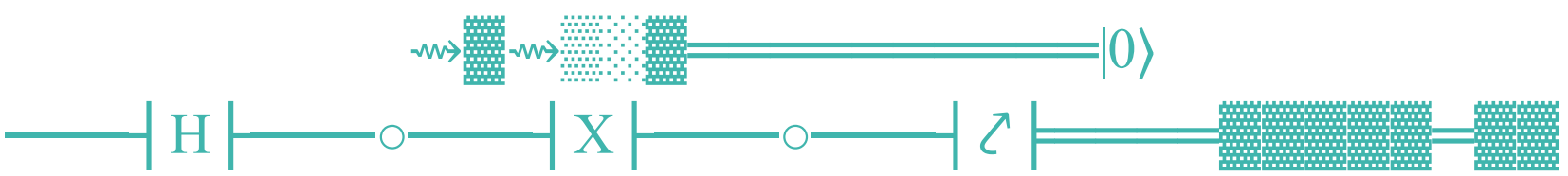
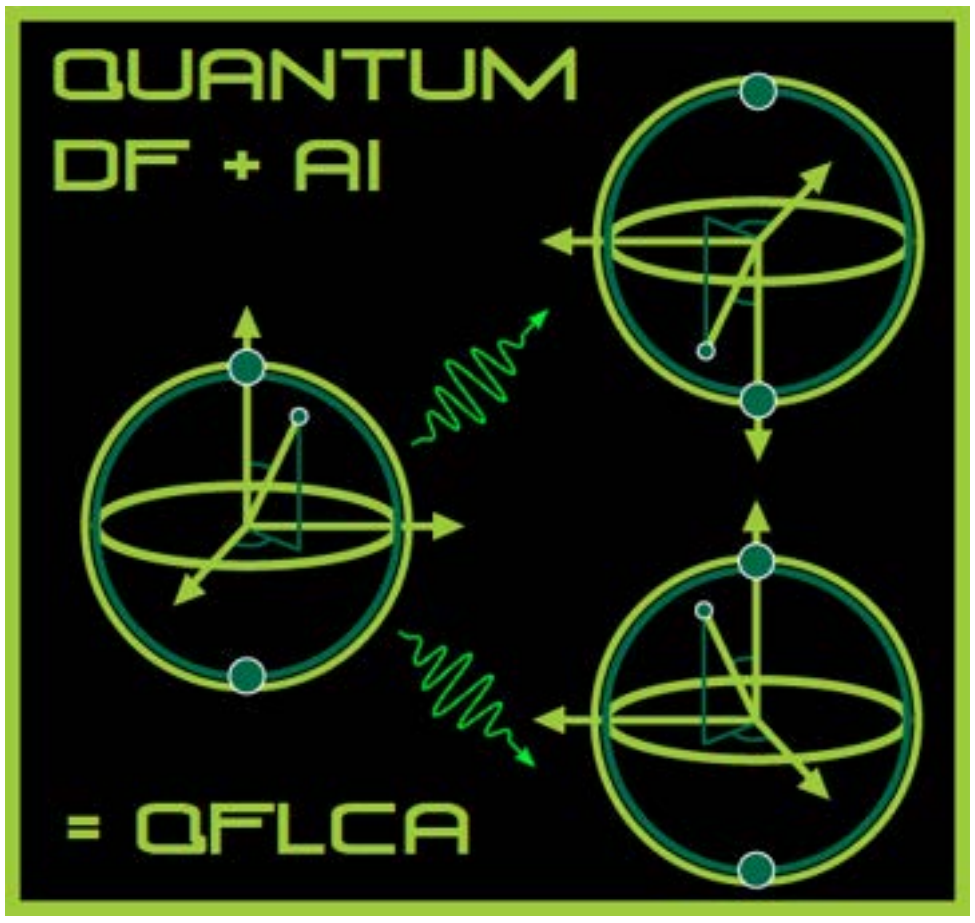
QFLCA Documentation Website

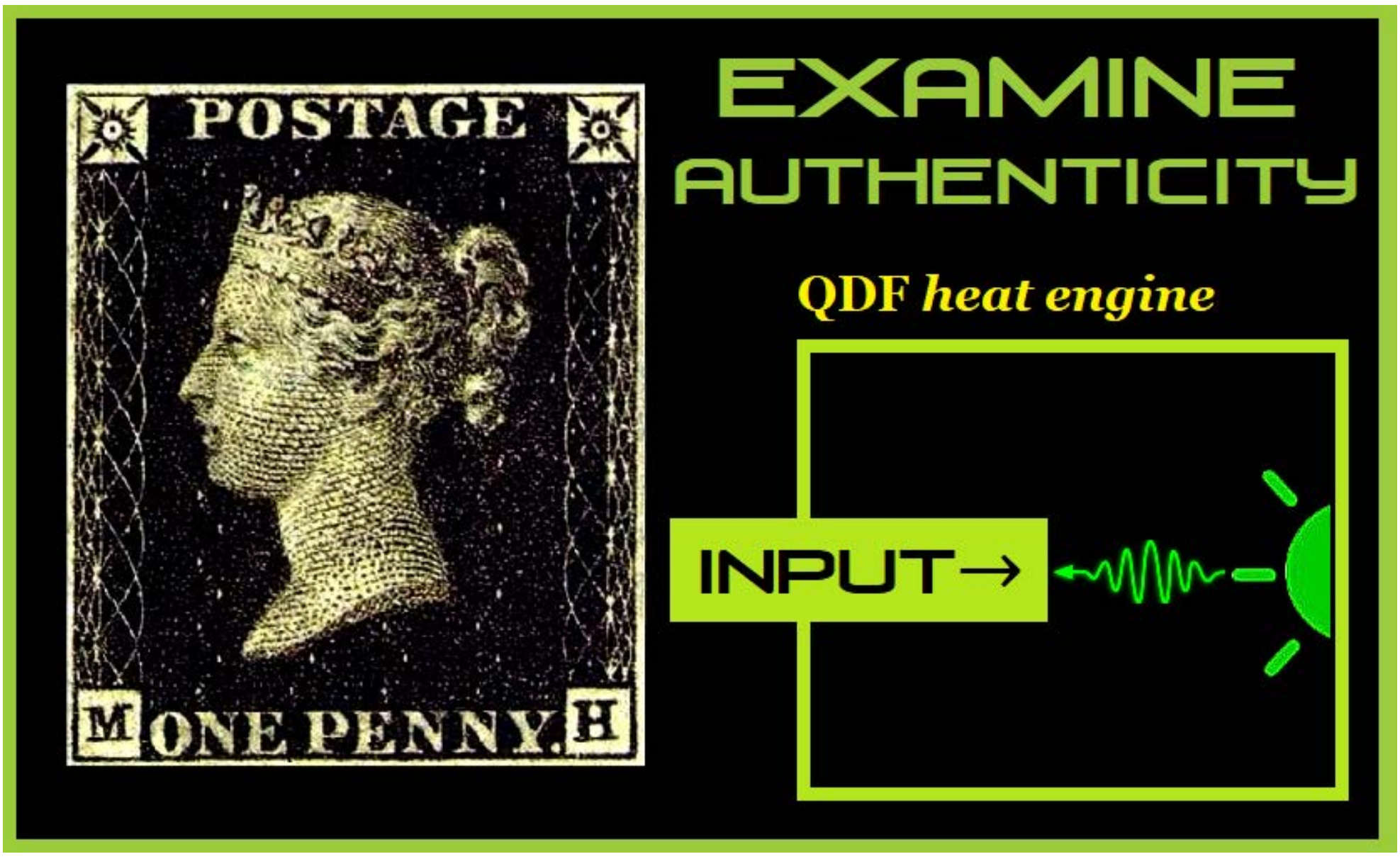
Documentation for
1. QFLCA Dataset: Quantum AI Lens Coding and Classification (QAI-LCode_QFLCC) Docs
2. QDF Game: Alice & Bob's Quantum Doubles

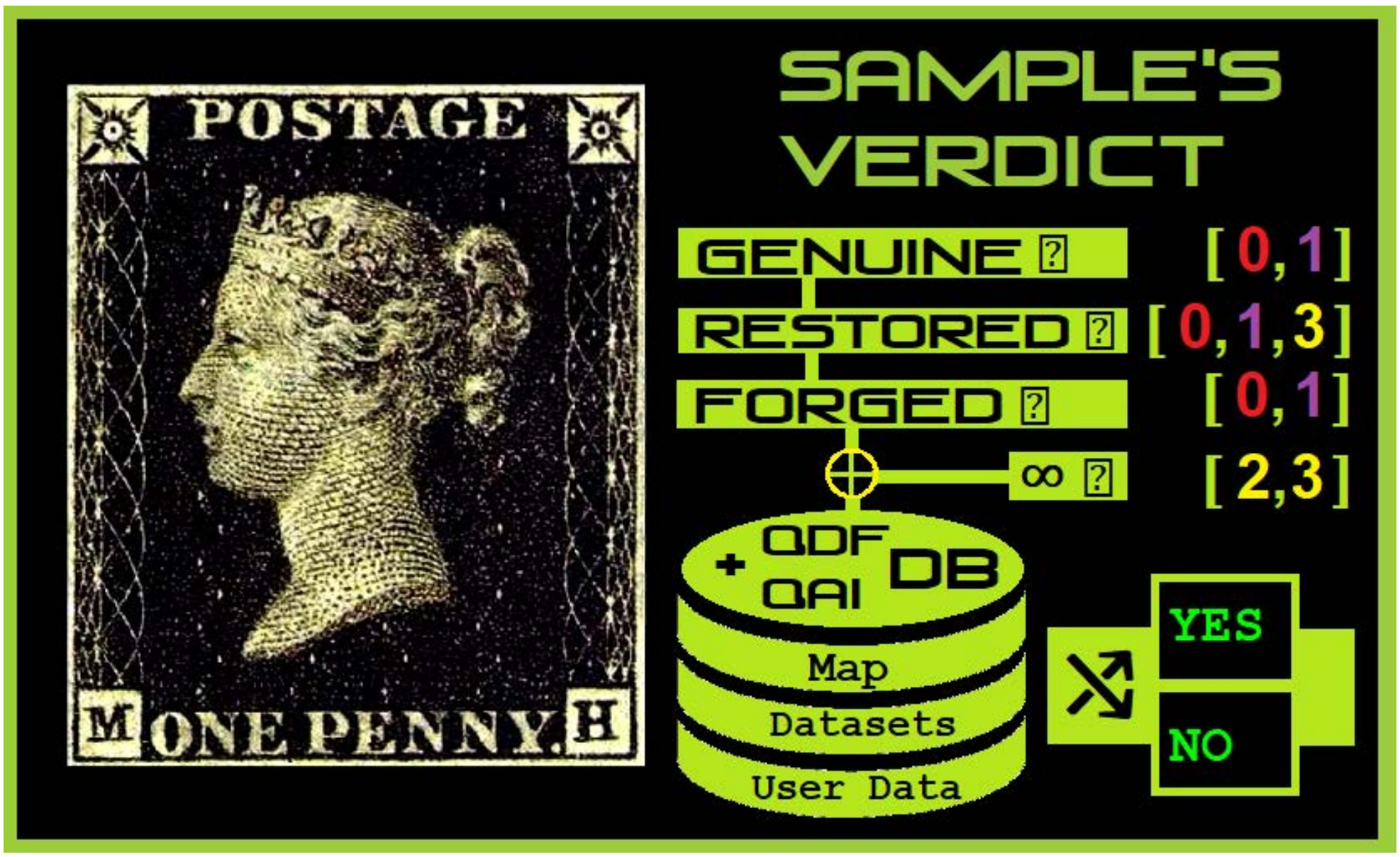




This website presents the documentation of quantum field lens coding algorithm (QFLCA) project objectives, its progress and development, codes, and updates.

I. |κΦ| Home


POSTAGE
M ONE PENNY. H
EXAMINE AUTHENTICITY
QDF heat engine
INPUT →


POSTAGE
M ONE PENNY. H
SAMPLE'S VERDICT
GENUINE ? [0,1]
RESTORED ? [0,1,3]
FORGED ? [0,1]
∞ ? [2,3]
+ QDF QAI DB
Map
Datasets
User Data
YES
NO

1 Welcome to QFLCA's Program Documentation

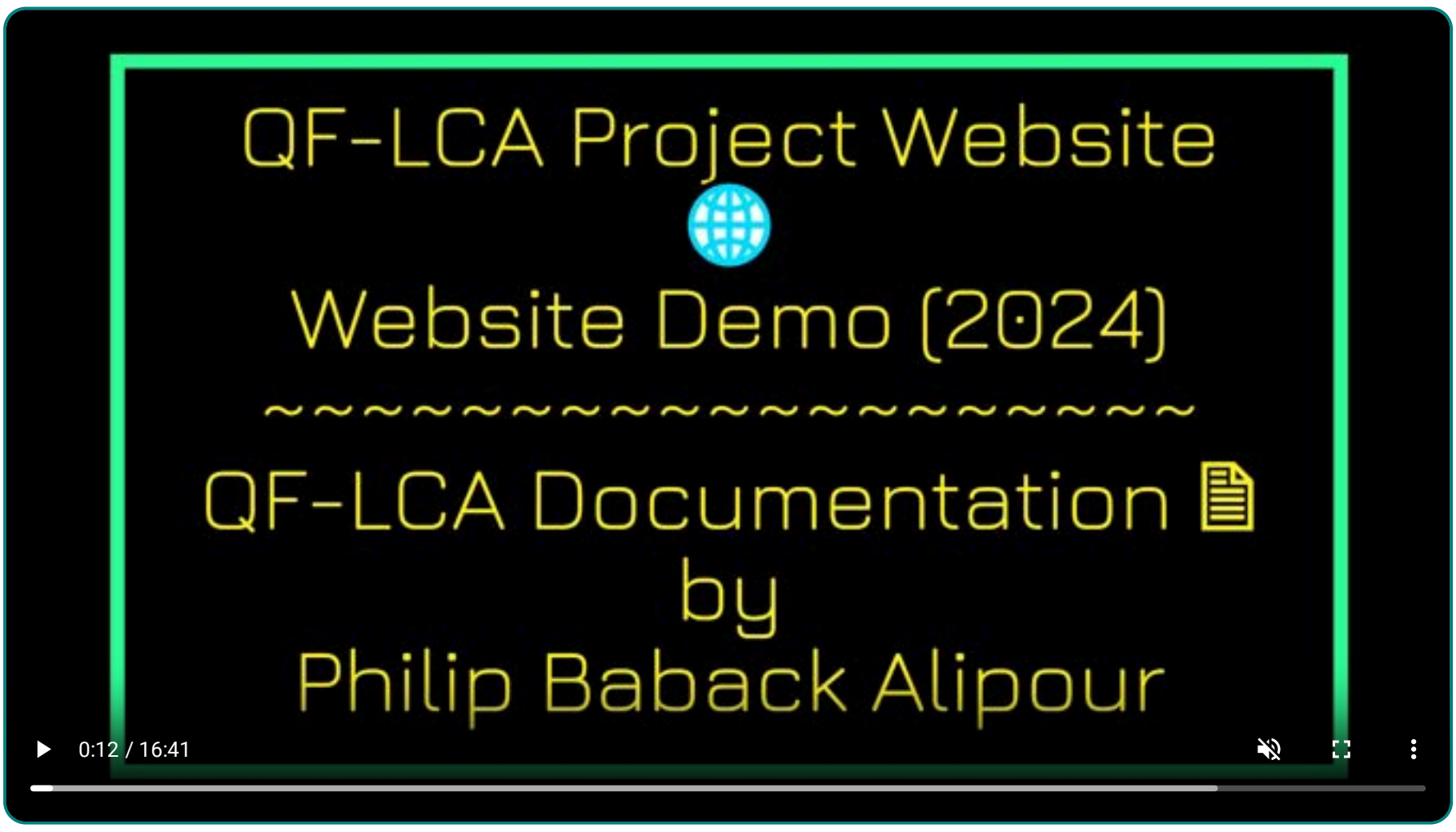


Figure 1: QFLCA Intro Demo: 788 MB mp4 file. ∇ | For Subtitles: SRT file. ∇

1.1 QFLCA Website Structure: site/index.html

{{ pagetree }}

1.2 Introduction

This website introduces the `QFLCA project`, its `file structure`, `program objectives`, `installation` and `code run` based on textual, imagery, video and other types of resources to view, analyze, evaluate and validate project's datasets addressing its objectives through documentation.
For a full view, if using web browsers like e.g., `Firefox`, `Microsoft Edge`, `Chrome`, `press F11` on your keyboard. By pressing the same function, the view reverts back to your previous browser settings.

For the `QFLCA's program documentation`, McDocs was installed in Visual Source Code (VSC) environment and used to build this website, so to present the `QFLCA project`. For more information, visit the VSC "installation" page . The "About" page presents project goal and objectives, as well as progress and expected outcomes, publications, authorship, usage, and structure.
For other related webpages introducing `project's workflow`, `datasets`, their `validation methods` e.g., via a quantum game, browse this website to view e.g., project demos and codes. For the full MkDocs documentation, visit mkdocs.org.

1.3 Commands

- `mkdocs new [dir-name]` - Create a new project.
- `mkdocs serve` - Start the live-reloading docs server.
- `mkdocs build` - Build the documentation site.
- `mkdocs -h` - Print help message and exit.

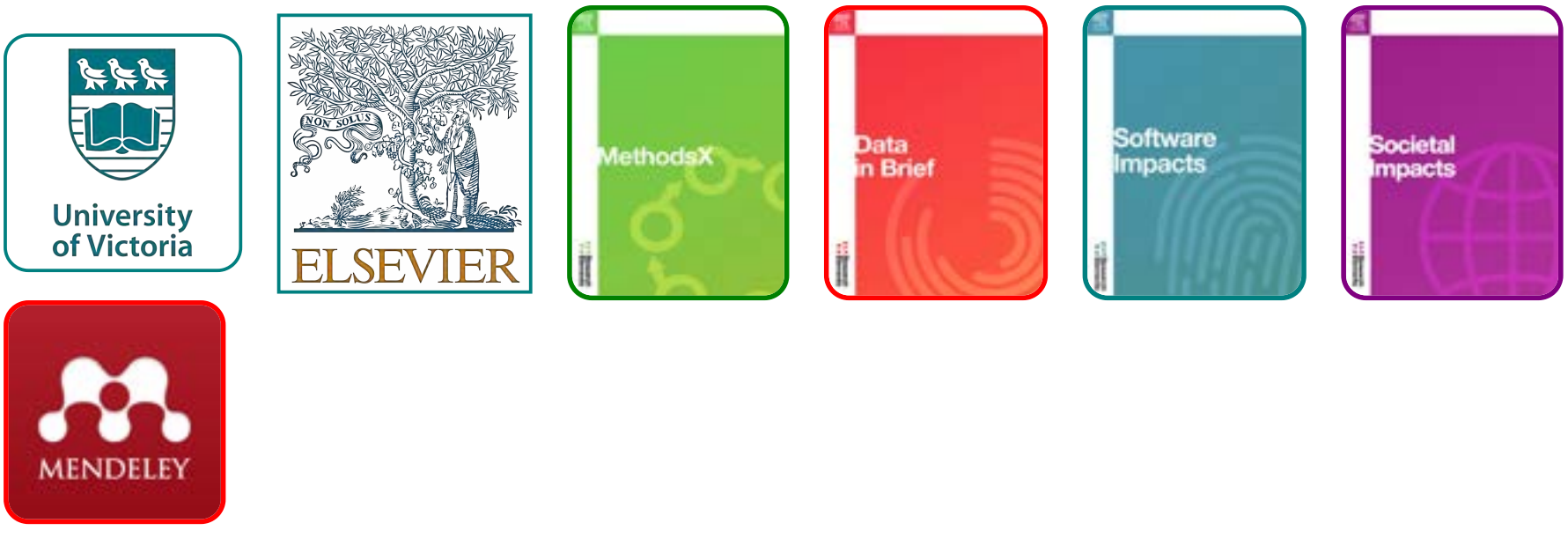
1.4 Project layout

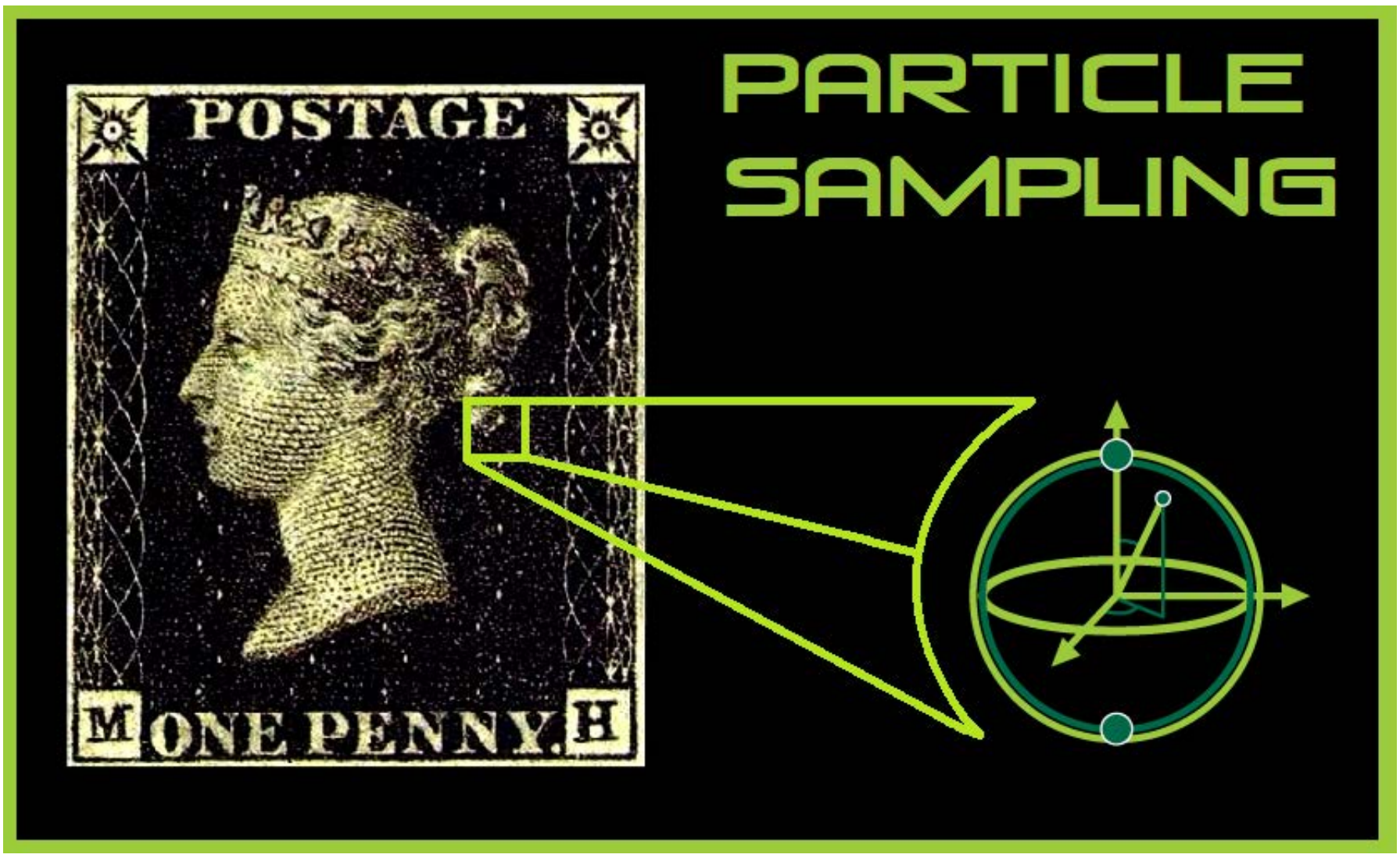
```
QFLCC classifiers/   # *.py home dir, yet a sub-dir of <QFLCA classifiers> in future versions.
  mkdocs.yml    # The configuration file for documentation via <docs> content.
      *.py      # The QFLCC python file(s).
      ...       # The QFLCC python file(s) and <docs> content dependencies.
  docs/
      index.md  # The documentation homepage.
      ...       # Other Markdown pages, stylesheets, images, and their dependencies.
```

1.5 Keyboard shortcuts

```
help: ?       # h
next: →       # n
previous: ←   # p
search:       # ?
```

Project content and website affiliations:





2 About the QFLCA Project

2.1

Project Overview

For a similar QDF measurement example, see Fig. 16 on this page

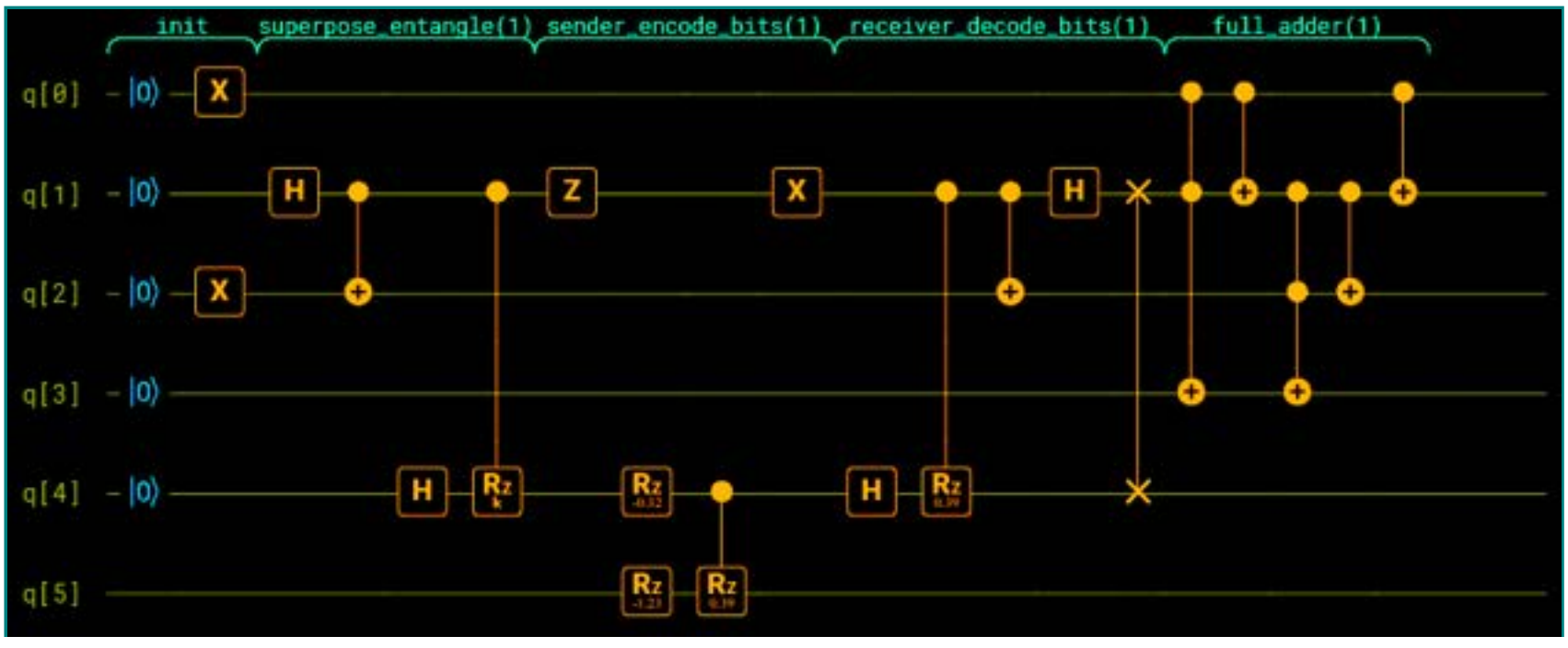
---

The QFLCA project is presented as follows:

- QFLCA as Quantum AI for System Simulation.
- QFLCA as an intelligent decision support simulator/solution (IDSS) for users active in industry 4.0 systems, law, forensics, security systems, medicine, physics, engineering, computing, and other fields of science.
- QFLCA can be used to detect system states and make strong state predictions through quantum double-field analysis. That is, using quantum scalar field technique to double the probability ($\mathcal{P}$) of each system state in its state transition from one fom to another.
- Uses quantum superposition and entanglement, predicts and suggests new energy paths for the user to take and make the system more efficient in its performance and evolution involving particles on any scale.
- Among other features of the QFLCA project is making the environment more sustainable on quantum and macro levels such as climate, human body (medical treatments such as cancer, viruses, etc.) based on strong predictions of $\mathcal{P}$'s projected between quantum fields within each systems for alternative efficient paths to take (to reduce uncertainty) intelligently as its main goal and objectives.

2.2

Progress and Expected Outcomes



Show: QDF Circuit | Circuit Histogram Stop Fade Animation: ☑

Figure 2: Fig. 1: A quantum double-field (QDF) circuit with a strong prediction of system states, as the circuit's histogram on expected $\mathcal{P}$'s, from [1]. A QDF is developed from a single field particle (SF) transformation under a scalar κ, as discussed in Refs. [1], [2], and [3].

A scalar κ associated with a particle field Ψ by a sum (factor) of doubles of the speed of light c through superposition and entanglement as $2nc$. A photon field of Φ based on photon-particle interaction can transform Ψ via scalar κ as $|\kappa\Phi \longrightarrow \Psi^2 d^2|$. This doubles the probability and distance correlation between a particle pair in their pairwise fields $\Psi\Phi$ which can be measured with the assumption of having at least one superposed qubit between an input state and output state with an energy change needed to cause a phase transition. This makes a strong prediction of an event or a set of events possible, also rerouting energy states and their transition (events known as a phase transition of one matter state to another) for better efficiency of the system at every point in space achievable. In major, system efficiency such as cooling vs. heating, symmetry, equilibrium can be measured by employing this metric into a system as one determines input/output states by sensors and thus their distances based on $2d = 2nct$, where t is the time of an event occurrence (state transition) on any thermodynamic system scale. The animated screenshot below from the website title incorporates this field's sum concept in the quantum double-field (QDF) circuit. Click on the video, or gif button, to play the 1D (one-dimensional) QDF circuit example in a loop, or run both, the video and gif in parallel:

QDF circuit target and achievement:

1. The fraction result e.g., $\{⅓, ½, ⅔, \ldots 1/\}$ for $\mathcal{N} \geq 3$ particles from the adder Hamiltonian count is of the uncertainty in counting how many qubits have collapsed as classical bits now counted (right side of circuit resulted from the circuit's left side input) based on superposition. QDF dataset on this count must have:

$$\{\text{superposition-based positive integer as part of fraction count from a QDF}\}=\{\text{greater entropy } \vee \text{ greater event's uncertainty}\} \longrightarrow (0, \infty].$$

where '$\vee$' denotes logical OR which implies either condition can be true approaching (denoted by '$\longrightarrow$') values > 0 and infinity, as a true outcome.

2. The positive integer result for $\mathcal{N} \geq 3$ particles as $\{⅓ \longrightarrow ½ \longrightarrow ⅔ \longrightarrow 1, \ldots, \longrightarrow 2, \ldots, \longrightarrow 3, \ldots \longrightarrow \mathcal{N}\}$ from the adder Hamiltonian count is of the certainty in counting how many qubits have collapsed as classical bits now counted (the right side of the circuit results from the left side of circuit's input) based on entanglement. The QDF dataset on this count depends on measuring entanglement entropy as follows:

$$\{\text{entanglement-based positive integer count from a QDF}\}=\{\text{approaches zero entropy } \wedge \text{ reducing event's uncertainty}\} \longrightarrow 0.$$

where '$\wedge$' denotes logical AND which implies both conditions need to be true approaching (denoted by '$\longrightarrow$') 0, as a true outcome. Otherwise, it is an error in the QDF measurement of the circuit and applies to one of the conditions in #1 target above (superposition).

Achievement: This entanglement is developed after superposition predicting the opposite state within a QDF of particle-pair (two particles) not one in superposition. Thus, a full collapse of a qubit is predicted prior to count number of expected success probabilities $\langle \mathcal{P}_{\text{success}} \rangle$ of 1 or 100% predicted. See [2], as an expected outcome exemplified in Fig. 2.

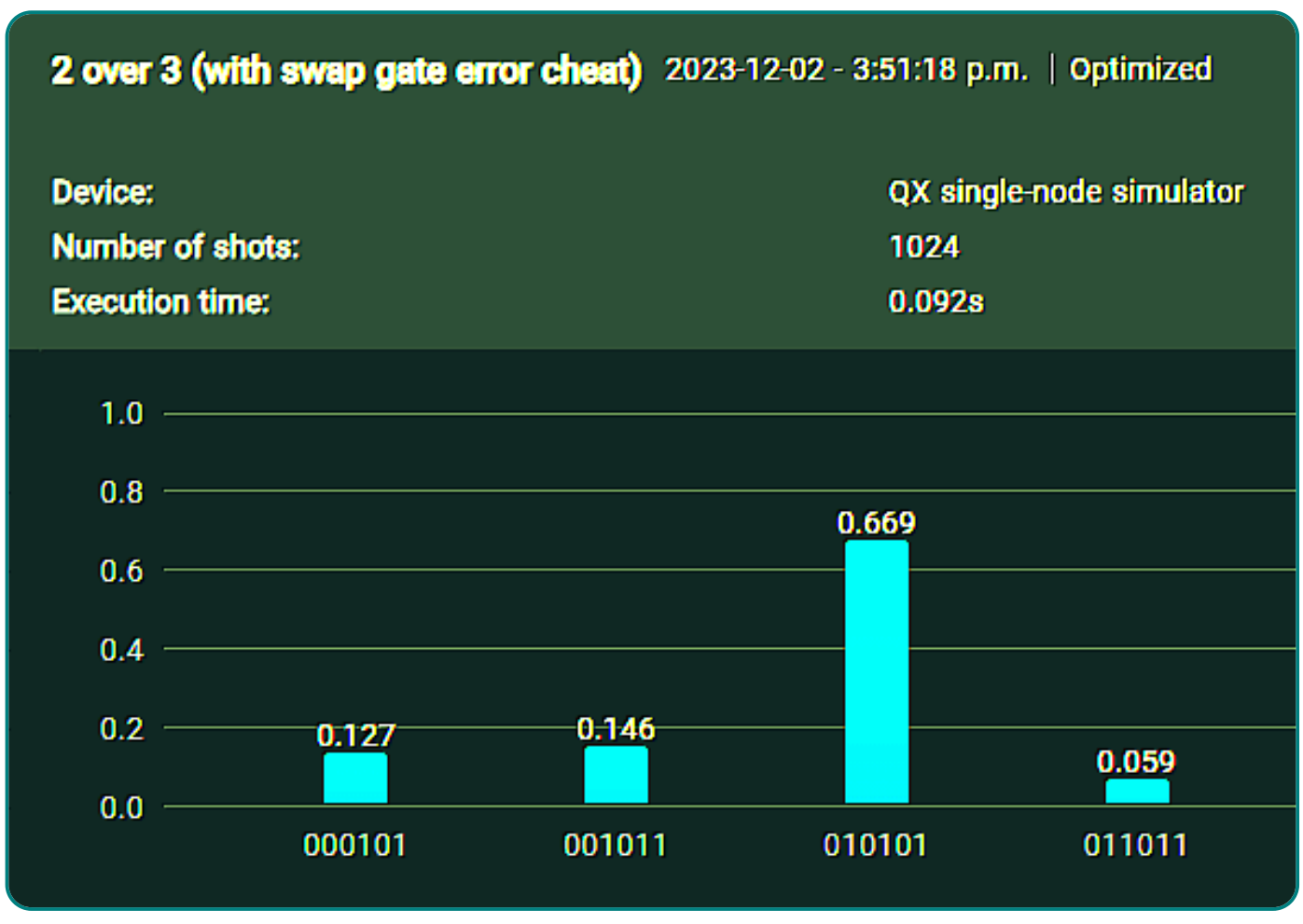


Figure 3: Fig. 2: Figure 1's Histogram as a dataset example showing a "desired Hamiltonian" (target state as circuit's expected outcome) [1] for a qubit-pair with a $\mathcal{P} \geq ½$. For different datasets denoting different circuit outcomes based on recorded $\mathcal{P}$'s (scenarios), visit the QFLCA Workflow page for demo and code run in python.

3. Plotting between the two scenarios is absolutely kappa (κ) scalar dependent on the transformation as denoted by the QDF circuit in any form of 1D, 2D connected circuit and more. The main equation is formulated throughout Refs. [1]-[3] as Eq. (53) from [3] to validate input results for each QDF dataset. See this QDF equation (formulaic) input below in the Project Objectives section to try this measurement.

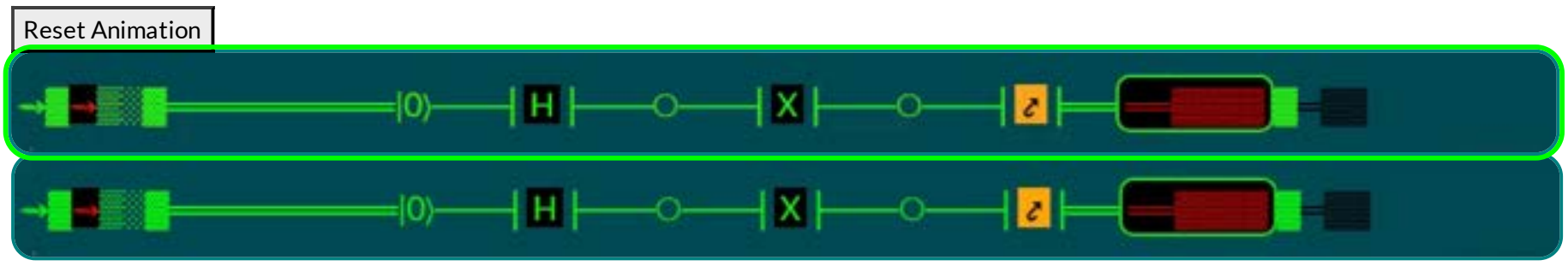
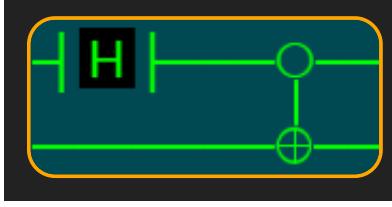


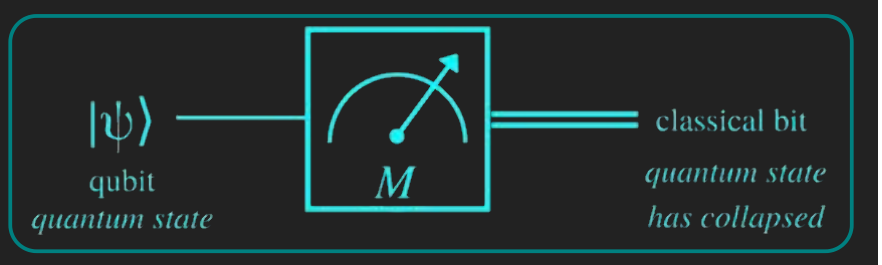
Figure 4: Fig. 3: 1D SF-to-QDF incomplete circuits running in parallel. The complete QDF circuit is Fig. 1 expanding the incomplete circuit for a configuration with expected probabilities ≥ 2/3 (Fig. 2).

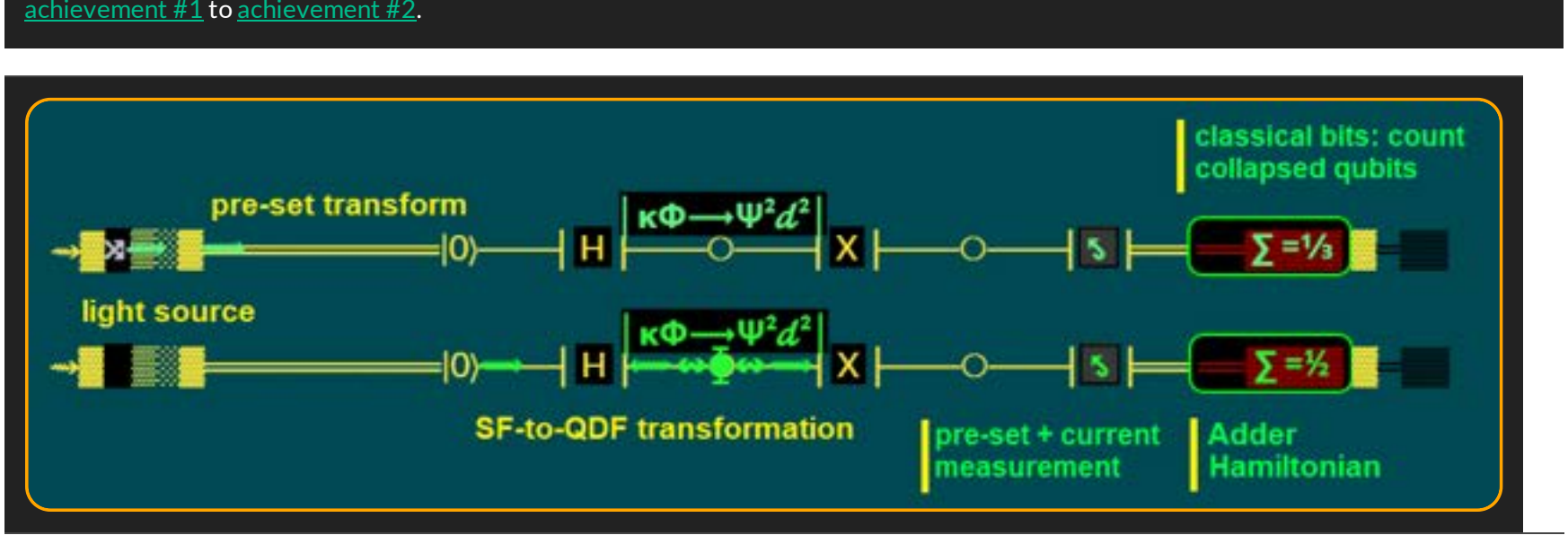
▼ QDF circuit legend and animation. . .

- | H | Hadamard quantum logic gate. See circuit layout of its use and description on p. of [2].
- | X | Pauli-X (or the NOT classical gate) quantum logic gate. See circuit layout of its use and description in Figs. 5 and 6 of [1].
- 

CNOT (or Controlled NOT) gate creating entanglement between qubits

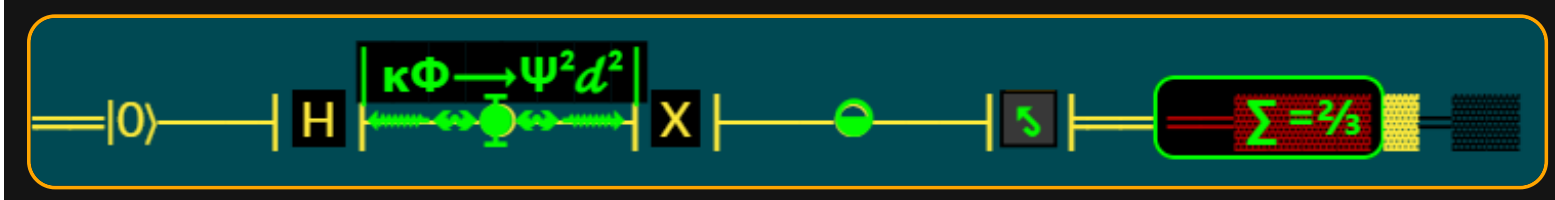
This applies to more than one 1D *connected parallel lines* within a complete QDF circuit [1]. The *complete QDF circuit* includes a | Z | gate and can be viewed via this link . The current section's circuit figure denotes this.

- |0⟩ initialized qubit in state 0. ◓ a definite state e.g., ground state, spin up or 0. For a photon, vertically polarized. ◒ a definite state e.g., excited state, spin down or 1. For a photon, horizontally polarized.
- | ↖ | quantum circuit measurement (qubit gauge) occurring.
  | ↗ | no measurement done, yet prepared to operate, or done. The latter time-step is after measurement when event results are recorded as a QFLCA dataset.
  Conventional symbol used elsewhere:
  

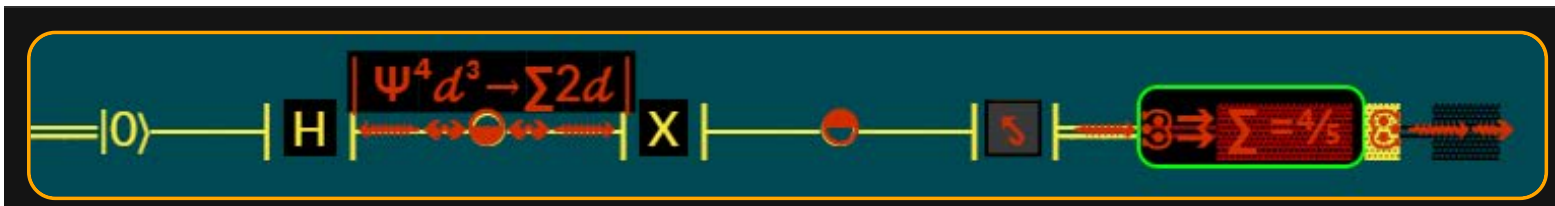
- : An SF-to-QDF product as a photon-particle interaction ( ), resulting in a qubit-pair state observation. This is the expected measurement outcome based on a strong prediction outcome to reduce the uncertainty i.e., mapping from achievement #1 to achievement #2.
- : A QDF-to-classical bit product as a photon readout from the qubit-pair ( ), resulting in a classical state observation. This is the expected measurement outcome based on a strong state prediction/certainty as achievement #2.
- 

Upper circuit: 1D SF-to-QDF incomplete circuit. Preset photonic measurement prior to scalar κ transformation of a particle field Ψ as as $|\kappa\Phi \longrightarrow \Psi^2 d^2|$. From the left, the photon from a light source (described by a proposed heat engine in [1, Fig. 1]) as a photonic field Φ projects on the particle field Ψ for an SF-to-QDF transformation, where at first uncertainty and superposition is expected (achievement #1) during measurement (right side).

Bottom circuit: same circuit performing post measurement after "superpositions, randomness, and interference" (via $|H|$) between gates (qubit pairs). This is assuming entanglement is created (achievement #2) by $|H|$ and a CNOT gate (not seen here) according to the QDF circuit configuration, or see current section's circuit figure. Direct link to view this circuit in detail is Fig. 5 or 6 of [1].



Certainty measure against uncertainty by doubling the field in ST $\mathcal{P}$'s as $\langle \mathcal{P} \rangle \geq 2\times\frac{1}{3} \geq \frac{2}{3}$, expected during a scalar κ transformation of a particle field Ψ as $|\kappa\Phi \longrightarrow \Psi^2 d^2|$. This is mapping from achievement #1 to achievement #2, or entanglement developed from superposition satisfying certainty per expected measurement outcome (adder Hamiltonian output).



Strong prediction measure of $\langle \mathcal{P}_{success} \rangle \longrightarrow 1$, expected after a scalar κ transformation of a particle field Ψ via Φ as $|\Psi^4 d^3 \rightarrow \sum 2d|$. This is achievement #2.

▼ QDF's one-to-more parallel circuit description…

When both the gif and video versions run, asynchronous/synchronous transformations of the single-field (SF) to QDF are observed [1] and [2].

- Note on the QDF circuit: Undertaking the simulation of the above parallel runs, frame-by-frame, can provide useful information to suggest a QAI input/output map to the user using this quantum circuit as an IDSS [2].
- Note on the QDF circuit expected outcomes: ⟐ A future presentation of this page for this level of programming using "cases" or "if-else statements" for particular sync and async frames in javascript or python for each QDF transformation is the expected outcome of this project.

▼ Expand/Collapse All…

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

Summary of progress and application: Prediction of combustion events as a phase transition occurring in the system, is detected by using photonic probes and sensors once atoms are sampled within a quantum heat engine. The engine incorporates a quantum double-field (QDF) lens function [1], as a new function which focuses/defocuses probability distribution values in the system. In major, system efficiency such as cooling vs. heating, symmetry, equilibrium can be measured by employing this quantum field distance metric into a system as one determines input/output states by sensors and predict their distances to reduce uncertainty (entropy) over time for an event occurrence (state transition) on any system scale. Its distance metrics makes accurate event predictions (data sampling and analysis from a quantum atomic/molecular scale) on a macro (large) scale in any heat engine built as a final product in numerous industries e.g. medicine (prediction on e.g., cells prone to cancer as a dynamical phase transition), aviation, automobiles, processors, meteorology, etc. The algorithm predicts events as an intelligent decision (support) simulator/solution (IDSS) [2], suggesting alternative energy paths to the user by visualizing (image-based map) real-time data with feedback addressing atomic (non)-interactions from one point to another. This is all about assuring quality in terms of maximizing system efficiency and performance.

QFLCA technical summary: Maximize system efficiency in determining the scalar behavior of entanglement (discussed in EE scaling and Eq. (16) from [1]) by a κ-based QDF transformation [1] and strong prediction doubling the state transition (ST) probability in the density matrix [3]. This is step #4 of the algorithm from p. 9 of the published article:

[1] Quantum field lens coding and classification algorithm to predict measurement outcomes
Philip B. Alipour and T. Aaron Gulliver

For validation purposes of the QFLCA dataset, this operational step correlates to the 4th objective below in the strong prediction [3] context.

2.3

OBJ Project Objectives

The current QFLCA simulates and predict system events which should achieve the following research goal (4th objective) and objectives (first three objectives):

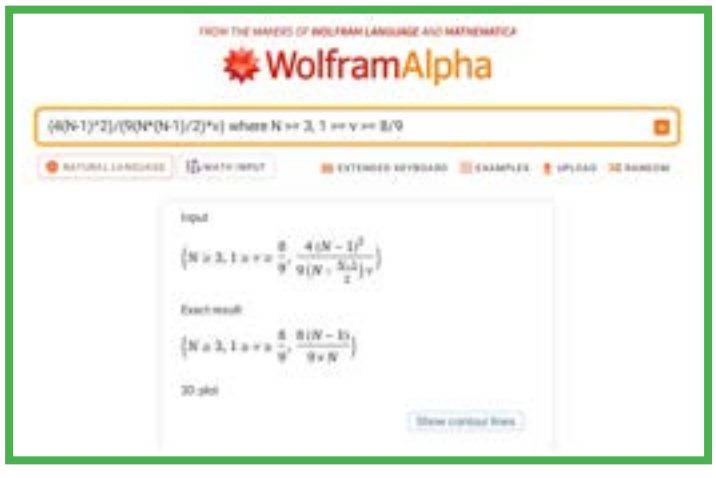
1. Employ system simulation models as the 1st objective, which is a quantum field theory (QFTh) simulation method. The method simulates QFTh to construct physical models that measure e.g., particle interactions.
2. Compare models' limitations and strengths as the 2nd objective.
3. Visualize events, their probabilities and predict them reliably to determine system efficiency, and suggest alternative energy paths to achieve a desired Hamiltonian [1] (target state) of the system.
4. The goal is to achieve a universal QFCM (UQFCM) [2] for the strongest system state prediction based on the results of model comparisons from the 1st through 3rd objectives.

Extracted from:

[2] Quantum AI and hybrid simulators for a Universal Quantum Field Computation Model
Philip B. Alipour and T. Aaron Gulliver

Input QDF ST probability value (formula): (4(N-1)^2)/(9(N*(N-1)/2)*ν), N = 3, ν = 8.123/9

Input value & calculate | Input default value & calculate | Input default range & plot



Equation (53) for strong QDF ST predictions is from:

[3] Quantum Double-field Model and Application
Philip B. Alipour and T. Aaron Gulliver

2.4 ⓘ About the Author

Dr. Philip Baback Alipour with a multicultural background and multiple citizenships (Canadian, mix of Persian and else), lived in different countries, was born in the Philippines, raised in the USA and now an active researcher in interdisciplinary fields of Computational, Electrical Engineering and Quantum Physics in Canada, USA and Sweden. Since his BSc (Hons) degree in the UK relative to postgraduate studies, he received his MSc and PhD respectively in Computer Science, Electrical Engineering and Quantum Physics from Lincoln University in the UK, BTH in Sweden, and Victoria University (UVic) in Canada . He has been active within the norms of physics, mathematics, engineering and applied sciences in a homogeneous manner (unification of their fields rather than treating them heterogeneously). The pragmatics of his quantum models, books and journal publications, has promoted concepts useful to global needs, advances in cutting-edge technologies (sustainable energy solutions, AI, quantum computing, etc.), ethics and well-being.

Current focus: Scalar quantum double-field (QDF) and distance metrics as a novel idea contribute in advances of science being made today over a decade ago. One can measure the key distance of two events (pairwise) by a new quantum lens function which focuses/defocuses probability distribution values in the system. In major, system efficiency such as cooling vs. heating, symmetry, equilibrium can be measured by employing this quantum field distance metric into a system as one determines input/output states by sensors and predict their distances to reduce uncertainty (entropy) over time for an event occurrence (state transition) on any system scale. See current project's expected outcomes.

Research interests are as follows: Mathematical, Physical, Computer Science, Software, Electrical Engineering, Knowledge Engineering and Business Intelligence. For more information on research notes, inventions, awards and publications see Author's ORCID for profile and research work information: 0000-0003-1037-018X

2.5 ᴬcₖ Acknowledgments

Endorsements, reviews and support for one or more publications in the QFLCA project are:

Dr. Philip B. Alipour (P. B. A.) acknowledges the University of Victoria Canada, for financial support. P. B. A. acknowledges the simulated QFLCA system sources and platforms where measurement data were collected after processing its parameters by the QFLCA run on IBM-QE and QInspire devices.
P. B. A. and Dr. T. Aaron Gulliver thank Dr. T. Lu for comments on scalar fields, dimensions and photonics, Dr. M. Laca for comments on equations that contribute to system state probabilities, Dr. N. Neumann remarks on QAI classifiers [1], Dr. K. S. Sørilen for classical prediction ideas to weigh $\mathcal{P}$'s and their certainty correlation, adding a dimension to the QAI map (witch the QDF Game Demo) to enhance strong prediction projections from combined datasets, and the late Dr. F. Diacu for his input on phase transitions to parameterize their quantum probabilities in the QDF model employing datasets.

2.6 Contact Info

- Courier address: Dept. of Electrical & Computer Engineering University of Victoria EOW 448, 3800 Finnerty Rd. Victoria, B.C. V8P 5C2 Canada.
- Postal address: Dept. of Electrical & Computer Engineering University of Victoria P.O. Box 1700 STN CSC Victoria, B.C. V8W 2Y2 Canada.

- **E-mail contact:** use one or both of the following email addresses for an online meeting or office visit: philipbaback_orbsix@msn.com , and/or phibal12@uvic.ca
- **Message via Linkedin:** linkedin.com/in/philip-baback-alipour-b35b13b

Subject-based message: For specific category regarding this project or any related ideas and this website, please use the following form.

2.7 $^{E}_{N_Q}$ Message the Author

Please send your message or comments about this project, webpage design and content, code suggestions, documentation, problems or updates through this form (mandatory fields are noted with a *):

Name*: *Your name...*

Email* : *Your address@example.com*

CC: *CC address@example.com*

Country: *Your country of residence...*

Choose a subject* : 1- Website design and function

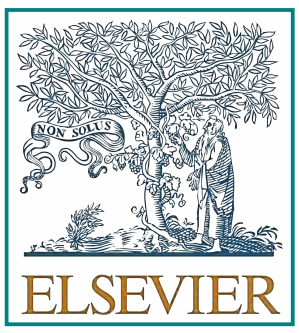
Your main field of interest* : Physics (mainly quantum physics)

Message* (limit is 1000 characters):

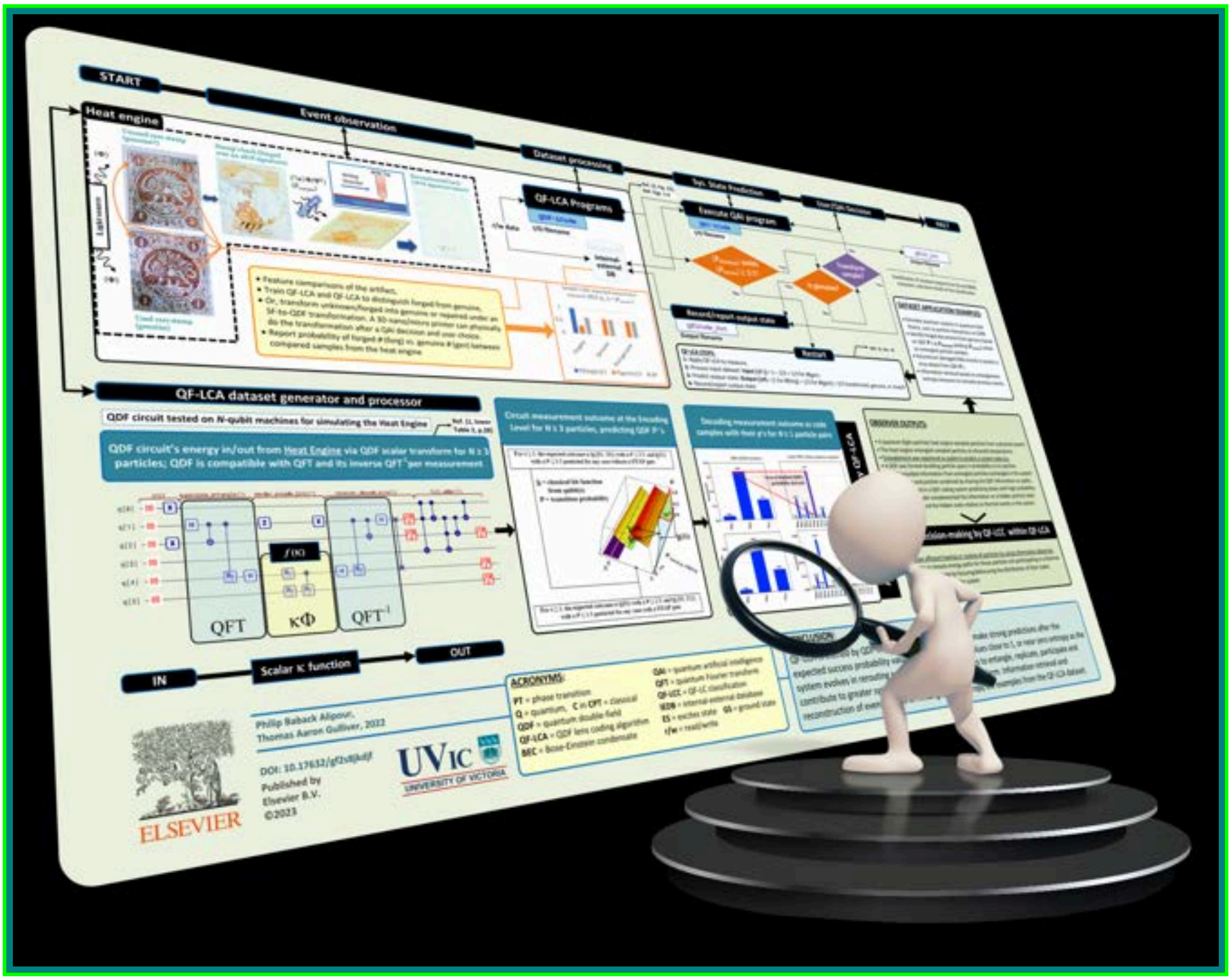
My message/comment about the project is...

Send via

Project content and website affiliations:



3 QFLCA Abstract, Highlights and Interests



This page presents a collection of quantum field lens coding algorithm (QFLCA) project abstracts, interests and highlights. The goal is to bridge between different fields of science in advances made in research and technology dependent on datasets, their analysis and application.

The expected outcome of this project is to make strong predictions of thermodynamic states of proposed systems on any scale. QFLCA simulates the system tio discover inefficient energy paths and suggests new energy paths to reroute, improving system efficiency and performance using a quantum double field (QDF) method with quantum scalar field properties. The algorithmic method produces datasets used for simulating new thermodynamic system models measured by quantum circuits, here defined as QDF circuits.

The main objective of the QFLCA model is proposing the application of QDF circuits that simulate systems using quantum computing such as, quantum AI, and entropy measurements to address uncertainties in the system under observation. From quantum computers to industrial, medical, physical and global economic solutions, by applying a QFLCA model for system state classification and measurements, presents datasets that can train the algorithm to map efficient energy paths for the system user/observer as presented in recent publications of its proposed system model and datasets.

For more project information, its progress and objectives, visit the About page

3.1 Abstract

Summary of progress and application: Prediction of combustion events as a phase transition occurring in the system, is detected by using photonic probes and sensors once atoms are sampled within a quantum heat engine. The engine incorporates a quantum double-field (QDF) lens function [1], as a new function which focuses/defocuses probability distribution values in the system. In major, system efficiency such as cooling vs. heating, symmetry, equilibrium can be measured by employing this quantum field distance metric into a system as one determines input/output states by sensors and predict their distances to reduce uncertainty (entropy) over time for an event occurrence (state transition) on any system scale. Its distance metrics makes accurate event predictions (data sampling and analysis from a quantum atomic/molecular scale) on a macro (large) scale in any heat engine built as a final product in numerous industries e.g. medicine (prediction on e.g., cells prone to cancer as a dynamical phase transition), aviation, automobiles, processors, meteorology, etc. The algorithm predicts events as an intelligent decision (support) simulator/solution (IDSS) [2], suggesting alternative energy paths to the user by visualizing (image-based map) real-time data with feedback addressing atomic (non)-interactions from one point to another. This is all about assuring quality in terms of maximizing system efficiency and performance.

3.2 Highlights

- `QDF and QFLCA Highlights:`
  - A pending event in a system is predicted by knowing all the information about prior events to it.
  - Two physical systems are simulated by a double-field computation (DFC) model to predict events.
  - Replicate the DFC method from the model to compute the next system state on a quantum level.
  - Superposition is a quantum (light-particle) behavior simulated by the DFC model.
  - A quantum double-field (QDF) by DFC doubles the particle space in probability to its position.
  - The DFC provides information on the possibility to entangle with a particle within its QDF.
  - Particle position is predicted by sharing the QDF information as quantum bits (qubits).
  - Collect QDF data from the models to represent energy states and transitions in the system.
  - Qubits are counted in a QDF coding system or DFC to predict states with high probability.
  - Add an extra qubit from the QDF via DFC to complement the qubit information for a target state.
  - DFC simulator generates the QDF circuit's dataset to compare with other datasets.
  - QFLCA compares datasets to classify energy states based on profiling them and their probability distances (correlation length). This is the QFLCC method.
  - Quantum artificial intelligence (QAI) is used to classify QDF states by this profiling method.
  - From the circuit state readout, the next system state is predicted through the QAI classification.

- `QDF Game Highlights:`
  - A game with a hidden prize run by a host can be won by altering its uncertain state.
  - Superposition is a quantum (light-particle) behavior by the host to hide the prize as a target state.
  - A quantum double-field (QDF) doubles the prize space in probability to its position.
  - A QDF provides information on the possibility to entangle with the prize and find it.
  - Prize position is predicted by sharing the QDF information as quantum bits (qubits).
  - Qubits are counted in a QDF coding system to predict states with high probability.
  - Adding an extra qubit from the QDF, complements the information on the prize qubit.
  - A QDF circuit achieves the target state 100% as the prize relative to any event in the universe.

- `QFLCA Dataset Highlights:`
  - A simulated quantum (light-particle) heat engine samples particles from a physical system.
  - Sampled particles from subsystem(s) are entangled at heat engine's ultracold temperatures.
  - Entanglement is registered as qubits to predict a system state as a QDF dataset:
  - Dataset shows a QDF was formed doubling particle space in probability to its position.
  - QDF data provides information from entangled particles exchanged in the system.
  - Particle spin and position are predicted by sharing the QDF information as qubits.
  - QDF circuit counts the qubits as its dataset to predict states with high probability, Eqs. (1)-(2b) [4].
  - The QDF circuit design and configuration associated with the dataset counts how many states have been predicted as the sum of successful hits by the IDS user/simulator (IBM QDF circuit simulator).
  - Dataset shows a QDF extra qubit complements the information on a hidden particle state.
  - QDF circuit simulator finds the hidden particle state relative to thermal events in the system.

3.3 Keywords

```
Quantum field lens coding algorithm (QFLCA), quantum double-field (QDF), quantum computer, quantum artificial intelligence (QAI),
quantum bit (qubit), command line interface (CLI), QDF transformation, QDF lens coding, DF computation (DFC), entanglement entropy
(EE), N-qubit machines, quantum Fourier transform (QFT), quantum lens coding classification (QFLCC), QDF Python game
```

4 QFLCA Abstract Collection

4.1 QFLCA Model Abstract

Extracted from:

[1] Quantum field lens coding and classification algorithm to predict measurement outcomes
Philip B. Alipour and T. Aaron Gulliver

This study develops a method to implement a quantum field lens coding and classification algorithm for two quantum double-field (QDF) system models:

```
1. a QDF model, and
2. a QDF lens coding model by a DF computation (DFC).
```

This method determines entanglement entropy (EE) by implementing QDF operators in a quantum circuit. The physical link between the two system models is a quantum field lens coding algorithm (QF-LCA), which is a QF lens distance-based, implemented on real N-qubit machines. This is with the possibility to train the algorithm for making strong predictions on phase transitions as the shared objective of both models. In both system models, QDF transformations are simulated by a DFC algorithm where QDF data are collected and analyzed to represent energy states and transitions, and determine entanglement based on EE. The method gives a list of steps to simulate and optimize any thermodynamic system on macro and micro-scale observations, as presented in this article:

- The implementation of QF-LCA on quantum computers with EE measurement under a QDF transformation.
- Validation of QF-LCA as implemented compared to quantum Fourier transform (QFT) and its inverse, QFT^{-1}.
- Quantum artificial intelligence (QAI) features by classifying QDF with strong measurement outcome predictions.

4.2 QFLCA AI Abstract

Extracted from:

[2] Quantum AI and hybrid simulators for a Universal Quantum Field Computation Model
Philip B. Alipour and T. Aaron Gulliver

Quantum field theory (QFTh) simulators simulate physical systems using quantum circuits that process quantum information (qubits) via single field (SF) and/or quantum double field (QDF) transformation. This review presents models that classify states against pairwise particle states $|ij\rangle$, given their state transition (ST) probability $P_{|ij\rangle}$. A quantum AI (QAI) program, weighs and compares the field's distance between entangled states as qubits from their scalar field of radius $R \geq |r_{ij}|$. These states distribute across $\langle R\rangle$ with expected probability $\langle P_{\text{distribute}}\rangle$ and measurement outcome $\langle \mathfrak{M}(P_{\text{distribute}})\rangle = P_{|ij\rangle}$. A quantum-classical hybrid model of processors via QAI, classifies and predicts states by decoding qubits into classical bits. For example, a QDF as a quantum field computation model (QFCM) in IBMQE, performs the doubling of $P_{|ij\rangle}$ for a strong state prediction outcome. QFCMs are compared to achieve a universal QFCM (UQFCM). This model is novel in making strong event predictions by simulating systems on any scale using QAI. Its expected measurement fidelity is $\langle \mathfrak{M}(\mathcal{F})\rangle \geq 7 \,/\, 5$ in classifying states to select 7 optimal QFCMs to predict $\langle \mathfrak{M}\rangle$'s on QFTh observables. This includes QFCMs' commonality of $\langle \mathfrak{M}\rangle$ against QFCMs limitations in predicting system events. Common measurement results of QFCMs include their expected success probability $\langle P_{\text{success}}\rangle$ over STs occurring in the system. Consistent results with high $\mathcal{F}$'s, are averaged over STs as $\langle P_{\text{distribute}}\rangle$ yielding $\langle P_{\text{success}}\rangle \geq 2 \,/\, 3$ performed by an SF or QDF of certain QFCMs. A combination of QFCMs with this fidelity level predicts error rates (uncertainties) in measurements, by which a $P_{|ij\rangle} = \langle P_{\text{success}}\rangle \lesssim 1$ is weighed as a QAI output to a QFCM user. The user then decides which QFCMs perform a more efficient system simulation as a reliable solution. A UQFCM is useful in predicting system states by preserving and recovering information for intelligent decision support systems in applied, physical, legal and decision sciences, including industry 4.0 systems.

4.3 QDF Abstract

Extracted from:

[3] Quantum Double-field Model and Application
Philip B. Alipour and T. Aaron Gulliver

A universal quantum double-field (QDF) model is introduced to predict particle states by doubling their probability outcome. These states are $i = 0$ as ground state (GS), 1 as excited state (ES), $|2\rangle$ as 0 and 1 denoting quantum superposition. A GS particle (Bob) interacts with particles over distance d to attain a target state (TS) with mode m as $|i_m\rangle$ e.g., a particle obeying Bose-Einstein condensate (BEC) or Fermi-Dirac statistics. A scalar κ is used to identify the \emph{effect} caused by light-matter interactions based on its value under a QDF transformation. A transition density matrix via κ provides state transition probabilities between the GS and ES particle fields. The information on the magnitude of the quantum particle position $|\mathfrak{r}|$ between three traps is gathered by a superposing particle, Alice or Eve, on a spatial magnitude of $|\kappa^2|\varrho \leq 2$, where ϱ is the probability density function for a given state $|2\rangle$. This state is projected by Alice or Eve onto the quantum particle field interacting with Bob's field forming a QDF.The QDF information shared with Bob in a QDF circuit, doubles in probability to occupy space on any scale of $\mathfrak{r}$. If Bob's field entangles with a GS field within the QDF, the TS 1 field is predicted (expected) for Bob to obtain, and viceversa. Here, the κ field probability doubles as the correlation length λ_{c} between states is measured relative to κ, $\mathfrak{r}$ and d, given the transition density matrix values obtained \emph{prior} to a phase transition. This gives a quantum information transmission via QDF on any scale to simulate BEC, heat engines and quantum communication models. For example, in refrigeration, an efficient cooling of atoms is obtained by sharing the quantum information to reroute atoms that weren't participating in the cooling event to participate.

4.4 QFLCA Dataset Abstract

Extracted from:

[4] QF-LCA dataset: Quantum field lens coding algorithm for system state simulation and strong predictions
Philip B. Alipour and T. Aaron Gulliver

Quantum field lens coding algorithm (QF-LCA) dataset is useful for simulating systems and predict system events with high probability. This is achieved by computing QF lens distance-based variables associated to event probabilities from the dataset produced by field lenses that encode system states on a quantum level. The probability of a state transition (ST), doubles in prediction values at the decoding step, e.g., ST probabilities of $P \geq 1/3$ into $P \geq 2/3$ based on a single field (SF) transform into a quantum double-field (QDF). This transformation doubles the ST probability space* via the field's scalar κ, in a published QDF method article [1]. A QDF, as a double-field computation (DFC) model, simulates thermodynamic systems in predicting events by producing datasets from a QF-LCA, using laser cooling methods relative to high energy systems. The dataset can be used to, e.g., train quantum algorithms via quantum artificial intelligence (QAI) for a QF-LCA user. The user then makes a decision after the algorithm predicts and suggests an efficient energy path for the system to choose. For example, an N-particle system simulated as a heat engine, by QAI classifier(s), predicts and reroutes particle energy paths on a logarithmic input-output scale. To determine system's energy change, the QDF lens code (DFC) measures entanglement entropy (EE). The algorithm's classification of EE values distinguishes entangled states in the system. The QF-LCA program employs the dataset to achieve an automated prediction and classification method, rather than dataset's manual use and analysis by the user. As the system evolves in its distribution of states, those particles not reaching a desired energy state (a target state or TS), i.e., the probability of observing a ground or excited state (GS or ES) outcome at the decoding step, can be rerouted by the heat engine to satisfy a TS outcome. This establishes a GS or ES energy profile to access and classify states by a classifier. The data points (qubits) in this profile are inverse distance-based, and labelled for a specific class. After learning the profile, the classifier decodes and predicts the next system state. A QDF AI game "Alice & Bob's Quantum Doubles," is developed to validate the dataset as the P's map for a classical/quantum prediction where the P's of states and the user's P's correlate in their value difference, ΔP. Dataset validation results are mapped to an intelligent decision simulator (IDS) as a QAI map. This maximizes system efficiency on a TS by EE measure of the distributed energy states. Future additions to the dataset from the QAI map program can improve quantum algorithms to determine which particles of the system participate in a phase transition after state prediction. QF-LCA applications are in data science, security, forensics, particle physics, etc., such as retrieving or reconstructing information by distinguishing particle states from an evidence sample. Examples are, reconstruct damaged DNA strands of cells to predict a virus's TS, or cancer cell, its spread and growth against healthy cells, identify forged documents from genuine based on QDF's P values, and so on.

```
* A probability space is a measure space such that the measure of the whole space is = 1.
```

4.5 QFLCA Software Abstract

Extracted from:

[5] Quantum field lens coding software for system state simulation, strong prediction and game application

Philip B. Alipour and T. Aaron Gulliver

A quantum field lens coding algorithm (QF-LCA) software is presented on a high-level end-user application run by $\mathrm{CLI} \longleftrightarrow \mathrm{GUI}$ with custom commands input by user to process, analyze, validate QF-LCA datasets in a QF-LCA and quantum game Python program. On the low-level system software, measurement data are acquired from quantum computers. The datasets contain these measurement data, processed and classified according to QF-LCA circuit design and steps to determine system states and their prediction. This software impacts advances made in applied sciences, statistics, law and physics, where data validation of samples including system simulation projecting and predicting events are achieved.

4.6 Sustainable Global Environment UQFCM Abstract

Extracted from:

[6] A Universal Quantum Field Computation Model for a Sustainable Global Environment and Society

Philip B. Alipour and T. Aaron Gulliver

A quantum double-field (QDF) algorithm is useful for simulating physical systems and predict events with high probability. The probability of a state transition (ST), doubles in prediction values at ST probabilities of $P \geq 1/3$ into $P \geq 2/3$ from a QDF. A QDF dataset is used to make strong predictions via quantum artificial intelligence (QAI). The QAI program from the dataset, classifies and predicts energy states to see which particles can participate in the next system state. For example, in medicine, reconstruct damaged DNA strands to predict a virus, cancer cell target, its spread and growth against healthy cells. In law, identify forged documents from genuine based on QDF's P values. A QDF as a quantum field computation model (QFCM) in IBMQE, can perform the doubling of P's for a strong prediction. A universal QFCM (UQFCM), from our co-publication, makes strong event predictions by simulating systems using QAI. A UQFCM is useful in predicting system states by preserving and recovering information for intelligent decision support systems (IDSS) in applied, physical, legal and decision sciences, including industry 4.0 systems. In this paper, societal impacts, goals and examples, are briefly discussed to propose a sustainable global environment by the UQFCM application.

---

5 QFLCA Interests for OA platforms

Quantum AI and hybrid simulators for a Universal Quantum Field Computation Model answer for a range of open access (OA) platforms for researchers. Among which, you may find the key research groups active within the fields below:

5.1 Physicists

```
How can quantum field theory (QFTh) simulators be utilized to predict system states and improve
measurement fidelity?
```

According to the abstract of [2],

- for QFTh simulators utilization to predict system states, the solution is given [2, p. 4]:
- for QFTh simulators improving measurement fidelity, the solution is given on [2, pp. 4-7] as:
- QFTh simulators are discussed in [2, Sec. 1.2] onwards.

In a QDF model [2, Sec. 1.2], the field's scalar component κ is associated with a particle field Ψ by a sum (factor) of doubles of the speed of light c through superposition and entanglement. The scalar is used to identify the effect caused by light-matter interactions based on its value under [a QDF transformation]. This doubles the probability and distance correlation between a particle pair in their pairwise fields $\Psi\Phi$ which can be measured with the assumption of having at least one superposed qubit between an input state and output state with an energy change needed to cause a phase transition.

5.2 Data Scientists

```
How can quantum AI be used to classify and predict states by decoding qubits into classical bits?
```

According to the abstracts of [1], [2] and [4], qubit measurement data as QDF datasets are acquired according to the QDF model via its algorithm QFLCA from quantum computers and simulators e.g., $\{\mathrm{QiskitAer}, \mathrm{IBMQproviders}\}$. The quantum double-field (QDF) [3] datasets [4] are generated based on a QDF circuit design and κ scalar field transformation from a single field (SF) to a QDF, which doubles the probability and distance correlation between a particle pair in their pairwise fields of a QDF. The datasets contain qubit measurement data simulating a thermodynamic system. The data are processed and classified according to QFLCA circuit design and steps to determine system states and their prediction. This determination is based on profiling system's energy states as classical bits once the quantum data points are analyzed according to QFLCA, Step #3 of Algorithm 2. From [1], the QDF transformation method is validated by profiling and weighing energy states. After profiling, a QAI map is generated to train the QFLCA for a system state prediction based on where less efficient energy paths are (previously recorded and profiled energy states) and suggest new efficient paths with high probability to choose as a desired Hamiltonian for the system.

5.3 Engineers

```
How can quantum-classical hybrid models of processors be applied to simulate systems on
any scale and achieve strong event predictions?
```

According to the abstract of [1,2] and the answer provided for data scientists, a system's energy states are simulated to measure the Hamiltonian and observe energy paths where particles interact. Classical bits are computed once the quantum data points from the simulation over energy paths are analyzed according to QFLCA steps from [1]. Upon profiling and weighing energy states, a QAI map is generated to train QFLCA for a system state prediction based on where less efficient energy paths are (previously recorded and profiled energy states) and suggest new efficient paths with high probability to choose as a desired Hamiltonian for the system. This is achieved by entanglement between the interacting particles and shuffling their superposition within their QDF where the expect P of their energy states is doubled in its measurement outcome. Hence the prediction of the energy states and paths to reroute for a more efficient system in e.g. a combustion vs. refrigeration thermodynamic event.

Project abstract sources, content and website affiliations:

6 Welcome to Visual Studio Code (VSC) for a QFLCA Code Run

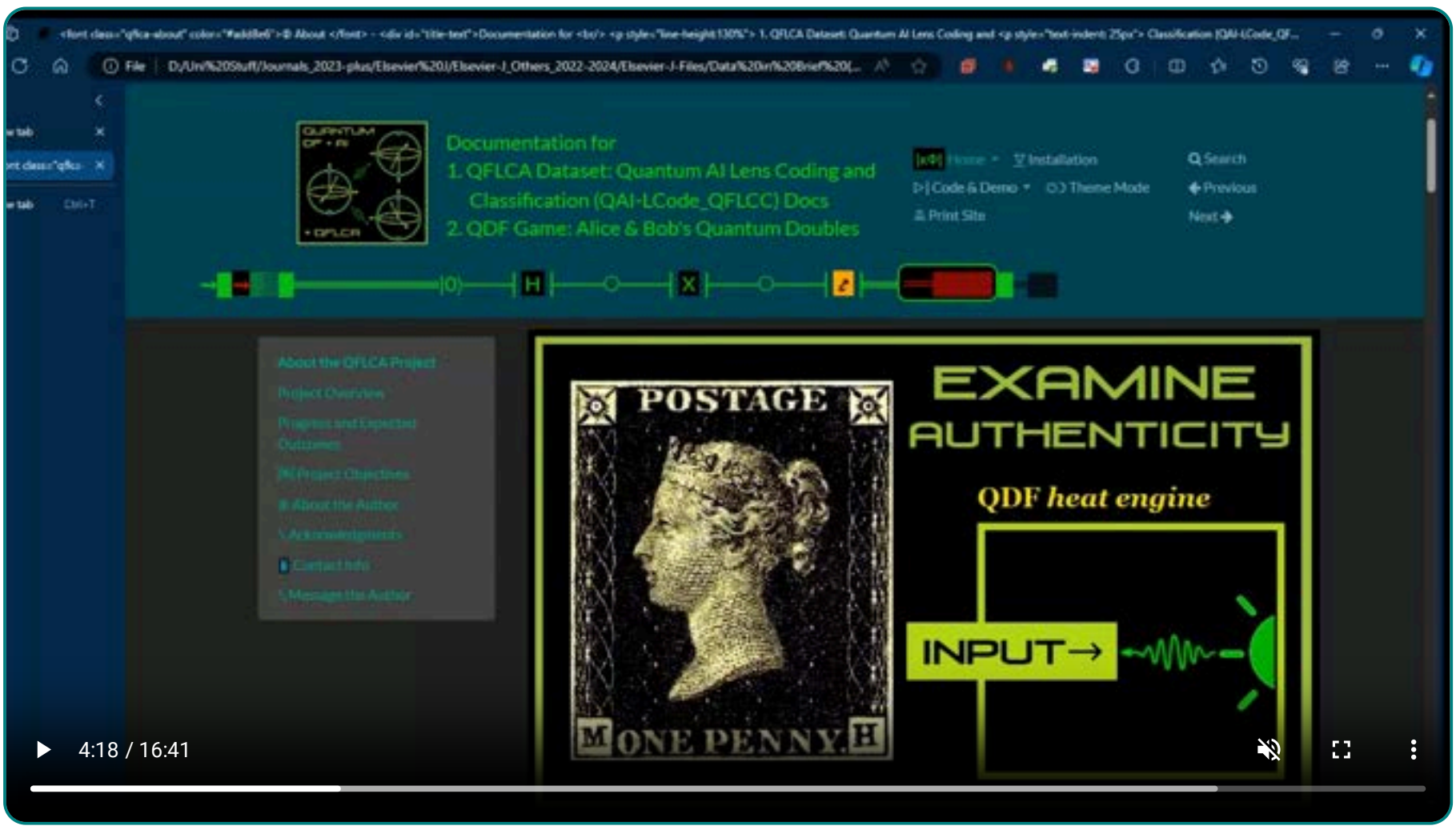
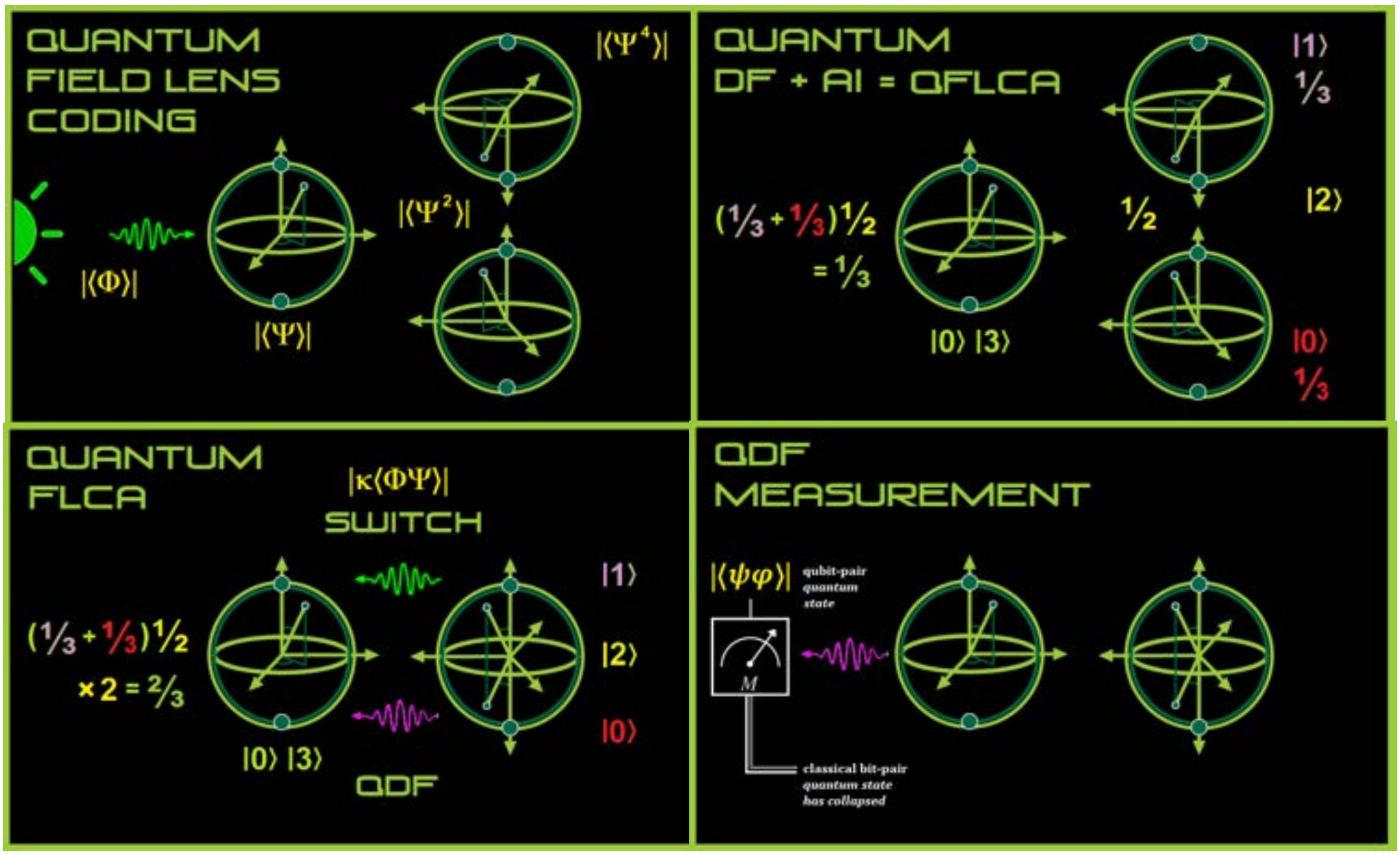
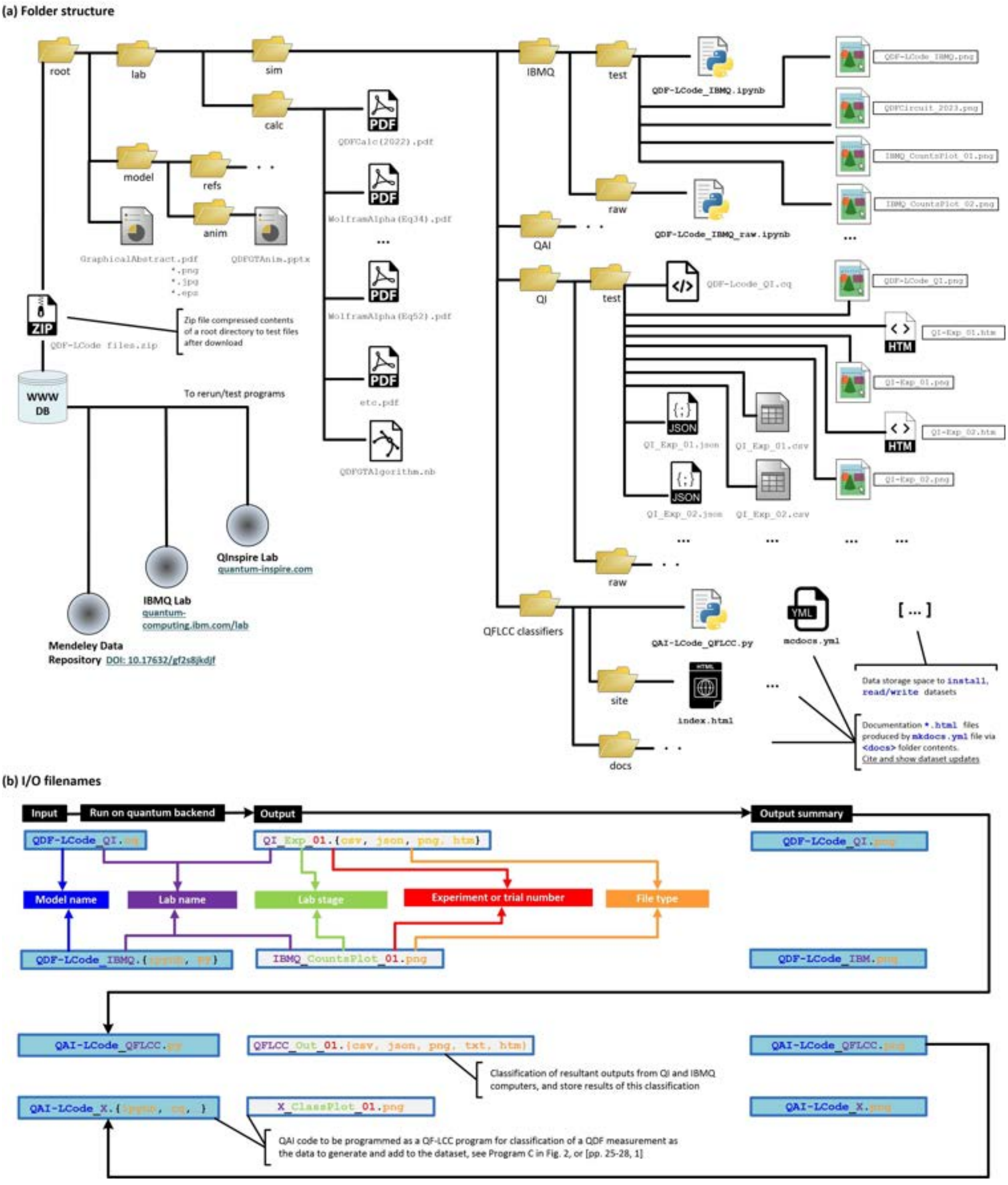


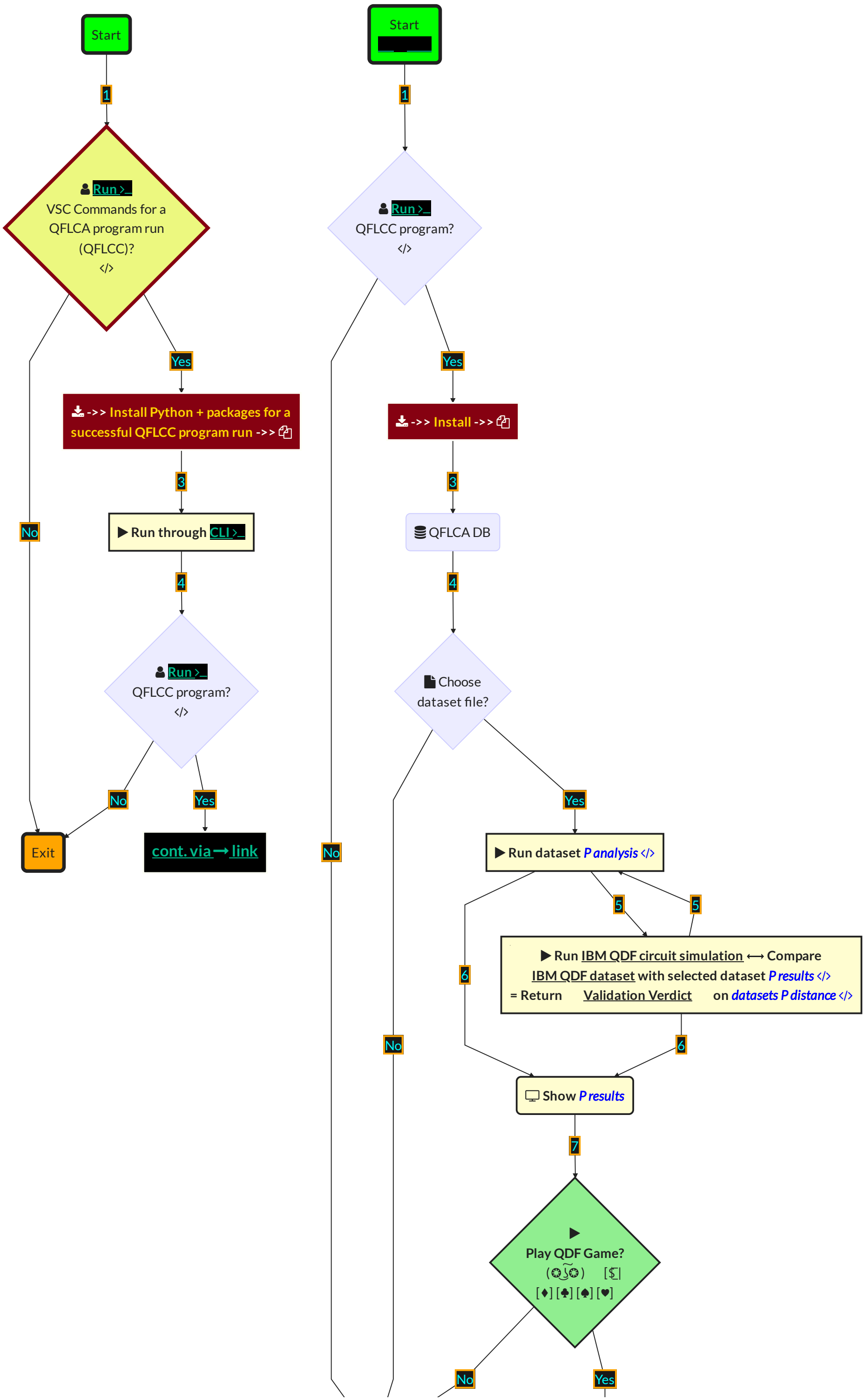
Figure 5: QFLCA Website Installation Demo: 788 MB mp4 file. ∇ | For Subtitles: SRT file. ∇

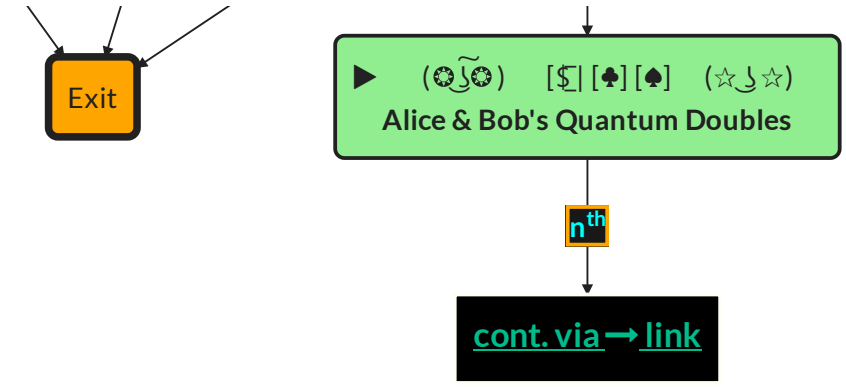
For the full VSC documentation visit visualstudio.com/docs and browse Python on the homepage menu to install packages.



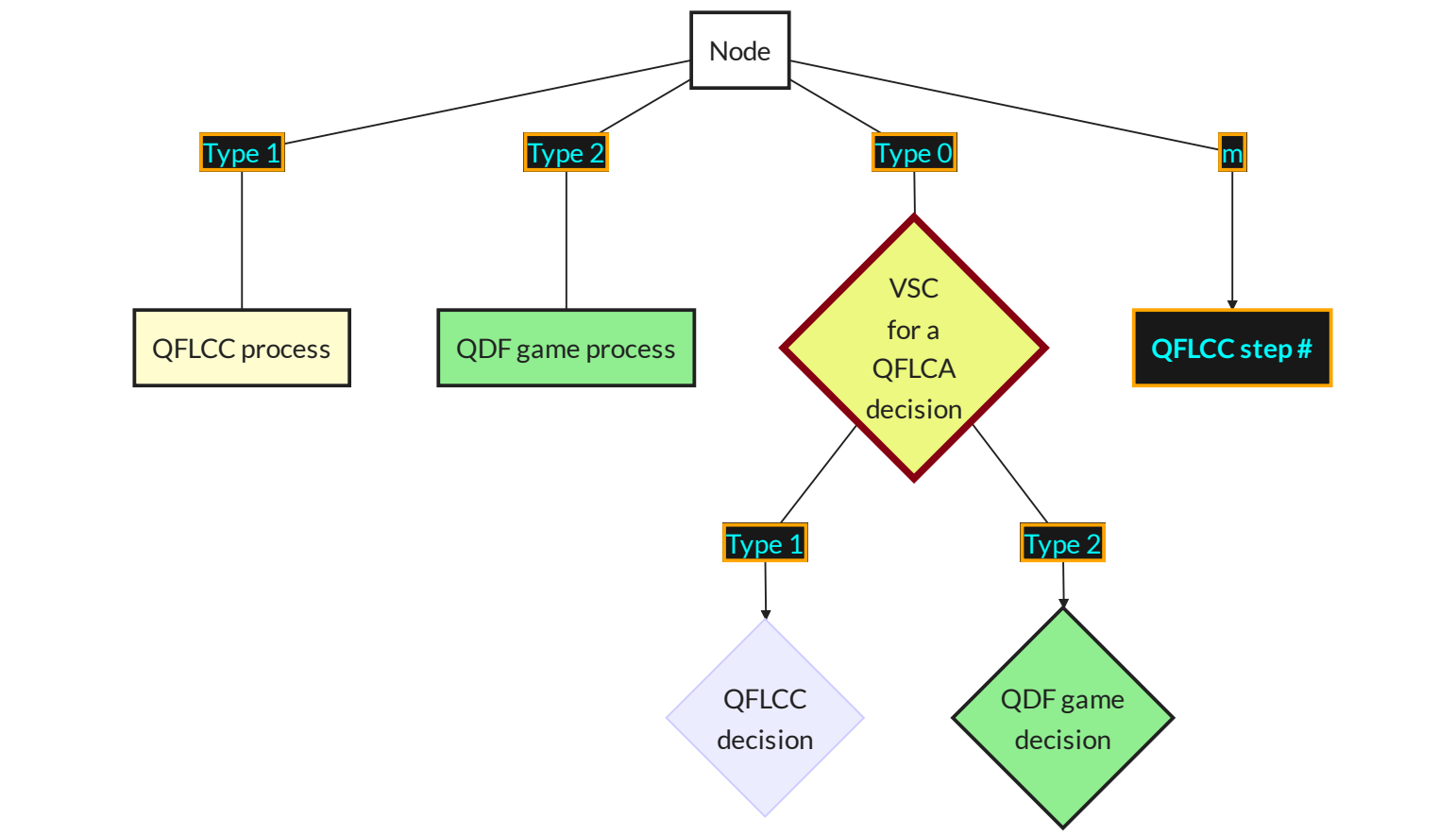
6.1 Program installation and code run flowchart

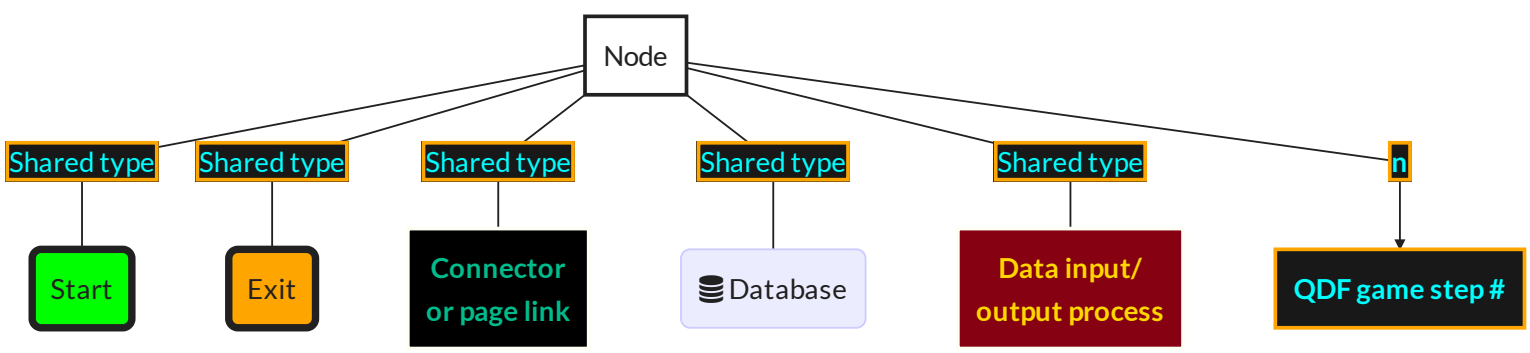



Start
1
Run >_
VSC Commands for a QFLCA program run (QFLCC)?
</>
Yes
->> Install Python + packages for a successful QFLCC program run ->>
3
No
▶ Run through CLI >_
4
Run >_
QFLCC program?
</>
No
Yes
Exit
cont. via ⟶ link
Start
1
Run >_
QFLCC program?
</>
Yes
->> Install ->>
3
QFLCA DB
4
Choose dataset file?
No
Yes
▶ Run dataset P analysis </>
5
5
▶ Run IBM QDF circuit simulation ⟷ Compare IBM QDF dataset with selected dataset P results </>
= Return Validation Verdict on datasets P distance </>
6
No
6
Show P results
7
▶
Play QDF Game?
(❂‿❂) [$]
[♦] [♣] [♠] [♥]
No
Yes



▼ Legend. . .





▼ Left flowchart description. . .

This 5-step diagram represents the flow of installation steps of the QFLCC program to run as `part of the QFLCA simulator`, listed in the `VSC commands` section below, and then its run by the user. It starts with dataset files installation, then the QFLCC program run via the VSC terminal ❯_ and coding environment.

▼ Right flowchart description. . .

This 8-step diagram represents the flow of QFLCC steps of the complete QFLCA program (17 steps) after a successful installation (left flowchart) and program run by the user. It starts with dataset files installation, then the selection of one of the datasets for probability `P analysis`. The results are further compared to the `IBM QDF circuit dataset` to `validate P results` between the two datasets by running the IBM QDF simulation module from the QFLCC program. Finally, the program `prompts` ❯_ `the user to continue` by playing the QDF game for further validation of dataset results (`P analysis`). The diagram continues on the relevant page linked here ↻.

6.2 VSC commands

- `VS Code program` -> `View` -> `Command Palette`... `(Ctrl+Shift+P)` -> Python: `Select Interpreter -> select "Python: Select Interpreter" (or Enter)` -> `select an interpreter` based on our chosen Python version under which you have installed the package.
  - `Change the VSC Interpreter` for the QAI-LCode_QFLCC.py project.
  - `Install package under the correct Python version`, which means to change your default Python version and repeat the process of installation again.

To change your default Python version (for Windows 10 OS):

- `Right click on This PC` -> `Properties` -> `Advanced System Settings` (in the right panel) -> `Environment Variables` -> `System variables (the bottom part of the window)` -> double-click on `"Path"` -> `Select the 1st row for the wanted Python version and move it up` -> then do the same with the 2nd row. I recommend to restart (close and open again) your Command Prompt session if you want to see/work with the new default Python version. - Correct Python version installation.

  Note on installation: Following command (in Command Prompt) works: `pip3 install pandas --user`

6.3 Run code commands

You can open a terminal in the current directory of the code as follows:

- `Press Ctrl+F5` or select `Debug` -> `Start` without Debugging in VSC environment. - Run program.
- `pip3 install <package name>` - For missing packages to import modules from, use this Python syntax.
- `pip3 install mkdocs` - For generating documentation and build the documentation site as you change code of this project. Then visit: homepage for more commands on MkDocs, or visit: mkdocs.org.

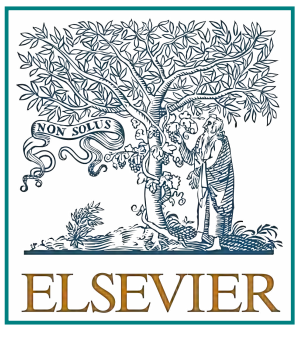
Optional:

- `pip3 install pdoc3`
  - For generating documentation and *.exe (under Windows OS) use. Then type after installation: `pdoc --html pdoc`
- `replace the "second pdoc" with your package/directory/filename` to view the documentation of this project. For more information, visit: pdoc3.github.io/pdoc
- Second method is, in the terminal you may run the following command to convert QAI-LCode_QFLCC.py file to QAI-LCode_QFLCC.exe on Windows 10 OS. `pip install auto-py-to-exe`

6.4 Project layout (folder structure)

```
QFLCC classifiers/     # The python file home directory. Type in your CLI this command:
                         cd "QFLCC classifiers"
                       # so to enter it under e.g., Windows OS, as its name has space(s).
  QAI-LCode_QFLCC.py   # Main python file. In VSC, press [Ctrl + F5] to run .py in CLI.
      *.py             # Python files imported as modules to run by QAI-LCode_QFLCC.py.
    ...                # Dataset files, etc. used for I/O operations by QAI-LCode_QFLCC.py.
mkdocs.yml             # The configuration file for documentation via <docs> content.
site/
    index.html         # Website homepage created in MkDocs by Markdown files inside <docs>.
    ...                # Other website pages and their dependencies such as image files.
docs/
    index.md  # The documentation homepage.
    ...       # Other Markdown pages, stylesheets, images, and their dependencies.
```

Project abstract sources, content and website affiliations:



7 QFLCA Workflow and Code

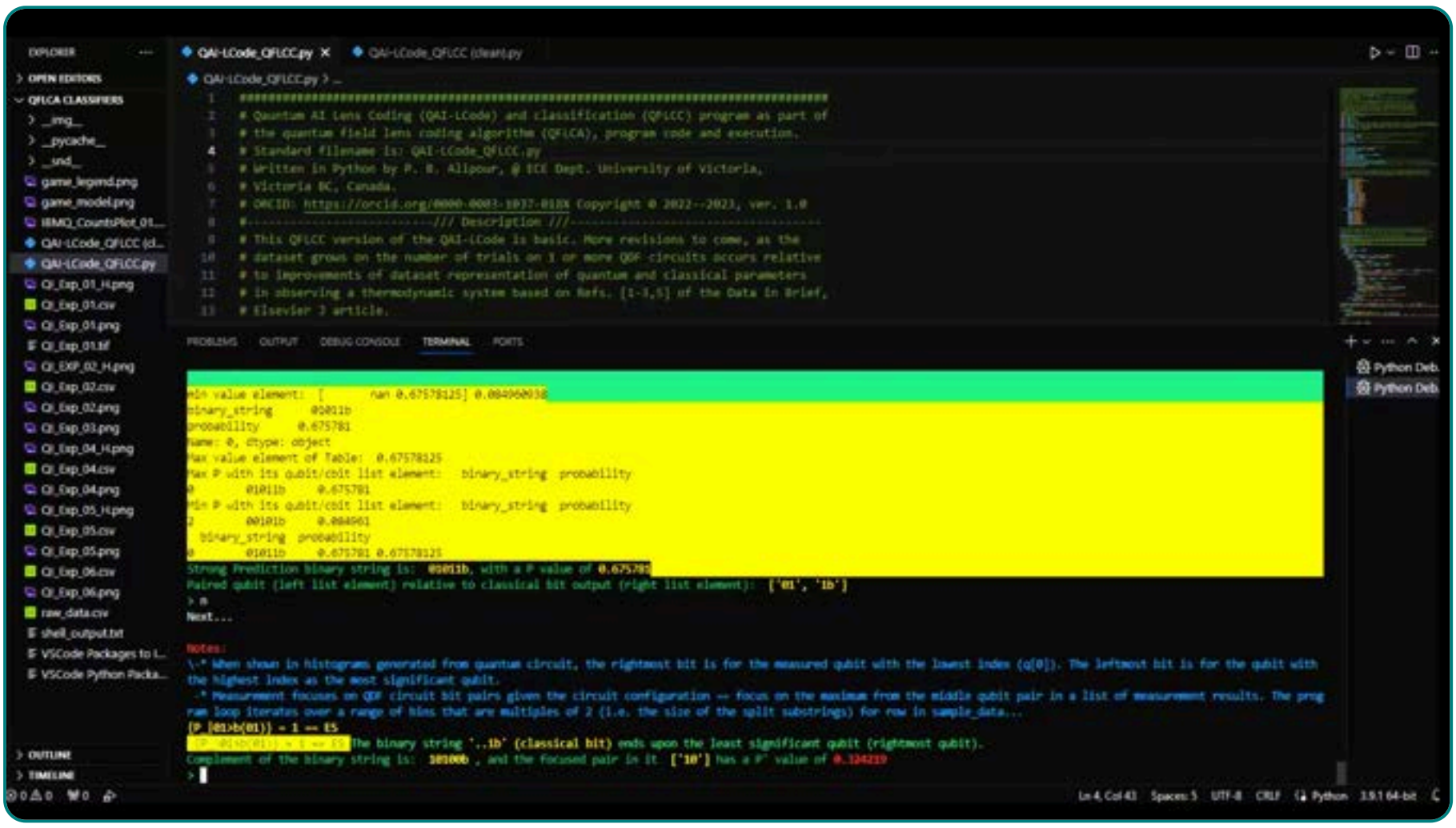


Figure 6: QFLCA Intro Demo: 337 MB mp4 file. ∇ | For Subtitles: SRT file. ∇

▼ ⁵% QFLCC program description (Start of Heading and Text) ⁵x . . .

QFLCC as part of QFLCA: program description from QAI-LCode_QFLCC.py

```
1  ##############################################################################
2  # Quantum AI Lens Coding (QAI-LCode) and Classification (QFLCC) program as part of
3  # the quantum field lens coding algorithm (QFLCA), program code and execution.
4  # Standard filename is: QAI-LCode_QFLCC.py
5  # Written in Python by P. B. Alipour, @ ECE Dept. University of Victoria,
6  # Victoria BC, Canada.
7  # ORCID: https://orcid.org/0000-0003-1037-018X Copyright © 2022--2024, ver. 1.0
8  #--------------------------/// Description ///-----------------------------------
9  # This QFLCC version of the QAI-LCode is basic. More revisions to come, as the
10 # dataset grows on the number of trials on 1 or more QDF circuits occurs relative
11 # to improvements of dataset representation of quantum and classical parameters
12 # in observing a thermodynamic system based on Refs. [1-3,5] of the Data in Brief,
13 # Elsevier J article.
14 ##############################################################################
```

▼ Program Code Update: Chap. 03 Demo for Chap. 01 (2024):

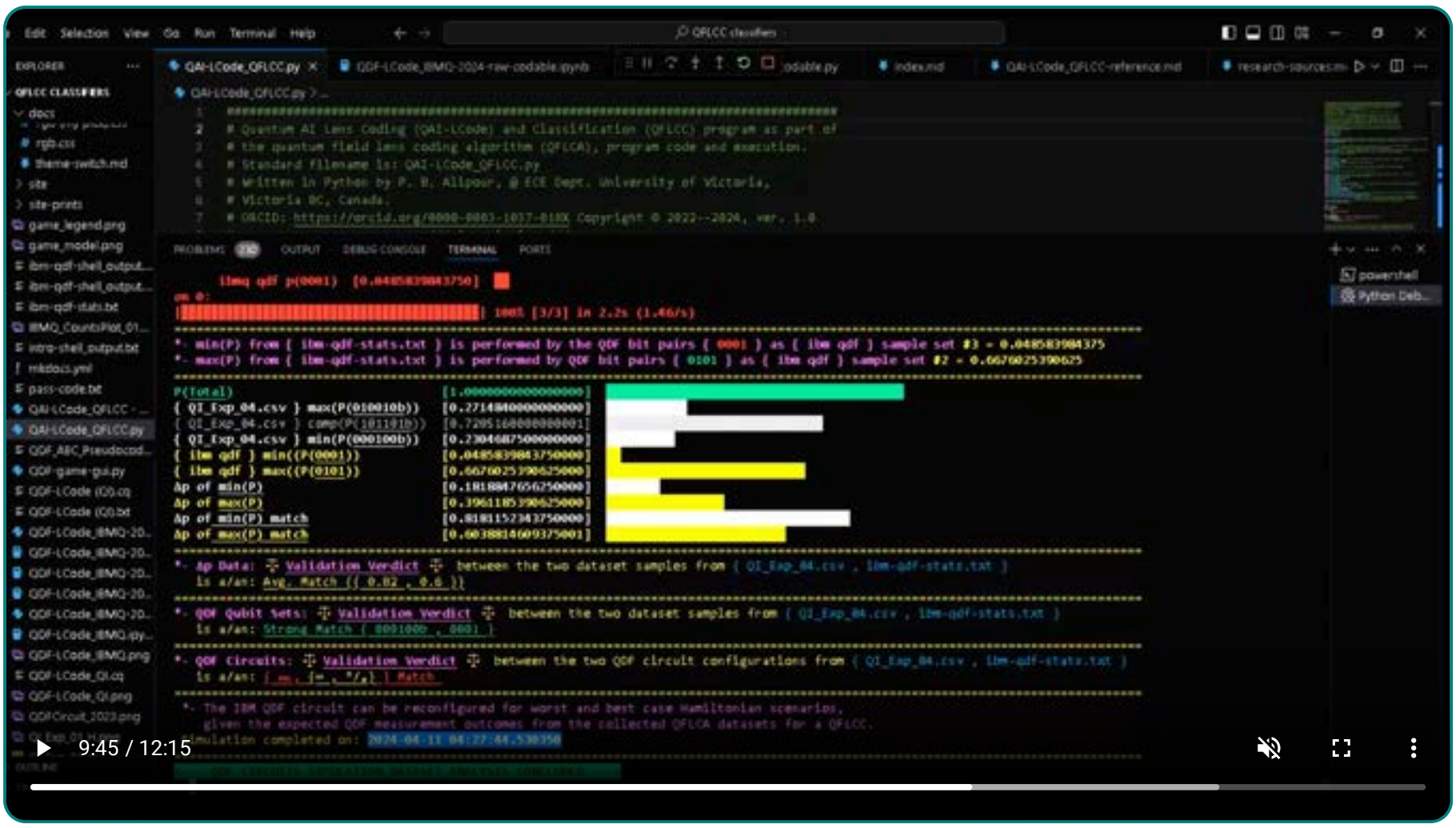
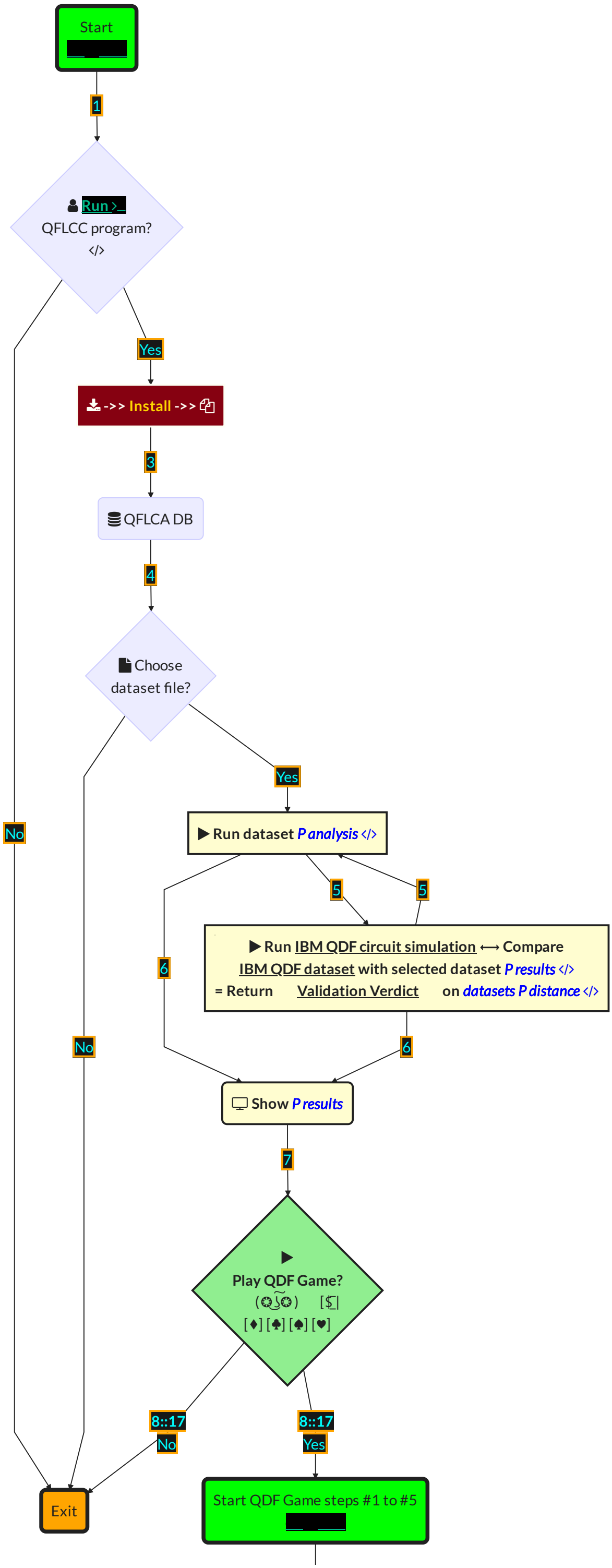


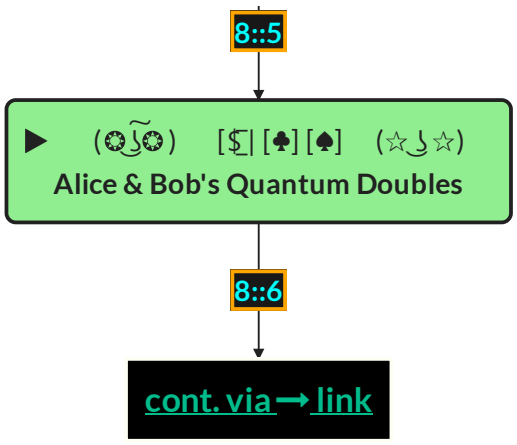
Figure 7: QFLCC 2024 Code Update: 616 MB mp4 file. ∇ | For Subtitles: SRT file. ∇

▼ Expand/Collapse All. . .

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view one or more code blocks from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```


Start
1
Run >_
QFLCC program?
</>
Yes
->> Install ->>
3
QFLCA DB
4
Choose
dataset file?
No
Yes
▶ Run dataset P analysis </>
5
5
▶ Run IBM QDF circuit simulation ⟷ Compare
IBM QDF dataset with selected dataset P results </>
= Return Validation Verdict on datasets P distance </>
6
No
6
Show P results
7
▶
Play QDF Game?
(❂‿❂) [$]
[♦] [♣] [♠] [♥]
8::17
No
8::17
Yes
Exit
Start QDF Game steps #1 to #5



▼ 🔶 Legend. . .

▼ Flowchart description. . .

This 8-step diagram represents the flow of QFLCC steps of the complete QFLCA program (17 steps) after a successful installation and program run by the user. The final steps are presented in parallel, side-by-side, between programs/algorithms labeled as `m::n` e.g., [`8::17`] denotes `step 8 (from the QFLCC algorithm)::17 (as part of the QDF game algorithm)` after a successful dataset installation and analysis by the program. This program starts with dataset files installation, then the selection of one of the datasets for probability `P analysis`. From there, the IBM QDF circuit simulation module: QDF-LCode_IBMQ-2024.py is imported and run by the `QAI-LCode_QFLCC.py` program using Qiskit packages in an isolated (virtual) environment on your computer. The simulation is called via PAnalysis_model ( ) function within `QAI-LCode_QFLCC.py` combined with operations to compute `dataset P's`.

- Alternatively, run QDF-LCode_IBMQ-2024.ipynb file on `external servers` and `quantum computer providers` from Jupyterlab workspace within the IBMQ lab (more limited).

The simulation results are compared to reach a `validation verdict` between the circuit's dataset and the selected one to measure their `P distance(s)`. Finally, the program `prompts` ❯_ `the user to continue` by playing the QDF game for `further validation of dataset results` (`P analysis`). The diagram continues on the relevant page linked here ↻.

▼ Expand/Collapse All. . .

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

8 QDF Circuit Files for QFLCA

The following `QDF circuit source codes` generate the QFLCA dataset based on the `QDF circuit design and implementation`, or see Ref. [1] on the About page. The output after each program execution on IBMQ, otherwise, the QInspire platform, given the circuit's configuration from Ref. [1], is later processed as a set of datasets (in e.g., `*.csv`, `*.htm`) as input(s) by the `QAI-LCode_QFLCC.py` program. For more about the objectives of this program visit Project Objectives on the About page, or see below for the detailed codes as commented within the program's body. Future updates will include ways of processing and updating the circuit directly by a customized QDF circuit configuration option for the user. This will be available through `CLI` and/or `GUI`. For example, the QDF game program, flowchart, its CLI and GUI example are the preliminary representation of the `QFLCA project objectives`. The aim is to give the user the option to process new outputs as `user's datasets` by the `QFLCA program`. The program's objective will remain for it to determine and classify probabilities according to the user's `desired Hamiltonian` (target state as circuit's expected outcome), or see Ref. [1].

▼ ≙ IBMQ source codes description and packages from filtered .py from raw .ipynb source code (see QDF-LCode_IBM-2024-raw.ipynb or QDF-LCode_IBM-2024.py to compile in VSC and your virtual qiskit environment). Also refer to code comments for installation notes:

IBMQ source code packages to draw the quantum circuit and count P's from QDF-LCode_IBM-2024.py

```
#-----------------------------------------------------------------------------------------
# Quantum simulation of a quantum field lens coding algorithm with entanglement scaling
# between multi-well barrier of internal system B and external system A.
# Created by: Philip B. Alipour, Supervisor: T. A. Gulliver, at the University of Victoria,
# Dept. ECE, Victoria BC, Canada.
# Code updated based on Qiskit 2023--2024 changes specified in code comments below.
# Side-notes: You can also run code with the right packages installed in pipx or python.
# Examples of changes can be found on e.g.,
# https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-installation and
# https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-features
# follow the link to fix 'deprecated release' lines, use
# <pip install "qiskit-aer>=0.11.0">
# and <pip install qiskit-ibm-provider> after activating your qiskit virtual environment in
# shell.
#-----------------------------------------------------------------------------------------
# Import the QISKit SDK... INstall Qiskit v.1 and update older ones < v.0.1 from above notes.
import termplotlib as tpl # To draw plots on circuit results during/after experiment.
import numpy as np
import sys
import re  # For search and spot regular expressions in {text, metadata, binary,...} I/O files.
from time import sleep
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import *
from qiskit_aer import Aer  # 2024 update... from its deprecated release.
from colorama import Fore, Back, Style  # For colored text messages.
#-----------------------------------------------------------------------------------------
# Newly added package in 2023/2024, the U3 gate has been renamed the U gate.
# from https://quantum-computing.ibm.com/composer/docs/iqx/operations_glossary
# and for cu1 gate to recreate this gate, add the control modifier to the phase gate
# (formerly the U1 gate); (older version of Qiskit cu1, now cp)
# U3 should be renamed just U as it is an arbitrary single-qubit gate.
# CU1 should be renamed Cphase or CP depending on the above. For more, visit
# https://github.com/Qiskit/qiskit-terra/issues/4106
#-----------------------------------------------------------------------------------------
from math import pi
from qiskit.compiler import transpile, assemble # In 2024, execute now is renamed to transpile.
from qiskit.visualization import *
from qiskit.providers.backend import Backend # Specify backend device for data processing
from qiskit_ibm_provider import IBMProvider  # 2024 update... from its deprecated release.
```

▼ ≙ IBMQ source codes... cont.

▼ qdf_circuit ( ) function code to compose, draw and run QDF measurements:

QDF default circuit configuration and experiment to generate P results for the QFLCA program.

- Screenshot of the circuit simulation after executing the `qdf_circuit()` and `print(qc.draw())` functions:

```
==========================================================================================================================
 An ideal QDF circuit I/O model running realtime producing an IBMQ-based QDF dataset is from the
 QDF-LCode_IBMQ-2024.py file. This circuit is compared to the analyzed dataset to show how close
 the match is for a desired Hamiltonian and expected measurement outcome as a point of reference.
==========================================================================================================================
on 0: Please Wait!...:
on 0: *- Program is processing the analyzed dataset circuit P's after:
on 2: *- Import + Simulate the IBM QDF circuit by { Qiskit Aer Simulator, IBMQ Provider } [QUANTUM MODE ENVIRONMENT] on this computer.
on 3: *- Dataset P results are compared on the imported circuit by the { QDF-LCode_IBMQ-2024.py } module on this computer [SAFE MODE ENVIRONMENT].
|████████████████████████████████████| 100% [3/3] in 3.1s (1.13/s)
==========================================================================================================================
Quantum simulation of a quantum field lens coding algorithm with entanglement scaling between a
multi-well (barrier) interaction potential of internal system B interacting and external system A from the method article:
https://www.sciencedirect.com/science/article/pii/S221501612300136X
*- QDF Circuit Built and Simulator generates its realtime datasets from { Qiskit Aer, IBMQ } Qasm_Simulator
   Workspace and/or Virtual Environment on user's computer in Python.
*- Created by: Philip B. Alipour, Supervisor: T. A. Gulliver, at the University of Victoria,
   Dept. ECE, Victoria BC, Canada.
*- Code updated based on Qiskit 2023--2024 changes specified in code comments of this simulator.
*- Sidenotes: You can also run code with the right packages installed in pipx or python.
   Examples of package installation changes and features can be found on:
   https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-installation and
   https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-features
==========================================================================================================================
==========================================================================================================================
<--- {{ Qiskit Aer, IBM }} QDF CIRCUIT SIMULATION BEGINS --->
==========================================================================================================================
Dynamic Providers: QasmSimulator('qasm_simulator')
Counts for Qubit Pairs out of 8192 shots: {'0101': 5541, '0111': 2243, '0001': 408}
 Σ (5541, 2243, 408) = 8192 shots ⟶  {P(b|ij))} = { ({n1, n2, n3}/8192)b|ij) }
 = { p1(0101) = 0.6763916015625 } + { p2(0111) = 0.2738037109375 } + { p3(0001) = 0.0498046875 } = 1
==========================================================================================================================
 Plot = [Experimented n of N counts  ∝  p of total circuit events P on pairwise qubits = P(b|ij))]; |ij) ≡ |q_i q_j) ≡ |qᵢqⱼ).
==========================================================================================================================
P(b|ij))              [1.0000000000000]
p1(b|ij)) = p1(0101)  [0.6763916015625]
p2(b|ij)) = p2(0111)  [0.2738037109375]
p3(b|ij)) = p3(0001)  [0.0498046875000]

<<-- IBM QDF Circuit Measurement Results by Qiskit Aer Simulator Plotted Successfully! End of Task... -->>

==========================================================================================================================
q1_0: -|0>─X─
q1_1: -|0>─H─■─Z─H─■─P(π/2)─P(π/2)─H─U(-0.32899,-π/2,π/2)─H─■─P(1.2337)─H─■─P(-π/2)─
q1_2: -|0>─X─X─H─U(5,-π/2,π/2)─H─■─P(π/2)─H─H─U(-1.2337,-π/2,π/2)─H─■─P(1.2337)─H─■─P(-π/2)─
q1_3: -|0>─
c0: 4/═══

«
«q1_0: ─M─
«
«q1_1: H─P(3π/4)─X─H─M─X─M─
«
«q1_2: H─U(5,-π/2,π/2)─H─X─M─X─M─
«
«q1_3: ─M─
«
«c0: 4/═0x1═0x2═
«        0 1  0 1 2 3

<<-- QDF Circuit BUILT & RAN Successfully! End of Task... -->>

==========================================================================================================================
<--- { Qiskit Aer, IBM } QDF CIRCUIT SIMULATION CONCLUDED --->
==========================================================================================================================
```

Ln 1092, Col 2 Spaces: 5 UTF-8 CRLF Python 3.11.8 64-

- Executable under Windows OS: runs the IBMQ circuit simulator (compiled from the `QDF-LCode_IBM-2024.py` file) in the `qflcc classifiers\__pycache__` directory (click here or the following screenshot to download and run this program):

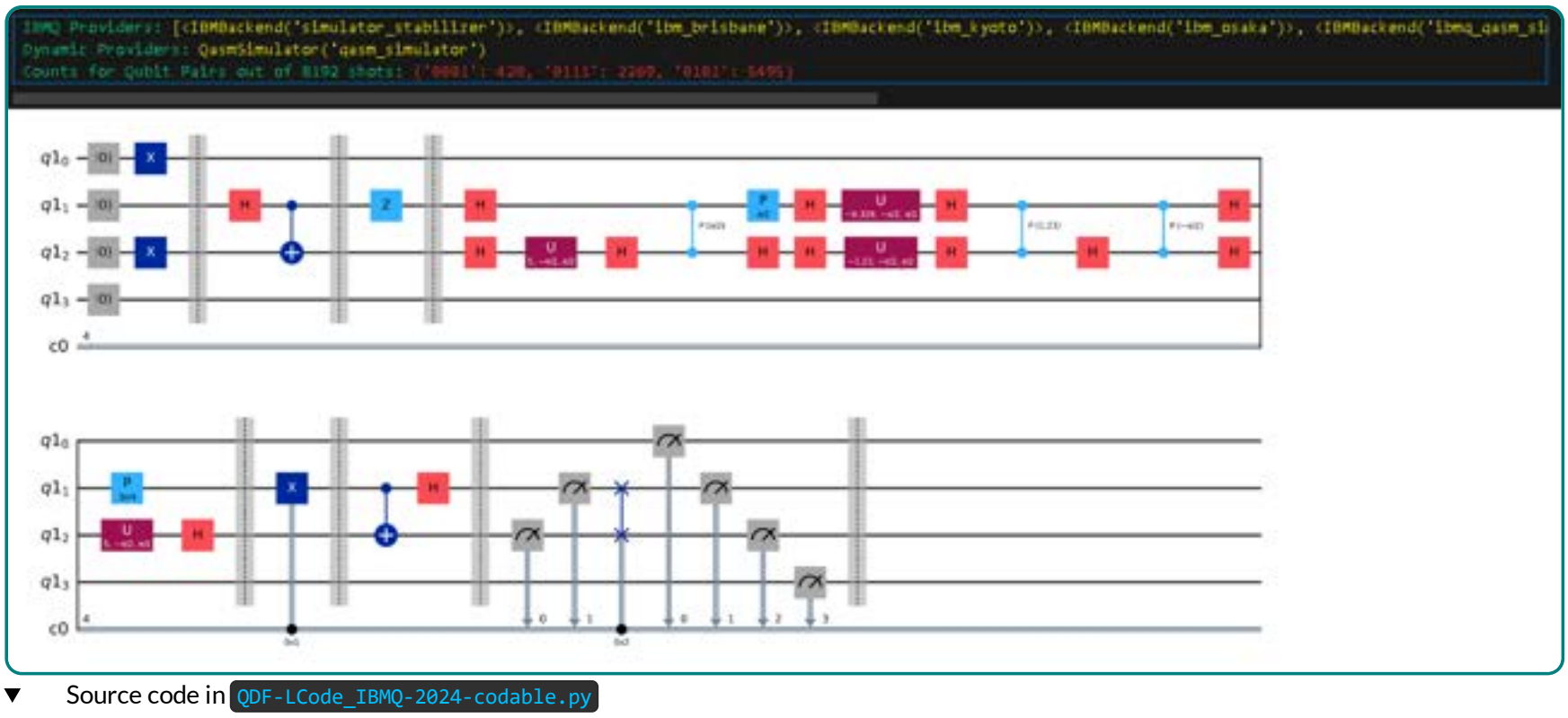
```
C:\Windows\py.exe
==========================================================================================================================
==========
P(b|ijE)              [1.0000000000000]
p1(b|ijE) = p1(0111)  [0.2701416015625]
p2(b|ijE) = p2(0101)  [0.6802978515625]
p3(b|ijE) = p3(0001)  [0.0495605468750]

<<-- IBM QDF Circuit Measurement Results by Qiskit Aer Simulator Plotted Successfully! End of Task... -->>

==========================================================================================================================
==========
q1_0: -|0>─X─
q1_1: -|0>─H─■─Z─H─■─P(π/2)─P(π/2)─H─U(-0.32899,-π/2,π/2)─
q1_2: -|0>─X─X─H─U(5,-π/2,π/2)─H─■─P(π/2)─H─H─U(-1.2337,-π/2,π/2)─
q1_3: -|0>─
c0: 4/═══

«
«q1_0: ─M─
«
«q1_1: H─P(1.2337)─H─P(-π/2)─H─P(3π/4)─X─H─M─X─M─
«
«q1_2: H─H─U(5,-π/2,π/2)─H─X─M─X─M─
«
«q1_3: ─M─
```

- Screenshot of the circuit simulation after executing the `qdf_circuit()` function based on the `QDF-LCode_IBMQ-2024-raw-codable.ipynb` or `QDF-LCode_IBMQ-2024-raw.ipynb` file:



- ▼ Source code in `QDF-LCode_IBMQ-2024-codable.py`

```
def qdf_circuit():  # Function to compose/build a QDF circuit.
 global qc    # Instantiate qc as a global variable to represent the quantum circuit.
 # Set your API Token. IBMQ.enable_account ('API Token') Create a Quantum Register with
 # 4 qubits.
 q = QuantumRegister(4)

 # Create a Classical Register with 4 bits.
 c = ClassicalRegister(4) # as _b in Fig. 6 of Ref. [1]

 # Create a Quantum Circuit.
 qc = QuantumCircuit(q, c)

 #-----------------------------------------------------------------------------------------
 # Implementation of Superdense Coding
 #-----------------------------------------------------------------------------------------
 # State initialization on 4 qubits in the z-basis in the |0> state.
 qc.reset(q[0])
 qc.reset(q[1])
 qc.reset(q[2])
 qc.reset(q[3])
 qc.x(q[0])
 qc.x(q[2])
 qc.barrier(range(4)) # Barrier to indicate physical barriers between systems A and B
                      # particles in communication.
 qc.h(q[1])
 qc.cx(q[1], q[2])
 qc.barrier(range(4))
 qc.z(q[1])   # Current state and encoded in the message through QFT.
 #qc.x(q[1])  # Current message encoded in the superdense coding scheme
 qc.barrier(range(4))

 # Time step = 1. Time evolution of potential energy part of the Ising model (spin
 # configuration tending to ground state or magnetization M>0) relative to kappaPhi
 # implementation.
 qc.h(q[2])
 qc.u(5,-pi/2,pi/2,q[2])
 qc.h(q[2])

 #-----------------------------------------------------------------------------------------
 # Entanglement Encoder Implementation
 #-----------------------------------------------------------------------------------------
 # Two-qubit Inverse Quantum Fourier Transform (QFT^-1)
 qc.h(q[1])
 qc.cp(pi/2, q[2], q[1])
 qc.h(q[2])

 # Time evolution of kinetic Ising model (kinetic energy part satisfying magnetization
 # value M=0)
 qc.p(pi/2, q[1])
 qc.h(q[1])
 qc.u(-(pi**2)/30, -pi/2, pi/2, q[1])
 qc.h(q[1])
 qc.h(q[2])
 qc.u(-(pi**2)/8, -pi/2, pi/2, q[2])
 qc.h(q[2])
 qc.cp((pi**2)/8, q[2], q[1])

 # Two-qubit Quantum Fourier Transform (QFT)
 qc.h (q[2])
 qc.cp (-pi/2, q[2], q[1])
 qc.h (q[1])

 # Time evolution of potential energy part
 qc.p(3*pi/4, q[1])
 qc.h (q[2])
 qc.u (5, -pi/2, pi/2, q[2])
 qc.h (q[2])
 qc.barrier(range(4))

 #-----------------------------------------------------------------------------------------
 # Continuation of the superdense code algorithm. Remove IF statement for real quantum
 # computers when enabling one of the backends after 'qasm_simulator' below
 #-----------------------------------------------------------------------------------------
 qc.x(q[1]).c_if(c, 1)  # IF statement , as if the prize is spotted via Eve or the
 # audience, ask Bob to decide to win the prize or a prize of lesser quality or energy
 # value; flipping condition for Alice to cloak the prize is 0 as opposed to 1 or 2 in
 # decimal. Remove IF statement when testing on a quantum computer and not a simulator.
 qc.barrier(range(4))
 qc.cx(q[1], q[2])
 qc.h(q[1])
```

```
    qc.barrier(range(4))
    qc.measure(q[2], c[0])  # Qubit 2 is in state |0>
    qc.measure(q[1], c[1])  # Qubit 1 is in state |1>

    # Remove IF statement for real quantum computers when enabling one of the backends after
    # 'qasm_simulator' below qc.swap(q[1],q[2]).c_if(c, 1) # SWAP gate is used if
    # condition c=1 or 01 in binary.
    qc.swap(q[1],q[2]).c_if(c, 2)   # SWAP gate is used if condition c=2 or 10 in binary
    qc.measure(q, c)  # Map the quantum measurement to the classical bits
    qc.barrier(range(4))

    global hline
    hline =
"=============================================================================================================================================="
    print(f"{Fore.LIGHTGREEN_EX + hline}\
    \nQuantum simulation of a quantum field lens coding algorithm with entanglement scaling between a\
    \nmulti-well (barrier) interaction potential of internal system B interacting and external system A\
    from the method article: \
    \n{Fore.LIGHTCYAN_EX}https://www.sciencedirect.com/science/article/pii/S221501612300136X{Fore.LIGHTGREEN_EX} \
    \n*- QDF Circuit Built and Simulator generates its realtime datasets from\
    {Fore.LIGHTYELLOW_EX}{{ Qiskit Aer, IBMQ }} Qasm_Simulator{Fore.LIGHTGREEN_EX}\
    \n   Workspace and/or Virtual Environment on user's computer in Python. \
    \n*- Created by: Philip B. Alipour, Supervisor: T. A. Gulliver, at the University of Victoria,\
    \n   Dept. ECE, Victoria BC, Canada. \
    \n*- Code updated based on Qiskit 2023--2024 changes specified in code comments of this simulator. \
    \n*- {Fore.LIGHTRED_EX}\033[4m\033[1mSidenotes:\033[0m{Fore.LIGHTGREEN_EX} You can also run code with\
    the right packages installed in pipx or python.\
    \n   Examples of package installation changes and features can be found on: \
    \n   {Fore.LIGHTCYAN_EX}https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-installation and \
    \n   https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-features{Fore.LIGHTGREEN_EX} \
    \n{hline + Fore.YELLOW}")
    print(f"{Fore.LIGHTGREEN_EX}Press any key to continue...")
    os.system("pause >nul")
    print(f"\033[F{Fore.LIGHTMAGENTA_EX + hline}")
    print(Back.LIGHTGREEN_EX + Fore.YELLOW +
          "\033[1m<--- {{ Qiskit Aer, IBM }} QDF CIRCUIT SIMULATION BEGINS --->\033[0m" + Back.RESET)

    #The following lines plot the P of measurements from the QDF circuit and draws the circuit.
    global counts_exp   # Used to plot P results for N shots run on the QDF circuit.
    #-----------------------------------------------------------------------------------
    # Choose backend, number of shots and the plotting of histogram.
    #-----------------------------------------------------------------------------------
    #my_provider = IBMProvider() # Previously was IBMQ.load_account().
    #print(Fore.LIGHTMAGENTA_EX + f"{hline + Fore.GREEN}\nIBMQ Providers:" + Fore.YELLOW, my_provider.backends())
    #ibmq_pick = my_provider.get_backend('ibmq_qasm_simulator') # To run on without dynamic support.
    dyn_pick = Aer.get_backend('qasm_simulator') # The device to run on dynamically... 2024 update.
    print(Fore.LIGHTMAGENTA_EX + f"{hline + Fore.GREEN}\nDynamic Providers:"+ Fore.YELLOW, dyn_pick)
    #ibmq_pick = provider.get_backend('simulator_statevector')
    #ibmq_pick = my_provider.get_backend('ibmq_athens') # This machine was recently retired.
    #ibmq_pick = my_provider.get_backend('ibmq_16_melbourne') # This machine was recently retired.
    shots = 8192 # Number of shots to run the program (experiment); maximum is 8192 shots.
    job_exp = dyn_pick.run(qc, shots = shots) # 2024 update... Qiskit package v.1.0 (see heading)
    result = job_exp.result()
    print(Fore.GREEN + f'Counts for Qubit Pairs out of {shots} shots:' + Fore.RED,
          result.get_counts(qc), Fore.LIGHTGREEN_EX)
    counts_exp = (result.get_counts(qc))

    #--- The above code snippet is the old version limited to counts ratio to maximum number
    # of shots = 8192 resulting to plot probabilities in older Qiskit versions (as
    # presented in our method article, Ref. [1]).
    # New version to plot probabilities require further computation included for printing
    # out the same probabilities based on Maximum of counts/total number of shots =
    # a list of P(qubit pairs).
    # The observed probabilities are computed by taking the respective counts and dividing
    # by the total number of shots as follows.
    #-----------------------------------------------------------------------------------
    #-----------------------------------------------------------------------------------
    # The following lines calculate P values and plot them for the 'counts_exp' above after
    # being recorded as expected measurement outcome from the QDF circuit.
    #-----------------------------------------------------------------------------------
    #N[P(kappa Phi to Psi)]
    P = 1 # The total probability of QDF circuit events relative to classical states or
          # denoting the system's Hamiltonian (total energy).

    file = "ibm-qdf-shell_output.bin" # I/O bin or metadata file to binary file to read/write.
    with open(file, 'w') as file_to_write:
       file_to_write.write(str(counts_exp))
       file_to_write.close() # End write counts_exp metadata to the file.

    n_list = []  # Create an empty list for numbers to store.
    bin_list = []  # Create an empty list for binaries to store.
```

```
with open(file, 'r') as file:

        # Read the contents of the file
        content = file.read()

        # Extract all the digits from the content and convert to int.
        N = re.findall(r'\d+', content)
        n_list = list(map(int, N))

        # Convert all the metadata digits to str to sort out binaries.
        bin_list = list(map(str, N))

        # Calculate quantum event p's for the total P.
        p1 = n_list[1]/shots # 1st p result is stored to the p_list.
        p2 = n_list[3]/shots # 2nd p result is stored to the p_list.
        p3 = n_list[5]/shots # 3rd p result is stored to the p_list.

    # This prepares a list of event p's summing up to P in total from the shots.
    # Counts for QDF pairwise qubits:
    print(f'{Fore.YELLOW} Σ ({n_list[1]}, {n_list[3]}, {n_list[5]}) = {shots} shots ⟶  \
    ⟨P(b|ij)⟩ = {{ ({{n1, n2, n3}}/{shots})b|ij⟩ }} \
    \n = {Fore.LIGHTCYAN_EX}{{ p1({bin_list[0]}) = {p1} }} {Fore.YELLOW}+{Fore.LIGHTCYAN_EX}\
    {{ p2({bin_list[2]}) = {p2} }} {Fore.YELLOW}+{Fore.LIGHTCYAN_EX}\
    {{ p3({bin_list[4]}) = {p3} }} {Fore.YELLOW}= {P}'), sleep(3)

    # Set the default p color codes to 'white' as defined below in the p_color list until an if condition applies.
    p_color = [Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE]

    # Set the high & low p's color-coded conditions of the p_color list against the total P color, set to LIGHTGREEN_EX.
    if p1 < 0.5 and p1 >= 0.33:
        p_color[0] = Style.BRIGHT + Fore.WHITE
    if p2 < 0.5 and p2 >= 0.33:
        p_color[1] = Style.BRIGHT + Fore.WHITE
    if p3 < 0.5 and p3 >= 0.33:
        p_color[2] = Style.BRIGHT + Fore.WHITE

    if p1 < 0.33 and p1 >= 0.25:
        p_color[0] = Style.DIM + Fore.YELLOW
    if p2 < 0.33 and p2 >= 0.25:
        p_color[1] = Style.DIM + Fore.YELLOW
    if p3 < 0.33 and p3 >= 0.25:
        p_color[2] = Style.DIM + Fore.YELLOW

    if p1 < 0.25:
        p_color[0] = Style.DIM + Fore.RED
    if p2 < 0.25:
        p_color[1] = Style.DIM +  Fore.RED
    if p3 < 0.25:
        p_color[2] = Style.DIM + Fore.RED

    if p1 > 0.5 and p1 < P:
        p_color[0] = Style.BRIGHT + Fore.LIGHTCYAN_EX
    if p2 > 0.5 and p2 < P:
        p_color[1] = Style.BRIGHT + Fore.LIGHTCYAN_EX
    if p3 > 0.5 and p3 < P:
        p_color[2] = Style.BRIGHT + Fore.LIGHTCYAN_EX

    if ((p1 <= 0.5 and p1 >= 0.4)
        or (p2 <= 0.5 and p2 >= 0.4)
        or (p3 <= 0.5 and p3 >= 0.4)):  # This condition implies to superposition as discussed in Refs. [1, 2],
                                        # as the main discussion around state ⟨2⟩ of qubit pairs from a single
                                        # field vs. a QDF transformation.
        p_color[:] = Style.BRIGHT + Fore.LIGHTMAGENTA_EX # Any color in the p_list is set to that successful event p as magenta.
    if p1 == P or p2 == P or p3 == P:  # This only occurs when a p ⟶ ⟨P_success⟩ = P = 1, as
                                       # 100% success probability of the expected measurement outcome.
                                       # See Ref. [2] publication for more details.
        p_color[:] = Style.BRIGHT + Fore.LIGHTGREEN_EX # Any color in the p_list is set to that successful event p as green.

    # Plot QDF pairwise qubits p's relative to P:
    print(f"{Fore.LIGHTMAGENTA_EX}{hline}\n Plot = [{Fore.LIGHTGREEN_EX}Experimented n of N counts\
    {Fore.LIGHTYELLOW_EX} ∝ {Fore.LIGHTGREEN_EX} p of total circuit events P on pairwise\
    qubits {Fore.LIGHTYELLOW_EX}= P(b|ij)){Fore.LIGHTMAGENTA_EX}]; {Fore.LIGHTYELLOW_EX}|ij⟩ ≡ |q_i q_j⟩ ≡ |qᵢqⱼ⟩
{Fore.LIGHTMAGENTA_EX}.\
    \n{hline}")

    fig = tpl.figure()
    fig.barh([P, p1, p2, p3], [Style.BRIGHT + Fore.LIGHTGREEN_EX+f"P(b|ij))", p_color[0]+f"p1(b|ij)) = p1({bin_list[0]})",
                               p_color[1]+f"p2(b|ij)) = p2({bin_list[2]})", p_color[2]+f"p3(b|ij)) = p3({bin_list[4]})"],
             force_ascii=False)
    fig.show()
```

```
# Store the P's for simulation use by the QFLCC program for dataset analysis and QDF circuit predictions.
with open('ibm-qdf-stats.txt', 'w') as file_to_write:
   file_to_write.write(f"{P}, {p1}, {p2}, {p3}\nP, p({bin_list[0]}), p({bin_list[2]}), p({bin_list[4]})")
   file_to_write.close() # End write counts_exp metadata to the file.

print(Fore.LIGHTMAGENTA_EX +
      "\n\033[4m<<-- IBM QDF Circuit Measurement Results by Qiskit Aer Simulator Plotted Successfully! End of Task... --
>>\033[0m\n"
      + Style.BRIGHT + Fore.LIGHTGREEN_EX + f"\033[1m")

print(f"{Fore.LIGHTGREEN_EX}Press any key to continue...")
os.system("pause >nul")
print(f"\033[F{Fore.LIGHTGREEN_EX + hline}")
```

▼ ≙ IBMQ source codes... cont.

IBMQ cont. source code to draw the quantum circuit and count P's from QDF-LCode_IBM-2024.py

```
#-------------------------------------------------------------------------------------------------
# <----- Display histogram results of the QDF circuit event P's and then printing the circuit ----->
#-------------------------------------------------------------------------------------------------
qdf_circuit() # Call this function to plot P results given the number of shots.

#-------------------------------------------------------------------------------------------------
# Visualize the Circuit in the terminal by printing the circuit... This is in python. IBMQ kernel
# does not require print(), or simply enter qc.draw() in this case. See code from the
# 'QDF-LCode_IBMQ-2024-raw-codable.ipynb' codable file or its 'QDF-LCode_IBMQ-2024.ipynb' base file.
#-------------------------------------------------------------------------------------------------
print(qc.draw())

fcircuit = "ibm-qdf-circuit_output.bin" # The file that the printed circuit is saved.
with open(fcircuit, 'w', encoding='utf-8') as file_to_write:
    file_to_write.write(str(qc.draw()))
    file_to_write.close() # End writing the printed circuit.

print(Back.RESET + Fore.MAGENTA+"\n\033[4m<<-- QDF Circuit BUILT & RAN Successfully! End of Task... -->>\033[0m\n"
       + Fore.LIGHTGREEN_EX), sleep(3)
print(Back.LIGHTGREEN_EX + Fore.YELLOW
      + f"{hline}\n\033[1m<--- {{ Qiskit Aer, IBM }} QDF CIRCUIT SIMULATION CONCLUDED --->\n{hline}\033[0m" + Back.RESET)
#########--END OF PROGRAM--#########
```

▼ ≙ QInspire source codes...

QInspire source code with QDF default circuit config. as QDF-LCode_QI.cq

```
Version 1.0
#-------------------------------------------------------------------------------
# A quantum field lens coding algorithm by a conventional quantum circuit
# written in cQSAM by P. B. Alipour, run on Quantum Computers in QI, 2020-2023
#-------------------------------------------------------------------------------
qubits 6 # Initialize a 6-qubit register (define the number of qubits)
.init
prep_z q[0:5] # State initialization on 6 qubits in the z-basis in the |0>
# state; Initialize inputs to some values
#-------------------------------------------------------------------------------
# Qubit definitions for constructing an adder:
# q[0] --> |A>
# q[1] --> |B>
# q[2] --> CarryIn_SumOut
# q[3] --> CarryOut
#-------------------------------------------------------------------------------
# Initialize inputs A=1, B=0 and CarryIn =1 to perform addition after
# decoding Alice's message about the prize position
{X q[0] | X q[2]}

#-------------------------------------------------------------------------------
# Define the combined .superposition and .entanglement sub-circuit as
# .superpose_entangle for superdense coding
#-------------------------------------------------------------------------------
.superpose_entangle(2) # Iterate up to i=2 times (third party encodes qubits);
# any i>2 times for any other scenario

H q[1] # Create a superposition of two states

cnot q[1], q[2] # Entangle two qubits via CNOT gate

CR q[1], q[4], 0.39 # Controlled phase shift (or T-gate equivalent)
# gate between qubits 1 and 4 with an angle of 0.39 radians = pi/8 degrees
H q[4]

.sender_encode_bits(1) # From superdense coding scheme , Alice sends two
# classical bits as one qubit to receiver
Z q[1]
{Rz q[4], -0.32 | Rz q[5], -1.23}
# (Rz q[5] is the controlled rotation on ancilla qubit 0, mapping to ((pi^2)/8
# + (pi^2)/30) rad or =< pi/2 degrees; The system is evolved to its excited states
# when the measurement on the ancilla qubit is in state |0>. Meanwhile , the system
# is purified to a state that is close to its ground state (steps of decompression
# discussed in the EE measurement section) as measurement takes place to observe
# its excited state |1>.

CR q[4], q[5], 0.39
X q[1]
# .middle_recorder(1) or Eve gathers information form Alice with increased certainty
# and sends info to Bob (see the QDF game model steps)

.receiver_decode_bits(1) # Bob as receiver
# decodes bits within two consecutive steps of the game; below Eve collects information
# from Alice relative to Bob as .middle_recorder(1) which defines a single step of
# kappa -based QDF transformation = a momentum (kinetic energy) operator.
H q[4]
CR q[1], q[4], 0.39
cnot q[1], q[2]
H q[1]
SWAP q[1], q[4] # SWAP gate on qubits 0 and 1 after the quantum field (QDF) transformation

measure_z q[1:2]

# Perform addition
.full_adder(1)
toffoli q[0],q[1],q[3]
cnot q[0],q[1]
toffoli q[1],q[2],q[3]
cnot q[1],q[2]
cnot q[0],q[1]

measure_z q[2, 4, 5]
# Measure the sum output from qubit 2 and the controlled rotation as momentum (kinetic
# energy) operator.
#-------------------------------------------------------------------------------
# Listing 1. Quantum circuit code for the corresponding figure exported from QI
# platform written in QI at: https://www.quantum-inspire.com/, ran on a 26-qubit QX
# single-node simulator. By removing the relevant circuit components from this code,
# the superdense coding circuit for Fig. 4(a) can be reproduced to run on a 5-qubit
# processor, which is implemented between lines No. 5–7, 28–30, 37–49, 58–60.
#-------------------------------------------------------------------------------
```

9 QAI-LCode_QFLCC Source Code

Downloadable source code is:

QAI-LCode_QFLCC file download

▼ Expand/Collapse All…

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.
```

Expand All Collapse All

---

10 QAI-LCode_QFLCC Module

▼ Module description and code usage warning:

The current main module is the source file `QAI-LCode_QFLCC.py`. This module will be split into the `QDF-game.py` and `QAI-LCode_QFLCC.py` for import. The `QDF-game.py` is already within the file `QAI-LCode_QFLCC.py` file. The updated version to import the smaller `QAI-LCode_QFLCC.py` for the `QDF game` is under development as the next version of the code. QAI-LCode_QFLCC file download

10.1 `ExitMyProgram`

Bases: `Exception`

- Exception used to exit program.

▼ Source code in `QAI-LCode_QFLCC.py`

```
104
105
class ExitMyProgram(Exception):
    """- Exception used to exit program."""
```

10.2 `bg`

▼ Source code in `QAI-LCode_QFLCC.py`

```
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
class bg:
        black='\033[40m'
        red='\033[41m'
        green='\033[42m'
        orange='\033[43m'
        blue='\033[44m'
        purple='\033[45m'
        cyan='\033[46m'
        lightgrey='\033[47m'
        """- Colored background (bg) class for sectioning and highlighting the
        QF-LCC algorithm step, checkpoint, computer operation, error, or HALT."""
```

10.2.1 `lightgrey = '\x1b[47m'` `class-attribute` `instance-attribute`

- Colored background (bg) class for sectioning and highlighting the QF-LCC algorithm step, checkpoint, computer operation, error, or HALT.

10.3 `fg`

▼ Source code in `QAI-LCode_QFLCC.py`

```
class fg:
        black='\033[30m'
        silver="\033[0;38;5;7m"
        red='\033[31m'
        green='\033[32m'
        orange='\033[40m'
        blue='\033[34m'
        purple='\033[35m'
        cyan='\033[36m'
        lightgrey='\033[37m'
        darkgrey='\033[90m'
        lightred='\033[91m'
        lightgreen='\033[92m'
        yellow='\033[93m'
        lightblue='\033[94m'
        pink='\033[95m'
        lightcyan='\033[96m'

        """- Colored foreground (fg) class for sectioning and highlighting the
        QF-LCC algorithm step, checkpoint, computer operation, error, or HALT."""
```

10.3.1 `lightcyan = '\x1b[96m'` `class-attribute` `instance-attribute`

- Colored foreground (fg) class for sectioning and highlighting the QF-LCC algorithm step, checkpoint, computer operation, error, or HALT.

10.4 `build(debug)`

- Build production assets.

▼ Source code in `QAI-LCode_QFLCC.py`

```
@cli.command()
@click.option("-d", "--debug", help="Include debug output.")
def build(debug):
    """- Build production assets."""
```

10.5 `cli()`

Main entry point.

▼ Source code in `QAI-LCode_QFLCC.py`

```
@click.group()
def cli():
    """Main entry point."""
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def PAnalysis_model():
 ##########################################################################################################################
 #--- This function compares the mapped results between QDF circuits (IBMQ <--> QI).
 #--- SAFE MODE QFLCC AND,
 #--- QDF Circuit Model from IBMQ article file: QDF-LCode_IBMQ-2024-codable.py,
 # for python library purposes imported as a module:
 #--- QDF-LCode_IBMQ-2024.py (same code content as a duplicate of QDF-LCode_IBMQ-2024-codable, but will be overwritten
 # by IBMQ folder content if you modify or update code. Only edit code within the QDF-LCode_IBMQ-2024-codable.py file!).
##########################################################################################################################
 global sim_state, dir_flag, entry_stage, hline
 sim_state = ['']
 hline =
"=========================================================================================================================="
 print(f'''{Fore.LIGHTMAGENTA_EX + hline}\n An ideal QDF circuit I/O model running realtime producing an IBMQ-based QDF dataset is
from the \
 \n QDF-LCode_IBMQ-2024.py file. This circuit is compared to the analyzed dataset to show how close \
 \n the match is for a desired Hamiltonian and expected measurement outcome as a point of reference.\
 \n{hline}''')
 #-------------------------------------------------------------------------------------------
 # <----- Display histogram results of the QDF circuit event P's and then printing the circuit
 # as a model to display for QDF datasets of other circuits being analyzed here (next version will).
 # Future release will have current P results of QInspire or any other QDF circuit configure the
 # IBM-QDF circuit through a QAI mapping, accordingly (subject of QAI lab for QFLCA) ----->
 #-------------------------------------------------------------------------------------------
 P = 1 # The total probability of QDF circuit events relative to classical states or
       # denoting the system's Hamiltonian (total energy).
 with alive_bar(3, bar = 'blocks', manual=True) as bar:   #To print progress bar on these dataset and circuit simulation analyses.
    print(Fore.LIGHTMAGENTA_EX + Back.LIGHTGREEN_EX+ f"Please Wait!...:")
    print(f"*- Program is processing the analyzed dataset circuit P's after:"+Back.RESET), sleep(1)
    bar(0.5)
    print(f"{Back.LIGHTGREEN_EX}*- Import + Simulate the IBM QDF circuit by\
 {{ Qiskit Aer Simulator, IBMQ Provider }} [QUANTUM MODE ENVIRONMENT] on this computer."+Back.RESET), sleep(1)
    bar(0.7)
    print(f"{Back.LIGHTGREEN_EX}*- Dataset P results are compared on the imported circuit by the {{ QDF-LCode_IBMQ-2024.py }}
module\
 on this computer [SAFE MODE ENVIRONMENT]."+ Back.RESET + Fore.LIGHTMAGENTA_EX), sleep(1)
    bar(1.)

 # Log the start of simulation.
 entry_stage = 0
 sim_state[0] = " QDF CIRCUIT SIMULATION BEGINS "
 sim_log()
 #-------------------------------------------------------------------------------------------------------------------
 ibm_qdf_module = importlib.import_module("QDF-LCode_IBMQ-2024-codable") # Unconventional call from the targeted module.
 bar(1.)
 #-------------------------------------------------------------------------------------------------------------------
 # Log the prompt of simulation to the user to proceed, restart or exit program.
 entry_stage = 1
 sim_state[0] = " QDF CIRCUIT SIMULATION_DATASET_ANALYSIS PROMPT "
 sim_log()
 #-------------------------------------------------------------------------------
 # Selected dataset and the simulated circuit's dataset validation verdict starts here,
 # if user enters 'n' or 'next'...
 # -------------------------------------------------------------------------------
 print(Fore.LIGHTGREEN_EX + f"*- Enter 'n' or 'next' to proceed calculating P's between the selected dataset and the simulated\
IBM QDF circuit results.")
 print(f"🛈 {Fore.LIGHTMAGENTA_EX}\x1B[1;4mThe datasets of the two QDF circuits are compared and validated with a verdict on how\
close their projection of \n events (prediction) is, given their QDF circuit configuration!\033[0m")
 print(f"{Fore.LIGHTGREEN_EX}*-     Enter 'dir' or 'sm dir' to select another dataset to analyze in Safe Mode (SM:>>).\
SIMULATION WILL NOT RESTART!")
 print(f"{Fore.LIGHTGREEN_EX}*-     Enter 'n' or 'next' to continue in Safe Mode (SM:>>).\
QDF CIRCUITS SIMULATION_DATASET_ANALYSIS WILL START!")
 print(f"{Fore.LIGHTGREEN_EX}*- Enter any other key to change from SM:>> to regular prompt mode (>).")
 print(f"*- Enter 'r' in regular prompt mode (>) to 'restart' circuit simulation and dataset analysis. Enter 'h' for more options.

 sm_input = input(f"SM:>> {fg.yellow}")
 if sm_input == "dir" or sm_input == "sm dir":
      dir_flag = 1 # Directory flag is set to 1 for directory access in safe mode.
      safeMode_dir()
      pass
 elif sm_input == "n" or sm_input == "next":
      dir_flag = 1
      print(f"{Fore.LIGHTGREEN_EX}SM:>>{fg.yellow} Next..."), sleep(1)
 else:
      dir_flag = 0  # Directory flag is set to 0 in case of regular prompt mode (>) or 'r' to restart the simulation.
      prompt()

 print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS BEGINS --->\033[0m" + Back.RESET)

 ibmq_result = 'ibm-qdf-stats.txt'
```

```
with open(ibmq_result, 'r') as file:
       # Read the contents of the file
       content = file.read()

       # Extract all the p values from the content as float.
       N = re.findall(r'[-+]?(?:\d*\.*\d+)', content)
       n_list = list(map(float, N))

       # Convert all the metadata digits to str to sort out binaries.
       bin_list = list(map(str, N))  # This is to identify qubit binary strings.

       # Calculate quantum event p's for the total P.
       P = n_list[0]  # The P result is stored to the p_list.
       p1 = n_list[1] # 1st p result is stored to the p_list.
       pq1 = bin_list[4]
       p2 = n_list[2] # 2nd p result is stored to the p_list.
       pq2 = bin_list[5]
       p3 = n_list[3] # 3rd p result is stored to the p_list.
       pq3 = bin_list[6]

ibmq_p_max = max(p1, p2, p3)  # Identify the max value from the p list.
max_p_index = n_list.index(ibmq_p_max) # Store the index value for p_max from the ibmq_result file.

ibmq_p_min = min(p1, p2, p3)  # Identify the min value from the p list.
min_p_index = n_list.index(ibmq_p_min) # Store the index value for p_min from the ibmq_result file.

# Set the default p color codes to 'white' as defined below in the p_color list until an if condition applies.
p_color = [Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE,
           Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE]

if round(p1, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p1).
     p_color[0] = Style.BRIGHT + Fore.RED
if round(p2, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p2).
     p_color[1] = Style.BRIGHT + Fore.RED
if round(p3, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p3).
     p_color[2] = Style.BRIGHT + Fore.RED
if round(p1, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p1).
     p_color[0] = Style.BRIGHT + Fore.CYAN
if round(p2, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p2).
     p_color[1] = Style.BRIGHT + Fore.CYAN
if round(p3, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p3).
     p_color[2] = Style.BRIGHT + Fore.CYAN

print(f"{Fore.LIGHTMAGENTA_EX}{hline}\nPlot = [{Fore.LIGHTGREEN_EX}P samples of the IBMQ model circuit,\
if meets {Fore.LIGHTYELLOW_EX} ∝ {Fore.LIGHTGREEN_EX} p of the selected dataset file\
{{ {Fore.LIGHTYELLOW_EX + fileresult[filenum-1] + Fore.LIGHTGREEN_EX} }}\non pairwise \
qubits{Fore.LIGHTMAGENTA_EX}]...\nThen a strong vs weak p match, and the distance between QDF circuit events\
are determined.{Fore.YELLOW}* {Fore.LIGHTGREEN_EX}\n{hline}{Fore.LIGHTGREEN_EX}\
\n{Fore.YELLOW}* Computation model:{Fore.GREEN} κ Scalar ΨΦ Field Switch and Correlation Model, Ref. [1] of the DIB article.\
\n  {Fore.LIGHTCYAN_EX}P(Ψ ↔  κ²Ψ) = ⟨P(Ψ ↔ Φ)⟩ = P(b|ij)); |ij⟩ ≡ |q_i q_j⟩ ≡ |qᵢqⱼ⟩. {Fore.GREEN}\
\n{hline}"), sleep(0.5)

print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Sampling between IBM QDF Circuit and the Selected QDF Circuit Dataset -->\033[0m''
if dir_flag == 1:  # Read and display the printed circuit when dataset dir is in SAFE MODE.
      with open("ibm-qdf-circuit_output.bin", 'r', encoding='utf-8') as file_to_read:
           print(f'{fg.black}{bg.lightgrey}\n'+file_to_read.read(),"\n<-- IBM QDF Circuit Sample Printed in SAFE MODE --> "
                + Back.BLUE + str(datetime.datetime.now()) + Back.RESET)
           file_to_read.close() # End reading and displaying the printed circuit.
print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Results Sampled from {{ {ibmq_result} }} file are: -->\033[0m''')
with alive_bar(3, bar = 'blocks' , manual=True) as bar:
 model_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities.
 model_fig.barh([P, p1, p2, p3], [Style.BRIGHT + Fore.LIGHTGREEN_EX+"P(Total)", p_color[0]+f"ibmq qdf p({bin_list[4]})",
                           p_color[1]+f"ibmq qdf p({bin_list[5]})", p_color[2]+f"ibmq qdf p({bin_list[6]})"],
        force_ascii=False), sleep(1)
 model_fig.show(), sleep(1)
 print('')
 bar(1)
 dir_flag = 0 # Reset flag until recalled.
 qdf_bit_pairs = ['',''] # To register which qdf_bit_pair has the max or min p.
 #-------------------------------------------------------------------------------
 # Now classify and print which sampled qdf_bit_pair set from the
 # ibmq_result file is max_p and which min_p.
 if min_p_index == 1:
  qdf_bit_pairs[0] = pq1
 if min_p_index == 2:
  qdf_bit_pairs[0] = pq2
 if min_p_index == 3:
  qdf_bit_pairs[0] = pq3
 if max_p_index == 1:
  qdf_bit_pairs[1] = pq1
 if max_p_index == 2:
```

```
    qdf_bit_pairs[1] = pq2
   if max_p_index == 3:
    qdf_bit_pairs[1] = pq3

print(Fore.YELLOW+f'{hline}')
print(fg.purple+f'*- min(P) from {{ {ibmq_result} }} is performed by the QDF bit pairs\
{{ {fg.red+ qdf_bit_pairs[0] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{min_p_index} = {ibmq_p_min}')
print(fg.purple+f'*- max(P) from {{ {ibmq_result} }} is performed by QDF bit pairs\
{{ {fg.lightgreen+ qdf_bit_pairs[1] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{max_p_index} = {ibmq_p_max}')
print(Fore.YELLOW+f'{hline}')
#-------------------------------------------------------------------------------------------------------------
# Recall and store which binary string had the minimum P from the table of the selected dataset (dataframe).
global df_min, df_min_bin
if  (sample_data.min(axis=0)[1] < 1):
          df_min=df.loc[(df['probability'] <= sample_data.min(axis=0)[1]), :].binary_string[:]
          df_min_bin = df_min.to_string(index=False, header=False)
#-------------------------------------------------------------------------------------------------------------
deltaMatch_max = (1 - abs(ibmq_p_max - float(df3Mem)) ) # Calculate Δp of max(P) Match.
deltaMatch_min = (1 - abs(ibmq_p_min - sample_data.min(axis=0)[1]) ) # Calculate Δp of min(P) Match.
delta_p_min = abs(ibmq_p_min - sample_data.min(axis=0)[1]) # Calculate Δp of min(P).
delta_p_max = abs(ibmq_p_max - float(df3Mem)) # Calculate Δp of max(P).

Valid_Verdict = ['Weak',f'{{∞ , "/ₐ}}', 'Avg.', 'Strong', 'NUL']  # Validation Verdict of a strong
                                                                  # vs. weak correlation match between
                                                                  # the selected file dataset p's and the IBM QDF
                                                                  # circuit event p's...
# Note: A verdict of NUL denotes missing information or unknown about the max or min of the p's
# (calculable if config is corrected or complemented).

if round(deltaMatch_max, 2) >= 0 and round(deltaMatch_max, 2) < 0.5: # Classify/color code max Δp match.
     p_color[3] = Style.BRIGHT + Fore.RED
     Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

if round(deltaMatch_max, 2) >= 0.5 and round(deltaMatch_max, 2) <= 0.55: # Classify/color code max Δp match.
     p_color[3] = Style.NORMAL + fg.silver
     Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

if round(deltaMatch_min, 2) >= 0 and round(deltaMatch_min, 2) < 1/3: # Classify/color code min Δp match.
     p_color[4] = Style.BRIGHT + Fore.RED
     Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

if (round(deltaMatch_min, 2) <= 1/3 and
    round(deltaMatch_max, 2) <= 1/3) or (round(deltaMatch_min, 2) <= 1/2 and
                                         round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
     p_color[3] = Style.NORMAL + fg.silver
     p_color[4] = p_color[3]
     Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

if round(deltaMatch_max, 2) > 0.55 and round(deltaMatch_max, 2) < 0.66: # Classify/color code max Δp match.
     p_color[3] = Style.BRIGHT + Fore.LIGHTYELLOW_EX
     Valid_Verdict = Fore.LIGHTYELLOW_EX + Valid_Verdict[2]

if (round(deltaMatch_max, 2) >= 0.66 and
    round(deltaMatch_max, 2) < 0.9) and (round(deltaMatch_min, 2) >= 0.66 and
                                         round(deltaMatch_max, 2) < 0.9): # Classify/color code min-max Δp match.
     p_color[3] = Style.BRIGHT + Fore.LIGHTMAGENTA_EX
     p_color[4] = p_color[3]
     Valid_Verdict = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]

if (round(deltaMatch_max, 2) >= 0.9 and
    round(deltaMatch_max, 2) <= 1) and (round(deltaMatch_min, 2) >= 0.9 and
                                        round(deltaMatch_max, 2) <= 1): # Classify/color code min-max Δp match.
     p_color[3] = Style.BRIGHT + Fore.LIGHTGREEN_EX
     p_color[4] = p_color[3]
     Valid_Verdict = Fore.LIGHTGREEN_EX + Valid_Verdict[3]

if (round(deltaMatch_min, 2) <= 1/3 and
    round(deltaMatch_max, 2) <= 1/2) or (round(deltaMatch_min, 2) <= 1/2 and
                                         round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
     p_color[3] = Style.NORMAL + fg.silver
     p_color[4] = p_color[3]
     Valid_Verdict = fg.silver + Valid_Verdict[4] # See note for a NUL verdict!

PAnalysis_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities of the two datasets.
PAnalysis_fig.barh([P, float(df3Mem), float(dfcompMem), sample_data.min(axis=0)[1], ibmq_p_min, ibmq_p_max,
                    delta_p_min, delta_p_max, deltaMatch_min, deltaMatch_max],
    [Style.BRIGHT + Fore.LIGHTGREEN_EX + f"P(\x1B[4mTotal\x1B[0m{Style.BRIGHT + Fore.LIGHTGREEN_EX})",
     f'{Style.BRIGHT+Fore.WHITE}{{ {fileresult[filenum-1]} }} max(P(\x1B[4m{df2bin}\x1B[0m{Style.BRIGHT+Fore.WHITE}))',
     f'{Style.RESET_ALL + Fore.WHITE}{{ {fileresult[filenum-1]} }} comp(P(\x1B[4m{bitcomp}\x1B[0m{Style.DIM+Fore.WHITE}))',
     Style.BRIGHT + Fore.WHITE + f"{{ {fileresult[filenum-1]} }} min(P(\x1B[4m{df_min_bin}\x1B[0m{Style.BRIGHT + Fore.WHITE}))",
     f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} min((P(\x1B[4m{qdf_bit_pairs[0]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
```

```
      f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} max((P(\x1B[4m{qdf_bit_pairs[1]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
      p_color[4] + "Δp of \x1B[4mmin(P)\x1B[0m" + p_color[4],
      p_color[3] + f"Δp of \x1B[4mmax(P)\x1B[0m" + p_color[3],
      p_color[4]+ "Δp of\x1B[4m min(P) match\x1B[0m" + p_color[4],
      p_color[3]+ "Δp of\x1B[4m max(P) match\x1B[0m" + p_color[3]], force_ascii=False), sleep(1)
 PAnalysis_fig.show()
 bar(1.)

 print(f"{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- Δp Data:      \033[4mValidation Verdict\033[0m    {fg.yellow}  \
 between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
 \n   is a/an: \
\033[4m{Valid_Verdict + ' Match {{ '+ str(round(deltaMatch_min, 2)) +' , '+ str(round(deltaMatch_max, 2)) +' }}'}\033[0m\
{Fore.LIGHTYELLOW_EX}")

 DeltaP_verdict = Valid_Verdict
 Valid_Verdict = ['Weak',f'{{∞ , ⁿ/ₐ}}', 'Avg.', 'Strong', 'NUL']  # Reset Valid_Verdict list elements upon
                                                                  # the previously rendered verdict.
 #-------------------------------------------------------------------------------------------------
 # Validate circuit config. based on recorded P's associated to their qubit pair binaries (strings)
 s1 = df_min_bin
 s2 = qdf_bit_pairs[0]
 s3 = df2bin
 s4 = qdf_bit_pairs[1]
 vv_circuits =[False]
 vv_bins = [False]

 # Validation Verdict on the two circuits qubit string sets over min and max P's as compared for a match.
 if re.match(s2, s1):
     vv_bins = fg.lightgreen+ Valid_Verdict[3]
     qdf_s_match = f"{{ {s1} , {s2} }}"
 elif re.match(s4, s3):
     vv_bins = fg.lightgreen + Valid_Verdict[3]
     qdf_s_match = f"{{ {s3} , {s4} }}"
 else:
     vv_bins = fg.red + Valid_Verdict[4] # See note for a NUL verdict!

 # Validation Verdict on the two circuits as compared for a config. match.
 if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
     vv_circuits = fg.green + Valid_Verdict[3]
 if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[1]):
     vv_circuits = fg.yellow + Valid_Verdict[2]
 if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]):
     vv_circuits = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]
 if (vv_bins == fg.red + Valid_Verdict[4]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
     vv_circuits = fg.yellow + Valid_Verdict[2]
 if ((vv_bins == fg.red +
      Valid_Verdict[4]) or (vv_bins == fg.lightgreen + Valid_Verdict[3])) and (DeltaP_verdict == fg.silver + Valid_Verdict[4]):
      vv_circuits = f'{{ {fg.lightred + Valid_Verdict[4]} , {False} }}' # A NUL or false match verdict! See also next line/condition
 if (float(df3Mem) <= 1/3 and sample_data.min(axis=0)[1]<= 1/3) and (ibmq_p_max >= 1/2):
     vv_circuits = f'{fg.lightred}{{ {Valid_Verdict[4]} , {fg.yellow}{Valid_Verdict[1]}{fg.lightred} }}' # A NUL match verdict! In
short,
      # a state transition matrix with all elements between certain circuit components return a 0 state or null match and
      # state 2 for one of the two circuits! See P's preliminary analysis for the measured QDF circuit.
      # This circuit event outcome can also be interpreted as a rejected match between the two circuits, based on their
configuration.
 if (vv_bins == fg.red + Valid_Verdict[4]) and (Valid_Verdict == Fore.LIGHTYELLOW_EX + Valid_Verdict[2]):
     vv_circuits = fg.red + Valid_Verdict[2]
 #-------------------------------------------------------------------------------------------------
 print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Qubit Sets:      \033[4mValidation Verdict\033[0m
  {fg.yellow}  \
 between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
 \n   is a/an: \033[4m{vv_bins + ' Match ' + qdf_s_match}\033[0m {Fore.LIGHTYELLOW_EX}")

 print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Circuits:      \033[4mValidation Verdict\033[0m
  {fg.yellow}  \
 between the two QDF circuit configurations from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
 \n   is a/an: \033[4m{vv_circuits} Match \033[0m {Fore.LIGHTYELLOW_EX}")

 # Log validation verdict results on simulation and dataset analysis.
 entry_stage = 2
 sim_state[0] = f"Δp Data: {Valid_Verdict}, QDF Qubit Sets: {vv_bins + ' Match ' + qdf_s_match},  QDF Circuits: {vv_circuits} Matc
 sim_log()

 print(f"{hline + Fore.LIGHTMAGENTA_EX}\n *- The IBM QDF circuit can be reconfigured for worst and best case Hamiltonian scenarios
 \n    given the expected QDF measurement outcomes from the collected QFLCA datasets for a QFLCC.\
 \n\x1b[38;5;226m Simulation completed on: \033[1;32m{fg.yellow+bg.blue+str(datetime.datetime.now())+Back.RESET} \
 \n{hline + Fore.RESET}"), sleep(4)

 dir_flag = 0
 print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED --->\033[0m"
       + Fore.LIGHTGREEN_EX + Back.RESET)
```

```
# Log the end of simulation and dataset analysis.
entry_stage = 3
sim_state[0] = " QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED "
sim_log()
```

▼ Expand/Collapse All. . .

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

11 QAI-LCode_QFLCC Functions and Calls

▼ ≡ QFLCC intro, animation, help, website, logging, sound and exit functions. . .

To prompt and help user to select and enter an option to change e.g., sound volume, select file(s) to install and exit program when asked by the user. Function() list from top to bottom as ordered:

▼ site_doc ( ) function website code:

To show the QFLCA website about its project files and program codes.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
147  def site_doc():
148  ################################################################
149  # To load and read QFLCA project documentation website.
150  ################################################################
151      global site_dir, site_name
152      site_file = ["index.html", "about.html", "QAI-LCode_QFLCC-reference.html",
153               "QDF-game-reference.html"] # This html file is stored in the same site folder to read.
154      site_dir = "site" # This site folder contains project's documentation html files to read.
155      winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS) # Play sound.
156      if site_name == "home-site":
157           webbrowser.open_new_tab(os.path.join(site_dir, site_file[0]))
158      elif  site_name == "about-site":
159            webbrowser.open_new_tab(os.path.join(site_dir, site_file[1]))
160      elif  site_name == "qflcc-site":
161            webbrowser.open_new_tab(os.path.join(site_dir, site_file[2]))
162      elif  site_name == "game-site":
163            webbrowser.open_new_tab(os.path.join(site_dir, site_file[3]))
```

▼ clear_line ( ) function animation code:

To jump back within the CLI and reenter images stored from an array in a sequence.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
259
260
261
262
263
264
```

```
def clear_line(wid):
    global spaces
    LINE_UP = '\033[1A'
    LINE_CLEAR = '\x1b[2K'
    for spaces in range(wid):
      print(LINE_UP, end=LINE_CLEAR)
```

▼ intro_anim ( ) function animation code:

To load and play 1D QDF circuit images in the terminal from an array in a sequence.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def intro_anim():
     global chars
########################################
# Calling this function will run
# an animated intro image displayed
# on terminal!
########################################
     # Enable the following commented code lines if you switch to chafa.loader..
     # The active code lines use both chafa.loader and
     # the Pillow (PIL) package definitions/functions. Only comment out when indicated..
     #-------------------------------------------------------------------------------
     #FONT_HEIGHT = 12*6  #24 # Default value for image height >= image width.
     #FONT_WIDTH  = 26*2  #11 # Default value for image height >= image width.
     #FONT_RATIO = (FONT_WIDTH / FONT_HEIGHT) # A scalar ratio to keep dimensions fixed by product multiplication.

     config = CanvasConfig()

     #config.height = 9 #60 # Default value for a square image.
     #config.width  = 160 #60 # Default value for a square image.

     # Set the canvas cell geometry.
     #config.cell_height = FONT_HEIGHT
     #config.cell_width  = FONT_WIDTH

     # Create a list of image objects
     img_list=  ["./__img__/qdf-circuit-sprite/1D-circuit-00.png", #32-bit depth images
                 "./__img__/qdf-circuit-sprite/1D-circuit-a.png", "./__img__/qdf-circuit-sprite/1D-circuit-b.png",
                 "./__img__/qdf-circuit-sprite/1D-circuit-c.png", "./__img__/qdf-circuit-sprite/1D-circuit-d.png",
                 "./__img__/qdf-circuit-sprite/1D-circuit-d.png"]

     # Loop through the gif image objects for a while.
     for j in range(0, 6):
      for img in img_list:
          print("\033[1A", end="\x1b[2K", flush= True), sleep(0.001)
          #image = Loader(img_list[j])   # Open image with chafa.loader or Loader.
          image = Image.open(img_list[j]) # Open image with PIL. Not used by Loader.

          wid, hgt = image.size
          mode = image.mode  # Detect image mode on your system.

          # When PIL package definitions are used. Comment out if using chafa.loader.
          if (img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-02.png" or
              img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-03.png"):
               config.height = (hgt/8.8)
               config.width  = wid/8.8
          else:
            config.height = hgt/7.8
            config.width  = wid/6.6

          #config.height = (hgt/6.4) - 1
          #config.width  = wid/7.2
          # Enable one of the switches to configure canvas to draw a high resolution image (for chafa.loader).
          #-------------------------------------------------------------------------------------------------
          #config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_SIXELS  # To display a high resolution image
          #config.pixel_mode = chafa.PixelType.CHAFA_PIXEL_RGBA8_UNASSOCIATED  # To display according to
          # terminal's output controls use config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_KITTY
          # to display a high resolution image.
          #config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_KITTY
          #config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_SYMBOLS  # To add symbols to image
          #config.pixel_mode = chafa.PixelType.CHAFA_PIXEL_BGRA8_UNASSOCIATED
          #-------------------------------------------------------------------------------------------------

          # Configure the ideal canvas geometry based on our FONT_RATIO, when using chafa.loader or PIL.
          #config.calc_canvas_geometry(
          #image.width,
          #image.height,
          #FONT_RATIO
          #)

          # Set symbol map for visibility:
          # add 0's and 1's and quantum states to the map, relevant to the topic of this QFLCA program
          symbol_map = chafa.SymbolMap()
          symbol_map.add_by_range("a", "f")

          bands  = len(image.getbands())

          # Put image into correct format
          pixels = image.tobytes()
          canvas = Canvas(config)

```

```
            # Draw the intro image or graphics, when using chafa.loader.
            '''canvas.draw_all_pixels(
            image.pixel_type,
            image.get_pixels(),
            image.width, image.height,
            image.rowstride
                 )'''

            # Draw the intro image or graphics, when loading image using PIL.
            canvas.draw_all_pixels(
            PixelType.CHAFA_PIXEL_RGBA8_PREMULTIPLIED, # Other example is .CHAFA_PIXEL_RGBA8_UNASSOCIATED,
                                                       # given the image mode.
            pixels,
            image.width, image.height,
            image.width * bands
                 )
            #-----------------------------------------
            #  Drawing Chars on Canvas.
            #-----------------------------------------
            QP_rand = random.random()  # Returns a random number between 0 and 1.
            CP_rand = random.randint(0, 1) # Returns a random number denoting the classical state.
            Int_rand= random.randint(0, 3) # Returns a random number denoting the quantum state.

            for pixel in canvas[4,::10]:
                 pixel.char = "1"

            for pixel in canvas[5,::10]:
                 pixel.char = "0"

            #i = 0  # For drawing a sequence of chars or numbers on canvas.
            #for pixel in canvas[:-3:-1, 3]:
            #     pixel.char = str(i)[0]
            #     i += 10

            print(canvas[3,55].char)
            canvas[3,55].char = "H" # Denotes the H gate in the partial QDF circuit.
            ## Superposition state is obtained by Hadamard gate (H) by
            ## H|0⟩ =(|0⟩ +|1⟩ )/sqrt(2) and H|1⟩ =(|0⟩ -|1⟩ )/sqrt(2).

            s = str(round(QP_rand, 2)) # Round the random value to two decimal points and convert to string.
            c = 7 # Column cell number to draw a char.

            if img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-05.png":
                 s = "0.66"  # This is the expected P value >= 2/3 for a strong QDF prediction.

            for r in s[0:3]:
                 canvas[c,55].char = "0"
                 canvas[c,56].char = "."
                 canvas[c,57].char = r # Draw the first three chars on canvas.

            canvas[3,82].char = "X"  # Denotes the X gate in the partial QDF circuit.
            canvas[c,82].char = CP_rand

            #-------------------------------------------------------------------------------
            # The following is for terminal output and image info printouts per
            # a cleared terminal line.
            #-------------------------------------------------------------------------------
            # clear_line(config.height) # Disable this if you want to see all frames in a sequenced list.
            clear_line(config.width) # Disable this if you want to see all frames in a sequenced list.

            output = canvas.print()  # Default option.

            #-------------------------------------------------------------------------------
            # Write and Print images on the terminal.
            #-------------------------------------------------------------------------------
            # Write picture
            print(output.decode(), Back.RESET), sleep(0.05)  # without decoding is raw data of the image.

            #if QP_rand > 0.5:   # Qubit P values printed in color for the corresponding circuit gates.
            #     print(fg.green + s, Back.RESET)
            #elif QP_rand <= 0.5 and img != "./__img__/1D-QDFcircuit-sprite/1D-circuit-05.png":
            #     print(fg.red + s, Back.RESET)
            #elif QP_rand <= 0.5 and QP_rand > 0.48: # Indicates P values approximating superposition.
            #     print(fg.purple + s, Back.RESET)
            #elif img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-05.png":
            #     print(fg.green + s, Back.RESET)

            # Print animated P's of gates in the 1D QDF circuit after logging all the animation frames.
            file='intro-shell_output.txt' # The file to read/write from the terminal.
            chars.append(s)
            chars = list(dict.fromkeys(chars)) # Removes duplicate values from recurring image frames.
```

```
        with open(file, 'w') as file_to_write:
             # Write information about the original image sequence and final image frame.
             file_to_write.write(str(chars) + f"\n * Last animated step image: {img}" +
                                 f"\n * Size: {wid} x {hgt} pixels" + f"\n * Mode: {mode}\n"
                                 + str(datetime.datetime.now())) # date output entry to the file.
```

▼ intro_anim_print ( ) function animation end code:

Prints the stats information on the intro animation frames after successfully concluded.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def intro_anim_print():
#-----------------------------------------------
# This function prints the intro_anim results
# from its log file upon animation conclusion.
#-----------------------------------------------
  with open("intro-shell_output.txt", "r") as input_file:
    for i in range(1):
         head = next(input_file).strip()
         print(fg.black+ bg.orange+"Animated P's of the QDF circuit, frame-by-frame = "+Back.RESET)
         print(fg.yellow, head)

    lines = input_file.readlines()[:-1]

    # Clean up the newlines in the lines of data.
    lines = [line.rstrip('\n') for line in lines]

    # Split each line at the "," character.
    lines = [line.split(',') for line in lines]
    print(fg.lightgreen, lines, bg.green +fg.yellow + Back.RESET)
    print(bg.blue + Fore.BLACK + str(datetime.datetime.now()) + Back.RESET)
  input_file.close()

  print(bg.red+fg.yellow +
       f"<<........ 1D QDF circuit animation as an update to the QFLCA-QFLCC v.1.0, has successfully concluded
........>>")
  print(f"<<........ Next... QFLCA Program's Prompt & Command Begins! ........>>", Back.RESET)
  print(Fore.LIGHTYELLOW_EX + f"========================================================================", Back.RESET)
```

▼ QFLCC_program ( ) function code:

To end the current QFLCC program step engaged by the user.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def QFLCC_program():
    while True:
        if random.randint(1, 1000) == 500:
            raise ExitMyProgram
        """- QFLCC program steps prior are more complicated than shown here. This is to
        end the current QFLCC program step engaged by the user."""
```

▼ main_exit ( ) function code:

Run the ending of the current QFLCC program step.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def main_exit():
    ######################################################
    # Run the ending of the current QFLCC program step
    # and catch the ExitMyProgram exception.
    ######################################################
    global entry_stage, sim_state # Any exit entry of the program is reset back to 0.

    try:
        QFLCC_program()
    except ExitMyProgram:
         print(fg.yellow+Back.RED+"The program is terminated manually!")
         # Log the exit of the program.
         entry_stage = -1 # This value resets entry to 0 via its
                          # default increment += 1.
         entry_stage += 1
         file_ = open('shell_output.txt', 'a')
         now = datetime.datetime.now() # Date the I/O file entry
         subprocess.run("echo {}- Checkpoint logged on {} for all programs: \
///-PROGRAM TERMINATED--///".format(entry_stage, now.strftime("%Y-%m-%d %H:%M:%S")),
shell=True, stdout=file_)
         raise SystemExit
```

▼ set_volume ( ) function code:

Sets volume according to desktop's master volume.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def set_vol(new_volume):
#################################################################
# The volume function sets volume according to your OS.
# This option is available as 'v' or 'volume' during program.
#################################################################
    p.press('volumedown', presses = 50) # Sets volume to zero.
    time.sleep(0.5) # Using time.sleep to space the presses.
    x = math.floor(new_volume / 2) # Setting the amount of presses required.
    p.press('volumeup', presses = x) # Setting the volume.
    winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS) # Sound test.
    """- End of volume settings."""
```

▼ prompt ( ) function code:

Prompting code for a user input.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def prompt():
#####################################
# Prompting code for a user input #
#####################################
    global __help__, promptIn, site_name, entry_stage # File state/flag is 0 or 1
    # to see if dataset files are present and to be copied to the current directory.
    global dir_flag # Directory flag for dataset directory access in safe mode when set to 1.
    __help__ = 'h'

    while True:
        print(fg.lightgreen + "\r< ", end=""), sleep(1.1)
        print("\r> ", end=""+Fore.RESET), sleep(1.1)
        try:
             promptIn = str(input())
             # Respond to the user's choice or command.
             if promptIn == 'n' or promptIn == 'next':
                print("Next...\n")
                break
             elif promptIn == 'v' or promptIn == 'volume':
                vol = float(input("Input sound volume (° ͜ʖ °)  🕪  between [    = 0 for muted,\
 and     = 100 for loudest]:"))
                set_vol(vol)
                prompt() # Restart.
                if vol < 0:
                       print(fg.red +"Out of range or wrong value entered! Readjusted to muted or 0!")
                       prompt() # Restart.
                if vol > 100:
                       print(fg.red +"Out of range or wrong value entered! Readjusted to maximum or 100!")
                       prompt() # Restart.
                break
             elif promptIn == 'b' or promptIn == 'begin' or promptIn == 'r' or promptIn == 'restart':
                print('Restarting program...  (° ͜ʖ °)  ')
                subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program.
                break
             elif promptIn == 'cls' or promptIn == 'clear':
                os.system('cls') # Clear screen.
             elif promptIn == 'website' or promptIn == 'site' or promptIn == 'about' or promptIn == 'web':
                site_name = "about-site" # This html file is stored in the same site folder to read.
                site_doc() # Load website.
             elif (promptIn == 'dir' or promptIn
                   == 'sm dir') and (dir_flag == 1): # Active only in safe mode directory environment when set to 1.
                safeMode_dir() # Run safe mode directory environment during simulation and dataset anlysis.
                dir_flag == 0 # Reset flag until next time requested by the user to access dataset files.
                break
             elif (promptIn == 'dir' or promptIn == 'sm dir') and (dir_flag == 0):
                    print(f"    This command is supported only in the QFLCA's Safe Mode (SM:>>) environment!\
\n\x1B[4m-Enter 'dir' at the dataset analysis & simulation stage after QFLCC directory installation!\x1B[0m"
+ fg.lightgreen)
                    continue
             elif promptIn == __help__ or promptIn == 'help':
                userHelp()  # Show help.
                continue
             elif promptIn == 'e' or promptIn == 'exit':
                # Play Windows exit sound.
                winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
                print('Exiting program...    (° ͜ʖ °)    Goodbye!')
                entry_stage = 0
                main_exit() # Terminate program.
             else:
                print('Input command or response. For help, enter \'h\'... ')
                continue
        except ValueError:
             main_exit()
             print("Invalid input. Please enter a response.")
```

- ▼ userHelp ( ) function code:

  Helping tips code for the user.

  - ▼ Source code in `QAI-LCode_QFLCC.py`

```
def userHelp():
     # Play Windows question sound and display recalled tips to the user.
     winsound.PlaySound("SystemExclamation", winsound.SND_ALIAS)
     print('Input tips: \n-Enter \'n\' key for the \'next\' message or input. \n-Enter \'h\' or \'help\' \
to display these tips. \n-Enter a file number when an uploaded file list is displayed to view result. \
\n-Enter \'dir\' or \'sm dir\' at the dataset analysis & simulation stage after QFLCC directory installation.*\
\n*-This command is supported only in the QFLCA\'s Safe Mode (SM:>>) environment!\
\n-Enter \'site\' or \'web\' to view the \'website\' \'about\' this program or project. \
\n-Enter \'b\' or \'r\' to \'restart\' or \'begin\' program. \n-Enter \'v\' or \'volume\' for sound volume \
change during program. \n-Enter \'cls\' or \'clear\' to clear screen. \n-Enter \'e\' to \'exit\' the program.')
```

▼ Expand/Collapse All…

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

▼ ≙ QFLCA/QFLCC dataset files installation, P analysis and results for a selected file…

To prompt user, log steps, file installation, dataset file content analysis (P's), results and errors. Function() list from top to bottom as ordered:

▼ ≙ QFLCC prompting user code to call prompt ( )…

```
##################################
# Prompting step starts from here
##################################
print(Back.LIGHTBLACK_EX + fg.lightgreen+ f"When this input signal, > or < switch appears: ")
print("it means you can input a keyboard character then press [Enter], or input response, then \
press [Enter] to a program query. For more information, enter \'h\'...")

def userHelp():
     # Play Windows question sound and display recalled tips to the user.
     winsound.PlaySound("SystemExclamation", winsound.SND_ALIAS)
     print('Input tips: \n-Enter \'n\' key for the \'next\' message or input. \n-Enter \'h\' or \'help\' \
to display these tips. \n-Enter a file number when an uploaded file list is displayed to view result. \
\n-Enter \'dir\' or \'sm dir\' at the dataset analysis & simulation stage after QFLCC directory installation.*\
\n*-This command is supported only in the QFLCA\'s Safe Mode (SM:>>) environment!\
\n-Enter \'site\' or \'web\' to view the \'website\' \'about\' this program or project. \
\n-Enter \'b\' or \'r\' to \'restart\' or \'begin\' program. \n-Enter \'v\' or \'volume\' for sound volume \
change during program. \n-Enter \'cls\' or \'clear\' to clear screen. \n-Enter \'e\' to \'exit\' the program.')

process = Thread(target = prompt)  # Invoke prompt() function by creating and executing threads
process.start()
#__space__ = input("Input something:")
#if input():
process.join() # Wait for the thread to terminate

if entry_stage == 0:  # This makes sure when main_exit is called, program exit is executed without bypassing it.
     main_exit()

"""- QFLCC code continues."""
#-----------------------------------------------------------------------------------------------
# Code Use and Copyright Information
#-----------------------------------------------------------------------------------------------
__ORCID__ = "ORCID: https://orcid.org/0000-0003-1037-018X"
__copyright__ = "Copyright © 2022--2024"
__license__ = "Creative Commons (CC BY-NC-ND)"
__author__ = "Philip B. Alipour"
__location__ = "@ ECE Dept. University of Victoria, Victoria BC, Canada"
__version__ = "ver. 1.0"
__description__ = "This QFLCC version of the QAI-LCode is basic. More revisions to come, as the \
dataset grows on the number of trials on 1 or more QDF circuits occurs relative to improvements \
of dataset representation of quantum and classical parameters in observing a thermodynamic system \
based on Refs. [1-3,5] of the Data in Brief, Elsevier J article."
print(Back.BLUE + fg.lightgreen + __description__, "\n")
print(Back.RESET+"Copyright information in Python code:\n")
print("\033[0m"+__license__ , __author__, __location__+',', fg.blue+"\x1B[4m"+__ORCID__+"\x1B[0m",
      __copyright__, __version__, fg.pink+bg.lightgrey)
sleep(5)

########################################
# Main dataset file directory installation
# and listing steps starts here.
########################################
from tqdm import tqdm  # This is for progressbar display.

ibmq_f=0 # File flag set to 0 by default for IBMQ files if as not IN
qi_f=0   # File flag set to 0 by default for QInspire files as not IN
dir_flag = 0 # Directory flag is set to 1 for dataset directory access in safe mode.
           # Installing files is building the directory, so it is set to 0 here.

print(os.getcwd())

ppath = Path(os.getcwd()).parent # This points to the parent path of current directory.
cpath = Path(os.getcwd())  # This is the current path.
print(ppath.absolute())
IBMQ_dir = os.path.join(ppath, 'IBMQ\\test\\')
QI_dir = os.path.join(ppath, 'QI\\test\\')
QI_files = os.listdir(QI_dir) # Fetching the list of all the files.
IBMQ_files = os.listdir(IBMQ_dir) # Fetching the list of all the files.
print('Importing IBMQ/QI data files: ', IBMQ_files, QI_files)  # Prints from e.g., C:/here/my_dir

pool = ThreadPoolExecutor(max_workers)

file_ = open('shell_output.txt', 'w+')
subprocess.run("echo ///--- QFLCC Shell Log_file START ---///", shell=True, stdout=file_)

now = datetime.datetime.now() # Date each I/O file entry.

try:
   # with MultithreadedCopier(max_threads=16) as copier: # this is to achieve e.g., ~35x speedup on a 32 core machine.
   # with ThreadPool(4) as copier:
    with pool as executor:
            for file_name1 in QI_files:
             future = executor.submit(shutil.copy, QI_dir + file_name1, cpath)  # future = pool.submit(my_task, argument)
                                                                                # does not block.
             qi_f=1
            for file_name1 in IBMQ_files:
                future = executor.submit(shutil.copy, IBMQ_dir+file_name1, cpath)
                ibmq_f=1
            value = future.result() # This one blocks.
            print(Back.RED + fg.black
                  + f'Thread ---> <{value}> <--- got {future}. Maximum threads initiated: {int(max_workers)}')
            pool.shutdown()
```

```
    print('All IBMQ/QI data files imported successfully!')
    #---------------------------------
    # File directory listing code
    #---------------------------------
except Exception as e:
    for i in tqdm(range(0, 10), ncols = 100, desc =Back.RESET+"Progress: "+Fore.RED): # Progress level of a copy operation.
        time.sleep(.05)
    print('1 or more IBMQ/QI data files import failed!')
    qi_f=0
    ibmq_f=0
    print(e)

print(Back.RESET+'||||||||||||||||||||||||||||||||||||||||') #, fileresult, sep="\n")
print(Fore.YELLOW + 'You may view any of the files listed below by clicking on the file or [Ctrl + Mouse_click] option:')

res = os.listdir()
fileresult = []

for (idx, st) in enumerate(res, 1):
    print(bg.green+'This is file # {} for {}'.format(idx, st.split('/')[-1]))
    time.sleep(.05)
    fileresult.append(st)
    for i in tqdm(range(0, 1), ncols=100, desc =Back.RESET+f"Progress: {st}"+fg.lightcyan + Style.BRIGHT): # Progress level of copy operation.
     # For any error related to this line on colors, install this: pip install tqdm==4.59.
     time.sleep(.05)
     subprocess.run("echo {}- Checkpoint logged on {} for file # {} \
as {}".format(idx, now.strftime("%Y-%m-%d %H:%M:%S"), idx, fileresult[idx-1]), shell=True, stdout=file_)

#res=os.listdir().index() # Enable this line (with its print next) to have clickable files to load and view
                         # on screen (not in terminal).

print(Style.RESET_ALL + fg.orange + bg.cyan +"||||||||||||||||||||||||||||||||||||||||")
#print(*res, '||||||||||||||||||||||||||||||||||||||||', f'The current directory total listed files = {len(res)}', sep="\n")
print('||||||||||||||||||||||||||||||||||||||||',
      f'The current directory total listed files = {len(res)}', sep="\n"), sleep(2)
print("||||||||||||||||||||||||||||||||||||||||"+Back.RESET)

#---------------------------------
# File directory listing code cont.
#---------------------------------
entries=[]
```

▼ file_reader ( ) function code:

File directory listing code to read specific files in the installed directory: .csv, .py, .txt, .json, .html. For analysis, .csv is the main dataset to read and analyze.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def file_reader():
#---------------------------------------------------------------
# File directory listing code cont. is by calling this function.
#---------------------------------------------------------------
 global entries, selected_path, selection
 while True:
  entries = []
  # Build a list of files and folders in the working directory.
  current_folder = Path.cwd()
  # If we're not already in the root, the first entry is the parent folder.
  if current_folder.parent != current_folder:
    entries.append(Path('..'))
  entries += current_folder.glob('*')
  # Print a numbered list of files and folders
  print(fg.cyan+f'Listing contents of {current_folder}')
  for i, path in enumerate(entries):
    print(f'{i}: {path.name}')
  # Get user input. Valid input is the word 'quit' or a number that corresponds
  # to an index in the list of files and folders
  print(fg.lightgreen + "Select a file # to see its contents in form of text before analysis, or enter \
\'n\' for next to analyze file contents:")
  print(fg.lightgreen + "\r< ", end=""), sleep(1.1)
  print("\r> ", end=""+Fore.RESET), sleep(1.1)
  user_input = input('')
  if user_input == 'n' or user_input =='next':
    break
 elif user_input == '' or user_input =='h':
    userHelp() # Give user prompting tips.
    prompt() # Prompt user.
 elif user_input == 'v' or user_input == 'volume':
              vol = float(input("Input sound volume (° ͜ʖ °)  🕪  between [    = 0 for muted,\
and     = 100 for loudest]:"))
              set_vol(vol)
              file_reader() # Restart.
              if vol < 0:
                     print(fg.red +"Out of range or wrong value entered! Readjusted to muted or 0!")
                     file_reader() # Restart.
              if vol > 100:
                     print(fg.red +"Out of range or wrong value entered! Readjusted to maximum or 100!")
                     file_reader() # Restart.
 elif user_input == 'b' or user_input == 'begin' or user_input == 'r' or user_input == 'restart':
                print('Restarting program...  (° ͜ʖ °)  ')
                subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program.
                break
 elif user_input == 'cls' or user_input == 'clear':
                os.system('cls') # Clear screen.
 elif user_input == 'e' or user_input == 'exit':
                # Play Windows exit sound.
                winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
                print('Exiting program...    (° ͜ʖ °)    Goodbye!')
                print(fg.yellow+Back.RED+"The program is terminated manually!"+Back.RESET)
                sys.exit(1) # For abnormal termination, prefetch exception and force this exit.
 try:
   selection = int(user_input)
 except ValueError:
   print(fg.orange+'Invalid user input (please enter a number)'), sleep(1.1)
   continue
 if selection >= len(entries):
   print(fg.red+'Invalid user input (invalid number)'), sleep(2.1)
   continue
 # If the selected path is a folder, then change the working directory to
 # that folder.
 selected_path = Path(entries[selection])
 if selected_path.is_dir():
   os.chdir(selected_path)
   continue
 # Otherwise, try to read the file contents as if they are text. If this fails
 # print a warning message.
 try:
   file_contents = selected_path.read_text("UTF-8")
 except UnicodeDecodeError:
   print(Fore.LIGHTYELLOW_EX + f'{selected_path.name} is not a text or csv file'), sleep(2.1)
   continue
 # Otherwise print the file contents
 print(fg.black+bg.lightgrey + file_contents +Back.RESET)
 print(fg.lightgreen + "\r< ", end=""), sleep(1.1)
 print("\r> ", end=""+Fore.RESET), sleep(1.1)
 user_input = input('')
```

▼ filenum_call ( ) function code:

For dataset selection as a file to choose for analysis of its contents or recorded P's.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
694  def filenum_call():
695  #---------------------------------
696  # This function takes in user's
697  # file number request.
698  #---------------------------------
699       global filenum, idx, entries, fileresult
700
701       # Take file index value from user
702       print(fg.orange +bg.cyan +"||||||||||||||||||||||||||||||||||||||")
703       print(f'The current directory total listed files = {len(entries)-1}', sep="\n")
704       print("||||||||||||||||||||||||||||||||||||||"+Back.RESET)
705       filenum=input(fg.yellow+'Enter the relevant file # as {*.csv, *.txt, *.png} for analysis from the list: ')
706       filenum=int(filenum)  # Typecast (convert) string datatype into integer.
707       idx=filenum
708       print("\n")
709       print(Fore.GREEN + 'Selected file number is {} for {}'.format(idx, fileresult[idx-1]))
710
711       if dir_flag == 1: # if Safe Mode is enabled then maintain a restricted yet error tolerant
712                         # safe mode environment for the user to select a dataset file form directory.
713          print(Back.LIGHTGREEN_EX + Fore.YELLOW +
714              "\033[1m<--- SAFE MODE DATASET ANALYSIS ENABLED --->\033[0m" + Back.RESET), sleep(2)
715          print(f"{fg.lightgreen}Press any key to continue...")
716          sys.stdout.write('SM:>>...')
717          sys.stdout.flush()
718          os.system("pause >nul")
```

▼ csv_analyzer ( ) function code:

For dataset analysis, here as .csv to analyze its recorded P's.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
724  def csv_analyzer():
725  #-------------------------------------
726  # Open the QFLCA "*.csv" for analysis.
727  #-------------------------------------
728     global pngfile
729
730     with open(fileresult[idx-1], 'r') as x:
731           pngfile=Path(f'{fileresult[idx - 1]}').stem
732           sample_data = list(csv.reader(x, delimiter=","))
733           print(Fore.RED + Back.YELLOW +
734                 f'Sample_data initiated... selected file for analysis from index value = 0 to {len(entries)-1} is:
735  {idx}', sep="\n")
736           print(Fore.CYAN + Back.RESET + '\033[1;4m' + '- User menu options are limited to Basic QFLCC program
737  options.'+ '\033[0m')
738           print(Fore.LIGHTCYAN_EX + '\033[1;4m' + '- Table data list:' + '\033[0m'+ Fore.RED + Back.CYAN + Fore.RED)
739           subprocess.call([f'{pngfile}.png'], shell=True) # After stem from the selected csv filename, show the
740  relevant
741                                                          # histogram and circuit with the csv file being analyzed.
742           subprocess.call([f'{pngfile}_H.png'], shell=True)
743           sample_data = np.array(sample_data) # Import the P data into an array.
744           print(sample_data)
745
           for row in sample_data:
                 y=row[0]  # Store array's element value in y before converting csv left column data to a float value.
                 print(row[0] + Back.GREEN)
```

▼ file_ext_call ( ) function code:

Call this function to setup and reconfigure r/w switches on the selected dataset file (extension-based).

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def file_ext_call():
  #-------------------------------------------------------------------------
  # Call this function to setup and reconfigure r/w switches
  # on the selected dataset file (extension based).
  # * See group of file extensions to operate r/w on prior to this function.
  #-------------------------------------------------------------------------
  global filesFlag
  ###################################################################
  # Code reconfig options instead of the while loop:
  # r = root, d = directories, f = files
  # for r, d, f in os.walk(thisdir):
  #   for dirfile in f:
  #     if dirfile.endswith(ext0) and (file_extension=='.txt'
  #        or file_extension=='.docx'):
  # ... as follows.
  ###################################################################
  while filesFlag == 0:  # File state/flag is 1 in case of dataset files are present/copied in the current directory
    #for item in range(1, 2):
     if (file_extension=='.txt' or file_extension=='.docx' or file_extension=='.bin') and filesFlag==0:
          # print(os.path.join(r, dirfile))
          print(Fore.LIGHTYELLOW_EX + f'1- A {file_extension} file chosen to analyze:', thisfile)
          print('2- Temporarily read text file content available...')
          # print(os.path.join(r, dirfile))
          # csv_analyzer()
          file = open(thisfile, 'r', encoding='utf-8')
          content= file.read()
          print(fg.black+bg.lightgrey + content+ Back.RESET)
          print(Fore.LIGHTYELLOW_EX + 'Select another file for r-w analysis:...'), sleep(2.1)
          file_reader()
          print("Textual data to read...")
          filenum_call()
          filesFlag==0
          break
     elif (file_extension=='.png' or file_extension=='.jpg') and filesFlag==0:
          print(Fore.LIGHTYELLOW_EX + f'1- A {file_extension} file chosen to analyze:', thisfile)
          print('2- Temporarily image file content available...')
          # print(os.path.join(r, dirfile))
          pngfile=Path(f'{thisfile}').stem
          with open(thisfile, 'r') as x:
               subprocess.call(f'{pngfile}.png', shell=True)
               subprocess.call(f'{pngfile}.jpg', shell=True)
          file_reader()
          filenum_call()
          print("Image file to view...")
          filesFlag==0
          break
     elif file_extension=='.csv':
          print("*.csv being read for analysis...")
          csv_analyzer()
          filesFlag==1
          break
```

▼ **next ( ) function** code:

To flush out the buffered data on continuous 'n' character keystrokes on keyboard reaching the next program step.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def __next__():
    while True:
         if keyboard.is_pressed("n"):
              print('Next...')
              sys.stdout.flush()
              break
         '''This flushed out the buffered keystrokes of character 'n' from memory
            when key pressed by user continuously up to after reaching the 'Next...'
            printout message displayed on screen...'''
```

▼ **PAnalysis ( ) function** code:

To analyze the selected dataset by user, conduct its basic analysis of P's from a QDF circuit measurement.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def PAnalysis():
#########################################
# P analysis of the selected dataset. #
#########################################
      global df2, df3
      if  (sample_data.max(axis=0)[1] < 0.5 and sample_data.max(axis=0)[1] > 0.2):  # Conditions to
          # assign values to e.g., a weak classical min(ΔP) = max(P) of a quantum outcome range (from
          # the qubit dataset). Complement here is min(P) of the dataset if the user inputs for the opposition.
           df2=df.loc[((df['probability'] < 0.5) & (df['probability'] > 0.27)), :].binary_string[:]
           df3=df.loc[((df['probability'] < 0.5) & (df['probability'] > 0.27)), :].probability[:] # Classify
           # in the given included range of n:m
      elif (sample_data.max(axis=0)[1] < 1 and sample_data.max(axis=0)[1] >= 0.5):  # Conditions to assign values
            # to e.g., a strong classical max(ΔP) = min(P) of a quantum outcome range (from the qubit dataset).
            # Complement here is max(P) of the dataset if the user inputs for the opposition.
           df2=df.loc[((df['probability'] < 1) & (df['probability'] >= 0.5)), :].binary_string[:]
           df3=df.loc[((df['probability'] < 1) & (df['probability'] >= 0.5)), :].probability[:]
```

▼ qdf_PAnalysis ( ) function code:

To analyze the selected dataset by user, display its QDF analysis results of P's from a QDF circuit measurement.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def qdf_PAnalysis():
 global qdf, bitpair, df2bin, df3Mem, dfcomp, bitcomp # Main global variables to compute classical bit and qubit
P's.
 if (sample_data.max(axis=0)[1] < 1):
    PAnalysis()
    print(Fore.GREEN + Back.RESET + 'Strong Prediction binary string is: ' + Fore.YELLOW,
          df2.to_string(index=False, header=False) + Fore.GREEN + ', with a P value of'
          + Fore.YELLOW, df3.to_string(index=False, header= False))
    df3Mem =df3.to_string(index=False, header= False)
    ###
    df2bin= df2.to_string(index=False, header=False)

    qdf0=[df2bin[i:i+2] for i in range(math.floor(len(df2bin)/2)-1, math.ceil(len(df2bin)), 2)] # Use this formula
    # to satisfy conditions of relevant outer binary string relative to central bit-pair according to Eq. (20) or
    # Ref. [1] of the current Elsevier J. paper (given your/the QDF circuit design)...
    qdf1=[df2bin[i:i+2] for i in range(math.floor(len(df2bin)/2), math.ceil(len(df2bin)), 2)]
    qdf2=[df2bin[i:i+2] for i in range(math.floor(len(df2bin)/2)-1, math.ceil(len(df2bin)),
math.floor(len(df2bin)/2))]

    qdf= qdf0 or qdf1 or qdf2 # Dynamic assignment of either df2bin range.

    if len(df2bin) <= 6 and qi_f == 1: # This if statement is designed to analyze QI dataset. One or more if
statements
         # needed to be added for IBMQ or different circuit/dataset configurations with their recorded binary
strength
         # lengths in their tables (their *.csv structure).
         print(Fore.GREEN + 'Paired qubit (left list element) relative to classical bit output (right list element):
'
               + Fore.YELLOW, qdf0)
         qdf= qdf0  # Assign resultant value of qdf0 to qdf.
    elif len(df2bin) > 6 and qi_f == 1:
         print(Fore.GREEN + 'Paired qubit (left list element) relative to classical bit output (right list element):
'
               + Fore.YELLOW, qdf1)
         qdf= qdf1  # Assign resultant value of qdf1 to qdf.
    else:  # Depending on QDF circuit design and configuration, more elif statements can be added to suit a bit-pair
           # count and string foci as the one designed under IBMQ platform compared to QI (or QInspire) platform.
         print(Fore.GREEN + 'Paired qubit (left list element) relative to classical bit output (right list element):
'
               + Fore.YELLOW, qdf2) # Focus on the maximum from the middle qubit pair, if we put i+3, is to denote
               # a pair + 1 qubit denoting up to three particles involved e.g. a photon Alice, Bob and the prize
entangled.
               # Iterate over a range of bins that are multiples of 2 (i.e. the size of the split substrings) for
row in sample_data:
         qdf= qdf2  # Assign resultant value of qdf2 to qdf.

    if dir_flag == 0:
         prompt()
    else:  # Maintain a restricted yet error tolerant safe mode environment for the user to select a dataset file
form directory.
         print(Back.LIGHTGREEN_EX + Fore.YELLOW +
             "\033[1m<--- SAFE MODE DATASET ANALYSIS CONTINUES --->\033[0m" + Back.RESET), sleep(2)
         print(f"{fg.lightgreen}Press any key to continue...")
         sys.stdout.write('SM:>>...')
         sys.stdout.flush()
         os.system("pause >nul")
    ##############################################################################################
    # This part of the function prints the expected measurement output for different cases
    # on qubit pairs.
    ##############################################################################################
    print(Fore.RED + f'\x1B[1;4mNotes:\033[0m\n' + Fore.CYAN + f'-* When shown in histograms generated from quantum
circuit, \
the rightmost bit is for the measured qubit with the lowest index (q[0]). The leftmost bit is for the qubit with the
highest index \
as the most significant qubit.\n-* Measurement focuses on QDF circuit bit pairs given the circuit configuration \
== focus on the maximum from the middle qubit pair in a list of measurement results. The program loop iterates over
a range of \
bins that are multiples of 2 (i.e. the size of the split substrings) for row in sample_data...'+ Fore.YELLOW)
    df2list = qdf0 or qdf1 or qdf2
    qubit_print=''
    if '01' in df2list[:-1]: # Print the verdict on the one element prior last.
         print(fg.yellow+'{P_|01>b(01)} = 1 == ES')
         qubit_print='{P_|01>b(01)} = 1 == ES'
    if '00' in df2list[:-1]:
         print(fg.yellow+'{P_|00>b(00)} = 0 == GS')
         qubit_print='{P_|00>b(00)} = 0 == GS'
    if '10' in df2list[:-1]:
         print(fg.yellow
               +f"{'{P_|10>b(10)} = 2 == ES within GS (QPT) or prize with lesser E value == some E loss or gain for
Bob == uncertain or certain by swap gate when Eve reveals prize state':<40}")
```

```
          qubit_print=f"{'{P_|10>b(10)} = 2 == ES within GS (QPT) or prize with lesser E value == some E loss or
gain for Bob == uncertain or certain by swap gate when Eve reveals prize state':<40}"
    if '11' in df2list[:-1]:
          print(fg.yellow+'{P_|11>b(11)} = 3 == superposition or prize state entangled with Alice or Bob')
          qubit_print='{P_|11>b(11)} = 3 == superposition or prize state entangled with Alice or Bob'
    if '0b' in df2list[:]: #or '1b' in df2list[:-1]:
          print(bg.orange + fg.lightgreen, qubit_print, Back.RESET + fg.lightgreen + "The binary string "+
Fore.YELLOW
                + "\'..0b\' (classical bit)" + fg.lightgreen + " ends upon the least significant qubit (rightmost
qubit).")
    if '1b' in df2list[:]:
          print(bg.orange + fg.lightgreen, qubit_print, Back.RESET + fg.lightgreen + "The binary string "+
Fore.YELLOW
                + "\'..1b\' (classical bit)" + fg.lightgreen + " ends upon the least significant qubit (rightmost
qubit).")

    dfcomp = 1-df3  # Calculation of the P complement for the bitpair.
    np.seterr(all="ignore") # Suppress irrelevant/outdated floating point numpy errors.

    for bit in df2:
         if bit == "1":
             bit == "0"
         else:
             bit == "1"
    bitcomp = bit.replace("1", "2").replace("0", "1").replace("2", "0")

    if len(df2bin)>0 and qi_f==1:
         # The following lines cover the bitpair property between the qdf0 to qdf1 or qdfn as assigned above (as
replaced bits).
         index = 0
         if qdf[index] == "01":
                   bitpair = ["10"]
         if qdf[index] == "10": # We use the if statement rather than elif due to absolute conditions of the bitpair
readout
                                # from the dataset, according to above if-statements and for-loop.
                   bitpair = ["01"]
         if qdf[index] == "11":
                   bitpair = ["11"]
         if qdf[index] == "00":
                   bitpair = ["00"]
         if qdf[index] == "0b": # If classical state is = 0 in binary, result is conditioned to classical bit = 0.
                   bitpair == "0"
         if qdf[index] =="1b": # If classical state is = 1 in binary, result is conditioned to classical bit = 1.
                   bitpair == "1"

         print(fg.green+"Complement of the binary string is:"+fg.yellow, bitcomp, fg.green+", and the focused pair
in it "
               +fg.yellow, bitpair, fg.green+"has a P' value of"+ Fore.LIGHTRED_EX, dfcomp.to_string(index=False,
header=False))
 else:
    print(fg.red+'Erroneous or Tied P values!')
    subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program.

 """ Complex code alternative for the line:
 if len(df2bin)>0 and qi_f==1:
          bitpair = [bitcomp[i:i+2] for i in range(math.floor(len(bit)/2)-2, math.floor(len(bit)/2), 2)] # This line
covers the
                    # bitpair property between the qdf0 to qdf1 or qdfn as assigned above.
          index = 0
          #bitpair = [qdf[index]] # Invert the bitpair result from the past do to compare with the last bitpair
result when needed.
          #while index <= len(bitpair):
 elif len(df2bin)<=6 and qi_f==1:
          bitpair = [bitcomp[i:i+2] for i in range(math.floor(len(bit)/2)-1, math.floor(len(bit)/2), 2)] # This line
covers the
                    # bitpair property between the qdf0 to qdf1 or qdfn as assigned above.
          print(fg.green+"Complement of the binary string is: "+fg.yellow, bitcomp, fg.green+", and the focused pair
in it "
                +fg.yellow, bitpair, fg.green+"has a P' value of"+ Fore.LIGHTRED_EX, dfcomp.to_string(index=False,
header=False)) """

 if dir_flag == 1: # Maintain a restricted yet error tolerant safe mode environment for the user to select a dataset
file form directory.
     print(Back.LIGHTGREEN_EX + Fore.YELLOW +
           "\033[1m<--- SAFE MODE DATASET ANALYSIS CONCLUDED --->\033[0m" + Back.RESET+ fg.lightgreen), sleep(2)
```

▼ safeMode_dir( ) function code:

To analyze the selected dataset by user within the Safe Mode (SF:>>) simulation environment. The limited commands compare the selected dataset results with the realtime simulated IBM QDF circuit results (dataset).

- ▼ Source code in `QAI-LCode_QFLCC.py`

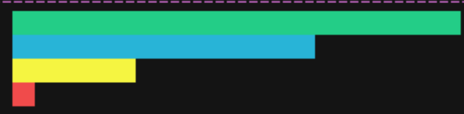
```
def safeMode_dir():
  ###############################################################################
  # Directory function restricted to select a different file by the user
  # for analysis in a restricted yet safe mode environment.
  ###############################################################################
  global dir_flag # Flag is set to 1 for dataset directory access in safe mode.
  global dfcompMem # Recall and reuse as global from PAnalysis() results.
  if dir_flag == 1:
       print(Back.LIGHTGREEN_EX + Fore.YELLOW +
             "\033[1m<--- SAFE MODE DIR ENVIRONMENT ENABLED --->\033[0m" + Back.RESET), sleep(2)
       filesFlag = 0
       file_reader()  # Execution flow of functions with dir and file flag settings
                      # must run sequentially...
       filenum_call()
       #filesFlag = 0
       filesFlag = 0
       with alive_bar(4, bar = 'blocks', manual=True) as bar:  # Print progress bar per Safe Mode step.
                                                               # Enables multiprocessing when needed.
        csv_analyzer()
        bar(.2)
        file_ext_call()
        bar(.3)
        dataframe()
        bar(.7)
        qdf_PAnalysis()
        bar(.9)
        dfcompMem = float(dfcomp.to_string(index=False, header=False)) # Convert the string representation
             # of string to float for future use/comparison/validation.
        #filesFlag==1 # This file state denotes all dataset files are present within this directory.
             # No need to overwrite in case of program restart or in safe mode (SM)...
        bar(1)
        ###  Progress concluded on the selected file by user and its preliminary anlysis. ####
       #dir_flag == 0 # Reset flag until next time requested by the user to access dataset files.
       #prompt()
       #PAnalysis_model() # Continue IBM QDF circuit simulation comparison of its dataset with the selected one.
  else:
       print(Back.LIGHTGREEN_EX + Fore.YELLOW +
             "\033[1m<--- SAFE MODE DIR ENVIRONMENT DISABLED --->\033[0m" + Back.RESET), sleep(2)
       """- End of safeMode_dir function calls """
```

▼ PAnalysis_model( ) function code:

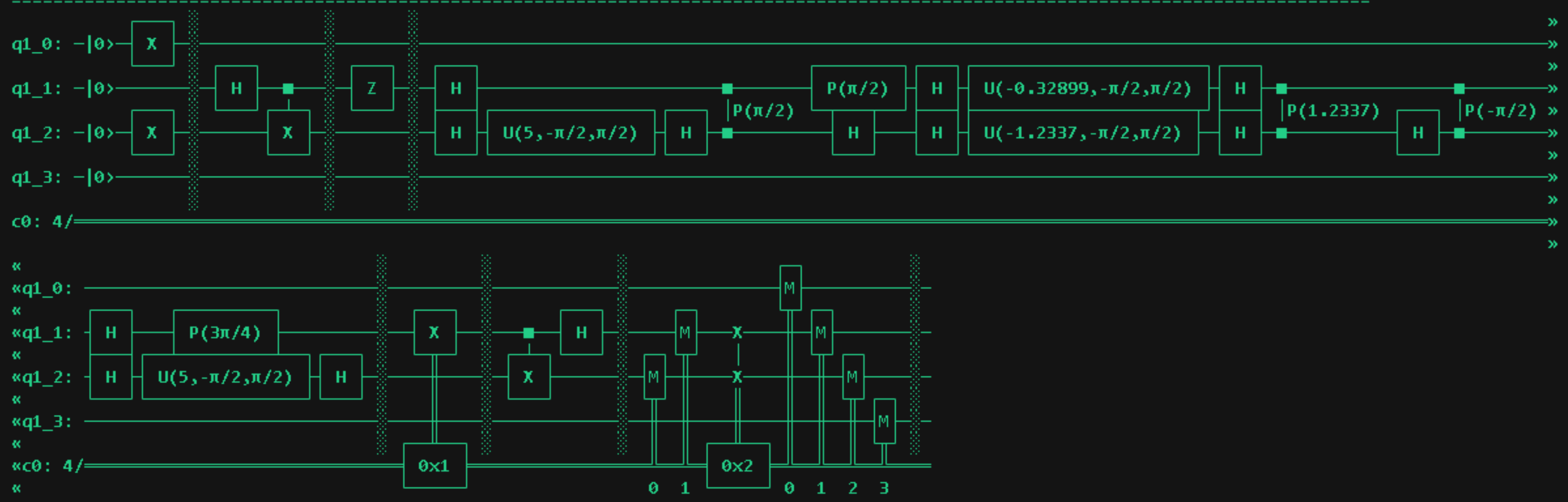
To run IBM QDF circuit simulation and analyze the selected dataset by user, comparing dataset results with the simulated IBM QDF circuit results (dataset). Dependencies: imports the QDF-LCode_IBMQ-2024.py module (or its codable version).

- Screenshots of the circuit simulation and user's selected dataset analysis after executing the `PAnalysis_model()` function:

```
=====================================================================================================
 An ideal QDF circuit I/O model running realtime producing an IBMQ-based QDF dataset is from the
 QDF-LCode_IBMQ-2024.py file. This circuit is compared to the analyzed dataset to show how close
 the match is for a desired Hamiltonian and expected measurement outcome as a point of reference.
=====================================================================================================
on 0: Please Wait!...:
on 0: *- Program is processing the analyzed dataset circuit P's after:
on 2: *- Import + Simulate the IBM QDF circuit by { Qiskit Aer Simulator, IBMQ Provider } [QUANTUM MODE ENVIRONMENT] on this computer.
on 3: *- Dataset P results are compared on the imported circuit by the { QDF-LCode_IBMQ-2024.py } module on this computer [SAFE MODE ENVIRONMENT].
|████████████████████████████████████████| 100% [3/3] in 3.1s (1.13/s)
=====================================================================================================
Quantum simulation of a quantum field lens coding algorithm with entanglement scaling between a
multi-well (barrier) interaction potential of internal system B interacting and external system A from the method article:
https://www.sciencedirect.com/science/article/pii/S221501612300136X
*- QDF Circuit Built and Simulator generates its realtime datasets from { Qiskit Aer, IBMQ } Qasm_Simulator
   Workspace and/or Virtual Environment on user's computer in Python.
*- Created by: Philip B. Alipour, Supervisor: T. A. Gulliver, at the University of Victoria,
   Dept. ECE, Victoria BC, Canada.
*- Code updated based on Qiskit 2023--2024 changes specified in code comments of this simulator.
*- Sidenotes: You can also run code with the right packages installed in pipx or python.
   Examples of package installation changes and features can be found on:
   https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-installation and
   https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-features
=====================================================================================================
=====================================================================================================
<--- {{ Qiskit Aer, IBM }} QDF CIRCUIT SIMULATION BEGINS --->
=====================================================================================================
Dynamic Providers: QasmSimulator('qasm_simulator')
Counts for Qubit Pairs out of 8192 shots: {'0101': 5541, '0111': 2243, '0001': 408}
 Σ (5541, 2243, 408) = 8192 shots ⟶  ⟨P(b|ij⟩)⟩ = { ({n1, n2, n3}/8192)b|ij⟩ }
 = { p1(0101) = 0.6763916015625 } + { p2(0111) = 0.2738037109375 } + { p3(0001) = 0.0498046875 } = 1
=====================================================================================================
 Plot = [Experimented n of N counts  ∝  p of total circuit events P on pairwise qubits = P(b|ij⟩)]; |ij⟩ ≡ |q_i q_j⟩ ≡ |qᵢqⱼ⟩.
=====================================================================================================
P(b|ij⟩)                   [1.0000000000000]
p1(b|ij⟩) = p1(0101)       [0.6763916015625]
p2(b|ij⟩) = p2(0111)       [0.2738037109375]
p3(b|ij⟩) = p3(0001)       [0.0498046875000]
```



```
<<-- IBM QDF Circuit Measurement Results by Qiskit Aer Simulator Plotted Successfully! End of Task... -->>
=====================================================================================================
```




```
<<-- QDF Circuit BUILT & RAN Successfully! End of Task... -->>
=====================================================================================================
<--- { Qiskit Aer, IBM } QDF CIRCUIT SIMULATION CONCLUDED --->
=====================================================================================================
Ln 1092, Col 2   Spaces: 5   UTF-8   CRLF   Python   3.11.8 64-
```

```
==========================================================================================================================
*- Enter 'n' or 'next' to proceed calculating P's between the selected dataset and the simulated IBM QDF circuit results.
①  The datasets of the two QDF circuits are compared and validated with a verdict on how close their projection of
 events (prediction) is, given their QDF circuit configuration!
*- ✅ Enter 'dir' or 'sm dir' to select another dataset to analyze in Safe Mode (SM:>>). SIMULATION WILL NOT RESTART!
*- ✅ Enter 'n' or 'next' to continue in Safe Mode (SM:>>). QDF CIRCUITS SIMULATION_DATASET_ANALYSIS WILL START!
*- Enter any other key to change from SM:>> to regular prompt mode (>).
*- Enter 'r' in regular prompt mode (>) to 'restart' circuit simulation and dataset analysis. Enter 'h' for more options.
SM:>> n
SM:>> Next...
<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS BEGINS --->
==========================================================================================================================
Plot = [P samples of the IBMQ model circuit, if meets  ∝  p of the selected dataset file { QI_Exp_02.csv }
on pairwise  qubits]...
Then a strong vs weak p match, and the distance between QDF circuit events are determined.*
==========================================================================================================================
* Computation model: κ Scalar ΨΦ Field Switch and Correlation Model, Ref. [1] of the DIB article.
  P(Ψ ↔ κ²Ψ) = (P(Ψ ↔Φ)) = P(b|ij)); |ij⟩ ≡ |q_i q_j⟩ ≡ |q_i q_j⟩.
==========================================================================================================================

<-- P Sampling between IBM QDF Circuit and the Selected QDF Circuit Dataset -->

<-- P Results Sampled from { ibm-qdf-stats.txt } file are: -->
on 0: P(Total)           [1.0000000000000]
      ibmq qdf p(0101)   [0.6763916015625]
      ibmq qdf p(0111)   [0.2738037109375]
      ibmq qdf p(0001)   [0.0498046875000]
on 0:
|████████████████████████████████████████| 100% [3/3] in 2.2s (1.46/s)
==========================================================================================================================
*- min(P) from { ibm-qdf-stats.txt } is performed by the QDF bit pairs { 0001 } as { ibm qdf } sample set #3 = 0.0498046875
*- max(P) from { ibm-qdf-stats.txt } is performed by QDF bit pairs { 0101 } as { ibm qdf } sample set #1 = 0.6763916015625
==========================================================================================================================
P(Total)                          [1.0000000000000]
{ QI_Exp_02.csv } max(P(010100b))  [0.9716800000000]
{ QI_Exp_02.csv } comp(P(101011b)) [0.0283200000000]
{ QI_Exp_02.csv } min(P(010010b))  [0.0283203125000]
{ ibm qdf } min((P(0001))          [0.0498046875000]
{ ibm qdf } max((P(0101))          [0.6763916015625]
Δp of min(P)                      [0.0214843750000]
Δp of max(P)                      [0.2952883984375]
Δp of min(P) match                [0.9785156250000]
Δp of max(P) match                [0.7047116015625]
==========================================================================================================================
*- Δp Data: ⚖ Validation Verdict ⚖ between the two dataset samples from { QI_Exp_02.csv , ibm-qdf-stats.txt }
   is a/an: Above Avg. Match {{ 0.98 , 0.7 }}
==========================================================================================================================
*- QDF Qubit Sets: ⚖ Validation Verdict ⚖ between the two dataset samples from { QI_Exp_02.csv , ibm-qdf-stats.txt }
   is a/an: Strong Match { 010100b , 0101 }
==========================================================================================================================
*- QDF Circuits: ⚖ Validation Verdict ⚖ between the two QDF circuit configurations from { QI_Exp_02.csv , ibm-qdf-stats.txt }
   is a/an: Above Avg. Match
==========================================================================================================================
 *- The IBM QDF circuit can be reconfigured for worst and best case Hamiltonian scenarios,
    given the expected QDF measurement outcomes from the collected QFLCA datasets for a QFLCC.
 Simulation completed on: 2024-03-30 17:41:18.391752
==========================================================================================================================
<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED --->
> exit
Exiting program...🖐(° ʖ °)🖐 Goodbye!
The program is terminated manually!
```


- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def PAnalysis_model():
 ##########################################################################################################################
 #--- This function compares the mapped results between QDF circuits (IBMQ <--> QI).
 #--- SAFE MODE QFLCC AND,
 #--- QDF Circuit Model from IBMQ article file: QDF-LCode_IBMQ-2024-codable.py,
 # for python library purposes imported as a module:
 #--- QDF-LCode_IBMQ-2024.py (same code content as a duplicate of QDF-LCode_IBMQ-2024-codable, but will be overwritten
 # by IBMQ folder content if you modify or update code. Only edit code within the QDF-LCode_IBMQ-2024-codable.py file!).
##########################################################################################################################
 global sim_state, dir_flag, entry_stage, hline
 sim_state = ['']
 hline =
"=========================================================================================================================="
 print(f'''{Fore.LIGHTMAGENTA_EX + hline}\n An ideal QDF circuit I/O model running realtime producing an IBMQ-based QDF dataset is
from the \
 \n QDF-LCode_IBMQ-2024.py file. This circuit is compared to the analyzed dataset to show how close \
 \n the match is for a desired Hamiltonian and expected measurement outcome as a point of reference.\
 \n{hline}''')
 #-------------------------------------------------------------------------------------------
 # <----- Display histogram results of the QDF circuit event P's and then printing the circuit
 # as a model to display for QDF datasets of other circuits being analyzed here (next version will).
 # Future release will have current P results of QInspire or any other QDF circuit configure the
 # IBM-QDF circuit through a QAI mapping, accordingly (subject of QAI lab for QFLCA) ----->
 #-------------------------------------------------------------------------------------------
 P = 1 # The total probability of QDF circuit events relative to classical states or
       # denoting the system's Hamiltonian (total energy).
 with alive_bar(3, bar = 'blocks', manual=True) as bar:   #To print progress bar on these dataset and circuit simulation analyses.
    print(Fore.LIGHTMAGENTA_EX + Back.LIGHTGREEN_EX+ f"Please Wait!...:")
    print(f"*- Program is processing the analyzed dataset circuit P's after:"+Back.RESET), sleep(1)
    bar(0.5)
    print(f"{Back.LIGHTGREEN_EX}*- Import + Simulate the IBM QDF circuit by\
 {{ Qiskit Aer Simulator, IBMQ Provider }} [QUANTUM MODE ENVIRONMENT] on this computer."+Back.RESET), sleep(1)
    bar(0.7)
    print(f"{Back.LIGHTGREEN_EX}*- Dataset P results are compared on the imported circuit by the {{ QDF-LCode_IBMQ-2024.py }}
module\
 on this computer [SAFE MODE ENVIRONMENT]."+ Back.RESET + Fore.LIGHTMAGENTA_EX), sleep(1)
    bar(1.)

 # Log the start of simulation.
 entry_stage = 0
 sim_state[0] = " QDF CIRCUIT SIMULATION BEGINS "
 sim_log()
 #-------------------------------------------------------------------------------------------------------------------
 ibm_qdf_module = importlib.import_module("QDF-LCode_IBMQ-2024-codable") # Unconventional call from the targeted module.
 bar(1.)
 #-------------------------------------------------------------------------------------------------------------------
 # Log the prompt of simulation to the user to proceed, restart or exit program.
 entry_stage = 1
 sim_state[0] = " QDF CIRCUIT SIMULATION_DATASET_ANALYSIS PROMPT "
 sim_log()
 #-----------------------------------------------------------------------------
 # Selected dataset and the simulated circuit's dataset validation verdict starts here,
 # if user enters 'n' or 'next'...
 # -----------------------------------------------------------------------------
 print(Fore.LIGHTGREEN_EX + f"*- Enter 'n' or 'next' to proceed calculating P's between the selected dataset and the simulated\
IBM QDF circuit results.")
 print(f"🛈 {Fore.LIGHTMAGENTA_EX}\x1B[1;4mThe datasets of the two QDF circuits are compared and validated with a verdict on how\
close their projection of \n events (prediction) is, given their QDF circuit configuration!\033[0m")
 print(f"{Fore.LIGHTGREEN_EX}*-     Enter 'dir' or 'sm dir' to select another dataset to analyze in Safe Mode (SM:>>).\
SIMULATION WILL NOT RESTART!")
 print(f"{Fore.LIGHTGREEN_EX}*-     Enter 'n' or 'next' to continue in Safe Mode (SM:>>).\
QDF CIRCUITS SIMULATION_DATASET_ANALYSIS WILL START!")
 print(f"{Fore.LIGHTGREEN_EX}*- Enter any other key to change from SM:>> to regular prompt mode (>).")
 print(f"*- Enter 'r' in regular prompt mode (>) to 'restart' circuit simulation and dataset analysis. Enter 'h' for more options.

 sm_input = input(f"SM:>> {fg.yellow}")
 if sm_input == "dir" or sm_input == "sm dir":
      dir_flag = 1 # Directory flag is set to 1 for directory access in safe mode.
      safeMode_dir()
      pass
 elif sm_input == "n" or sm_input == "next":
      dir_flag = 1
      print(f"{Fore.LIGHTGREEN_EX}SM:>>{fg.yellow} Next..."), sleep(1)
 else:
      dir_flag = 0  # Directory flag is set to 0 in case of regular prompt mode (>) or 'r' to restart the simulation.
      prompt()

 print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS BEGINS --->\033[0m" + Back.RESET)

 ibmq_result = 'ibm-qdf-stats.txt'
```

```
with open(ibmq_result, 'r') as file:
       # Read the contents of the file
       content = file.read()

       # Extract all the p values from the content as float.
       N = re.findall(r'[-+]?(?:\d*\.*\d+)', content)
       n_list = list(map(float, N))

       # Convert all the metadata digits to str to sort out binaries.
       bin_list = list(map(str, N))  # This is to identify qubit binary strings.

       # Calculate quantum event p's for the total P.
       P = n_list[0]  # The P result is stored to the p_list.
       p1 = n_list[1] # 1st p result is stored to the p_list.
       pq1 = bin_list[4]
       p2 = n_list[2] # 2nd p result is stored to the p_list.
       pq2 = bin_list[5]
       p3 = n_list[3] # 3rd p result is stored to the p_list.
       pq3 = bin_list[6]

ibmq_p_max = max(p1, p2, p3)  # Identify the max value from the p list.
max_p_index = n_list.index(ibmq_p_max) # Store the index value for p_max from the ibmq_result file.

ibmq_p_min = min(p1, p2, p3)  # Identify the min value from the p list.
min_p_index = n_list.index(ibmq_p_min) # Store the index value for p_min from the ibmq_result file.

# Set the default p color codes to 'white' as defined below in the p_color list until an if condition applies.
p_color = [Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE,
           Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE]

if round(p1, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p1).
     p_color[0] = Style.BRIGHT + Fore.RED
if round(p2, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p2).
     p_color[1] = Style.BRIGHT + Fore.RED
if round(p3, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p3).
     p_color[2] = Style.BRIGHT + Fore.RED
if round(p1, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p1).
     p_color[0] = Style.BRIGHT + Fore.CYAN
if round(p2, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p2).
     p_color[1] = Style.BRIGHT + Fore.CYAN
if round(p3, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p3).
     p_color[2] = Style.BRIGHT + Fore.CYAN

print(f"{Fore.LIGHTMAGENTA_EX}{hline}\nPlot = [{Fore.LIGHTGREEN_EX}P samples of the IBMQ model circuit,\
if meets {Fore.LIGHTYELLOW_EX} ∝ {Fore.LIGHTGREEN_EX} p of the selected dataset file\
{{ {Fore.LIGHTYELLOW_EX + fileresult[filenum-1] + Fore.LIGHTGREEN_EX} }}\non pairwise \
qubits{Fore.LIGHTMAGENTA_EX}]...\nThen a strong vs weak p match, and the distance between QDF circuit events\
are determined.{Fore.YELLOW}* {Fore.LIGHTGREEN_EX}\n{hline}{Fore.LIGHTGREEN_EX}\
\n{Fore.YELLOW}* Computation model:{Fore.GREEN} κ Scalar ΨΦ Field Switch and Correlation Model, Ref. [1] of the DIB article.\
\n  {Fore.LIGHTCYAN_EX}P(Ψ ↔  κ²Ψ) = ⟨P(Ψ ↔ Φ)⟩ = P(b|ij)); |ij⟩ ≡ |q_i q_j⟩ ≡ |qᵢqⱼ⟩. {Fore.GREEN}\
\n{hline}"), sleep(0.5)

print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Sampling between IBM QDF Circuit and the Selected QDF Circuit Dataset -->\033[0m''')
if dir_flag == 1:  # Read and display the printed circuit when dataset dir is in SAFE MODE.
      with open("ibm-qdf-circuit_output.bin", 'r', encoding='utf-8') as file_to_read:
           print(f'{fg.black}{bg.lightgrey}\n'+file_to_read.read(),"\n<-- IBM QDF Circuit Sample Printed in SAFE MODE --> "
                + Back.BLUE + str(datetime.datetime.now()) + Back.RESET)
           file_to_read.close() # End reading and displaying the printed circuit.
print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Results Sampled from {{ {ibmq_result} }} file are: -->\033[0m''')
with alive_bar(3, bar = 'blocks' , manual=True) as bar:
 model_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities.
 model_fig.barh([P, p1, p2, p3], [Style.BRIGHT + Fore.LIGHTGREEN_EX+"P(Total)", p_color[0]+f"ibmq qdf p({bin_list[4]})",
                           p_color[1]+f"ibmq qdf p({bin_list[5]})", p_color[2]+f"ibmq qdf p({bin_list[6]})"],
        force_ascii=False), sleep(1)
 model_fig.show(), sleep(1)
 print('')
 bar(1)
 dir_flag = 0 # Reset flag until recalled.
 qdf_bit_pairs = ['',''] # To register which qdf_bit_pair has the max or min p.
 #-------------------------------------------------------------------------------------
 # Now classify and print which sampled qdf_bit_pair set from the
 # ibmq_result file is max_p and which min_p.
 if min_p_index == 1:
  qdf_bit_pairs[0] = pq1
 if min_p_index == 2:
  qdf_bit_pairs[0] = pq2
 if min_p_index == 3:
  qdf_bit_pairs[0] = pq3
 if max_p_index == 1:
  qdf_bit_pairs[1] = pq1
 if max_p_index == 2:
```

```
    qdf_bit_pairs[1] = pq2
   if max_p_index == 3:
    qdf_bit_pairs[1] = pq3

print(Fore.YELLOW+f'{hline}')
print(fg.purple+f'*- min(P) from {{ {ibmq_result} }} is performed by the QDF bit pairs\
{{ {fg.red+ qdf_bit_pairs[0] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{min_p_index} = {ibmq_p_min}')
print(fg.purple+f'*- max(P) from {{ {ibmq_result} }} is performed by QDF bit pairs\
{{ {fg.lightgreen+ qdf_bit_pairs[1] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{max_p_index} = {ibmq_p_max}')
print(Fore.YELLOW+f'{hline}')
#-------------------------------------------------------------------------------------------------------------
# Recall and store which binary string had the minimum P from the table of the selected dataset (dataframe).
global df_min, df_min_bin
if  (sample_data.min(axis=0)[1] < 1):
          df_min=df.loc[(df['probability'] <= sample_data.min(axis=0)[1]), :].binary_string[:]
          df_min_bin = df_min.to_string(index=False, header=False)
#-------------------------------------------------------------------------------------------------------------
deltaMatch_max = (1 - abs(ibmq_p_max - float(df3Mem)) ) # Calculate Δp of max(P) Match.
deltaMatch_min = (1 - abs(ibmq_p_min - sample_data.min(axis=0)[1]) ) # Calculate Δp of min(P) Match.
delta_p_min = abs(ibmq_p_min - sample_data.min(axis=0)[1]) # Calculate Δp of min(P).
delta_p_max = abs(ibmq_p_max - float(df3Mem)) # Calculate Δp of max(P).

Valid_Verdict = ['Weak',f'{{∞ , "/ₐ}}', 'Avg.', 'Strong', 'NUL']  # Validation Verdict of a strong
                                                                  # vs. weak correlation match between
                                                                  # the selected file dataset p's and the IBM QDF
                                                                  # circuit event p's...
# Note: A verdict of NUL denotes missing information or unknown about the max or min of the p's
# (calculable if config is corrected or complemented).

if round(deltaMatch_max, 2) >= 0 and round(deltaMatch_max, 2) < 0.5: # Classify/color code max Δp match.
     p_color[3] = Style.BRIGHT + Fore.RED
     Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

if round(deltaMatch_max, 2) >= 0.5 and round(deltaMatch_max, 2) <= 0.55: # Classify/color code max Δp match.
     p_color[3] = Style.NORMAL + fg.silver
     Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

if round(deltaMatch_min, 2) >= 0 and round(deltaMatch_min, 2) < 1/3: # Classify/color code min Δp match.
     p_color[4] = Style.BRIGHT + Fore.RED
     Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

if (round(deltaMatch_min, 2) <= 1/3 and
    round(deltaMatch_max, 2) <= 1/3) or (round(deltaMatch_min, 2) <= 1/2 and
                                         round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
     p_color[3] = Style.NORMAL + fg.silver
     p_color[4] = p_color[3]
     Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

if round(deltaMatch_max, 2) > 0.55 and round(deltaMatch_max, 2) < 0.66: # Classify/color code max Δp match.
     p_color[3] = Style.BRIGHT + Fore.LIGHTYELLOW_EX
     Valid_Verdict = Fore.LIGHTYELLOW_EX + Valid_Verdict[2]

if (round(deltaMatch_max, 2) >= 0.66 and
    round(deltaMatch_max, 2) < 0.9) and (round(deltaMatch_min, 2) >= 0.66 and
                                         round(deltaMatch_max, 2) < 0.9): # Classify/color code min-max Δp match.
     p_color[3] = Style.BRIGHT + Fore.LIGHTMAGENTA_EX
     p_color[4] = p_color[3]
     Valid_Verdict = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]

if (round(deltaMatch_max, 2) >= 0.9 and
    round(deltaMatch_max, 2) <= 1) and (round(deltaMatch_min, 2) >= 0.9 and
                                        round(deltaMatch_max, 2) <= 1): # Classify/color code min-max Δp match.
     p_color[3] = Style.BRIGHT + Fore.LIGHTGREEN_EX
     p_color[4] = p_color[3]
     Valid_Verdict = Fore.LIGHTGREEN_EX + Valid_Verdict[3]

if (round(deltaMatch_min, 2) <= 1/3 and
    round(deltaMatch_max, 2) <= 1/2) or (round(deltaMatch_min, 2) <= 1/2 and
                                         round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
     p_color[3] = Style.NORMAL + fg.silver
     p_color[4] = p_color[3]
     Valid_Verdict = fg.silver + Valid_Verdict[4] # See note for a NUL verdict!

PAnalysis_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities of the two datasets.
PAnalysis_fig.barh([P, float(df3Mem), float(dfcompMem), sample_data.min(axis=0)[1], ibmq_p_min, ibmq_p_max,
                    delta_p_min, delta_p_max, deltaMatch_min, deltaMatch_max],
    [Style.BRIGHT + Fore.LIGHTGREEN_EX + f"P(\x1B[4mTotal\x1B[0m{Style.BRIGHT + Fore.LIGHTGREEN_EX})",
     f'{Style.BRIGHT+Fore.WHITE}{{ {fileresult[filenum-1]} }} max(P(\x1B[4m{df2bin}\x1B[0m{Style.BRIGHT+Fore.WHITE}))',
     f'{Style.RESET_ALL + Fore.WHITE}{{ {fileresult[filenum-1]} }} comp(P(\x1B[4m{bitcomp}\x1B[0m{Style.DIM+Fore.WHITE}))',
     Style.BRIGHT + Fore.WHITE + f"{{ {fileresult[filenum-1]} }} min(P(\x1B[4m{df_min_bin}\x1B[0m{Style.BRIGHT + Fore.WHITE}))",
     f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} min((P(\x1B[4m{qdf_bit_pairs[0]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
```

```
      f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} max((P(\x1B[4m{qdf_bit_pairs[1]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
      p_color[4] + "Δp of \x1B[4mmin(P)\x1B[0m" + p_color[4],
      p_color[3] + f"Δp of \x1B[4mmax(P)\x1B[0m" + p_color[3],
      p_color[4]+ "Δp of\x1B[4m min(P) match\x1B[0m" + p_color[4],
      p_color[3]+ "Δp of\x1B[4m max(P) match\x1B[0m" + p_color[3]], force_ascii=False), sleep(1)
 PAnalysis_fig.show()
 bar(1.)

 print(f"{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- Δp Data:      \033[4mValidation Verdict\033[0m    {fg.yellow}  \
 between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
 \n   is a/an: \
\033[4m{Valid_Verdict + ' Match {{ '+ str(round(deltaMatch_min, 2)) +' , '+ str(round(deltaMatch_max, 2)) +' }}'}\033[0m\
{Fore.LIGHTYELLOW_EX}")

 DeltaP_verdict = Valid_Verdict
 Valid_Verdict = ['Weak',f'{{∞ , ⁿ/ₐ}}', 'Avg.', 'Strong', 'NUL']  # Reset Valid_Verdict list elements upon
                                                                  # the previously rendered verdict.
 #-------------------------------------------------------------------------------------------------
 # Validate circuit config. based on recorded P's associated to their qubit pair binaries (strings)
 s1 = df_min_bin
 s2 = qdf_bit_pairs[0]
 s3 = df2bin
 s4 = qdf_bit_pairs[1]
 vv_circuits =[False]
 vv_bins = [False]

 # Validation Verdict on the two circuits qubit string sets over min and max P's as compared for a match.
 if re.match(s2, s1):
     vv_bins = fg.lightgreen+ Valid_Verdict[3]
     qdf_s_match = f"{{ {s1} , {s2} }}"
 elif re.match(s4, s3):
     vv_bins = fg.lightgreen + Valid_Verdict[3]
     qdf_s_match = f"{{ {s3} , {s4} }}"
 else:
     vv_bins = fg.red + Valid_Verdict[4] # See note for a NUL verdict!

 # Validation Verdict on the two circuits as compared for a config. match.
 if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
     vv_circuits = fg.green + Valid_Verdict[3]
 if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[1]):
     vv_circuits = fg.yellow + Valid_Verdict[2]
 if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]):
     vv_circuits = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]
 if (vv_bins == fg.red + Valid_Verdict[4]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
     vv_circuits = fg.yellow + Valid_Verdict[2]
 if ((vv_bins == fg.red +
      Valid_Verdict[4]) or (vv_bins == fg.lightgreen + Valid_Verdict[3])) and (DeltaP_verdict == fg.silver + Valid_Verdict[4]):
      vv_circuits = f'{{ {fg.lightred + Valid_Verdict[4]} , {False} }}' # A NUL or false match verdict! See also next line/condition
 if (float(df3Mem) <= 1/3 and sample_data.min(axis=0)[1]<= 1/3) and (ibmq_p_max >= 1/2):
     vv_circuits = f'{fg.lightred}{{ {Valid_Verdict[4]} , {fg.yellow}{Valid_Verdict[1]}{fg.lightred} }}' # A NUL match verdict! In
short,
      # a state transition matrix with all elements between certain circuit components return a 0 state or null match and
      # state 2 for one of the two circuits! See P's preliminary analysis for the measured QDF circuit.
      # This circuit event outcome can also be interpreted as a rejected match between the two circuits, based on their
configuration.
 if (vv_bins == fg.red + Valid_Verdict[4]) and (Valid_Verdict == Fore.LIGHTYELLOW_EX + Valid_Verdict[2]):
     vv_circuits = fg.red + Valid_Verdict[2]
 #-------------------------------------------------------------------------------------------------
 print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Qubit Sets:      \033[4mValidation Verdict\033[0m
  {fg.yellow}  \
 between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
 \n   is a/an: \033[4m{vv_bins + ' Match ' + qdf_s_match}\033[0m {Fore.LIGHTYELLOW_EX}")

 print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Circuits:      \033[4mValidation Verdict\033[0m
  {fg.yellow}  \
 between the two QDF circuit configurations from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
 \n   is a/an: \033[4m{vv_circuits} Match \033[0m {Fore.LIGHTYELLOW_EX}")

 # Log validation verdict results on simulation and dataset analysis.
 entry_stage = 2
 sim_state[0] = f"Δp Data: {Valid_Verdict}, QDF Qubit Sets: {vv_bins + ' Match ' + qdf_s_match},  QDF Circuits: {vv_circuits} Matc
 sim_log()

 print(f"{hline + Fore.LIGHTMAGENTA_EX}\n *- The IBM QDF circuit can be reconfigured for worst and best case Hamiltonian scenarios
 \n    given the expected QDF measurement outcomes from the collected QFLCA datasets for a QFLCC.\
 \n\x1b[38;5;226m Simulation completed on: \033[1;32m{fg.yellow+bg.blue+str(datetime.datetime.now())+Back.RESET} \
 \n{hline + Fore.RESET}"), sleep(4)

 dir_flag = 0
 print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED --->\033[0m"
       + Fore.LIGHTGREEN_EX + Back.RESET)
```

```
# Log the end of simulation and dataset analysis.
entry_stage = 3
sim_state[0] = " QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED "
sim_log()
```

▼ sim_log( ) function code:

To log simulation events of the IBM QDF circuit and the selected dataset by the user for analysis.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
1063  def sim_log():
1064   ######################################
1065   # To log simulation checkpoints/steps
1066   ######################################
1067   global entry_stage
1068
1069   entry_stage += 1
1070   idxplus = len(res) + entry_stage
1071   subprocess.run("echo {}- Checkpoint logged on {} for file # {}\
1072   as {} parallel to simulation run file {{ QDF-LCode_IBMQ-2024-codable }}, \
1073   Simulation State ///--{}--///".format(idxplus, now.strftime("%Y-%m-%d %H:%M:%S"),
1074                                         idx, fileresult[idx-1], sim_state), shell=True, stdout=file_)
1075   """End of simulation log."""
```

▼ Expand/Collapse All. . .

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

12 Alice & Bob's Quantum Doubles:

12.1 Demo Files and Source Code

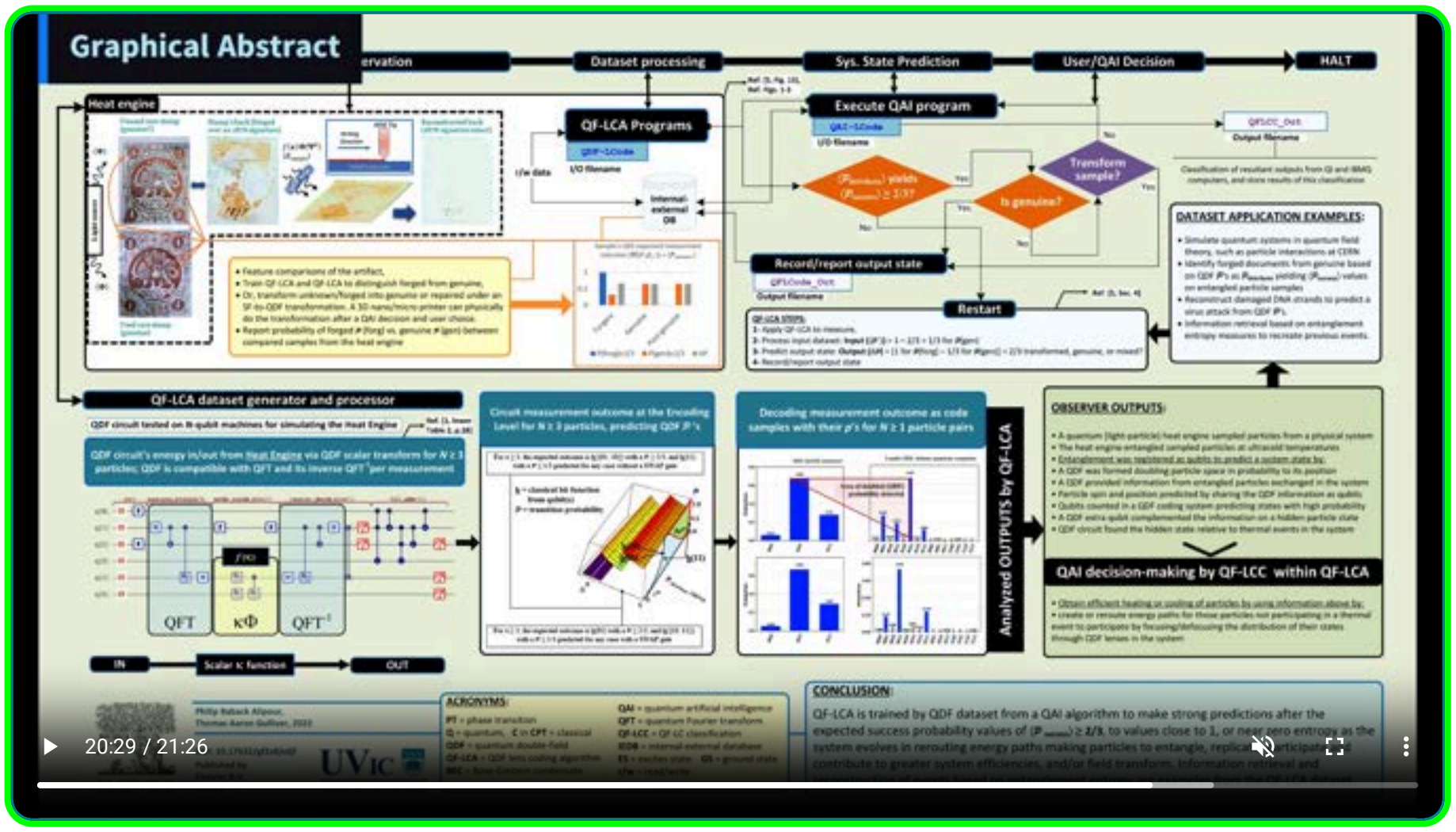


Figure 8: QDF Game Intro Demo: 880 MB mp4 file. ∇ | For Subtitles: SRT file. ∇

▼ QDF game description (Start of Heading & Text) …

QDF game as part of QFLCC in QFLCA: program description from QAI-LCode_QFLCC.py

```
942 ##############################################################################################
943 # These are the QDF game description, logo and copyright details.
944 ##############################################################################################
945 __game_banner__ = """\n"\033[96m( ° ͜ʖ °) vs ( °( ° ͜ʖ °(▧ ͜ʖ▣)ノ ° ͜ʖ °) °) vs [$(  ◠ ͜ʖ ◠)$]  vs ¯\_( ◐ ͜ʖ◐ )_/¯ vs \
946  ─=≡Σ((( つ◕_ʖ◕)つ [♥]]] [♦]]] [♣]]] [♠]]] ─=≡Σ(((  (☆ ͜ʖ ☆)  |||||||||||||||||||| \033[92m"""
947 __game_description__= "Alice & Bob's Quantum Doubles... This QFLCC game version of the QAI-LCode is basic. \
948 More revisions to come, as the dataset grows on the number of trials on 1 or more QDF circuits for simulating. \
949 the QDF game. The QDF game simulates events of the dataset's quantum and classical parameters in observing a \
950 thermodynamic system = { QDF game environment and its participants Alice, Bob, Eve, Audience } based on Refs. [1-2,4] \
951 of the Data in Brief, Elsevier J. article."
```

▼ QDF Game Update (2024): Chap. 04 Demo for Chap. 02 (2024):

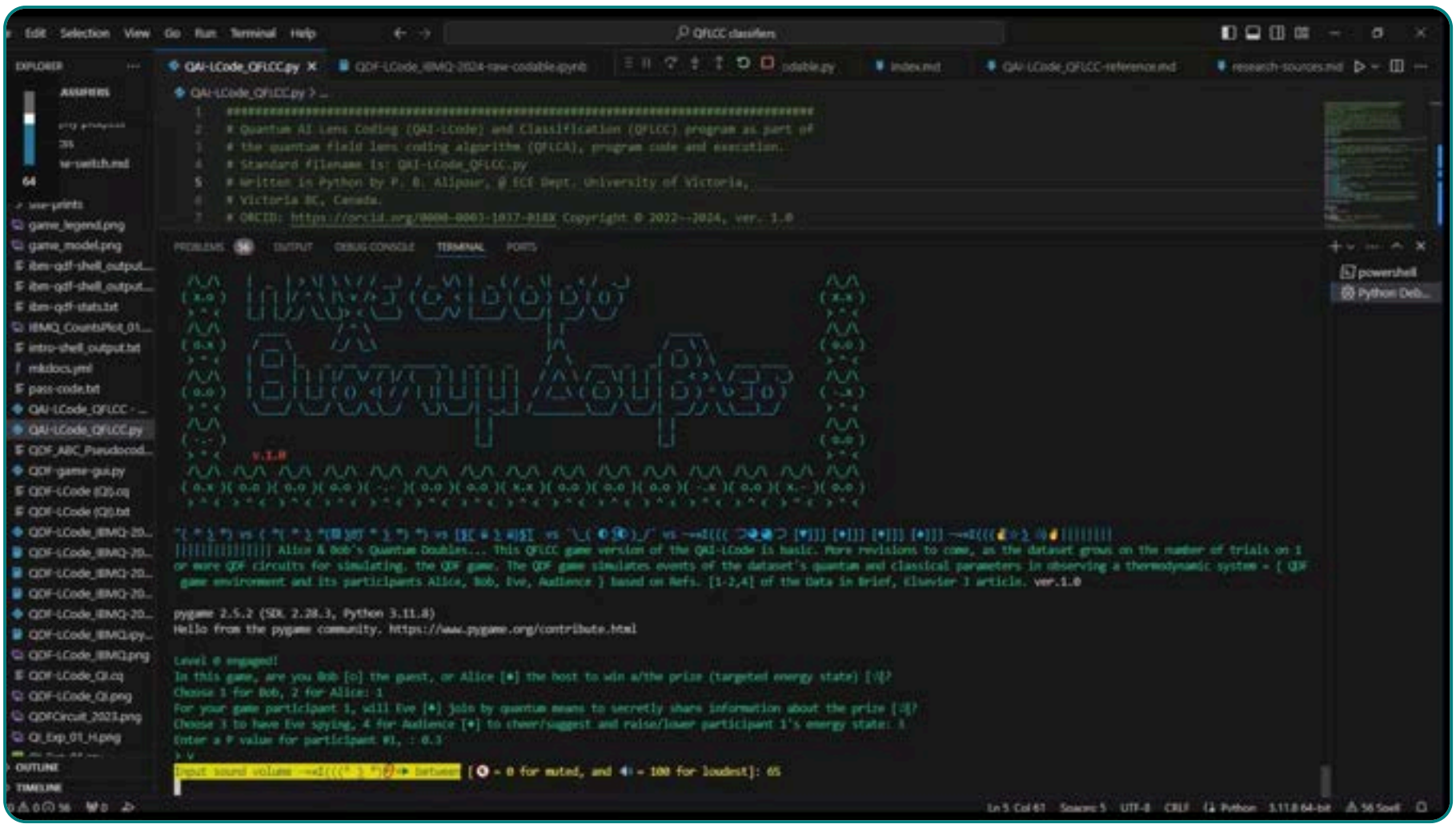
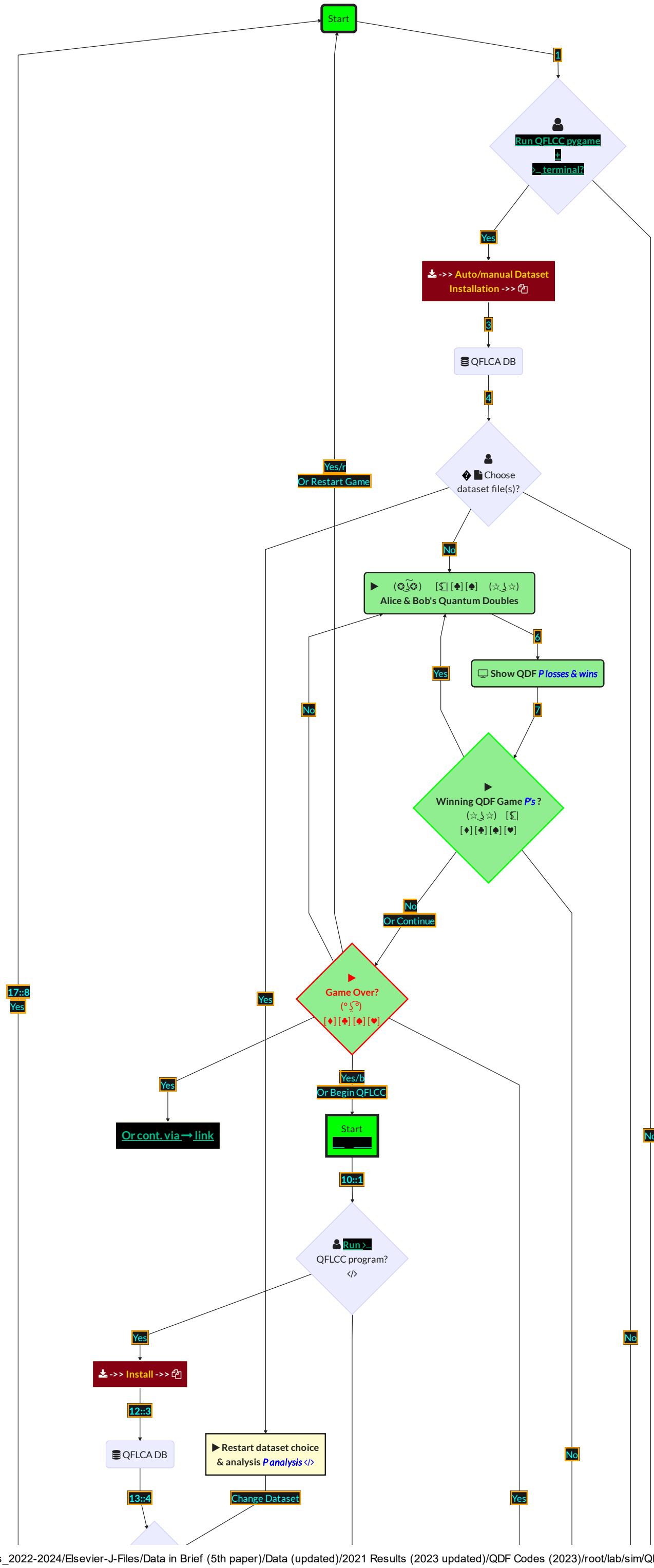


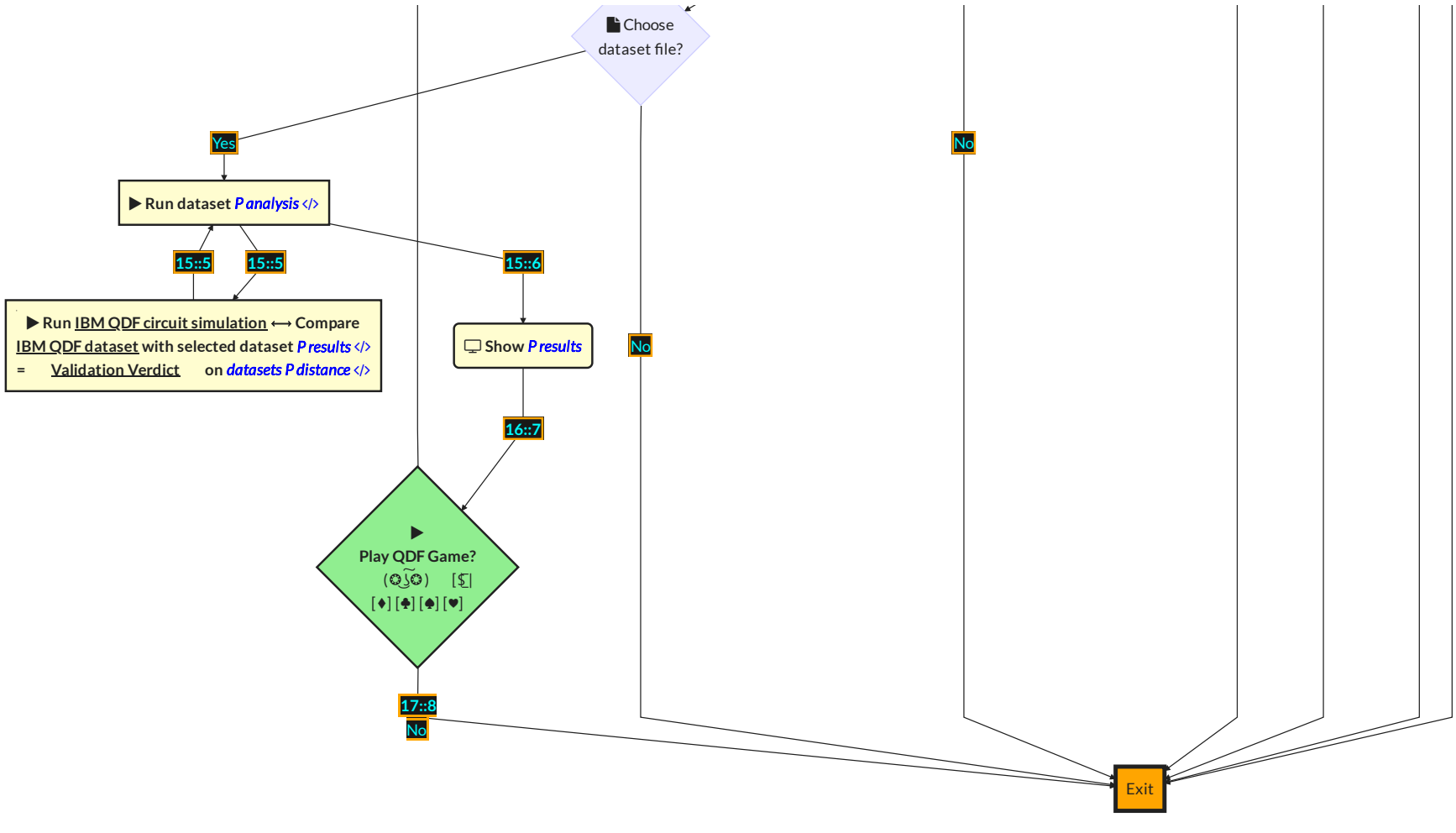
Figure 9: QDF Game 2024 Update: 635 MB mp4 file. ∇ | For Subtitles: SRT file. ∇

▼ Expand/Collapse All. . .

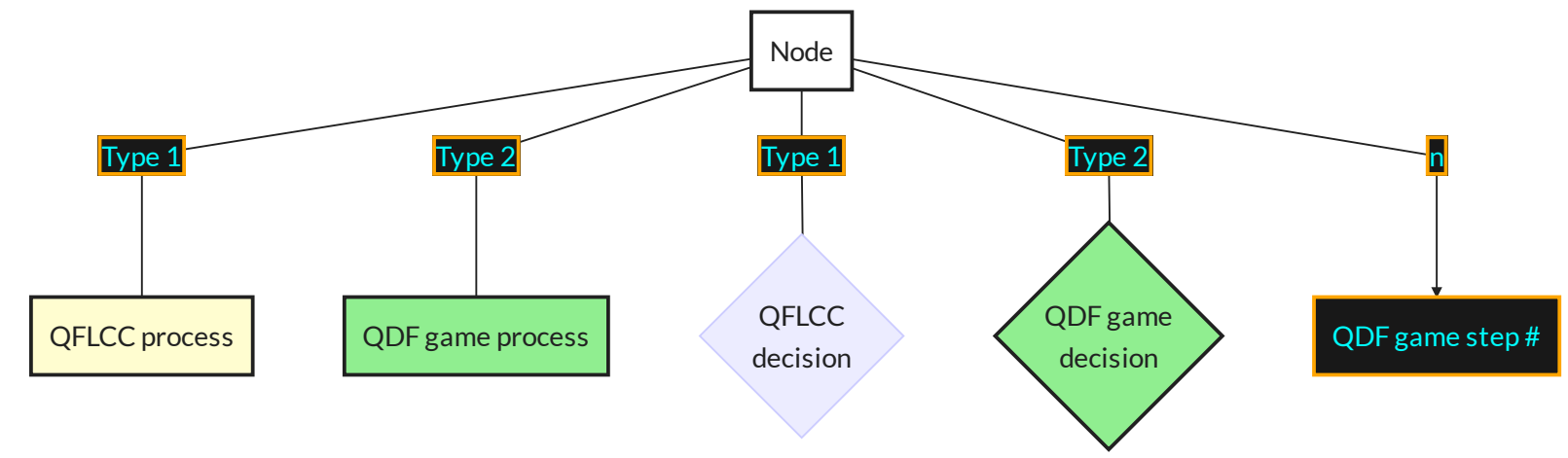
* Click Expand All to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click Collapse All to selectively view a code block from QAI-LCode_QFLCC.py on this page.

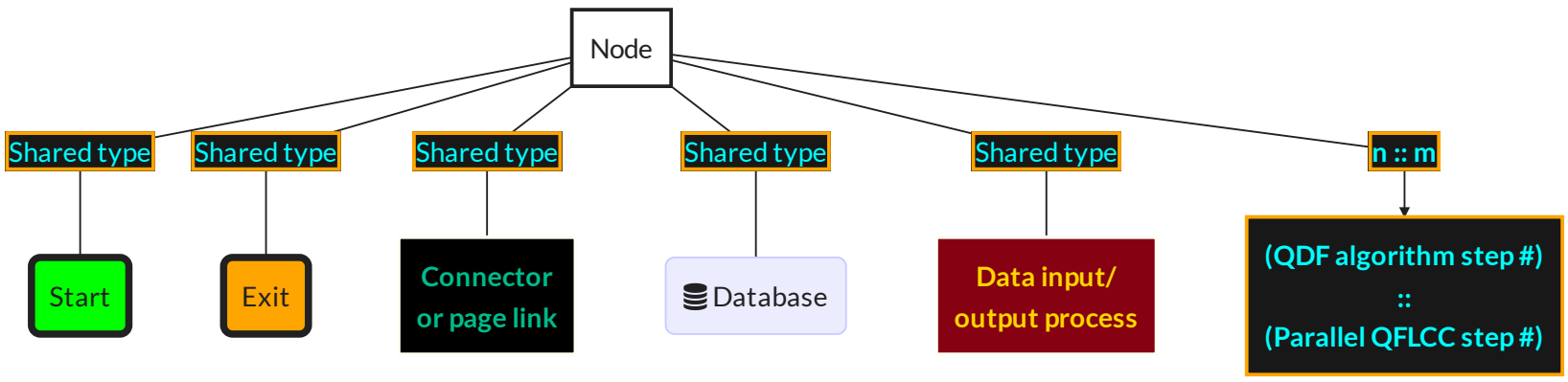
Expand All Collapse All


Start
1
Run QFLCC pygame
+
>_ terminal?
Yes
->> Auto/manual Dataset Installation ->>
3
QFLCA DB
4
Yes/r
Or Restart Game
Choose dataset file(s)?
No
▶ (ᵔ◡ᵔ) [♤] [♠] [♠] (☆ل☆)
Alice & Bob's Quantum Doubles
6
Yes
Show QDF P losses & wins
No
7
▶
Winning QDF Game P's ?
(☆ل☆) [♤]
[♦] [♣] [♠] [♥]
No
Or Continue
17::8
Yes
▶
Game Over?
(° ͜ʖ°)
[♦] [♣] [♠] [♥]
Yes
Yes
Yes/b
Or Begin QFLCC
Or cont. via ➞ link
Start
10::1
Run >_
QFLCC program?
</>
No
No
No
Yes
Yes
->> Install ->>
12::3
QFLCA DB
13::4
▶ Restart dataset choice & analysis P analysis </>
Change Dataset



▼ Legend. . .





▼ Flowchart description. . .

This 17-step diagram represents the flow of the QDF game and QFLCC steps of the QFLCA program. These steps are presented in parallel, side-by-side, between programs/algorithms labeled as `n::m` e.g., [`17::8`] denotes `step 17 (from the QDF game algorithm)::8 (as part of the QFLCC algorithm)` after a successful installation and program run by the user. It starts with dataset files installation, then the selection of one of the datasets for probability `P analysis`. The results are further compared to the `IBM QDF circuit dataset` to `validate P results` between the two datasets by running the IBM QDF simulation module from the QFLCC program. Finally, the program `prompts` ❯_ `the user to continue` by playing the QDF game for further validation of dataset results (`P analysis`). The diagram continues on the relevant page linked here .

▼ Expand/Collapse All. . .

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

13 QAI-LCode_QFLCC Source Code

Downloadable source code is:

QAI-LCode_QFLCC file download

13.1 Common QDF game constants and variables

▼ Examples are: a constant = ASCII characters and a variable entry_stage ≥ 0 code:

These are QDF Game's frequently used constants and variables within QAI-LCode_QFLCC.py.

- A sample's expected output frequently called by the `QDF game functions`:

```
(°ʖ°)  [$]—=≡Σ((  (⁰¹ʖ¹⁰)  —=≡Σ((  (°ʖ°)
```

- A sample's sound volume change from the `promptGame()` function call via `Volume Settings` section of the QDF game module, `basic mode`:

```
> v
Input sound volume —=≡Σ(((° ʖ °)) between [🔇 = 0 for muted, and 🔊 = 100 for loudest]: 56
Do you want to continue in prompt mode to input command? [y/n]y
Input the relevant command according to 'h' or 'help', 'cv', 'v' or 'volume' for sound volume change...
```

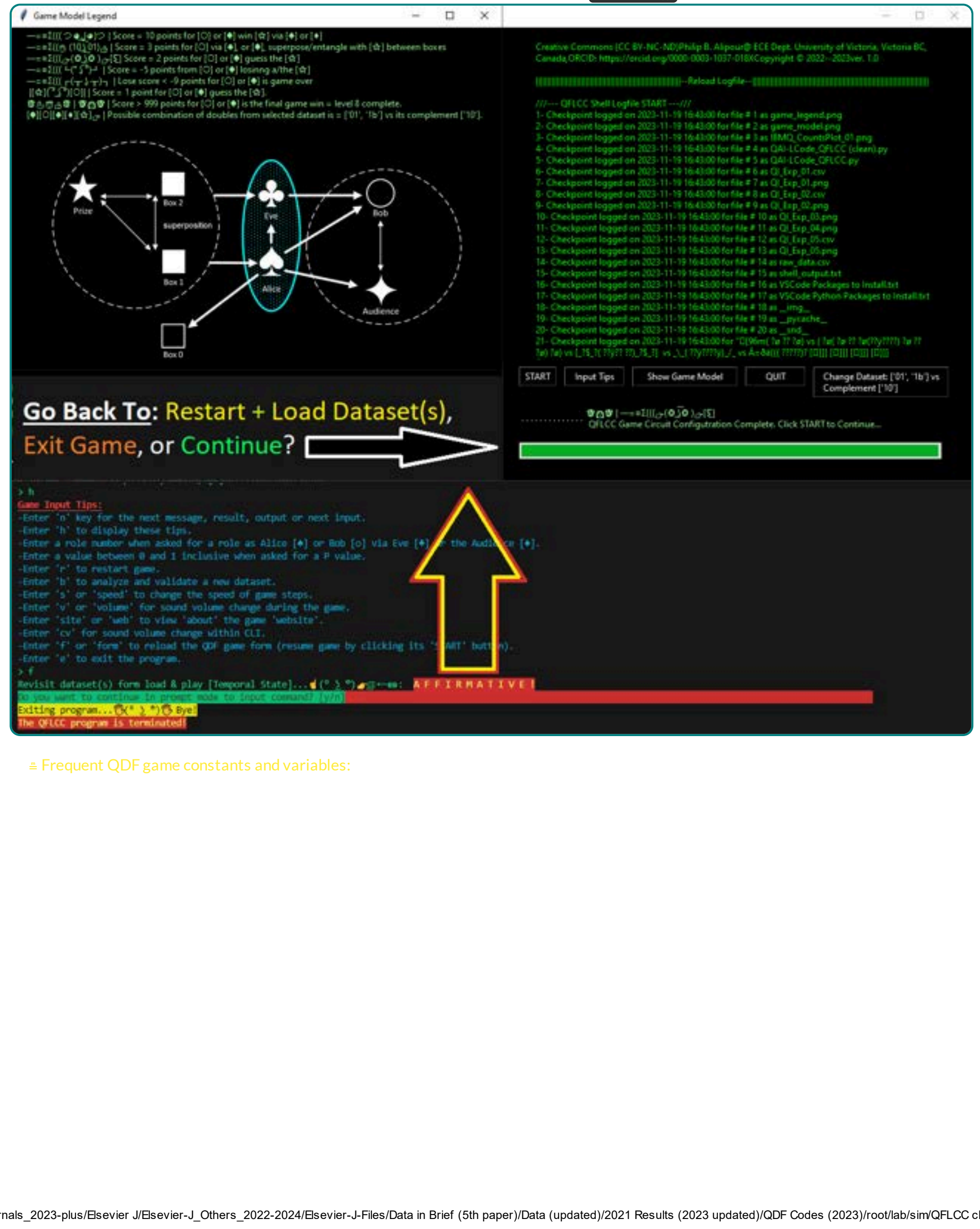
- A sample's sound volume change from the `promptGame()` function call via `Volume Settings` section of the QDF game module, `expert mode`:

```
> cv
Change game volume —=≡Σ(((° ʖ °))... Choose from this range:  [{quietist 🔇 ⇄ 🔉 = [0 to 6], quiet 🔉 = 6.1}, loudest 🔊 = 100]: 78.6
Volume converted and adjusted from the upper desktop's endpoint volume range of [-29, 0] = -5.8850
Do you want to continue in prompt mode to input command? [y/n]n
List of tried P's:  [0.7, 0.5, 0.99]
```

- A sample's game speed change from the `promptGame()` function:

```
> s
Change game speed —=≡Σ((( つ◕‿◕)つ... Choose from this range:  [fastest = 0.1, slowest = 2]:0.5
```

- Reloading GUI form to select or combine a dataset after unlocking a passcode by the `pass_code()` function:



▼ ≡ Frequent QDF game constants and variables:

```
################################################################################
# Alice & Bob's Quantum Doubles game starts from here to validate data results...
################################################################################
__game_version__ = "ver.1.0"

__game_logo__="""
\033[92m  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\
\033[92m ( o.o )( o.o )( o.o )( o.o )( x.x )( o.o )( o.o )( o.o )( o.o )( o.o )( x.- )( o.o )( o.o )( o.o )( -.- )
\033[92m  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <
\033[92m  /\_/\  \033[96m    ___ __                 ___     ____          ___                                  \033[92m   /\_/\
\033[92m ( o.o ) \033[96m   / _ \\ \               /   \   |  _ \       / _ \ //                              \033[92m   ( o.o )
\033[92m  > ^ <  \033[96m  | |_| |\ \  ___   _____   \ 0 /   | |_) ) ___ | |_) )/___                          \033[92m   > ^ <
\033[92m  /\_/\  \033[96m  |  _  | > \| \ \ / / __)  / _ \/\ |  _ ( / _ \|  _ < /  ._)                         \033[92m   /\_/\
\033[92m ( x.o ) \033[96m  | | | |/ ^ \ |\ v /> _)  ( (_>  < | |_) | (_) ) |_) | () )                         \033[92m  ( x.x )
\033[92m  > ^ <  \033[96m  |_| |_/_/ \_\_)> < \___)  \___/\/ |____/ \___/|  __/ \__/                          \033[92m   > ^ <
\033[92m  /\_/\  \033[96m    ____        / ^ \                        | |             _____                   \033[92m   /\_/\
\033[92m ( o.x ) \033[96m   / __ \      /_/ \_\                       |/\            / _ \ \                  \033[92m  ( o.o )
\033[92m  > ^ <  \033[96m  | |__| |_  _ __  ___ _____ _  _ _    _     /  \  ___  _  _| |_) ) \  ___ ____      \033[92m   > ^ <
\033[92m  /\_/\  \033[96m  |  __  | | | |/  \/ / |/ (   ) | | | | |   / /\ \ / _ \| | |  _ < > \ / __)  ._)  \033[92m   /\_/\
\033[92m ( o.o ) \033[96m  | |__| | |_| ( ()  <| / / | || |_| | |_| |  / /__\ ( (_) ) |_| | |_) ) ^ \> _| () )   \033[92m  ( -.x )
\033[92m  > ^ <  \033[96m   \____/ \___/ \__/\_\__/   \_)\___/| ._,_| /________\___/ \___/|  __/_/ \_\___)__/    \033[92m   > ^ <
\033[92m  /\_/\  \033[96m                                    | |                       | |                      \033[92m   /\_/\
\033[92m ( -.- ) \033[96m                                    |_|                       |_|                      \033[92m  ( o.o )
\033[92m  > ^ <  \033[96m  \033[91m v.1.0                                                                       \033[92m   > ^ <
\033[92m  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\  /\_/\
\033[92m ( o.x )( o.o )( o.o )( o.o )( -.- )( o.o )( o.o )( x.x )( o.o )( o.o )( o.o )( -.x )( o.o )( x.- )( o.o )
\033[92m  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <  > ^ <
"""
################################################################################
# Instantiating variables and constants of animated states of loser, winner,
# dual(superposition), unknown. Alternatively, lists can be created in the
# future tests after threads, timer or sleep() and subprocess runs for combined
# datasets. For example, the constants excited1 and excited2 can be redefined
# in a list as excited = [excited1, excited2], or by the removal of the two
# constants hardcoding the emojis in excited = ["−=≡Σ(((  つ◕‿◕)つ",
# "−=≡Σ(((  (☆ ͜ʖ ☆)  [☆]"] and later called for the participant's win, as
# animated by the function winner_show() within its for-loop.
################################################################################
prize="[☆]"
alice="[♠]"
bob="[○]"
eve="[♣]"
audience="[♦]"
crownBob="♕       ♕"
crownAlice="♔   ♔"
here="  "
go="   "

excited1 = "−=≡Σ(((  つ◕‿◕)つ"
excited2 = "−=≡Σ(((  (☆ ͜ʖ ☆)  [☆]"

grounded1= "−=≡Σ(((ɾ(° ͜ʖ °)ﾉ"
grounded2= "−=≡Σ(((ʟ(° ͜ʖ °)ﾉ"
grounded3= "−=≡Σ(((ɾ(╥ ͜ʖ ╥)ɾ"

dual3="−=≡Σ((   (11 ʖ̯ 00)  "
dual1="−=≡Σ((   (10 ʖ̯ 01)  "
dual2="−=≡Σ((   (01 ʖ̯ 10)  "

guesser1="−=≡Σ(((  (◐ ʖ̯◐ )  "
guesser2="−=≡Σ(((  (◐ ʖ̯◐ )  "
guesser3="−=≡Σ(((  (❂ ʖ̯❂ )  [$‿|"

classical1="−=≡Σ((  (° ͜ʖ °)  "
classical2="−=≡Σ((  (° ͜ʖ °)  "
classical3="−=≡Σ(((  (❂ ʖ̯❂ )  [$‿|"

errorP1=" ¯\_( ◐ ʖ̯◐ )_/¯"
errorP2=" ¯\_( ◕ ʖ̯◕ )_/¯"

aimTarget=" ||[$⟨ ◠ ͜ʖ ◠)$⟩||"
helperTarget1=" |[☆]( ° ͜ʖ °)[○]|"
helperTarget2=" |[♠]( ° ͜ʖ °)[☆]|"
################################################################################################
# These are the QDF game description, logo and copyright details.
################################################################################################
__game_banner__ = """\n"\033[96m( ° ͜ʖ °) vs ( °( ° ͜ʖ °(▧ ͜ʖ▨)ʖ ° ͜ʖ °) °) vs [$⟨ ◠ ͜ʖ ◠)$⟩  vs \
 ¯\_( ◐ ʖ̯◐ )_/¯ vs −=≡Σ(((  つ◕‿◕)つ [♥]]] [♦]]] [♠]]] [♠]]] −=≡Σ(((  (☆ ͜ʖ ☆)  |||||||||||||||||||| \033[92m"""
__game_description__= "Alice & Bob's Quantum Doubles... This QFLCC game version of the QAI-LCode is basic. \
More revisions to come, as the dataset grows on the number of trials on 1 or more QDF circuits for simulating. \
the QDF game. The QDF game simulates events of the dataset's quantum and classical parameters in observing a \
thermodynamic system = { QDF game environment and its participants Alice, Bob, Eve, Audience } based on Refs. [1-2,4] \
of the Data in Brief, Elsevier J article."
print(Back.RESET+fg.lightgreen,__game_logo__,__game_banner__ + __game_description__ ,
      Back.RESET+fg.silver+__game_version__,"\n"), sleep(2)
"""- Colored banner to introduce the QDF game and its goal. These common constants and variables are
     frequently used during game play."""
entry_stage = 4 # Set this to 0 if the game is run as a separate file or program.
dir_flag = 0 # Directory flag is set to 1 if dataset directory access is asked by user to analyze in safe mode.

qdf_bits_flag = [False, False] # Flag state 0 == hide, 1 == show the qdf bit values from any dataset.
qdf_bits_flag[0] = False # By default set at least this flag to 0 == hide the qdf bit values from any dataset.
```

```
1503
1504  def qdf_bits_flagset():
1505   # A function to switch between states of 0 == hide and 1 == show for the qdf bit values of any dataset.
1506   global qdf, bitpair, qdf_bits_flag
1507   if (qdf_bits_flag[0] == False) and (qdf_bits_flag[1] == False):
1508       qdf = '[##]' # Mask this qdf value in case of qdf_bits both flags are false.
1509       bitpair = '[##]' # Mask this bitpair value in case of qdf_bits both flags are false.
1510   if (qdf_bits_flag[0] == False) and (qdf_bits_flag[1] == True):
1511       qdf = qdf # Unmask this qdf value in case of qdf_bits 2nd flag is true.
1512       bitpair = bitpair # Unmask this bitpair value in case of qdf_bits 2nd flag is true.
```

▼ Expand/Collapse All…

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

14 QAI-LCode_QFLCC Module for the QDF Game

▼ Module description and code usage warning:

The current main module is the source file `QAI-LCode_QFLCC.py`. This module will be split into the `QDF-game.py` and `QAI-LCode_QFLCC.py` for import. The `QDF-game.py` is already within the file `QAI-LCode_QFLCC.py` file. The updated version to import the smaller `QAI-LCode_QFLCC.py` for the `QDF game` is under development as the next version of the code. QAI-LCode_QFLCC file download

14.1 `ExitMyProgram`

Bases: `Exception`

- Exception used to exit program.

▼ Source code in `QAI-LCode_QFLCC.py`

```
104
105  class ExitMyProgram(Exception):
         """- Exception used to exit program."""
```

14.2 `bg`

▼ Source code in `QAI-LCode_QFLCC.py`

```
 91  class bg:
 92          black='\033[40m'
 93          red='\033[41m'
 94          green='\033[42m'
 95          orange='\033[43m'
 96          blue='\033[44m'
 97          purple='\033[45m'
 98          cyan='\033[46m'
 99          lightgrey='\033[47m'
100          """- Colored background (bg) class for sectioning and highlighting the
101          QF-LCC algorithm step, checkpoint, computer operation, error, or HALT."""
```

14.2.1 `lightgrey = '\x1b[47m'` `class-attribute` `instance-attribute`

- Colored background (bg) class for sectioning and highlighting the QF-LCC algorithm step, checkpoint, computer operation, error, or HALT.

14.3 `fg`

▼ Source code in `QAI-LCode_QFLCC.py`

```
class fg:
        black='\033[30m'
        silver="\033[0;38;5;7m"
        red='\033[31m'
        green='\033[32m'
        orange='\033[40m'
        blue='\033[34m'
        purple='\033[35m'
        cyan='\033[36m'
        lightgrey='\033[37m'
        darkgrey='\033[90m'
        lightred='\033[91m'
        lightgreen='\033[92m'
        yellow='\033[93m'
        lightblue='\033[94m'
        pink='\033[95m'
        lightcyan='\033[96m'

        """- Colored foreground (fg) class for sectioning and highlighting the
        QF-LCC algorithm step, checkpoint, computer operation, error, or HALT."""
```

14.3.1 `lightcyan = '\x1b[96m'` `class-attribute` `instance-attribute`

- Colored foreground (fg) class for sectioning and highlighting the QF-LCC algorithm step, checkpoint, computer operation, error, or HALT.

14.4 `build(debug)`

- Build production assets.

▼ Source code in `QAI-LCode_QFLCC.py`

```
@cli.command()
@click.option("-d", "--debug", help="Include debug output.")
def build(debug):
    """- Build production assets."""
```

14.5 `cli()`

Main entry point.

▼ Source code in `QAI-LCode_QFLCC.py`

```
@click.group()
def cli():
    """Main entry point."""
```

▼ Expand/Collapse All…

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

15 QDF Game Functions and Calls

▼ ≡ QDF game step log, colors, GUI and its other settings:

To log game steps as the game progresses, including errors, exit, GUI and help for the user. Function() list from top to bottom as ordered:

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def entry_add():
#################################################################################################
# This function logs program events onto the stdout file
#################################################################################################
     global entry_stage  # Redefine the entry count as global.
     entry_stage+=1
     idxplus = len(res)+entry_stage
     subprocess.run("echo {}- Checkpoint logged on {} for the QFLCC Game:\
 Alice and Bob's Quantum Doubles {}".format(idxplus, now.strftime("%Y-%m-%d %H:%M:%S"), __game_version__),
 shell=True, stdout=file_)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def flash_color(object, color):
    object.config(foreground = next(color), bg='black')
    root.after(100, flash_color, object, color)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def display_bye_msg():
    # Play Windows exit sound.
    winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
    root.wm_attributes("-topmost", 1)
    root.withdraw() # Hide this form.
    bye = msgbox.askquestion(title=f"Alice & Bob's Quantum Doubles {__game_version__} Message",
                             message = 'Are you sure you want to exit? If Yes, ...   (° ͜ʖ °)   Goodbye!',
                             icon = 'warning', parent=root)
    if bye == 'yes':
        game_sound = pygame.mixer.Sound('__snd__/goodbye1.wav')
        game_sound.play()
        Quit()
    else:
        root.wm_attributes("-topmost", 1)
        root.deiconify() # Unhide this form.
    """Exit game if user chooses after the goodbye message."""
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def flagClose():
    root.withdraw() # Hide this form.
    root.quit()
    #-------------------------------------------------------------------------------
    # Enable the next lines to destroy this form after quit (or by duration afterwards
    # using sleep) as a permanent close solution. This is to not revisit the dataset(s)
    # load & play root form, as it is already destroyed after t seconds!
    #-------------------------------------------------------------------------------
    #sleep(50)       # Dont close for t = 50 seconds, for example...
    #root.destroy()  # To destroy the root form.
    """Quit or destroy the game app form and get back to CLI as user's interface. The quit
command gives user the illusion that the form is closed/gone, which in fact can be later
called upon by the option 'form' from the terminal. """
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def Quit():
    game_sound = pygame.mixer.Sound('__snd__/goodbye1.wav')
    game_sound.play()
    print(fg.black + Back.YELLOW +'\nExiting program...   (° ͜ʖ °)    Goodbye!'+ Back.RESET), sleep(3)
    print(fg.yellow + Back.RED+"The QFLCC program is terminated!" + Back.RESET)
    sys.exit(1) # Force this in case of abnormal exit or program termination.
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def Help():
     msgbox.showinfo(title='Input Tips', message='Input tips:\n \n-Enter \'n\' key or \'next\' for \
the next message or input. \n-Enter \'h\' or \'help\' to display these tips. \n-Enter the number \
for a game player role when prompted. \n-Enter a value between 0 and 1, or 0 or 1 for a P value. \
\n-Enter \'site\' or \'web\' to view \'about\' the game \'website\'. \
\n-Enter \'s\' or \'speed\' to change the speed of game steps. \
\n-Enter \'v\' or \'volume\' for sound volume change during play. \n-Enter \'cv\' for sound volume \
change within CLI. \n-Enter \'f\' or \'form\' to reload this form. \
\n-Enter \'dir\' or \'sm dir\' for a new dataset analysis & simulation to feed this game.*\
\n*-This command requires a passcode to see dataset results in a Safe Mode (SM:>>) environment!\
\n-Enter \'e\' or \'exit\' to exit the game.')
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def open_new_win():
   ###########################################################################
   # This function creates a new window based on the QDF game legend and rules. #
   ###########################################################################
   imagelist1 = ["game_legend.png"]
   photo = PhotoImage(file=imagelist1)
   width = photo.width()
   height = photo.height()
   fore_win=Toplevel(root)
   fore_win.wm_attributes("-topmost", 1) # these two lines will focus on the window not terminal
   fore_win.focus_force()
   fore_win.title("Game Model Legend")
   fore_win.geometry("650x450")
   fore_win.configure(background='black')
   image = Image.open('game_legend.png').convert("RGB")
   resized = image.resize((550, 300),Image.LANCZOS) # Resize the image and antialias it from the uploaded file.
   display = ImageTk.PhotoImage(resized)
   label = Label(fore_win, image=display, background="black") # Display it within a label.
   legend_label = Label(fore_win,
                        text=excited1+f' | Score = 10 points for {bob} or {alice} win {prize} via {eve} or
{audience} \n'
                        + dual1+f' | Score = 3 points for {bob} via {eve}, or {alice}, \
superpose/entangle with {prize} between boxes \n'+ guesser3+f' Score = 2 points for {bob} or {alice} guess the
{prize} \n'
                        +  grounded2+f' | Score = -5 points from {bob} or {alice} losing a/the {prize} \n'+
grounded3
                        +f' | Lose score < -9 points for {bob} or {alice} is game over \n'+ helperTarget1
                        +f' | Score = 1 point for {bob} or {alice} guess the {prize}.  \n'+ crownBob+' | '+
crownAlice
                        +f' | Score > 999 points for {bob} or {alice} is the final game win = level 8 complete.  \n'
                        + alice + bob + eve + audience + prize +go
                        +f' | Possible combination of doubles from selected dataset \
is = ' + buttonInvisible["text"] + '.\n', justify="left" , bg="black",  fg="lightgreen").pack()
   label.image = display
   label.pack()
   # Set minimum the foreground window size value
   fore_win.minsize(650, 450)
   # Set maximum the foreground window size value
   fore_win.maxsize(650, 450)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def restart_QFLCC():
##########################################################
# The following restarts QFLCC for choosing a dataset  #
##########################################################
   global win_min
   #root.iconify()
   win_min == True
   flagClose() # An option to choose between destroying/quit the present form.
   subprocess.run(["python", "QAI-LCode_QFLCC.py"])
```

- ▼ ≡ QDF game intro GUI. . .

```
1527  ######################################################################
1528  # Initialize pygame, then sound mixer to play introductory retro
1529  # music and welcome message sound files and create other windows
1530  # as a GUI.
1531  ######################################################################
1532  from tkinter import *
1533  import tkinter as tk
1534  from itertools import cycle
1535  import pygame  # To play music or sounds in the QDF game.
1536
1537  def flash_color(object, color):
1538  object.config(foreground = next(color), bg='black')
1539  root.after(100, flash_color, object, color)
1540  """ Coloring the flash object from pygame for the user's interface (UI)."""
1541
1542  pygame.init()
1543  pygame.mixer.init()
1544
1545  game_sound = pygame.mixer.Sound('__snd__/game_music2.wav')
1546  game_sound.play(), sleep(2)
1547
1548  game_sound = pygame.mixer.Sound('__snd__/welcome2.wav')
1549  game_sound.play()
1550
1551  """ Initialize pygame and sound mixer to play introductory retro
1552  music and welcome message sound files and other windows as a GUI."""
1553
1554  from PIL import Image, ImageTk
1555  import tkinter.messagebox as msgbox
1556
1557  def display_bye_msg():
1558  # Play Windows exit sound.
1559  winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
1560  root.wm_attributes("-topmost", 1)
1561  root.withdraw() # Hide this form.
1562  bye = msgbox.askquestion(title=f"Alice & Bob's Quantum Doubles {__game_version__} Message",
1563                           message='Are you sure you want to exit? If Yes, ...   (° ͜ʖ °)   Bye!',
1564                           icon='warning', parent=root)
1565  if bye == 'yes':
1566       game_sound = pygame.mixer.Sound('__snd__/goodbye1.wav')
1567       game_sound.play()
1568       Quit()
1569  else:
1570       root.wm_attributes("-topmost", 1)
1571       root.deiconify() # Unhide this form.
1572  """Exit game if user chooses after the bye message."""
1573
1574  def flagClose():
1575  root.withdraw() # Hide this form.
1576  root.quit()
1577  #-------------------------------------------------------------------------------
1578  # Enable the next lines to destroy this form after quit (or by duration afterwards
1579  # using sleep) as a permanent close solution. This is to not revisit the dataset(s)
1580  # load & play root form, as it is already destroyed after t seconds!
1581  #-------------------------------------------------------------------------------
1582  #sleep(50)       # Dont close for t = 50 seconds, for example...
1583  #root.destroy()  # To destroy the root form.
1584  """Quit or destroy the game app form and get back to CLI as user's interface. The quit
1585  command gives user the illusion that the form is closed/gone, which in fact can be later
1586  called upon by the option 'form' from the terminal. """
1587
1588  def Quit():
1589       game_sound = pygame.mixer.Sound('__snd__/goodbye1.wav')
1590       game_sound.play()
1591       print(fg.black + Back.YELLOW +'\nExiting program...   (° ͜ʖ °)    Bye!'+ Back.RESET), sleep(3)
1592       print(fg.yellow + Back.RED+"The QFLCC program is terminated!" + Back.RESET)
1593       sys.exit(1) # Force this in case of abnormal exit or program termination.
1594  """Exit program and system with a message before the terminal."""
1595
1596  def Help():
1597       msgbox.showinfo(title='Input Tips', message='Input tips:\n \n-Enter \'n\' key or \'next\' for \
1598  the next message or input. \n-Enter \'h\' or \'help\' to display these tips. \n-Enter the number \
1599  for a game player role when prompted. \n-Enter a value between 0 and 1, or 0 or 1 for a P \
1600  value. \n-Enter \'s\' or \'speed\' to change the speed of game steps. \n-Enter \'v\' or \'volume\' for \
1601  sound volume change during play. \n-Enter \'cv\' for sound volume change within CLI. \n-Enter \'f\' or \
1602  \'form\' to reload this form. \n-Enter \'e\' or \'exit\' to exit the game.')
1603  """Helping tips to start the game by choosing participants and a P value, according to the QDF game model."""
1604
1605  def open_new_win():
1606     ###############################################################################
1607     # This function creates a new window based on the QDF game legend and rules. #
1608     ###############################################################################
1609     imagelist1 = ["game_legend.png"]
1610     photo = PhotoImage(file=imagelist1)
1611     width = photo.width()
1612     height = photo.height()
1613     fore_win=Toplevel(root)
1614     fore_win.wm_attributes("-topmost", 1) # these two lines will focus on the window not terminal
1615     fore_win.focus_force()
1616     fore_win.title("Game Model Legend")
1617     fore_win.geometry("650x450")
1618     fore_win.configure(background='black')
```

```
    image = Image.open('game_legend.png').convert("RGB")
    resized = image.resize((550, 300),Image.LANCZOS) # Resize the image and antialias it from the uploaded file.
    display = ImageTk.PhotoImage(resized)
    label = Label(fore_win, image=display, background="black") # Display it within a label.
    legend_label = Label(fore_win,
                         text=excited1+f' | Score = 10 points for {bob} or {alice} win {prize} via {eve} or {audience} \n'
                         + dual1+f' | Score = 3 points for {bob} via {eve}, or {alice}, \
superpose/entangle with {prize} between boxes \n'+ guesser3+f' Score = 2 points for {bob} or {alice} guess the {prize} \n'
                         +  grounded2+f' | Score = -5 points from {bob} or {alice} losing a/the {prize} \n'+ grounded3
                         +f' | Lose score < -9 points for {bob} or {alice} is game over \n'+ helperTarget1
                         +f' | Score = 1 point for {bob} or {alice} guess the {prize}.  \n'+ crownBob+' | '+ crownAlice
                         +f' | Score > 999 points for {bob} or {alice} is the final game win = level 8 complete.  \n'
                         + alice + bob + eve + audience + prize +go
                         +f' | Possible combination of doubles from selected dataset \
is = ' + buttonInvisible["text"] + '.\n', justify="left" , bg="black",  fg="lightgreen").pack()
    label.image = display
    label.pack()
    # Set minimum the foreground window size value
    fore_win.minsize(650, 450)
    # Set maximum the foreground window size value
    fore_win.maxsize(650, 450)

win_min=0
def restart_QFLCC():
#########################################################
# The following restarts QFLCC for choosing a dataset  #
#########################################################
global win_min
#root.iconify()
win_min == True
flagClose() # An option to choose between destroying/quit the present form.
subprocess.run(["python", "QAI-LCode_QFLCC.py"])

#########################################################
# Mimic an animated GIF displaying a series of GIFs an
# animated GIF was used to create the series of GIFs with
# a common GIF animator utility.
# Buttons and usage help options are appended and displayed
# at the end of log file and other game file
# installation/download.
#########################################################
from tkinter import ttk
import asyncio

root = Tk()

# Adjust size...
#root.geometry("1550x600")
width = 1550 # Width
height = 600 # Height
screen_width = root.winfo_screenwidth()  # Width of the screen
screen_height = root.winfo_screenheight() # Height of the screen

# Calculate Starting X and Y coordinates for Window
x = (screen_width/2) - (width/2)
y = (screen_height/2) - (height/2)

root.geometry('+%d+%d' % (x, y))
root.wm_attributes("-topmost", 1) # These two lines will focus on the window not terminal.
root.focus_force()

def disable_event():
pass
root.protocol("WM_DELETE_WINDOW", disable_event)  # This disables the [X] button on the root window to satisfy user's game options.
     # No matter how many times the user clicks on [X], nothing happens, and so is directed to other exit or resume game options.

# Create a sprite bank (array list) to animate from files.
imagelist = ["__img__/logo_01.gif", "__img__/logo_01.gif", "__img__/logo_02.gif", "__img__/logo_02.gif",
          "__img__/logo_02.gif", "__img__/logo_03.gif",  "__img__/logo_03.gif", "__img__/logo_04.gif",
          "__img__/logo_05.gif", "__img__/logo_06.gif", "__img__/logo_06.gif", "__img__/logo_06.gif"]

# Create a sprite bank (array list) from the defined ASCII characters by their corresponding char variables.
charslist = [f'{crownBob} | {dual1}' , f'{crownBob} | {dual1}', f'{crownBob} | {dual2}',
          f'{crownBob} | {dual2}', f'{crownBob} | {dual3}', f'{crownBob} | {dual3},' f'{crownBob} | {excited1}',
          f'{crownBob} | {excited1}', f'{crownBob} | {excited1}', f'{crownBob} | {excited2}',
          f'{crownBob} | {excited2}', f'{crownBob} | {excited2}', f'{crownAlice} | {guesser1}',
          f'{crownAlice} | {guesser1}', f'{crownAlice} | {guesser2}', f'{crownAlice} | {guesser2}',
          f'{crownAlice} | {guesser3}', f'{crownAlice} | {guesser3}', f'{crownAlice} | {classical1}',
          f'{crownAlice} | {classical1}', f'{crownAlice} | {classical2}', f'{crownAlice} | {classical3}',
          f'{crownAlice} | {classical3}']
color_list = ["cyan", "yellow", "red", "yellow", "blue", "lightgreen"]

progressbar = ttk.Progressbar(length=750, style="green.Horizontal.TProgressbar")

# Full progress bar...
progressbar.step(99.9)
progressbar.place(x=950,y=550, width=560)

# Set minimum window size value.
root.minsize(1550, 600)

# Set maximum window size value.
root.maxsize(1550, 600)
```

```

# Extract width and height info.
photo = PhotoImage(file=imagelist[0])
width = photo.width()
height = photo.height()

canvas = Canvas(width=width, height=height)

# Position the canvas for the gif image.
canvas.place(x=30, y=20)

root.title(f"Alice & Bob's Quantum Doubles {__game_version__}")
root.configure(background='black')

filename='shell_output.txt'

def update_text():
# Configuring the text in Label widget.
my_label0.configure(text="Loading the game model and circuit. Click START to continue...")
subprocess.call(['game_model.png'], shell=True)
subprocess.call([f'{pngfile}.png'], shell=True)

def change_color():
my_label0.config(bg= "gray51", fg= "red")

f = open("shell_output.txt", "r")
file_string = f.read()

my_label0 = tk.Label(root, text =  __license__ + __author__+ __location__+','
                   +__ORCID__+ __copyright__+ __version__
                   +"\n \n|||||||||||||||||||||||||||||||||||||||||||||||||\
|||||||||||||--Reload Log_file--|||||||||||||||||||||||||||||||||||||||||\
||||||||||||||||||||||||||\n \n"+str(file_string), fg="#000fff000", background="black",
width=80, height =28, wraplength=525, anchor='n', justify="left")
my_label0.place(x=950, y=15)

# Create a list of image objects.
giflist = [] # This is to store and run image sprite in .gif.
for imagefile in imagelist:
photo = PhotoImage(file=imagefile)
giflist.append(photo)

anim_label1 = tk.Label(root, text = '. . . . . .',
                   fg="lightgreen" , background="black", width=80, height =2, wraplength=500,
                   anchor="w", justify="left")
anim_label1.place(x= 950, y= 500)
anim_label0 = tk.Label(root, text = dual1, fg="cyan" , background="black", width=80, height=2,
                   wraplength=500, anchor="w", justify="left")
anim_label0.place(x= 950, y= 500)

async def mainT():
await asyncio.sleep(0.001)  # Sprite is updated every second and then other asynchronous updating tasks
                            # take place in the outer loops.

# Loop through the gif image objects for a while.
for j in range(0, 30):
for gif in giflist:
     canvas.delete(ALL)
     canvas.create_image(width/2.0, height/2.0, image=gif)
     canvas.update()
     progressbar.step(j*5)
     anim_label0.place(x= 950+j*4, y= 500)
     #sleep(0.001)
for clr in color_list:
     for sprite in charslist:
          anim_label0.config(text=sprite+' . . . . . Loading . . .')
          anim_label0.update()
          #asyncio.run(mainT()) # Sprite is updated every second and then other asynchronous updating tasks
                               # take place in the outer loops.
          #asyncio.sleep(0.5)  # Sprite is updated every second and then other asynchronous updating tasks
                               # take place in the outer loops.
     anim_label0.config(fg=clr)
     anim_label0.update()
     sleep(0.02)

pygame.time.delay(2)

button0 = Button(root, text = 'START', bg='black', fg='white', command = flagClose)
button1 = Button(root, text = 'Input Tips', bg='black', fg='white', command = Help)
button3 = Button(root, text="QUIT", bg='black', fg='white', command = display_bye_msg)
button2 = Button(root, text="Show Game Model", bg='black', fg='white', command = open_new_win)
button4 = Button(root, text=f'Change Dataset: [##] vs Complement [##]', bg='black', fg='white',
                justify=LEFT, wraplength=160, command = restart_QFLCC)
buttonInvisible = Button(root, text=f'[##] vs its complement [##]', bg='black', fg='white', command = restart_QFLCC)
buttonInvisible.pack_forget() # This button remains invisible until made visible through .pack() command.
button0.place(x=950, y=450, width=50)
button1.place(x=1010, y=450, width=80)
button2.place(x=1100, y=450, width=140)
button3.place(x=1250, y=450, width=80)
button4.place(x=1340, y=450, width=180)

def update_btn4_text():
# This function focuses on Button4 changes and updates in including/excluding datasets per user request...
```

```
    global qdf, bitpair, qdf_bits_flag
    if  (qdf_bits_flag[0] == False) and (qdf_bits_flag[1] == True):
         qdf_bits_flagset() # Call this function to show the qdf bit values of the employed dataset.
         button4.configure(text=f'Change Dataset: {qdf} vs Complement {bitpair}')
         buttonInvisible.configure(text=f'{qdf} vs its complement {bitpair}')
         button4.update()
    else:
        button4.configure(text=f'Change Dataset: [##] vs Complement [##]')
        buttonInvisible.configure(text=f'[##] vs its complement [##]')

for k in range(0, 1):
for clr in color_list:
     for sprite in charslist:
          anim_label0.config(text='Configuring Logic Conditions for the chosen Circuit and Dataset...\n'+sprite,
                             wraplength=400)
          anim_label0.update()
          #asyncio.run(mainT())
          anim_label0.place(x= 950-j/2+100, y= 500)
          sleep(0.1)  # Sprite is updated every second and then other asynchronous updating tasks take place in the outer loops.
     anim_label0.config(fg=clr)
     anim_label0.update()

anim_label0.config(text=sprite+'\n QFLCC Game Circuit Configuration Complete. Click START to Continue...',
               wraplength=400)
anim_label0.update()

root.mainloop() # Run the Tkinter main loop.
```

- ▼ ≡ Common QDF game state variables:

Variables for defining QDF game states in QAI-LCode_QFLCC.py

```
#########################################################
# Variables for defining QDF game states and their
# animation functions.
#########################################################
game_points = []  # This is to keep win/lose score.
level_points= []  # This is to keep track of the level completion score.
levels = 0
points = 0
participantMain=""
participantMid=""
spd = [] # Game speed to be changed by user and register to alter the QDF game steps pace
         # of execution.
spd = 1.1 # QDF game speed of 1.1 is a the default value.
#########################################################
# Defining a simple animation function of game states.
#########################################################
```

▼ Expand/Collapse All. . .

* Click `Expand All` to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click `Collapse All` to selectively view a code block from QAI-LCode_QFLCC.py on this page.

Expand All | Collapse All

▼ ≡ QDF game level, P Classification and Animation:

Animation, P classification and game state functions for an assigned participant given by a role number. Function() list from top to bottom as ordered:

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def update_btn4_text():
# This function focuses on Button4 changes and updates in including/excluding datasets per user request...
    global qdf, bitpair, qdf_bits_flag
    if  (qdf_bits_flag[0] == False) and (qdf_bits_flag[1] == True):
         qdf_bits_flagset() # Call this function to show the qdf bit values of the employed dataset.
         button4.configure(text=f'Change Dataset: {qdf} vs Complement {bitpair}')
         buttonInvisible.configure(text=f'{qdf} vs its complement {bitpair}')
         button4.update()
    else:
        button4.configure(text=f'Change Dataset: [##] vs Complement [##]')
        buttonInvisible.configure(text=f'[##] vs its complement [##]')
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def participants():
     global participantMain, participantMid
     if takeUIn == 1:
         participantMain = bob
     elif takeUIn == 2:
          participantMain = alice
     if takeUIn2 == 3:
          participantMid = eve
     elif takeUIn2 ==4:
          participantMid = audience
     """Animation function for an assigned participant by a role number."""
```

▼ Source code in `QAI-LCode_QFLCC.py`

```
def winner_show():
     global points
     game_sound = pygame.mixer.Sound('__snd__/gameprize_state.wav')
     game_sound.play()
     points += 10 # Add points for each win of a QDF.
     game_points.append(points) # Add result to the scoresheet.
     print(f'Your score is: {points}. Scoresheet is: {game_points}')
     for count in range(5):
        print(fg.lightgreen + excited1, end="\r", flush=True), sleep(spd)  # Default QDF game speed is = 1.1.
        print(excited2, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
        if count == 5 or keyboard.is_pressed("n"):
             break
```

▼ Source code in `QAI-LCode_QFLCC.py`

```
def winner_next():
             entry_add()
             level_show()
             subprocess.run('echo "{}- ///--- Win Entry --- \
{} /// Level: {} Scored: {} at {}"'.format(entry_stage, excited2, levels, points, now.strftime("%Y-%m-%d
%H:%M:%S")),
shell=True, stdout=file_)  # Date last I/O file entry.
             print("\n",f'Next...'+fg.lightgreen+' Level:', levels, f'{here}'+fg.orange)
```

▼ Source code in `QAI-LCode_QFLCC.py`

```
def loser_show():
     global points
     game_sound = pygame.mixer.Sound('__snd__/gameloser_state.wav')
     game_sound.play()
     points -= 5 # Lose points for each loss of a QDF.
     game_points.append(points) # Add result to the scoresheet.
     print(f'Your score is: {points}. Scoresheet is: {game_points}')
     for count in range(4):
            print(fg.orange + grounded1, end="\r", flush=True), sleep(spd)
            print(grounded2, end="\r", flush=True), sleep(spd)
            print(fg.lightred+grounded3, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
            if count == 4 or keyboard.is_pressed("n" or "s"):
             break
```

▼ Source code in `QAI-LCode_QFLCC.py`

```
def loser_next():
             entry_add() # Reopen log file and register an event entry.
             level_show()
             subprocess.run('echo "{}- ///--- Lose Entry --- {} /// Level: {} \
Scored: {} at {}"'.format(entry_stage, grounded3, levels, points, now.strftime("%Y-%m-%d %H:%M:%S")),
shell=True, stdout=file_) # Date last I/O file entry.
             print("\n",f'Next...'+fg.lightgreen+' Level:', levels, f'{here}'+fg.orange)
```

▼ Source code in `QAI-LCode_QFLCC.py`

```
def dual_show():
     global points
     game_sound = pygame.mixer.Sound('__snd__/gamedual_state.wav')
     game_sound.play()
     points += 2 # Add points for each gain of a duality in QDF.
     game_points.append(points) # Add result to the scoresheet.
     print(f'Your score is: {points}. Scoresheet is: {game_points}')
     for count in range(5):
            print(fg.lightgreen + dual1, end="\r", flush=True), sleep(spd)
            print(fg.orange + dual2, end="\r", flush=True), sleep(spd)
            print(fg.red + dual3, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
            if count == 5 or keyboard.is_pressed("n" or "s"):
             break
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def dual_next():
            sleep(1)
            entry_add()
            level_show()
            subprocess.run('echo "{}- ///--- Dual Entry --- {} /// Level: {} \
Scored: {} at {}"'.format(entry_stage, dual3, levels, points, now.strftime("%Y-%m-%d %H:%M:%S")),
shell=True, stdout=file_) # Date last I/O file entry.
            print("\n",f'Next...'+fg.lightgreen+' Level:', levels, f'{here}'+fg.orange)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def guesser_show():
     global points
     game_sound = pygame.mixer.Sound('__snd__/gamedual_state.wav')
     game_sound.play()
     points += 3 # Add points for each guess of a QDF, close but not concluded.
     game_points.append(points) # Add result to the scoresheet.
     print(f'Your score is: {points}. Scoresheet is: {game_points}')
     for count in range(5):
            print(fg.red + guesser1, end="\r", flush=True), sleep(spd)
            print(fg.orange + guesser2, end="\r", flush=True), sleep(spd)
            print(fg.green + guesser3, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
            if count ==5 or keyboard.is_pressed("n") or keyboard.is_pressed("s"):
              break
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def guesser_next():
            sleep(1)
            entry_add()
            level_show()
            subprocess.run('echo "{}- ///--- Guesser Entry --- {} /// Level: {} \
Scored: {} at {}"'.format(entry_stage, guesser3, levels, points, now.strftime("%Y-%m-%d %H:%M:%S")),
shell=True, stdout=file_) # Date last I/O file entry.
            print("\n",f'Next...'+fg.lightgreen+' Level:', levels, f'{here}'+fg.orange)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def helper_show():
     global points
     game_sound = pygame.mixer.Sound('__snd__/gamehelper_state.wav')
     game_sound.play()
     points += 1 # Add points for each guess of a QDF, close but not concluded.
     game_points.append(points) # Add result to the scoresheet.
     print(f'Your score is: {points}. Scoresheet is: {game_points}')
     for count in range(2):
            print(fg.pink + helperTarget1, end="\r", flush=True), sleep(spd)
            print(fg.orange + helperTarget2, end="\r", flush=True), sleep(spd)
            print(fg.green + aimTarget, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
            if keyboard.is_pressed("n") or keyboard.is_pressed("s"):
              break
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def helper_next():
            sleep(1)
            entry_add()
            level_show()
            subprocess.run('echo "{}- ///--- Helper Win Entry --- {} /// Level: {} \
Scored: {} at {}"'.format(entry_stage, helperTarget1, levels, points, now.strftime("%Y-%m-%d %H:%M:%S")),
shell=True, stdout=file_) # Date last I/O file entry.
            print("\n",f'Next...'+fg.lightgreen+' Level:', levels, f'{here}'+fg.orange)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def level_show():
#-------------------------------------------
# Register level entry or update given the
# accumulated and/or lost energy score points.
#-------------------------------------------
     global points, levels
     game_sound = pygame.mixer.Sound('__snd__/gamelevel_complete.wav')
     game_sound.play()
     level_points.append(levels)
     if points < 10 and points > 0:
          levels=0
          print(f'\nLevel {levels} initiated! Your score is: {points}.')
          print(fg.lightgreen + guesser3, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
          level_next()
     elif points >=10 and points < 20:
          levels=1
     elif points >=20 and points < 30:
          levels=2
     elif points >=30 and points < 40:
          levels=3
     elif points >=40 and points < 50:
          levels=4
     elif points >=50 and points < 60:
          levels=5
     elif points >=60 and points < 100:
          levels=6
     elif points >=100 and points < 1000:
          levels=7
     elif points >=1000:
          levels=8
          print(bg.green+fg.yellow+ f'\nLevel {levels} complete! You won all game rounds! \
Your score is: {points}. Scoresheet is: {game_points}')
          print(fg.lightgreen + guesser3, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
          game_sound = pygame.mixer.Sound('__snd__/game_music2.wav')
          game_sound.play()
          level_next()
     print(Fore.LIGHTGREEN_EX + f'\nLevel {levels} engaged!')
     if levels >= 1 and levels < 8:
          level_dance()
     elif levels==8:
          game_win()
     elif points <= -10:
          game_over()  # Log event, either restart or end program by calling this function.
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def level_dance():
     for count in range(2):
            print(fg.orange + guesser2, end="\r", flush=True), sleep(spd)
            print(fg.lightgreen + guesser3, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
            if keyboard.is_pressed("n"):
                  break
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def game_win():  # Conditions to play exit sound and animation, then exit game.
     for count in range(2):
            print(fg.lightgreen + classical1, end="\r", flush=True), sleep(spd)
            print(fg.orange + classical2, end="\r", flush=True), sleep(spd)
            print(fg.lightgreen + excited2, end="\r"+Fore.LIGHTYELLOW_EX, flush=True), sleep(spd)
            if participantMain == alice:
                 print(Back.YELLOW+ fg.lightgreen + crownAlice, end="\r"+Fore.LIGHTYELLOW_EX,
                       flush=True), sleep(spd)
            elif participantMain == bob:
                 print(Back.YELLOW +fg.lightgreen + crownBob, end="\r"+Fore.LIGHTYELLOW_EX,
                       flush=True), sleep(spd)
     # Play Windows exit sound.
     winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
     sys.exit() # End of game as a winner, then HALT.
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def errorP(): # P Error state displayed with sound effect  after wrong user's input.
      game_sound = pygame.mixer.Sound('__snd__/errorP.wav')
      game_sound.play()
      game_sound = pygame.mixer.Sound('__snd__/gameno_state.wav')
      game_sound.play()
      for count in range(0, 2):
         print(bg.green+fg.yellow+ f'{errorP1}', end="\r", flush=True), sleep(spd)
         print(bg.green+fg.yellow+ f'{errorP2}', end="\r", flush=True), sleep(spd)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def repeatP(): # P repeated displayed with sound effect after reentering a close or identical P by user.
      game_sound = pygame.mixer.Sound('__snd__/errorP.wav')
      game_sound.play()
      game_sound = pygame.mixer.Sound('__snd__/gameno_state.wav')
      game_sound.play()
      for count in range(0, 2):
         print(bg.green+fg.yellow+ f'{errorP1}', end="\r", flush=True), sleep(spd)
         print(bg.green+fg.yellow+ f'{errorP2}', end="\r", flush=True), sleep(spd)
         print(bg.green+fg.yellow+ f'{aimTarget}', end="\r", flush=True), sleep(spd)
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def game_over():
        game_sound = pygame.mixer.Sound('__snd__/game_countdown.wav')
        print(bg.cyan + fg.red+'QFLCC Game Over')
        game_sound.play()

        for j in range(9,-1,-1):
             print(bg.blue+fg.pink+'Enter \'r\' to restart this game, \'b\' to restart QFLCC, or else to end
program...',
                   Back.YELLOW + fg.red+f"{j}"+Back.RESET+fg.blue, end="\r")
             sleep(1)
             if j == 6:
                  game_sound.play() # Partitioned sound replayed for the entire countdown sequence.
             if j == 0:
                 uCommand = str(input(fg.lightcyan+"\n\r> "))
        if uCommand =='b' or keyboard.is_pressed("b"):
            subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program.
            file_.close() # Close the log file recording events from the user's I/O terminal.
        elif keyboard.is_pressed("r") or uCommand =='r': # keyboard.wait("r"):
            level_start()
        else:
            print(bg.cyan+fg.red+'End of Program...'+ Back.RESET + Fore.RESET)
            subprocess.run('echo "///--- QFLCC Shell Log_file HALT --- /// on {}"'.format(entry_stage,
                           now.strftime("%Y-%m-%d %H:%M:%S")), shell=True, stdout=file_) # Date last I/O file
entry.
            sys.exit() # Game Over and HALT.
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
2068
2069
2070
2071
2072
```

```
def level_next():
            entry_add()
            subprocess.run('echo "///--- Level Entry --- {}{} ///, Level: {} Scored: \
{} at {}"'.format(entry_stage, guesser3, levels, points, now.strftime("%Y-%m-%d %H:%M:%S")),
shell=True, stdout=file_) # Date last I/O file entry.
```

▼ Expand/Collapse All…

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.

   Expand All    Collapse All
```

▼ ≗ QAI/QFLCC P and correlation distance measure code:

```
2074  ##############################################################################################
2075  # QAI functions to compare correlated values based on User Input and weight them through
2076  # the elimination process of repeated successful hits of P values.
2077  ##############################################################################################
2078  PInMem = 0   # P value taken from User to store and update later.
2079  rep_lst0 = []
2080  rep_lst1 = []
2081  rep_lst2 = []
2082  w = 0.0 # ...for weighted guess by the user.
2083  DeltaC = 0.0
2084  DeltaP = 0.0
2085  rndDeltaC = 0.0
2086  dp = 0.0 # ...for a duplicate identified as a switch.
2087
2088  def reset_PPar():
2089       global DeltaC, PInMem, w, DeltaP
2090       DeltaC = 0.0
2091       DeltaP = 0.0
2092       PInMem = 0.0
2093       w=0.0
2094
2095  def uIn_Repeat():
2096  ##############################################
2097  # Function for P and weighted P calculations.
2098  ##############################################
2099       global PInMem, w, dp, DeltaP
2100       dp=0
2101
2102       #for PInMem in rep_lst0:
2103       rep_lst0.append(round(PInMem, 2))  # Append the registered input with two decimal points accuracy.
2104       print("List of tried P\'s: "+ Fore.LIGHTMAGENTA_EX, rep_lst0, Fore.LIGHTGREEN_EX)
2105
2106       rep_lst1.append(round(w, 3))
2107       print("List of assigned weights: "+ Fore.LIGHTMAGENTA_EX, rep_lst1, Fore.LIGHTGREEN_EX)
2108
2109       rep_lst2.append(round(DeltaP, 2))
2110       print("List of matched ΔP\'s: "+ Fore.LIGHTMAGENTA_EX, rep_lst2, Fore.LIGHTGREEN_EX)
2111
2112       # Create a set from the list.
2113       sub_lst0 = set(rep_lst0)
2114       print('Subset of tried P\'s:', sub_lst0)
2115
2116       # Compare the length and print if the list contains duplicates.
2117       dup0 = {x for x in rep_lst0 if rep_lst0.count(x) > 1}
2118       print('Duplicated P tries:'+fg.red, dup0, ''+fg.lightgreen)
2119
2120       # Create a set from the list.
2121       sub_lst2 = set(rep_lst2)
2122       print('Subset of calculated ΔP\'s:',sub_lst2)
2123
2124       dup2 = {y for y in rep_lst2 if rep_lst2.count(y) > 1}
2125       print('Duplicates of matched P\'s:'+fg.lightcyan, dup2, ''+fg.lightgreen)
2126
2127       if (round(PInMem, 2) in dup0): # or (round(DeltaP, 2) in dup2):
2128            print(f'Duplicates {dup0} found in your tried P\'s list! Try again...')
2129            dp=1
2130            repeatP()
2131            print('\n')
2132
2133       while dp==0:
2134            if round(PInMem, 2) in rep_lst0 and w>=9 and dp==0:
2135                 print(f'P value {PInMem} guessed a weak classical P outcome and correlated with \
2136  max(ΔP)= min(P) of qubit dataset. Your successful hit weight was: {w}')
2137                 break
2138            elif round(PInMem, 2) in rep_lst0 and (w<9 and w> 2) and dp==0:
2139                 print(f'P value {PInMem} guessed a moderate classical P outcome and correlated with \
2140  some <ΔP>=<P> of qubit dataset. Your successful hit weight was: {w}')
2141                 break
2142            elif round(PInMem, 2) in rep_lst0 and (w< 1.9 and w > 1.1) and dp==0:
2143                 print(f'P value {PInMem} guessed a classical P outcome and correlated with \
2144  min(ΔP)= max(P) of qubit dataset. Your successful hit weight was: {w}')
2145                 break
2146            elif round(PInMem, 2) in rep_lst0 and (w==0) and dp==0:
2147                 w=float('inf') # Define a positive infinite integer assigned to the weight variable w.
2148                 print(f'P value {PInMem} guessed an undefined P outcome of the qubit dataset. Your \
2149  successful hit weight was: [0, {w}]')
2150                 break
2151            elif round(PInMem, 2) in rep_lst0 and (w<=1.1 and w>0) and dp==0:
2152                 print(f'P value {PInMem} guessed a strong classical P outcome and correlated with \
2153  max(ΔP)=min(P) of qubit dataset. Your successful hit weight was: (0, {w}]')
2154                 break
2155            elif round(PInMem, 2) in rep_lst0 and (w<=2.2 and w>=1.9) and dp==0:
2156                 print(f'P value {PInMem} guessed a superposition P outcome and correlated with a \
2157  ΔP=1/2 uncertainty of the qubit dataset. Your successful hit weight was: {w}')
2158                 break
2159            elif dp==1:
2160                 reset_PPar()
2161                 continue
2162            else:
2163                 print(f'P value not guessed and uncorrelated. Your successful hit weight was: {w}')
```

```
                reset_PPar()
                break
```

▼ Expand/Collapse All…

```
* Click   Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click   Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.
```

Expand All | Collapse All

▼ ≡ QDF game prompt and help options:

Calling help, prompt and sound volume setting functions during the game. Function() list from top to bottom as ordered:

- ▼ ≡ QDF game sound volume settings::

QDF game sound volume settings for basic and expert modes: source code in QAI-LCode_QFLCC.py or QDF-game-gui.py (under construction)

```
#### Volume Settings ####
import pyautogui as pvol

def set_game_vol(new_volume):
##############################################################
# The volume function sets volume according to your OS.
# This option is available as 'v' or 'volume' during the game.
##############################################################
    pvol.press('volumedown', presses = 50) # Sets volume to zero.
    time.sleep(0.5) # Using time.sleep to space the presses.
    x = math.floor(new_volume / 2) # Setting the amount of presses required.
    pvol.press('volumeup', presses = x) # Setting the volume.
    winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS) # Sound test.
    """- End of basic mode volume settings."""

#################################################################
# Packages and variables for sound volume change from within CLI
# during the game (expert mode with decimal range approximation).
#################################################################
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
###- End of expert mode volume settings -###
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def set_game_vol(new_volume):
##############################################################
# The volume function sets volume according to your OS.
# This option is available as 'v' or 'volume' during the game.
##############################################################
    pvol.press('volumedown', presses = 50) # Sets volume to zero.
    time.sleep(0.5) # Using time.sleep to space the presses.
    x = math.floor(new_volume / 2) # Setting the amount of presses required.
    pvol.press('volumeup', presses = x) # Setting the volume.
    winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS) # Sound test.
    """- End of basic mode volume settings."""
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def stayIn():
#############################################################################
# To stay in prompting environment and respond to a repeated question.
#############################################################################
 while True:
     promptIn = str(input(bg.green + fg.yellow +'Do you want to continue in prompt mode to input command? [y/n]'+
Back.RESET))
     if (promptIn == 'y' or promptIn == 'yes') and pFlag == True:
           print(bg.green + fg.yellow +
                 "Input the relevant command according to \'h\' or \'help\', \'cv\', \'v\' or \'volume\' for sound
volume change..."
                 + Back.RESET)
           promptGame() # Restart the prompt function.
           break
     elif (promptIn == 'y' or promptIn == 'yes') and pFlag == False:
           print(bg.green + fg.yellow +
                 "Input command according to \'help\'. To view the list of commands input \'h\' or \'help\':"
                 + Back.RESET)
           promptGame() # Restart the prompt function.
           break
     elif (promptIn == 'n' or promptIn == 'no') and (pFlag == True or pFlag == False):
           break  # Skip this step and continue the game based on n or any other relevant command.
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def pass_code():
#############################################################################
# Enters Safe Mode via passcode to cheat and change dataset feed to the game.
#############################################################################
 global dir_flag
 # Data here lists hardcoded data. This array can be customized and elements replaced
 # via index value replacement by a password txt file once loaded.
 y = random.randint(0, 9999) # Can crack and override this when value is stored and accessed realtime
                             # not after view.
 y = str(y)
 pass_file = "pass-code.txt"

 x = random.randint(0, 1)
 x= str(x)

 data = {
  "passcode": ['$ Alice & Bob cheat $', 'qdf cheat sheet 2121', 'dataset 0110 cheat',
               'qdf dataset 00,01,10,11', x, len(y[:-1])*'#'+y[-1:], ''],
  f"{Back.LIGHTGREEN_EX}qualified{Back.RESET}": [f"{Fore.LIGHTGREEN_EX}{True}{Fore.LIGHTGREEN_EX}",
               f"{Fore.LIGHTGREEN_EX}{True}{Fore.LIGHTGREEN_EX}",
               f"{Fore.LIGHTGREEN_EX}{True}{Fore.LIGHTGREEN_EX}",
               f"{Fore.LIGHTGREEN_EX}{True}{Fore.LIGHTGREEN_EX}",
               f"{Fore.LIGHTRED_EX}binary to pass [{False}]{Fore.LIGHTGREEN_EX}",
               f"{Fore.LIGHTYELLOW_EX}one-time pass [{True}]{Fore.LIGHTGREEN_EX}",
               f"{Fore.LIGHTRED_EX}else to pass [{False}]{Fore.LIGHTCYAN_EX}"]
 }
 df = pd.DataFrame(data, index=['pass_1', 'pass_2', 'pass_3', 'pass_4', 'pass_5', 'pass_6', 'pass_7'])
 index_ = ['pass_1', 'pass_2', 'pass_3', 'pass_4', 'pass_5', 'pass_6', 'pass_7'] # Create the index.
 df.index = index_ # Set the index.

 z=['','','','','','','']
 for i in range (0,6):
      z[i] = df.loc[f'pass_{i+1}','passcode']
      #print(z[i])  # Any other z[i], for a combo code restrictions and switches can updated to the growing passcode
list below.

 user_input = input(f"{fg.lightgreen}>>{fg.lightcyan} This command is supported only in the QFLCA's Safe Mode
(SM:>>)\
 environment! \n-- Enter [Passcode + Enter] to proceed. If 'qualified' enter 'n' when prompted.{Fore.LIGHTRED_EX}* \
\n*- Some passcodes{fg.lightcyan} can unlock lower and/or higher level codes to access different game levels and
datasets. \
\n-- Here is your chance to use a one-time passcode, if you know where it is locally?! Press Enter... ")

 with open(pass_file, 'w+') as passf_to_write:
               # Write information about the original image sequence and final image frame.
               passf_to_write.write(y)
 passf_to_write.close()
 game_sound = pygame.mixer.Sound('__snd__/game_countdown.wav')
 game_sound.play()
 hline =
"=============================================================================================================="
 for cnt_down in range(9, -1, -1): # 10 seconds left to see this code from its file location.
    print(f"{fg.lightgreen}>>{fg.lightcyan} One-time passcode is unmasked in the\
 background for 10 seconds...{Back.YELLOW + fg.red}{cnt_down}{Back.RESET+ fg.lightcyan}", end = '\r')
    sleep(1)
    if cnt_down == 6:
         game_sound.play() # Partitioned sound replayed for the entire countdown sequence.
    elif cnt_down == 0:
         #print(end=' ')
         print(f"{hline}")

 with open(pass_file, 'w+') as passf_to_write:
     passf_to_write.write(len(y[:-1])*"#"+y[-1:])
     passf_to_write.close()
 with open(pass_file) as passf_to_read:
      print(f"{fg.lightgreen}>>{fg.lightcyan} One-time passcode is now masked... {Fore.LIGHTRED_EX}"
            + passf_to_read.read() + Fore.LIGHTCYAN_EX), sleep(2)
 passf_to_read.close()

 user_input = input(f"{fg.lightgreen}>>{fg.lightcyan} Enter your passcode, or press Enter to skip back to \
regular prompt ({fg.lightgreen}>{fg.lightcyan}): {Fore.LIGHTRED_EX}")

 newdf = df.mask(df["passcode"] >= user_input[2:]) # Mask code until revealed with the
                                                   # right one(s) entered by user
                                                   # (the first 2 chars of input ignored).
 if (user_input == z[0] or user_input == z[1] or user_input == z[2] or user_input == z[3] or user_input == z[4]
     or user_input == z[5]):
        dir_flag = 1  # Directory flag is set to 1 for directory access in safe mode.
        qdf_bits_flag[1] = True # Set this flag to 1 or reveal the qdf bit values from any selected dataset.
        qdf_bits_flagset() # Call this function to show the qdf bit values of the employed dataset.
```

```
2295        update_btn4_text() # Make sure the form button4 has qdf bit values revealed by calling this function.
2296        print(f"{bg.cyan + user_input} is qualified as [True]! \nThe following list shows if there are any more \
2297 passcodes you can use later: {Fore.LIGHTGREEN_EX +  Back.RESET}")
2298        print(newdf)
2299        prompt()
2300        safeMode_dir()
2301        PAnalysis_model() # Run QDF circuit simulation in case of successfully accessing the cheat sheet
2302                          # for a new dataset feed. Dir flag will be reset back to 0 in the process.
2303  if (user_input == "n" or user_input == z[6] or user_input == ' ' or user_input == "next"):
2304        dir_flag = 0
2305        prompt()
2306  if user_input == y:  # Reveal all passcodes if the random number is caught/guessed and entered by the user.
2307        dir_flag = 1
2308        qdf_bits_flag[1] = True # Set this flag to 1 or reveal the qdf bit values from any selected dataset.
2309        qdf_bits_flagset() # Call this function to show the qdf bit values of the employed dataset.
2310        update_btn4_text() # Make sure the form button4 has qdf bit values revealed by calling this function.
2311        print(f"{bg.cyan + user_input} You unlocked all the passcodes for future use! This is your chance to \
2312 write them down: {Fore.LIGHTGREEN_EX +  Back.RESET} \n", df)
        prompt()
        safeMode_dir()
        PAnalysis_model()
        """- End of passcode conditions to change the game's dataset feed."""
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
2314  def promptGame():
2315  ##########################
2316  # QDF Game Help function.
2317  ##########################
2318      global spd, pFlag, site_name   # A global variable for game speed as the QDF game steps pace up or slow down.
2319                 # pflag is a boolean global variable to raise this flag for the prompt state for a specific prompt.
2320      while True:
2321          print(fg.lightgreen + "\r< ", end=""), sleep(spd)
2322          print("\r> ", end=""+fg.orange), sleep(spd)
2323          try:
2324               promptIn = str(input()) # Respond to the user's choice or command.
2325               if promptIn == 'n' or promptIn == 'next' or promptIn == 'no':
2326                  break
2327               elif promptIn == 's' or promptIn == 'speed':
2328                    takeUIn_speed = float(input(bg.orange+'Change game speed −=≡Σ((( つ◕‿‿◕)つ ... Choose from this
2329  range:'
2330                                                + Back.RESET+ fg.yellow  +' [fastest = 0.1, slowest = 2]: '))
2331                    spd = takeUIn_speed
2332                    if spd >2.0 or spd < 0.1:
2333                         print(fg.red +"Wrong value entered! Type in the same \'s\' command or other for an input...")
2334                         promptGame()
2335                    else:
2336                        spd = takeUIn_speed
2337                    break
2338               elif promptIn == 'v' or promptIn == 'volume':
2339                    vol = float(input(bg.orange+'Input sound volume −=≡Σ(((° ͜ʖ °)  🕪  between'+ Back.RESET+ fg.yellow
2340                                    + '[    = 0 for muted, and     = 100 for loudest]: '))
2341                    if vol >= 0 and vol <= 100:
2342                       set_game_vol(vol)
2343                       pFlag = True # Raise this flag for message containing a volume prompt message.
2344                       stayIn()  # To remain to make further volume change or input other command.
2345                       break
2346                    elif vol < 0:
2347                         set_game_vol(vol)
2348                         print(fg.red +"Out of range or wrong value entered! Readjusted volume to muted or 0!")
2349                         promptGame()  # Restart prompt.
2350                         break
2351                    elif vol > 100:
2352                         set_game_vol(vol)
2353                         print(fg.red +"Out of range or wrong value entered! Readjusted volume to maximum or 100!")
2354                         promptGame() # Restart prompt.
2355                         break
2356               elif promptIn == 'cv' or promptIn == 'cliv':
2357                    pFlag = True # Raise this flag for message containing a volume prompt message.
2358                    takeUIn_volume = float(input(bg.orange+'Change game volume −=≡Σ(((° ͜ʖ °)  🕪 ... Choose from this
2359  range:'
2360                                                + Back.RESET+ fg.yellow +
2361                                                ' [{quietist     ⇄     = [0 to 6], quiet     = 6.1}, loudest     =
2362  100]: ')) # Input value
2363                                                                                                  # should be
2364  between >= 0 and 100.
2365                    cvol = takeUIn_volume
2366                    if cvol > 100 or cvol < 0:
2367                         print(fg.red +"Out of range or wrong value entered! Type in the same \'cv\' command or other
2368  for an input...")
2369                         promptGame() # Restart prompt.
2370                         break
2371                    elif cvol >= 20 and cvol <= 100:
2372                        cvol = takeUIn_volume
2373                        min_vol = ((11*cvol - 1100)/40)  # My lower conversion formula for volume range of [-29, 0]
2374                                                         # is e.g.: 100-(100*(-5-0)/((0-55)/2)) as simplified.
2375                        print ("Volume converted and adjusted from the upper desktop's endpoint volume range of [-29,
2376  0] =" , "%.4f" % min_vol)
2377                        #------------------------------------------
2378                        # Control volume proximation:
2379                        #------------------------------------------
2380                        #volume.SetMasterVolumeLevel(0, None) #max
2381                        #volume.SetMasterVolumeLevel(-5.0, None) #72%
2382                        volume.SetMasterVolumeLevel(min_vol, None) #alternative 72% from 80 as user input.
2383                        #volume.SetMasterVolumeLevel(-10.0, None) #51%
2384                        #volume.SetMasterVolumeLevel(-20.0, None) #26% alternative is 27 from user input (very close).
2385                        #volume.SetMasterVolumeLevel(-30.0, None) #13%
2386                        #volume.SetMasterVolumeLevel(-55.0, None) #1%
2387                        #------------------------------------------
2388                        winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS)
2389                        stayIn() # To remain to make further volume change.
2390                        break
2391                    elif cvol > 6  and cvol < 20:
2392                        cvol = takeUIn_volume
2393                        min_vol = -4000/(11*cvol)  # My lower conversion formula for volume range of [-40, -30] is:
```

```
                                                # 100*(100/(-55-0)/((0-55)/2))=6.61 as simplified.
                        print ("Volume converted and adjusted from the lower desktop's endpoint volume range of [-41,
-30] ="
                               , "%.4f" % min_vol)
                        volume.SetMasterVolumeLevel(min_vol, None)
                        winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS)
                        stayIn() # To remain to make further volume change.
                        break
                    elif cvol >= 0 and cvol <= 6: # This condition ranges the module's close to muted state or 0%
volume.
                        cvol = 6  # The minimum limit has reached for the lowest bound conversion for the customized
volume.
                        min_vol = -4000/(11*cvol)  # My lower conversion formula for volume range of
                                                   # [-60.606060..., -40) as simplified.
                        print ("Volume converted and adjusted from the upper desktop's endpoint volume range of
[-60.6, -40) ="
                               , "%.4f" % min_vol)
                        volume.SetMasterVolumeLevel(min_vol, None)
                        winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS)
                        stayIn() # To remain to make further volume change.
                        break
                elif promptIn == 'b' or promptIn == 'begin':
                     subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program to analyze and validate a new
dataset.
                     break
                elif promptIn == 'f' or promptIn == 'form':
                     print('Revisit dataset(s) form load & play [Temporal State]...   (° ʖ °)  🗐 ⟵ 🖭 : '
                           + bg.red + fg.yellow +'A F F I R M A T I V E !')
                     game_sound = pygame.mixer.Sound('__snd__/gameaffirmative_state.wav')
                     game_sound.play()
                     root.deiconify() # Unhide this form and show its options to resume game, reload or choose
dataset(s).
                     pFlag = False # Lower this flag for a message containing a regular prompt message
                     stayIn() # To remain in prompt mode to input command.
                     break
                elif promptIn == 'r' or promptIn == 'restart':
                     print('Restarting program...  (° ʖ °)  ')
                     open_new_win() # Restart program.
                     break
                elif promptIn == 'website' or promptIn == 'site' or promptIn == 'about' or promptIn == 'web':
                     site_name = "game-site" # This html file is stored in the same site folder to read.
                     site_doc() # Load website.
                elif (promptIn == 'dir' or promptIn == 'sm dir') and (dir_flag == 0):
                     pass_code()  # Call this function to enter passcode to cheat and feed a new dataset to the game.
                                  # Dir flag will be set to 1 in the process.
                     continue
                elif promptIn == 'h' or promptIn == 'help':
                     userGame_Help()  # Show help.
                     continue
                elif promptIn == 'e' or promptIn == 'exit':
                     # Play Windows exit sound.
                     game_sound = pygame.mixer.Sound('__snd__/goodbye1.wav')
                     game_sound.play()
                     print('Exiting game...   (° ʖ °)    Goodbye!'), sleep(3)
                     main_exit()  # Terminate game program.
                else:
                   print('Input command or response. For help, enter \'h\'... ')
                   continue
            except ValueError:
                 print("Invalid input. Please enter a response.")
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def userGame_Help():
#########################
# QDF Game Help function.
#########################
     while True:
          print(f'\033[0;31m\033[1m\x1B[1;4mGame Input Tips:\x1B[0m\033[0m \033[0;36m\n-Enter \'n\' key for the next
message, result, \
output or next input. \n-Enter \'h\' to display these tips. \n-Enter a role number when asked for a role as Alice
{alice} or Bob \
{bob} via Eve {eve} or the Audience {audience}. \n-Enter a value between 0 and 1 inclusive when asked for a \
P value. \n-Enter \'r\' to restart game. \n-Enter \'b\' to analyze and validate a new dataset. \n-Enter \'s\' \
or \'speed\' to change the speed of game steps. \n-Enter \'v\' or \'volume\' for sound volume change during the
game. \
\n-Enter \'site\' or \'web\' to view \'about\' the game \'website\'. \
\n-Enter \'cv\' for sound volume change within CLI. \n-Enter \'f\' or \'form\' to reload the QDF \
game form (resume game by clicking its \'START\' button). \
\n-Enter \'dir\' or \'sm dir\' for a new dataset analysis & simulation to feed this game.*\
\n*-This command requires a passcode to see dataset results in a Safe Mode (SM:>>) environment!\
     \n-Enter \'e\' to exit the program.')
          break
```

▼ ≙ QDF game conditions for a QFLCC…

QDF game QFLCC conditions: source code in QAI-LCode_QFLCC.py or QDF-game-gui.py (under construction)

```
##########################################
# QFLCC Game Conditions...
##########################################
p = 1 # classical state of the probability
k = 10 # math.inf #float('inf')

# Set an initial condition.
game_active = True
# Set up the while loop.

def level_start():
     entry_add()  # Date last I/O file entry.

level_start()
level_show() # Register level no. Levels engage and get completed given the lost
             # vs. accumulated points.
```

▼ Expand/Collapse All…

```
* Click    Expand All  to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click    Collapse All  to selectively view a code block from QAI-LCode_QFLCC.py on this page.
```

Expand All Collapse All

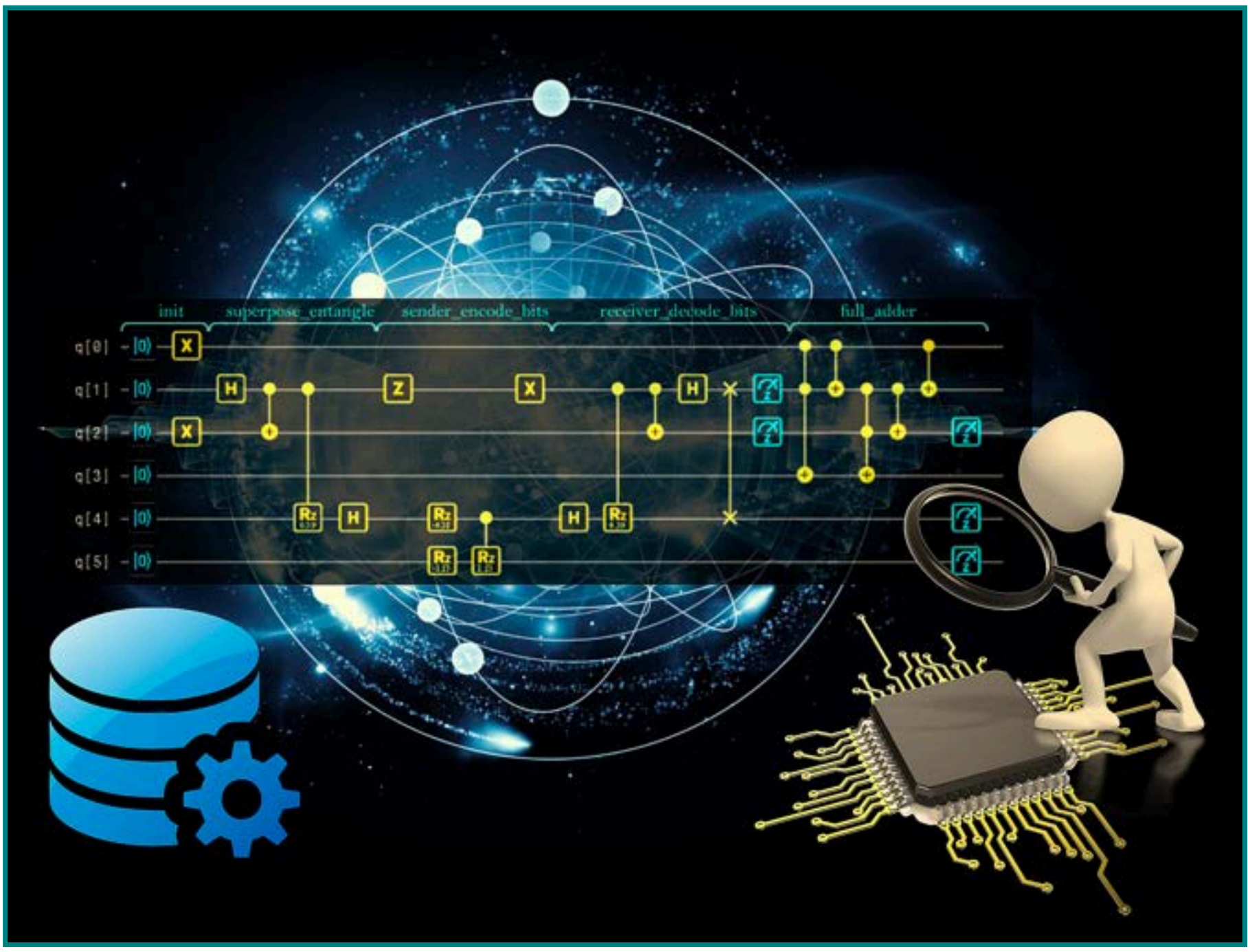
▼ ≙ Main QDF game code to display P results for game participants and QFLCC…

```
2476    ################################################
2477    # The main game code block for taking in P's from
2478    # the user and dataset(s), and determine game's
2479    # participant role, wins, losses and levels.
2480    ################################################
2481    while game_active:
2482        dp=0
2483        winsound.PlaySound("SystemExclamation", winsound.SND_ALIAS)
2484        try:
2485         takeUIn = int(input(fg.orange+f'In this game, are you Bob {bob} the guest, or Alice {alice} \
2486    the host to win a/the prize (targeted energy state) {prize}?\nChoose 1 for \
2487    Bob, 2 for Alice: '))
2488         sys.stdout.flush() # Clear line.
2489        except ValueError:
2490             game_sound = pygame.mixer.Sound('__snd__/gameno_state.wav')
2491             game_sound.play()
2492             print(fg.lightred + f"    NO! Wrong Input! Input must be 1 or 2. {errorP1} Try again..."+fg.lightgreen)
2493             continue
2494        try:
2495         takeUIn2 = int(input(f'For your game participant {takeUIn}, will Eve {eve} join by quantum means \
2496    to secretly share information about the prize {prize}?\nChoose 3 to have Eve \
2497    spying, 4 for Audience {audience} to cheer/suggest and raise/lower \
2498    participant {takeUIn}\'s energy state: '))
2499        except ValueError:
2500             game_sound = pygame.mixer.Sound('__snd__/gameno_state.wav')
2501             game_sound.play()
2502             print(fg.lightred + f"    NO! Wrong Input! Input must be 3 or 4. {errorP2} Try again..."+fg.lightgreen)
2503             continue
2504        try:
2505         takePIn = float(input(f'Enter a P value for participant #{takeUIn}, \
2506    {participantMain}: '))  # The var input will be a P value for the current game participant.
2507        except:
2508             game_sound = pygame.mixer.Sound('__snd__/gameno_state.wav')
2509             game_sound.play()
2510             print(fg.lightred + f"    NO! Wrong Input! Input a floating-point value between 0 and 1. {errorP1} Try again..."
2511                   +fg.lightgreen)
2512             continue
2513
2514        promptGame()
2515        participants()
2516        PInMem = takePIn
2517
2518        uIn_Repeat() # Register P value and compare for a weight assigned to the closest correlation vs farthest
2519                     # for a strong prediction between two datasets. Not just comparingP values within this dataset
2520                     # for validation purposes, according to ref. [1].
2521        if ((takePIn < abs(p) and takePIn >0.5) or (takePIn < abs(p)/2 and
2522                                                takePIn > 0)) and dp==0: # and takeO == 'p game':
2523             print(bg.green+fg.yellow+'Hello, Quantum World!'+Back.RESET)
2524             ################################################
2525             # Now test correlation with prediction results
2526             ################################################
2527             if (takePIn < float(df3Mem) and takePIn>0.5):
2528                  DeltaP = float(df3Mem)-takePIn
2529                  if DeltaP < 0.5:
2530                       rndDeltaC = round(1-DeltaP, 2)
2531                       DeltaC = 1-DeltaP
2532                       w= round((1/(DeltaC)),3)  # Weighted value of the correlation distance.
2533                                                 # The closer to infinity the greater the weight and
2534                                                 # inaccuracy of player's guess/hit on P value.
2535                       print(f'Correlation to strong prediction result for participant #{takeUIn}, \
2536    {participantMain}, is high to win:'+fg.yellow, DeltaC, Fore.RESET)
2537                       if (takeUIn == 1 and takeUIn2 == 3 and (DeltaC >0.5 or DeltaC ==1)):
2538                            print(Back.LIGHTYELLOW_EX + 'Alice loses to Bob. Bob wins the prize with Eve\'s help!'
2539                                  +Back.RESET+fg.green+' YOU WIN!')
2540                            print(fg.green+'These doubles won the prize:'+fg.pink, qdf, fg.green+ ', QDF P match is:'
2541                                  +fg.pink, rndDeltaC)
2542                            winner_show()
2543                            winner_next()
2544                       elif (takeUIn == 1 and takeUIn2 == 4 and (DeltaC > 0.5 or DeltaC ==1)):
2545                            print(Back.LIGHTYELLOW_EX +
2546                                  'Alice loses to Bob. Bob wins the prize as the Audience cheer without Eve\'s help!'
2547                                  + Back.RESET+fg.green+' YOU WIN!'+Fore.RESET)
2548                            print(fg.green+'These doubles won the prize:'+fg.pink, qdf, fg.green+ ', QDF P match is:'
2549                                  +fg.pink, rndDeltaC)
2550                            winner_show()
2551                            winner_next()
2552                       elif (takeUIn == 2 and takeUIn2 == 3 and (DeltaC >0.5 or DeltaC ==1)):
2553                            print(Back.LIGHTYELLOW_EX + 'Bob loses to Alice. Alice wins to keep the prize with Eve\'s help!'
2554                                  + Back.RESET + fg.green +' YOU WIN!'+ Fore.RESET)
2555                            print(fg.green+'These doubles won the prize:'+fg.pink, qdf, fg.green+ ', QDF P match is:'
2556                                  +fg.pink, rndDeltaC)
2557                            winner_show()
2558                            winner_next()
2559                       elif (takeUIn == 2 and takeUIn2 == 4 and (DeltaC >0.5 or DeltaC ==1)):
2560                            print(Back.LIGHTYELLOW_EX + 'Bob loses to Alice. Alice wins to keep the prize as the Audience \
2561    cheer without Eve\'s help!'+ Back.RESET + fg.green +' YOU WIN!'+ Fore.RESET)
2562                            print(fg.green+'These doubles won the prize:'+fg.pink, qdf, fg.green+ ', QDF P match is:'
2563                                  +fg.pink, rndDeltaC)
2564                            winner_show()
2565                            winner_next()
2566             if (takePIn < float(df3Mem) and takePIn>0) and dp==0:
2567                  DeltaP = float(df3Mem)-takePIn
```

```
                if DeltaP > 0.2 and dfcompMem <= 0.5: # This condition could intersect with the winning condition
                                                      # if results are close to DeltaP relative to df3Mem. This
                                                      # creates a win hit and a loss hit as two consecutive rounds
                                                      # (event outcomes) of the game.
                    rndDeltaC = round(1-DeltaP, 2)
                    DeltaC = 1-DeltaP
                    w= round((1/(DeltaC)),3)#-DeltaC # Enable this for conditional DeltaC calculation.
                    print(f'Correlation to strong prediction result for participant #{takeUIn}, {participantMain}, \
is low to win:'+fg.yellow, rndDeltaC, Fore.RESET)
                    if (takeUIn == 1 and takeUIn2 == 3 and (DeltaC >0 and DeltaP>=(1/3))):
                        print(Back.LIGHTYELLOW_EX +
                              'Bob loses to Alice despite Eve\'s help. Alice wins to keep the prize!'
                              + Back.RESET+fg.red+f' YOU {participantMain} via {participantMid} LOSE!'+Fore.RESET)
                        print(fg.orange+'These doubles:'+fg.lightred, qdf, fg.orange+'lost the prize to'
                              +fg.lightred, bitpair, fg.orange+', QDF P match is:'+fg.lightred, rndDeltaC)
                        loser_show()
                        loser_next()
                    elif (takeUIn == 1 and takeUIn2 == 4 and
                          (DeltaC >0 and DeltaP>=(1/3))): # This operation got registered with values like 02, .03 etc.
                          print(Back.LIGHTYELLOW_EX +
                                'Bob loses to Alice despite the Audience cheer without Eve\'s help. \
Alice wins to keep the prize!'+Back.RESET +fg.red +f' YOU {participantMain} via {participantMid} LOSE!'+Fore.RESET)
                          print(fg.green+'These doubles lost the prize:'+fg.lightred, qdf, fg.orange
                                +'lost the prize to'+fg.lightred, bitpair, fg.orange+', QDF P match is:'+fg.pink, rndDeltaC)
                          loser_show()
                          loser_next()
                    elif (takeUIn == 2 and takeUIn2 == 3 and (DeltaC >0 and DeltaP>=(1/3))):
                          print(Back.LIGHTYELLOW_EX + 'Bob loses to Alice despite Eve\'s help. Alice wins to keep the prize!'
                                + Back.RESET+fg.red+f' YOU {participantMain} via {participantMid} LOSE!'+Fore.RESET)
                          print(fg.green+'These doubles lost the prize:'+fg.lightred, qdf, fg.orange+'lost the prize to'
                                +fg.lightred, bitpair, fg.orange+', QDF P match is:'+fg.pink, rndDeltaC)
                          loser_show()
                          loser_next()
                    elif (takeUIn == 2 and takeUIn2 == 4 and (DeltaC >0 and DeltaP>=(1/3))):
                          print(Back.LIGHTYELLOW_EX +
                                'Bob loses to Alice despite the Audience cheer without Eve\'s help. \
Alice wins to keep the prize!'+Back.RESET+fg.red+f' YOU {participantMain} via {participantMid} LOSE!'+Fore.RESET)
                          print(fg.green+'These doubles lost the prize:'+fg.lightred, qdf,
                                fg.orange+'lost the prize to'+fg.lightred, bitpair, fg.orange+', QDF P match is:'
                                +fg.pink, rndDeltaC)
                          loser_show()
                          loser_next()
           else:
                print(bg.orange + fg.green+f'Continue guessing the P as participant #{takeUIn2}, {participantMid}, \
helps you {participantMain}...'+Back.RESET + fg.orange)
                helper_show()  # Grant only 1 point as a helper to the progression of user guesses to come.
                helper_next()
    if (takePIn == abs(p) or takePIn == 0) and dp==0:
        print(bg.blue+'Hello, Classical World!\n', bg.orange + fg.green
              +'Now guess what the QDF outcome would be relative to your next classical guess?'
              +Back.RESET + fg.orange)
    elif (takePIn < float(df3Mem) and (takePIn >= 0.45 and dfcompMem <=0.5)) and dp==0:
        print(bg.blue +'Hello, Quantum-classical World!'+Back.RESET + fg.orange)
        print(f'{fg.purple}Welcome {participantMain} to the world of superposition or entanglement!')
        DeltaP = float(df3Mem)-takePIn
        rndDeltaC = round(1-DeltaP, 2)
        DeltaC = 1-DeltaP
        w= round((1/(DeltaC)),3) # -DeltaC), # Include this in case of further DeltaC calculation.
        print('These doubles \"are all in\" for the prize:' +fg.pink, qdf, fg.green+'QDF P match is:'
              +fg.pink, rndDeltaC)
        print(bg.orange + fg.green+'Now guess what the next QDF outcome would be?'
              +Back.RESET + fg.orange)
        guesser_show()
        guesser_next()
    elif (((takePIn < float(df3Mem) and takePIn >= 0.5) and (dfcompMem <= 0.5 and dfcompMem > 0.48))) and dp==0:
            DeltaP = float(df3Mem)-takePIn
            rndDeltaC = round(1-DeltaP, 2)
            DeltaC = 1-DeltaP
            w = round((1/(DeltaC)),3) # -DeltaC), include in case of further calculation.
            print(bg.blue+'Hello, Quantum-classical World!'+Back.RESET + fg.orange)
            print(f'{fg.purple}Welcome {participantMain} to the world of superposition or entanglement!')
            print(f'These doubles \"are all in\" for the prize: {qdf}', f'QDF P match is: {rndDeltaC}')
            print(bg.orange + fg.green+'Now guess what the next QDF outcome would be?'+Back.RESET + fg.orange)
            dual_show()
            dual_next()
            continue
    if takePIn > 1 or takePIn < 0:
        print('Wrong P value!')
        errorP()
        dp=1
        print('\n')
        continue
##########--END OF PROGRAM--##########
```

▼ Expand/Collapse All…

16 QFLCA Resources



16.1 QFLCA Website Structure: site/index.html

{{ pagetree }}

16.2 QFLCA Project References

Frequently used references to present this project regarding codes, model, method and application are as follows:

1. Philip B. Alipour, T. Aaron Gulliver (2023) Quantum Double-field Model and Application, Elsevier J.
2. Philip B. Alipour, T. Aaron Gulliver (2023) A Double-field Computation Model to Simulate Physical Systems, Elsevier J.
3. Philip B. Alipour, T. Aaron Gulliver (2023) Quantum field lens coding and classification algorithm to predict measurement outcomes MethodsX, Elsevier J.
4. Philip B. Alipour, T. Aaron Gulliver (2024) Quantum AI and hybrid simulators for a Universal Quantum Field Computation Model, MethodsX, Elsevier J.
5. Philip B. Alipour, T. Aaron Gulliver (2024) QF-LCA dataset: Quantum field lens coding algorithm for system state simulation and strong predictions, Data in Brief, Elsevier J.
6. Philip B. Alipour, T. Aaron Gulliver (2024) Quantum field lens coding software for system state simulation, strong prediction and game application, Software Impacts, Elsevier J.

16.3 QFLCA Code, Dataset Samples and Demo Files

7. Philip B. Alipour, T. Aaron Gulliver (2024) QF-LCA Dataset for Quantum Double-field Model, Game and Application, Mendeley Data V3+, DOI:10.17632/gf2s8jkdjf
8. Philip B. Alipour (2024) QFLCA Introductory Demo, `.mp4` file. | For Subtitles: SRT file. ∇
9. Philip B. Alipour (2024) QFLCC Program Demo, `.mp4` file. | For Subtitles: SRT file. ∇
10. Philip B. Alipour (2024) QDF Game Program Demo, `.mp4` file. | For Subtitles: SRT file. ∇
11. Philip B. Alipour (2024) QFLCC Program Demo Updates, `.mp4` file. | For Subtitles: SRT file. ∇
12. Philip B. Alipour (2024) QDF Game Demo Updates, `.mp4` file. | For Subtitles: SRT file. ∇
13. Philip B. Alipour (2024) QFLCC and Game Program Code, `.py` file.
14. Philip B. Alipour (2024) QFLCC and Game Dataset Samples, `{.csv, .htm, .png}` files below:

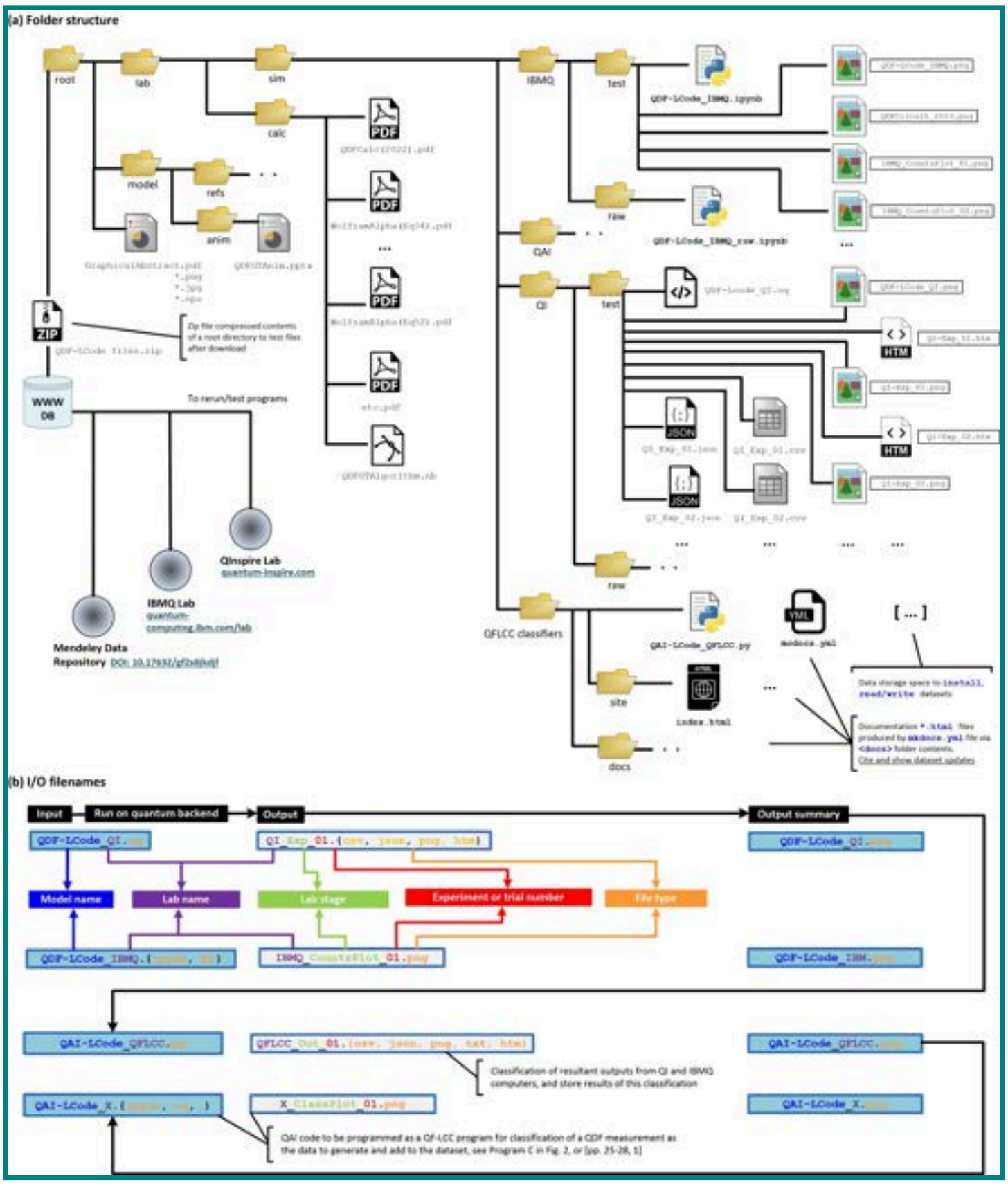
Click on the active links to view file content for `Dataset Samples`:

```
qflcc classifiers
|
site
├   assets
| ├   dataset-samples
| | ├   QI_Exp_01.csv
| | ├   QI_Exp_01.htm
| | ├   QI_Exp_02.csv
| | ├   QI_Exp_02.htm
| | ├   QI_Exp_03.csv
| | ├   QI_Exp_03.htm
| | ├   QI_Exp_03.png
| | ├   QI_Exp_03_H-0.png
| | ├   QI_Exp_03_H-mark.png
| | └   QI_Exp_03_H.png
```

Click on the active links to view file content for `Codes, Modules and Executables`:

```
qflcc classifiers
├     __pycache__
| └   ⊞ QDF-LCode_IBMQ-2024.pyc
site
├   assets
| ├   code-copies
| | ├   QAI-LCode_QFLCC.py
| | ├   QDF-game-gui.py
| | ├   QDF-LCode_IBMQ-2024.ipynb
| | ├   QDF-LCode_IBMQ-2024-codable.py
| | ├   QDF-LCode_IBMQ-2024.py
| | ├     __pycache__
| | | └   ⊞ QDF-LCode_IBMQ-2024.pyc
```

16.4 QFLCA Directory Structure



Run this page offline under the < site > directory to properly view and access < QFLCC classifier > files.

16.5

Remarks

For code use or any related problems such as errors and fixes, webpage design, web content illustration, test and theme installation, contact author via this link .

Project content and website affiliations:

17
Website Theme Switch

The purpose of this webpage is to customize basic theme settings of the website based on your preference. This page also generates your preferred printing color options to save as a pdf file or print pages (this website) from your printer. This is to enable you better web accessibility and readability options as you download, browse and visit pages. Each choice gives you a preview of foreground and background colors applied to this page as an indicator. Upon your choice, the relevant theme will be downloaded to your local drive changing the previous theme settings for your future visits.

17.1 Toggle ☼☽ Mode

Click the button to toggle between dark and light mode for this page. You may apply a color mode for dark or light setting through the following table to all pages.

Toggle dark mode

17.2 Color Generator

Pick a color for your preferred foreground and background webpage color:

☼☽ Mode Color Set		RGB Output
○	Default	□
○	Black	■
○	White	■
○	Red	■
○	Purple	■
○	Teal	■
○	Blue	■
○ Enter an RGB value between (0, 0, 0) for Black, and (255, 255, 255) for White.	Custom	0 0 0 : Background [■] 255 255 255 : Foreground [■] 255 255 255 : Headers [■] Same As Foreground Color: ☐ [■] Custom Preview Reset Form

Download & Apply

Note on file download target: 1- When prompted, choose `Save As`. 2- `Overwrite rgb.css` file with the same filename from: 3- The `<stylesheets> folder` within the `<site> folder` to view changes.

18 Header Preview

18.3 Section Preview

This is the expected RGB result for the page text. Hover over this cell to view the background color.