

A General Framework for Gradient-Based Optimization of Superconducting Quantum Circuits using Qubit Discovery as a Case Study

Taha Rajabzadeh¹, Alex Boulton-McKeehan^{*2}, Sam Bonkowsky^{*3}, David I. Schuster², and Amir H. Safavi-Naeini²

¹Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA

²E. L. Ginzton Laboratory and the Department of Applied Physics, Stanford University, Stanford, CA 94305 USA

³Department of Physics, Stanford University, Stanford, CA 94305 USA

Engineering the Hamiltonian of a quantum system is fundamental to achieving novel goals or better performance. Automating Hamiltonian design through gradient-based optimization can dramatically accelerate this process. However, computing the gradients of eigenvalues and eigenvectors of the Hamiltonian—a large, sparse matrix—with respect to system properties poses a significant challenge, especially for arbitrary systems. Superconducting quantum circuits offer substantial flexibility in Hamiltonian design, making them an ideal platform for this task. In this work, we present a comprehensive framework for the gradient-based optimization of superconducting quantum circuits, leveraging the SQcircuit software package. By addressing the challenge of calculating the gradient of the eigensystem for large, sparse Hamiltonians and integrating automatic differentiation within SQcircuit, our framework enables efficient and precise computation of gradients for various circuit properties and custom metrics, streamlining the optimization process. We apply this framework to discover qubit designs with superior performance metrics. The optimized circuits show improvements in a heuristic measure of gate count, upper bounds on gate speed, decoherence time, and resilience to noise and fabrication errors compared to existing qubits. While this methodology is showcased through qubit optimization and discovery, it is versatile and can be extended to tackle other optimization challenges in superconducting quantum hardware design.

Taha Rajabzadeh: tahar@stanford.edu

^{*}A.B. and S.B. contributed equally to this work.

Contents

1	Introduction	2
2	Superconducting Quantum Circuit Analysis Review	3
3	Circuit Optimization and Gradient Calculation	4
4	Qubit Discovery	6
4.1	Practical Qubit Design	7
4.2	Search Problem as an Optimization . . .	9
4.3	Optimization Pipeline	13
4.4	Discovered Qubits	13
5	Outlook	15
6	Online Presence	16
7	Conclusion	16
	Author Contributions	16
	Acknowledgments	17
A	PyTorch Computational Graph	17
B	SQcircuit Tutorial	18
C	Dephasing Time Gradient	19
D	Upper Bound on Gate Speed	20
E	Diagonalization Convergence Test	21
F	Automatic Truncation Assignment	21
	References	24

1 Introduction

Superconducting quantum circuits are a leading hardware platform for realizing quantum computing in a laboratory environment, and are strong candidates for achieving fault-tolerant quantum computation. Consequently, there has been tremendous effort in this field to utilize superconducting circuits to create qubits with long coherence times [1, 2], build local couplers to enable efficient qubit interaction [3, 4], and develop hybrid systems for quantum sensing and memory applications [5, 6]. Additionally, significant efforts are focused on studying quantum error correction [7–10] and autonomous quantum error correction [11, 12] in these hardware platforms.

These applications, among many others, involve increasing complexities in circuit design and require a comprehensive approach for analyzing larger and more intricate circuits. Fundamental studies [13–21] have led to the development of open-source software packages [13, 22–25] for analyzing the Hamiltonian of superconducting quantum circuits. In this work, we use and extend the `SQcircuit` Python package [13], which is able to efficiently analyze arbitrary circuits by employing an advantageous coordinate transformation.

As is often the case with quantum systems, as circuits become larger, it becomes not only harder to simulate them numerically but also more challenging to build intuition. Although existing packages have made the task of analysis easier, the design process, even for a relatively simple qubit such as the transmon, requires consideration of many parameters and constraints. This indicates a need to automate the design process of superconducting circuits via optimization over superconducting quantum hardware.

There has been significant progress in the field, including remarkable work applying optimization and machine learning techniques to superconducting quantum hardware design. For example, [26] applied an evolutionary algorithm to design a 4-local coupler for flux qubits, and [27] used gradient-based joint optimization to discover fluxonium qubits with better gate properties. However, applying these methodologies to new problems often requires starting from scratch, as the obstacles and constraints can change with each new problem. Therefore, there is a need for a universal approach that can be applied to arbitrary superconducting quantum circuits.

We construct an optimization framework for arbitrary superconducting quantum hardware. Based on mature approaches to gradient-based optimization employed in the machine learning community, we establish a similar method with a user-friendly interface in the `SQcircuit` package. To demonstrate its applications,

we use it to automatically discover circuits which perform well as single qubits. This approach can be easily extended to other problems in superconducting quantum hardware, due to the ability to optimize for virtually any differentiable loss function constructed from element values and the eigenvalues and eigenvectors of the circuit Hamiltonian.

Our primary contributions from this work are as follows:

- i. Extended `SQcircuit` with automatic differentiation to compute the gradient of arbitrary metrics with respect to circuit element values.
- ii. Developed a robust optimization pipeline for superconducting quantum circuit discovery using BFGS optimization, requiring only the definition of an appropriate loss function for various optimization problems.
- iii. Implemented automatic assignment of Hilbert space truncation numbers for each circuit mode, along with diagonalization convergence tests, to ensure thorough and reliable exploration of the search space. This approach addresses significant bottlenecks commonly encountered in the search process.
- iv. Introduced metrics to compare distinct qubits based on their laboratory implementability and total gate count. Leveraged our pipeline to optimize over this objective and thoroughly explore the space of single-loop circuit topologies for candidate qubits.

The remainder of this manuscript is structured as follows: In Section 2, we review the superconducting quantum circuit analysis and coordinate transformations used by `SQcircuit`, establishing the theoretical foundation for subsequent sections. In Section 3, we formulate the general optimization problem for superconducting circuits, emphasizing the importance of gradient-based optimization. We then explain the theory behind the automatic differentiation implemented in `SQcircuit`. In Section 4, we focus on the qubit discovery problem as a demonstration of automatic differentiation and the optimization pipeline, which easily extends to other discovery problems. We also discuss the metrics we use to frame qubit discovery as a search problem and provide a detailed analysis of the optimization results and the discovered qubits. Finally, we present our conclusions and an outlook in Section 7.

2 Superconducting Quantum Circuit Analysis Review

In this section, we present the analysis of an arbitrary superconducting quantum circuit constructed with lumped elements, as described in [13]. The Hamiltonian of a circuit with $n_N + 1$ nodes (n_N degrees of freedom) and n_L independent inductive loops (number of constraints) is described by

$$\hat{H} = \frac{1}{2} \hat{\mathbf{Q}}^T \mathbf{C}^{-1} \hat{\mathbf{Q}} + \frac{1}{2} \hat{\mathbf{\Phi}}^T \mathbf{L}^* \hat{\mathbf{\Phi}} + \sum_{k \in \mathcal{S}_L} \left(\frac{\Phi_0}{2\pi} \frac{\mathbf{b}_k^T \boldsymbol{\varphi}_{\text{ext}}}{l_k} \right) \mathbf{w}_k^T \hat{\mathbf{\Phi}} - \sum_{k \in \mathcal{S}_J} E_{J_k} \cos \left(\frac{2\pi}{\Phi_0} \mathbf{w}_k^T \hat{\mathbf{\Phi}} + \mathbf{b}_k^T \boldsymbol{\varphi}_{\text{ext}} \right), \quad (1)$$

where:

- $\hat{\mathbf{Q}} = [\hat{Q}_1, \dots, \hat{Q}_{n_N}]$ is an n_N -dimensional vector containing charge operators, and $\hat{\mathbf{\Phi}} = [\hat{\Phi}_1, \dots, \hat{\Phi}_{n_N}]$ is an n_N -dimensional vector containing flux operators. The operators satisfy the canonical commutation relation $[\hat{\mathbf{\Phi}}, \hat{\mathbf{Q}}] = i\hbar \mathbf{1}$.
- $\mathbf{C}$ and $\mathbf{L}^*$ are $n_N \times n_N$ capacitance and susceptance matrices, respectively. Diagonal elements are the sum of capacitances and susceptances connected to a node, and off-diagonal elements are the negative values of capacitance and susceptance between nodes.
- E_{J_k} is the junction energy of the Josephson junction at branch k .
- $\mathcal{S}_L$ and $\mathcal{S}_J$ are the sets of branches which contain inductors and Josephson junctions, respectively.
- $\mathbf{w}_k$ is an n_N -dimensional vector; for branch k between nodes i and j , the m th element is $\delta_{im} - \delta_{jm}$ for element $m \in \{1, 2, \dots, n_N\}$.
- $\mathbf{b}_k$ is an n_L -dimensional vector where the l th element is 1 if the k th branch completes loop l , and 0 otherwise.
- $\boldsymbol{\varphi}_{\text{ext}}$ is an n_L -dimensional vector that is related to external fluxes via $\boldsymbol{\varphi}_{\text{ext}} = 2\pi \mathbf{\Phi}_{\text{ext}} / \Phi_0$. Here, $\mathbf{\Phi}_{\text{ext}} = [\phi_{\text{ext},1}, \dots, \phi_{\text{ext},n_L}]$ is a vector of the external fluxes applied to each independent inductive loop in the circuit.

Quantum analysis of superconducting circuits becomes computationally intensive as the circuit size increases. Efficient numerical simulation necessitates coordinate transformations that output a sparse Hamiltonian and lead to rapidly decaying off-diagonal matrix elements. To meet these criteria, we transform the Hamiltonian of

Equation (1) by performing a canonical transformation of charge and flux operators:

$$\hat{\hat{\mathbf{Q}}} = \mathbf{R}^{-1} \hat{\mathbf{Q}}, \quad \hat{\hat{\mathbf{\Phi}}} = \mathbf{S}^{-1} \hat{\mathbf{\Phi}}. \quad (2)$$

The matrices $\mathbf{R}$ and $\mathbf{S}$, which are $n_N \times n_N$ real invertible matrices, are used in a canonical transformation such that the transformed charge and flux operators, $\hat{\hat{\phi}}_m$ and $\hat{\hat{Q}}_n$, satisfy the commutation relation $[\hat{\hat{\phi}}_m, \hat{\hat{Q}}_n] = \delta_{mn} i\hbar$. This necessitates the constraint of $\mathbf{S}^T = \mathbf{R}^{-1}$. The transformed Hamiltonian then takes the form:

$$\hat{\hat{H}} = \frac{1}{2} \hat{\hat{\mathbf{Q}}}^T \tilde{\mathbf{C}}^{-1} \hat{\hat{\mathbf{Q}}} + \frac{1}{2} \hat{\hat{\mathbf{\Phi}}}^T \tilde{\mathbf{L}}^* \hat{\hat{\mathbf{\Phi}}} + \sum_{k \in \mathcal{S}_L} \left(\frac{\Phi_0}{2\pi} \frac{\tilde{\mathbf{b}}_k^T \boldsymbol{\varphi}_{\text{ext}}}{l_k} \right) \tilde{\mathbf{w}}_k^T \hat{\hat{\mathbf{\Phi}}} - \sum_{k \in \mathcal{S}_J} E_{J_k} \cos \left(\frac{2\pi}{\Phi_0} \tilde{\mathbf{w}}_k^T \hat{\hat{\mathbf{\Phi}}} + \tilde{\mathbf{b}}_k^T \boldsymbol{\varphi}_{\text{ext}} \right).$$

The corresponding matrices and vector transformations are:

$$\tilde{\mathbf{C}}^{-1} = \mathbf{R}^T \mathbf{C}^{-1} \mathbf{R}, \quad \tilde{\mathbf{L}} = \mathbf{S}^T \mathbf{L} \mathbf{S}, \quad \tilde{\mathbf{w}}_k^T = \mathbf{w}_k^T \mathbf{S}.$$

As detailed in Appendix B of [13], matrices $\mathbf{S}$ and $\mathbf{R}$ can be chosen to arrange $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{C}}$ into a block diagonal structure shown below:

$$\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C}^{\text{ha}} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}^{\text{ch}} \end{bmatrix}, \quad \tilde{\mathbf{L}} = \begin{bmatrix} \mathbf{L}^{*\text{ha}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix},$$

where $\mathbf{C}^{\text{ha}}$ and $\mathbf{L}^{*\text{ha}}$ are $n_H \times n_H$ invertible diagonal matrices termed the “harmonic” components, and $\mathbf{C}^{\text{ch}}$ is a $n_C \times n_C$ symmetric matrix (not necessarily diagonal), termed the “charge” component. This leads to a division of the transformed charge and flux operators into harmonic and charge contributions:

$$\hat{\hat{\mathbf{Q}}}^T = [\hat{Q}_1^{\text{ha}}, \dots, \hat{Q}_{n_H}^{\text{ha}} | \hat{Q}_1^{\text{ch}}, \dots, \hat{Q}_{n_C}^{\text{ch}}], \\ \hat{\hat{\mathbf{\Phi}}}^T = [\hat{\phi}_1^{\text{ha}}, \dots, \hat{\phi}_{n_H}^{\text{ha}} | \hat{\phi}_1^{\text{ch}}, \dots, \hat{\phi}_{n_C}^{\text{ch}}].$$

The above transformation describes the system’s dynamics in terms of (1) n_H uncoupled harmonic oscillators with their dynamics defined by the diagonal elements of $\mathbf{C}^{\text{ha}}$ and $\mathbf{L}^{*\text{ha}}$, (2) n_C isolated superconducting islands with charging energy described by $\mathbf{C}^{\text{ch}}$, and (3) a flux distribution across various junctions determined by the vector $\tilde{\mathbf{w}}_k^T$. In representing the Hamiltonian, we then select the Fock basis for harmonic modes and the charge basis for charge modes and diagonalize the Hamiltonian in these respective bases. This entire process is automated by the **SQcircuit** software, which is described in more detail in [13].

Table 1: Derivatives of the Hamiltonian in original and computational bases for various circuit parameters. These derivatives are computed by the custom computational node integrated into the PyTorch engine of SQcircuit.

Hamiltonian Derivative	Original Basis	Computational Basis
$\partial \hat{H} / \partial c_j$	$-\frac{1}{2} \hat{Q}^T C^{-1} \frac{\partial C}{\partial c_j} C^{-1} \hat{Q}$	$-\frac{1}{2} \hat{Q}^T R^T C^{-1} \frac{\partial C}{\partial c_j} C^{-1} R \hat{Q}$
$\partial \hat{H} / \partial l_j$	$\frac{1}{2} \hat{\Phi}^T \frac{\partial L^*}{\partial l_j} \hat{\Phi}$	$\frac{1}{2} \hat{\Phi}^T S^T \frac{\partial L^*}{\partial l_j} S \hat{\Phi}$
$\partial \hat{H} / \partial E_{J_j}$	$-\cos\left(\frac{2\pi}{\Phi_0} \mathbf{w}_j^T \hat{\Phi} + \mathbf{b}_j^T \varphi_{\text{ext}}\right)$	$-\cos\left(\frac{2\pi}{\Phi_0} \tilde{\mathbf{w}}_j^T \hat{\Phi} + \mathbf{b}_j^T \varphi_{\text{ext}}\right)$
$\partial \hat{H} / \partial n_{g_j}$	$\mathbf{e}_j^T C^{-1} \hat{Q}^a$	$\mathbf{e}_j^T C^{-1} R \hat{Q}$
$\partial \hat{H} / \partial \varphi_{\text{ext}_j}$	$\sum_{k \in \mathcal{S}_J} E_{J_k} b_{kj} \sin\left(\frac{2\pi}{\Phi_0} \mathbf{w}_k^T \hat{\Phi} + \mathbf{b}_k^T \varphi_{\text{ext}}\right)^b$	$\sum_{k \in \mathcal{S}_J} E_{J_k} b_{kj} \sin\left(\frac{2\pi}{\Phi_0} \tilde{\mathbf{w}}_k^T \hat{\Phi} + \mathbf{b}_k^T \varphi_{\text{ext}}\right)$

^aHere, $\mathbf{e}_j$ is an n_N -dimensional vector with its j -th element equal to 1 and all other elements equal to 0.

^bWe assume that all external fluxes are assigned to junctions, which implies that $\mathcal{S}_L$ is empty. This assumption holds true in the absence of a time-dependent Hamiltonian [13, 28, 29].

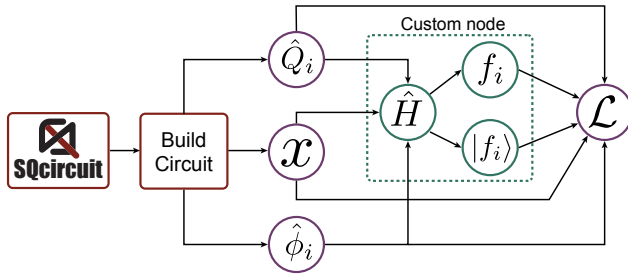


Figure 1: **Computational graph in SQcircuit.** This diagram illustrates the integration of a custom computational node into the SQcircuit framework, facilitating automatic differentiation of circuit properties. The process begins with construction of the circuit using SQcircuit. The charge operators $\hat{Q}_i$, flux operators $\hat{\Phi}_i$, and circuit elements $\mathbf{x}$ are implemented in PyTorch (represented above as purple nodes). Subsequently, eigenfrequencies f_i and eigenvectors $|f_i\rangle$ are computed within a custom node (depicted in green). These results are then fed into the loss function $\mathcal{L}$, as defined in Equation (3). This custom node performs the computation of gradients with respect to circuit elements $\mathbf{x}$, enabling the complete optimization process for superconducting circuit design within the PyTorch engine.

3 Circuit Optimization and Gradient Calculation

The optimization of superconducting quantum hardware systems can be framed in terms of minimizing a loss function, denoted as $\mathcal{L}$. This function quantifies the discrepancy between the properties of a simulated circuit and those of an ideal target circuit. A lower value of $\mathcal{L}$ indicates closer alignment with the desired circuit characteristics. Within this work, we consider loss functions of the form:

$$\mathcal{L} = \mathcal{L}\left(\hat{Q}_i, \hat{\Phi}_i, f_i, |f_i\rangle, \mathbf{x}\right), \quad (3)$$

where $\hat{Q}_i$ and $\hat{\Phi}_i$ represent the charge and flux operators at node i , respectively; f_i and $|f_i\rangle$ denote the eigenfrequency and eigenvector of the i^{th} mode, respectively; and $\mathbf{x}$ denotes a concatenated vector of all lumped circuit elements (capacitors, inductors, and junctions) and externally tunable parameters (gate charges n_g and external fluxes φ).

Optimization of the objective then equates to finding the vector $\mathbf{x}$ for a given circuit topology that minimizes $\mathcal{L}$, constrained by the physical and fabrication limitations of the circuit element values. This can be formulated mathematically as:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathcal{L}\left(\hat{Q}_i, \hat{\Phi}_i, f_i, |f_i\rangle, \mathbf{x}\right) \\ \text{s.t.} \quad & \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}, \end{aligned} \quad (4)$$

where $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$ are the lower and upper bounds on the circuit elements, respectively. These bounds are dictated by the fabrication capabilities and physical constraints of the circuit components, which are discussed in greater detail in Section 4.

Importance of Gradient in Optimization. Loss functions of the form (4) are not necessarily convex, so finding global optimum is difficult. However, it is feasible to find satisfactory local optima through the following two general approaches:

1. First- or second-order optimization methods; and
2. Zero-order optimization methods.

First-order and second-order optimization methods, such as gradient descent, Adam [30], quasi-Newton [31, 32], and Newton’s method utilize first-order derivatives $\partial\mathcal{L}/\partial\mathbf{x}$ and second-order derivatives $\partial^2\mathcal{L}/\partial x_i\partial x_j$ (the Hessian matrix) to solve the minimization problem. These algorithms are efficient in converging to local optima of the loss function but do not specifically target global optima. They are more suited for exploitation (efficient convergence to local optima) rather than exploration (sacrifice of computational resources to seek out better local optima).

In contrast, zero-order optimization methods (also known as black-box optimization) such as particle swarm optimization (PSO) [33], Bayesian optimization [34], reinforcement learning [35–37], and evolutionary strategies [38] attempt to solve the optimization problem without information about derivatives. These algorithms are advantageous for global optimum searches, albeit slower compared to first-order and second-order methods. They are more suitable for exploration, but require substantial computational resources to yield satisfactory results.

Generally, zero-order optimization is employed in scenarios where gradient calculations are either costly or impractical. However, supplementation of these algorithms with the local gradient can enhance their performance and balance exploration and exploitation more effectively. For instance, references [39] and [40] demonstrate improved performance in PSO and Bayesian optimization via incorporation of gradient information.

In conclusion, gradient information is highly valuable when it can be accurately computed or approximated, significantly enhancing the effectiveness of optimization algorithms in achieving more precise results or leading to simple and effective optimization when applied towards gradient-based techniques on their own. Within this work, we demonstrate how gradient-based optimization can be implemented straightforwardly and applied to achieve satisfactory results on its own.

Automatic Differentiation. Our goal is to compute the gradient of an arbitrary loss function with respect to individual circuit parameters and elements. Analytically

evaluating the gradient of the loss function is laborious and must be recalculated whenever the target problem changes, making manual computation impractical. However, the machine learning community has invested significant effort into automating the differentiation and gradient computation processes for diverse loss functions, originally designed for development of AI models. This need has led to the development of comprehensive automatic differentiation packages such as `PyTorch` [41], `JAX` [42], and `TensorFlow` [43]. For instance, `PyTorch` calculates gradients using a ‘computational graph,’ a network of nodes that represent variables and operations defining a function. The ‘forward pass’ involves execution of these operations, applying a series of computational operations to input to produce output. Each operation yields intermediate results, forming a graph that traces the progression from input to output. During the ‘backward pass,’ `PyTorch` traverses this graph in reverse, from outputs to inputs, applying the chain rule to efficiently propagate gradients backwards and facilitate input parameter optimization. The details of this gradient computation in `PyTorch` are illustrated in Appendix A.

In this work, we have integrated the `PyTorch` engine into the `SQcircuit` package, while maintaining backward compatibility with prior versions [13]. This integration allows `SQcircuit`, in conjunction with `PyTorch`, to compute derivatives of any arbitrary loss function of the form (3) with respect to circuit parameters by applying the following chain rule:

$$\frac{d\mathcal{L}}{d\mathbf{x}} = \frac{\partial\mathcal{L}}{\partial\mathbf{x}} + \sum_i \frac{\partial f_i}{\partial\mathbf{x}} \frac{\partial\mathcal{L}}{\partial f_i} + \sum_i \frac{\partial |f_i\rangle}{\partial\mathbf{x}} \frac{\partial\mathcal{L}}{\partial |f_i\rangle}. \quad (5)$$

Note that the charge and flux operators, represented as $\hat{Q}_i$ and $\hat{\Phi}_i$, do not depend on the circuit elements, hence $\partial\hat{Q}_i/\partial\mathbf{x} = \partial\hat{\Phi}_i/\partial\mathbf{x} = 0$. However, we cannot directly implement (5) using `PyTorch`’s internal functionalities since the gradients of eigenvalues and eigenvectors for sparse matrices are not currently supported. This feature is crucial for `SQcircuit`, which computes the Hamiltonian in a highly sparse basis to perform faster diagonalization.

To address this limitation, we introduce a custom node into the computational graph (shown in Figure 1) that calculates the gradients of the eigenvalues and eigenvectors independently, and integrates these calculations with the `PyTorch` engine. This is achieved using perturbation theory, which approximates the derivatives of the eigenvalues and eigenvectors with respect to

a parameter x as follows:

$$\frac{\partial f_i}{\partial x} = \langle f_i | \frac{\partial \hat{H}}{\partial x} | f_i \rangle, \quad (6)$$

$$\frac{\partial |f_i\rangle}{\partial x} = \sum_{m \neq i}^{n_E} \frac{\langle f_m | \frac{\partial \hat{H}}{\partial x} | f_i \rangle}{f_i - f_m} |f_m\rangle, \quad (7)$$

where $\hat{H}$ is the Hamiltonian defined in (1), n_E specifies the number of eigenvalues and eigenvectors calculated in the diagonalization process, and x represents some circuit element or property such as E_{J_j} (the junction at the j th branch), c_j (the capacitor at the j th branch), l_j (the inductor at the j th branch), n_{g_j} (the gate charge number at the j th charge island), or $\varphi_{\text{ext}j}$ (the external flux through the j th inductive loop).

It is important to note that the user cannot choose an arbitrarily large value for n_E , because the computational cost of the circuit diagonalization is proportional to n_E . However, n_E must be sufficiently large to ensure adequate accuracy in the eigenvector gradient calculation. Calculating the gradient of the eigenvalues and eigenvectors thus requires determining $\partial \hat{H} / \partial x$ based on (1). However, as **SQcircuit** represents the Hamiltonian and operators in the charge and harmonic computational bases, it is necessary to transform these operators from the original bases to the computational bases using the transformations defined in (2). Gradients of the Hamiltonian in both the original and computational bases are summarized in Table 1.

We provide an example of using **SQcircuit** to calculate these gradients in Appendix B, showcasing the implementation of the computational graph in Figure 1.

In certain scenarios, it is necessary to compute the gradients of eigenfrequencies up to the second order. This requirement arises during the calculation of dephasing rates or T_φ times, as they are themselves proportional to the gradient of the eigenfrequencies, as detailed in [13]. Consequently, these gradients are dependent on the second-order gradients of the eigenfrequencies. Due to the utilization of a custom calculation node within **PyTorch**, the second-order gradients of the eigenfrequencies are not computed automatically. We address this limitation by implementing a separate custom node that combines analytical expressions for second-order Hamiltonian derivatives with first-order automatic differentiation, as detailed in Appendix C.

4 Qubit Discovery

The automatic differentiation capability of **SQcircuit**, discussed in Section 3, facilitates the study of many intriguing physical problems in superconducting circuit design that would be challenging to analyze without an

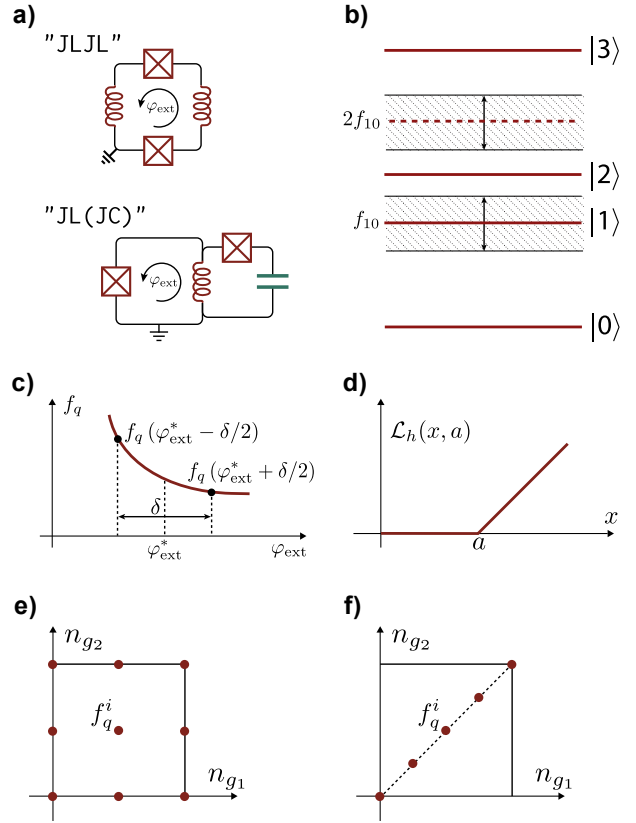


Figure 2: Illustration of various qubit properties and metrics. **a)** The circuit topology for the circuit codes "JLJL" and "JL(JC)" is presented. This circuit also includes an all-to-all capacitive connection; for the sake of simplicity, we omit these in the diagram. **b)** Energy-level diagram showing the transition frequencies between different qubit states. The dashed line represents twice the fundamental frequency $2f_{10}$, highlighting the transition caused by driving the qubit frequency f_{10} . The hatched box around these frequencies illustrates the sidebands, which are dependent on the gate speed. The edges of the box must not intersect with other states to prevent unwanted transitions from the qubit state to higher energy levels. Consequently, the maximum size of the hatched box defines the upper bound for the gate speed $\mathcal{G}$. **c)** Flux sensitivity $\mathcal{S}_\varphi$ as a function of external flux φ_{ext} , demonstrating the effect of flux variations on qubit frequency f_q . **d)** Hinge loss function $\mathcal{L}_h(x, a)$ used in the optimization problem to penalize constraint violations. **e)** Sampling grid for charge sensitivity $\mathcal{S}_n$ with two charge islands n_{g1} and n_{g2} , showing sampled charge values. **f)** Uniform sampling along the diagonal of the hypercube for charge sensitivity calculation, demonstrating another method of charge sampling for circuits with two charge islands.

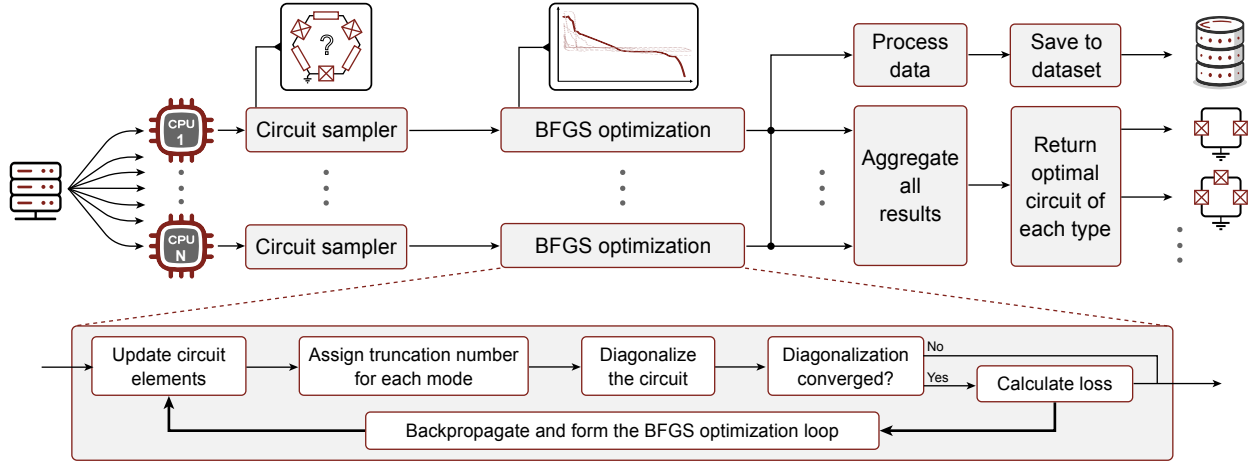


Figure 3: **Optimization pipeline.** The pipeline demonstrates the parallelized optimization process utilizing multiple CPU cores. Each core randomly initializes a superconducting quantum circuit through the circuit sampler module. The BFGS optimization algorithm is then applied, incorporating automatic truncation number assignment and convergence testing for robust optimization. The optimization loop iterates by calculating the loss and updating the circuit elements using gradient information and backpropagation. Aggregated results are processed to identify the optimal circuit configurations. Throughout the process, all circuits that pass the convergence test are stored and saved, creating a high-quality dataset useful for training AI and machine learning models.

appropriate tool. One of the main problems of interest is automating the process of designing useful and practical superconducting circuits. This issue is crucial because as superconducting circuits grow larger they become not only more difficult to simulate but also harder to understand intuitively. The number of circuit properties that designers must track also increases, further complicating the design process.

In this section, we address the problem of qubit discovery and automate the process by framing it as an optimization problem, which we solve using the autodifferentiation capability of `SQcircuit`. Though we have chosen to focus on qubit discovery in this section, our optimization pipeline and the techniques can also be applied to other superconducting quantum circuit discovery problems.

4.1 Practical Qubit Design

To evaluate and compare qubit designs, we need to establish criteria for their performance and feasibility. In the following, we describe the terminology and metrics used.

Search space ($x_{\min}, x_{\max}$). The element values for the circuit should fall within a range that can be fabricated. Therefore, we define the following conservative lower and upper bounds for capacitors, inductors, and junc-

tions:

$$\begin{aligned} c_{\min} &= 1 \text{ femtofarad}, & c_{\max} &= 12 \text{ picofarad}, \\ l_{\min} &= 1 \text{ femtohenry}, & l_{\max} &= 5 \text{ microhenry}, \\ E_{J_{\min}} &= 1 \text{ gigahertz}, & E_{J_{\max}} &= 100 \text{ gigahertz}. \end{aligned}$$

Accordingly, any search algorithm employed must ensure that the qubit components fall within these specified value ranges.

Circuit topology. The topology and arrangement of circuit elements within a graph fundamentally determines the behavior of superconducting circuits. This includes information about the circuit modes (either charge or harmonic), the control knobs for tuning the qubit, and the sources of noise that apply to the circuit. Consequently, this consideration is essential in our quest to identify the optimal qubit design. However, it is neither practical nor efficient to study all possible circuit topologies. Very large circuits are not only difficult to simulate, but also challenging to fabricate and test in a laboratory setting.

A complementary recent approach by Weissler *et al.* systematically enumerates superconducting circuit topologies up to five nodes using graph-isomorphism methods, providing a useful catalogue of the broader circuit-design space [44].

In this work, we then focus on circuits featuring a single flux-tunable inductive loop and up to four nodes, with all-to-all capacitive coupling. Circuits without an inductive loop lack control knobs for tuning the qubit frequency, whereas circuits with more than one induc-

tive loop become difficult to fabricate and tune. Limiting our study to circuits with up to four nodes ensures that the search space remains manageable with the computational resources at our disposal. An additional significant aspect of our selected topologies is that they encompass all qubits documented in the literature that have been experimentally realized. This allows us to use these well-established circuits as baselines for comparison with our optimized circuits.

Since our goal in this manuscript is to exhaustively study all possible circuit configurations, it becomes impractical to plot each circuit separately when referring to them. To address this, we have devised a coding scheme that represents individual circuit topologies as strings:

$$\text{circuit code} = " \prod_{j=1}^N \mathcal{I}_j \left(\mathcal{E}_1^j \mathcal{E}_2^j \dots \mathcal{E}_{n_j}^j \right) ", \quad (8)$$

where $\mathcal{I}_j$ denotes either an inductor (L) or a Josephson junction (J) as an inductive element within a single inductive loop of the circuit. The parentheses next to the $\mathcal{I}_j$ element describes the non-inductive loop to which $\mathcal{I}_j$ belongs, and $\mathcal{E}_l^j$ represents the elements of that non-inductive loop, which can be either an inductor (L), a Josephson junction (J), or a capacitor (C). The parameter n_j indicates the total number of elements in that non-inductive loop. It is important to note that at least one $\mathcal{E}_l^j$ must be a capacitor to prevent the formation of a new inductive loop. If $\mathcal{I}_j$ is not part of any non-inductive loop, the parentheses notation for that element is omitted. For example, the circuits with codes "JLJL" and "JL(JC)" are depicted in Figure 2a.

As previously mentioned, our target topology search includes all known circuits from the literature. We summarize these circuits as follows: for "JJ", we have the flux-tunable transmon from Google Sycamore [45], for "JL", we have the heavy fluxonium [46] and Pop's fluxonium [47], for "JJJ", we have the flux qubit [48] and quantonium [49], and for "JLJL", we have the pokemon [50] and bifluxon [51]. This collection serves as our baseline reservoir, against which we compare our discovery results.

Decoherence time (T). The primary sources of decoherence in a qubit are depolarization and dephasing. Depolarization time T_1 is influenced by capacitive loss, inductive loss, and quasiparticle loss, while dephasing time T_φ is affected by flux noise, charge noise, and critical current noise. The objective is to identify the qubit with the longest decoherence time, defined as:

$$T = \frac{1}{\frac{1}{2T_1} + \frac{1}{T_\varphi}}. \quad (9)$$

This study considers all mentioned loss channels with an experimental basis defined in [13]. Such loss channels represent inherent environmental sources of noise and statistical fluctuations that affect qubits. In contrast, the flux and charge sensitivities help address crucial experimental considerations such as changes in qubit properties due to accumulation of electrons on charge islands as well as control of applied external flux at the critical flux point.

Single-qubit gate speed ($\mathcal{G}$). Another crucial metric for a qubit is the speed at which a single-qubit gate operation can be performed. Although this value generally depends on a time-dependent control signal, this work neglects the drive signal and instead calculates an upper bound for the maximum single-qubit gate speed under ideal conditions. This upper bound is determined by the largest sideband frequency around an integer multiple of the qubit frequency that does not excite the qubit out of its computational states.

We depict this consideration in Figure 2b, and provide further discussion in Appendix D. The mathematical expression for the upper bound is given by:

$$\mathcal{G} = \min_{i \geq 2} (f_i - f_1, |f_i - 2f_1|), \quad (10)$$

where f_i represents the frequency of the i -th mode. Specifically, for $i = 2$, the second term reduces to the standard definition of anharmonicity $\mathcal{A}$ within the literature. This demonstrates the generality of our definition, as it is shown that for fluxonium and transmon, the fastest single-qubit gate with optimized drive signals equals the anharmonicity.

Number of single-qubit gates ($\mathcal{N}$). The number of single-qubit gates that can be performed serves as a principal metric for comparing different qubits and benchmarking them against those reported in the literature.

This number can be approximated by determining how many sequential gates can be executed as quickly as possible before the qubit loses its information to the environment, which we express by

$$\mathcal{N} = T\mathcal{G}, \quad (11)$$

where T represents the decoherence time, and $\mathcal{G}$ is the gate speed upper bound defined previously. This metric aims to strike a balance between maximizing the decoherence time and minimizing the gate operational time, making it a suitable unitless metric for optimization purposes.

Flux sensitivity ($\mathcal{S}_\varphi$). External fluxes coupling to the inductive loops of superconducting circuits serve as an

important and practical degree of freedom in the lab to tune the qubit frequency to its desired working point. However, if the external flux working point φ_{ext}^* is too sensitive to external flux variations, noise fluctuations in the lab can compromise qubit operation. We must then ensure that flux fluctuations due to noise remain within a tolerable regime.

Assuming δ as the maximum range of fluctuation, we define flux sensitivity as (see Figure 2c):

$$\mathcal{S}_\varphi = \frac{|f_q(\varphi_{\text{ext}}^* + \frac{\delta}{2}) - f_q(\varphi_{\text{ext}}^* - \frac{\delta}{2})|}{f_q(\varphi_{\text{ext}}^*)}, \quad (12)$$

where $f_q(\cdot)$ is the qubit frequency as a function of external flux.

Charge sensitivity ($\mathcal{S}_n$). The charge on charge islands can be controlled using DC voltages, offering another degree of freedom for qubit control. However, charge dispersion in qubits is not practical because charge islands accumulate charges over time. To maintain a stable qubit, we must eliminate charge dispersion.

We define charge sensitivity by sampling charge values across all charge degrees of freedom (only charge modes), then calculating the ratio of the maximum qubit frequency range to the average of the minimum and maximum qubit frequency for these samples:

$$\mathcal{S}_n = \frac{\max_i f_q(\mathbf{n}_g^i) - \min_i f_q(\mathbf{n}_g^i)}{\frac{1}{2}(\max_i f_q(\mathbf{n}_g^i) + \min_i f_q(\mathbf{n}_g^i))}, \quad (13)$$

where $f_q(\cdot)$ is the qubit frequency as a function of charge numbers on the islands, and $\mathbf{n}_g^i = (n_{g_1}^i, n_{g_2}^i, \dots)$ is a vector of charge numbers for the j th island $n_{g_j}^i$ in the i th sample. Sampling can be performed over a grid (see Figure 2e for a circuit with two charge islands) or uniformly across the diagonal of the hypercube formed by charge island degrees of freedom (see Figure 2f for a circuit with two charge islands), the latter being the method used in this work.

Element sensitivity ($\mathcal{S}_e$). Another fundamental factor to consider is the qubit sensitivity to fabrication errors. A qubit might exhibit good metrics, but if it is highly sensitive to fabrication errors, achieving a practical yield of this qubit may be challenging. Consequently, this qubit may not provide the expected number of gates.

To account for this effect, we first sample our element values from a normal distribution. For the i th element with value x_i , we sample from a normal distribution with a mean of x_i and a standard deviation of $e_i \cdot x_i$, where e_i represents the fabrication error of the i th element. By sampling our circuits n_S times, we generate n_S circuits, each with a different number of gates. We define the element sensitivity as the standard deviation

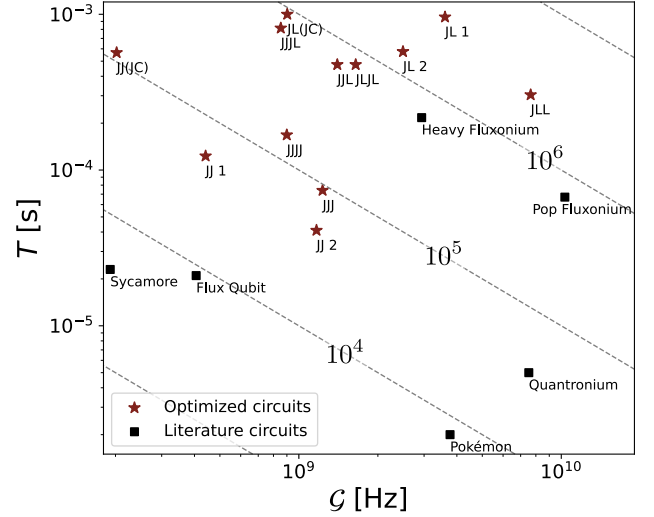


Figure 4: **Gate number of discovered qubits.** The qubits from Table 2 plotted based on decoherence time T and upper bound gate speed G . The x-axis represents G and the y-axis represents T , both presented on a logarithmic scale. The dashed lines indicate constant contours of the gate number, following the expression $GT = \text{const}$. Qubits towards the top right corner of the plot have a larger number of gates. Optimized qubits are marked with red stars, while qubits from the literature are marked with black squares.

of the calculated gate numbers divided by their average, expressed as:

$$\mathcal{S}_e = \frac{\text{std}\{\mathcal{N}\}}{\text{mean}\{\mathcal{N}\}}. \quad (14)$$

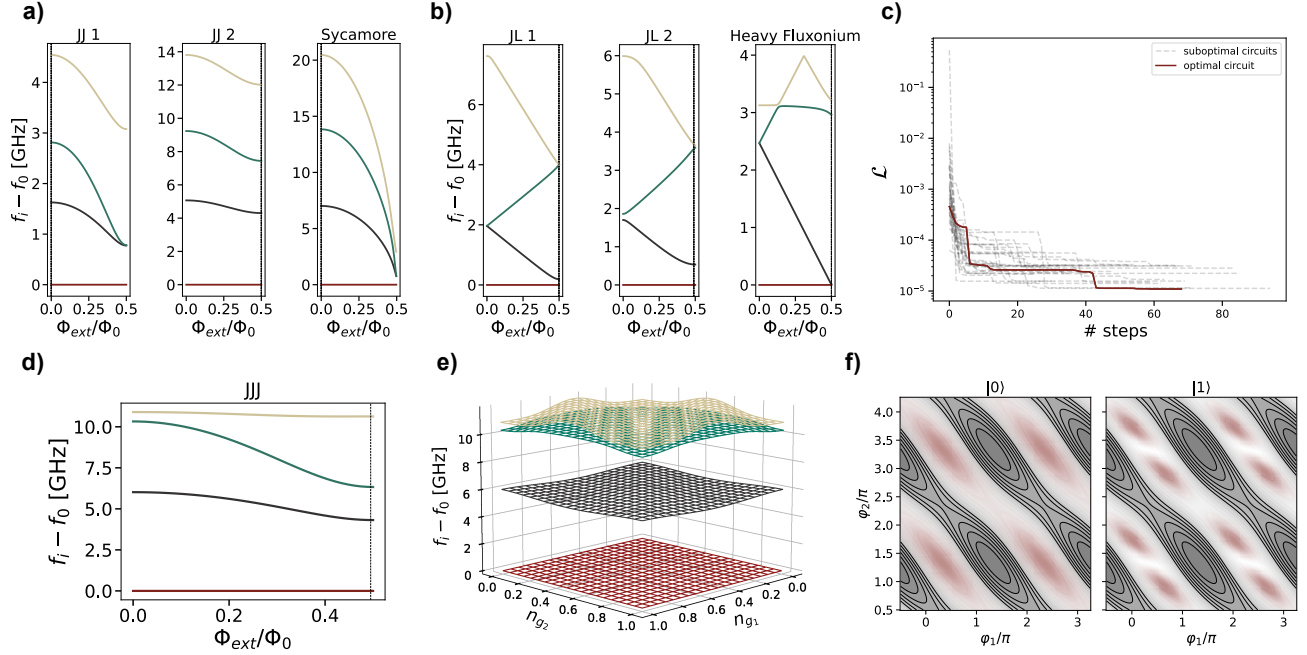
4.2 Search Problem as an Optimization

Now that we have defined all necessary metrics for evaluating and benchmarking a qubit, we can formulate our search for the best qubit as a maximization problem. This problem aims to maximize the number of gates while imposing constraints to ensure the qubits are implementable.

The optimization problem can be written as:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathcal{N} \\ \text{s.t.} \quad & \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}, \\ & \mathcal{S}_i \leq \mathcal{S}_i^*, \quad i \in \{\varphi, n, e\}, \\ & f_q \leq f_q^*, \end{aligned} \quad (15)$$

where $\mathcal{S}_i^*$ is the maximum tolerance for the sensitivity of type i , which can be flux, charge, or element sensitivity, and f_q^* is the maximum possible frequency imposed by the material from which the qubit is built. We propose a dual optimization problem whose solution is equivalent to that of the optimization problem in (15).



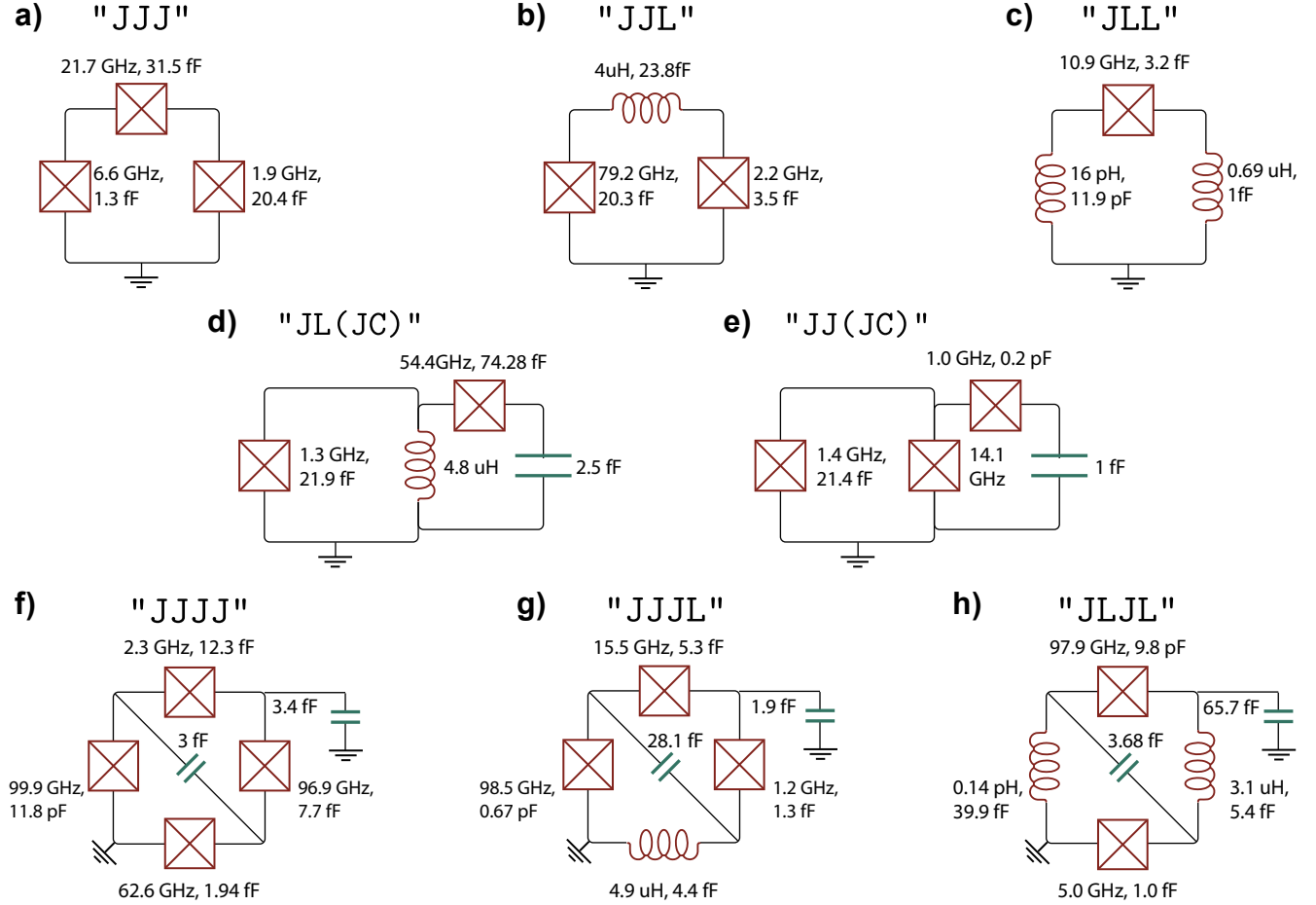


Figure 6: **Optimal circuit element values and their topologies.** **a) "JJJ"** qubit: Three Josephson junctions in an inductive loop with associated capacitors. **b) "JJL"** qubit: Two Josephson junctions and one inductor forming an inductive loop with associated capacitors. **c) "JLL"** qubit: Composed of two inductors and one junction, forming an inductive loop. Despite suggesting a large number of gates, it effectively reduces to fluxonium over the course of the optimization process. **d) "JL(JC)"** qubit: A fluxonium capacitively coupled to a Cooper pair box. The optimization process makes the fluxonium off-resonant with the Cooper pair box, causing it to dominate the lower levels of the system. **e) "JJ(JC)"** qubit: A flux-tunable transmon capacitively coupled to a non-flux-tunable Cooper pair box. **f–h)** Various other qubits such as "JJJJ", "JJJL", and "JLJL", which present interesting qubit designs. Note while these circuits have all-to-all capacitive coupling, we only explicitly indicate the capacitors for branches without an inductive element.

Table 2: **Qubit Discovery Results.** The table below presents the discovered qubits along with their corresponding circuit codes, metrics, and performance characteristics. The metrics include the total loss function $\mathcal{L}$ (16), the number of gates $\mathcal{N}$ (11), the gate speed $\mathcal{G}$ (10), the decoherence time T (9), and the sensitivities $\mathcal{S}_\varphi$ (12), $\mathcal{S}_n$ (13), and $\mathcal{S}_e$ (14). Additionally, the qubit frequency f_q and the external flux working point $\varphi_{\text{ext}}^*/2\pi$ are listed. Optimized qubits are compared with existing qubits from the literature to highlight the improvements achieved through the optimization pipeline.

Circuit Code	$\mathcal{L}$	$\mathcal{N}$	$\mathcal{G}$ [GHz]	T [s]	$\mathcal{S}_\varphi$	$\mathcal{S}_n$	$\mathcal{S}_e$	f_q [GHz]	$\varphi_{\text{ext}}^*/2\pi$
"JJ 1"	1.84E-05	5.43E+04	0.441	1.23E-04	1.08E-03	2.00E-02	5.78E-03	1.626	0.0200
"JJ 2"	2.09E-05	4.77E+04	1.167	4.09E-05	2.18E-04	2.00E-02	4.15E-02	4.305	0.4900
Sycamore [45]	2.32E-04	4.32E+03	0.191	2.25E-05	7.61E-04	8.61E-14	6.87E-03	7.013	0.0100
"JL 1"	2.89E-07	3.46E+06	3.605	9.60E-04	1.35E-02	0.00E+00	1.99E-02	0.192	0.4990
"JL 2"	6.97E-07	1.43E+06	2.493	5.76E-04	1.99E-03	0.00E+00	2.28E-02	0.535	0.4900
Heavy fluxonium [46]	2.56E+00	6.38E+05	2.937	2.17E-04	2.66E+00	0.00E+00	4.35E-02	0.014	0.4999
Pop fluxonium [47]	1.44E-06	6.94E+05	10.32	6.73E-05	3.35E-02	0.00E+00	2.43E-02	0.639	0.5000
"JJJ"	1.10E-05	9.13E+04	1.231	7.42E-05	1.06E-03	8.62E-13	5.99E-03	2.596	0.0200
Flux qubit [48]	1.15E-04	8.68E+03	0.406	2.14E-05	1.70E-02	2.89E-09	3.33E-02	4.530	0.0500
Quantonium [49]	1.90E-01	3.86E+04	7.520	5.14E-06	1.24E-03	2.10E-01	9.62E-03	16.204	0.0100
"JJ(JC)"	8.74E-06	1.14E+05	0.202	5.67E-04	7.37E-10	2.00E-02	1.04E-02	0.745	0.4200
"JLJ"	1.50E-06	6.66E+05	1.401	4.75E-04	1.09E-02	2.32E-10	4.33E-02	0.093	0.4990
Pokemon [50]	9.79E+01	7.67E+03	3.768	2.04E-06	9.80E+01	3.71E-08	9.82E-03	0.002	0.4999
"JL(JC)"	1.10E-06	9.06E+05	0.903	1.00E-03	1.24E-03	3.41E-11	1.16E-02	0.210	0.4990
"JLL"	4.31E-07	2.32E+06	7.643	3.04E-04	9.68E-04	1.00E-14	2.11E-02	0.802	0.4990
"JJJJ"	6.62E-06	1.51E+05	0.899	1.68E-04	2.78E-05	3.32E-04	1.36E-02	1.891	0.4990
"JJJL"	1.44E-06	6.95E+05	0.853	8.15E-04	2.10E-03	2.74E-04	4.69E-03	0.163	0.4990
"JLJL"	1.28E-06	7.82E+05	1.646	4.75E-04	2.03E-03	5.41E-06	5.74E-03	0.269	0.4990

To this end, we define the optimization problem as:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathcal{L}_{\text{obj}} + \mathcal{L}_{\text{const}} \\ \text{s.t.} \quad & \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}, \end{aligned} \quad (16)$$

where

$$\begin{aligned} \mathcal{L}_{\text{obj}} &= \frac{1}{\mathcal{N}}, \\ \mathcal{L}_{\text{const}} &= \sum_{i \in \{\varphi, n, e\}} \beta_i \mathcal{L}_h(\mathcal{S}_i, \mathcal{S}_i^*) + \beta_f \mathcal{L}_h\left(\frac{f_q}{f_q^*}, 1\right), \end{aligned}$$

with β_i being the loss function weights, and $\mathcal{L}_h(\cdot, \cdot)$ the hinge loss function depicted in Figure 2d defined as:

$$\mathcal{L}_h(x, a) = \begin{cases} x - a & \text{if } x > a, \\ 0 & \text{if } x \leq a. \end{cases}$$

We use hinge loss to penalize the unsatisfiability of constraints in the optimization problem of Equation (15), returning a zero gradient when each constraint is satisfied. Wrapping the sensitivity loss terms in a hinge loss ensures that once the flux and charge sensitivities are below some threshold, the model can divert its attention to focus on scalar loss terms such as the decoherence and gate speed.

In this work, we set all loss weights to $\beta_\varphi = \beta_n = \beta_e = \beta_f = 1$. This choice is motivated by the fact that the constraint terms are normalized hinge losses, so unit weights assign comparable penalties to order-one violations of each constraint. Since the hinge penalties vanish once the constraints are satisfied, the final feasible optima are determined primarily by maximizing $\mathcal{N} = T\mathcal{G}$ rather than by fine tuning the β_i values. In practice, we find that the results are not strongly sensitive to the precise choice of β_i within roughly an order of magnitude around unity; outside this range, too-small values can allow constraint violations, while much larger values can make the optimization more conservative.

We denote $\mathbf{x}^*$ as the global optimum of (15) which satisfies each constraint and maximizes the gate number $\mathcal{N}^*$. If the coefficients β_i are sufficiently large, it is clear that $\mathbf{x}^*$ is also the global optimum of (16). This is because $\mathcal{L}_{\text{const}} = 0$ (as all hinge losses are zero due to constraint satisfaction), and $\mathcal{N}^*$ also minimizes $\mathcal{L}_{\text{obj}}$ by definition. Since the optimization problem in (16) satisfies the form of (4), we can use `SQcircuit` functionalities to perform gradient-based optimization and solve this problem.

4.3 Optimization Pipeline

To solve the optimization problem in Equation (16), we employ the BFGS algorithm [31] with random restarts. The BFGS algorithm is a quasi-Newton method that approximates the Hessian matrix of second-order partial derivatives of the loss function using updates derived from gradient evaluations. This approach avoids the need for a fixed step size, which is crucial for a fast and efficient optimization. The random restart strategy involves running multiple instances of BFGS with different circuit initializations and comparing their results to identify better local optima (or potentially the global optimum). By leveraging multiple CPU cores to execute these BFGS optimizations in parallel, we can significantly accelerate the search for the global optimum. The entire optimization pipeline is illustrated in Figure 3, with details elaborated upon below.

The process begins by assigning the optimization tasks to n_{CPU} CPU cores. Each core randomly initializes a superconducting quantum circuit via the circuit sampler module, within the range of elements and circuit topologies discussed earlier.

We then assign a truncation number to each mode of the circuit. This altogether defines the total size of the Hilbert space K and represents the main constraint upon available computational power and memory. Typically, equal truncation numbers are assigned to each mode due to the uncertainty of which mode requires more truncation. However, in this manuscript we employ our own method, detailed in Appendix F, which is more efficient than equal truncation and leads to more efficient circuit diagonalization.

In the subsequent step, we diagonalize the circuit and check for convergence of the eigenvalues and eigenvectors using the methodology described in Appendix E. If the circuit does not converge, we terminate the process and archive the circuit for further investigation. If the circuit does converge, we calculate the loss function and use automatic differentiation within **SQcircuit** to compute the gradient and run the optimization loop for BFGS. After all runs have completed, we aggregate the results and select the best circuit for that topology, as each topology involves distinct physics worth studying.

Our automatic truncation number assignment, combined with our convergence test, allows us to robustly explore the entire target search space and reliably diagonalize explored circuits with adequate accuracy. As a beneficial side effect, this process generates a collection of well-characterized circuits. While not the primary focus of our work, this data could potentially be used in future research, such as for training machine learning models to analyze superconducting quantum circuits.

As a final point, it is important to note that we do not directly optimize over unitful circuit element val-

Table 3: **"JJ" element values.** Optimized circuit element values for "JJ" qubits compared to the Sycamore qubit. The table lists the Josephson junction energies (E_{J_1} and E_{J_2}) and capacitance (C) for each circuit.

Circuit Code	E_{J_1} [GHz]	E_{J_2} [GHz]	C [fF]
"JJ 1"	1.15	1.04	101
"JJ 2"	6.79	1.00	38.1
Sycamore [45]	18.0	18.0	108

Table 4: **"JL" element values.** Optimized circuit element values for "JL" qubits compared to the heavy fluxonium qubit. The table presents the Josephson junction energy (E_J), inductance (L), and capacitance (C) for each circuit.

Circuit Code	E_J [GHz]	L [uH]	C [fF]
"JL 1"	6.86	1.61	9.76
"JL 2"	4.26	1.83	8.08
Heavy fluxonium [46]	3.39	1.23	40.4

ues $\mathbf{x}$. The typical units in which we choose to express the circuit element properties vary significantly over orders of magnitude. For example, capacitors can easily range from picofarads to femtofarads. This variation can make the loss landscape ill-conditioned for optimization. To address this issue, we apply a reparameterization technique to our problem, optimizing over α_x rather than $\mathbf{x}$, with the following relationship:

$$\log x = \frac{\log x_{\max} + \log x_{\min}}{2} + \left(\frac{\log x_{\max} - \log x_{\min}}{2} \right) \cos \alpha_x.$$

This approach ensures that the values of x remain within the specified range while optimizing over the unitless values α_x , which lie between 0 and 2π . The logarithmic scale aids in exploring the range of element values across orders of magnitude and prevents the optimizer from neglecting smaller element values.

4.4 Discovered Qubits

We utilized the Sherlock High-Performance Computing cluster at Stanford University to execute the optimization pipeline illustrated in Figure 3. This pipeline optimized the loss function defined in Equation (16) using 100 steps of the BFGS algorithm for each circuit instance. The hyperparameters related to the loss function weights were set to $\beta_\varphi = \beta_n = \beta_e = \beta_f = 1$, with maximum tolerances for external flux, charge, and element sensitivity set to $\mathcal{S}_\varphi^* = 0.1$, $\mathcal{S}_n^* = 0.02$, and $\mathcal{S}_e^* = 0.1$ respectively.

To determine the total truncation number K (proxy for Hilbert space size; see Appendix F) for each circuit topology, we first initialized one hundred random circuits for each topology. We then plotted the flux spectrum for the first ten eigenfrequencies of these circuits vs total truncation number and selected the minimum truncation number required for the convergence of all one hundred circuits. This approach helps validate the output and methodology of the convergence test that we describe in Appendix E, and provides a total Hilbert space size that is sufficiently large to explore the search space, but not so large as to make optimization excessively slow. The maximum K used in this experiment was 1.6×10^4 , which was crucial for exploring larger circuits, such as those with four nodes.

The optimization results for some of the discovered qubits are summarized in Table 2. The table includes all necessary metrics for comparing optimized qubits, such as the total loss function $\mathcal{L}$, the number of gates $\mathcal{N}$, decoherence time T , and other relevant metrics. The optimized qubits are represented by their circuit code in a red string format. Additionally, we have included the corresponding circuits from the literature for each circuit topology next to the corresponding circuit code. This facilitates a more precise comparison between the discovered qubits and those previously built and documented in the literature.

Since the estimated number of gates $\mathcal{N}$ was our primary optimization metric, and $\mathcal{N}$ depends on the decoherence time T and the upper bound gate speed $\mathcal{G}$, we have plotted the qubits from Table 2 in Figure 4 based on these metrics. In the plot, the x -axis represents $\mathcal{G}$, and the y -axis represents T , both on a logarithmic scale. The dashed lines indicate constant contours of the gate number, expressed as $\mathcal{G}T = \text{const}$. Thus, qubits located towards the top-right corner of the plot correspond to those with a larger number of gates.

It is important to note that Figure 4 does not display the metrics for flux, charge, and element sensitivities. For example, although the heavy fluxonium qubit's properties implies a high number of gates, it suffers from flux sensitivity, which our search has penalized. This issue is not present in our optimized "JL" circuits. Generally, all optimized circuits are designed to avoid such sensitivities, as specified in our loss function design in Equation (16).

To ensure that our optimization pipeline discovers circuits as intended, we focus on "JJ" circuits, which are analytically calculable and thus easier to understand. These circuits consist of two Josephson junctions, E_{J_1} and E_{J_2} , which form an inductive loop. The junctions are not necessarily equal and are shunted by a single capacitor, C . Our search has identified two types of "JJ" qubits. The external flux spectrum and flux working

points of these qubits, along with their literature counterpart, the Sycamore transmon [45], are depicted in Figure 5a. The corresponding circuit element values are provided in Table 3.

The first discovered qubit, "JJ 1", has φ_{ext}^* close to zero and two junctions with values close to each other, combined with a large shunt capacitance to remove charge sensitivity. This suggests that our optimization pipeline has rediscovered the equivalent of the transmon qubit.

The second discovered qubit, "JJ 2", has $\varphi_{\text{ext}}^* = \pi$, which is a less common working point for the transmon since it is very sensitive to external flux. However, our discovery pipeline circumvents this issue by identifying asymmetric Josephson junctions that reduce the flux sensitivity at $\varphi_{\text{ext}}^* = \pi$ [52]. Interestingly, both optimized qubits have an order of magnitude larger number of gates compared to their literature counterparts. It is important to clarify that our comparison of the optimized qubits and their literature counterparts, in terms of gate count, is intended solely to underscore the inherent potential of each circuit topology. For example, the Google Sycamore qubit may have been optimized for other functionalities, including superior two-qubit gates. Our intent is to emphasize the inherent capabilities of this type of qubit relative to other circuit topologies.

One of the key outcomes of our numerical experiments is the optimization of the "JL" circuits, which is a class of circuits that includes the fluxonium qubit. This topology consists of a Josephson junction E_J and a superinductor L , which together form an inductive loop, and a capacitor C . The optimized "JL" circuits exhibit a number of gates that are an order of magnitude superior to Pop *et al.*'s fluxonium [47] and heavy fluxonium [46]. Not only do we predict a higher gate number metric, we also predict that the optimized qubits are significantly less sensitive to external fluxes. This reduced sensitivity mitigates the primary drawback of the heavy fluxonium circuit, which is prone to dephasing due to external flux interference.

To underscore the differences in flux sensitivity, we have plotted the flux spectrum of our optimized "JL" circuits alongside that of the heavy fluxonium in Figure 5b. The figure clearly illustrates that our optimized qubits are substantially less sensitive to external flux perturbations at their operational flux points. This innovative qubit design comprises a range of circuit element values that have not been investigated previously. For a detailed comparison, Table 4 provides a summary of the circuit element values for the optimized "JL" circuits, heavy fluxonium, and Pop's fluxonium.

Another interesting discovered qubit is the "JJJ" qubit depicted in Figure 6a, which we examine here to

elaborate more on our optimization pipeline. This circuit consists of three Josephson junctions in a loop: E_{J_1} , E_{J_2} , and E_{J_3} , with capacitors C_1 , C_2 , and C_3 . This circuit features two charge islands and one inductive loop, for which we must ensure that the islands have minimal charge dispersion and the loop is not flux-sensitive. The BFGS optimization of 100 instances of this circuit topology is depicted in Figure 5c. The dashed black lines represent suboptimal circuits, while the solid red line represents the optimal (best-performing) circuit. Notably, the final results are sensitive to the initial choice of sampled circuit parameters. A significant majority of the initial sampled parameter choices converged to final loss values that were considerably higher than those of the optimal circuit, suggesting that the landscape is dominated by local optima, and depends strongly on the choice of initial condition.

The flux spectrum of the optimal circuit is shown in Figure 5d, and the charge spectrum for the excited states of the qubit is depicted in Figure 5e. The charge spectrum of the qubit shows that this qubit's first excited state is insensitive to the charge degree of freedom, as intended.

Another notable aspect of this qubit is that it has an order of magnitude higher number of gates compared to other circuits in the literature with the same topology, such as the flux qubit [48] and quantronium [49]. The eigenstates for the ground state and first excited state of the qubit, as a function of phase space coordinates, are plotted in the Hamiltonian potential landscape in Figure 5f. This suggests that this qubit has two-dimensional transmon-like eigenvectors localized in the minima of the circuit potential.

There are several qubit codes presented in Table 2 and Figure 4, such as "JLL", "JL(JC)", and "JJ(JC)", which propose intriguing qubits (see Figures 6c-e) that can be strong candidates as they support large $\mathcal{N}$ when optimized. However, we find that the optimized designs are ultimately nearly equivalent to previously discovered qubits with fewer elements. For example, the "JLL" qubit is composed of two inductors and one junction, forming an inductive loop. This qubit suggests a large number of gates comparable to "JL" qubit. However, the optimization achieves this number by shorting one of the inductors, reducing the circuit to fluxonium.

We observe a similar phenomenon with "JL(JC)". This qubit also indicates a high value for $\mathcal{N}$. By examining the architecture of this qubit in Figure 2a, we see that the circuit consists of a fluxonium capacitively coupled to a Cooper pair box. However, the optimizer achieves favorable numbers for this qubit by making the fluxonium off-resonant with the Cooper pair box, resulting in the lower levels of the eigensystem being dominated by the properties of fluxonium.

The optimized "JJ(JC)" qubit suffers from the same issue. This circuit is essentially a flux-tunable transmon capacitively coupled to a non-flux-tunable Cooper pair box. The optimizer moves the Cooper pair box resonance away from the tunable transmon's resonance, leading to the lowest energy eigenstates of the total system being inherited from the Cooper pair box. Since this part of the system is not flux-tunable, we obtain a flux-insensitive spectrum for the qubit levels, and a qubit which is practically undesirable.

Further qubit designs such as "JLL", "JJJJ", "JJJL", and "JLJL", did not demonstrate significantly higher performance (by the considered metric) when compared to the optimized "JL" qubit. However, these qubits seemed to have spectra and physics that differed from fluxoniums and transmons. Moreover, given their greater number of nodes and richer physics, these qubits may offer additional degrees of freedom for encoding states, performing readout and realizing two-qubit interactions. These factors were not captured by the loss function employed in this study, presenting a potential area for future research. Their circuit element values and their topologies are summarized in Figure 6.

Overall, we demonstrate that our optimization pipeline can be utilized for search problems, specifically that of qubit discovery. This approach can lead to the identification of new qubits and improvements over the current state of qubits in the literature. Our pipeline adapts easily to other problems and can be improved by simply changing the loss function to suit the new problem. In summary, our search suggests the fluxonium qubit has the best overall properties for the objective function we have considered.

5 Outlook

In this work, we have developed an optimization pipeline for optimizing properties of superconducting circuits. We have used this to search for superconducting qubit designs with the best performance. Our search was constrained to a specific range of element values and circuit topologies. We also assumed that the logical state of the qubit is encoded in the first excited state and ground state, leaving out for now alternative encodings such as bosonic [53] or dual-rail qubits [54, 55].

A second limitation in our approach is that, in estimating the upper bound of gate speed, we focused solely on the frequency spacing between states, without accounting for the coupling strength between them. However, coupling strength can also play a critical role in a practical setting where drive power is limited.

Additionally, our bounded estimate of gate speed does not account for certain considerations such as control constraints or sources of leakage induced by the

drive source. Incorporating these quantities via a more robust modeling of the readout and control schemes used to interact with the qubit could increase numerical precision and help adapt the qubit design to more realistic scenarios.

Finally, we have only considered circuits with up to 4 nodes. Searching through circuits with a higher number of nodes required computational resource beyond those available to us currently. This is due to the exponential growth in memory requirements. To address this issue, more efficient representations of the circuit’s Hilbert space, such as tensor networks [56] could be developed.

Our optimization pipeline can be directly applied to discover or optimize classes of superconducting circuits beyond qubits. For instance, this pipeline could be used to design coupler circuits that control the interaction between systems, such as toggling the interaction between two qubits on and off, enhancing qubit readout properties, or improving specific gate implementations for certain qubits. Additionally, it could be used to discover circuits that enhance the performance of quantum sensors or transducers.

6 Online Presence

SQcircuit is a robust, BSD-3 licensed, open-source Python package that has been substantially enhanced in this iteration to include automatic differentiation capabilities. These capabilities are detailed in Section 3 and are now integrated into the **SQcircuit** software as a new computational engine. The updated version is available on our GitHub repository, which also hosts a range of examples demonstrating the new features. Comprehensive documentation on how to install, update, and deploy these new features can be found on the official **SQcircuit** website. For developers and researchers interested in the optimization processes and qubit discovery algorithms discussed in Section 4, detailed source code and tutorials are available in a dedicated GitHub repository. We actively encourage the community to contribute to the ongoing development of **SQcircuit** through bug reports, comments, and improvements, or by initiating pull requests on our GitHub repositories. For a list of links to these resources, see Table 5.

7 Conclusion

In this paper, we introduced a generalized framework for gradient-based optimization of superconducting quantum circuits using the **SQcircuit** software package. This framework incorporates automatic differentiation,

Table 5: **Online Resources.** Links to the project’s main website, source code repositories on GitHub, and example notebooks.

SQcircuit website and documentation: sqcircuit.org
GitHub repository for SQcircuit source code: github.com/stanfordLINGS/SQcircuit
GitHub repository for SQcircuit example notebooks: github.com/stanfordLINGS/SQcircuit-examples
GitHub repository for qubit discovery source code and examples: github.com/stanfordLINGS/Qubit-Discovery

enabling the efficient computation of gradients for various circuit properties with respect to arbitrary circuit elements. We demonstrated this approach by applying it to the qubit discovery problem, allowing for the identification of qubit designs with improved performance metrics. The methodology presented in this work is not limited to qubit discovery but can be extended to other optimization problems within the realm of superconducting quantum hardware. By leveraging the capabilities of **SQcircuit**, researchers can explore a wide range of circuit configurations and optimization criteria, ultimately accelerating the development of advanced quantum technologies. We encourage the community to utilize and contribute to the open-source **SQcircuit** project, as detailed in Section 6, to further enhance the tool’s functionality and broaden its application scope.

Author Contributions

T.R. and A.B. developed the initial theory for the back-propagation pass and wrote the initial implementation in **PyTorch**. T.R. and S.B. calculated the Hamiltonian derivatives. T.R., A.B., and S.B. implemented the optimization infrastructure in **SQcircuit**. T.R. and A.B. performed experiments on diagonalization convergence. T.R. implemented the BFGS algorithm. A.B., S.B., and T.R. performed experiments on Stanford’s Sherlock cluster and analyzed the results. T.R. wrote the bulk of the manuscript, and used AI for grammatical editing of portions of the manuscript. A.B. and S.B. also wrote parts of the manuscript, including appendices. T.R. created **SQcircuit**’s website and established its online presence pipeline. A.H.S.-N. supervised the project from its inception. D.I.S. reviewed the manuscript and provided additional feedback. No AI tools were used at any stage in analytical derivations or in developing the code, unless explicitly stated. All authors reviewed the final manuscript and take full responsibility for its content.

Acknowledgments

We gratefully acknowledge funding from Amazon Web Services Inc., and NTT Research for their valuable financial and technical support throughout this project. We also recognize the support provided by the U.S. Government through the Air Force Office of Scientific Research and the Office of Naval Research under award number FA9550-23-1-0338, and Army Research Office (ARO)/Laboratory for Physical Sciences (LPS) Modular Quantum Gates (ModQ) program (Grant No. W911NF-23-1-0254), and the National Science Foundation CAREER award No. ECCS-1941826. Some of the computing for this project was performed on the Sherlock cluster. We would like to thank Stanford University and the Stanford Research Computing Center for providing computational resources and support that contributed to these research results.

A PyTorch Computational Graph

In this section, we explain the computational graph, forward pass, and backward pass functionalities of **PyTorch** with a simple example depicted in Figure 7. **PyTorch**, an open-source machine learning library developed by Facebook’s AI Research lab, provides a flexible and efficient platform for building deep learning models, performing tensor computations with GPU acceleration, and utilizing automatic differentiation to compute gradients. Its dynamic computational graph and ease of use make it widely popular in both academia and industry.

A computational graph is a directed graph where nodes represent operations or variables, and edges represent the flow of data. In **PyTorch**, computational graphs are created dynamically during the forward pass of a network, allowing for flexibility and ease of debugging.

In Figure 7a, the computational graph consists of the following elements:

- x : An input tensor with an initial value of 1.0, which requires gradient computation.
- Squaring: An operation that squares the value of x , producing an intermediate variable a .
- a : The result of squaring x .
- y : Another input tensor with an initial value of 1.0, which also requires gradient computation.
- Multiplication: An operation that multiplies a by y , resulting in the output $\mathcal{L}$.
- $\mathcal{L}$: The final output, representing the loss function $\mathcal{L} = x^2 y$.

The forward pass is the process of computing the output of a computational graph given an input. Each operation in the graph is applied sequentially to propagate the input through the network to produce the output. In the given example, the forward pass involves the following steps:

1. x : Input tensor with $x = 1.0$,
2. y : Input tensor with $y = 1.0$,
3. a : Intermediate variable $a = x^2$,
4. $\mathcal{L}$: Output loss $\mathcal{L} = a \cdot y = x^2 \cdot y$.

The corresponding **PyTorch** code for the forward pass is:

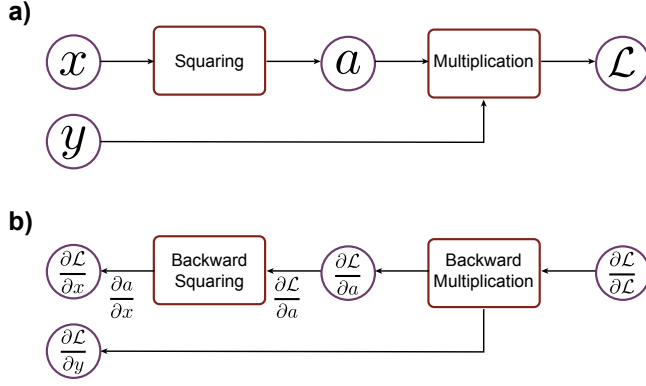


Figure 7: **Example of a computational graph.** **a)** The graph illustrates the sequence of operations and data flow during the forward pass. The input tensor x is squared to produce the intermediate variable a , which is then multiplied by another input tensor y to obtain the output loss $\mathcal{L} = x^2 y$. **b)** The backward pass through the computational graph, showing the gradients computation. The gradient of the loss with respect to each variable is computed by applying the chain rule. The gradients flow backwards through the graph: from the loss $\mathcal{L}$ to y and a , and finally to x . The gradients of x and y are $\frac{\partial \mathcal{L}}{\partial x} = 2xy$ and $\frac{\partial \mathcal{L}}{\partial y} = x^2$, respectively.

```
import torch

x = torch.tensor([1.0], requires_grad=True)
y = torch.tensor([1.0], requires_grad=True)

a = torch.pow(x, 2)
L = torch.mul(y, a)
```

The backward pass is the process of computing the gradients of the loss function with respect to the input variables by applying the chain rule of calculus. This process, known as backpropagation, is essential for optimizing the parameters. In the given example that is also depicted in Figure 7b, the backward pass involves the following steps:

1. Compute the gradient of the loss with respect to the output $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1.$$

2. Compute the gradient of the loss with respect to y :

$$\frac{\partial \mathcal{L}}{\partial y} = x^2$$

3. Compute the gradient of the loss with respect to a :

$$\frac{\partial \mathcal{L}}{\partial a} = y.$$

4. Compute the gradient of the intermediate variable a with respect to x :

$$\frac{\partial a}{\partial x} = 2x.$$

5. Apply the chain rule to compute the gradient of the loss with respect to x :

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial x} = y \cdot 2x.$$

Using the PyTorch code provided, the backward pass is automatically handled by calling the `backward()` method on the loss tensor:

```
L.backward()
```

After executing the backward pass, the gradients of x and y can be accessed via their `.grad` attributes:

```
print(x.grad) # Output: tensor(2.0)
print(y.grad) # Output: tensor(1.0)
```

The output confirms that `x.grad` returns 2.0 and `y.grad` returns 1.0, as expected.

B SQcircuit Tutorial

In this appendix, we will provide a short tutorial on using SQcircuit to calculate the gradient of circuit properties.

First, we import the SQcircuit and PyTorch library and use the `set_engine` function to switch the SQcircuit engine from "NumPy" to "PyTorch":

```
# Import the libraries
import SQcircuit as sq
import torch

# Switch to the PyTorch engine
sq.set_engine("PyTorch")
```

Next, we construct the circuit using SQcircuit. This is similar to the procedure described in [13], but here we specify the elements or properties for which we want to calculate gradients by setting `requires_grad=True`. In the following example, we construct a transmon circuit and calculate the gradient of some of its properties with respect to its capacitor. We define the circuit as follows:

```
# Define the transmon elements
C = sq.Capacitor(
    12, "fF", requires_grad=True
)
JJ = sq.Junction(10, "GHz")
elements = {(0, 1): [C, JJ]}

# Define the transmon circuit
transmon = sq.Circuit(elements)
```

Here, we set `requires_grad=True` only for the capacitor and not for the Josephson junction, as we are interested in the gradient with respect to the capacitor, not the junction. Next, we set the truncation number for the only charge mode of the circuit and diagonalize the circuit to calculate its eigenfrequencies and eigenvectors:

```
# Set truncation number
transmon.set_trunc_nums([100])

# Diagonalize the circuit
efreqs, evecs = transmon.diag(n_eig=10)
```

In contrast to the "NumPy" engine of `SQcircuit`, the variables `efreqs` and `evecs` are represented as `PyTorch` tensors instead of `NumPy` or `QuTiP` objects. Additionally, properties such as decoherence times, coupling operators, and element values are all represented by `PyTorch` tensors. This is essential for gradient calculation and maintaining an appropriate computational graph. It is also important to note that the precision of the eigenvector gradient is proportional to `n_eig`, denoted by n_E in Equation (7).

After diagonalizing the circuit Hamiltonian—an essential step before calculating any circuit properties—we aim to compute the gradient of the matrix element between the first excited state and the ground state with respect to the capacitor. The matrix element is defined as:

$$g_{10} = \frac{1}{c} \left| \langle f_1 | \hat{Q} | f_0 \rangle \right|.$$

Here, g_{10} is the matrix element, c is the capacitance, $|f_0\rangle$ and $|f_1\rangle$ are the ground and first excited states respectively, and $\hat{Q}$ is the charge operator. As g_{10} takes the form of a scalar metric described in Equation (3), we can use `SQcircuit` to calculate its gradient with respect to the capacitor:

```
# Calculate the matrix element
g_10 = 1 / C.get_value() * torch.abs(
    evecs[1].conj()
    @ transmon.charge_op(1, "original")
    @ evecs[0]
)

# Backpropagation step
g_10.backward()

# Access the gradient
print(C.grad)
```

In the above code, `.backward()` propagates the gradient in the reverse direction (applying the chain rule) to compute the gradient of any property, here `g_10`, with respect to any element or property that has `requires_grad=True`. In this case, `C` is the only variable with gradient computation enabled. The units of

the gradient in `SQcircuit` are generally given by:

$$\text{Unit}[\text{gradient}] = \frac{\text{Unit}[\text{property}]}{\text{Unit}[\text{element}]},$$

where, since the unit of the capacitor is farads, the unit of the gradient becomes the unit of the matrix element per farad. If the element is an inductor or junction, the default units for the gradient calculations are henry or hertz respectively (external flux and charge values are unitless).

C Dephasing Time Gradient

In `SQcircuit`, dephasing rates between eigenstates i and j are calculated for critical current, charge, and flux noise via the equation

$$\Gamma_{\varphi,\lambda} = \sqrt{|2 \ln \omega_{\text{low}} t_{\text{exp}}|} \left| A_{\lambda} \frac{\partial f_{ij}}{\partial \lambda} \right|,$$

where ω_{low} is a fixed low-frequency cutoff, t_{exp} is the measurement time, A_{λ} is the noise amplitude, and λ is the noisy parameter for the channel of interest. Specifically $\lambda = E_{J_j}$ for critical current noise, n_{g_j} for charge noise, and φ_{ext_j} for flux noise (summing over all instances present in the circuit).

Because this rate is dependent on the first derivative of f_{ij} , derivatives of Γ_{φ} will depend on the second derivative of f_{ij} . In particular, for a parameter x of the circuit,

$$\begin{aligned} \frac{\partial \Gamma_{\varphi,\lambda}}{\partial x} &= \text{sgn} \left(A_{\lambda} \frac{\partial f_{ij}}{\partial \lambda} \right) \sqrt{|2 \ln \omega_{\text{low}} t_{\text{exp}}|} \\ &\quad \times \left(\frac{\partial A_{\lambda}}{\partial x} \frac{\partial f_{ij}}{\partial \lambda} + A_{\lambda} \frac{\partial^2 f_{ij}}{\partial x \partial \lambda} \right). \end{aligned}$$

As discussed at the end of Section 3, because the first-order derivatives are implemented as a custom computational node, `PyTorch` cannot compute the second-order derivatives automatically. Therefore, we implement an additional computational node to calculate the gradient of T_{φ} . The first-order derivatives of A_{λ} are all straightforward and are given in Table 6.

An expression for the second-order derivative of f_i can be found by differentiating (6), which gives

$$\frac{\partial^2 f_i}{\partial x \partial \lambda} = \langle f_i | \frac{\partial^2 \hat{H}}{\partial x \partial \lambda} | f_i \rangle + 2\Re \langle \frac{\partial f_i}{\partial x} | \frac{\partial \hat{H}}{\partial \lambda} | f_i \rangle. \quad (17)$$

The partial derivatives for $|f_i\rangle$ are given by (7), and the first-order derivatives of $\hat{H}$ in Table 1, so the only novel feature of (17) is the second-order Hamiltonian derivative. Luckily, these vanish for many combinations of (x, λ) . In Table 7, we present all nonvanishing second-order derivatives for $\lambda \in \{E_{J_j}, n_{g_j}, \varphi_{\text{ext}_j}\}$ and $x \in \{c_i, l_i, E_{J_i}, \varphi_{\text{ext}_i}, n_{g_i}\}$. As in Table 1, we assume all external fluxes are assigned to junctions.

Table 6: Derivatives of A_λ for supported noise channels.

Noise channel	A_λ	$\partial A_\lambda / \partial x$
Critical current	$A_{cc} E_J^a$	$A_{cc} \delta_{xE_J}$
Charge	A_{ch}^b	0
Flux noise	A_{flux}^c	0

^aThe default value is $A_{cc} = 10^{-7}$.

^bThe default value is $A_{ch} = e \times 10^{-4}$.

^cThe default value is $A_{flux} = 2\pi \times 10^{-6}$.

 Table 7: Nonvanishing second-order derivatives of the Hamiltonian required for computing gradients of T_φ .

Hamiltonian Derivative	Original Basis
$\partial^2 \hat{H} / \partial \varphi_{\text{ext}_i} \partial E_{J_j}$	$b_{ji} \sin \left(\frac{2\pi}{\Phi_0} \mathbf{w}_j^T \hat{\Phi} + \mathbf{b}_j^T \varphi_{\text{ext}} \right)$
$\partial^2 \hat{H} / \partial c_i \partial n_{g_j}$	$-e_j^T \mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial c_i} \mathbf{C}^{-1} \hat{\mathbf{Q}}$
$\partial^2 \hat{H} / \partial n_{g_i} \partial n_{g_j}$	$e_j^T \mathbf{C}^{-1} e_i$
$\partial^2 \hat{H} / \partial E_{J_i} \partial \varphi_{\text{ext}_j}$	$b_{ij} \sin \left(\frac{2\pi}{\Phi_0} \mathbf{w}_i^T \hat{\Phi} + \mathbf{b}_i^T \varphi_{\text{ext}} \right)$
$\partial^2 \hat{H} / \partial \varphi_{\text{ext}_i} \partial \varphi_{\text{ext}_j}$	$\sum_{k \in S_J} \left[E_{J_k} b_{ki} b_{kj} \times \cos \left(\frac{2\pi}{\Phi_0} \mathbf{w}_k^T \hat{\Phi} + \mathbf{b}_k^T \varphi_{\text{ext}} \right) \right]$

D Upper Bound on Gate Speed

To implement a single-qubit gate, we need to drive the qubit with a time-dependent signal to achieve the desired unitary transformation that evolves the state of the qubit according to the requirements of the quantum gate. This involves externally coupling to the qubit and applying a signal with a frequency that matches the qubit frequency f_{10} , thereby enabling excitation between the two states of the qubit. The evolution of the system can be described by the total Hamiltonian, which is given abstractly as

$$\hat{H}(t) = \hat{H}_q + f(t) \hat{H}_d,$$

where $\hat{H}_q$ is the Hamiltonian of the qubit without any drive, $\hat{H}_d$ is the drive Hamiltonian that depends on the type of external coupling, and $f(t)$ is a time-dependent function typically expressed as

$$f(t) = s(t) \cos(2\pi f_{10} t),$$

where $s(t)$ is a pulse with a characteristic width of $2T_G$, which defines the speed of the gate. In the Fourier domain of the pulse signal, as shown in Figure 8a, the pulse creates sidebands with a width of $2f_G \propto 1/2T_G$ around the excitation frequency f_{10} and its harmonics nf_{10} , where n is an integer. This is represented as a box around these excitation frequencies in Figure 8b. In this

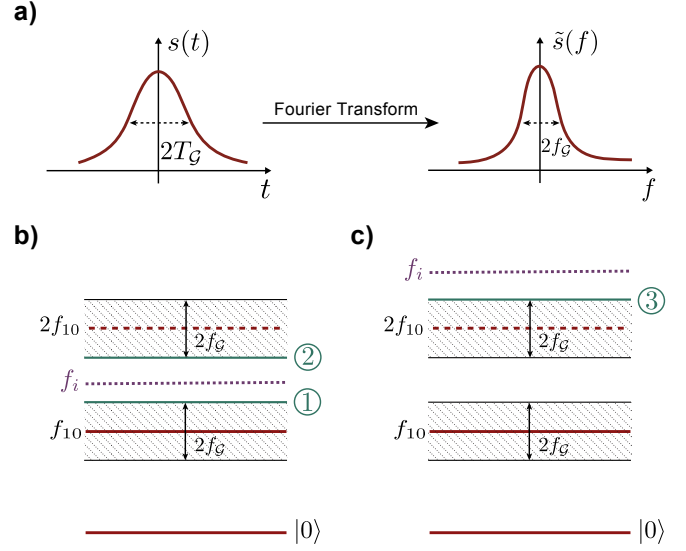


Figure 8: **Illustration of pulse signal and upper bound for Gate Speed.** **a)** The pulse signal $s(t)$ is depicted on the left with a width of $2T_G$. The right figure shows the Fourier transform of the pulse $\tilde{s}(f)$, which has a width $2f_G \propto 1/2T_G$. **b)** The energy levels of the qubit with drive excitation frequencies f_{10} and $2f_{10}$, along with a sideband box with a width of $2f_G$. The eigenenergy f_i of the i th excited state is lower than $2f_{10}$ and should fall between the green boundaries (1 and 2) to avoid excitation of the qubit states. **c)** The eigenenergy f_i is higher than $2f_{10}$ and should be above the green boundary (3).

analysis, we avoid $n \geq 2$ because we assume the effect of the drive signal on these higher excitations is negligible. Therefore, increasing the speed of the gate (reducing the operation time) results in a wider box around the excitation frequencies. However, this increase is limited by the point at which higher excited states fall within these boxes, leading to excitations outside the computational space and a reduction in gate fidelity. To determine the permissible size of these boxes, we need to establish criteria to avoid overlap between the box edges and higher excited states.

Assuming f_i is the eigenenergy of the excited states for $i \geq 2$, we consider two cases based on the value of f_i relative to $2f_{10}$. First, if $f_i < 2f_{10}$, we must ensure that f_i falls within the green boundaries of 1 and 2 in Figure 8b. This requirement translates to the following conditions:

$$\begin{aligned} f_G &< f_i - f_1, \\ f_G &< 2f_1 - f_i. \end{aligned} \quad (18)$$

Second, if $f_i > 2f_{10}$, we must ensure that f_i exceeds the green boundary of 3 in Figure 8c. This condition can be expressed as

$$f_G < f_i - 2f_1. \quad (19)$$

Therefore, we conclude that the upper bound for f_G should be the minimum of all criteria given in (18) and (19) for $i \geq 2$:

$$\mathcal{G} = \min_{i \geq 2} (f_i - f_1, |f_i - 2f_1|).$$

E Diagonalization Convergence Test

When diagonalizing the Hamiltonian, the accuracy of the computed eigensystem is limited because we truncate the Hilbert space to a finite dimension. Moreover, during optimization we lack prior knowledge of the circuit's spectrum, so it is essential to establish a method to ensure the calculated results are trustworthy.

Assume the circuit has n_N modes. To construct a finite-dimensional Hamiltonian, we restrict to a product Hilbert space formed from the m_i lowest-energy eigenstates of each mode. (For harmonic modes, these are the Fock states $|0\rangle, \dots, |m_i - 1\rangle$; for charge modes these are the charge number states $|-(m_i - 1)/2\rangle, \dots, |0\rangle, \dots, |(m_i - 1)/2\rangle$.) We call each m_i the dimension of mode i . For a harmonic mode this is equivalent to its truncation number; for a charge mode with truncation number c_i this is given by $m_i = 2c_i - 1$ (see Appendix F).

The total size of the Hilbert space is therefore

$$D = \prod_{i=1}^{n_N} m_i. \quad (20)$$

Thus, the Hamiltonian $\hat{H}$ we use is a $D \times D$ matrix, representing a truncated version of the true infinite-dimensional Hamiltonian. Ideally, the lowest-energy eigenenergies of the truncated Hamiltonian are close to those of the infinite-dimensional system. Traditionally, to test this, one increases each m_i until the eigenvalues of interest stabilize. Once these eigenvalues no longer change, the diagonalization is considered converged. However, this iterative process is impractical during optimization. Therefore, an alternate method is required to verify the convergence of the eigenvalues and eigenvectors.

The eigenvalue problem for eigenenergy f and eigenstate $|f\rangle$ is given by

$$\hat{H} |f\rangle = f |f\rangle.$$

Now, suppose the dimension of each mode is increased from m_i to $m_i + t$, resulting in a Hilbert space of dimension D' . The eigenvalue problem for the corresponding $D' \times D'$ Hamiltonian $\hat{H}'$ is

$$\hat{H}' |f'\rangle = f' |f'\rangle.$$

If the Hamiltonian had already converged for the eigenvector so that $f = f'$, we expect that $|f'\rangle$ lies entirely in the original D -dimensional Hilbert space, and thus the components due to system expansion are zero.

To check if this is the case, we embed the eigenvector $|f\rangle$ of the smaller system in the D' -dimensional system by concatenating it with zeros to match the size of $|f'\rangle$, resulting in $|\tilde{f}\rangle$. We then define the convergence error by

$$\epsilon = 1 - |\langle \tilde{f} | f \rangle|^2. \quad (21)$$

The criteria for convergence of the eigenvalue f and eigenvector $|f\rangle$ is $\epsilon < \epsilon^*$. This has been implemented in **SQcircuit** and can be run on a diagonalized circuit as:

```
epsilon_star = 1e-5
circuit.check_convergence(
    t=2,
    threshold=epsilon_star
)
```

Here, **circuit** is an object of the **Circuit** class in **SQcircuit**, and the **check_convergence()** function returns a boolean indicating the convergence state of the second excited state. The value of ϵ^* should be chosen small enough to ensure desired accuracy.

F Automatic Truncation Assignment

In this appendix, we present a general algorithm to choose truncation numbers for charge and harmonic modes. The algorithm is designed to minimize the total dimension of the composite Hilbert space while maximizing the accuracy of computed results of interest. By combining this algorithm with the convergence test described in the previous appendix, we ensure that our search algorithm is both computationally efficient and maximizes the region of parameter space explored, given fixed computational resources.

As in the previous appendix, assume the circuit has n_N total modes, which are divided into n_H harmonic modes and n_C charge modes. We fix an upper bound K on the product of the modes' truncation numbers, which acts as a proxy for the total Hilbert space dimension and consequently for the total memory and computational resources required. For instance, the complexity of sparse diagonalization scales as $O(K^2)$. Following the implementation in **SQcircuit**, a harmonic mode with truncation number h_i retains Fock states $|0\rangle, \dots, |h_i - 1\rangle$, while a charge mode with truncation number c_j encompasses the $2c_j - 1$ charge states $|-(c_j - 1)\rangle, \dots, |c_j - 1\rangle$.

The algorithm discussed here outputs truncation numbers h_i for each harmonic mode and c_j for each

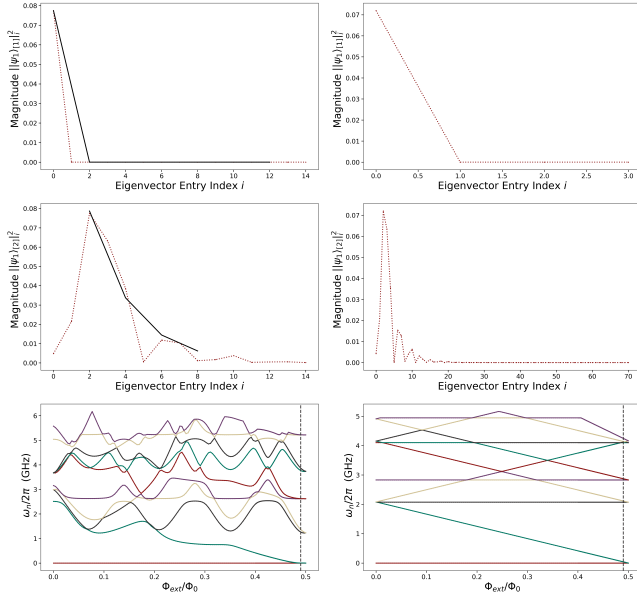


Figure 9: **Spectrum and eigenvector magnitudes for a "JLJL" circuit with truncation numbers allocated evenly (left column) or with the heuristic algorithm (right column).** In both cases, the product of truncation numbers is bounded by $K = 3000$. The top two rows show the mode magnitudes by index for each of the two harmonic modes present. The black lines in the first column are exponential fits to the eigenvector decay used in the heuristic algorithm. The bottom row shows the circuit spectrum for the lowest 10 states. On the left, stochasticity within the flux spectrum indicates that the assigned truncation numbers are insufficient to accurately compute the exact spectrum. On the right, the spectrum is converged below the same bound of K . The dashed black lines in the bottom row indicate the external flux used when applying the convergence test discussed in Appendix E.

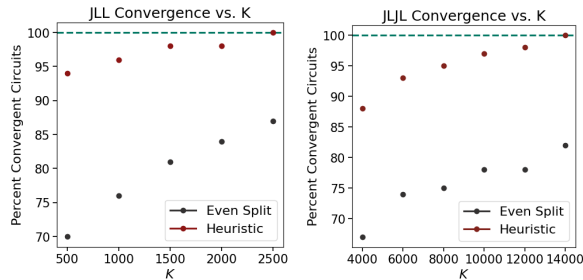


Figure 10: **Percentage of converged circuits at varying K using even allocation and the heuristic algorithm.** 100 random circuits with the "JLL" (left) and "JLJL" topologies (right) were sampled, truncation numbers allocated for each fixed K , and convergence checked with the test (Appendix E) and by eye.

Algorithm 1 Heuristic for truncation number assignment

Require: Hamiltonian $\hat{H}$ expressed in a basis of n_H harmonic and n_C charge modes with $n_N := n_H + n_C$. Bound K on the product of truncation numbers and maximum charge truncation number $c_{\max}$.

- 1: Assign equal truncation numbers $\lfloor K^{1/n_N} \rfloor$ to each mode and diagonalize $\hat{H}$.
- 2: **for** each charge mode j **do**
- 3: **if** charge-spread estimation enabled **then**
- 4: Compute effective $E_{C_j, \text{eff}}$ and $E_{J_j, \text{eff}}$ as in Eq. (25).
- 5: Calculate approximate width of ground state $\sigma_j = (E_{J_j, \text{eff}}/8E_{C_j, \text{eff}})^{1/4}$.
- 6: Set $c_j = \min(c_{\max}, \lfloor 3\sigma_j \rfloor)$.
- 7: **else**
- 8: Assign fixed charge truncation $c_j = c_{\max}$.
- 9: **end if**
- 10: **end for**
- 11: Compute remaining available Hilbert space dimension $K_h = K / \prod_{j=1}^{n_C} c_j$.
- 12: **for** each harmonic mode i **do**
- 13: For the highest eigenvector of interest $|\psi\rangle$, determine the highest index peak d_i in the eigenvector's squared magnitudes $|\langle \psi | i \rangle|^2$.
- 14: Fit an exponential decay curve of the form $f(x) \propto \exp(-\alpha_i(x - d_i))$ to the exponential decay in magnitudes following d_i .
- 15: Calculate preliminary truncation numbers $\{h_i\}$ using Eq. (29).
- 16: Shift each truncation number $h_i \leftarrow h_i + d_i$ and renormalize using Eq. (30).
- 17: **end for**
- 18: **return** Truncation numbers $\{c_j\}_{j=1}^{n_C}$ for charge modes and $\{h_i\}_{i=1}^{n_H}$ for harmonic modes.

charge mode satisfying

$$\left(\prod_{i=1}^{n_H} h_i\right) \left(\prod_{j=1}^{n_C} c_j\right) \leq K. \quad (22)$$

A naïve approach to this problem is to allocate truncation numbers evenly, by setting

$$h_i = c_j := \lfloor K^{1/n_N} \rfloor, \quad \text{for all } i, j. \quad (23)$$

Since we diagonalize the circuit in the tensor-product basis of weakly coupled modes, the probability density of low-energy eigenstates is concentrated in low-energy excitations of the distinct modes. However, depending on the circuit, these eigenstates may have more excitations in some modes than others. Uniformly setting truncation numbers allocates excessive computational resources to modes in which the eigenstates have relatively few excitations, and deprives computational resources from those modes in which the eigenstates have many.

To more efficiently allocate truncation numbers, we aim to estimate how much weight the eigenstates have in each mode. For charge modes, we have an explicit approximation. The Hamiltonian in **SQcircuit** for these modes takes the form

$$\hat{H}^{\text{ch}} = \frac{1}{2} \hat{Q}^{\text{ch},T} (C^{\text{ch}})^{-1} \hat{Q}^{\text{ch}} + \sum_{k \in \mathcal{S}_J} E_{J_k} \cos(\mathbf{w}_k^T \hat{\boldsymbol{\varphi}} + \mathbf{b}_k^T \boldsymbol{\varphi}_{\text{ext}}). \quad (24)$$

After a change of basis which diagonalizes $(C^{\text{ch}})^{-1}$, and expanding the cosine terms to second order, we have

$$\hat{H}^{\text{ch}} = \sum_{j=1}^{n_C} \left[4E_{C_j, \text{eff}} \hat{Q}_j^2 + E_{J_j, \text{eff}} \varphi_j^2 \right] + \sum_{k \in \mathcal{S}_J} A_{ij} \varphi_i \varphi_j, \quad (25)$$

where we have discarded gate charges and external fluxes. Further ignoring the coupling terms $A_{ij} \varphi_i \varphi_j$, this is equivalent to n_C uncoupled harmonic oscillators, with ground state of approximate width $\sigma := (E_{J, \text{eff}}/8E_{C, \text{eff}})^{1/4}$ in the number basis [23]. This estimate suggests choosing $c_j = \min(c_{\text{max}}, \lfloor \Delta \sigma_j \rfloor)$, where the number of standard deviations Δ retained determines the accuracy of calculated values. We implemented this option with $\Delta = 3$ and $c_{\text{max}} = 12$, and used this to produce the example in Fig. 9. For the optimization runs reported in Section 4, we instead assigned a fixed charge mode truncation number $c_j = c_{\text{max}} = 12$ (23 charge states) to circuits that also include harmonic modes. Circuits comprised only of charge modes used the even allocation of truncation numbers laid out in Eq. (23).

After choosing truncation numbers for the charge modes, the dimension of the harmonic mode subspace

is bounded by

$$K_h := K / \prod_{j=1}^{n_C} c_j \quad (26)$$

For the harmonic modes, we do not have an *a priori* estimate. However, because the Hamiltonian matrix has exponential off-diagonal decay, the eigenvectors generically have exponentially decaying weight in higher excitations for each mode [57]. By comparing the decay rates between different harmonic modes, we can infer the relative distribution of excitations between them.

We begin by uniformly allocating truncation numbers as in (23) and diagonalizing the circuit Hamiltonian. Given a state written in the product basis of modes $1, \dots, n_N$ as

$$|\psi\rangle = \sum_{l_1, \dots, l_{n_N}} c_{l_1 \dots l_{n_N}} |l_1\rangle \cdots |l_{n_N}\rangle,$$

we define the state's component in mode i by

$$||\psi\rangle_{[i]}|^2 := \sum_{l_i} \max_{l_1 \dots l_{i-1} l_{i+1} \dots l_{n_N}} \{|c_{l_1 \dots l_i \dots l_{n_N}}|^2\} |l_i\rangle. \quad (27)$$

This is a $\lfloor K^{1/n_N} \rfloor$ -length column vector, where the l_i th entry indicates the maximum weight of $|\psi\rangle$ in the l_i th excitation of the i th mode.

Using this definition, we extract the magnitudes $\{||\psi\rangle_{[i]}|^2\}_{i=1}^{n_H}$ for some proxy eigenvector $|\psi\rangle$, conventionally the first excited eigenstate. Typically, $||\psi\rangle_{[i]}|^2$ will be dominated by a small number of excitations, with an exponential decay away from them. We identify the largest index $d_i \in [1, \lfloor K^{1/n_N} \rfloor]$, corresponding to the highest-energy excitation in mode i , for which $||\psi\rangle_{[i]}|^2$ has a peak. A peak is defined by $||\psi\rangle_{[i]}|^2$ being lower for $l < d_i$ and $l > d_i$, as well as having a magnitude above a minimum threshold. Given that peak, we fit an exponential curve to the tail of the distribution, allowing us to extract the relative decay rates at the tail end of each mode.

Provided such a fit $||\psi\rangle_{[i]}|^2 \sim A_i e^{-\alpha_i(l-d_i)}$, we aim to allocate more mode numbers to circuits with smaller α_i because a slower decay indicates there are more excitations in that mode. To quantify the tradeoff, we minimize the convergence error ϵ defined in Equation (21). Assuming exponential decay, for large α then

$$\epsilon \sim \prod_{i=1}^{n_H} e^{-\alpha_i h_i}. \quad (28)$$

Minimizing this subject to the constraint $\prod_{i=1}^{n_H} h_i \leq K_h$ sets the truncation numbers to

$$h_i = \left\lceil \frac{1}{\alpha_i} \left(K_h \prod_{j=1}^{n_H} \alpha_j \right)^{1/n_H} \right\rceil. \quad (29)$$

While this accounts for the relative decay rates, we also need to account for the relative positions of each harmonic mode's peak d_i . To address this, we adjust each truncation number by the relative position of each mode's peak, taking $h_i \leftarrow h_i + d_i$. To ensure that constraint on dimension remains satisfied, we then renormalize by setting

$$h_i \leftarrow h_i \left(\frac{K_h}{\prod_{i=1}^{n_H} h_i} \right)^{1/n_H}. \quad (30)$$

This yields the final harmonic number cutoffs $\{h_i\}_{i=1}^{n_H}$, which are applied in conjunction with the charge cutoffs $\{c_j\}_{j=1}^{n_C}$. The overall process is summarized in Algorithm 1.

In practice, this heuristic algorithm represents a substantial improvement over the even allocation of truncation numbers defined by Eq. (23). One example is shown in Fig. 9, which demonstrates the improvement afforded by the heuristic allocation for a fixed $K = 3000$. The left half of the plot shows an even truncation allocation of $h_1 = h_2 = 14$, where the upper plot shows a faster decaying mode and the lower plot shows a relatively slower decaying mode. After applying the heuristic algorithm, a higher mode cutoff is allocated to the slower decaying harmonic mode leading to $h_1 = 4$ and $h_2 = 71$. For the same bound K , the flux spectra on the bottom row demonstrate a clear difference between the non-convergent eigenvalues on the left and the convergent values on the right.

To check the performance of this algorithm, we compared even allocation to the truncation numbers from Algorithm 1 by randomly sampling a set of 100 circuits of multiple circuit types and testing the proportion of convergent circuits produced by both. These results for circuits of type "JLL" (two harmonic modes) and "JLJL" (two harmonic modes and one charge mode) are presented in Fig. 10, which demonstrates that the heuristic algorithm produces a higher proportion of convergent circuits than the even distribution for a wide range of values for K . As this trend continues even to 100% convergence for a wide class of circuits, we have strong evidence that the heuristic generally helps allocate computational resources more efficiently and explore a wider swath of parameter space with the same computational constraints. In these tests, charge modes were assigned the same even split of K under both allocations, so this comparison isolates the effect of the heuristic algorithm used to assign harmonic modes.

References

- [1] András Gyenis, Pranav S. Mundada, Agustin Di Paolo, Thomas M. Hazard, Xinyuan You, David I. Schuster, Jens Koch, Alexandre Blais, and Andrew A. Houck. "Experimental realization of a protected superconducting circuit derived from the 0- π qubit". *PRX Quantum* **2**, 010339 (2021).
- [2] W. C. Smith, A. Kou, X. Xiao, U. Vool, and M. H. Devoret. "Superconducting circuit protected by two-cooper-pair tunneling". *npj Quantum Information* **6**, 8 (2020).
- [3] Leon Ding, Max Hays, Youngkyu Sung, Bharath Kannan, Junyoung An, Agustin Di Paolo, Amir H. Karamlou, Thomas M. Hazard, Kate Azar, David K. Kim, Bethany M. Niedzielski, Alexander Melville, Mollie E. Schwartz, Jonilyn L. Yoder, Terry P. Orlando, Simon Gustavsson, Jeffrey A. Grover, Kyle Serniak, and William D. Oliver. "High-fidelity, frequency-flexible two-qubit fluxonium gates with a transmon coupler". *Phys. Rev. X* **13**, 031035 (2023).
- [4] Pranav Mundada, Gengyan Zhang, Thomas Hazard, and Andrew Houck. "Suppression of qubit crosstalk in a tunable coupling superconducting circuit". *Phys. Rev. Appl.* **12**, 054023 (2019).
- [5] Nathan R.A. Lee, Yudan Guo, Agnetta Y. Cleland, E. Alex Wollack, Rachel G. Gruenke, Takuma Makihara, Zhaoyou Wang, Taha Rajabzadeh, Wentao Jiang, Felix M. Mayor, Patricio Arrangoiz-Arriola, Christopher J. Sarabalis, and Amir H. Safavi-Naeini. "Strong dispersive coupling between a mechanical resonator and a fluxonium superconducting qubit". *PRX Quantum* **4**, 040342 (2023).
- [6] Takuma Makihara, Nathan Lee, Yudan Guo, Wenyan Guan, and Amir Safavi-Naeini. "A parametrically programmable delay line for microwave photons". *Nature Communications* **15**, 4640 (2024).
- [7] Barbara M. Terhal. "Quantum error correction for quantum memories". *Rev. Mod. Phys.* **87**, 307–346 (2015).
- [8] Rajeev Acharya, Igor Aleiner, Richard Allen, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Juan Atalaya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Joao Basso, Andreas Bengtsson, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Dripto M. Debroy, Alexander Del Toro Barba, Sean Demura, Andrew Dunsworth, Daniel Eppens, Catherine Erickson, Lara Faoro, Edward Farhi, Reza Fatemi, Leslie Flores Burgos, Ebrahim Forati, Austin G.

- Fowler, Brooks Foxen, William Jiang, Craig Gidney, Dar Gilboa, Marissa Giustina, Alejandro Grajales Dau, Jonathan A. Gross, Steve Habegger, Michael C. Hamilton, Matthew P. Harrigan, Sean D. Harrington, Oscar Higgott, Jeremy Hilton, Markus Hoffmann, Sabrina Hong, Trent Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Pavol Juhas, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Tanuj Khattar, Mostafa Khezri, Mária Kieferová, Seon Kim, Alexei Kitaev, Paul V. Klimov, Andrey R. Klots, Alexander N. Korotkov, Fedor Kostritsa, John Mark Kreikebaum, David Landhuis, Pavel Laptev, Kim-Ming Lau, Lily Laws, Joonho Lee, Kenny Lee, Brian J. Lester, Alexander Lill, Wayne Liu, Aditya Locharla, Erik Lucero, Fionn D. Malone, Jeffrey Marshall, Orion Martin, Jarrod R. McClean, Trevor McCourt, Matt McEwen, Anthony Megrant, Bernardo Meurer Costa, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Shirin Montazeri, Alexis Morvan, Emily Mount, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, John Platt, Andre Petukhov, Rebecca Potter, Leonid P. Pryadko, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Michael J. Shearn, Aaron Shorter, Vladimir Shvarts, Jindra Skrzynny, Vadim Smelyanskiy, W. Clarke Smith, George Sterling, Doug Strain, Marco Szalay, Alfredo Torres, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraf Heidweiller, Theodore White, Cheng Xing, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Google Quantum AI. "Suppressing quantum errors by scaling a surface code logical qubit". *Nature* **614**, 676–681 (2023).
- [9] Youwei Zhao, Yangsen Ye, He-Liang Huang, Yiming Zhang, Dachao Wu, Huijie Guan, Qingling Zhu, Zuolin Wei, Tan He, Sirui Cao, Fusheng Chen, Tung-Hsun Chung, Hui Deng, Daojin Fan, Ming Gong, Cheng Guo, Shaojun Guo, Lianchen Han, Na Li, Shaowei Li, Yuan Li, Futian Liang, Jin Lin, Haoran Qian, Hao Rong, Hong Su, Lihua Sun, Shiyu Wang, Yulin Wu, Yu Xu, Chong Ying, Jiale Yu, Chen Zha, Kaili Zhang, Yong-Heng Huo, Chao-Yang Lu, Cheng-Zhi Peng, Xiaobo Zhu, and Jian-Wei Pan. "Realization of an error-correcting surface code with superconducting qubits". *Phys. Rev. Lett.* **129**, 030501 (2022).
- [10] Sebastian Krinner, Nathan Lacroix, Ants Remm, Agustin Di Paolo, Elie Genois, Catherine Leroux, Christoph Hellings, Stefania Lazar, Francois Swiadek, Johannes Herrmann, Graham J. Norris, Christian Kraglund Andersen, Markus Müller, Alexandre Blais, Christopher Eichler, and Andreas Wallraff. "Realizing repeated quantum error correction in a distance-three surface code". *Nature* **605**, 669–674 (2022).
- [11] Zhaoyou Wang, Taha Rajabzadeh, Nathan Lee, and Amir H Safavi-Naeini. "Automated discovery of autonomous quantum error correction schemes". *PRX Quantum* **3**, 020302 (2022).
- [12] Ziqian Li, Tanay Roy, David Rodríguez Pérez, Kan-Heng Lee, Eliot Kapit, and David I. Schuster. "Autonomous error correction of a single logical qubit using two transmons". *Nature Communications* **15**, 1681 (2024).
- [13] Taha Rajabzadeh, Zhaoyou Wang, Nathan Lee, Takuma Makihara, Yudan Guo, and Amir H. Safavi-Naeini. "Analysis of arbitrary superconducting quantum circuits accompanied by a Python package: SQcircuit". *Quantum* **7**, 1118 (2023).
- [14] Andrew J. Kerman. "Efficient numerical simulation of complex Josephson quantum circuits" (2020). [arXiv:2010.14929](https://arxiv.org/abs/2010.14929).
- [15] W. C. Smith, A. Kou, U. Vool, I. M. Pop, L. Frunzio, R. J. Schoelkopf, and M. H. Devoret. "Quantization of inductively shunted superconducting circuits". *Physical Review B* **94**, 144507 (2016).
- [16] Guido Burkard, Roger H. Koch, and David P. DiVincenzo. "Multilevel quantum description of decoherence in superconducting qubits". *Physical Review B* **69**, 064503 (2004).
- [17] Bernard Yurke and John S. Denker. "Quantum network theory". *Physical Review A* **29**, 1419–1437 (1984).
- [18] Andrew Osborne, Trevyn Larson, Sarah Garcia Jones, Ray W. Simmonds, András Gyenis, and Andrew Lucas. "Symplectic geometry and circuit quantization". *PRX Quantum* **5**, 020309 (2024).
- [19] I. L. Egusquiza and A. Parra-Rodriguez. "Algebraic canonical quantization of lumped superconducting networks". *Phys. Rev. B* **106**, 024510 (2022).
- [20] A. Parra-Rodriguez and I. L. Egusquiza. "Geometrical description and Faddeev-Jackiw quantization of electrical networks". *Quantum* **8**, 1466 (2024).
- [21] Uri Vool and Michel Devoret. "Introduction to quantum electromagnetic circuits". *International Journal of Circuit Theory and Applications* **45**, 897–934 (2017).

- [22] Mario F Gely and Gary A Steele. “QuCAT: quantum circuit analyzer tool in Python”. *New Journal of Physics* **22**, 013025 (2020).
- [23] Peter Groszkowski and Jens Koch. “Scqubits: a Python package for superconducting qubits”. *Quantum* **5**, 583 (2021).
- [24] Sai Pavan Chitta, Tianpu Zhao, Ziwen Huang, Ian Mondragon-Shem, and Jens Koch. “Computer-aided quantization and numerical analysis of superconducting circuits”. *New Journal of Physics* **24**, 103020 (2022).
- [25] Philipp Aumann, Tim Menke, William D Oliver, and Wolfgang Lechner. “CircuitQ: an open-source toolbox for superconducting circuits”. *New Journal of Physics* **24**, 093012 (2022).
- [26] Tim Menke, Florian Häse, Simon Gustavsson, Andrew J. Kerman, William D. Oliver, and Alán Aspuru-Guzik. “Automated design of superconducting circuits and its application to 4-local couplers”. *npj Quantum Information* **7**, 49 (2021).
- [27] Xiaotong Ni, Hui-Hai Zhao, Lei Wang, Feng Wu, and Jianxin Chen. “Integrating quantum processor device and control optimization in a gradient-based framework”. *npj Quantum Information* **8**, 106 (2022).
- [28] Xinyuan You, J. A. Sauls, and Jens Koch. “Circuit quantization in the presence of time-dependent external flux”. *Physical Review B* **99**, 174512 (2019).
- [29] R. P. Riwar and D. P. DiVincenzo. “Circuit quantization with time-dependent magnetic fields for realistic geometries”. *npj Quantum Information* **8**, 36 (2022).
- [30] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization” (2014). [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- [31] David F Shanno. “Conditioning of quasi-Newton methods for function minimization”. *Mathematics of computation* **24**, 647–656 (1970).
- [32] Jorge Nocedal. “Updating quasi-Newton matrices with limited storage”. *Mathematics of Computation* **35**, 773–782 (1980).
- [33] J. Kennedy and R. Eberhart. “Particle swarm optimization”. In Proceedings of ICNN’95 - International Conference on Neural Networks. Volume 4, pages 1942–1948. (1995).
- [34] Donald R Jones, Matthias Schonlau, and William J Welch. “Efficient global optimization of expensive black-box functions”. *Journal of Global Optimization* **13**, 455–492 (1998).
- [35] Richard S. Sutton and Andrew G. Barto. “Reinforcement learning: An introduction”. MIT Press. Cambridge, MA, USA (2018). url: <http://incompleteideas.net/book/the-book-2nd.html>.
- [36] Mahyar Abbasian, Taha Rajabzadeh, Ahmadreza Moradipari, Seyed Amir Hossein Aqajari, Hongsheng Lu, and Amir M. Rahmani. “Controlling the Latent Space of GANs through Reinforcement Learning: A Case Study on Task-based Image-to-Image Translation”. In Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing. Page 1061–1063. New York, NY, USA (2024). Association for Computing Machinery.
- [37] Ahmadreza Moradipari, Mohammad Ghavamzadeh, Taha Rajabzadeh, Christos Thrampoulidis, and Mahnoosh Alizadeh. “Multi-environment meta-learning in stochastic linear bandits”. In 2022 IEEE International Symposium on Information Theory (ISIT). Pages 1659–1664. (2022).
- [38] Nikolaus Hansen and Andreas Ostermeier. “Completely Derandomized Self-Adaptation in Evolution Strategies”. *Evolutionary Computation* **9**, 159–195 (2001).
- [39] Mathew M. Noel. “A new gradient based particle swarm optimization algorithm for accurate computation of global minimum”. *Applied Soft Computing* **12**, 353–359 (2012).
- [40] Jian Wu, Matthias Poloczek, Andrew G Wilson, and Peter Frazier. “Bayesian optimization with gradients”. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems. Volume 30. Curran Associates, Inc. (2017). url: https://proceedings.neurips.cc/paper_files/paper/2017/file/64a08e5f1e6c39faeb90108c430eb120-Paper.pdf.
- [41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems. Volume 32. Curran Associates, Inc. (2019). url: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.
- [42] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake

- VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. “JAX: composable transformations of Python+NumPy programs”. <http://github.com/google/jax> (2018). Version 0.3.13.
- [43] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems” (2016). [arXiv:1603.04467](https://arxiv.org/abs/1603.04467).
- [44] Eli J. Weisler, Mohit Bhat, Zhenxing Liu, and Joshua Combes. “Enumeration of all superconducting circuits up to 5 nodes” (2024). [arXiv:2410.18497](https://arxiv.org/abs/2410.18497).
- [45] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerín, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. “Quantum supremacy using a programmable superconducting processor”. *Nature* **574**, 505–510 (2019).
- [46] Helin Zhang, Srivatsan Chakram, Tanay Roy, Nathan Earnest, Yao Lu, Ziwen Huang, D. K. Weiss, Jens Koch, and David I. Schuster. “Universal fast-flux control of a coherent, low-frequency qubit”. *Phys. Rev. X* **11**, 011010 (2021).
- [47] Ioan M. Pop, Kurtis Geerlings, Gianluigi Cate-lani, Robert J. Schoelkopf, Leonid I. Glazman, and Michel H. Devoret. “Coherent suppression of electromagnetic dissipation due to superconducting quasiparticles”. *Nature* **508**, 369–372 (2014).
- [48] Fei Yan, Simon Gustavsson, Archana Kamal, Jeffrey Birenbaum, Adam P. Sears, David Hover, Ted J. Gudmundsen, Danna Rosenberg, Gabriel Samach, S. Weber, Jonilyn L. Yoder, Terry P. Orlando, John Clarke, Andrew J. Kerman, and William D. Oliver. “The flux qubit revisited to enhance coherence and reproducibility”. *Nature Communications* **7**, 12964 (2016).
- [49] D. Vion, A. Aassime, A. Cottet, P. Joyez, H. Pothier, C. Urbina, D. Esteve, and M. H. Devoret. “Manipulating the quantum state of an electrical circuit”. *Science* **296**, 886–889 (2002).
- [50] J. Q. You and Franco Nori. “Pokemon: Protected logic qubit derived from the $0-\pi$ qubit” (2021). [arXiv:2112.03584](https://arxiv.org/abs/2112.03584).
- [51] Konstantin Kalashnikov, Wen Ting Hsieh, Wenyuan Zhang, Wen-Sen Lu, Plamen Kamenov, Agustin Di Paolo, Alexandre Blais, Michael E. Gershenson, and Matthew Bell. “Bifluxon: Fluxon-parity-protected superconducting qubit”. *PRX Quantum* **1**, 010307 (2020).
- [52] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. “A quantum engineer’s guide to superconducting qubits”. *Applied Physics Reviews* **6**, 021318 (2019).
- [53] Philip Reinhold, Serge Rosenblum, Wen-Long Ma, Luigi Frunzio, Liang Jiang, and Robert J. Schoelkopf. “Error-corrected gates on an encoded qubit”. *Nature Physics* **16**, 822–826 (2020).
- [54] James D. Teoh, Patrick Winkel, Harshvardhan K. Babla, Benjamin J. Chapman, Jahan Claes, Stijn J. de Graaf, John W. O. Garmon, William D. Kalfus, Yao Lu, Aniket Maiti, et al. “Dual-rail encoding with superconducting cavities”. *Proceedings of the National Academy of Sciences* **120**, e2221736120 (2023).
- [55] H. Levine, A. Haim, J. S. C. Hung, N. Ali-doust, M. Kalaei, L. DeLorenzo, E. A. Wollack, P. Arrangoiz-Arriola, A. Khalajhedayati, R. Sanil, H. Moradinejad, Y. Vaknin, A. Kubica, D. Hover, S. Aghaieimbodi, J. A. Alcid, C. Baek, J. Barnett, K. Bawdekar, P. Bienias, H. A. Carson, C. Chen, L. Chen, H. Chinke-zian, E. M. Chisholm, A. Clifford, R. Cosmic, N. Crisosto, A. M. Dalzell, E. Davis, J. M. D’Ewart, S. Diez, N. D’Souza, P. T.

Dumitrescu, E. Elkhoully, M. T. Fang, Y. Fang, S. Flammia, M. J. Fling, G. Garcia, M. K. Gharzai, A. V. Gorshkov, M. J. Gray, S. Grimberg, A. L. Grimsmo, C. T. Hann, Y. He, S. Heide, S. Howell, M. Hunt, J. Iverson, I. Jarrige, L. Jiang, W. M. Jones, R. Karabalin, P. J. Karalekas, A. J. Keller, D. Lasi, M. Lee, V. Ly, G. MacCabe, N. Mahuli, G. Marcaud, M. H. Matheny, S. McArdle, G. McCabe, G. Merton, C. Miles, A. Milsted, A. Mishra, L. Monceli, M. Naghiloo, K. Noh, E. Oblepias, G. Ortuno, J. C. Owens, J. Pagdilao, A. Panduro, J.-P. Paquette, R. N. Patel, G. Peairs, D. J. Perello, E. C. Peterson, S. Ponte, H. Putterman, G. Refael, P. Reinhold, R. Resnick, O. A. Reyna, R. Rodriguez, J. Rose, A. H. Rubin, M. Runyan, C. A. Ryan, A. Sahmoud, T. Scaffidi, B. Shah, S. Siavoshi, P. Sivarajah, T. Skogland, C.-J. Su, L. J. Swenson, J. Sylvia, S. M. Teo, A. Tomada, G. Torlai, M. Wistrom, K. Zhang, I. Zuk, A. A. Clerk, F. G. S. L. Brandão, A. Retzker, and O. Painter. “Demonstrating a long-coherence dual-rail erasure qubit using tunable transmons”. *Phys. Rev. X* **14**, 011051 (2024).

- [56] Agustin Di Paolo, Thomas E Baker, Alexandre Foley, David Sénéchal, and Alexandre Blais. “Efficient modeling of superconducting quantum circuits with tensor networks”. *npj Quantum Information* **7**, 1–11 (2021).
- [57] Michele Benzi and Valeria Simoncini. “Exploiting hidden structure in matrix computations: Algorithms and applications”. *Springer*. (2016).