented in figure 2.2) is equidistant to both poles. These can give a hint on what the measuring outcome could be: since it is equidistant, the plus and minus state outcome of the measure can be equiprobable 0 or 1. In fact, considering the plus state:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle)$$

$$\alpha = \frac{1}{\sqrt{2}},\ \ \|\alpha\|^2 = \frac{1}{2}$$

$$\beta = \frac{1}{\sqrt{2}},\ \ \|\beta\|^2 = \frac{1}{2}$$

Bloch sphere can be a good tool for visualizing one qubit, although generalizing it for two or more is not so trivial.

2.2.2 Multiple qubits

Until now, single qubit systems have been described. As for classical computation, different qubits can be packed together, forming a *quantum register*. In this context, qubits exhibit an interesting behavior, not fully understood by physicists, called *entanglement*. Let's start by considering a 2-qubit system. Each qubit is in a superposition of 2 states. A general 2 qubit system can be described by its statevector:

$$|\psi\rangle = \alpha_0\,|00\rangle + \alpha_1\,|01\rangle + \alpha_2\,|10\rangle + \alpha_3\,|11\rangle \tag{2.4}$$

Where each of the α_i is a complex number, and their square modulo represents the probability of the associated outcome. Since $|\psi\rangle$ is a quantum system, it belongs to the Hilbert space $\mathcal{H}$, so it must have the same properties of the single system (i.e., the statevector component squared modulo should sum up to 1). it is easy to notice that the statevector dimension grows exponentially with the number of qubits in the register. In particular, if there are N qubits in the register, the statevector will have 2^N component. Multiple qubit states can be divided into two categories:

- *Product states*: Consider a set of N qubits, each in the state described the statevector $|\psi_i\rangle = \alpha_i\,|0\rangle + \beta_i\,|1\rangle$. If the overall state can be described by the **tensor product** of the single qubit state:

  $$|\psi\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_N\rangle$$

  This means that the amplitudes of the statevector can be rewritten in terms of products of the single qubits amplitudes. For this reason, *product states* are called also *simply separable states*.

- *Entangled states*: are the states characterized by a set of amplitudes that cannot be factorized based on their single qubits amplitude. The qubits of this system are linked together in a mathematical (and physical way), and behave as a single system.

Entanglement is a key phenomenon in quantum computing. Entangled states cannot be separated, and performing an operation on one qubit will cause an instantaneous consequence on the other qubit. However, entanglement is not only a mathematical consequence of the selected framework. In 1935, Einstein, Podolski, and Rosen published an article about the completeness of Quantum Mechanics, where they present the EPR paradox [16], a mental experiment that shows the non-completeness of QM. The experiment consists of the preparation of two particles in an *entangled* state, that are separated in space. Later a person measures the state of one of the particles, and due to the entanglement, the other particle collapses to the other state. This can be seen as a violation of the locality principle, because of the information of the state propagating at a speed higher than light. The paradox was solved in 1964 by Bell [17], together with the formulation of the *non-communication-theorem*. The entangled states described by the EPR paradox (in the Bohm version) are called the EPR states or the Bell States:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad |\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \tag{2.5}$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \quad |\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \tag{2.6}$$

Takes as reference the $|\Phi^+\rangle$ state. it is possible to expand the product, remembering the general form of a 2-qubit product state:

$$\begin{aligned} |\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) = \\ = \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \alpha_2\beta_1 |10\rangle + \beta_1\beta_2 |11\rangle \end{aligned}$$

The system of equations to solve is:

$$\alpha_1\alpha_2 = \frac{1}{\sqrt{2}}, \quad \alpha_1\beta_2 = 0, \quad \alpha_2\beta_1 = 0, \quad \beta_1\beta_2 = \frac{1}{\sqrt{2}}$$

it is easy to notice that this system of equations cannot have a solution. These prove that $|\Phi^+\rangle$ is an *entangled* state. The same consideration can be done for the other Bell states. Bell states and in particular *entanglement* is the key for some important quantum algorithms such as quantum teleportation and dense coding, that take advantage of the strong correlation (i.e., the entanglement) among the qubits.

2.2.3 Operators

As for classical information theory, also quantum computer has its set of operations in the form of gates, called *quantum gates*, the quantum analogy of the logic gates. Quantum gates are formally called *operators*, which act on the quantum state. Quantum operators, once a basis is chosen, are represented by square matrices of dimension n. For single qubit gates, n is equal to 2. A matrix, to represent a quantum gate should be a *Unitary operator* U, and should satisfy the following constraint:

- *Linearity*: $U|\psi\rangle$ distributes among the components of the superposition state, i.e. $U|\psi\rangle = U(\alpha|0\rangle + \beta|1\rangle) = \alpha U|0\rangle + \beta U|1\rangle$.
- *Bounded*: applying an operator on a quantum state returns a quantum state. This means that $\| U|\psi\rangle \|^2 = 1$.
- *Inverse operator*: if exist a quantum operator U, there must exist the inverse operator $U^\dagger$ (Hermitian Conjucate) such that $UU^\dagger = I$, where I is the identity operator $I|\psi\rangle = \psi$.

Again for single qubit gates, the Bloch sphere can help visualize the effect of the operator, since every single qubit operation can be seen as a rotation around one of the axes. Particularly important for quantum computing, and in general for quantum mechanics, are the four *Pauli operators* (or *Pauli matrices*).

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.7}$$

Where I is the identity, and X gate is commonly known as the NOT operator. Applying an operator to a qubit is done by means of matrix-vector multiplication (in this example, the application of a NOT gate to the $|0\rangle$ state):

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

Obtaining the $|1\rangle$ (this is the reason why X is called NOT gate). Another important operator, the first responsible for superposition, is the *Hadamard* gate:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.8}$$

Hadarmard gate application on the basis state is:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = |+\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = |-\rangle$$

So, the Hadamard gate transforms the classical state $|0\rangle$, in the superposition state $|+\rangle$. All the previously defined operators are *Hermitian*, which means that applying twice the X, Y, Z, or H gates takes again to the starting point. Other commonly

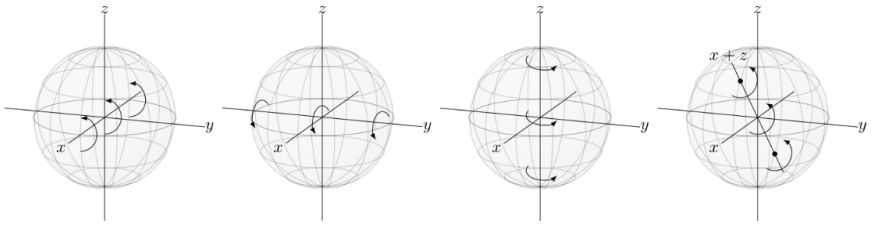


Figure 2.3: X, Y, Z, H gates and their effect representing on the bloch sphere.

employed single qubits operators are the S and T gate, respectively corresponding to a rotation of $\pi/2$ and $\pi/4$ around the Z-axis:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \tag{2.9}$$

Intuitively, these operators are not Hermitian, since 4 S gates are required to make a full rotation around the Z-axis. There exists also a certain number of n-qubit operators (common number are 2 or 3). One of the simplest is the SWAP gate. SWAP gates simply perform a swap operation between qubits:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.10}$$

Another important gate, responsible for the construction of entangled gate is the C-NOT (Controlled-NOT), defined by the following 4×4 matrix:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.11}$$

CNOT gate is called also the "quantum XOR", because if applied to a pair of qubits returns the XOR function between them, following:

$$CNOT\,|\psi\rangle\,|\phi\rangle = |\psi\rangle\,|\psi \oplus \phi\rangle$$

CNOT belongs to a family of gates called "controlled unitary operator", OR CU gates, that have the form:

$$CU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{pmatrix} \tag{2.12}$$

The CNOT gate can be derived from 2.12, by set $U = X$. In the context of controlled gates, order counts: the first qubit is the *control* qubit, while the second in the *target* qubit. Another interesting property of the CNOT gate is the possibility of generating entangled states. For example, let's consider the following quantum circuit (a sequence of quantum gates) in figure 2.4:

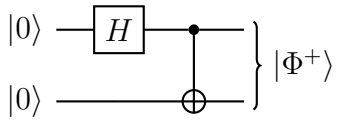


Figure 2.4: Quantum circuit for generating the Bell state

The first Hadamard gate transform the state $|00\rangle$ into the superposition

$$\frac{(|0\rangle|0\rangle + |1\rangle|0\rangle)}{\sqrt{2}}$$

. Then, the CNOT gate flips the logical value of the second bit when the first bit is $|1\rangle$ (in this case, the second part of the superposition state, $|1\rangle|0\rangle \rightarrow |1\rangle|1\rangle$), obtaining:

$$\frac{(|0\rangle|0\rangle + |1\rangle|1\rangle)}{\sqrt{2}} = |\Phi^+\rangle$$

This is one of the Bell states. From this example, it is clear that the CNOT gate plays a crucial role in quantum computing and quantum algorithms, thanks to its ability to create entangled states.
The last multiple qubit gate presented in this section is the *Toffoli* gate, or CCNOT (Controlled-Controlled NOT), which is a 3 qubit gate, corresponding to a controlled

CNOT. His matrix form consists of a 8×8 matrix defined like this:

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.13}$$

2.2.4 Measurements

Qubits exhibit the peculiarity of existing in a superposition of 2 states, each with a probability associated. Considering a state

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

, the Born's rule state that the probability of outcome 0 is $||\alpha||^2$ and the probability for the outcome 1 is $||\beta||^2$. The coefficient of the statevector (i.e. α and β) expresses the probability of the outcome of the measurement in the **Z** basis, or the computational basis. By expressing the basis in terms of other states, it is possible to measure on a other basis than the computational. According to the *wave function collapse* axiom, after the measurement process (if it is not destructive) the state collapses to the measured state (i.e. loses its quantum information and becomes a classical state). The outcome of further measurement will be always the same. These behaviors are well characterized in the mathematical framework of quantum computing. Let's consider a quantum state $|\psi\rangle$. Assuming an observable $\mathcal{A}$ with eigenvalues a_k associated with eigenvector k, the probability of observing a_k is:

$$\mathbb{P}(a_k) = |\,\langle k|\psi\rangle\,|^2$$

according to the Born rule. Let's consider k being one of the vectors of the computational basis, either $|0\rangle$ or $|1\rangle$. For example, considering the superposition state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, the probability of measuring respectively the classical state $|0\rangle$ or $|1\rangle$ is:

$$\mathbb{P}(|0\rangle) = |\,\langle 0|\psi\rangle\,|^2 = \left|\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix}\right|^2 = |\alpha|^2$$

$$\mathbb{P}(|1\rangle) = |\,\langle 1|\psi\rangle\,|^2 = \left|\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix}\right|^2 = |\beta|^2$$

These measurements are *projective measurement* (also called Von Neumann measurement) in the computational basis or **Z** basis. it is possible to measure in other bases rather than the computational basis.

Let's consider $M_k = |k\rangle \langle k|$ is a *projective operator* (or measurement operator). The probability of the state ψ to be observed in the state k is:

$$\mathbb{P}(k) = \langle \psi | M_k^\dagger M_k | \psi \rangle$$

where $M_k^\dagger$ is the complex Hermitian Conjugate. According to the Born rule, the state collapses into:

$$|\psi\rangle \to \frac{M_k |\psi\rangle}{\sqrt{\langle \psi | M_k^\dagger M_k | \psi \rangle}} \tag{2.14}$$

For example, the measurement in the **Z** basis, is defined by the set of the 2 measurement operators:

$$M_0 = |0\rangle \langle 0|, \quad M_1 = |1\rangle \langle 1| \tag{2.15}$$

As an example, the probability distribution for the usual $|\psi\rangle$ in the case of $|0\rangle$:

$$\begin{aligned} \langle \psi | M_0^\dagger M_0 | \psi \rangle = \langle \psi | 0 \rangle \langle 0 | 0 \rangle \langle 0 | \psi \rangle = \langle \psi | 0 \rangle \langle 0 | \psi \rangle = \\ = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha^* \alpha = \|\alpha\|^2 \end{aligned}$$

According to the axiom of quantum mechanics previously stated. The state collapses to

$$|\psi\rangle \to \frac{M_k |\psi\rangle}{\sqrt{\langle \psi | M_k^\dagger M_k | \psi \rangle}} = \frac{|0\rangle \langle 0 | \psi \rangle}{\|\alpha\|} = \frac{\alpha}{\|\alpha\|} |0\rangle \simeq |0\rangle$$

showing the wave function collapsing to the $|0\rangle$ state. Other possible and used bases are the **X** basis, consisting of the plus $|+\rangle$ and minus $|-\rangle$, and the **Y** basis, consisting of $|i\rangle$ and $|-i\rangle$ states. Once the measurement mathematical framework has been detailed, let's introduce a concept that has been used before, when deriving the Bloch sphere. Consider two quantum state $|\psi\rangle$ and $|\phi\rangle$, such that:

$$|\psi\rangle = \alpha |\phi\rangle$$

Since α is a complex number such that $\|\alpha\|^2 = 1$, so it can be expressed in polar coordinates $\alpha = e^{i\theta}$, for some real number $\theta \in [0, 2\pi]$. For this reason, the two states differ by a *global phase* [18]. State that differs only from a global phase are indistinguishable (from a measurement point of view):

$$\| M_k |\psi\rangle \|^2 = \| M_k \alpha |\phi\rangle \|^2 = \|\alpha\| \| M_k |\phi\rangle \|^2 = \| M_k |\phi\rangle \|$$

2.2.5 Quantum circuits

One way of representing quantum algorithms is by means of *quantum circuits*. Quantum circuits are simply a sequence of operations (represented by box, or gates) and wire, each for each qubit in the quantum register. Every circuit can be modeled as a Directed Acyclic Graph (DAG). Operations are executed from left to right, and each row (or wire) represents a qubit. Some examples of gates can be found in figure 2.5.

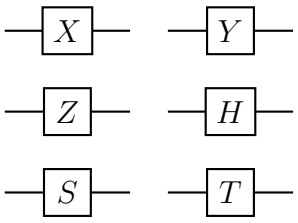


Figure 2.5: Single qubit quantum gates. In order (top-down from left to right) the X, Y, and Z Pauli operators, the Hadamard gate, and the S and T gate.

Other commonly employed gates are the SWAP and CNOT:

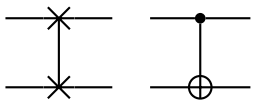


Figure 2.6: SWAP and CNOT gates

Finally, the measurement operator can be applied to a single qubit or to the whole system.

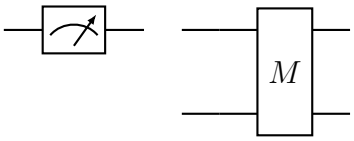


Figure 2.7: Measurement operators

it is important to notice that the measurements are always assumed in the logical Z bases. In order to make measurements on another base, an operator corresponding to the bases can be applied.

2.3 Quantum computer technologies

At the end of 1990, the first experiment showing the possibility of creating a Quantum Computer emerged. Since then, several platforms have been proposed and realized, each one using different technologies and providing tools and software for developing and researching on quantum computing.

2.3.1 DiVincenzo Criteria for quantum computing

At the beginning of the new millennium, the theoretical physicist David P. DiVincenzo formulated 5 requirements (plus 2 related to quantum communication) for the physical implementation of a Quantum Computer [10], that became famous as *DiVincenzo Criteria*. These criteria represent the starting point when discussing the physical realization of quantum computers since they are basic requirements to be met in order to achieve quantum advantage over classical computation. Before delving into some examples of current quantum computer technologies, an insight into the principles, and a short analysis of the meaning and the effects of every principle are conducted. In the following, the name is reported as in the original paper.

1. *A scalable physical system with well-characterized qubits*
   Qubits are the key component of a quantum computer, and of course, the analysis should start from the physical realization of the qubits. Scalable means that it is preferred that the possibility of adding more qubits to the system should be feasible, and does not have any sort of limitation. Qubits should also be *well-characterized*, this means that their internal Hamiltonian (in which the eigenstates corresponding to the states $|0\rangle$ and $|1\rangle$ are encoded) should be known, and the presence of coupling and interaction with other states should be known.

2. *The ability to initialize the state of the qubits to a simple fiducial state*
   This requirement is straightforward when thinking of most algorithms that need the state to be initialized to the state $|000\cdots0\rangle$. In addition, error-correction protocol requires a certain number of qubits in a low entropy state ($|0\rangle$).

3. *Long relevant decoherence times, much longer than the gate operation time*
   Decoherence is the phenomenon in which qubits interact with the environment, causing a quantum state $|\psi\rangle$ to "decay" into a mixture state. More importantly, since the classical behavior of nature arises from decoherence, decoherence times define the time in which quantum computer remains "quantum", and have inside advantages with respect to classical computation. How much this

time should be can be defined with respect to the gate time and the depth of the protocol, in order to finish the computation much before decoherence arises.

4. *A "universal" set of quantum gates*
   This requirement can be seen as the counterpart of the need for a set of logical gates in classical computation. it is obviously true, but there is some other consideration to do on this requirement. Generally, quantum algorithms come in the form of a sequence of unitary transformations, each one defined by an evolution of a quantum state following a Hamiltonian. One should define a Hamiltonian for every transformation. This is not feasible. However, has been demonstrated that any n-gates known can be implemented using a set of gates and a 2-qubit gate. This cannot stop the discussion, since in some physical systems (for example Neutral Atoms Quantum Computer) the interaction Hamiltonian cannot be turned off, so other ways of implementing gates should be found, case by case.

5. *A qubit-specific measurement capability*
   Since the output of a quantum computation should be read, a quantum computer should have measurement capability at the qubit level, and the measurement of a qubit should not interfere with other components of the machine or qubits.

2.3.2 Noisy Intermediate-Scale Quantum Computer Era

Today, the current quantum computers belong to the Noisy Intermediate-Scale Quantum Computer (NISQ) era. This term was introduced by John Preskill in 2018 [19], highlighting in his paper the current situation of quantum computing in different fields. The term *intermediate-scale* refers to the size of quantum computers, with a qubits number in the order of hundreds of qubits. With 100 qubits, we are theoretically capable of reaching *quantum advantage* over classical machines. However, machines have sources of noise in the quantum gates, that limit the size of the circuit or protocol that could be executed reliably.

Current quantum computers can be divided into mainly 5 categories, based on the underlying technology for building qubits [20]:

- **Superconducting Josephson junctions**
  Superconducting quantum computers are one of the most promising technologies for building quantum computers. it is based on the physical properties of the Josephson junctions (based on the properties of semiconductors), and his capability of emulating a 2-level quantum system. Superconducting quantum computers have around 100 qubits. Although being *universal*, the current

limitation consists of the amount of controlling hardware required for controlling and measuring the qubits. Among these categories belong the 2048 qubit Quantum annealer of D-Wave. This quantum computer is not universal but works as an optimizer for combinatorial problems. Current companies that build and conduct research on this technology are IBM, Google, D-Wave Systems, and Rigetti.

- **Ion trap**
  Consist of ions, trapped using electromagnetic fields using an electromagnetic field. Quantum information is encoded in the electronic state. Lasers can be used to control the qubits (through single qubit rotation) and induce entanglement. They are the first proposal for realizing large-scale quantum computers, with a proposal for implementing CNOT gate in 1995, and met all the requirements of DiVincenzo criteria. Although this, they are very difficult to implement, with a maximum number of qubits of 20, reached in 2018. Current companies that build and conduct research on this technology are IONQ and Quantinuum.

- **Photons**
  The paradigm of *Linear Optical Quantum Computer* (LOQC) consists of using the property of light to encode quantum state (polarization of light, angular momentum of photons, etc.). It has been demonstrated that Photonics Quantum Computing is universal and has the possibility of merging in a single framework quantum computation and quantum communication. However, the number of resources required to control light does not scale well with qubits number. Current companies that build and conduct research on this technology are PsiQuantum and Xanadu.

- **Neutral atoms**
  Neutral atoms are a prominent technology discussed in the last 20 years, but realized recently thanks to the recent advantages in the laser fields. It consists of neutral atoms encoding quantum information in the electronic state of the valence electron, whose energy state can be controlled by lasers. Currently, the quantum computer realized with this technology (that has 256 qubits) can be used only in analog mode, as a Hamiltonian simulator. Current companies that build and conduct research on this technology are QuEra, Pasqal, ColdQuanta, and Atom Computing.

- **Quantum dots**
  Also called *artificial atoms*, they are nanoscale miniaturized devices that exhibit quantum mechanics properties, used for realizing qubits. The quantum information is encoded by the spin of the electron, as proposed by Loss–DiVincenzo. Among the others (like IBM), Silicon Quantum Computer

has a goal of building a fault-tolerant quantum dots quantum computer by the end of 2028.

These machines, as said by Preskill, do not have the power to change society, but can be seen as a step towards reliable and fault-tolerant quantum computers.

2.3.3 Software Tools

Developing of software development kit (SDK) for experimenting and interacting with Quantum Computer has followed the hardware implementation effort. In the last year, libraries, tools, and programming languages have been proposed by companies for interacting with their hardware. Among these Braket with IBM has introduced the paradigm of Cloud Quantum computing, with the possibility of running quantum protocol on real machines by submitting a job.

- **Qiskit**
  it is a Python ecosystem of libraries developed and maintained with **IBM**, that allows circuit prototyping, emulating quantum hardware with the local simulator, introducing noise with different models, and interacting with their quantum device through their backend. it is a reference point for quantum developers.

- **Bloqade**
  Developed by Quera Computing Inc. in Julia, based on Yao, a library for simulating quantum mechanics. It allows emulating Aquila, their analog quantum computer, providing different utilities for building Hamiltonians and Observables. In addition, provides a library for interacting with real hardware. Moreover, it has recently released a Python version of the SDK.

- **Pulser**
  Is the analog of Bloqade, but with the Pasqal ecosystem. it is an open-source Python library and provides utilities for pulse definition and emulation that act on an array of neutral atoms.

- **Q#**
  Is a programming language part of the Quantum Development kit (QDK) of Azure Quantum.

- **Pennylane**
  Platform developed by Xanadu, but allowing access to a variety of quantum computing platforms, including IBM. it is written in Python and allows quantum simulation and in particular quantum machine learning.

- **Cirq**
  it is the Google SDK built in combination with *qsim*, written in C++, allowing quantum simulation of up to 40 qubits on a classical processor.

- **Rigetti Forest**
  Is the Python ecosystem of Rigetti, consisting of *PyQuill* (their library for quantum emulation), a compiler for their machine, and a quantum virtual machine (qvm) for simulation.

- **D-Wave Ocean**
  Is a suite of open-source Python ecosystems that allows access to the D-wave quantum annealer. It provides primitives for mapping problems in the format of the quantum solver.

2.4 Quantum Machine Learning

Quantum Machine Learning (QML) has emerged in the last few years as a field of study that combines Machine Learning and Quantum computing. it is a relatively new framework, being introduced in *2013* [21], thanks to the rapid advantages in both Quantum computing and machine learning. The possibility of a Quantum Computer processing a high quantity of data in high dimensional space fits well with the kind of data that usually ML analyzes In addition, quantum systems are able to produce different patterns than classical computers, exploring many more possibilities. In the last decade, there were several works working on QML, demonstrating the wideness of the area, which allows different interpretations. it is usual to divide QML into 4 main tasks, as the 4 quadrants of a space with 2 axes: one is the *algorithm type* (or in some work the device used for processing) axes, and the second one is the *data source* axes:

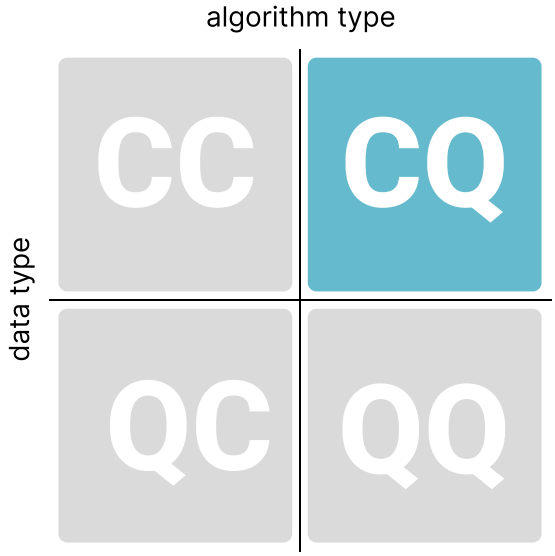


Figure 2.8: Taxonomy of Quantum Machine Learning. The objective of this work is highlighted in blue. It consists of a quantum algorithm that analyzes classical data.

1. *Classical* - *Classical* (CC) that refers to classical data processed with classical computers. In the context of QML is commonly referred to *quantum-inspired algorithm*, which are algorithms inspired by the linear algebra used in QC (like *tensor networks*) or attempting to emulate quantum systems on classical machines.

2. *Classical* - *Quantum* (CQ): Classical data processed using Quantum devices. it is the most mature branch of QML, since most sources of data are classical, and could provide the best advantages over classical algorithms. Examples of these methods are Quantum Support Vector Machines and Quantum Neural Networks.

3. *Quantum* - *Classical* (QC) tries to extract information from quantum data, using a classical algorithm. An example can be the computation of the gradient of quantum states, a necessary step in Variational Quantum Eigensolvers (VQE).

4. *Quantum* - *Quantum* (QQ), that is the quantum processing of quantum data, which can be defined as the purest approach. An example can be the quantum post-processing of the output of a quantum simulation. These can have several applications in molecule study.

In this thesis's work, a Classical - Quantum architecture will be analyzed. By

now, considering the situation of quantum devices and the data availability is one of the most prominent areas of QML.

2.5 Emulation against Simulation

At this point, should be clear that the main advantage of quantum computing is the possibility of executing operations with high parallelism thanks to the principle of superposition, and the possibility of creating entangled states, that create states that behave in a peculiar way, and the operation on a component modify also the other components. In this context is important to highlight the difference between **quantum emulation** and **quantum simulation**.

As the term suggests, emulation is a general term for grouping the software and hardware tools for executing code that is not built for the specific platform, usually older. In this context, quantum emulation is the execution of a quantum algorithm on a classical machine, **emulating** the operation. As described in the previous section, each qubit can be described with 2 complex numbers, and consequently, the space necessary for describing N qubit is 2^N complex numbers. This means that emulating space complexity grows exponentially, making it not feasible for real-world problems. In terms of operation, qubit operations can be of two types:

- *Digital*: the most common gates are single o 2-qubits, this means that applying an operator means making matrix multiplication with 2×2 or 4×4 matrices, that are not so computationally expensive.
- *Analog*: in this case, simulation is more computationally expensive, since it requires the solution of a system of differential equations, and computing the time evolution.

Emulation can be done both on a common laptop (with 10-20 qubits) or supercomputers (full emulation is possible with at most 55 qubits, while with compression techniques or tensor networks, this limit rises to 100 qubits).

Quantum simulators are a different thing. They are more similar to the first idea of quantum computers presented by Richard Feynman, as simulators of quantum systems, currently called *analog quantum computers*. They are real quantum system that realizes the evolution or operation directly, without solving any system. it is clear that simulators do not have the disadvantage of the exponential space required to store the state, and so can behave like real quantum computers.

Chapter 3

Quantum computing on Neutral atoms machine

In chapter 2, how the quantum computing paradigm emerges from the quantum mechanics principles is explained, discussing the peculiarities and opportunities of realizing a quantum computer. In the end, some examples of technologies that could enable quantum computer production have been shown, looking at the DiVincenzo criteria, the prerequisites for realizing the quantum advantage. In this chapter, Neutral atoms quantum computers, in particular, Aquila from QuEra computing are discussed, presenting applications where they can be employed, and finally showing the quantum evolution kernel and a possible implementation on a quantum computer.

3.1 Aquila: A 256's qubit quantum computer

Neutral atoms have been proposed as candidates for qubits implementation since the early 2000 [22]. In the following decades, some experiments have demonstrated the possibility of using arrays of Rydberg atoms as qubits. However, the enabling researchers were the ones who demonstrated how to load arrays of atoms in position using optical tweezers [23], showing the non-equilibrium dynamics of a chain of 51 atoms [24]. As a result, around this promising technology, several companies have been born, with the aim of building, maintaining, and developing algorithms for Neutral Atoms Quantum Computer. The most promising are Pasqal, ColdQuanta, Atom Computing, Planqc, M Squared, and **QuEra Computing**. Aquila [25], is a Neutral Atoms Quantum Computer built by **QuEra Computing Inc.**, available through the Braket cloud service on Amazon Web Services. Aquila is a room-temperature *Field Programmable Qubit Array* (FPQA), currently operating as a **user-defined** Hamiltonian simulator on up to 256 qubits. Aquila is composed of

several components, that allow the simulation of such Hamiltonian: neutral atoms as physical qubits, that allow entanglement through Rydberg interaction, driven and positioned in space by lasers. In the following, the different characteristics and components will be detailed. An overview of the machine, taken from the Aquila whitepaper [25], is present in figure 3.1.

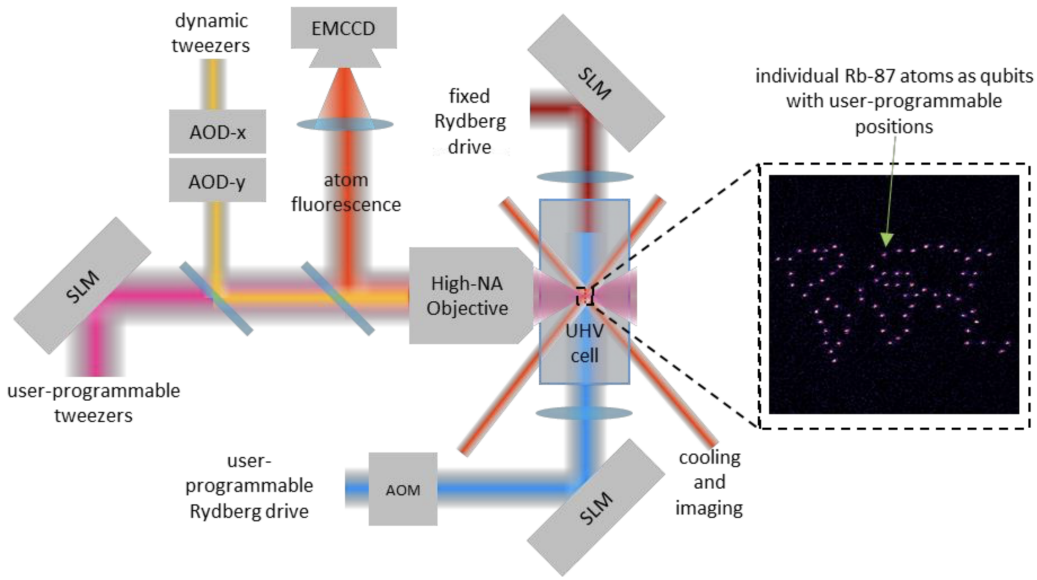


Figure 3.1: Aquila architecture, highlighting the different components used. Image taken from Aquila whitepaper from QuEra [25].

The qubits are physically implemented using neutral Rubidium-87 atoms, cooled, moved, and controlled using laser beams. The quantum information is encoded into the electron orbital of the valence atom. Different set of states are possible using Rb-87, with different characteristics:

- **Hyperfine qubit**: it is a long-lived qubit, with long coherence time ($\simeq 1\,s$) and no interactions between qubits and the environment. It will be the key component of the digital mode (or gate-based) and a hybrid analog-digital mode. It is characterized by two ground states, $|0\rangle = |g\rangle = |5S_{1/2}, F = 1\rangle$ and $|1\rangle = |g'\rangle = |5S_{1/2}, F = 2\rangle$, separated by an energy gap of $\simeq 6.8\,\text{GHz}$. Entangling operations will be implemented by passing to the Rydberg state ($|r\rangle = |70S_{1/2}\rangle$).

- **Ground-Rydberg qubit**: relatively short-lived qubit, but with a strong interaction between atoms in the Rydberg states, responsible for the entanglement implementation. it is the key component of the current main operating

mode of Aquila, the analog mode. it is characterized by a ground state, logically representing $|0\rangle = |g\rangle = |5S_{1/2}\rangle$, and the Rydberg state, represented by a highly excited S orbital, $|1\rangle = |r\rangle = |70S_{1/2}\rangle$. The Rydberg state, as explained later, generates a huge electric dipole, making it a key ingredient for entanglement.

Both qubits can be hosted on the same physical atom, making the platform versatile (for example, in the possibility of having a hybrid dual digital-analog mode). A schematic and simplified overview of the electronic states and their correspondence with the qubits state can be found in figure 3.2. Transitions between electronic orbitals are made by absorbing or emitting photons, according to Plank's theory. In order to trigger the transitions, the atom should be exposed to light with frequency proportional to the amount of energy between the two energy states, according to the *Plank* equation $E = h\nu$. This is one of the most technologically difficult challenges, realizing laser beams that are simultaneously *ultra-stable* and deliver *high powers*. Recent advantages have allowed the realization of these lasers, consisting of ultra-stable lasers locked to a cavity. In figure 3.2, together with the electronic states, the laser wavelength used for the transitions is shown.

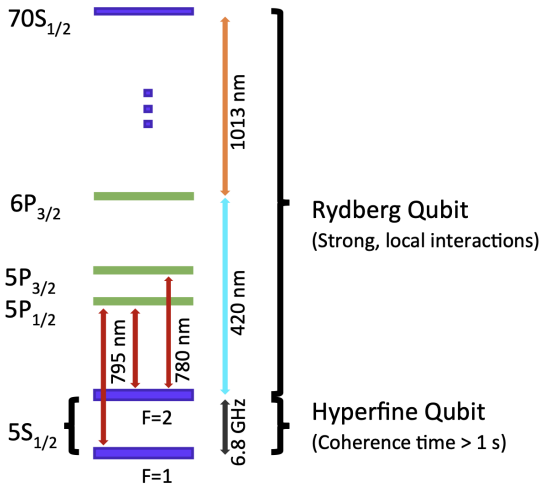


Figure 3.2: Electronic states diagram and qubits states. Image taken from [25].

Quantum computation is possible thanks to the precise control of the laser parameters. The **Rabi frequency Ω**, is directly related to the laser amplitude, which controls the rate of the transition between ground and Rydberg states. The parameter that represents how the laser is off-resonant with the atomic energy transition is called **detuning Δ**, while the value ϕ, called the **phase**, represent time offset of the laser. These 3 parameters can be controlled in time (i.e. the functional form $\Omega(t), \Delta(t)$ and $\phi(t)$) using optical components called *acousto-optical*

modulators (AOM), which generate sound waves, propagating into crystals that are crossed by light, modulating the laser intensity, detuning, and phase.

The possibility to define a waveform in time for each of the Rabi frequency $\Omega(t)$, the detuning $\Delta(t)$, the phase $\phi(t)$, and a set of atom positions $\{\vec{x}\}$ for each of the qubits, is one of the peculiarities of Aquila operating in analog mode. An example of an Aquila algorithm can be found in figure 3.3.

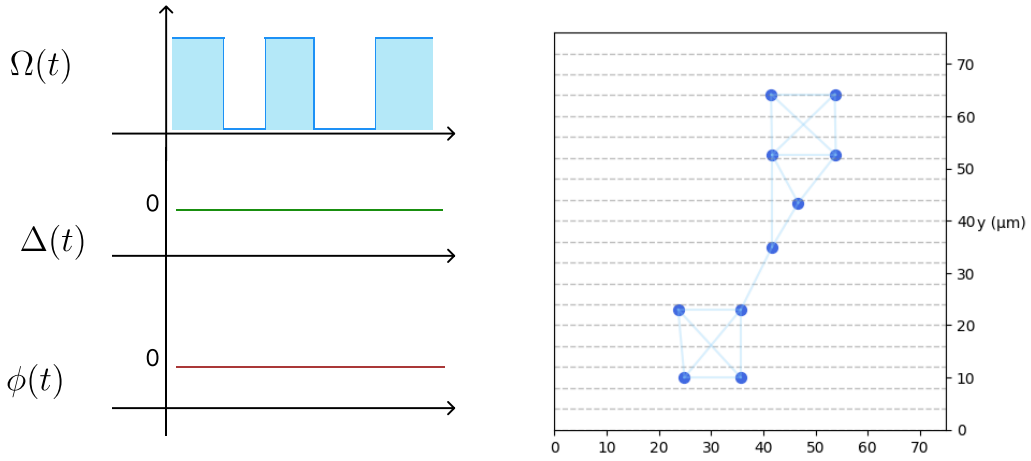


Figure 3.3: An example of an analog program that can be executed on Aquila, corresponding to the Quantum Evolution kernel detailed in the next section

Common quantum algorithms and programs allow users to specify a sequence of quantum gates applied in a certain order, called *digital mode*. Digital mode is universal but does not scale well, because of the noise connected to the depth of the circuits, while Analog mode is not universal, but is well-suited for machine learning or optimization tasks.

In the Aquila processor, atoms are moved and kept in position (or trapped) using traps created by focused laser beams generated by *optical tweezers*. These optical components use a laser to create optical dipole forces, that interact with the dipole induced in the atom by using a laser in resonance with one of the unused intermediate states (in this case $6P_{3/2}$), creating an area of high radiation pressure that traps the neutral atom. A second set of lasers optically cools the atoms by converting kinetic energy into photonic energy and initializing every atom into the ground state. Optical trapping is used in 2 modes in Aquila:

- **Spatial Light Modulator** (SLM), that uses the principle of the *holography* to create a phase mask that creates the tiny spot that serves as locations of trap for individual atoms.

- **Acousto-Optical Deflectors** (AOD), to dynamically move atoms, allowing position reconfigurability. This feature is a key component for future error-corrected qubits, and for a protocol that uses dynamical positioning of atoms. To accelerate this sorting process, atoms are required to be located in discrete rows, adding an extra constraint on atoms positioning. The rows should be $4\,\mu m$. In addition, due to the resolution of the SLM, atoms in the same row should be $4\,\mu m$ apart.

Both techniques and components are used by Aquila for preparing the neutral atoms array. Atoms are captured from a diluted vapor at room temperature and moved using AOD towards the traps. This process is not deterministic, and a trap can be filled with a probability of $\simeq 60\%$. For this reason, is important to post-process the result after each shot. Aquila helps with this task by providing pre-measurement and post-measurement, which allows filtering the shots with incorrectly filled traps.

Positioning the atoms is crucial for realizing entanglement, through the mechanism of the Rydberg blockade. Atoms in the Rydberg state, i.e. highly excited electronic orbitals state, enable a strong **Van Der Walls** interaction between atoms, conditional on the state. This interaction is typical of atoms in the Rydberg state and is both position-dependent and state-dependent and follows the following relation:

$$V_{i,j} = \frac{C_6}{|\vec{x}_i - \vec{x}_j|^6} \tag{3.1}$$

With $C_6 = 5{,}420{,}503 \frac{\mu m^6 rad}{\mu s}$. This interaction prevents two atoms from being both in the Rydberg state, enabling entangling dynamics. These phenomena take the names of **Rydberg Blockade**. Rydberg blockade phenomena allows also a very useful approximation, very used in emulation on a classical machine. Since the energy of the doubly excited state is much larger than the energy associated with the laser drive, it can be eliminated from dynamics, allowing a speed-up. A more precise analysis of the emulation time and associated performances is conducted in the next chapter.

The positions of the atoms play a crucial role in defining interaction (i.e. entanglement) between qubits. In the Aquila processor, atoms are moved and kept in position using traps created by focused laser beams generated by *optical tweezers*. The laser uses optical dipole forces, dipole induced in the atom by using a laser in resonance with one of the unused intermediate states (in this case $6P_{3/2}$), creating an area of high radiation pressure that traps the neutral atom. A second set of lasers optically cools the atoms by converting kinetic energy into photonic energy and initializing every atom into the ground state. Optical trapping is used in 2 modes in Aquila:

- **Spatial Light Modulator** (SLM), that uses the principle of the *holography*

to create a phase mask that creates the tiny spot that serves as locations of trap for individual atoms.

- **Acousto-Optical Deflectors** (AOD), to dynamically move atoms, allowing position reconfigurability. This feature is a key component for future error-corrected qubits.

Both trapping is used for loading the atoms at the selected positions. Atoms are captured from a diluted vapor at room temperature and moved using AOD towards the traps. This process is not deterministic, and a trap can be filled with a probability of $\simeq 60\%$. For this reason, is important to post-process the result after each shot. Aquila helps with this task by providing pre-measurement and post-measurement, which allows filtering the shots with incorrectly filled traps.

Measurements are restricted to the logical **Z** basis only. Rydberg measurements are implemented by re-trapping the atoms. During the quantum evolution, the optical tweezers are turned off to not interfere with the dynamics. After the evolution, traps are turned on, and the wavefunction collapses: the atoms in the ground states are re-trapped while the atoms in the Rydberg state are repelled. The state can be measured using fluorescence, through the absence (0, so atom in the Rydberg state) or presence of an atom (1, atom in ground state).

it is important to notice that measures are **destructive**, because every time an atom is measured in the Rydberg state, this is pushed out of the array, and a hypothetical next experiment needs to rebuild the atom configuration.

Finally, the full dynamics of the machine in the analog mode can be summarized by the **Rydberg Hamiltonian**, incorporating laser beams term, Rydberg atoms Hamiltonian, and the Rydberg interaction:

$$\frac{\mathcal{H}(t)}{\hbar} = \sum_i \left(\frac{\Omega(t)}{2} \left(e^{i\phi(t)} |0\rangle \langle 1| + e^{-i\phi(t)} |1\rangle \langle 0| \right) \right) - \Delta(t) \sum_i \hat{n}_i + \sum_{j<k} V_{jk} \hat{n}_j \hat{n}_k \quad (3.2)$$

Where:

- V_{ij} is the interaction strength where both atoms are in the Rydberg state, as defined in 3.1,
- $\Delta(t)$ Detuning. This sets how off-resonant the global Rabi drive is.
- $\Omega(t)$ The Rabi drive amplitude. This sets the frequency at which each qubit oscillates between ground and Rydberg state in the absence of interactions.
- $\phi(t)$ The phase of the Rabi drive. This sets the direction on the Bloch sphere around which the qubit is driven.
- $\hat{n}_i$ is the *Rydberg occupancy operator*, defined by $\hat{n}_i = |r_i\rangle \langle r_i|$, count the number of Rydberg occupation.

The quantum state evolves following the time-dependent unitary that follows directly from the Rydberg Hamiltonian:

$$|\psi\rangle = \mathcal{T} \exp \int_0^T H(t)dt \, |0\rangle \tag{3.3}$$

Finally, a full cycle, consisting of one shot, can be shown in figure 3.4.

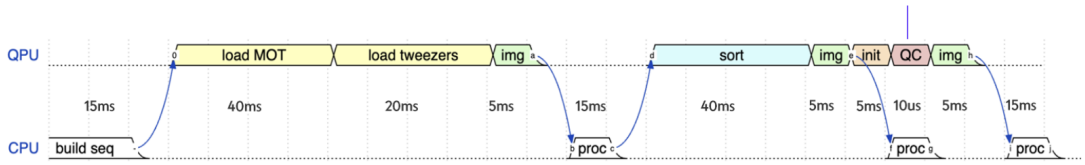


Figure 3.4: A single shot from Aquila, from building the register, to measurement [25]

All the previously discussed components are put together in a precise way in order to execute the quantum computation.

1. The optical trap is loaded, and the occupancy of the randomly filled trap is imaged and processed, producing a list of occupied traps, called *pre-sequence array*. These steps are necessary to prevent the spurious presence of atoms from previous computations.

2. The tweezers load the atoms (in the ground state) and another image is taken to assess the successful execution of the sorting process.

3. Finally the quantum computation is executed through the loaded protocol.

4. The traps are turned back on as described before, and another image is taken, producing the *post-sequence* array, used to make the bitstring measured in the Rydberg basis.

5. At the end, atoms are released and the cycle can be repeated.

Despite the short time for the quantum computation is relatively short (maximum $\simeq 10\,\mu s$, the full process takes much time. This makes the average Aquila shot rate of about 10 computations per second.

3.1.1 Noise

Analog protocols are quantum programs executed by means of a simulation of a Hamiltonian on a quantum system. When the execution time increase, various

sources of noise shows up, reducing the fidelity of the state and consequently of the measurements. In the Aquila whitepaper [25], the various sources of noise are discussed, and the associated limitations of the Aquila processor are shown:

Laser noise Lasers, while being ultra-stable, still have some noise in their phase and amplitude. This causes a difference from shot-to-shot in the Rabi frequency Ω ($0.008 rad/\mu s$) and detuning Δ).

Atom motion Atoms are positioned by optical tweezers and then cooled by lasers, to temperature in the order of μK. This implies always a certain amount of thermal motion in the traps during the evolution ($\sigma_x = \sigma_y = 0.2\,\mu m$). The 2 major effects of these sources of noise are:

- $0.18 rad/\mu s$ variance in the detuning, due to the Doppler effect.
- Differences from shot to shot of the interaction strength constant V_{jk}

State decoherence and scattering A common source of noise consists of the incoherent decay of the atoms in the Rydberg states $|r\rangle$ towards the ground state $|g\rangle$. The coherence time is measured in the case of a single qubit driven by the maximum Rabi frequency of $7.5\,\mu s$, and of $8.9\,\mu s$ in the case of interacting qubits.

Inhomogeneity The little variations in the components responsible for the holographic process cause differences in the rabi drive and the detuning across the 2D array, causing different operations across the qubits.

Measurement The measurement process is implemented through retrapping the atoms by turning on the optical tweezers. This process can cause the incorrect retrapping of atoms, causing a readout error, that can be of both types (reading a $|0\rangle$ as $|1\rangle$ with probability 1% and reading $|1\rangle$ as $|0\rangle$ with probability 8%).

As suggested in the whitepaper, in order to reduce the majority of noise sources, it can be useful to minimize the total protocol time, corresponding to the length of the pulse. This can be done by choosing the maximum Rabi drive Ω, in order to minimize the time to implement the same rotation.

3.2 Application of neutral atoms

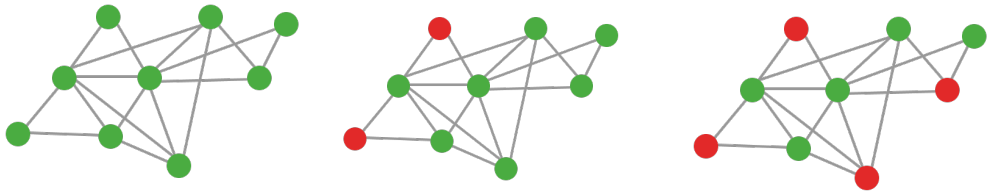
In this section, near-term applications of neutral atoms quantum computers are presented, before delving into the details of the quantum kernel. The particular nearest-neighbor interaction of atoms in the Rydberg state allows embedding

different graph problems into the structure of the Hilbert space. In particular, some classes of graphs (*Unit Disk graph*) can be directly embedded into the register with nodes represented by atoms and edges represented by the Rydberg interaction.

3.2.1 Maximum Independent Set

Indipendet Set Consider an undirect graph $\mathcal{G} = (V, E)$, where V is the vertices set and E is the edge set. A subset of the vertices set $I \subseteq V$ is an *independent set* if for every pair of vertices $v_i, v_j \in I,\ i \neq j$ there does not exist an edge (i.e. they are not adjacent) $(i, j) \notin E$.

The size of an independent set is $n = |I|$ (i.e. the number of nodes in I). An independent set is said to be *maximal* if there does not exist another independent set I' such that $|I'| > |I|$ and I' including I. The problem of finding I, the largest of the maximal independent sets, is called *Maximum Independent Set* (MIS) and is NP-Hard. Another variant of the problem, where each vertex has a weight w_i, consists of finding the independent set with the maximum sum of weights is called the *Maximum Weighted Independent Set* (MWIS) problem. Examples of graphs, and independent sets are shown in figure 3.5.



(a) a generic Graph $\mathcal{G}$ **(b)** An independent set **(c)** A maximum IS

Figure 3.5: Difference between an independent set and a maximum independent set

Independent sets have a great interest since they belong to NP-C, so every problem in NP can be polynomially reduced to an instance of MIS. In addition, several real-world problems can be mapped to MIS, like *Antenna placement* that guarantees a certain coverage [26]. This problem in particular consists of finding a MWIS of a Unit Disk Graph.

MIS problem can be efficiently solved on a Neutral Atom QPU, with a superlinear quantum speedup [26] [27]. The most natural approach consists of mapping the solution of the MIS to the ground state of the Rydberg Hamiltonian. In particular, the Rydberg constraint is particularly well-suited to implement the independence

constraint. When setting the Rabi frequency to 0 ($\Omega = 0$), and the detuning to $\Delta > 0$ (where the detuning is set according to the unit disk radius), the low energy state prefers to have the maximum number of atoms in the Rydberg state. However, there is an energy penalty in Hamiltonian, based on distance. In particular, if two atoms are within a certain radius (called the Blockade Radius R_b), the energy of the double excited state is larger than a single atom in Rydberg states, incorporating the independence constraint on adjacent atoms. In particular:

$$E = \begin{cases} -2\Delta + C_6/R_b^6 & \text{2 near atom} \\ -\Delta & \text{single atom} \end{cases} \tag{3.4}$$

The ground state of the Hamiltonian defined above is closely related to the solution of the MIS.

3.3 Quantum evolution kernel

In the previous section, a possible application of analog quantum computing for solving graph problems has been introduced. The neutral atoms platform suits well with graph data, thanks to its ability to embed the graph topology into the system Hamiltonian. The quantum evolution of a state, based on a proper Hamiltonian that reflects the graph topology can be used as a feature extractor for graph data, without relying on complex data preprocessing. The key to the method is defining a proper probability distribution that reflects the graph characteristics. Later, the kernel function can be naturally defined as a function of the distance between these two graphs' probability distributions and used for downstream tasks such as *graph classification* (using a support vector machine) and *regression* (using ridge regression). The time evolution of a quantum state is associated with the graph $\mathcal{G}$, since the topology of the graph can be directly integrated into the system Hamiltonian (in particular, by the atom's position in the register).

Let's consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes of $\mathcal{G}$, while $\mathcal{E}$ is the set of edges. We refer with $N = |\mathcal{V}|$ (i.e. the number of nodes of $\mathcal{G}$) and $E = |\mathcal{E}|$ (i.e. the number of nodes of $\mathcal{G}$). We can define a quantum state $|\psi_0\rangle$ of N qubits, representing $\mathcal{G}$, governed by a Hamiltonian $\hat{\mathcal{H}}_\mathcal{G}$ that has the same topology of the interaction of $\mathcal{G}$ (i.e. of its edges set). Different choices of $\hat{\mathcal{H}}_\mathcal{G}$ could be possible:

- **Ising Hamiltonian**: represent a model of spin, with nearest neighbor interaction.

$$\hat{\mathcal{H}}_I = \sum_{(i,j)\in\mathcal{E}} \hat{\sigma}_i^z \hat{\sigma}_j^z$$

  Where $\hat{\sigma}_i^z$ are the Pauli Z operators.

- **XY Hamiltonian**: represent a model of spin, with anisotropic interactions.

$$\hat{\mathcal{H}}_{XY} = \sum_{(i,j)\in\mathcal{E}} \hat{\sigma}_i^+ \hat{\sigma}_j^- + h.c.$$

  Where $\hat{\sigma}_i^+$ and $\hat{\sigma}_j^-$ are the *rising* and *lowering* operators.

Both Hamiltonians are chosen for mainly 2 reasons: they reflect the topology of the graph through the interaction terms, and they are implementable into current Neutral Atoms platforms [28]. Other Hamiltonians that reflect other graph characteristics could be considered, and employed in quantum-inspired classical algorithms, as mentioned in [29]. From now on, when referring to $\hat{\mathcal{H}}_{\mathcal{G}}$ it is always the Ising Hamiltonian since it is easier to implement in hardware and faster to emulate on a classical machine. Another Hamiltonian $\hat{\mathcal{H}}_\theta$ (parameterized by a set of parameters θ, referred to as *mixing Hamiltonian*) is introduced to apply pulses to the system, letting the system evolve with a duration τ[1].

$$\hat{\mathcal{H}}_\theta = \sum_i^N \theta_i \sigma_x^i \tag{3.5}$$

An outline of the whole time evolution can be represented by a set of parameters:

$$\Lambda = \{\theta_0, t_1, \theta_1, \ldots, t_p, \theta_p\}$$

where θ_i represents the driving Hamiltonian, and t_i represents the free evolutions under the graph Hamiltonian. it is important to notice that since the graph Hamiltonian $\hat{\mathcal{H}}_{\mathcal{G}}$ is intrinsic in the register topology, cannot be turned off during the mixing Hamiltonian pulse, but his effect is negligible compared to $\hat{\mathcal{H}}_\theta$. The number p represents the number of layers or the number of alternating pulses. A schematic description of the layered approach can be found in image 3.6

[1]As explained in [29], we can consider that $\tau = 1$ and including in the definition of $\theta.\hat{\mathcal{H}}_\theta$ is applied in the same way to the whole qubits, representing a rotation around the x Axis of an angle θ in the Bloch sphere.

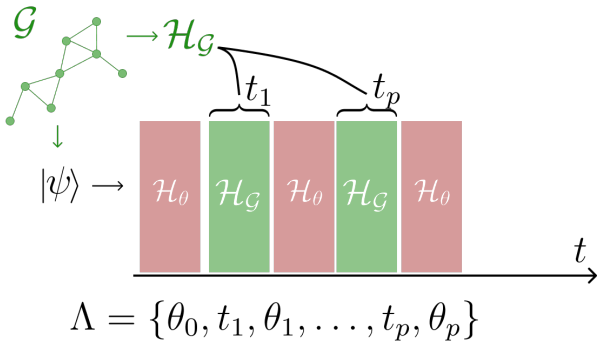


Figure 3.6: Layered time evolution, highlighting the order of application of the Hamiltonian, and how the parameter Λ are reflected into the Hamiltonian definition.

Following the protocol, the system reaches the final state

$$|\psi_f\rangle = \prod_{i=1}^{p} \left(e^{-i\hat{\mathcal{H}}_{\theta_i}} e^{-i\hat{\mathcal{H}}_\mathcal{G}}\right) e^{-i\hat{\mathcal{H}}_{\theta_0}} |\psi_0\rangle$$

Measuring a proper observable $\hat{\mathcal{O}}$ on the final state can be used to construct a probability distribution depending on the graph $\mathcal{G}$. Several approaches can be used and were analyzed in the literature [29], but one is particularly suited for its ability to well estimate a probability distribution reducing the number of the required measurements. The approach consists in sampling the final state $|\psi_f\rangle$, obtaining a series of M measurements $\{m_1, m_2, \ldots m_M\}$ of the observable $\hat{\mathcal{O}}$. The measurements correspond to one of the k eigenvalues of the observable. We can construct a probability distribution out of them:

$$\mathcal{P}_\mathcal{G}^{\hat{\mathcal{O}}}(\Lambda) = (p_1, \ldots, p_k), \;\; p_i = \langle o_i|\psi_f\rangle$$

In practice, in particular when k is too large, estimating the probability distribution will require an exponential number of measurements. For this reason, the proposed approach will be based on binning the computed energies. The energy distribution computation is one of the most crucial operations, and further consideration of the method will be conducted in the next chapter. The probability distribution allows to define a graph kernel by means of distances between distributions. Following the approach presented in [29], we select the Jensen-Shannon [30] divergence. Given two probability distribution $\mathcal{P}$ and $\mathcal{P}'$, the Jensen-Shannon divergence is defined as:

$$JS(\mathcal{P}, \mathcal{P}') = H\left(\frac{\mathcal{P} + \mathcal{P}'}{2}\right) - \frac{H(\mathcal{P}) + H(\mathcal{P}')}{2}$$

where $H(\cdot)$ is the Shannon entropy of a probability distribution, defined as

$$H(\mathcal{P}) = -\sum_k p_k \log p_k$$

The image of $JS(\cdot,\cdot)$ is the closed set $[0, \log 2]$, and reach the maximum value $(\log 2)$ when the distribution have disjoint support. Finally, the graph kernel function of two graphs, $\mathcal{G}$, $\mathcal{G}'$ and their associated probability distributions $\mathcal{P}$ and $\mathcal{G}'$ is defined as:

$$\mathcal{K}_\mu(\mathcal{G}, \mathcal{G}') = e^{-\mu\, JS(\mathcal{P}, \mathcal{P}')} \tag{3.6}$$

with image in $[2^{-\mu}, 1]$. This hyperparameter can be optimized with the pulse parameter Λ, allowing the kernel to take value in a wider range. Following [29], μ is set equal to 1.

3.3.1 Implementation on neutral atoms hardware

In the first section, *Aquila* [25] a Neutral Atoms Quantum Computer, working principally in the analog mode, is presented. An application showing his potential to characterize and solve a common NP-H was presented (the maximum independent set), exhibiting the possibility of the platform embedding in the system Hamiltonian (and in particular in the ground state) the graph problem solution. In the third section, a quantum evolution kernel for graph data is presented and derived. In this section, a possible implementation of the kernel (in particular how to implement to 2 Hamiltonian) will be showcased. First, let's recall the graph Hamiltonian, corresponding to an Ising model $\hat{\mathcal{H}}_\mathcal{G}$:

$$\hat{\mathcal{H}}_\mathcal{G} = \hat{\mathcal{H}}_I = \sum_{(i,j)\in\mathcal{E}} \hat{\sigma}_i^z \hat{\sigma}_j^z \tag{3.7}$$

As demonstrated in [28], arrays of Rydberg atoms can be used to simulate the Ising Hamiltonian. In particular, the interaction term derived from the strong Van-Der-Walls interaction of two atoms in the excited Rydberg states (corresponding to the state $|r\rangle = |1\rangle$) is equivalent to the Ising interaction, up to a constant term $V_{i,j}$ that depends on the position of the atoms. This term is negligible when the atoms are way apart, with respect to the other term in the Hamiltonian. The Ising model, and the corresponding graph Hamiltonian can be equivalently expressed as:

$$\mathcal{H}_\mathcal{G} = \sum_{i<j} V_{i,j} n_i, n_j \tag{3.8}$$

Where $V_{i,j}$ is the Rydberg Blockade interaction strength, and n_i is the occupancy operator. The other fundamental component is the *mixing Hamiltonian* $\hat{\mathcal{H}}_\theta$, representing a rotation around the X axis of an angle θ applied to all qubits. The

rotations can be implemented using the rotation induced by the laser beam, using a single constant Rabi drive:

$$\hat{\mathcal{H}}_\theta = \hat{\mathcal{H}}_\mathcal{G} + \frac{\Omega_0}{2} \sum_{i \in \mathcal{V}} \hat{\sigma}_i^x$$

it is important to notice that the graph Hamiltonian cannot be turned off, since it is intrinsic to the system and related to the atom positions. However, his effect is much lower than the mixing Hamiltonian, making it negligible. Starting from the full Hamiltonian defined in 3.2 it is possible to obtain the mixing Hamiltonian using a driving laser with Rabi frequency $\Omega(t) = \Omega_0$, phase $\phi(t) = 0$ and detuning $\Delta(t) = 0$ (here considered for a single atom):

$$\begin{aligned} \frac{\mathcal{H}}{\hbar} = \frac{\Omega_0}{2} \left(e^{i\phi(t)} \left|0\right\rangle \left\langle 1\right| + e^{-i\phi(t)} \left|1\right\rangle \left\langle 0\right| \right) = \\ \frac{\Omega_0}{2} \left(\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \right) = \\ = \frac{\Omega_0}{2} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \frac{\Omega_0}{2} \sigma_x \end{aligned}$$

Obtained by expanding the "ket" "bra" product and remembering the matrix form of the Pauli X operator (as in eq. 2.7). The expression found is in the case of a single qubit. Aquila allows each atom can be driven by a different laser field (through the holography principle), In this setting, however, every atom is driven by the same laser waveform.

Finally, the whole layered time evolution can be packed up in a unique time-dependent Hamiltonian $\mathcal{H}(t)$ that depends both on the graph $\mathcal{G}$ (atoms positions and interaction term) and by the selected waveform parameter Λ. The power of the Aquila analog mode is the high customizability of the laser parameter, allowing time-dependent protocols also for the parameter Ω, Δ and ϕ, that reflect into the Hamiltonian dynamics. An example of the waveform that realizes the layered evolution can be found in figure 3.7.

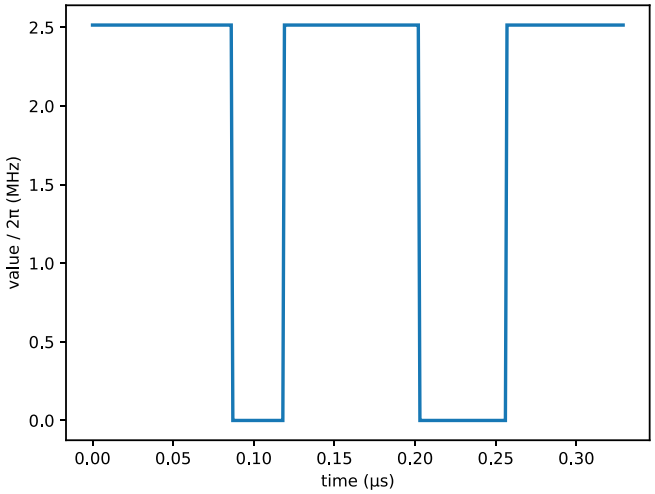


Figure 3.7: $\Omega(t)$ waveform that simulate the Quantum Evolution Kernel layered evolution

The amplitude of the Rabi frequency is set to the maximum value reachable by the QPU ($\Omega_0 = 15.8 rad/\mu s$, in order to have a shorter protocol, that is less sensitive to noise). In this specific figure, a 2 layer approach is implemented. The shorter pulse corresponds to the Rabi frequency set to 0, i.e *free evolution*, while in the other 3 pulses correspond to the *mixing Hamiltonian*. A more extensive analysis on the waveform parameters, that are optimized through Bayesian optimization, is done in Chapter 4

Chapter 4

Experimental results

While in the previous chapter, the proposed quantum evolution kernel is proposed and discussed, in this chapter its effectiveness is proven through experiments involving both emulation on a classical machine and simulation on a real quantum computer. The intrinsic difficulties of working with graph data are shown, in particular the preprocessing procedure involved in mapping to a quantum register. Later, the main discussion topic becomes hyperparameter tuning, in a double fashion: support vector machine parameter and quantum parameter, in the form of the waveform parameter and the necessary number of measurements. Several operative choices were taken during the work and are discussed in the following. In the end, the results are discussed, comparing the method with a classical kernel.

4.1 Dataset

PROTEINS [31, 32] is one of the most used datasets for benchmarking graph machine learning algorithms. Protein data are the perfect candidate to be modeled as graphs since they are macromolecules consisting of *amino acids* chain, disposed in a 3-dimensional space. Starting from a protein, a graph $\mathcal{G} = (\mathcal{V}, E)$ is obtained by modeling each amino acid as a node $v \in \mathcal{V}$, and an edge between amino acids if they are less than 6 angstroms apart in space. Each protein comes with a binary label that describes whether it is an *enzyme* or not. Working with graph data can be difficult, not only because of the intrinsic difficulty of the data but also because of the absence of homogeneous data representation. Some researchers [33] tried to overcome this problem by collecting and uniforming graph data representation. It comes in the form of different `.txt` files, each describing graphs, nodes, edges, and labels. The dataset consists of 1113 graphs, divided into 663 enzymes and 450 non-enzymes. Some overall statistics on the dataset are shown below in table 4.1:

Overall graph statistics	
min number of nodes	4
avg number of nodes	39.5
max number of nodes	620
min number of edges	5
avg number of edges	72.81
max number of edges	1049

Table 4.1: Overall statistics of the PROTEINS dataset

In order to fit the graph on a quantum register, it should be limited to a maximum of 256 nodes (i.e. the maximum number of physical qubits available on Aquila). By plotting node number distribution, one can notice that most of the graphs contain less than 100 nodes. Details on the distribution of the nodes and edges are shown in figure 4.1.

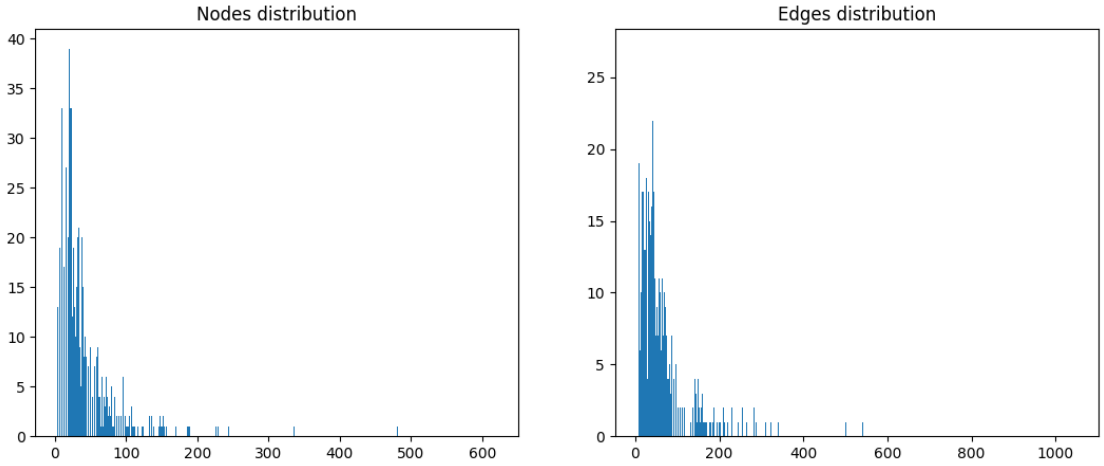


Figure 4.1: Nodes and edges distribution in PROTEINS dataset

Since emulation time on classical machine grows exponentially with the number of qubits (i.e. the graph nodes) the dataset has been reduced, producing 2 different but overlapping datasets:

- `PROTEINS256`: limited to graphs with a maximum of 256 nodes (Aquila current limit, corresponding to the number of physical qubits). It consists of 276 graphs[1].

[1]For both datasets, it is important to notice that are limited also by the constraint of being embeddable as unit Disk graphs, as detailed later.

- `PROTEINS12`: limited to graphs with a maximum of 12 nodes. The number of nodes is selected based on a qualitative benchmarking of emulation time. It consists of 143 graphs.

4.1.1 Graph preprocessing

As described in the previous chapter, computing the probability distribution associated with each graph requires measurements from a quantum state that evolves following a time-dependent Hamiltonian (i.e. the alternation of mixing Hamiltonian $\mathcal{H}_\theta$ and Ising Hamiltonian $\mathcal{H}_\mathcal{G}$) that depends on the graph topology. Starting from a graph, different methods can be implemented:

1. Simulating the Hamiltonian without explicitly mapping to a register composed of atoms, in this way positions don't require to be approximated or computed. The Hamiltonian is built directly from the definition and emulated using classical machines. In the current settings, this method is not applicable for execution on real hardware.

2. Pasqal researchers used a dataset with node position such that the graphs were local [29]. A simple rescaling method such that the minimum edge distance is equal to $5\mu m$ is applied.

3. Find the unit disk representation of the graph. In this way, the topology of the interaction of neutral atoms machine directly corresponds to the Ising Hamiltonian. This method is the only one that is applicable to real quantum hardware and does not provide an approximation of the Hamiltonian.

Since the final objective of this work is to assess the quality of the methods on real hardware, the third method is the only one that guarantees the possibility of execution on a quantum computer. In the following, details about the preprocessing step, involving the transformation of a graph into his UD representation, are discussed.

4.1.2 Unit Disk Graph

Consider a set of n equal-sized circles in the plane. The *intersection graph* of these circles is an n-vertex graph; each vertex corresponds to a circle and an edge appears between two vertices when the corresponding circles intersect (the tangent circles are assumed to intersect). Such intersection graphs are called unit disk graphs, and the set of n circles is an intersection model [34]. An example of circles in planes associated with a unit disk graph is shown in figure 4.2.

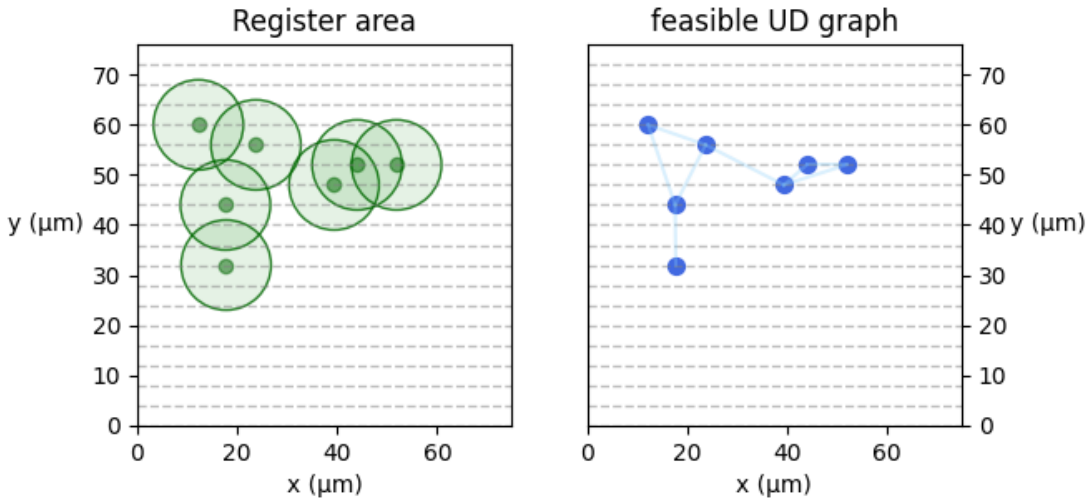


Figure 4.2: Circles in the register area and corresponding unit disk graph. In this case, the unit disk radius $R_{UD} = 9\mu m$

The problem that starts from a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and finds $\mathbf{x} \in \mathbb{R}^{n,m}$ (where $n = |\mathcal{V}|$ and m is the dimension of the selected euclidean space, usually 2 or 3), such that the Unit Disk graph $\mathcal{U} = (\mathcal{V}', \mathcal{E}')$ defined by unit circle has the same topology (i.e the same edge set $\mathcal{E}$ of $\mathcal{G}$), is non-convex and NP-Hard [35]. Several works explored solutions to this problem since some classical graph problems have a simpler solution when UD graphs are considered. In addition, UD graphs have grown in interest thanks to some applications in solving QUBO problems, and for their application in antenna placement. In addition, we would like to find a feasible solution that respects also the domain-specific constraint of neutral atoms quantum computers (in the particular case of Aquila, the additional constraints are related to atoms placement and maximum register area, both related to optical tweezers limitations). To solve the problem of *Constrained Unit Disk Graph* (CUDG), the neural-enhanced framework developed by LINKS has been used [36]. The two main components of the framework are:

- *Distance Encoder Network (DEN)*: the network that maps a set of initial coordinates into a new set of coordinates of the unit disk graph

- *Embedding Loss Function (ELF)*: a custom loss function that starting from the pairwise distances computes a loss function that incorporates different constraints.

Distance Encoder Network

The DEN model is instantiated and trained independently for each graph, as its architecture depends on the number of nodes of the graph. it is composed of 2 blocks:

- A *trainable autoencoder*, wich takes as an input a set of coordinate $\mathbf{x}_i \in \mathbb{R}^{n,m}$ and compute the set of unit disk graph coordinates $\mathbf{x}_f$. This part of the network, as all autoencoders, is composed of two parts: an encoder that starting from an input of size $n \times m$ maps to a latent vector of size 9, and the *decoder networks, that reconstructs* the coordinates starting from the latent vector. Both the encoder and the decoder are constituted by several fully connected networks, with reLU activation functions, and with dropouts. The last layer of the decoder consists of *tanh* activations.
- A *fixed-weight distance calculator*, that starting from the output of the *trainable autoencoder* computes the pairwise distances between each pair of nodes. It is composed of the first fully connected layer that calculates the differences, followed by a square activation function, and a fully connected layer that computes the sum. The output of the last layer is the $\binom{n}{2}$ pairwise square distances $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$, between every vertex i and vertex j. The distance layer is then the output of the loss function.

An example of the detailed architecture is presented in figure 4.3:

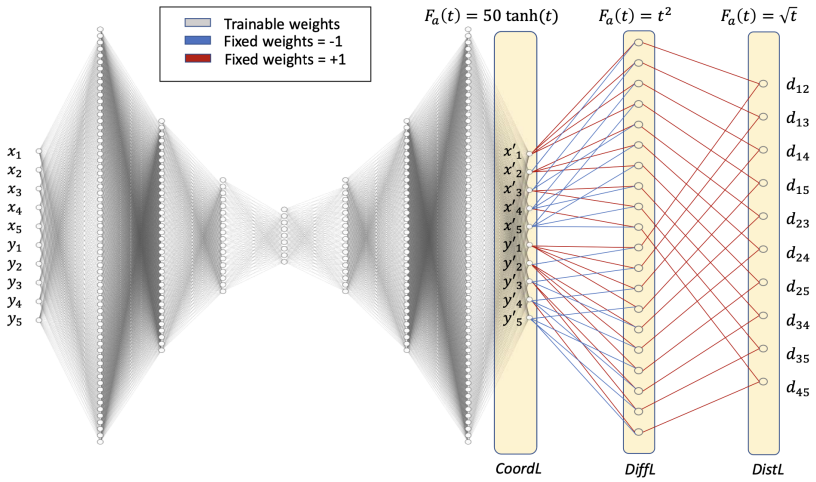


Figure 4.3: DEN model. Image taken from [36]
. The employed version of the model does not have the last square root layer, since the experiment has proven that this improves convergence.

Embedding Loss function

The embedding loss function is made of two components:

$$ELF(d) = ELF_{min}(d) + ELF_{max}(d)$$

Each component handles differently two sets of constraints:

- The $ELF_{min}(\cdot)$ define a lower bound on feasible distances, handling the $\geq$-based constraint
- The $ELF_{max}(\cdot)$ define an upper bound on feasible distances, handling the $\leq$-based constraint

Both components use the *Margin Ranking* loss function. The structure of the loss function allows one to easily define and incorporate additional constraints (both in the ELF-min or ELF-max loss components) proving the extreme flexibility of this framework.

4.1.3 Constrained Unit Disk Graph Problem

The neural-enhanced framework mentioned above allows further generalization, thanks to its flexibility. Before showing the architecture and the contribution, it can be useful to list the additional constraints for the CUDG problem, specific to the selected platform:

- *Register area constraint*: The atoms' position should belong to a rectangle of size $75\mu m \times 76\mu m$.
- *Row spacing constraint*: the atoms should be positioned in discrete rows, with $4\mu m$ spacing in between.

The last constraint is specific to Aquila, it has no correspondence with other Neutral Atom platforms and is related to how the Aquila scheduler decides how optical tweezers move the atoms. This implies that atoms' positions should be in discrete rows. The first constraint can be easily inserted in the ELF loss (implementing by modifying the multiplier of the *tanh* activation function), while the second constraint is implemented through an approximation layer that adjusts the position in discrete rows and by adding a new component to the loss function, that penalizes pairs that are closer than 4 μm by row without belonging to the same row. The UD constraints are later re-verified, in order to produce a feasible embedding.

4.1.4 Embedded dataset discussion

Once the methods and the reason are presented, the final datasets of the embedded graphs are presented. As described above, the datasets are produced by first filtering on the number of nodes and a second operation employing DEN, which reduces the number of graphs by filtering out the graphs that cannot have a feasible UD embedding. The characteristics of the generated dataset are presented in 4.2. In addition, an example of a starting graph and the generated feasible embedding are presented in figure 4.4.

	Number of graphs	#Class 1	#class 2
PROTEINS12	143	33	110
PROTEINS256	276	108	168

Table 4.2: Final dataset composition

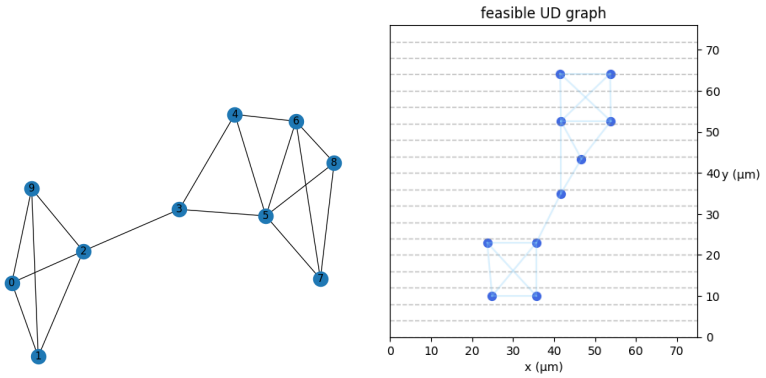


Figure 4.4: Graph and associated Unit Disk embedding. In the right picture, the additional discrete row constraint of Aquila is more easily noticeable.

Unfortunately, the DEN model does not guarantee convergence, since it is an approximation method employing neural networks. Not all graphs admit a Unit Disk representation, and this problem is more evident when the position space is limited to $\mathbb{R}^2$. However, the number of embedded graphs is sufficient to make a preliminary analysis of the quantum evolution kernel benchmark. It is important to note that the reduced dataset (i.e. `PROTEINS12`) presents a high-class imbalance. In the following sections, some methods to overcome this problem are analyzed. Even if `PROTEINS256` does not present the same problem (class 1, representing the 39% of the total dataset, while in `PROTEINS12` only the 23%), the methods are used anyway.

4.2 Emulation on classical Hardware

The emulation is performed using *Bloqade* [37], a framework written in Julia [38] developed by **QuEra computing** for experimenting and interacting with their Neutral Atoms quantum computer. Through its ecosystem, it supports emulating quantum systems, measuring different observables (even user-defined), and interacting with Neutral Atoms hardware by validating and creating a representation for the Hamiltonian that can be sent to Aquila Quantum Computer and run. In particular, the used libraries are:

- *BloqadeODE*: contains all the code for defining and emulating the time evolution of a quantum system by solving the Schrodinger equation.
- *BloqadeSchema*: contains all the functions for validating the Hamiltonian, in particular for checking the feasibility of the atoms' position, and contains the function for smoothing the waveform in the case of piecewise protocol, according to the bandwidth of the lasers.

All the code of this work is built on these two components. As described before, the emulation step is very expensive, but still necessary, for mainly two reasons:

1. Confirm the validity of the method before wasting precious quantum resources.
2. Each iteration of the Bayesian Optimization algorithm requires the $N \cdot M$ (where N is the number of graphs and M is the number of measurements) to run on the quantum computer, which is too expensive at the current time. This step is required to find the waveform parameters, i.e. the duration of the Omega pulse.

Before delving into the details of each part of the experiment, the full *hybrid quantum-classical* training procedure is detailed, summarized in the following figure 4.5:

1. **Quantum algorithm**: For every graph sample $\mathcal{G}_i$ and its position, a quantum register is instantiated and let to evolve following the Hamiltonian $\mathcal{H}_\Lambda$ defined by the parameters Λ. A set of measurements $M_{\mathcal{G}_i}$ is then obtained.
2. **Classical post-processing**: From each measurement M_i corresponding to a graph, a probability distribution is obtained by computing the corresponding interaction energy. The output of this step is a set of probability distribution $\mathcal{P}_{\mathcal{G}_i}$ for each graph.
3. **Machine Learning algorithm**: The probability distribution of each graph is used to compute a kernel function (as described in 3.6) between each

graph, used to train the support vector machine and validates using K-fold cross-validation (to find the optimal SVM hyperparameter). This final step produces a score, representing the performance of the method given the set of parameters Λ.

4. **Bayesian Optimization**: receives the new pair $(\Lambda, f(\Lambda)$ and updates the posterior distributions. The acquisition function produces a new set of parameters Λ for the next iteration. The set of parameters is used to compute the Rabi frequency waveform.

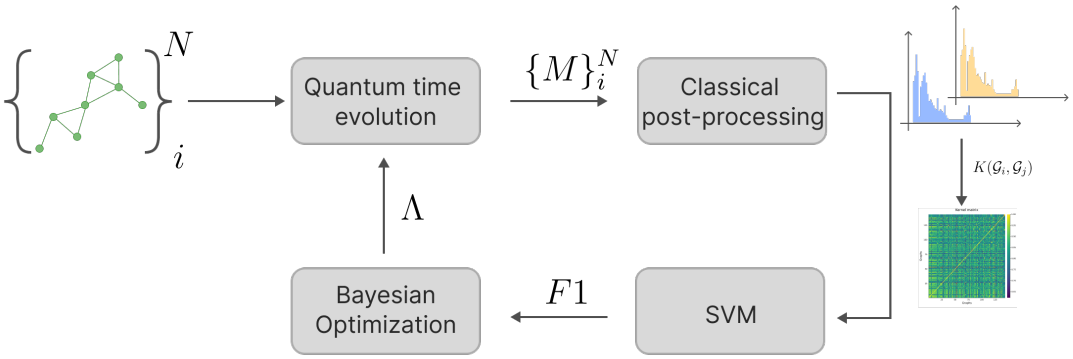


Figure 4.5: The full Hybrid quantum-classical emulation approach.

In the following, each part of the approach is detailed and discussed, in terms of the experimental settings and results.

4.2.1 Details on Energy distribution computation

As detailed in Chapter 3, the core of the method is the computation of a graph kernel as the distance between two probability distributions representing the graphs. In the following, the operative approach to computing the energy distribution is discussed. Aquila allows measurements only in the Rydberg basis, defined by the two states ($|r\rangle$, the Rydberg state that we use as $|1\rangle$, and the ground state $|g\rangle = |0\rangle$). The *energy observable* is then computed by post-processing the set of measurements.

Consider a graph $\mathcal{G}$, and suppose that the quantum procedure has produced a set of measurements M:

$$M = \{m_1, \ldots, m_k\},\ m_i = [a_1, \cdots, a_n],\ a_i = \begin{cases} 1 & \text{if atom i is measured in } |g\rangle \\ 0 & \text{if atom i is measured in } |r\rangle \end{cases}$$

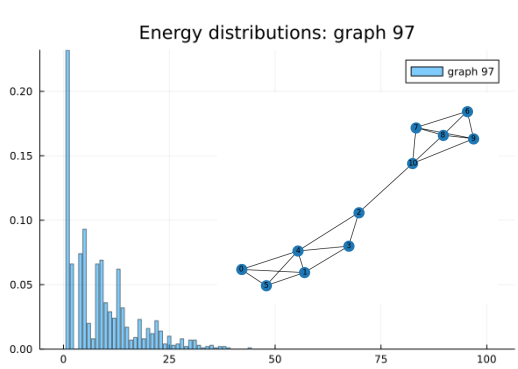
Here, the number of single measurements is $k \leq$ number of shots because the measurement process can fail (due to an erroneous atom initialization). This means that the energy distribution computation should take into account this possibility. Each measurement m_i is composed by a *bitstring* of length $|\mathcal{V}|$ (i.e. the number of nodes/atoms). It is important to notice that the single bit is flipped with respect to how is expected. The value 1 is measured when the atom is presented into the ground state. Since is common to find in quantum algorithms the measured value is equal to the logical state, a bit-flipping operation is applied to each measurement. For each measurement m_i we can associate an energy: e_i:

$$e_i = \sum_{j<k} V_{j,k} n_j n_k$$

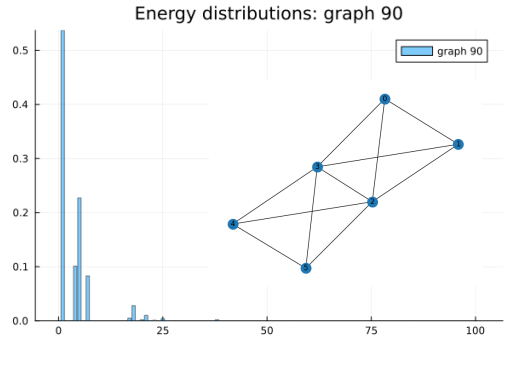
where $V_{j,k}$ is the *interaction strength*, typical of Rydberg atoms, and n_j is the *occupation operator*, equal to one only if the atom is in the excited state. By repeating the process for every measurement, a set of energies related to every single graph is computed, indicated as $E_{\mathcal{G}_i}$. In order to have comparable distributions, they should have the same support. For this reason, a binning procedure is used. Once every energy set is calculated for each graph, these 2 values are calculated:

$$e_1 = \min_i \{\min_{e_j} E_{\mathcal{G}_i}\} \ , e_2 = \max_i \{\max_{e_j} E_{\mathcal{G}_i}\}$$

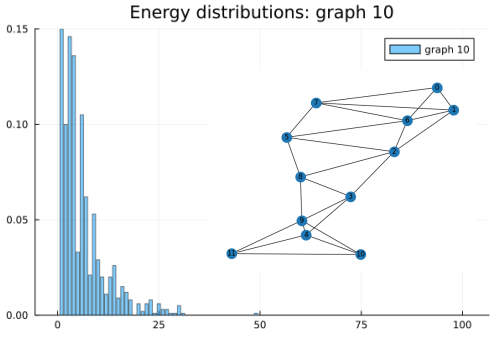
Where e_1 is the minimum of all measured energies, and e_2 is the maximum measured energy. Selecting 100 equal-sized bins in the interval $[e_1, e_2]$ allows for finding a probability distribution $\mathcal{P}(\mathcal{G})$. Bins are normalized to sum up to 1 by dividing by the number of measurements k. Examples of the probability distributions and the associated graph can be found in figure 4.6.



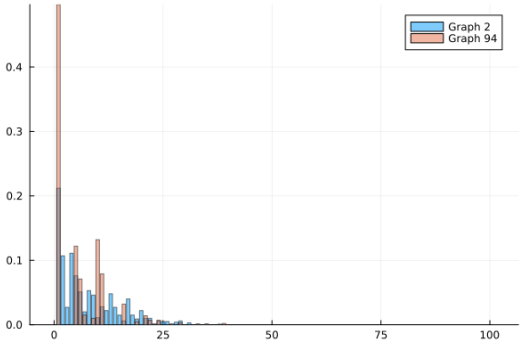
(a) Energy distribution and the associated graph



(b) Energy distribution and the associated graph



(c) Energy distribution and the associated graph



(d) Comparison of 2 energy distributions of 2 graphs belonging to different classes

Figure 4.6: Obtained energy distributions

4.2.2 Kernel estimation and training protocol

Once the probability distribution is estimated, the kernel is computed as described in eq. 3.6. The only hyperparameter involved in this part is μ. Following what is done in [29], μ is set equal to 1. Further hyperparameter tuning is possible, knowing that μ influences the image of the kernel. With $\mu = 1$ $K(\cdot) \in \left[\frac{1}{2}, 1\right]$, while for bigger values the image increases, and in the limit of $\mu \to \infty$ reaches $[0, 1]$. Once the quantum kernel is estimated, is organized in a kernel matrix K. The entries are organized as:

$$K_{i,j} = K(\mathcal{G}_i, \mathcal{G}_j)$$

The kernel matrix expresses the similarity between pairs of graphs and can be visualized graphically using a heatmap, as in figure 4.7.

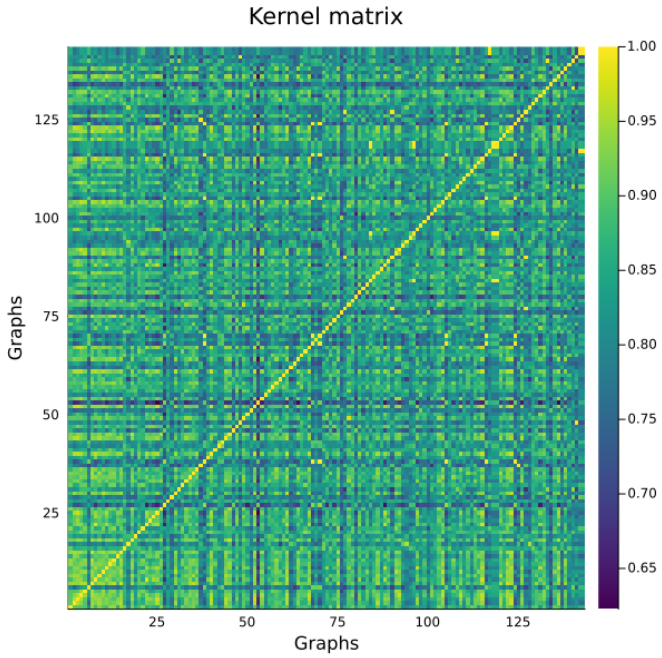


Figure 4.7: Kernel matrix of PROTEISN12 dataset, with $\mu = 1$

As we can notice, there is a zone in the bottom-left corner associated with a brighter color, which means higher kernel values. This group is composed of graphs associated with the labels *enzymes*, demonstrating qualitatively that the quantum kernel is able to find structures in data. The kernel matrix can be used to train the Support Vector Machine (more information in Appendix B). To evaluate the approaches and tune the SVM hyperparameter, validation data are generated using a K-fold cross-validation scheme (with $K = 10$), in combination with a *grid search* to find the best parameter. The hyperparameters tuned in this phase are summarized in table 4.3. For every possible combination of the parameter, 10 models are trained on 9 fold of the data, and validated on the remaining. Each model score is then collected and averaged, producing single metrics used for evaluation.

C	100 point in $[10^{-4}, 10^{4}]$, logarithm scale
w	30 pairs $[1, x_i]$ with x_i in $[1, 1000]$

Table 4.3: Grid search hyperparameter

The C hyperparameter expresses the possibility of the classifiers of making misclassification errors during the training procedure. These improve the generality of the solution and prevent overfitting. The additional pair of class weights w is

used to prevent the model from predicting always the majority class. They act as a penalty for misclassifying the minority class.

4.2.3 Bayesian optimization of Waveform parameter

The architecture follows the one proposed in [29] and detailed in Chapter 3. The selected approach consists of 2 layers, composed of 2 free evolution and 3 pulses consisting of a single qubit drive (i.e. the mixing Hamiltonian). The amplitude of the Rabi drive is kept constant and equal to the maximum amount reachable by the QPU ($\Omega_0 = 15.8\, rad/\mu s$, in order to reduce the overall evolution time, and consequently to reduce the noise sources). As discussed at the end of Chapter 3, the full dynamics can be realized using a time-dependent Hamiltonian with a $\Omega(t)$ governed by a pulse that alternates values where the laser is on (corresponding to the mixing Hamiltonian) with a period where the laser is off, corresponding to free evolution of the quantum state. Given this consideration, the parameters Λ can be rewritten as 5 numbers:

$$\Lambda = \{\tau_0, t_0, \tau_1, t_1, \tau_2\}$$

Where τ_i represents the drive duration in time (and consequently the angle of rotation), while t_i represents the free evolutions. Given a set of parameters Λ. the training protocol described before returns the best SVM hyperparameter that reaches the best average score among the 10 folds. However, the pulse parameter has to be tuned, in order to find the pulse that extracts a good amount of knowledge (i.e. has the maximum F1 score). Formally can be seen as a maximization problem, find the best Λ that maximizes $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is the space of the parameters Λ, and f is the average selected score (F-1 in this case) among the 10 folds of the PROTEINS12 dataset. However, the evaluation of f is costly, since it consists of training a model and computing for every graph the state several times, and common heuristic methods are not feasible. Bayesian optimization [39] is particularly well suited for optimizing *black-box functions*, in which evaluating different sets of parameters can be costly. The framework is made of two components:

- *Surrogate function* $\tilde{f}$, that approximates the costly objective function starting from a set of evaluations of f. In Bayesian optimization, it is usually called *prior* and reflects the knowledge we have on the function f.

- *Acquisition function* $\alpha(x)$: indicates where to sample the next point to evaluate f.

It is common to select the surrogate function as a Gaussian Process $\mathcal{GP}$, characterized by a mean value μ and a covariance matrix Σ: The Gaussian process so

defined has a probability density function of the form:

$$P(X) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} \cdot \exp\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right)$$

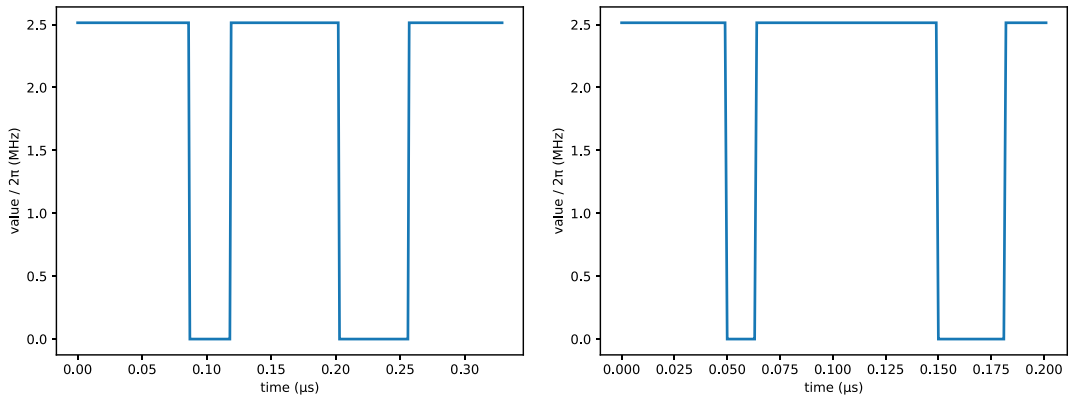
The selected mean μ is Normal distributed as $\mathcal{N}(0, 10)$, while the covariance function is selected as the *Isotropic* 5/2 *Matèrn Kernel*, with length scale $\ell = 10$ and signal standard deviation $\sigma = 10$:

$$K(x, x') = \sigma^2(1 + \sqrt{5|x-x'|/\ell + 5|x-x'|^2/(3\ell^2)})\exp\left(-\sqrt{5|x-x'|/\ell}\right)$$

The acquisition function is the commonly employed *Upper Confidence Bound*, $\alpha(x) = \mu(x) + k\sigma(x)$, where K is a hyperparameter that expresses the trade-off between exploration (μ parameter) and exploitation (σ parameter). The kernel hyperparameters (length scale and signal standard deviation) are optimized every 10 iterations, using maximum a posteriori (MAP). The maximum number of iterations is set to 50. In addition, some constraints are applied to the optimized variables:

$$\begin{aligned} \tau_0 + t_0 + \tau_1 + t_1 + \tau_2 &< 500ns \\ t_i &> 5ns \qquad \forall i \in [0, 1] \\ \tau_i &> 5ns \qquad \forall i \in [0, 1, 2] \end{aligned}$$

The first constraint limits the total emulation time, preventing the *decoherence* phenomena and reducing consequently the emulation time, while the second and third constraints are related to the finite bandwidth of the optical components. The waveform obtained by the entire process on the PROTEINS12 is then compared with the one obtained by Pasqal in their paper [29], in Figure 4.8:



(a) Pasqal parameter **(b)** Bayesian Optimization parameter

Figure 4.8: Waveform parameters comparison

In general, the two pulses seem equivalent. Both are characterized by shorter free-evolution and longer periods of the mixing Hamiltonians $\mathcal{H}_\theta$. The first great difference is in the total duration of the pulse: the one found by Pasqal has a duration of $317\,ns$, in contrast to the $201\,ns$ of the one found in this work. These differences of $\simeq 100\,ns$ have two major benefits: first, the emulation and simulation time is shorter, allowing more executions in the same time. In addition, the shorter protocol shows less noise since most sources of noise come from longer time evolution. This allows the simulation to be more precise and should provide the same results as the emulation on the classical machine. It is possible to retrieve the angle of rotation of the mixing Hamiltonian by remembering that:

$$\theta = \Omega_0\, t \tag{4.1}$$

The second substantial difference consists in the mixing Hamiltonian duration: for the Pasqal parameter, the three mixing Hamiltonian have similar duration ($87\,ns$, $84\,ns$ and $72\,ns$), while the optimized parameter found in this setting are characterized by a central and long pulse (comparable to the one in Pasqal waveform, $85\,ns$ corresponding to a rotation $\simeq 80°$) and two shorter pulses of $50\,ns$ and $20\,ns$, corresponding respectively to a rotation of 45° and 18°. Qualitatively, it is possible to compare the energy distribution obtained with the two different sets of parameters, as shown in figure 4.9.

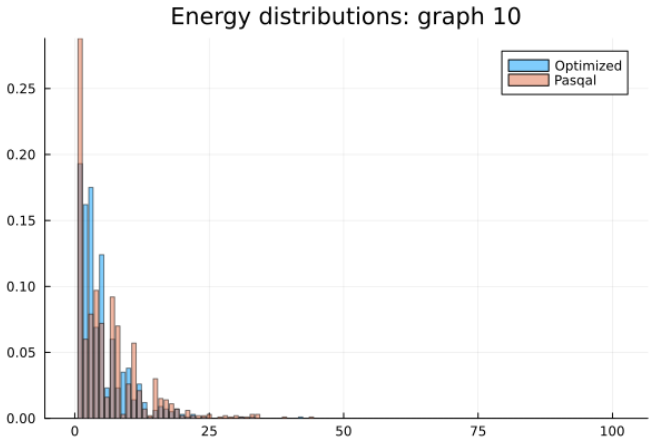


Figure 4.9: Comparison of the energy distributions obtained using two different sets of waveform parameters on the same graph

4.2.4 Results on PROTEINS12

Finally, the results obtained in the emulation experiment can be presented, in the form of 4 overall metrics, presented in Chapter 1. As a benchmark, the method

is compared with itself, but with the parameter found by Pasqal [29], and with a classical algorithm, the *Shortest Path Graph kernel* [40]. The emulation platform is based on a *Dell-XPS 15 7590*. The results are presented in table 4.4.

	F-1 (%)	Accuracy (%)	Precision (%)	Recall (%)
Pasqal	41.9	44.8	29.3	89.1
Bayesian Optimization	**46.3**	52.4	37.1	85
Shortest Path kernel	44.1	**61.4**	35	68

Table 4.4: Quantum evolution kernel with Pasqal parameter, with Bayesian Optimization parameter compared to a classical kernel, the Shortest Path Graph Kernel

The best results of the three models are comparable. The kernel computed with the optimized parameter outperforms the one obtained by Pasqal (By $\simeq 5\%$ on the F1 score). The improvements on F1 score reflects also in an higher accuracy (44 % against 52.4%) and in an higher precision (29.3% against 37%). Compared to the classical kernel, the method presented in this thesis still performs better in terms of F1 score, but with a minor margin (44.1% of the Shortest Path Kernel vs. the 46.1% of quantum evolution kernel). However, the accuracy of the classical model is higher, because of the high-class imbalance. This shows how in this case the accuracy score can be misleading. In fact, both the precision and the recall metrics are lower.

It can be useful to analyze also the time taken by the presented method, in particular by the quantum emulation. Considering that once the kernel matrix is estimated the Support Vector Machine training time is constant, the benefits in time execution should be considered for the kernel computation time only. As described in the previous section, the quantum evolution is done for all the graphs, and the energy distribution is computed once for all. As described in Chapter 3, the Rydberg blockade mechanism, besides providing a way of implementing entanglement, allows to run emulation in a reduced subspace, allowing to save time. Details are reported in Figure 4.10 and in Table 4.5.

Full space	27 s
Rydberg subspace	25 s

Table 4.5: Total emulation time, comparison between full emulation and subspace emulation

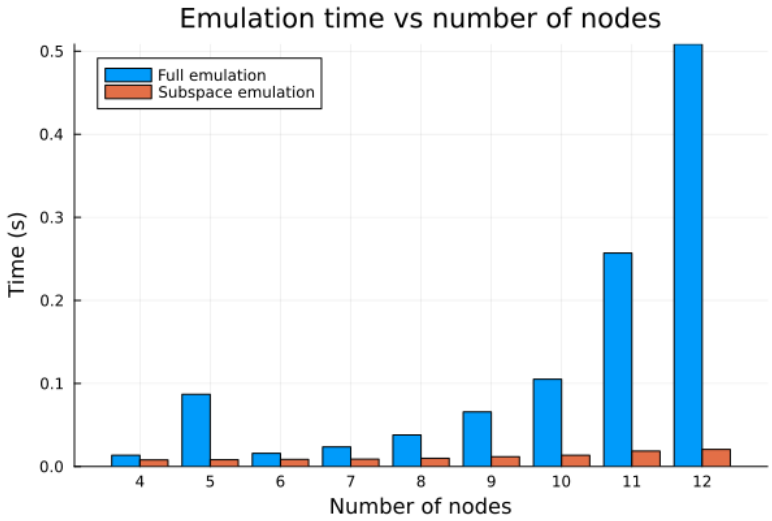


Figure 4.10: Emulation time in the full space against the subspace defined by the Rydberg Blockade, in the function of the number of atoms

The Figure shows that the emulation time follows an exponential behavior with the number of nodes, while in the case of the Rydberg subspace emulation, the time is much shorter. This could be expected since the subspace emulation allows deleting the high energy states corresponding to the doubly excited Rydberg state. Since the emulated dynamics consist of graphs and the edges corresponding to atoms in the range of the Rydberg radius, a lot of states are eliminated. However, it could be useful to analyze the results in the case of the subspace emulation, both qualitatively and quantitatively. For example, in figure 4.11

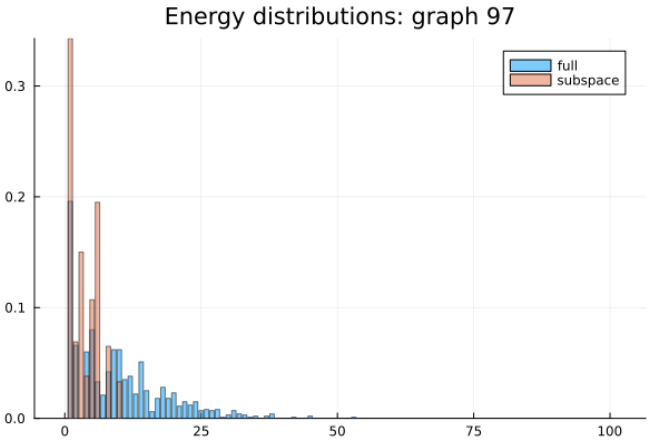


Figure 4.11: Comparison between the energy distribution in the full subspace and in the Rydberg Blockade subspace

As can be expected, the Rydberg subspace energy distribution is more squeezed towards the lower energies, as an effect of the truncating of the high energy states. This obviously reflects the performances, as shown in Table 4.6.

	F1(%)	Accuracy(%)	Precision(%)	Recall(%)
Rydberg subspace	41.8	44.8	29.3	89.1
Full space	**46.3**	52.4	37.1	85

Table 4.6: Rydberg subspace emulation performances on PROTEINS12

As expected the Rydberg subspace emulation performances are worse than the full space emulation. Since the training protocol consists of several iterations when the kernel matrix is kept constant, the time performances of emulation don't play a so crucial role, because the total training time is dominated by the grid search.

4.3 Simulation on Aquila

Thanks to the collaboration with **LINKS foundation** and **QuEra Computing**, it was possible to execute the proposed algorithm on a real noisy Quantum Computer, Aquila, presented in Chapter 3. Aquila Quantum computer is accessible through Amazon Web Service (in particular *Amazon Braket*, AWS cloud quantum computing service). Braket provides access to different devices and a simulator:

- *Superconducting QPUs*: provided by Rigetti and Oxford Quantum Circuits.
- *Ion trap QPUs*: provided by ionQ.
- *Neutral atom QPU*: provided by QuEra.

QuEra provides all the software tools to interact with his hardware through *BloqadeHardware* libraries, in particular providing support for transforming the Hamiltonian in a format that Aquila can understand and for validation of the various constraints. Before delving into the results obtained through quantum simulation, it is important to highlight the costs involved with running on quantum hardware:

- **Activity cost**: 0,30 USD, it is a fixed cost, can be associated with a single task. A single task consists of a single Hamiltonian (that embeds in his structure both atoms positions and waveform parameters). In the end, the final state can be measured.
- **Shot cost**: 0.01 USD for a single measurement, if belongs to the same activity.

For example, let's consider a task. Since a single task can be associated with only a single Hamiltonian, every task is associated with a single graph. The choosen number of shots is 1000. In this case, the total cost for estimating the graph probability distribution is:

$$C = \text{Activity cost} + \#\text{shots} \times \text{Shot cost} = 0.30 + 1000 \times 0.01 = 10{,}30\,\text{USD}$$

Considering the number of graphs, and the 2 sets of parameters tested, the total cost of the whole experiment is 4315.7 USD.

4.3.1 Comparison of energy distributions

Before executing the full experiment, consisting of running the time evolution for every graph, a random subset is extracted, and the results of the quantum simulation are compared to the one obtained with the emulation on classical hardware. The comparison is done in term of the obtained energy distribution since computing the statevector would require an exponential number of measurements. The obtained energy distribution are compared in figure 4.12.

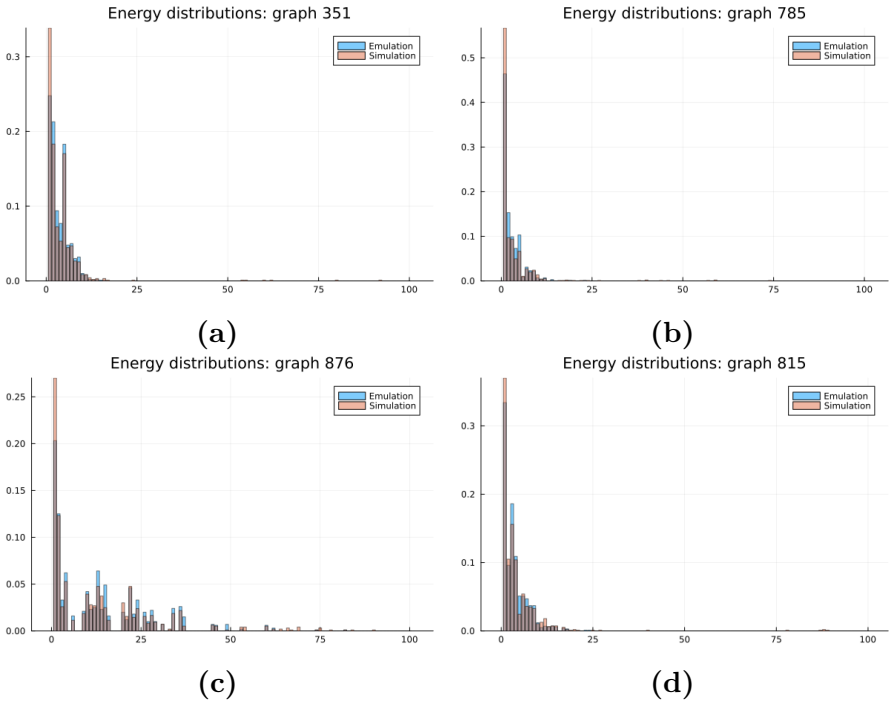


Figure 4.12: Comparison between the emulated energy distribution and the simulated energy distribution on a single random graph

The simulated energy distribution is very similar to the emulated one. An important feature to notice is that higher energies are more probable in the

simulated than the emulated one. These can be partially explained by the detection errors that affect atoms in the ground state $|g\rangle$ to be detected as in the Rydberg state $|r\rangle$, increasing the probability of detecting a double-excited Rydberg state. This phenomenon is particularly noticeable in figure 4.12 (c).

4.3.2 Results on PROTEINS12

Once a first round of simulation is done in order to check for possible errors in the Hamiltonian definition (runtime error in simulation, due to violation of Aquila constraints), the experiment is repeated for the full PROTEINS12, in order to compare the emulated and simulated results. The experimental setup is the same as presented before, with the same classical post-processing in order to compute the energy distribution and the K-fold cross-validation approach in order to find the SVM hyperparameter. The experiment is presented with the Pasqal parameter and with the parameter found by the Bayesian optimization method. The choice of simulating both parameters comes from the consideration that the authors of the original Pasqal paper [29] did not simulate on a real machine, and they highlighted that an interesting way to continue their work would be to simulate on an actual Neutral Atoms machine. The second consideration is to highlight the differences in terms of the result for 2 protocols with different durations. As explained in the previous section, the optimized protocol has a duration of $100\,ns$ less, the 33% less. These can help to characterize the noise that characterizes longer protocols. The results are presented in table 4.7.

	F-1 (%)	Accuracy (%)	Precision (%)	Recall (%)
Pasqal	41.9	44.8	29.3	89.1
Bayesian Optimization	**46.3**	52.4	37.1	85
Shortest Path kernel	44.1	**61.4**	35	68
Aquila simulation (Pasqal)	**48**	62.2	35.9	80.8
Aquila simulation	**49**	63.7	39.5	78

Table 4.7: Quantum evolution kernel with Pasqal parameter, with Bayesian Optimization parameter compared to a classical kernel, the Shortest Path Graph Kernel

The result confirms that the method is effective, even in the case of the real noisy simulation. The simulation on the Aquila quantum computer, with the optimized waveform parameter, has the best result, outperforming both the chosen classical method and the emulated one (of about 3 percentage points in the F1 metric). The better result in terms of the F1 metric also reflects the accuracy metric, with an accuracy of 63.7%, the best among all the methods (even the classical kernel that

performs better in terms of the accuracy metric than the emulated QEK). Also, for the Pasqal parameters, the simulation scores are better than the emulated ones. In terms of noise, the longer protocol does not seem to interfere with the model performance, registering one of the best results among all the methods.

4.3.3 Results on PROTEINS256

Once the results are compared among the methods, comparing the emulated and simulated results, an experiment can be conducted using the full dataset. The results can be found in table 4.8.

	F-1 (%)	Accuracy (%)	Precision (%)	Recall (%)
Aquila simulation	**65,6**	**65,4**	**55,1**	86,6
SPK	65,3	64,9	53,5	**87,1**

Table 4.8: PROTEINS256 Quantum Evolution Kernel performances

As expected, the performance on the reduced dataset was not so good since the number of training sample, even if using a K-fold-cross validation approach with an high K ($K = 10$). When the model has some additional datapoints, the performances rises. Regarding the performances of the simulation, is clear that the performances are much higher (about 20 percentage points on the F1 metric) than the reduced dataset. This can be partially explained due to the high-class imbalance. Even if a weighting training objective is used and a cross-validation approach with a high value of K is used, the number of training samples is not enough to provide reliable classification results.

The full dataset does not present an evident imbalance between classes. Even in this experimental setting, the simulation of the Quantum Evolution kernel on Aquila with the optimized parameter has performances that are comparable to the one of the classical SPK kernel.

An additional analysis can be done. The number of shots required to compute the energy distribution should be carefully chosen. The number of shots is directly related to the cost of the experiment, both in monetary cost and in time. As seen in chapter 3, Aquila is able to do a full cycle at a rate of 10 Hz. This means that while an experiment with a single graph could last 10 seconds with 100 shots, with 1000 shots it would last 10 times more. To assess how many shots are necessary for the model to reach an overall good performance, a sampling approach has been used. From the whole set of measurements, a number k of shots is drawn randomly and without replacement, simulating an experiment with k number of shots. Then, the same post-processing and cross-validation approach was used. Results are shown in Table 4.9.

number of shots	F-1 (%)	Accuracy (%)	Precision (%)	Recall (%)
10	59,3	63,1	52,9	71
100	**65,1**	64,4	53,8	87
1000	**65,6**	65,4	55,1	86,6

Table 4.9: PROTEINS256 performances against the number of shots

As reported in the table, the overall performance does not degrade much when the number of shots is reduced to 100 (The average degradation in performance is about 1 point percent), while it rapidly decreases when the number of samples is reduced to 10, as it could be expected.

Chapter 5

Conclusions and Future Works

5.1 Future works

The main objective of this work was to follow the one proposed by Pasqal [29]. Different paths can be followed to improve this thesis work. In the following, I'll list the improvements that I would make:

- *Using different datasets*: the first work can be integrating the most used datasets(like Fingerprint, IMDB, etc.), in combination with trying to improve the DEN model performances. An interesting subsequent work can be creating the UD embedding for all these datasets.
- *Different layered structures*: in this thesis, a 2-layer approach with an Ising Hamiltonian as graph Hamiltonian is used. In follow-up work, analyzing the effect of the number of layers on the performances can be interesting. In addition, different Hamiltonians can be used. In Chapter 3, the XY Hamiltonian is presented as an alternative.
- *Different observables*: the energy observable is taken as a candidate for describing the characteristics of the graph. One can explore the possibility of using another observable, like the Rydberg occupancy. In addition, the method for building the distribution is based on binning the measured values. Other methods, employing the Fourier transform of the measurement operator can be employed.

Appendix A

Bayesian Optimization

A.1 Introduction

Bayesian optimization is a mathematical framework for global optimization for expensive to-evaluate functions. Global optimization because the objective is typically to find the global optima, rather than local optima. The typical function characteristics are[39]:

- f is a *black-box* function, this means that f has no special structure such concavity or linearity that allows using other methods.
- f is expensive to evaluate, both in terms of time required to compute a value in a specific point $f(x)$ or in terms of resources. It's usually required that f is continuous.
- it's not possible(because it's not known or it's not possible to do) to compute first and second-order derivatives of the function f, so it's not possible to use gradient-based methods.
- f takes in input an array $\mathbf{x}$ with a limited dimension(typical 20). In addition, $\mathbf{x}$ belongs to a set A, where it's easy to assess the membership. A is an hyper-rectangle in $\mathbb{R}^d$, where d is the dimensionality of the search space.

Bayesian Optimization is well suited for solving tasks of the type

$$arg\max_{x \in A} f(x)$$

where f can be summarized as to be *black-box derivative-free global optimization*[39]. The approach consists of two components:

- A *surrogate function* $\tilde{f}$, for modeling the function f, constructing a Bayesian posterior probability based on the already evaluated point.

- An acquisition function $\alpha(x)$ that decides where to sample the next point to evaluate, based on the posterior defined by the surrogate function.

The Bayesian optimization framework belongs to a wider part of methods resumed with the name of "surrogate methods". The main difference between the other is the use of Bayesian statistics to model the surrogate function and consequently the acquisition function. Given these two components, the generic Bayesian optimization algorithm is detailed in algorithm 1:

Algorithm 1 Bayesian Optimization algorithm[39]

Place a Gaussian process prior on f
Observe f at n_0 points uniformly drawed in A
Set $n = n_0$
while $n \leq N$ **do**
    Update the posterior probability distribution on f using all available data
    Compute the acquisition function using the current posterior distribution.
    Let x_n be a maximizer of the acquisition function over x.
    Observe $y_n = f(x_n)$.
    Increment n
end while
Return a solution: either the point evaluated with the largest $f(x)$, or the point with the largest posterior mean.

A.2 Surrogate function and Gaussian Processes

Surrogate functions are a key component of the Bayesian Optimization framework, since it models the function f to be optimized. Bayesian methods relies on a prior knowledge on the surrogate function, usually a Gaussian process(GP), that approximates the values of $f(x)$. The process of approximating f is called GP regression. The method consist of observing f in a collection of point $x_1, \cdots, x_k$ $x_i \in \mathbb{R}^d$, where d is the dimension of the input space. The observed values $f(x_1), \cdots, f(x_k)$ are assumed to be drawn from some unknown prior probability distribution. In the Gaussian process hypothesis, the prior is assumed to be multivariate normal, with a particular mean vector μ and covariance matrix Σ_0.

$$f(x_{1:k}) = [f(x_1), \cdots, f(x_k)] \sim \mathcal{N}(\mu_0(f(x_{1:k})), \Sigma_0(f(x_{1:k}))) \tag{A.1}$$

Where μ_0 is the mean vector, computed with the mean function $\mu(x)$ and Σ_0 is the covariance matrix, computed using the kernel Σ between each pair x_i and x_j. Once that the kernel matrix and mean vector are computed, a Bayesian posterior probability is obtained as in eq. A.1. As described in section 3 of [39],

it's possible to model the *posterior probability distribution* $f(x)|f(x_{1:n})$(that is Normally distributed according to a mean and covariance matrix that depend on the ones computed on the point $f(x_{1:n})$), that represent the probability of obtaining a certain value of $f(x)$ given the knowledge we have on the previously assumed value of f.

Since, the prior is assumed a Gaussian in the majority of the case of application of Bayesian optimization, the unique choice to made in this context are mean function and kernel function. Starting from the easiest, the mean function is used to a *constant*, $\mu_0(x) = \mu$. On the other hand, for the kernel function, different choices can be done. The choce of the kernel function is particularly relevant, since it should reflect the properties of the function to be approximated. In particular, the kernel function should behave in a way that the closer point in the input space should have higher kernel function, i.e. it should be strongly correlated. The most common employed kernel is the *Màtern Kernel*:

$$\Sigma_0(x, x') = \alpha \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\|x - x'\|\right)^{\nu} K_\nu(\sqrt{2\nu}\|x - x'\|) \tag{A.2}$$

Where K_ν is the modified Bessel function, and $\Gamma()$ is the gamma function. The kernel function comes with different parameter, α that can be optimized trough MAP or MLE, while ν is an hyperparameter that characterize the kernel.

A.3 Acquisition function

The acquisition functions are another key component of the methods, that starting from the posterior probability distribution defined by the surrogate function select the next point where to evaluate. Different choice are possible, such *expected improvement*, *knowledge gradient*, and *lower(or upper) confidence bound*. In the following, the *upper confidence bound* is discussed[41][42], consisting of sampling point maximizing the expected reward(i.e., an upper bound) on the function to be optimized. The fuction takes the form of

$$\alpha(x) = \mu(x) + k\sigma(x) \tag{A.3}$$

Where k is a hyperparameter that decide how the exploration(high K) should be preferred against exploitation(lower K).

Bibliography

[1] Alan Mathison Turing. «Computing machinery and intelligence (1950)». In: (2021) (cit. on p. 1).

[2] Tom M Mitchell. *Machine learning*. 1997 (cit. on p. 3).

[3] C Bishop. «Pattern recognition and machine learning». In: *Springer google schola* 2 (2006), pp. 531–537 (cit. on p. 5).

[4] Corinna Cortes and Vladimir Vapnik. «Support-vector networks». In: *Machine learning* 20 (1995), pp. 273–297 (cit. on p. 6).

[5] QUANTUMPEDIA - The Quantum Encyclopedia. *A Brief History of Quantum Computing*. 2023. URL: https://quantumpedia.uk/a-brief-history-of-quantum-computing-e0bbd05893d0 (visited on 2024) (cit. on p. 16).

[6] Paul Benioff. «Quantum mechanical Hamiltonian models of Turing machines». In: *Journal of Statistical Physics* 29 (1982), pp. 515–546 (cit. on p. 16).

[7] David Deutsch. «Quantum theory, the Church–Turing principle and the universal quantum computer». In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117 (cit. on p. 16).

[8] David Deutsch and Richard Jozsa. «Rapid solution of problems by quantum computation». In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558 (cit. on p. 16).

[9] Daniel R Simon. «On the power of quantum computation». In: *SIAM journal on computing* 26.5 (1997), pp. 1474–1483 (cit. on p. 16).

[10] David P DiVincenzo. «The physical implementation of quantum computation». In: *Fortschritte der Physik: Progress of Physics* 48.9-11 (2000), pp. 771–783 (cit. on pp. 17, 29).

[11] URL: https://ocw.mit.edu/courses/22-51-quantum-theory-of-radiation-interactions-fall-2012/9cdcc3c1e36da2ae3e2f925d5b435ab6_MIT22_51F12_Ch3.pdf (visited on 03/01/2024) (cit. on p. 17).

[12] Benjamin Schumacher. «Quantum coding». In: *Phys. Rev. A* 51 (4 Apr. 1995), pp. 2738–2747. DOI: 10.1103/PhysRevA.51.2738. URL: https://link.aps.org/doi/10.1103/PhysRevA.51.2738 (cit. on p. 18).

[13] Paul Adrien Maurice Dirac. «A new notation for quantum mechanics». In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 35. 3. Cambridge University Press. 1939, pp. 416–418 (cit. on p. 19).

[14] Felix Bloch. «Nuclear induction». In: *Physical review* 70.7-8 (1946), p. 460 (cit. on p. 19).

[15] URL: https://it.wikipedia.org/wiki/Sfera_di_Bloch (visited on 03/01/2024) (cit. on p. 20).

[16] Albert Einstein, Boris Podolsky, and Nathan Rosen. «Can quantum-mechanical description of physical reality be considered complete?» In: *Physical review* 47.10 (1935), p. 777 (cit. on p. 22).

[17] John S Bell. «On the einstein podolsky rosen paradox». In: *Physics Physique Fizika* 1.3 (1964), p. 195 (cit. on p. 22).

[18] URL: https://github.com/qiskit-community/qgss-2023 (visited on 03/01/2024) (cit. on p. 27).

[19] John Preskill. «Quantum computing in the NISQ era and beyond». In: *Quantum* 2 (2018), p. 79 (cit. on p. 30).

[20] Salonik Resch and Ulya R Karpuzcu. «Quantum computing: an overview across the system stack». In: *arXiv preprint arXiv:1905.07240* (2019) (cit. on p. 30).

[21] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. «Quantum algorithms for supervised and unsupervised machine learning». In: *arXiv preprint arXiv:1307.0411* (2013) (cit. on p. 33).

[22] Dieter Jaksch, Juan Ignacio Cirac, Peter Zoller, Steve L Rolston, Robin Côté, and Mikhail D Lukin. «Fast quantum gates for neutral atoms». In: *Physical Review Letters* 85.10 (2000), p. 2208 (cit. on p. 36).

[23] Manuel Endres et al. «Atom-by-atom assembly of defect-free one-dimensional cold atom arrays». In: *Science* 354.6315 (2016), pp. 1024–1027 (cit. on p. 36).

[24] Hannes Bernien et al. «Probing many-body dynamics on a 51-atom quantum simulator». In: *Nature* 551.7682 (2017), pp. 579–584 (cit. on p. 36).

[25] Jonathan Wurtz et al. «Aquila: QuEra's 256-qubit neutral-atom quantum computer». In: *arXiv preprint arXiv:2306.11727* (2023) (cit. on pp. 36–38, 42, 43, 48).

[26] Jonathan Wurtz, Pedro LS Lopes, Nathan Gemelke, Alexander Keesling, and Shengtao Wang. «Industry applications of neutral-atom quantum computing solving independent set problems». In: *arXiv preprint arXiv:2205.08500* (2022) (cit. on p. 44).

[27] Sepehr Ebadi et al. «Quantum optimization of maximum independent set using Rydberg atom arrays». In: *Science* 376.6598 (2022), pp. 1209–1215 (cit. on p. 44).

[28] Peter Schauss. «Finite-range interacting Ising quantum magnets with Rydberg atoms in optical lattices-From Rydberg superatoms to crystallization». In: *arXiv preprint arXiv:1706.09014* (2017) (cit. on pp. 46, 48).

[29] Louis-Paul Henry, Slimane Thabet, Constantin Dalyac, and Loïc Henriet. «Quantum evolution kernel: Machine learning on graphs with programmable arrays of qubits». In: *Physical Review A* 104.3 (2021), p. 032416 (cit. on pp. 46–48, 53, 61, 63, 64, 66, 70, 73).

[30] Jianhua Lin. «Divergence measures based on the Shannon entropy». In: *IEEE Transactions on Information theory* 37.1 (1991), pp. 145–151 (cit. on p. 47).

[31] Paul D Dobson and Andrew J Doig. «Distinguishing enzyme structures from non-enzymes without alignments». In: *Journal of molecular biology* 330.4 (2003), pp. 771–783 (cit. on p. 51).

[32] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. «Protein function prediction via graph kernels». In: *Bioinformatics* 21.suppl_1 (2005), pp. i47–i56 (cit. on p. 51).

[33] Ryan A. Rossi and Nesreen K. Ahmed. «The Network Data Repository with Interactive Graph Analytics and Visualization». In: *AAAI*. 2015. URL: `https://networkrepository.com` (cit. on p. 51).

[34] Brent N Clark, Charles J Colbourn, and David S Johnson. «Unit disk graphs». In: *Discrete mathematics* 86.1-3 (1990), pp. 165–177 (cit. on p. 53).

[35] Heinz Breu and David G Kirkpatrick. «Unit disk graph recognition is NP-hard». In: *Computational Geometry* 9.1-2 (1998), pp. 3–24 (cit. on p. 54).

[36] Chiara Vercellino, Paolo Viviani, Giacomo Vitali, Alberto Scionti, Andrea Scarabosio, Olivier Terzo, Edoardo Giusto, and Bartolomeo Montrucchio. «Neural-powered unit disk graph embedding: qubits connectivity for some QUBO problems». In: *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE. 2022, pp. 186–196 (cit. on pp. 54, 55).

[37] *Bloqade.jl*. URL: `https://queracomputing.github.io/Bloqade.jl/stable/` (visited on 02/13/2024) (cit. on p. 58).

[38] *The Julia Programming Language*. URL: `https://julialang.org/` (visited on 02/13/2024) (cit. on p. 58).

[39] Peter I Frazier. «A tutorial on Bayesian optimization». In: *arXiv preprint arXiv:1807.02811* (2018) (cit. on pp. 63, 74, 75).

[40] Karsten M Borgwardt and Hans-Peter Kriegel. «Shortest-path kernels on graphs». In: *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE. 2005, 8–pp (cit. on p. 66).

[41] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. «Practical bayesian optimization of machine learning algorithms». In: *Advances in neural information processing systems* 25 (2012) (cit. on p. 76).

[42] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. «Gaussian process optimization in the bandit setting: No regret and experimental design». In: *arXiv preprint arXiv:0912.3995* (2009) (cit. on p. 76).