EasyChair Preprint

№ 16020

Introducing QAL: the Quantum Algorithms Lab (extended version)

Moez Abdelgawad

May 17, 2026

Introducing QAL: the Quantum Algorithms Lab

(extended version)

Moez A. AbdelGawad
Informatics Research Institute (IRI), SRTA-City, Egypt
moez@srtacity.sci.eg

***Abstract*—This paper introduces QAL, the Quantum Algorithms Lab. QAL is a novel and innovative visual interactive tool (currently in the form of a web app, as a virtual playground) for experimenting with quantum algorithms and their underlying math, such as linear algebra, group theory, and graph theory. In addition to quantum computing, the mathematics presented intuitively in QAL has applications in fields such as machine learning and other mathematical sciences. (A lightweight edition of QAL is available online at https://q-info.github.io/QAL-Lite.)**

I. Introduction

Quantum computing is an emerging technology that has significant potential. Interest in quantum computing is mainly due to the potential of exploiting the unusual laws of quantum physics to significantly speedup some computations. But *which* computations? And *how much* speedups can quantum computers achieve for these privileged computations? These two crucial questions are the main topics of interest in quantum algorithms research, and in the closely related fields of quantum complexity theory and quantum informatics/information science [27].

Discovering truly efficient quantum algorithms (*i.e.*, computations with an exponential quantum speedup, such as Shor's renowned epoch-making quantum algorithm for factoring integers [57]) is, unfortunately, an extraordinarily challenging and extremely difficult endeavor [48], [33]. The field of researching and teaching quantum algorithms, to find practical efficient ones, seems to be of the hardest and most counterintuitive fields.

Fortunately however, modern tools for research and pedagogy may help. In particular, human brains seem to be hardwired to think in pictures. Paraphrasing [31], harnessing the power of *interactive pictures* in quantum science education and research has the potential to unleash a huge untapped creative power to innovate, communicate, and problem solve.[1]

Given these facts, researching quantum algorithms, in particular, may benefit immensely if the "right" tools are developed and used to help in teaching and researching them, *e.g.*, using a "calculated trial and error" (or "what if") experimentation approach[2]. Plenty of visual interactive tools that are successfully used to teach and research various scientific fields, such as machine learning/AI, robotics, chemistry, physics, ..., etc., testify to the effectiveness of such an approach.

QAL (Quantum Algorithms Lab) is a new visual interactive tool for experimenting and playing with general-purpose gate-based quantum algorithms, built with the specific goal of helping learners and researchers develop a solid, intuitive and deep understanding of *the underlying mathematical properties* of quantum algorithms, as expressed in linear algebra, group theory, and graph theory.

As a teaching tool, QAL targets school students (*e.g.*, at STEM schools) and undergraduate students of computer science, mathematics, and physics as its main users.

As a research tool, researchers and graduate students in various scientific fields can strongly benefit from QAL. In addition to quantum computing, the math nicely and intuitively presented in QAL has applications in plenty of mathematical sciences, such as machine learning/AI, computer graphics, cyber-security, robotics, and mathematical physics[3].

This paper is structured as follows. Sec. II briefly describes QAL, its design philosophy, its UI layout, and it introduces QAL's 'algorithm presentations.' The section also gives two simple examples of how QAL may be used in mathematical "investigations" of quantum algorithms and linear algebra. It also describes QAL's code architecture and QAL Lite—the lightweight edition of QAL[4]—and it discusses the difference (called QA-LA NIVs) between the 'full' and 'lite' editions of QAL.

Sec. III describes, via example scenarios, the two main potential use cases of QAL, *i.e.*, in teaching and in research. Sec. IV then presents a summary of works that are similar to QAL, and compares QAL Lite and QAL to some of these works. Sec. V presents an outline indicating and hinting at what future versions of QAL may offer and look like, and, very briefly, discusses how QAL may be adapted to present inner workings on matrices in other scientific areas. Sec. VI concludes the paper. (The Appendix presents few screenshots of QAL Lite that illustrate its description and how QAL may be used.).

[1] According to [31], "Albert Einstein, best known for a textual equation, once said he didn't think in words at all."

[2] Where educated guesses of experiments are performed until a (mathematical) property is well-understood or a satisfactory answer is found. In the context of quantum algorithms, informed guesses (*e.g.*, quantum operations and algorithms) are made, tested, and refined based on resulting errors.

[3] Many programming and computing systems are built around linear algebra – including machine learning frameworks, graphics pipelines, and scientific computing – because matrix operations (*e.g.*, multiplication) are parallelizable and hardware-friendly.

[4] Freely available for use online at https://q-info.github.io/QAL-Lite.

II. Description of QAL

An algorithm is a finite sequence of steps (*i.e.*, operations, moves, or actions) that fulfill or perform a task. In other words, an algorithm is a sequence of steps (with a common goal, to solve a specific problem). A *quantum algorithm* is a finite sequence of quantum steps/operations. Mathematically-speaking, a quantum operation is a linear map that can be represented or encoded as a *matrix* of complex numbers, with matrix entries representing quantum amplitudes [48], [67], [35], [33], [42]. Steps of the algorithm (*i.e.*, elements of the sequence) are *composed* to define the *overall* action/operation of the algorithm (*i.e.*, its function and purpose). When algorithm steps are represented as matrices, the composition of steps is represented mathematically as *matrix multiplication*[5].

A. QAL Overview

QAL helps its users explore and experiment with quantum algorithms and related linear algebra concepts by exhibiting a number of standard and novel colored diagrams (or *visuals*) that present a quantum algorithm as a sequence of matrices.

In QAL, a sequence of matrices that represents an algorithm is visually presented in one or more series of colored diagrams, with each diagram in a series corresponding to an algorithm step. Each series of visuals is typically presented to the QAL user going from left to right, modeling the flow of time. A colored diagram (*i.e.*, visual) presenting the overall operation of the algorithm, resulting from the composition of its steps, is also presented on the far right of each series of visuals (right next to the usually blue arrow $\Rrightarrow$ that follows each series).[6] Further, a visualization of quantum states (*i.e.*, of state vectors) before/after each step of the algorithm can be displayed or turned off. These visuals constitute the main 'QAL Presentations' section of QAL. (See Fig. 2 in the Appendix.)

QAL can present the sequence (of matrices) representing an algorithm using *multiple* different kinds of visuals. Each kind of diagrams is called an *algorithm presentation*. A QAL user can choose which algorithm presentations to display and which ones to turn off[7]. Some of these kinds of presentations are standard ones (*e.g.*, heatmaps, also called HM diagrams, circuit diagrams, ..., etc.), and are available for use in QAL Lite. Other kinds of QAL algorithm presentations are novel ones. These novel ones are implemented in QA-LA NIVs, which we are seeking to patent, and are available for use only in QAL but not in QAL Lite[8]. (Sec. II-C and Sec. II-D, below, present more on this difference—the main and most significant difference—between QAL and QAL Lite.)

[5] As such, QAL can equally, if less attractively and less accurately, be called 'matrix multiplication lab.'

[6] A QAL *subalgorithm* is a contiguous section of a QAL algorithm, whose boundaries (*i.e.*, beginning and ending steps) can be easily specified by the QAL user. To give QAL users more flexibility in experimentation and exploration, the right-most visual (in an algorithm presentation) actually presents the overall operation of the currently specified subalgorithm (which, unless changed by the user, defaults to the whole QAL algorithm).

[7] Via on/off settings in the Presentation Settings of the QAL Dashboard.

[8] Thus, we cannot currently publish the details of these novel algorithm presentations.

The QAL user can directly interact with these visuals, and can use them (or use the 'QAL Dashboard,' in the top foldable section of QAL) to modify the steps of his or her quantum algorithm and *immediately* see "live" (*i.e.*, in real-time or near-real-time) these changes reflected in the currently displayed presentations of the algorithm, and, much more importantly, immediately see their effect on the overall operation of the algorithm.

In particular, each entry (a complex number in *polar* form) of a matrix that represents an algorithm step and each entry of the vector (typically displayed on the very far left) that represents the input state to the algorithm can be easily—*i.e.*, directly and intuitively—selected and modified in QAL, via few simple mouse clicks, or touch screen taps, or via simple drag/drop actions of visuals and on-screen objects (*i.e.*, *without* involving a keyboard to type in, for example, numbers, symbols, or words.)

In most QAL algorithm presentations, colors are used to visually present the complex number (*i.e.*, the quantum amplitude) of each entry, where the 'hue' of the color typically reflects the phase of the amplitude while the 'saturation' of the color typically reflects the magnitude of the amplitude. The magnitude and phase of the amplitude can then be changed directly (*e.g.*, via two sliders in the 'Current Selection Settings' section of the QAL Dashboard, or via direct drag/drop actions of graphical elements in QAL Presentations). Entries of the *result* matrix (representing the *overall operation* of the algorithm, presented in the diagram at the far right of algorithm presentation) and of the 'final state histogram' (on the right of the result matrix of the topmost displayed algorithm presentation) change accordingly, and *instantaneously*.

Doing such 'select, then change, then observe' steps in QAL is available for *all* input matrix and vector entries presented in QAL (*i.e.*, all their amplitudes). In addition to single matrix/vector entries, QAL allows changing multiple matrix entries or *whole* matrices and vectors—or whole columns or rows, in matrices—directly and simultaneously, in one step, by clicking or tapping one button (again, with no textual or symbolic editing), and it allows observing in real-time the effect of these changes.[9]

By QAL presenting vectors and matrices visually, in multiple parallel views/presentations (*i.e.*, kinds of diagrams, optionally presented simultaneously), and by it allowing direct interactions with the entries of vectors and matrices and allowing to change them, and by it presenting the immediate effect of these changes on (presentations of) the overall operation, QAL allows its users to *experiment readily* with the vectors and matrices, which helps significantly improve understanding linear algebra and group theory, as well as understanding the subject matters—such as quantum algorithms, neural networks, encryption/decryption algorithms, etc.—that are modeled by the vectors and matrices that QAL visually presents.

[9] A brief video illustrating changing matrices and settings in QAL, and more, is available online at https://q-info.github.io/QAL-Lite/vids/. QAL users are invited to explore and experiment with— *i.e.*, "play with" — the gadgets available in the 'Current Selection Settings' and 'Presentation Settings' sections of the QAL Dashboard.

In summary, QAL tries to motivate developing intuitions—about noiseless/fault-tolerant quantum algorithms—by QAL allowing its users to "dig into" the mathematics underlying quantum algorithms and motivating them to develop strong intuitions about this math, via QAL getting very close, technologically-speaking, to linear algebra (*i.e.*, to intuitive presentations of vectors and matrices of complex numbers, and of their composition/multiplication) and via QAL further allowing and motivating interacting and experimenting with linear algebra rather than abstracting out this underlying math or abstracting it aside.[10]

B. Experimental Mathematics in QAL

In this section we describe, by giving visual examples, how QAL can be solidly used to do experimental mathematics [19], [17][18], [13], [12], [26], [53], to discover and ascertain sample properties and facts of quantum algorithms and linear algebra (among many others).

1) Simple Examples of Mathematical Investigations using QAL: To illustrate how visualization and simple interaction in QAL may be used (in teaching or in research) to improve understanding the mathematics underlying quantum algorithms, consider the following two simple examples.

Example 1. (QFT^2 and NOT/X)

Two intentionally simple reproducible observations and mathematical facts that QAL, and even QAL Lite, easily allows making and confirming are that:

1) Following a NOT operation (logical negation, also called X gate) by two applications of a QFT (Quantum Fourier Transform) operation is equivalent to the 'add 1 (with wraparound)' operation, *a.k.a.*, rotate clockwise, and
2) Following two applications of QFT by a NOT operation is equivalent to the 'subtract 1 (with wraparound)' operation, *a.k.a.*, rotate counterclockwise.

Expressed algebraically, QAL allows easily confirming that

$$QFT \circ QFT \circ X = AddOne \quad (\hat{=} \; Succ \hat{=} \; RotCW)$$

and

$$X \circ QFT \circ QFT = SubOne \quad (\hat{=} \; Pred \hat{=} \; RotCCW).$$

Fig. 3 and Fig. 4 in the Appendix present how these two facts can be easily confirmed, visually, in QAL.

Example 2. (NOT/X and flipping matrices)

Two even simpler mathematical facts of linear algebra that QAL can easily help in confirming are:

1) *Right/post*-multiplication of any matrix R by NOT (*i.e.*, the matrix $R \circ X$ or RX in mathematical notation, or $X; R$ in programming notation[11]) *flips columns* of the matrix R around a vertical axis that goes along the vertical center of the matrix (*i.e.*, RX is equivalent to $fliplr(R)$, the left-to-right flipping of R), and
2) *Left/pre*-multiplication by NOT (*i.e.*, matrix $X \circ R$, XR, or $R; X$) *flips rows* of the matrix R around a horizontal axis that goes along the horizontal center of the matrix (*i.e.*, XR is equivalent to $flipud(R)$, the upside-down flipping of R).

Fig. 5 and Fig. 6 in the Appendix present how these two facts can be easily confirmed, visually and interactively, in QAL.[12]

If seeing and interacting with one example (*i.e.*, one sample random matrix R) is not convincing enough to the QAL user—as should be the case (since 'looks may be deceptive')—then after visually experimenting with multiple different *random* matrices (which can be done easily, in *few seconds*, in QAL), to try finding *counterexample(s)* to their tentative facts, the user then usually gets *visually certain* of both facts: multiplying a matrix by X on the right flips columns of the matrix, while multiplying the same matrix by X on the left flips its rows (recall: matrix multiplication is *not* commutative).

2) More Complex Conjectures & Visual Interactive Mathematics: Most interestingly, in case the QAL user (whether a student, a researcher, or otherwise, a curious explorer of mathematics) makes observations of deeper tentative facts—*i.e.*, ones that are more complex than the ones in Example 1 and Example 2—that may involve, for example, multiple and more complex gates, vectors, and matrices[13], then the QAL user[14] may *later* try to spend some time to formally prove his or her tentative facts—*a.k.a.*, 'conjectures'—mathematically, using, *e.g.*, a formal mathematical linear algebraic proof [15].

Even further, if the QAL user *cannot find* a formal proof, he or she may *try to use QAL to do more investigations* (*i.e., design more QAL experiments*) *to figure out what may be missing*

[10] The simple mathematical equation $1 + 1 = 2$ abstractly expresses or corresponds to a more fundamental underlying fact of life, that is, that if an object of a kind is grouped together with—*a.k.a.*, 'added to'—another object (that is considered) of the same kind, then one *always* gets **two** objects of that same kind. This same 'abstraction power of math' exists in *all* branches of mathematics, including linear algebra and group theory, with their notions, facts, and theorems expressing many general intuitive real-life notions. QAL tries to vividly bring out to its users' attention and knowledge (using visualization and interaction) some of these intuitive notions and facts of life that underlie these branches of math. (Consider, for example, the fact that 'pulling weaves'—or, '"pulling" concatenated graphs'—in real-life corresponds to '(permutation) matrix multiplication' in linear algebra—a fact illustrated vividly in some diagrams of QAL.)

[11] Meaning, time-wise, that operation X is performed *first*, then R—which is counter-intuitive to the order suggested by the name *post* whose origin is the ordering of ops in math notation, not in programming notation.

[12] Note that these two facts of Example 2, together with the two facts in Example 1, imply that QFT^2 is both a flipped left-to-right $Succ$ and a flipped upside-down $Pred$, which in turn implies that $Succ$ and $Pred$ are closely related by each of them being the 'flip upside-down and flip left-to-right' of the other (recall: each flipping is its own inverse). This fact in turn—given that $Succ$ and $Pred$ are symmetric along the anti/minor diagonal (See Fig. 3 and Fig. 4 in the Appendix)—implies that both $Succ$ and $Pred$ are transposes of each other ('transposing inverts permutations' is a well-known linear algebraic fact, which QAL merely confirms visually).

Clearly, linear algebra is strongly (and sometimes confusingly) interconnected. And having a *visual* view of matrix operations (*e.g.*, to sort out—*i.e.*, express and visually confirm—its interconnections, relations, theorems, and facts) is "obvious fun!"

[13] For starters, try facts involving the *conjugates* and *commutators* of non-commutative quantum operations, for example.

[14] To be certain beyond doubt, and to be extremely cautious (since 'looks may still be deceptive' and given that 'the visually obvious is not necessarily always true').

[15] Possibly getting help from increasingly-automated proof assistants such as Lean [46], [44], QEC [20], QHL Prover [65], ..., etc.

in their (linear algebra) understanding and knowledge so that the user is able to express the "visually indisputable" facts that they are seeing in QAL using formulae and standard mathematical expressions—ultimately leading (if a formal proof is found) to the tentative facts becoming true *indisputable* facts to the user (*i.e.*, acceptable even by the highly-rigorous, high-quality, standard norms and rules of the mathematical community, which allow no "deception" or misconception, and thus allow zero chance of perception errors).

These examples and other more complex ones (*e.g.*, in QAL's nascent user guide) illustrate that QAL significantly improves the intuitive understanding of quantum algorithms, and of their underlying mathematics, through *simple* (*i.e.*, non-distracting) *interaction with mathematically-oriented visualizations*. By QAL (1) visually suggesting mathematical facts and properties, as illustrated by the simple Example 1 and the very simple Example 2 above, and by it (2) offering simple means to visually confirm these facts by designing and doing more experiments ([19], [17]), and (3) by the tentative facts motivating QAL users to *search for possible flaws* in their understanding (and to fix any they may find), the examples illustrate the true power and value of QAL (and of similar visual interactive tools of mathematics, for that matter): a powerful, versatile, simple, and enjoyable game-like *guide* to proper mathematical understanding.

C. QAL Architecture and Code Components

QAL, as so far implemented, is relatively-speaking a simple standard web app, since QAL is built using software technologies for web development that are widely and easily available on today's standard web browsers.

Namely, QAL is implemented using plain HTML5/CSS and JavaScript/TypeScript (ES2016) code. In particular, for rendering its characteristic trademark graphics, in QAL Presentations, QAL makes use of the excellent `D3` JavaScript library (d3js.org), and for GUI controls and dials in the QAL Dashboard, QAL makes use of the `jQueryUI` JavaScript library (jqueryui.com). The wide availability of these standard and strongly supported web technologies on most platforms allows QAL to run smoothly on all standard recent web browsers. (See Sec. IV for a comparison of QAL to similar tools.)

As a software program, QAL employs the powerful MVC (model-view-controller) software architecture. As depicted in Fig. 1, the code-base and main modules of QAL are structured as follows. The source code of QAL's 'model' component resides in two modules: one (`QAL-math.ts`) for handling mathematics relevant to quantum algorithms (*i.e.*, mostly supporting mathematical complex number operations, linear algebra—*i.e.*, vector/matrix—operations, as well as permutation operations), and the other (`QAL-algorithm.ts`) for handling and managing QAL's algorithms (as sequences of complex-valued vectors and matrices, implemented using JavaScript arrays), and for saving/opening algorithms to local (JSON-formatted) files. These two model modules form the *common* basis for both QAL and QAL Lite.

The source code for QAL's 'view' component and for QAL's 'controller' component (*i.e.*, code for implementing its graphics, in algorithm presentations, and code for implementing its controller and GUI controls, respectively) resides in multiple separate modules (`QAL-graphics-xx.ts` and `QAL-control-xx.ts`). Collectively, these modules are composed *functionally* of two distinct parts, which have separate responsibilities in the app, and that are thus maintained largely separately inside each module. The main QAL user interface (UI), including most of the QAL Dashboard, and few (currently 3) basic interactive visuals constitute **QAL Lite**—the first part of QAL's graphics and UI.

A number (currently 7) of new advanced interactive visuals, together with few Dashboard dials and UI controls, constitute—functionally—the second part of QAL. These advanced visuals, collectively, are named the 'Quantum Algorithms/Linear Algebra Novel Interactive Visuals' (**QA-LA NIVs**), and are implemented in separate pluggable submodules (`QAL-adv-graphics-xx.ts` and `QAL-adv-control-xx.ts`). Without these advanced submodules QAL, *by design*, is automatically just QAL Lite.[16]

QAL versus QAL Lite: An Analogy: The relation between QAL Lite, QAL, the basic interactive visuals (in QAL Lite), and QA-LA NIVs (the novel advanced interactive visuals in QAL, which connect QAL to graph theory, and to bipartite graph theory in particular [38], [42], [47], [54]) can probably be best explained via an analogy—one that likens interactive visuals to engines/motors of vehicles (*e.g.*, cars, trains, or airplanes).

In such an analogy, QAL Lite is like a basic vehicle that has a basic engine (corresponding to the 3 basic interactive visuals). With no interactive visuals, QAL and QAL Lite are like the vehicle's frame & body that, having no "engines" (corresponding to interactive visuals), cannot move. With a basic engine (as in QAL Lite), the tool/vehicle can move but only slowly and not too far.

Adding QA-LA NIVs to QAL Lite to form QAL, *i.e.*, having

$$\text{QAL Lite} + \text{QA-LA NIVs} = \text{QAL},$$

is analogous to providing the vehicle with a more powerful engine. This new engine extends the power and reach of the vehicle *many* folds—corresponding to 'QA-LA NIVs significantly increasing the pedagogic and research value of QAL in exposing the *structure* of quantum operations/complex matrices and their mathematical properties.'

Using such an analogy, QAL Lite is like a regular car (with a basic engine) while QAL is like a powerful, sports & luxurious

[16] Both QAL Lite and QA-LA NIVs have current basic implementations, in JavaScript/TypeScript, and both can be significantly developed further (see Sec. V). QAL Lite is a copyrighted (temporarily obfuscated) open-source software that is available to use *for free* online. Since we are seeking to patent QA-LA NIVs at the USPTO (United States Patent and Trademark Office) as novel software technologies, more details on QA-LA NIVs are currently available only after signing an NDA (Non-Disclosure Agreement). As hinted to below (Sec. III-B), QA-LA NIVs add significantly to the pedagogy and research value of QAL, *e.g.*, by them employing *graph theory* and *group theory* tools and notions to visually and interactively reveal more of the mathematical (*e.g.*, recursive, spectral, and group-theoretic) structure of quantum operations.

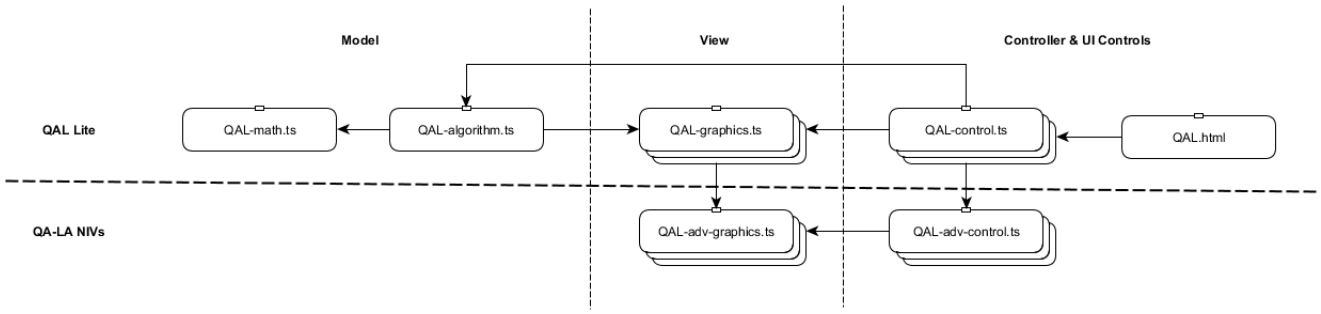

Figure 1. The structure of the QAL code-base. An arrow from a module to another indicates the source module *calls* the target module. QA-LA NIVs 'view' and 'control' modules — in `QAL-adv-xx.ts` files — are called only if their availability is detected by QAL.html and their advanced algorithm presentations are requested by the user. (QAL module names can be easily recalled using the acronym MAGiC, where 'M' is for 'math', 'A' is for 'algorithm', ... and 'i' is for QA-LA NiVs, pun *unintended*.)

car (with a much more advanced and powerful engine). Similar to how a sports or luxury car (supposedly) can do everything a regular car can do, and much more, via adding QA-LA NIVs to QAL Lite the features of QAL include—*i.e.*, are a strict superset of—all the features of QAL Lite.

D. QAL Lite Presentations

The three basic kinds of interactive visuals currently available in QAL Lite present a quantum algorithm as

1) a sequence of heatmaps (HMs) and a result/overall heatmap,
2) a sequence of circuit blocks, *i.e.*, quantum steps/ops in familiar quantum circuit diagrams, and
3) (mainly for completeness and debugging purposes) a sequence of complex matrices, in textual form, and a result/overall matrix.

As mentioned earlier, visualization of quantum states (*i.e.*, of vectors representing state) before/after each step of the algorithm can be displayed or turned off. (Few screenshots of QAL Lite are presented in the Appendix, and more ones can be found in the 'QAL Lite Screenshots' document available online.)

Further, in QAL, the visualizations in QA LA-NIVs help reveal *more* of the structure of quantum operations, leading to better comprehension of complex concepts of quantum algorithms (such as their recursive/self-similar structure [67]).

III. Usage of QAL

A. QAL in Teaching

As discussed earlier, an *algorithm* (also sometimes called a *classical algorithm* in quantum computing contexts) is a sequence of classical steps or operations (*a.k.a.*, moves or actions) — basic operations like AND, OR, NOT, XOR, or composite ones, like HALF-ADD, ADD, MULT, and so on.

A *reversible algorithm* is a sequence of (basic or composite) reversible operations — ones like NOT, SWAP, CNOT, CCNOT, CSWAP, and so on.

A *quantum algorithm* is a sequence of (basic or composite) quantum operations — ones like NOT, NEG, CNOT, CCNOT, CSWAP, CNEG, HAD (*a.k.a.*, X, Z, CX, Toff(oli), Fred(kin), CZ, and H, respectively), QFT (Quantum Fourier Transform), and so on.

Now, if a teacher talks to some high-school students (say, *e.g.*, of quantum programming [7]), firstly, about complex numbers, then, secondly, talks to them about matrices and vectors in general then, thirdly, about matrices and vectors of complex numbers in particular, then, fourthly, about the composition or multiplication of sequences of such matrices and vectors, then likely–given the many layers of abstract mathematics involved–many students will feel overwhelmed, and may think of mathematical models of quantum algorithms as complex, too abstract, and probably non-intuitive. Consequently, most of these students will, likely, feel disconnected from quantum algorithms and quantum informatics.

But in agreement with teachers encouraging their students to 'not think in words,' if the teacher uses QAL to present to the students some paintings and drawings that can be "merged," or uses QAL to present the students with some pipes or threads that can be connected, or presents them with some other everyday objects that students can easily interact with in QAL (more so in QAL VR. See below), the students will likely feel more "at home" (since they are dealing with everyday objects, not abstract words and symbols) and they will think of quantum algorithms and quantum operations as being somewhat similar to concrete objects that the students are already familiar with, ones that—like pieces of Lego™ or cubies of a Rubik's Cube—they can interact, experiment, and "play" with readily.[17]

B. QAL in Research

As mentioned earlier, quantum computing is an important emerging technology that has significant potential to make large real-world impact. As a tool designed to guide and help, QAL is also potentially a large and useful software

[17] According to [31], "Visual [and interactive] thinking and visual [and interactive] communication can make audiences more engaged in what they are watching [and interacting with], help them better understand the messages delivered, make them feel positive and make them far more likely to share the information with others. When people employ the techniques of visual [and interactive, game-like] thinking in their communication, it can literally *make the invisible visible* [and enjoyable]."

that ambitiously aims to help quantum computing reach its potential.

Because quantum computation is connected to various branches of science including physics, computer science, informatics [28], [27], and mathematics, quantum computing (QC, henceforth) is a vast field. Like classical computing, QC has two components: a hardware component and a software component. Unlike classical computing, large-scale fault-tolerant QC has not become a reality so far, and thus it has not reached its full potential yet, for two main reasons: (1) lack of large-scale quantum computers (a hardware challenge), and (2) lack of *many* quantum computing applications (a software challenge). QC, so far, is proven to solve only very few practical computing problems more efficiently than classical computing does, *e.g.*, factoring whole numbers[18], searching unstructured data, solving systems of linear equations, and simulating certain chemical and physical systems (see [52], [48], [37]).

As a visual and interactive "virtual lab" the Quantum Algorithms Lab (QAL) can function as a quantum algorithms research assistant (QA RA) that can help in addressing the algorithmic *software* challenges of QC, by it visually and interactively helping graduate students and researchers understand and develop intuitions for the structural properties of quantum algorithms, and of their power (and limitations).

Examples: Among the subareas of quantum algorithms research that QAL has an inherent affinity to aid in are:

1) Analyzing the eigenvectors and eigenvalues (known as *spectrum*) of (efficient) quantum operations—using tools such as ones used in Spectral Graph Theory (SGT) [22], [30], [45],
2) Visually and interactively studying the recursive (self-similar) structure of *efficient* quantum operations [67], [5], [6]. Given that QAL attempts to present algorithms as conceptually intuitive as possible, this research may possibly involve using mathematical tools from order theory and category theory [40], [59] (such as operads and monads [58], [4], [3], [2], or such as monoidal categories, as used to define ZX-calculus [24], [23]).
3) Also, we are currently working on using QAL (particularly some of the graph-theoretic power of QA-LA NIVs), together with a new QLS algorithm [41], to better understand the non-commutativity of quantum operations and some particular related properties (*e.g.*, the 'support' of an operation), visually and interactively. This research may possibly help resolve the non-commutative hidden subgroup problem (HSP), and it may thus possibly help in finding (or proving the non-existence of) an efficient quantum algorithm for the Graph Isomorphism (GI) problem—a well-known computational problem that has applications in areas such as drug discovery and logistics.[19]

[18]Presented in [57], which can be used to break current RSA-based (and, with a twist, elliptic curve-based) secure communication systems and, thus, indirectly help in building future secure "quantum-resistant" ones.

IV. Related and Similar Work

QAL strongly adopts a visual interactive approach to mathematics. Like-minded software includes popular apps such as Desmos, which uses interactive and animated visuals and a foldable dashboard to teach basic high school math (desmos.com). A much earlier software (for desktop PCs) that adopted a similar approach was Geometer. GeoGebra (geogebra.org) is a more recent software similar to Geometer that's available for all major platforms.[20]

Of the many factors involved more directly in its inception, QAL has been initially and mainly motivated and inspired by the Circle Notation of Eric Johnston [37], which we effectively used in teaching this course [7], and Colin Williams' heatmap diagrams of the QFT in [67]. Also the diagrams in [38] (which barely mentions quantum computing) and in [42], [24] have been secondary motivations behind developing QAL.

Many desktop and smartphone apps have been developed that attempt to present quantum programming and the construction of quantum algorithms as a game. These apps include Meqanic (for iOS, meqanic.com), Quantum Park (for iPad, on App Store), QLogic (for Android, qlogic.paltangames.com; currently offline), and IBM's Hello Quantum (helloquantum.research.ibm.com).

Also, many web apps (such as IBM's Quantum Composer [51], Quirk, algassert.com/quirk, and Quantum Flytrap, quantumgame.io) exist that attempt to present and teach quantum theory and quantum software visually and interactively.

Recent works on the use of visual interactive approaches to quantum pedagogy and research include VENUS [56], Quantivine [66], and Quaver [14] as well as QCVis (github.com/fh-igd-iet/qcvis), VQS (github.com/gmenier/VisualQuantumSimulator/), QuantumEyes [55], and, most recently, QSynVis [34]. Some, in fact most, of these tools and projects are now inactive, defunct, or discontinued. More importantly, none of these works visually presents quantum algorithms nor allows interacting directly and simply with them using visuals as QAL does. (Table I, Table II, as well as Sec. VI below, offer more details on how QAL compares to some of these somewhat similar tools.)

Other software bearing some similarities to QAL includes mobile, desktop, and web apps and games built using R, Julia, Mathematica, SageMath, Matlab, Python and other similar platforms and frameworks for developing visual interactive mathematical software. Such apps have gained popularity in various subfields of computer science, *e.g.*, data science, machine learning/AI, and high-performance computing (HPC).

[19]That is, help decide with certainty, in a constructive yes/no manner, whether a truly efficient quantum algorithm exists for solving GI. Since the fastest classical GI algorithm has quasi-polynomial runtime [11], [32], this means finding, for example, a *polynomial-time* quantum algorithm for GI (so as to be a quantum algorithm with a *significant* speedup. See Sec. I).

[20]Of the excellent textbooks that adopt a strongly visual approach to mathematics is [21], which (in hindsight), together with its accompanying *Group Explorer* software, was a very early inspiration for developing QAL as a visual tool, as well as numerous other older and more recent mathematics literature (such as [49], [62], [10] and [60], [16]).

Tool / Feature	Primary Focus	Target Audience	Key Features
QAL	*Low* (*i.e.*, entries/amplitudes)-*level and high* (*i.e.*, digits/gates)-*level* visualization and interaction with quantum algorithms and their underlying mathematical concepts (*e.g.*, vectors, matrices, graphs, groups)	Beginners, researchers, hobbyists, *visual artists*, developers, advanced students	Simple intuitive visualization and interaction, multi-view simultaneous display, speed, real-time feedback, no coding required, novel *math-oriented* interactive visuals (incl. CN-HMs, AN-HMs, F-HMs, CCDs, F-CCDs, CCSDs, SCCSDs, ...)
QAL Lite	Matrix entry/amplitude-level visualization and interaction with quantum algorithms and their underlying mathematical concepts	Beginners (students & researchers), hobbyists	Simple intuitive visualization and interaction, multi-view simultaneous display, real-time feedback, no coding, standard interactive visuals (*e.g.*, heatmaps)
IBM Quantum Composer	Quantum circuits (drag & drop), (learning full stack quantum programming, run circuits on real QCs or advanced simulators)	Researchers, developers, advanced students	Real hardware execution, professional workflow, OpenQASM / Qiskit code editor, tightly-integrated visualization (Q-sphere, phase disks, ..., etc.)
Quirk	Quantum circuits (drag & drop)	Beginners, hobbyists	Simple, speed, real-time feedback, no coding required
Quantum Flytrap	Game-like web app (visual, drag & drop), building physics intuition	Educators, students	Optical table, "many worlds" tree
QSynVis	Circuit synthesis and optimization	Researchers, advanced students	Visualizing algorithm optimization and synthesis
QuantumEyes	Visual analytics to interpret and debug quantum circuits, research oriented	Researchers, algorithm designers	Visual analytics, Dandelion chart (amplitude to probability), probability evolution views
GeoGebra	General math (customizable)	Educators, students (custom content)	Custom simulation building, coding optional (to build simulations)

Table I
A COMPARISON OF QAL AND SIMILAR TOOLS (PART I)

Tool / Feature	Access and cost (Platform)	Limitations
QAL	Proprietary, partially open-source (prototype web app; similar to QAL Lite)	No quantum hardware access, no noise modeling, # qubits limited only by client memory; in practice, slows down around 10 qubits/1024 "parts/eigenstates"
QAL Lite	Free (no accounts required), temporarily obfuscated open-source web app	No quantum hardware access, no noise modeling, real-time feedback slows down after 6 qubits
IBM Quantum Composer	Free (with IBM account, web app/cloud access)	Max. 6 qubits for free accounts, steeper learning curve
Quirk	Open source (Free web app)	No quantum hardware access
Quantum Flytrap	Free web app	No quantum hardware access, simulates up to 3 photons
QSynVis & QuantumEyes	Research prototypes	No quantum hardware access
GeoGebra	Open source (free mobile / web / desktop app)	No quantum hardware access

Table II
A COMPARISON OF QAL AND SIMILAR TOOLS (PART II)

V. FUTURE WORK

Based on the large number of software development projects that can be built on top of QAL, can be derived from it, or can improve it, QAL is potentially a mega software project. QAL can thus be considered currently an alpha software, *i.e.*, one in its early development phases. (The latest version of QAL is QAL 0.9.5.)

In particular, QAL can be further developed in multiple directions, including, for example, (1) by adding ZX diagrams (see [24], [23]) as a basic interactive visual to the set of QAL Lite algorithm presentations, (2) by adding more features to QAL such as supporting *constrained* changes to matrix entries (*e.g.*, to preserve the unitarity of matrices), (3) by activating and expanding the Game Mode of QAL (*e.g.*, in imitation of QLogic), (4) by developing a more immersive edition of QAL that makes use of VR's pedagogic and interactive capacities[21], and/or (5) by porting QAL to other platforms (*e.g.*, to improve its performance on smartphones and other handheld devices) and programming languages (*e.g.*, Julia, R, Matlab, or C++). Also, other advanced yet standard technologies, such as using 'web assembly' files (which are widely used in developing online games), may be later used to speedup QAL's graphics rendering performance.

As hinted to in Sec. II, QAL has its proprietary (JSON-based) format for saving and opening algorithm files. As such, another "quantum software" development project that is strongly related to QAL is building **QAL-Lib**, as a library/repository of quantum algorithms. When developed, QAL-Lib can be used to *visually illustrate* the quantum and

[21] Where users, for example, can get even closer to the graphical objects QAL presents by "touching" them and interacting with them more immersively. Also, as a "game," sound effects can be added to QAL VR, where, for example, "sounds" (for vector/matrix-entries) can be heard upon interaction with an entry's amplitudes (with an option to silence/turn-off all sounds, of course). In QAL VR, "diving into linear algebra and graphs" will not merely be a nice metaphor or a nice figure of speech but a reality—a virtual reality, that is. For the latest publicly-available news on QAL VR, check http://eng.staff.alexu.edu.eg/staff/moez/QAL/VR/.

reversible/permutation algorithms[22] presented in multiple references and textbooks on quantum algorithms, linear algebra, and group theory (such as [38], [64], [8], [15], [42], [47]). Also, to help QAL better adhere to established quantum computing standards and frameworks, QAL can be made to interface with standard quantum circuit representations (*e.g.*, OpenQASM 2.0), quantum intermediate representations (QIR), and existing software development kits (SDKs) like Qiskit [36] or Cirq [61]. This standardization will make QAL better integrate with the existing quantum computing ecosystem and will make its contributions easier to appreciate and to integrate into existing workflows, and to situate its contributions within the existing landscape of quantum simulators and educational tools.

Also, QA-LA NIVs currently include seven fully implemented novel interactive visuals. Few more novel interactive visuals are currently under active development for inclusion in future versions of QAL. Several novel advanced interactive visuals can be further added, in future work, to QA-LA NIVs.

In its philosophy and design, QAL is strongly geared towards modeling quantum algorithms—hence its name. Given the high relevance of linear algebra and group theory to other subfields of computer science, QAL can also be adapted to offer derived future software that focuses on presenting workings on vectors and matrices as used in other areas of science and technology, such as machine learning algorithms [25], cybersecurity algorithms [39], [63], [1], robotics [43], ... etc.

Also, QAL has been tested informally in small-scale pilot experiments, where QAL was used to successfully teach essentials of quantum algorithms and linear algebra to few students of different ages and backgrounds (*i.e.*, school, undergraduate, and graduate students, of science, math, and engineering). Yet a careful larger-scale controlled user study of using QAL in pedagogy, and possibly in research, is planned as future work.

Finally, while QAL is intended to be as hardware-independent as possible, and is presented here purely as a conceptual mathematical and pedagogical tool that has no connection to actual quantum hardware, yet in an era where cloud access to real quantum processors is standard, QAL may offer in the future the possibility of compiling circuits to actual quantum backends. Also, how the noiseless/fault-tolerant algorithms QAL presents relate to the realities of noisy intermediate-scale quantum (NISQ) devices [50], [29] may be addressed in future versions of QAL.

VI. Conclusion

In this paper we introduced the Quantum Algorithms Lab (QAL) as an intuitive visual interactive tool that sits between a quantum simulator and a gamified learning environment. There are many quantum computing software simulators available online, commercially and for free. QAL is *not* one of them, and it should not be considered a quantum simulator (nor should it be compared extensively in depth to any quantum simulator), even when similarities do exist between QAL and some extant simulators. While most simulators allow interacting with quantum algorithms—with the ultimate goal being the discovery of efficient algorithms—they do so while assuming the mathematical concepts behind the user's experiments to be invisible (inside the head of the user) while experimenting (*i.e.*, the math is not presented intuitively in front of the user's eyes, with the option of using no mathematical text, notation, or formulas, like in a video game for example, which QAL, as a visual virtual lab, aims to offer). As such, to our knowledge, we know of the existence of no tool that is quite similar to QAL.

In particular, QAL is unique in it allowing *direct and simple* visual interaction with quantum algorithms at the level of *individual* eigen/observable states and gate components (*i.e.*, entries of matrix representation), and also doing so for arbitrary 'selections' (*i.e.*, user-specified groups of entries), not only allowing interaction at the higher level of qubits and gates (as most simulators do). Further, the main aim and goal of QAL is to *deepen intuitive mathematical understanding* of the properties and the internal structure of quantum algorithms (including properties of efficient ideal quantum algorithms, *i.e.*, ones running on ideal fault-tolerant gate-based quantum computers) via allowing the *playing with quantum gates* (*i.e.,* matrices) *and amplitudes* (*i.e.*, matrix/vector entries) **like pieces of Lego**[23] **or a Rubik's Cube**[24]. To our knowledge, there's no tool that allows experimenting with quantum algorithms in the same way as QAL does, nor using similar means as QAL does (*i.e.*, using its interactive visuals), nor as QAL aims to do in the future.

Given the immense power of visual interactive approaches in pedagogy and research, we *cautiously* but strongly believe that QAL offers a significantly different, new, and very useful approach to understanding properties of quantum operations (*e.g.*, their non-commutativity, their recursive structure, and their spectrum), and that QAL is thus suitably positioned

[22] Including Rubik's cube speedcubing algorithms, which are classical reversible/permutation algorithms, hinting to the fact that quantum operations—in a mathematical sense—are "natural" generalizations of permutations, going from binary 0/1 matrix entries (with permutation matrices, *i.e.*, matrices with one 1 in each row and each column, representing permutations) along the well-trodden "mathematical path" $\mathcal{B}$(binary nums) $\rightarrow\mathcal{N}$(naturals) $\rightarrow\mathcal{Z}$(integers) $\rightarrow\mathcal{Q}$(rationals) $\rightarrow\mathcal{R}$(reals) $\rightarrow\mathcal{C}$(complex nums) all the way to complex-valued matrix entries (with unitary matrices representing quantum operations).

This fact (*i.e.*, that quantum ops generalize permutations) is in fact partially responsible for QAL focusing on using a *polar* (modulus-magnitude/argument-phase) form of complex numbers rather than a rectangular (real/imaginary) one, with complex modulus/magnitude (usually denoted r) corresponding to "distance/strength," (in the range 0..1 for unitaries, a necessary but insufficient condition) and complex argument/phase (usually denoted θ) corresponding to "direction," *i.e.*, as a generalization from the "unidirectional" binary black/white of permutations to the continuous-valued but bounded multidirectional colors of quantum operations.

[23] "Playing with (*pieces* of) quantum algorithms," is the motto and slogan of QAL.

[24] Incidentally and interestingly, the set of moves of a Rubik's cube forms a large **non-commutative** group—of size over 43×10^{18} moves—that is a subgroup (of index 12) of the direct product of two (wreath) product groups, one for the cube's eight three-colored corners and the other for its twelve two-colored edges [9], [21]. Finding efficient and optimal algorithms for solving Rubik's cube (cube20.org), also known as "speed-cubing," has been also a motivation behind developing QAL, since speed-cubing typically involves a lot of (virtual and physical/real) experimentation with the cube, and of careful exploration of its "solutions space" (*i.e.*, the Rubik's cube group). For more details, see [38], for example, as well as much online Rubik's cube material and literature.

to help in teaching and researching quantum algorithms—potentially, given the strong mathematical orientation of QAL's interactive visuals, it helping to resolve hard quantum algorithm research problems such as the non-commutative hidden subgroup problem, and to particularly help in reaching a definite resolution of the quantum graph isomorphism problem.

References

[1] Moez A. Abdel-Gawad and Ahmed A. Belal. 2D-Encryption Mode. In *Mathematics of Data/Image Coding, Compression, and Encryption V, with Applications*, volume 4793 of *Proceedings of SPIE*, pages 211–220. SPIE, 2003. 8

[2] Moez AbdelGawad. Modeling object-oriented generics: A lattice- and category-theoretic approach (poster). http://eng.staff.alexu.edu.eg/staff/moez/research/OOP/gen-act19.pdf (sometimes inaccessible), July 2019. 6, 9

[3] Moez AbdelGawad. Towards an order and category theoretic model of Java generics (extended version). EasyChair Preprint 3631 (accepted for poster presentation at *Oxford University*, in ACT 2019 [2]), June 2020. 6

[4] Moez A. AbdelGawad. Towards a Java subtyping operad. *Proceedings of FTfJP'17, Barcelona, Spain (Extended version available at arXiv:cs.PL/1706.00274)*, 2017. 6

[5] Moez A. AbdelGawad. Java subtyping as an infinite self-similar partial graph product. *arXiv:cs.PL/1805.06893*, 2018. 6

[6] Moez A. AbdelGawad. Induction, coinduction, and fixed points: Intuitions and tutorial. *arXiv:cs.LO/1903.05127*, 2019. 6

[7] Moez A. AbdelGawad. Quantum Programming. http://eng.staff.alexu.edu.eg/staff/moez/teaching/pqc-f19/ (sometimes inaccessible), Fall 2019. 5, 6

[8] Martin Aigner. *Markov's Theorem and 100 Years of the Uniqueness Conjecture: A Mathematical Journey from Irrational Numbers to Perfect Matchings*. Springer International Publishing, Cham, Switzerland, 2013. 8

[9] Jr. Alexander H. Frey and David Singmaster. *Handbook of Cubik Math*. Enslow Publishers, Hillside, NJ, 1982. The classic text on the group theory of the Rubik's Cube. 8

[10] Claudi Alsina and Roger B. Nelsen. *Math Made Visual: Creating Images for Understanding Mathematics*. Classroom Resource Materials. Mathematical Association of America, 2006. 6

[11] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 684–697. ACM, 2016. 6

[12] David H. Bailey and Jonathan M. Borwein. *Exploratory Experimentation in Mathematics: Selected Works*. PSI Press, Portland, OR, 2012. 3

[13] David H. Bailey, Jonathan M. Borwein, Neil J. Calkin, Roland Girgensohn, D. Russell Luke, and Victor H. Moll. *Experimental Mathematics in Action*. A K Peters, Wellesley, MA, 2007. 3

[14] Ishan Shivansh Bangroo and Samia Amir. Quaver: Quantum unfoldment through visual engagement and storytelling resources. https://arxiv.org/abs/2309.11511, 2023. 6

[15] Chris Bernhardt. *Quantum Computing for Everyone*. MIT Press, Cambridge, MA, 2019. 8

[16] Robert Q. Berry III, Zachary Champagne, Randall I. Charles, Francis Fennell, Eric Milou, Jane F. Schielack, and Jonathan A. Wray. *enVision Mathematics 2024 Common Core Student Edition Grade 7*. Savvas Learning LLC, 2024. 6

[17] Jonathan M. Borwein and David H. Bailey. *Mathematics by Experiment: Plausible Reasoning in the 21st Century*. A K Peters/CRC Press, Wellesley, MA, 2nd edition, 2008. 3, 4

[18] Jonathan M. Borwein, David H. Bailey, and Roland Girgensohn. *Experimentation in Mathematics: Computational Paths to Discovery*. A K Peters, Natick, MA, 2004. 3

[19] Jonathan M. Borwein and Keith Devlin. *The Computer as Crucible: An Introduction to Experimental Mathematics*. A K Peters, Wellesley, MA, 2008. 3, 4

[20] Lukas Burgholzer and Robert Wille. Advanced equivalence checking for quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(9):1810–1824, 2021. 3

[21] Nathan Carter. *Visual Group Theory*. Mathematical Association of America, 2009. 6, 8

[22] Fan R. K. Chung. Spectral graph theory. https://fanchung.ucsd.edu/research/revised.html, 1997. 6

[23] Bob Coecke and Ross Duncan. ZX-calculus. zxcalculus.com, 2026. 6, 7

[24] Bob Coecke and Alex Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. 6, 7

[25] Romain Couillet and Zhenyu Liao. *Random Matrix Methods for Machine Learning*. Cambridge University Press, 2022. 8

[26] Soren Eilers and Rune Johansen. *Introduction to Experimental Mathematics*. Cambridge University Press, Cambridge, UK, 2017. 3

[27] A Ekert, T Hosgood, A Kay, and C Macchiavello. Introduction to Quantum Information Science. https://qubit.guide. 1, 6

[28] Artur Ekert. Introduction to quantum information science. https://youtube.com/ArturEkert, 2022. 6

[29] Kishor Bharti et al. Noisy intermediate scale quantum NISQ algorithms. *Reviews of Modern Physics*, 6, 2022. 8

[30] Chris Godsil and Gordon Royle. *Algebraic Graph Theory*, volume 207. Springer, 2001. 6

[31] Alex Gore and Lance Cayko. *The Creativity Code: The Power of Visual Thinking*. Alexander K. Gore, 2016. 1, 5

[32] Harald Helfgott, Jitendra Bajpai, and Daniele Dona. Graph isomorphisms in quasi-polynomial time. 2017. Translation from French with additions; main text by Helfgott. 6

[33] Jack D. Hidary. *Quantum Computing: An Applied Approach*. Springer-Verlag, 2019. 1, 2

[34] Cheng-Yen Hua, Shu-Yu Kuo, Yu-Chi Jiang, Ching-Hsuan Wu, and Yao-Hsin Chou. QSynViz: An interactive visualization tool for quantum circuit synthesis, optimization, and education. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 02, pages 556–557, 2025. 6

[35] V. Harizanov J. Chubb, A. Eskandarian, editor. *Logic and Algebraic Structures in Quantum Computing*. Cambridge University Press, 2016. 2

[36] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with Qiskit, 2024. 8

[37] Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia. *Programming Quantum Computers: Essential Algorithms and Code Samples*. O'Reilly, 2019. 6

[38] David Joyner. *Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys*. Johns Hopkins University Press, 2nd edition, 2008. 4, 6, 8

[39] Delaram Kahrobaei, Ramon Flores, Marialaura Noce, Maggie E. Habeeb, and Christopher Battarbee. *Applications of Group Theory in Cryptography: Post-Quantum Group-Based Cryptography*. Mathematical Surveys and Monographs. American Mathematical Society, Providence, RI, 2024. 8

[40] F William Lawvere and Stephen H Schanuel. *Conceptual mathematics: a first introduction to categories*. Cambridge University Press, 2009. 6

[41] Jianqiang Li. A new quantum system algorithm beyond the condition number and its application to solving multivariate polynomial systems. https://arxiv.org/abs/2510.05588, Feb. 2026. 6

[42] Richard J. Lipton and Kenneth W. Regan. *Introduction to Quantum Algorithms via Linear Algebra*. MIT Press, 2nd edition, 2021. 2, 4, 6, 8

[43] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, Cambridge, 2017. 8

[44] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 367–381, New York, NY, USA, 2020. Association for Computing Machinery. 3

[45] Piet Van Mieghem. *Graph Spectra for Complex Networks*. Cambridge University Press, 2011. 6

[46] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction*, pages 625–635. Springer, 2021. 3

[47] Jamie Mulholland. *Permutation Puzzles: A Mathematical Perspective*. Self-Published, 2021. 4, 8

[48] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, 10th anniversary edition, 2010. 1, 2, 6

[49] Dan Pedoe. *Geometry and the Visual Arts*. Dover Publications, New York, 1983. Reprint. Originally published: Geometry and the liberal arts. New York : St. Martin's Press, c1976. 6

[50] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2, 2018. 8
[51] IBM Quantum. Quantum composer. https://quantum.cloud.ibm.com/composer, 2026. Accessed: 2026-04-19. 6
[52] V. Raseena. Quantum computing: foundations, algorithms, and emerging applications. *Frontiers in Quantum Science and Technology*, 4, 2025. 6
[53] Matthew P. Richey and Matthew L. Wright. *Experimental Mathematics: A Computational Perspective*, volume 74 of *AMS/MAA Textbooks*. American Mathematical Society, Providence, RI, 2025. 3
[54] Kenneth A. Ross and Charles R. B. Wright. *Discrete Mathematics*. Prentice Hall, fifth edition, 2003. 4
[55] Shaolun Ruan, Qiang Guan, Paul Griffin, Ying Mao, and Yong Wang. QuantumEyes: Towards better interpretability of quantum circuits. *IEEE Transactions on Visualization and Computer Graphics*, 30, 2024. 6
[56] Shaolun Ruan, Ribo Yuan, Qiang Guan, Yanna Lin, Ying Mao, Weiwen Jiang, Zhepeng Wang, Wei Xu, and Yong Wang. Venus: A geometrical representation for quantum state visualization. https://arxiv.org/abs/2303.08366, 2023. 6
[57] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. 1, 6
[58] David Spivak. The operad of wiring diagrams: Formalizing a graphical language for databases, recursion, and plug-and-play circuits. *eprint available at http://arxiv.org/abs/1305.0297*, 2013. 6
[59] David Spivak. *Category theory for the sciences*. MIT Press, 2014. 6
[60] Colin Stuart and Ximo Abadía. *The Language of the Universe: A Visual Exploration of Mathematics*. Big Picture Press, Somerville, MA, 2020. Illustrated by Ximo Abadía. 6
[61] Cirq Developers Team. Cirq, 2021. 8
[62] Bernd Thaller. *Visual Quantum Mechanics: Selected Topics with Computer-Generated Animations of Quantum-Mechanical Phenomena*. Springer/TELOS, New York, 2000. Includes CD-ROM. 6
[63] Maria Isabel Gonzalez Vasco and Rainer Steinwandt. *Group Theoretic Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. Chapman & Hall/CRC, Boca Raton, FL, 2024. 8
[64] Alexis De Vos. *Reversible Computing: Fundamentals, Quantum Computing, and Applications*. WILEY-VCH Verlag GmbH, 2010. 8
[65] Shuling Wang, Mingsheng Ying, Naijun Yu, and Yuan Feng. A theorem prover for quantum Hoare logic and its applications. 2016. 3
[66] Zhen Wen, Yihan Liu, Siwei Tan, Jieyi Chen, Minfeng Zhu, Dongming Han, Jianwei Yin, Mingliang Xu, and Wei Chen. Quantivine: A visualization approach for large-scale quantum circuit representation and analysis. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–11, 2023. 6
[67] Colin P. Williams. *Explorations in Quantum Computing*. Springer-Verlag, 2nd edition, 2011. 2, 5, 6

APPENDIX

QAL LITE SCREENSHOTS

The following figures (on pages 11 to 12) show QAL/QAL Lite screenshots that illustrate the description and use of QAL as presented in Sec. II.

For consistency, the 'Add to Custom Gates' feature of QAL (available also in QAL Lite, as *all* QAL features other than QA-LA NIVs do) was made use of in illustrating Example 2 (*i.e.*, in Fig 5 and Fig 6) to have the *same* random matrix R used in visually confirming both facts of the example.

For more illustrative screenshots of QAL Lite (and blurry ones of QAL), see the QAL Lite Screenshots document available online at https://q-info.github.io/QAL-Lite/doc/.

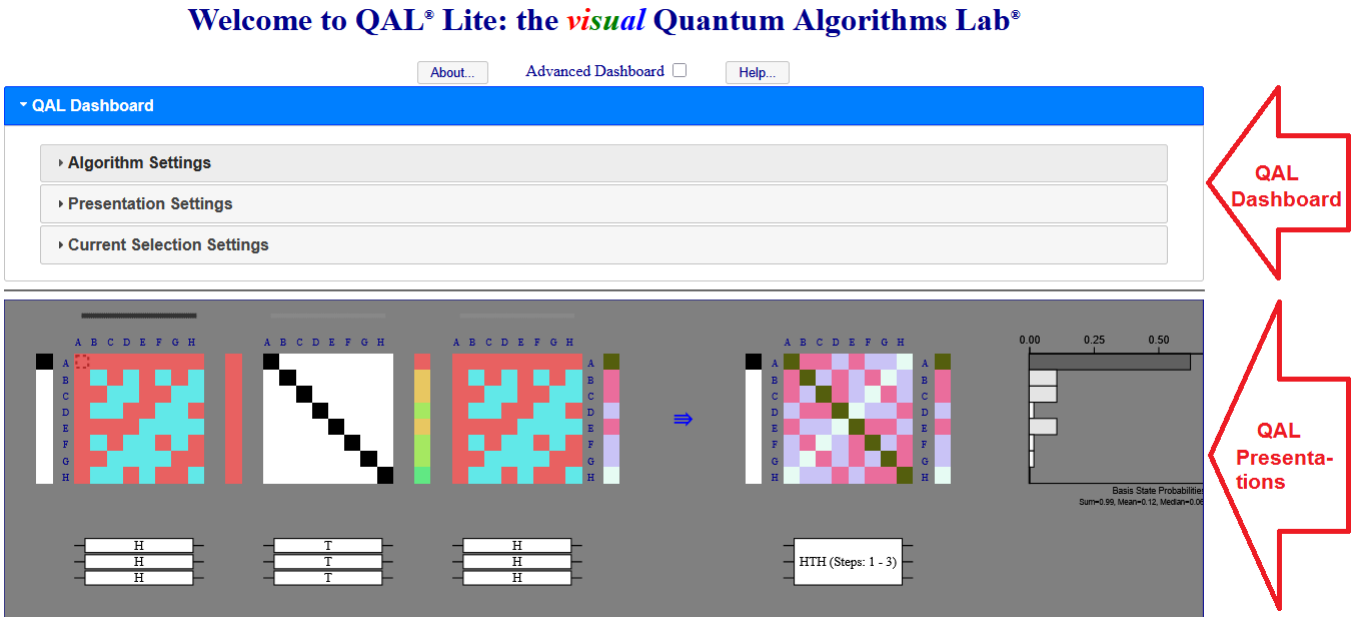

Figure 2. Initial screen of QAL, with a folded QAL Dashboard and two algorithm presentations (heatmap and circuit diagram presentations) of a simple 3-steps quantum algorithm. The *result* matrix, representing the *overall operation* of the algorithm, is presented in the *result heatmap* to the right of the blue ⇛ arrow (with the input/initial state vector presented on its left, and the output/final state vector on its right). To the right of the result heatmap is the *final state histogram*.

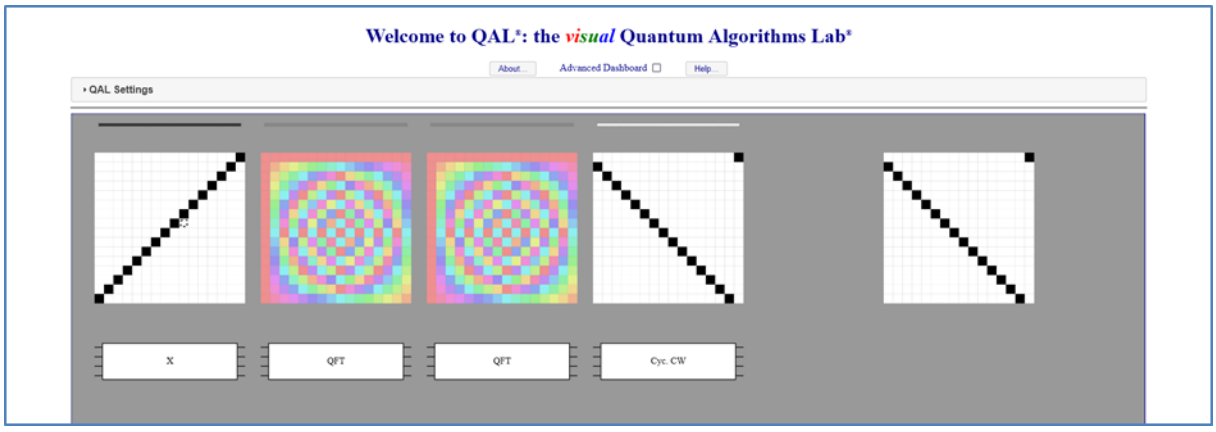

Figure 3. Visual confirmation that 'X then QFT^2' is "successor" (=Cycle/Rotate Clockwise). The result heatmap (for the $QFT^2 \circ X$ *subalgorithm*, presented in algorithm steps 1-3) is *the same* as the heatmap of step 4 (for the successor operation; included for comparison).

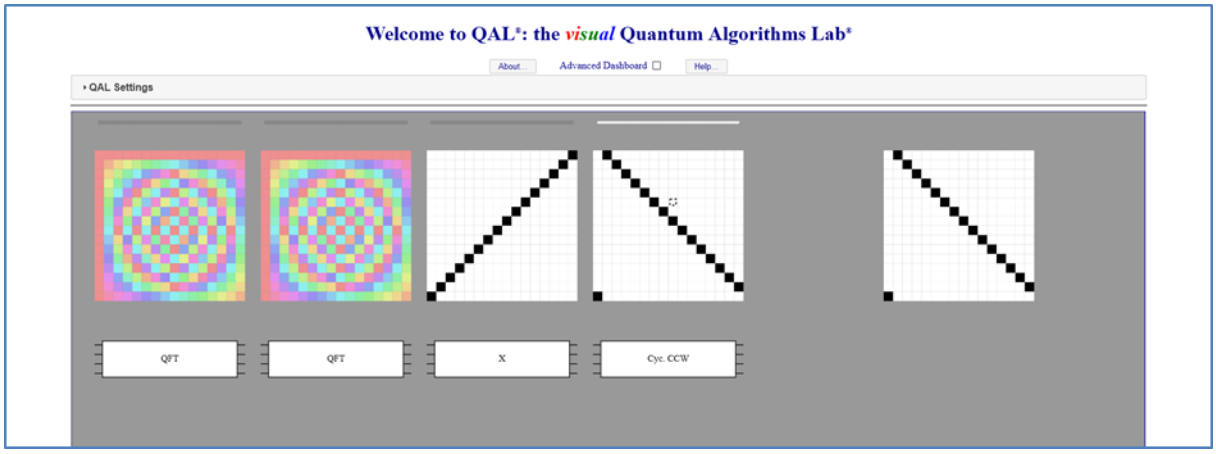

Figure 4. Visual confirmation that 'QFT^2 then X' is "predecessor" (=Cycle/Rotate Counter Clockwise). The result heatmap (for the $X \circ QFT^2$ *subalgorithm*, presented in algorithm steps 1-3) is *the same* as the heatmap of step 4 (for the predecessor operation; included for comparison).

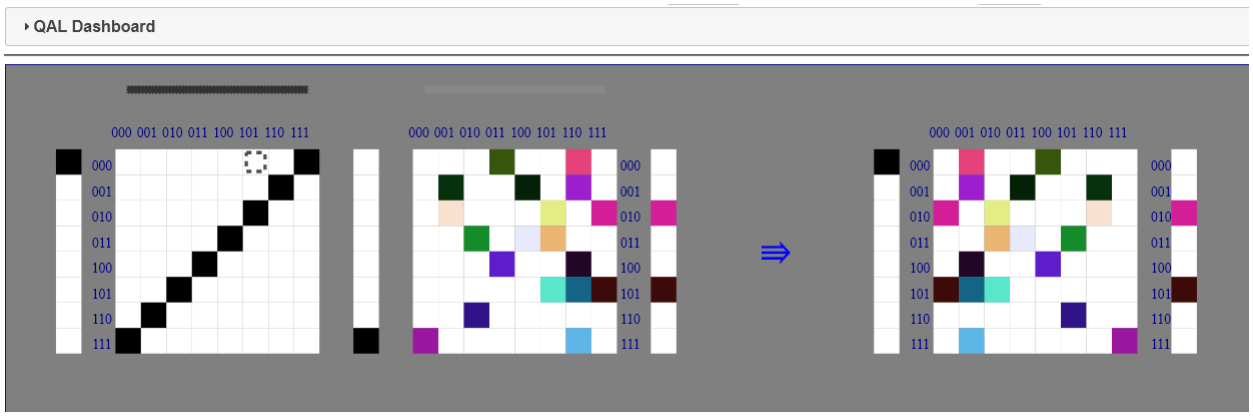

Figure 5. Visual confirmation that 'X then R' (*i.e.*, RX, the post/right-multiplication of R by X) flips columns of R. The result heatmap is the left-to-right flipping (along a vertical axis) of R (presented in step 2's heatmap).

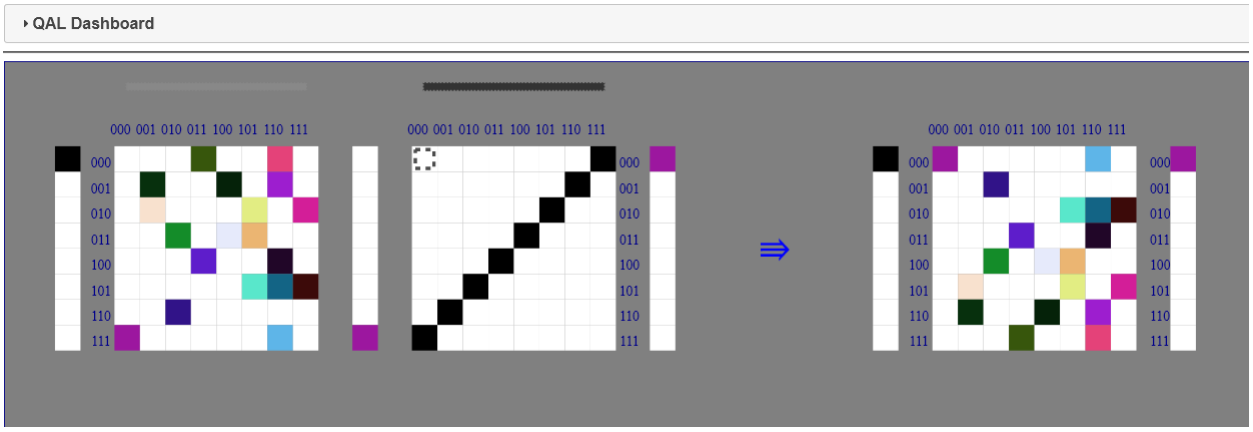

Figure 6. Visual confirmation that 'R then X' (pre/left-multiplication by X) flips rows of R. The result heatmap is the upside-down flipping (along a horizontal axis) of R (presented in step 1's heatmap).