Quantum Reservoir Computing (QRC) Algorithms to Address Credit Risk Modeling

Brian Brueggert

January 10, 2025

Abstract

The focus of this paper is to describe the solution to a Project proposed by Bloq Quantum, namely; to build a AQSVM or Quantum Reservoir Computing (QRC) algorithm to address credit risk modeling, specifically using the Statlog German Credit Data dataset. Following an introduction, the basic principles of Quantum Reservoir Computing are discussed, and then the paper proceeds to cover the descriptions, algorithms and code implementations for each of the Classical Logistic Regression Model, introducing the Continuous and Binary Ising Models each with a linear, ridge and QUBO regression predictor version. It is intended to accompany the code for this Project, and contains a list of the prerequisites libraries in the Requirements section at the end.

1 Introduction

This work is the result of a Project proposed by Bloq Quantum to build a AQSVM or Quantum Reservoir Computing (QRC) algorithm to address credit risk modeling (a useful metric for estimating economic capital required for ...), specifically using the Statlog German Credit Data dataset. The Project encourages unconventional thinking, and seeks new and original ideas for implementing quantum reservoir systems or innovative algorithms, pushing the boundaries of what is possible.

The work here begins with a classical algorithm (Classical Logistic Regression Model.py), for a baseline comparison. Next, two variations of QRC algorithms are considered; named "Continuous Quadratic Ising Model" (CQIM) and "Binary Quadratic Ising Model" (BQIM). Each algorithm is explored for generating feature vectors, with two variants of each considered - both simple linear regression, as well as a more complex ridge regression, for making predictions. Finally, some Quadratic Unconstrained Binary Optimization (QUBO) algorithms are explored in conjunction with each of the feature vector extraction algorithms (CQIM and BQIM) to make predictions.

Some prerequisites to running the code in this Project include setting up a D-Wave Leap account, and leveraging the API token therein to activate the Cloud environment. In this case, this was configured via a terminal prompt inside of a Python 3.11.1 virtual environment, from where all of the code was run. The additional Python package/version requirements can be found in the requirements.txt document included in the Project, and are listed in the Requirements section at the end of this paper.

2 Basic Principles of Quantum Reservoir Computing

The basic principles of utilizing dissipation as a resource for QRC [QAISG [1]] involve, firstly, the association of physical points in a substrate x_k with states ρ_k in the reservoir.

Second, a spin state preparation (non-unitary transformation) must be performed on the substrate:

$$|\psi_{s_k}\rangle = \sqrt{1-s_k}|0\rangle + \sqrt{s_k}|1\rangle \xrightarrow{\text{(non-unitary)}} \rho' = |\psi_{s_k}\rangle\langle\psi_{s_k}| \otimes \mathrm{Tr}^{(1)}\{\rho_{k-1}\} \tag{1}$$

Figure 1: N = 5 node clique spin network. (Image credit QAISG [1].)

Due to the disordered (spin) quantum ensemble dynamics of the reservoir interacting with the environment through dissipation, the states within the reservoir evolve unitarily as:

$$\begin{aligned} \rho_k &= e^{-iH\Delta t}\rho' e^{iH\Delta t} \quad \text{(unitary)} \\ &= e^{-iH\Delta t}[|\psi_{s_k}\rangle\langle\psi_{s_k}| \otimes \mathrm{Tr}^{(1)}\{\rho_{k-1}\}]e^{iH\Delta t} \end{aligned} \tag{2}$$

Here the Hamiltonian provides the unitary evolution. The concept is that by putting the reservoir into a frustrated quantum superposition of states and allowing it to evolve unitarily via interaction with the environment through dissipation, we are essentially sampling low energy states representative of random feature vectors which capture quadratic relations or non-linear dynamics of the system. It's based on the idea that you can't contain neither heat nor energy, as it eventually dissipates into the environment. As such, it is noteworthy to mention that the information processing we seek should necessarily lie outside of the coherent quantum regime; past the boundary with the environment defined by extension of the Kibble-Zurick mechanism, with the information processing rate directly proportional to the rate of energy dissipation of the system.

3 Algorithm Overview

3.1 Classical Logistic Regression Model

3.1.1 Description

The data is briefly explored, processed, split and one hot encoded, as is standard practice. This model simply runs a classical logistic regression on the dataset, and is intended as a warm up to understanding the problem, as well as a baseline comparison.

3.1.2 Algorithm

Algorithm 1 Classical (Binary) Logistic Regression Model

Require: $n \geq 1$

for $\forall n$ **do**
  $d_i : i = 1...k$
  while $i \leq k$ **do**
    for each training data instance d_i **do** set the target value for the regression to

$$z_i = \frac{y_{i-P(1|d_j)}}{[P(1|d_j)(1 - P(1|d_j))]}$$

      $d_j \leftarrow [P(1|d_j)(1 - P(1|d_j))]$
      for $f(j)$ **do**

$$\text{class value} \leftarrow Z_j, \text{ weight } \leftarrow w_j$$

      end for
    end for
  end while
end for
return classical label decision

3.1.3 Code Implementation

```
sudo python3 Classical_Logistic_Regression_Model.py
```

3.2 Continuous Quadratic Ising Model

3.2.1 Description

In this version of a QRC model, herein named the "Continuous Quadratic Ising Model", the logical structure of the problem is composed as follows: Consider a datum consisting of four attribute columns and a single class column (N = 5) as a training set schema for random feature vector extraction. In order to capture all of the quadratic terms (first-order non-linear dynamics), the logical structure of the problem assumes the topology of an N = 5 node clique graph, G (Fig.2).

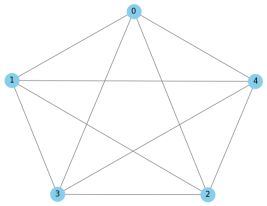

Figure 2: N = 5 clique graph structure.

Attribute columns that are continuous valued (non-binary) are then represented in this logical structure by splitting the representative node into k new nodes (four in this example), each retaining the clique structure with the original (non-replaced) graph nodes, and forming a bipartite group among themselves (Fig.3). The value of k is determined on a per training set basis, and distributed evenly among the available range. In practice, the implementation of a hybrid-quantum workflow would likely begin with data in one hot encoded format, as it was kept for use with classical systems. This means that we will begin by one hot un-encoding (decrypting) the one hot encoded data that we prepared in the Classical Logistic Regresison Model as a starting point, to recover categorical and integer (non-binary) attributes.

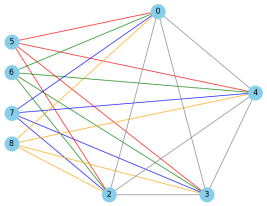

Figure 3: N = 5 clique with one node (Node 1) split into 4 bipartite nodes (Nodes 5, 6, 7, 8), each retaining clique connections with the rest of the original graph structure.

Logically, according to this model, a single datum row from the training set might be represented with its continuous valued attribute at node 5, and then the star clique shown in red in (Fig.4) would represent our problem.

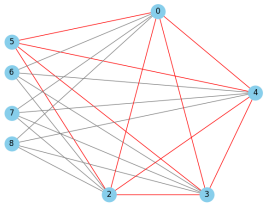

Figure 4: A single row of data might be represented as one of the replacement nodes forming the N = 5 clique with the original nodes.

A different datum row, with a different continuous value (node 6), might be represented as in (Fig.5), where the star clique shown in blue is our problem structure.

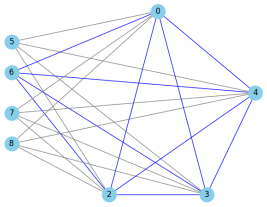

Figure 5: The structure of another row of datum with a different 'Node 1' value might be represented in this manner.

Note that any of the newly created 5-node clique structures is isomorphic to any other:

$$\forall G_j \in G : j \in [1, ..., k]; \qquad G_1 \cong G_2 \cong ... \cong G_k \tag{3}$$

with k equal to the number of nodes in the replacement set (four in this example); in this sense, indicative that they each represent the same topological structure. These mathematical results can be extended to values of N greater than five, but the current maximum inter-qubit connectivity available with the QPU's used here is 20-way connectivity. This, in turn, means that we can have a maximum of $N = 21$ nodes in our logical structure, or 20 attribute columns plus one class column in our training data schema (there are $N = 21$, $k_1 = X$ continuous variables with our specific dataset). This leads to N choose k (different $k_{\text{binary}} = 2$ this time) of:

$$\begin{aligned} C(N,k) &= \frac{N(N-1)}{k} \\ &= \frac{21(21-1)}{2} = 210 \text{ quadratic terms.} \end{aligned} \tag{4}$$

The D-Wave QPU's used in this Project are CC JJ niobium RF-SQUID's (Superconducting Quantum Interference Devices) that operate according to the transverse-field Ising model, to perform adiabatic quantum annealing.

Transverse-field Ising model:

$$H = h\sum_{i=1}^{N}\sigma_i^z + \sum_{i,j+1,i>j}^{N} J_{i,j}\sigma_i^x\sigma_j^x \tag{5}$$

$$| \; h: \text{ transverse-field strength,}$$

$$J_{i,j}: \text{ inter-qubit coupling strength,}$$

$$\sigma_{i,j}^{x,z}: \text{ Pauli spin matrices (computational basis: } \sigma^x)$$

The dynamics of the Hamiltonian will generate the unitary evolution necessary in Eq.(2). The transverse-filed strength h is fed by the values in the training datum, thereby effectively setting the spin of a particular qubit to the datum column value (21 total), while the inter-qubit coupling strength $J_{i,j}$ is set to a constant value for all couplings. The assumption here is that every inter-attribute relationship among the datum (the data is in a state of quantum superposition) is of equal importance in determining a class label. As such, this becomes a hyper-parameter that will require tuning. (Not yet imlplemented in the code; will require a subscription for QPU time.)

A process of minor embedding is used to embed the logical structure of the problem graph G_j onto the specific QPU topology, which for this model is D-Wave's Advantage2 prototype2.6 (Zephyr QPU topology, 1,248 qubits) (Fig6). This embedding is reused for every row in the training data, consecutively, setting the h, $J_{i,j}$ values, and letting the Hamiltonian evolve unitarily ($t_{\text{anneal}} = 20\ \mu$s), and then sampling the (low-energy) spin states of the system (10 samples each for testing mode code, and typically 1,024 samples each for non-test mode code.) The best of the samples is used as the resulting extracted random feature vector, and the next row is processed.

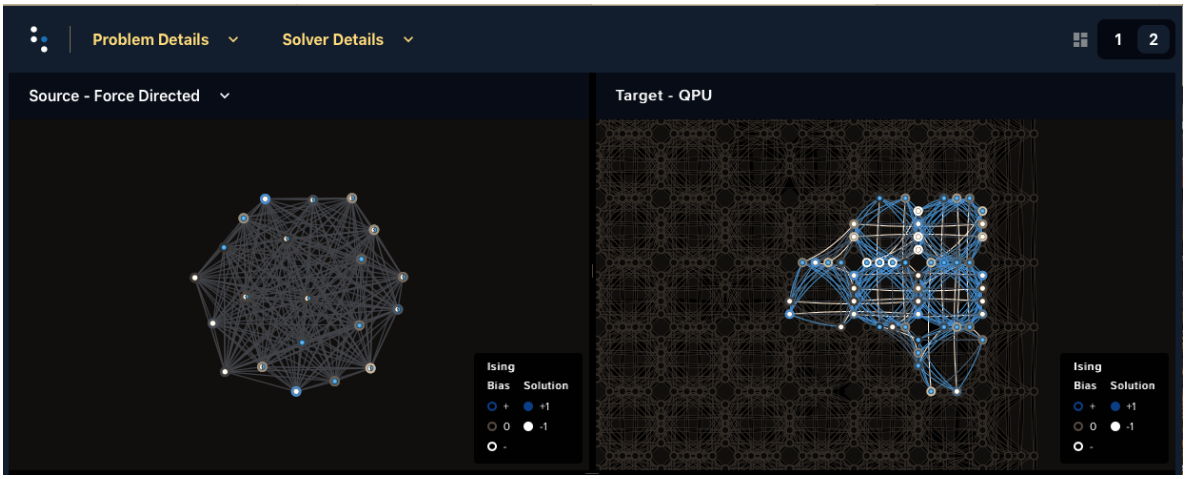

Figure 6: An $N = 21$ graph G_j example, brought up through a local host connection using D-Wave's Problem Inspector (post-processing), with the force directed logical graph topology displayed on the left, and the embedding on the target QPU, Advantage2 prototype2.6 (Zephyr QPU topology, 1,248 qubits) in this case, displayed on the right.

At the end of training, we have a set of random feature vectors, which we use to make predictions. First, in 'CQIM QRC LinearRegression test.py', we use a simple linear regressor as the predictor. Then, in 'CQIM QRC RidgeRegression test.py', we use a repeated 10-fold cross-validated, RidgeCV hyper-parameter tuned linear regressor as the predictor.

3.2.2 Algorithm

Algorithm 2 Continuous Quadratic Ising Model

Require: $1 \leq n \leq 21$ (current max.inter-qubit connectivity = 20)
Ensure: one hot encoded training data

for one hot encoded training data **do**
  decrypt one hot encoded data
  for $\forall$ attribute $\in$ one hot encoded training data **do**
    find attribute categories
    for $\forall$ attribute categories **do**
      find attribute subcategories
      for $\forall$ attribute subcategories **do**
        find data type
        band gap $\leftarrow \frac{1}{(\text{num distinct values} \in \text{attribute})+2}$ ▷ +2: $\nexists$ val $\in$ [0,1]
        if data type is Categorical **then**
          float64(data) $\leftarrow$ float64(data) $\times$ float64(band gap)
        else if data type is Integer **then**
          float64(data) $\leftarrow$ float64(data) $\times$ float64(band gap)
        else if data type is Binary **then**
          float64(data) $\leftarrow$ float64(data)
        end if
      end for
    end for
    return new df col with combined data
  end for
  return decrypted one hot encoded training data
end for
$G_{CQIM} \leftarrow$ logical problem structure (CQIM logic structure)
qpu $\leftarrow$ zephyr topology
embedding $\leftarrow$ minor embedding G_{CQIM} (map logical topology of G to qpu topology)
fixed embedding $\leftarrow$ *embedding*
num samples $\leftarrow$ 1024 (test mode: 10)
if test mode **then** ▷ $\forall$ files ending in 'test'
  training data $\leftarrow$ first 2 rows of training data
end if
for $\forall$ rows $\in$ training data **do**
  $h \leftarrow$ spin: training data values
  $J_{i,j} \leftarrow$ transverse-filed: constant ▷ hard-coded until J hyper-parameter tuning implemented
  $\sigma_{i,j}^{x,z} \leftarrow$ fixed embedding
  spin state preparation:

$$|\psi_{s_k}\rangle = \sqrt{1-s_k}|0\rangle + \sqrt{s_k}|1\rangle \xrightarrow{\text{(non-unitary)}} \rho' = |\psi_{s_k}\rangle\langle\psi_{s_k}| \otimes \mathrm{Tr}^{(1)}\{\rho_{k-1}\}$$

Algorithm 3 Continuous Quadratic Ising Model (continued)

for $\forall$ samples $\in$ num samples **do**
    quantum annealing:

$$H = h\sum_{i=1}^{N}\sigma_i^z + \sum_{i,j+1,i>j}^{N} J_{i,j}\sigma_i^x\sigma_j^x$$

$$\begin{aligned}\rho_k &= e^{-iH\Delta t}\rho' e^{iH\Delta t} \quad \text{(unitary)} \\ &= e^{-iH\Delta t}[|\psi_{s_k}\rangle\langle\psi_{s_k}| \otimes \mathrm{Tr}^{(1)}\{\rho_{k-1}\}]e^{iH\Delta t}\end{aligned}$$

    sample $\leftarrow$ reservoir system spin measurement
end for
end for
random feature vector $\leftarrow$ lowest energy sample
random feature vector list $\leftarrow$ random feature vector
X $\leftarrow$ attr cols $\in$ random feature vector list
y array $\leftarrow$ class col $\in$ random feature vector list
y $\leftarrow$ y array binary conversion
for X, y **do**
    if linear regression **then**
        prediction regressor $\leftarrow$ linear regression
    else if ridge regression **then**
        prediction regressor $\leftarrow$ ridge regression
    end if
end for
for $J \in$ J range **do**
    hyper-parameter tuning ▷ not yet implemented
end for

3.2.3 Code Implementation

```
sudo python3 CQIM_QRC_LinearRegression_test.py
```

```
sudo python3 CQIM_QRC_RidgeRegression_test.py
```

3.3 Binary Quadratic Ising Model

3.3.1 Description

This QRC model, introduced as the "Binary Quadratic Ising Model", begins with the one hot encoded training data, and applied a different problem logic. The entire topology can't be mapped in the same manner as in the CQIM, since there are too many columns present ($N = 1076 > 21$), and the current maximum inter-qubit connectivity is limited at 20. Here we will instead split the original one hot encoded training data into all permutation combinations of pairs attribute columns,

$$\begin{aligned}C_k(N) &= \frac{N!}{(N-k)!k!} \\ &= \frac{(1076-1)!}{((1076-1)-2)!2!} = \frac{1075!}{1073!*2} = \frac{1075*1074}{2} = 5.77275 \text{ x}10^5 \text{ quadratic terms.}\end{aligned} \tag{6}$$

and add the class column to each resulting pair in the combination (Fig.7). The model can be modified to run $m < \left\lfloor \frac{1248 \text{ qubits}}{3} \right\rfloor$ batches of triangle graphs G_j at a time.

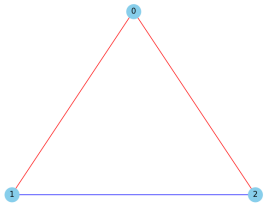

Figure 7: N = 3 clique graph structure $G_{j,m=1}$, where qubits 1 and 2 are every quadratic (non-self) combination of attribute column pairs and qubit 0 is the class column.

Each of the pairwise permutation combinations recombined with the class column is run through the process of setting the transverse-filed strength h to the values in the training datum, using the minor embedding process to embed the logical graph G_j onto the target QPU (again, D-Wave's Advantage2 prototype2.6 (Zephyr QPU topology, 1,248 qubits)), while the inter-qubit coupling strength $J_{i,j}$ is set to a constant value for all couplings. Again, the assumption being that every inter-attribute relationship among the datum is of equal importance in determining a class label, in addition to each inter-attribute relation being of equal importance. Again, this becomes a hyper-parameter that will require tuning, possibly on the triangle graph G_j level (i.e. varied $\forall$ permutation combinations). (Not yet imlplemented in the code; will require a subscription for QPU time.)

The embedding is reused for every attribute pair in every row in the training data, consecutively, setting the h, $J_{i,j}$ values, and letting the Hamiltonian evolve unitarily ($t_{\text{anneal}} = 20$ μs), and then sampling the (low-energy) spin states of the system (10 samples each for testing mode code, and typically 1,024 samples each for non-test mode code.) The best of the each of the samples are combined into a random feature vector, and the next row is processed.

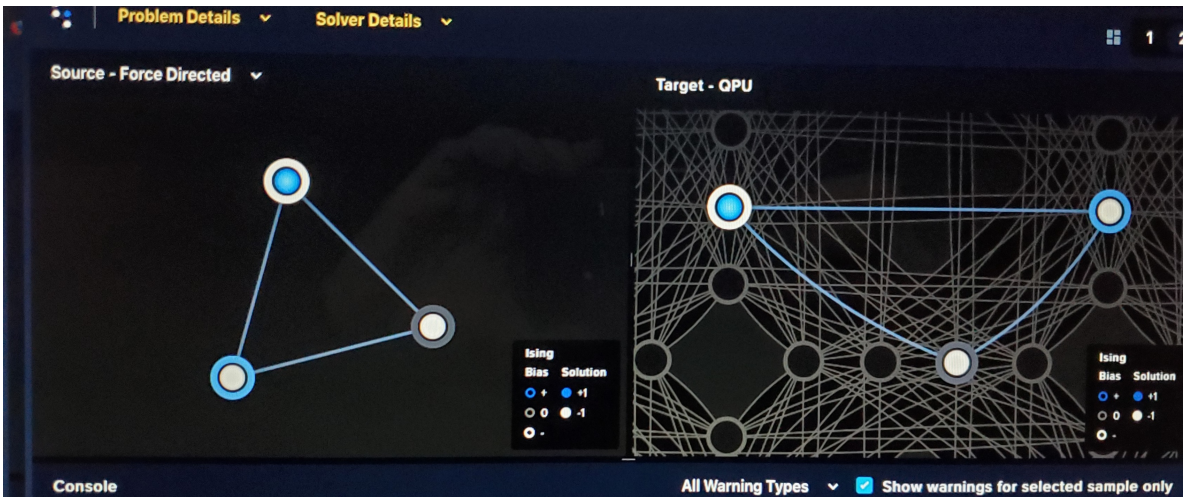

Figure 8: An $N = 3$ graph G_j example, brought up through a local host connection using D-Wave's Problem Inspector (post-processing), with the force directed logical graph topology displayed on the left, and the embedding on the target QPU, Advantage2 prototype2.6 (Zephyr QPU topology, 1,248 qubits) in this case, displayed on the right.

3.3.2 Algorithm

Algorithm 4 Binary Quadratic Ising Model

Require: Enough QPU time to run
Ensure: one hot encoded training data

if test mode **then** ▷ ∀ files ending in 'test'
    training data ← first 2 rows of training data
end if
find pairwise permutation combinations ∀ training data attribute cols
for each pairwise attribute permutation combination **do**
    datum ← pairwise attribute permutation combination + class col
    G_{BQIM} ← logical problem structure (BQIM logic structure)
    $G_{\text{BQIM,m}} \leftarrow \{G_{j,1}, ..., G_{j,m}\} \quad | \quad m \in \mathbb{N}^+, \{1, ..., \lfloor \frac{1248 \text{ qubits}}{3} \rfloor\}$ ▷ not yet implemented
    qpu ← zephyr topology
    embedding ← minor embedding G_{BQIM} (map logical topology of G to qpu topology)
    fixed embedding ← *embedding*
    num samples ← 1024 (test mode: 10)
    for ∀ datum ∈ $G_{BQIM}/G_{\text{BQIM,m}}$ **do**
        h ← spin: training data values
        $J_{i,j}$ ← transverse-filed: constant▷ hard-coded until J hyper-parameter tuning implemented
        $\sigma_{i,j}^{x,z}$ ← fixed embedding
        spin state preparation:

$$|\psi_{s_k}\rangle = \sqrt{1 - s_k}|0\rangle + \sqrt{s_k}|1\rangle \xrightarrow{\text{(non-unitary)}} \rho' = |\psi_{s_k}\rangle \langle\psi_{s_k}| \otimes \text{Tr}^{(1)} \{\rho_{k-1}\}$$

        for ∀ samples ∈ num samples **do**
            quantum annealing:

$$H = h \sum_{i=1}^{N} \sigma_i^z + \sum_{i,j+1,i>j}^{N} J_{i,j} \sigma_i^x \sigma_j^x$$

$$\begin{aligned} \rho_k &= e^{-iH\Delta t} \rho' e^{iH\Delta t} \quad \text{(unitary)} \\ &= e^{-iH\Delta t} [|\psi_{s_k}\rangle \langle\psi_{s_k}| \otimes \text{Tr}^{(1)} \{\rho_{k-1}\}] e^{iH\Delta t} \end{aligned}$$

            sample ← reservoir system spin measurement
        end for
        sample ← separate $G_{\text{BQIM,m}}$ samples ▷ not yet implemented
    end for
    random feature vector piece ← lowest energy sample
    random feature vector piece list ← random feature vector piece
end for

Algorithm 5 Binary Quadratic Ising Model (continued)

random feature vector ← random feature vector piece list
random feature vector list ← random feature vector
X ← attr cols ∈ random feature vector list
y array ← class col ∈ random feature vector list
y ← y array binary conversion
for X, y **do**
    if linear regression **then**
        prediction regressor ← linear regression
    else if ridge regression **then**
        prediction regressor ← ridge regression
    end if
end for
for $J \in$ J range **do**
    hyper-parameter tuning ▷ not yet implemented
end for

3.3.3 Code Implementation

```
sudo python3 BQIM_QRC_LinearRegression_test.py
```

```
sudo python3 BQIM_QRC_RidgeRegression_test.py
```

3.4 Quadratic Unconstrained Binary Optimization (QUBO) Models

3.4.1 Description

3.4.2 Algorithm

3.4.3 Code Implementation

```
sudo python3 CQIM_QRC_QUBO_1_test.py
```

4 Index

List of Algorithms

5 Requirements

The list below is in addition to a D-Wave Leap account with Cloud access.

Python 3.11.1, as well as the list below; information which is also contained in requirements.txt file included in Project.

Package Version ——————- ———
annotated-types 0.7.0
Authlib 1.3.2
blinker 1.9.0
certifi 2024.12.14
cffi 1.17.1
charset-normalizer 3.4.0
click 8.1.7
contourpy 1.3.1
cryptography 44.0.0
cycler 0.12.1
Deprecated 1.2.15
dimod 0.12.14
diskcache 5.6.3
dwave-cloud-client 0.11.3
dwave-greedy 0.3.0
dwave-hybrid 0.6.11
dwave-inspector 0.4.4
dwave-inspectorapp 0.3.1
dwave-neal 0.6.0
dwave-networkx 0.8.14
dwave-ocean-sdk 6.9.0
dwave-preprocessing 0.6.5
dwave-samplers 1.2.0
dwave-system 1.23.0
dwave-tabu 0.5.0
dwavebinarycsp 0.3.0
fasteners 0.19
Flask 3.1.0
fonttools 4.55.3
homebase 1.0.1
idna 3.10
imageio 2.36.1
importlib metadata 8.5.0
itsdangerous 2.2.0
Jinja2 3.1.4
joblib 1.4.2
kiwisolver 1.4.7
lazy loader 0.4
MarkupSafe 3.0.2
matplotlib 3.10.0
minorminer 0.2.13
networkx 3.4.2
numpy 1.26.4
packaging 24.2

6 Bibliography

[1] "Dissipation as a resource for quantum reservoir computing" YouTube, uploaded by QAISG, May 7, 2024, https://youtu.be/hxvTRJDj6ok?si=qD6N98Q8xB3UdU90