National College of Ireland

Configuration Manual

MSc Research Project
Cyber Security

Saud Shaikh
Student ID: 23276509

School of Computing
National College of Ireland

Supervisor: Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Saud Suhail Shiakh		
Student ID:	23276509		
Programme:	MSc Cyber Security	**Year:**	2024-25
Module:	Research Project		
Lecturer:	Imran Khan		
Submission Due Date:	11th August 2025		
Project Title:	Hybrid Post Quantum Cryptography: Benchmarking and Machine Learning Based Optimization		
Word Count: 1789	**Page Count:** 11		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Saud Suhail Shaikh
Date:	11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Saud Shaikh
Student ID: 23276509

1 Introduction

This configuration manual is an instruction set on how to replicate, set-up and run the Hybrid Post-Quantum Cryptography study authored as part of the Research Practicum MSc Cybersecurity research project at the National College of Ireland.

This project will compare and optimize both implementation and benchmarking of quantumresistant cryptographic algorithms (particularly, Kyber, McEliece, and RSA), with a combined encryption scheme. Additional performance optimisation is achieved through the inclusion of an AI/ML-based workload predictor to analyse how much CPU resources to use when generating encryption keys. The system supports encryption, decryption, benchmarking, comparing algorithms, and visualisation of files, in addition to thus being user friendly due to a graphical user interface (GUI).

This guide will help you with the overall assembly process of the project. This guide contains all required steps and illustrations for assembly.

- Scholars who were imitating the application
- Developers adding to PQC systems
- Researchers studying the use of post-quantum cryptography and AI

It discusses everything on getting started, setting up the environment, libraries and all the way up to execution of a hand and GUI deployment. Each section is accompanied with screenshots so that the same can be replicated to help them avoid ambiguity when carrying out configurations.

2 System Requirements

It discusses all the steps of the initial set up, the environment configuration, library installation, complete project implementation and GUI deployment. Each section is supported by screenshots so that there would be the possibility to reproduce the same in case of ambiguity in configuration. In this section, the hardware and software specifications necessary to construct and execute the project of Hybrid Post-Quantum Cryptography are recommended. The entire project has been tested and developed on a local windows computer with Visual Studio compiler to write C++ and Python to write ML and graphical user interface raw materials.

2.1 Host Machine Specification:

Table 1: Host Machine Specification

Component	Specification
Operating System	Windows 11 (64-bit)
Processor	Intel Core i7 / AMD Ryzen 7 or higher
RAM	16 GB (minimum), 32 GB (recommended)
Storage	100 GB free disk space
Python Version	Python 3.11 or later
Visual Studio	Visual Studio 2022 (C++ & CMake)
Git	Git for Windows (latest version)

2.2 Project Dependencies Overview:

The project is written in C++ and Python, and features incorporating open-source postquantum and AI/ML libraries used to optimise the workload.

Table 2: Host Machine Specification

Tool/Library	Purpose
liboqs	Open Quantum Safe C library for PQC algorithms
OpenSSL (OQS Fork)	PQC-integrated OpenSSL fork
CMake	Cross-platform build system for liboqs/OpenSSL
PyQt5	GUI development in Python
Matplotlib, Seaborn	Visualisation of benchmarking results
scikit-learn, joblib	ML model training and prediction

2.3 Structure of Directory:

```
PQC_GUI_PROJECT/
├── benchmarking optimization/
│   └── pqc.ipynb
│
├── gui/
│   └── gui.py
│
├── liboqs-python/
│   └── (Cloned liboqs repo with wrappers and bindings)
│
├── payloads/
│   └── (Auto-generated test files of various sizes)
│
├── pqc/
│   ├── kyber.py
│   ├── mceliece.py
│   ├── rsa.py
│   └── utils.py
│
├── results/
│   ├── auto_benchmark.csv
│   ├── benchmark_sorted_final.csv
│   └── gui_benchmark_results.csv
│
├── venv/
│   └── (Virtual environment files)
│
├── generate_payload_file.py
├── main.py
└── requirements.txt
```

Figure 1: Report Structure

Figure 1 shows the directory structure view of the hybrid post-quantum cryptography benchmarking and optimization project in Figure 1 displays the whole set of files organized in the form of a tree. It enhances all important folders and files consisting of benchmarking scripts, GUI code, post-quantum algorithm implementation, payload files of test and requirements to configure the environment.

3 Tools and Dependencies installation:

This section entails step wise instructions into installing as well as setting up all the required tools, libraries and environments mandatory to build, run and test the Hybrid Post-Quantum Cryptography with AIML Optimisation project.

It covers installations for:

- Visual Studio (for C/C++ development)

- CMake (for building liboqs/OpenSSL)
- Python and dependencies (for benchmarking, ML, and GUI)
- liboqs and OpenSSL setup (for Kyber and McEliece integration)
- Additional optional tools

3.1 Install Visual Studio 2022

1. Download: https://visualstudio.microsoft.com/downloads/
2. Select the following workloads:
   - Desktop Development with C++
   - C++ CMake Tools for Windows

3. Ensure the following components are included:
   - MSVC v143 - VS 2022 C++ x64/x86 build tools
   - Windows 10 SDK (10.0.19041.0 or later)
   - C++ CMake tools

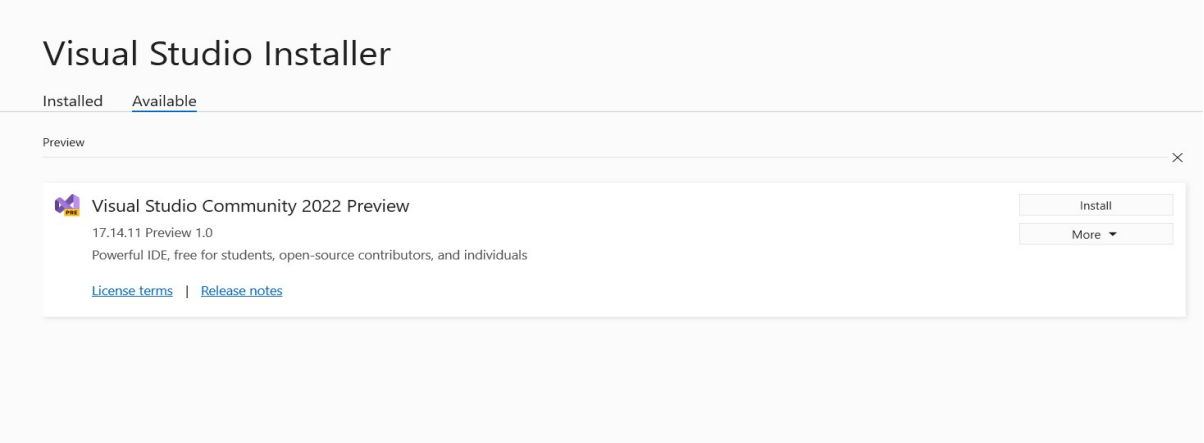

Figure 2: Visual Studio Installer

Figure 2 shows the Visual Studio Installer on the page Available, where it is possible to choose the option of installing the program named Visual Studio Community 2022 Preview (version 17.14.11 Preview 1.0), necessary to configure the project in the C / C ++ developer language.

3.2 Install CMake

1. Download: https://cmake.org/download/
2. During installation:
   Select “Add CMake to the system PATH for all users”

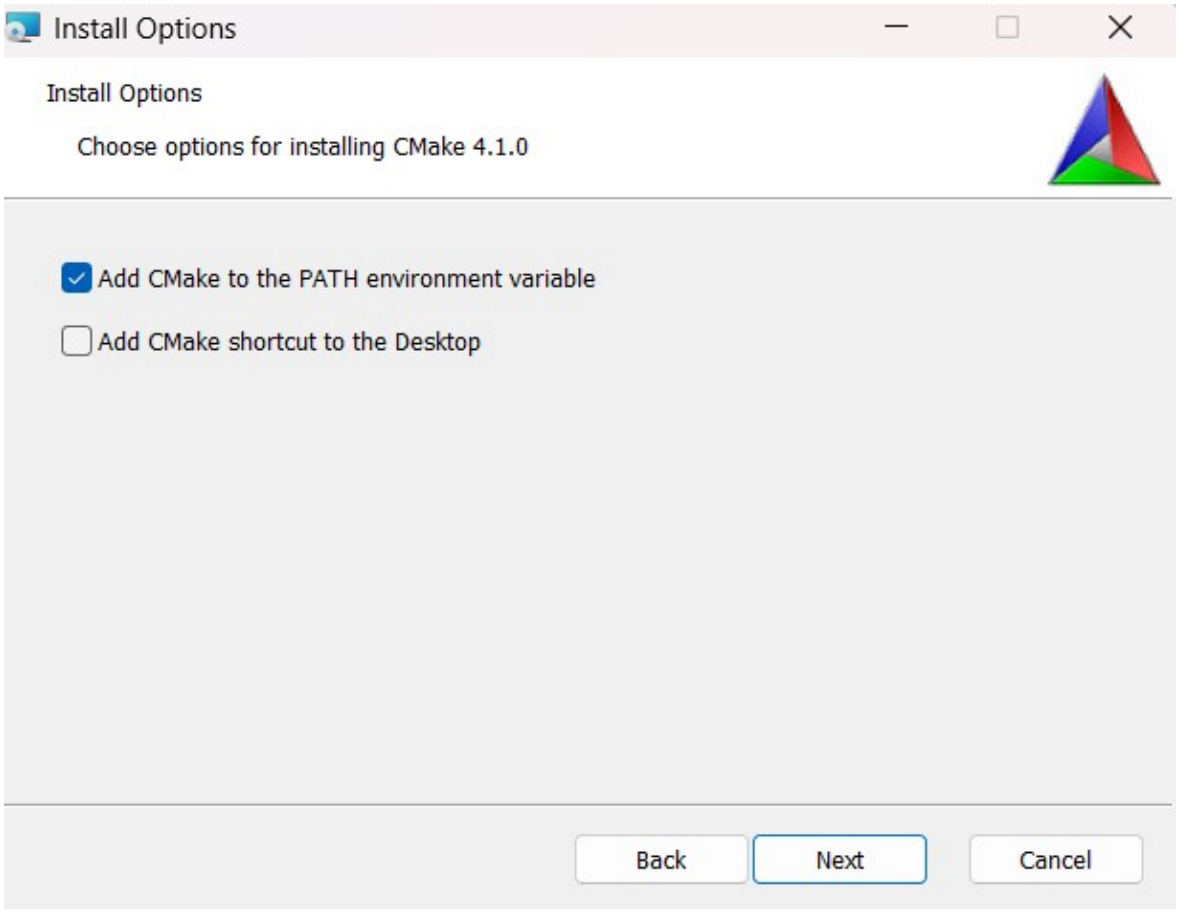

Figure 3: Install options

Figure 3 shows the CMake installer installation, the user chose to add CMake to the system PATH, which makes it much easy to access using the command line to build system C/C++ dependency liboqs and OQS-OpenSSL.

3.3 Install Git

Download: https://git-scm.com/downloads

Install with default options.

Confirm installation by running in PowerShell:

Figure 4: Windows PowerShell

Figure 4 shows a PowerShell run confirmation that Git is installed and accessible successfully, which is necessary in order to clone necessary Git required repositories to be able to work on the project.

3.4 Clone liboqs and OQS-OpenSSL

In the project folder run the following command:

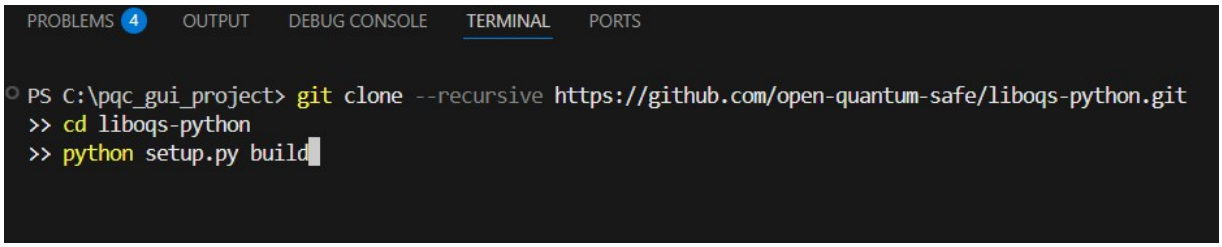

Figure 5: Clone liboqs and OQS-OpenSSL

Figure 5 shows a series of actions of copying the liboqs-python repo and building it with Python and showing quantum-safe cryptography binding setup steps.

The wrapper uses prebuilt liboqs binaries and allows calling Kyber and McEliece from Python.

3.5 Install Python 3.11+ and Create Virtual Environment

1. Download Python: https://www.python.org/downloads/
2. During install:
   - Check **"Add Python to PATH"**
   - Enable pip
3. Create and activate a virtual environment using following command:

Figure 6: Install Python 3.11+ and Create Virtual Environment

Figure 6 shows activating a Python virtual environment in the project directory which is very important to make the project independent of any programming environment and allow a reproducible build.

3.6 Install Python Dependencies:

Run command: pip install -r requirements.txt

The text file contains the following

requirements.txt

```
contourpy==1.3.2
cycler==0.12.1
fonttools==4.59.0
kiwisolver==1.4.8
matplotlib==3.10.3
numpy==2.3.1
oqs==0.10.2
packaging==25.0
pandas==2.3.1
pillow==11.3.0
psutil==7.0.0
pycryptodome==3.23.0
pyparsing==3.2.3
python-dateutil==2.9.0.post0
pytz==2025.2
scipy==1.16.0
six==1.17.0
tzdata==2025.2
xgboost==3.0.2
```

Figure 7: requirements.txt

Figure 7 shows Python dependencies necessary to create PQC GUI project with specific versions to make the project reproducible and compatible in a virtual environment.

4 Initial Setup and Configuration

This part describes how to configure project environment, test all modules and configure system to perform encryptions, benchmarking components and machine learning components. It contains the steps to initialization of payload generation, the use of the GUI and integration of the algorithms through liboqs.

The first Step is to activate the python environment using command: .\venv\scrips\activate

```
PS C:\pqc_gui_project> .\venv\scripts\activate
(venv) PS C:\pqc_gui_project>
```

Figure 8: PowerShell -venv activation

Figure 8 shows how a Python virtual environment, project dependencies will be encapsulated, and are not a part of the global Python install.

Then generate the payload for benchmarking using the python script:

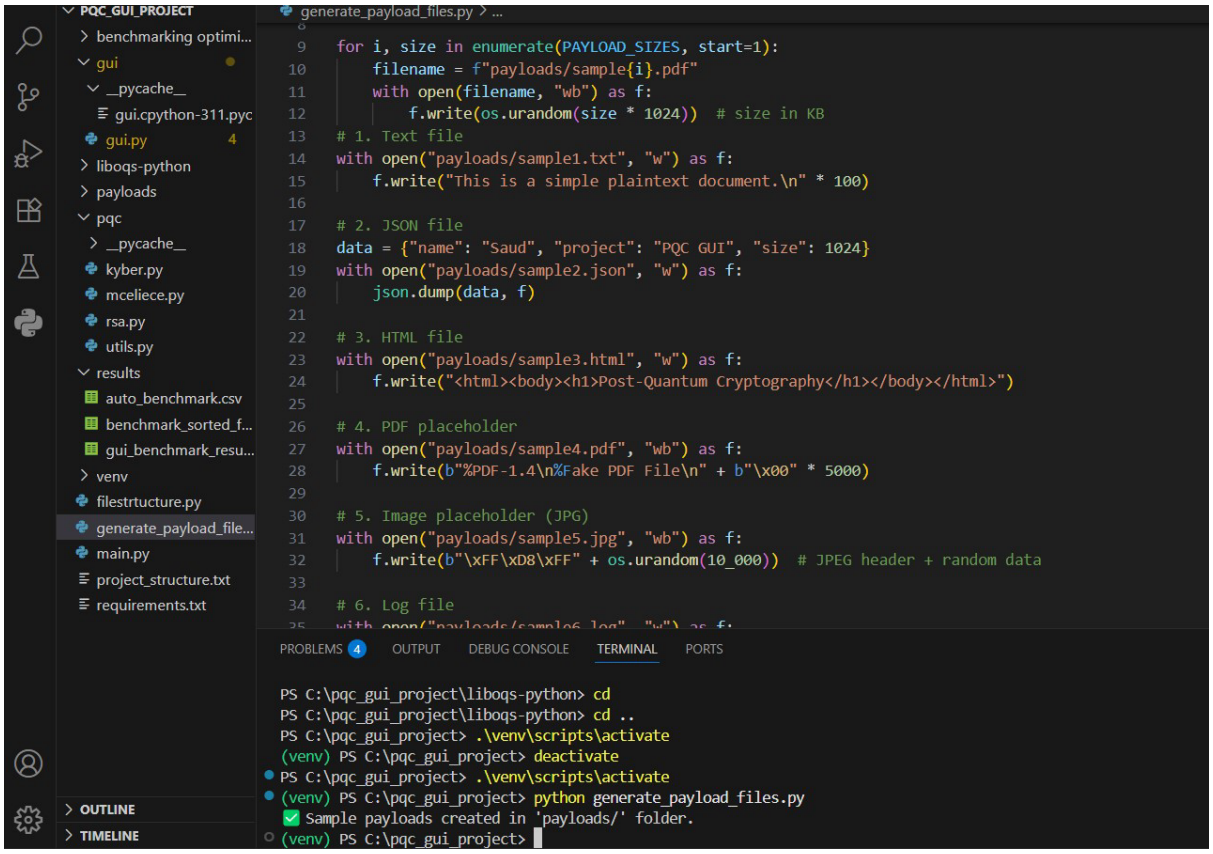

Figure 9: VS Code -run script and generate payload

Figure 9 shows how to launch a script in the active virtual environment of the VS Code that creates sample benchmarking and testing crypto algorithm payload numbers and files.

Then run the GUI application using the main script

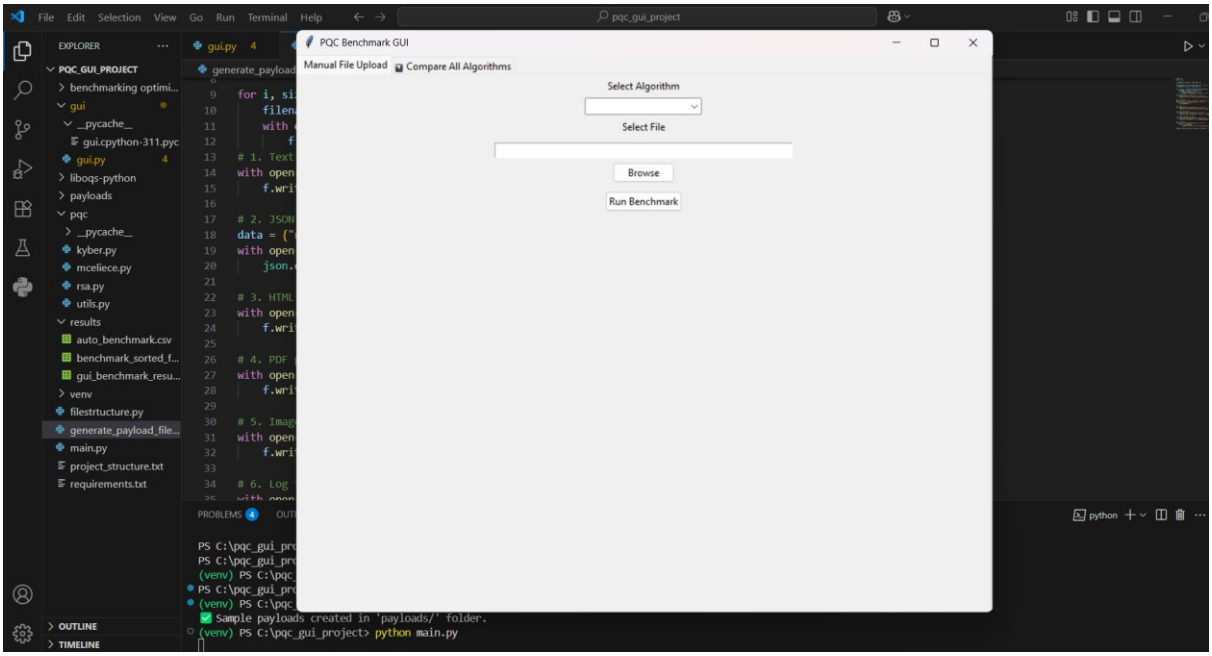

Figure 10: PQC Benchmark GUI in action

Figure 10 shows the PQC Benchmark GUI application running, in which users may choose algorithms, upload files, and run benchmarking of operations of post-quantum cryptography.

The GUI includes:

File upload for encryption/decryption
Algorithm selection: Kyber, McEliece, RSA
Benchmark visualisation (matplotlib/seaborn)
AIML-predicted CPU usage graph

5 Initial Setup and Configuration:

This section describes the end-to-end execution of the project and outlines the output generated at each stage. The project supports interactive encryption and benchmarking through a graphical interface, as well as automated analysis through CLI scripts and Jupyter Notebooks.

5.1 GUI execution

Run **main.py** in vs studio to launch the gui interface:

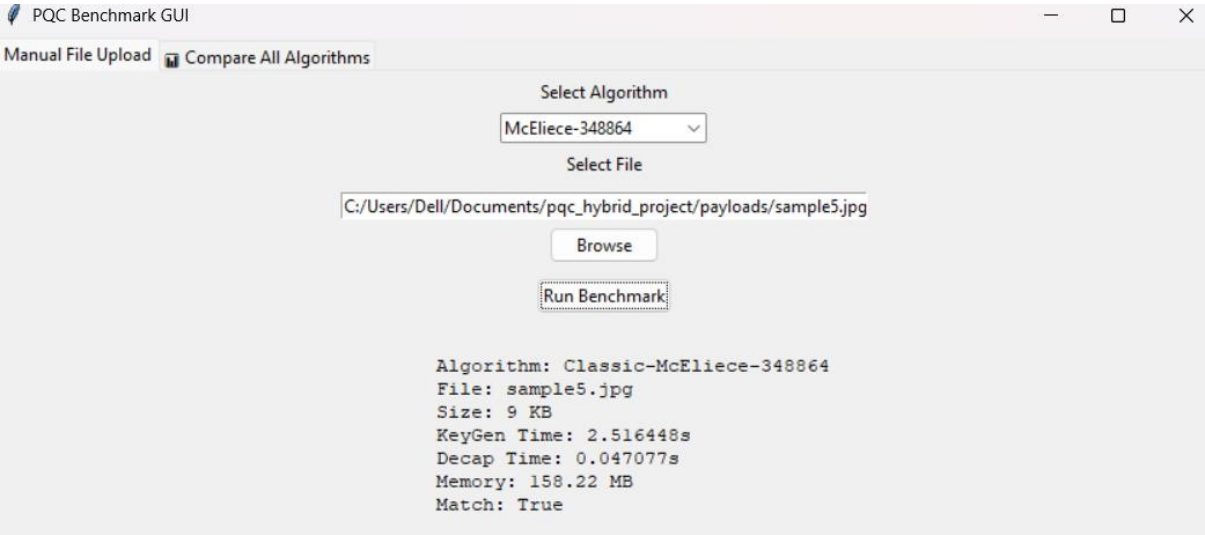

Figure 11 GUI execution

Figure 11 shows the PQC Benchmark GUI living, showing benchmarking results of the Classic-McEliece-348864 algorithm on an uploaded file, with such statistics as key generation time, decapsulation time, memory usage and correctness.

Table 3: Features of GUI

Feature	**Description**
File Upload	Selects a file for encryption/decryption
Algorithm Selection	Choose from **Kyber**, **McEliece**, or **RSA**
Encrypt/Decrypt Button	Performs hybrid encryption/decryption using OpenSSL/liboqs bindings
Benchmark Tab	Benchmarks all three algorithms on selected or auto-generated payloads
Feature	**Description**
CPU Usage Prediction (AIML)	Predicts expected CPU usage using a trained ML model
Result Visualization	Bar/line plots showing keygen time, encryption time, memory usage, etc.

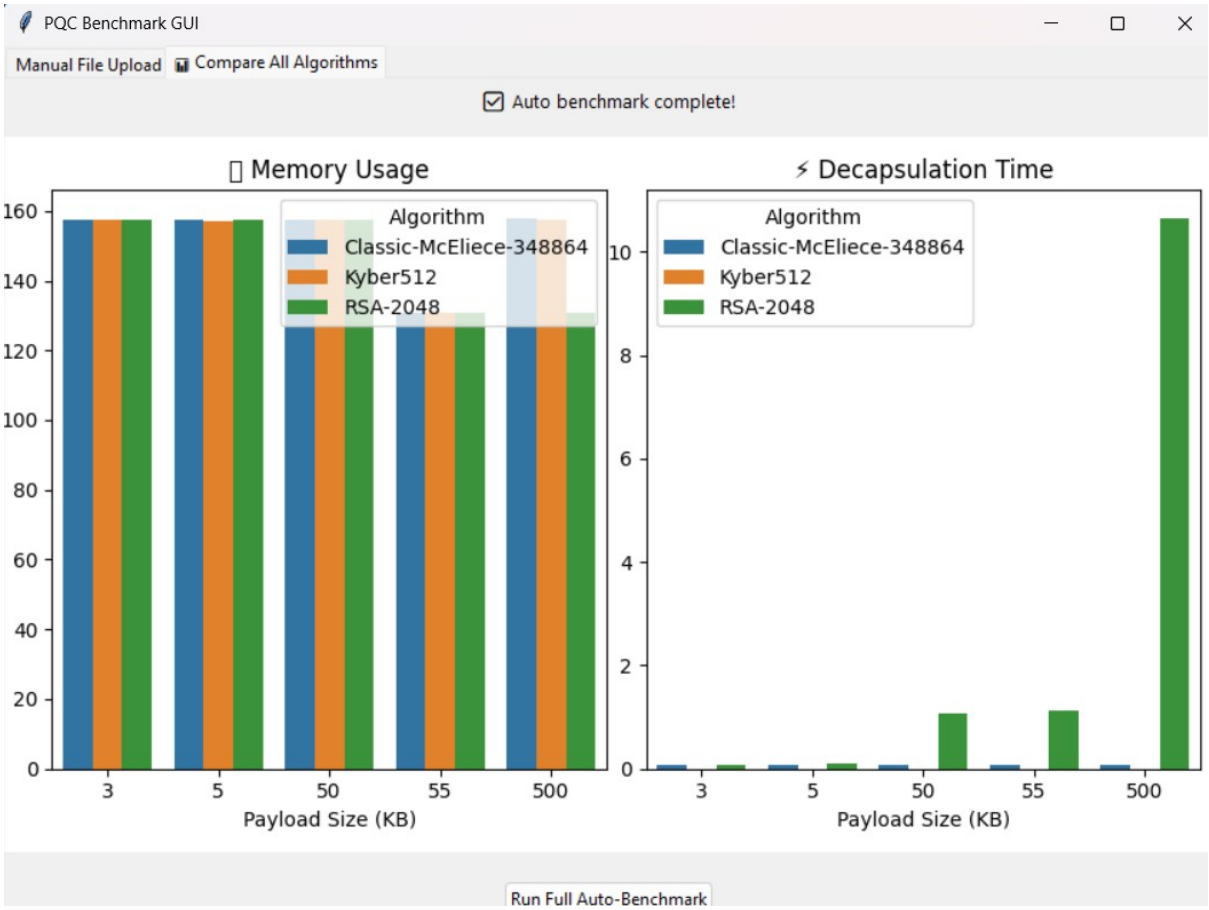

Figure 12: GUI execution

Figure 12 shows comparison bar charts using PQC Benchmark GUI, which show memory of different cryptographic algorithms (Classic-McEliece, Kyber512, RSA-2048) over a range of payloads and the memory usage as well as decapsulation of Classic-McEliece, Kyber512, RSA-2048.

5.2 Output CSVs and logs:

Table 4: Output and logs

File Name	Description
auto_benchmark.csv	Auto-generated benchmark results across all algorithms and payloads
benchmark_sorted_final.csv	Sorted/processed benchmark data for visualisation
gui_benchmark_results.csv	Benchmark results captured from GUI interactions
model_predictions.csv (optional)	Predicted CPU usage for each payload/algorithm combination

Each file includes metrics such as:

- Algorithm name
- Payload size (KB)
- Key generation time (ms)

- Encryption time (ms)
- Decryption time (ms)
- Memory/CPU usage

5.3 AIML Notebook Execution:

Table 5: AIML Notebook components

Section	Description
Data Generation	Creates synthetic benchmark data (if needed)
Data Preprocessing	Normalisation, cleaning, and splitting
Model Training	Trains a model (Random Forest / DNN) to predict CPU usage
Prediction & Evaluation	Compares predicted vs actual CPU usage using graphs
Export Model	Saves model as model.pkl for reuse in GUI

By using the CSV data that we generated we can then train our AI Optimization model and visualize our results using Jupiter notebook on google colab

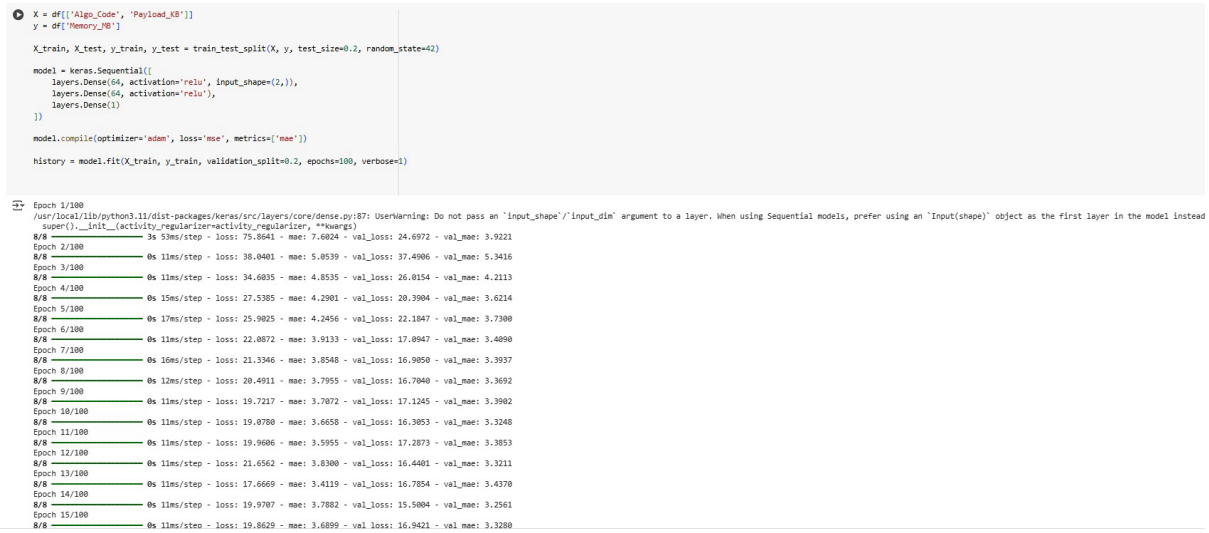

Figure 13: AIML Notebook Execution

Figure 13 shows a machine learning model training process in a Jupyter notebook, during which a neural networks are fitted to benchmarking data in order to predict memory requirements as a function of algorithm and payload size.

Figure 14: PQC metrics

Figure 14 shows PQC metrics in pairwise comparison plot with Seaborn to be visualized showing the interdependence of payload size, key generation time, encapsulation/decapsulation time, and the memory consumption of various algorithms.

5.4 Final Output and summary:

Table 6: Output Structure

Module	Output Format	File/Location
GUI	Visual + File	GUI screen + results/ directory
Benchmarking	CSV	auto_benchmark.csv, etc.
AIML Prediction	Visual + Model File	model.pkl, model_predictions.csv
Encryption Result	Encrypted file + log	User-specified output location
Decryption Result	Restored plaintext file	Same as input file (or renamed)

6 Conclusion

In this configuration guide, the entire composition and implementation procedure of the Hybrid Post-Quantum Cryptography: Benchmarking and Machine Learning-Based Optimization project has been illustrated. All the way through preparation of the initial environment, GUI-based interactions, and AIML-based prediction, sets are modular, reproducible and can be easily expanded in the future.

Table 7: Tools and Libraries used

Component	**Tools/Libraries Used**
Cryptography	liboqs, OQS-OpenSSL, Python bindings
ML Optimisation	scikit-learn, joblib, pandas, Jupyter Notebook
Benchmarking	Python, CSV logging, Seaborn/Matplotlib
GUI Interface	PyQt5
Build Environment	Visual Studio 2022, CMake, Git
Automation	Python scripts, auto-payload generation

The project shows the possibility of combining quantum-resilient encryption with real-time benchmarking and AI-powered insights into deployment on systems that see performance as a policy concern. The crystallographic, or ML models, can be easily swapped or upgraded with the modular architecture. The results can be exported as well as reproduced and they can be visualised via the GUI or notebook.