Quantum Information Processing (2025) 24:184
https://doi.org/10.1007/s11128-025-04759-w

ArtA: automating Design Space Exploration of spin-qubit architectures

Nikiforos Paraskevopoulos[1,2] · David Hamel[1,2] · Aritra Sarkar[1,2] · C. G. Almudever[3] · Sebastian Feld[1,2]

Received: 31 July 2024 / Accepted: 25 April 2025

Abstract

In the fast-paced field of quantum computing, identifying the architectural characteristics that will enable quantum processors to achieve high performance across a diverse range of quantum algorithms continues to pose a significant challenge. Given the extensive and costly nature of experimentally testing different designs, this paper introduces the first Design Space Exploration (DSE) for quantum-dot spin-qubit architectures. Utilizing the upgraded *SpinQ* compilation framework, this study explores a substantial design space comprising 29,312 spin-qubit-based architectures and applies an innovative optimization tool, ArtA (***Art***ificial ***A***rchitect), to speed up the design space traversal. ArtA can leverage 17 optimization configurations, significantly reducing exploration times by up to 99.1% compared to a traditional brute force approach while maintaining the same result quality. After a comprehensive evaluation of best-matching optimization configurations per quantum circuit, ArtA suggests specific as well as universal architectural features that provide optimal performance across the examined circuits. Our work demonstrates that combining DSE methodologies with optimization algorithms can be effectively used to generate meaningful design insights for quantum processor development.

✉ Nikiforos Paraskevopoulos
N.Paraskevopoulos@tudelft.nl

David Hamel
davidhamelnl@gmail.com

Aritra Sarkar
aritra.sarkar@live.com

C. G. Almudever
cargara2@disca.upv.es

Sebastian Feld
s.feld@tudelft.nl

[1] Quantum and Computer Engineering Department, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

[2] QuTech, Delft University of Technology, Lorentzweg 1, 2628 CJ Delft, The Netherlands

[3] Computer Engineering Department, Universitat Politècnica de València, Camino de Vera, 46022 Valencia, Spain

Published online: 17 June 2025

Keywords Design Space Exploration · Quantum dots · Spin-qubit architectures · Optimization algorithms · Compilation · Quantum processing unit design

1 Introduction

As the field of quantum computing rapidly advances, different qubit technologies exhibit unique hardware and performance characteristics. It still remains to be seen which one (e.g., superconducting, trapped ions, quantum dots, photonics, defect based on nitrogen vacancy diamond centers) will succeed in scaling up quantum computing systems with high-quality qubits [1, 2]. Among them, spin qubits in quantum dots emerge as a compelling avenue for achieving scalability for practical quantum computation [2]. To this day, spin qubits are at an early stage in their development, with Intel's Tunnel Falls chip currently boasting the highest count of twelve spin qubits [3]. Despite this, their inherent scalability advantages suggest that a robust two-dimensional design [4–10] could be scaled up relatively easily once fabrication techniques and quality improve up to a certain level. Designing a chip, however, involves multiple architectural design choices whose impact in the future can only be fully assessed through experimental studies post-scaling the technology. Exploring all these design possibilities simultaneously can be expensive, time intensive, and impractical. Therefore, simulating how different design choices can affect performance in a range of quantum applications is crucial to assist the development. This process facilitates highlighting architectural insights that can guide the technology forward and will allow quantum researchers to make more informed decisions, streamline efforts, and hasten the development of these devices.

Recognizing the intricate challenges of designing quantum processor architectures, this study initiates the first Design Space Exploration (DSE) for quantum-dot spin-qubit architectures, both from a current technological perspective and a future one. Therefore, in this work, we have identified a wide range of representative architectural features and abstracted them into usable input variables, resulting in 29, 312 different architectures. To facilitate this exploration, we have enhanced the compilation capabilities of *SpinQ* compilation framework [11] to handle all these input variables and updated the definition of the Estimated Success Probability (ESP) metric to include not only operational errors but also crosstalk and decoherence errors. These transformations, and many more, establish *SpinQ* as the first compilation and DSE framework for spin-qubit architectures.

Standard DSE processes [12, 13], however, will be impractical time-wise, especially for such large spaces, as they rely on a brute force approach to traverse and subsequently analyze the design space. To address this, our approach employs ArtA (***Art***ificial ***A***rchitect), a built-in tool containing multiple optimization methods for automating the DSE process of *SpinQ*, thus taking less time than the brute force approach with the same quality of results. The abilities of ArtA are (a) to suggest which of the seventeen optimization configurations can find the architecture with the desired ESP the fastest compared to the brute force approach and (b) which architectural design characteristics are key for building high-performance spin-qubit devices per quantum circuit.

Our results demonstrate ArtA's ability to compare all optimization techniques for each quantum circuit and obtain a solution up to 99.1% faster, on average, than brute forcing. Then, equipped with these insights, we move on to conduct a DSE analysis of 29, 312 spin-qubit architectures and provide valuable insights into best design practices. Firstly, we highlight the critical need to maximize the parallelization of quantum gates rather than minimizing crosstalk between qubits. Secondly, we find that the communication method via shuttle operations is achieving higher ESP in large-scale circuits than SWAPs, but for single-qubit gates, pulse-based rotations are preferred over shuttle-based ones. Finally, a universal recommendation based on all tested circuits indicates that the likelihood of success increases when prioritizing the parallelization of single-qubit gates over two-qubit gates. This demonstrates that the combination of DSE techniques and optimization algorithms can effectively guide quantum processor designers with meaningful recommendations in the current and future stages of spin-qubit technology. Notably, we also observe that certain quantum circuits achieve optimal performance even with reduced hardware capabilities. This highlights the capability of our approach to uncover optimal architectural configurations without relying on highly complex or resource-intensive hardware designs.

The main contributions of this paper are:

1. The design space definition which comprises five main characteristics of current and, potentially, future spin-qubit architectures.
2. The upgraded *SpinQ*, the first compiler and DSE framework for spin-qubit architectures. In this version, we have updated the ESP formula to include operational, crosstalk, and decoherence-induced errors.
3. ArtA, the first tool consisting of seventeen optimization method configurations automating the DSE process of spin-qubit architectures.
4. Evaluation of best-matching optimization configurations per quantum circuit.
5. Evaluation of best-matching architectural characteristics per quantum circuit and universally (i.e., for all used circuits) in terms of ESP.

The remainder of this paper is structured as follows: In Sect. 2, we discuss the scalability potential of quantum-dot spin qubits as well as the importance of addressing engineering challenges while taking into account the performance of the resulting architecture. Then, in Sect. 3, we motivate the need for an automated DSE analysis for spin-qubit devices and formulate four research questions for this work. After that, in Sect. 4, we define the design space consisting of various input architectural variables and establish the new functionalities of *SpinQ*. Another upgrade in *SpinQ* is the enhanced ESP metric, described in Sect. 5, which is used as the figure of merit for our exploration. In Sect. 6, we introduce ArtA and two new metrics to evaluate its performance. In the results Sect. 7, we first evaluate ArtA's performance across all used quantum circuits and determine the best optimization method configuration. Then, we conduct a detailed analysis of the best architectural designs for each quantum circuit and form valuable insights. We conclude our work and provide ideas for future directions in Sect. 8.

2 Challenges for scaling up spin-qubit devices

Spin-qubit technologies are distinguished by their unique physical features, which position them as a highly scalable solution for quantum computing. The advantages of spin qubits include a significantly smaller size—up to a thousand times less than other qubit technologies—combined with decades of semiconductor manufacturing expertise, long coherence times coupled with short gate durations, and high operational temperatures [2, 4, 6, 14–20]. At the core of this technology lies the quantum dot, which can contain a trapped electron(s) or hole(s) to form a physical qubit [21]. Spin qubits are manipulated electromagnetically using multiple precision-engineered gate electrodes that facilitate either single- or two-qubit operations through exact timing of pulse sequences across various quantum-dot configurations. Studies have expanded these systems into one-dimensional and two-dimensional arrays [6–10], exploring different structures and material combinations.

Despite the mentioned advantages, spin-qubit quantum processors are not as advanced as other qubit technologies in terms of qubit counts and device availability. Major technological hurdles are related to the so-called interconnect bottleneck [4] and various fabrication challenges toward scaling up [22–25]. On the upside, there have been significant efforts [4, 5, 26–32] to tackle these challenges.

However, solving them can not guarantee successful quantum algorithm executions. This is because the quality and quantity of qubits are not the only factors determining a high-performing quantum processor. The architectural constraints qubits need to comply with in *how* they are operated are equally important. For instance, the benefits of qubits with excellent operational fidelity can easily be outweighed by low qubit connectivity and limited natively supported quantum gates. Similarly, high-quality qubits with low crosstalk interference are not enough when they can not be addressed in parallel. These, and many more, are interlinked architectural trade-offs that affect the actual performance of the quantum processors.

During the Noisy Intermediate-Scale Quantum (NISQ) era [33], predicting which architectural features and trade-offs will facilitate successful quantum circuit executions while maintaining reasonable hardware requirements remains challenging. To date, there has not been a systematic study exploring spin-qubit architectures through this prism of providing concrete guidelines for future development. Therefore, in this study, we exploit such an opportunity and alleviate the need for time-consuming, expensive, and technology-dependent experimental studies.

3 Problem statement

The need to apply DSE techniques for designing and optimizing full-stack quantum computing systems, or components thereof, has been emphasized in previous work [34–36]. These techniques and similar ones have been successfully applied across different levels of the quantum computing stack which predominantly rely on brute force exploration (i.e., exhaustive search). This is largely due to the inherently small design spaces in current quantum computing systems, which renders brute force evaluation computationally feasible. For instance, DSE methodologies have been used to explore

ion-trap quantum processors [37] and evaluate compilation techniques [38] and qubit connectivities [39–41] for superconducting qubits. Such techniques can abstractly convert architectural characteristics into design variables and, in this way, quantize the design space, expressing current and future design possibilities that otherwise would be impossible or too time-consuming to physically realize. Starting from there [12, 13], the steps for a proper DSE: i) Describe the problem in terms of input variables or parameters (design choices), ii) select the performance metrics, iii) choose a global cost function as a figure of merit that combines different performance metrics, and iv) find models (behavioral, analytical, from experimental data) that connect input parameters with metrics or directly to the figure of merit. Applying these DSE techniques allows researchers to uncover performance trends, model current and future multidimensional design spaces, and identify optimal design points across a wide range of application use cases.

Although this methodology is well established and understood in many disciplines, it can be challenging to implement in such early stages for spin-qubit technologies. One of the most difficult aspects is the development of a simulation framework incorporating a range of representative architectural variables with a large-scale perspective. Firstly, it is difficult to predict which architectural features will be relevant for many generations of devices to come, and secondly, the framework itself has to be flexible to incorporate new ones easily.

In practice, performing exhaustive exploration can be highly time-consuming; thus, even in classical systems, the design space is typically constrained to a manageable size by applying simplifying technological assumptions. While optimization algorithms have been successfully employed to efficiently explore significantly larger design spaces in classical computing domains [42–49], such techniques remain largely unexplored in quantum computing DSE applications. As spin-qubit technology rapidly advances, the design space continues to grow with new considerations, making earlier explorations quickly outdated. Even assuming a brute force approach already exists, the exponential growth in possible configurations can soon make it impractical to evaluate thousands of designs. Therefore, there is a clear need for an automated framework capable of efficiently managing this DSE process, significantly reducing the exploration time across current and future spin-qubit architectures.

To capture the challenges of the above problem statement, we ask the following four research questions:

1. What characteristics of spin-qubit architectures are representative, and how can they be incorporated in a DSE framework? Answered in Sect. 4
2. How can a DSE framework for investigating preferred spin-qubit architectural characteristics be constructed? Answered in Sect. 4
3. Which optimization methods are most effective for automating the DSE process for each quantum circuit individually, as well as across all used circuits collectively? Answered in Sect. 7.1
4. Which architectural characteristics in the selected design space are key for building high-performance spin-qubit devices for the specifically given quantum circuits? Answered in Sect. 7.2

Table 1 Summary of values of the considered architecture variables

Variable	Values	
xy_z	0 → No parallelization 1 → Parallelization	**Sec. IV.A**
xy_tqg	idem	
z_tqg	idem	
xy_z_tqg	idem	
xyD	-1 → Non applicable 1 → No parallelization 25 → X/Y gate parallelization capped at 25% of qubits per cycle 50 → idem, 75 → idem, 100 → idem	
zD	idem	
tqgD	1 → No parallelization 25 → two-qubit gate parallelization capped at 25% of qubits/2 per cycle 50 → idem, 75 → idem, 100 → idem	
sD	1 → No parallelization 25 → Shuttle parallelization capped at 25% of qubits per cycle 50 → idem, 75 → idem, 100 → idem	
single_qubit_impl	0 → Sequential 1 → Local 2 → Global 3 → Semi-global	
z_rot_impl	0 → Shuttle-based 1 → Pulse-based	**Sec. IV.B**
router	0 → *Shuttle-based SWAP* router 1 → *beSnake* router	**Sec. IV.C**
degree	4 → Square grid connectivity 6 → One extra diagonal 8 → Two extra diagonals	**Sec. IV.D**
SWAP_opt	0 → SWAP replacement used 1 → SWAP replacement is not used	**Sec. IV.E**

4 *SpinQ* and the design space

To answer the first two research questions, we conducted an analysis of relevant spin-qubit architectural properties with a long-term view and distilled them into numerous variables. These properties were selected to be representative of spin-qubit devices as well as to be able to form performance-related trade-offs between them. The latter will prevent "maximizing" one without causing a performance penalty through another (refer to Sect. 5). The defined design space is summarized in Table 1, including their possible values. An architecture is then characterized by one (valid) combination of variable values. Below, we describe the considered design space analytically. It should be noted that the DSE analysis in this work uses either discrete or categorical variables

[50], as we aim to provide performance trends and rankings between optimization methods and architectures.

4.1 Operational gate constraints

The set of gate constraints can be vast; therefore, we decided to abstract them into the following variables that affect which (a) gate types can be executed simultaneously and (b) how many of them can be parallelized at the same time step.

a) *Combinations of gate types that can be parallelized:*
(`xy_z`, `xy_tqg`, `z_tqg`, `xy_z_tqg` = *0, 1*)
As the names suggest, these variables are boolean and signify whether specific gate types can be executed in parallel. For example, `xy_z` defines if X or Y gates can be executed simultaneously with Z gates.

b) *Constraints affecting the parallelization per gate type expressed in percentage*:
(`xyD`, `zD`, `tqgD`, `sD` = *-1, 1, 25%, 50%, 75%, 100%*)
These variables indicate how many gates of each type can be parallelized, expressed as a percentage relation to the total number of qubits. For example, `xyD` = 25 in a 40-qubit architecture means that at most ten X or Y gates can exist in a single cycle. Note that the notion of a cycle refers to the basic unit of time representing one step in a sequence of gates of a quantum circuit, and each step may contain multiple gates. The `zD` and `sD` variables indicate the same for Z rotations and shuttle operations, respectively. The `tqgD` variable indicates the parallelization amount for two-qubit gates. As these always involve two qubits, they are expressed as a percentage of half of the total number of qubits. A value of -1 means that the parallelization degree is not user defined but dictated by a specific single-qubit gate implementation explained next.

c) *Single-qubit gate implementation:*
(`single_qubit_impl` = *0* $\rightarrow$ *Sequential, 1* $\rightarrow$ *Local, 2* $\rightarrow$ *Global, 3* $\rightarrow$ *Semi-global*)
This variable indicates in what manner single-qubit gates are carried out. A *Global* single-qubit gate implementation means the same rotation axis and angle are applied to all qubits. For qubits that do not participate in a cycle, locally applied disable instructions prevent their rotation. Then, the *Semi-Global* implementation is considered with the same rotation scheme as in [51] and [11], which is essentially equivalent to the implementations of the crossbar architecture proposed in [5]. In this implementation, qubit rotations are implemented semi-globally, meaning that either all qubits in odd or even column parities can be rotated at a time with the same axis and angle, and unwanted qubit rotations have to be reversed afterward by additional instructions [11]. A Local implementation enables arbitrary parallel execution of single-qubit gates, allowing any combination of rotation axes and angles to be applied simultaneously within a single cycle. Finally, a *Sequential* single-qubit gate implementation does not allow parallelizations, meaning that every cycle will be occupied by one gate only. It should be noted that depending on the `single_qubit_impl` value, only certain combinations of categories (a) and (b) are possible. For instance, in the *Global* and *Semi-Global* implemen-

tations, the `xyD` variable is not applicable as these implementations predetermine the upper limit of single-qubit gates allowed during a cycle.

4.2 Implementation of Z rotations

(`z_rot_impl` = *0, 1*)

This boolean variable stipulates the hardware-level implementation of Z rotation gates. The first option is a high-fidelity shuttle-based Z rotation, which is achieved with two time-sensitive qubit shuttles to and from a neighboring column [5, 11, 51, 52]. This single-qubit manipulation belongs to the so-called "hopping spins" type [53], and it is possible to achieve an arbitrary rotation axis, but in this work, we only assume the aforementioned variety for simplicity. The second option is a regular pulse-based Z rotation, where Z gates are implemented similarly to X/Y gates by magnetic pulse interactions. With the first choice being a unique characteristic of spin qubits, we are interested in exploring which one of the two provides better performance across many quantum circuits. Similar to the variables mentioned earlier, only certain combinations are possible depending on `single_qubit_impl` values. For example, with a *Semi-Global* single-qubit gate implementation and a pulse-based Z rotation gate, the `zD` variable is not applicable for the same reason the `xyD` variable is not applicable, as we explained before in category (b).

4.3 Connectivity of the device

(`degree` = *4, 6, 8*)

This variable stipulates the considered coupling graphs by addressing them through their average node degree. A coupling graph is a graph of possible direct interactions (edges) between qubits (nodes). The first coupling graph we used is a square grid (`degree` = 4). The second is identical to the first but adds one diagonal connection in each square (i.e., each qubit is coupled with six neighboring qubits and with three at the corners, `degree` = 6). Finally, the third adds the second diagonal in each square (i.e., each qubit is coupled with eight neighboring qubits and with three at the corners, `degree` = 8). Figure 1a presents the coupling graphs considered with color-coded edges for each of the three types. The actual dimensions of each coupling graph, and thus the overall size of the quantum architecture, are determined by the number of qubits required by the quantum algorithm under execution.

To further ease the understanding of the architectural landscape of the design space, in Fig. 1b, we depict the crossbar architecture [5] as a real-world example of an architecture represented in the design space. In this figure, the operational lines confine sixteen quantum dots, eight of which are occupied by spin qubits. The operational gate constraints in this architecture arise from the shared control lines and other physical properties as described in [5, 11, 51, 52]. Among these constraints, the coupling connectivity is particularly relevant, as it corresponds to a square grid with `degree` = 4 in this implementation. Consequently, the topology of the crossbar architecture can be abstracted as a two-dimensional grid, as depicted in Fig. 1c.

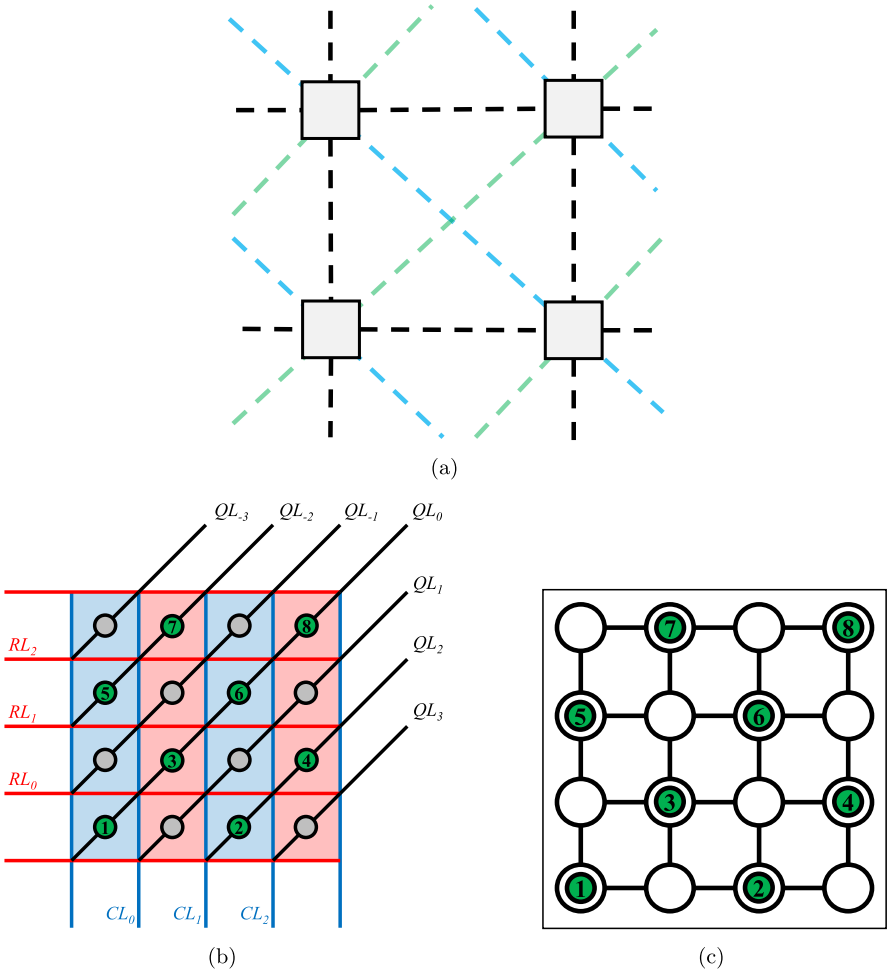

Fig. 1 **a** Different coupling graphs considered in the design space. The coupling graph consists of the black line edges when `degree` = *4*, black and green edges when `degree` = *6*, and all edges when `degree` = *8*. **b** Schematic overview of the crossbar architecture, taken from [11], with various shared operational control lines: vertical (column line, CL), horizontal (row line, RL), and diagonal (qubit line, QL). These lines are used with precise pulse sequences and shared among multiple sites, sixteen quantum dots in this figure, to perform operations on qubits [5, 51, 52] under specific operational constants. Here, the qubits (green circles with numbers) are initialized in a checkerboard pattern. **c** Abstraction of the crossbar architecture taken from [54] representing the coupling graph between qubits with `degree` = 4. Each circle represents a quantum dot, and each edge represents a coupling link signifying allowed interactions

We have now reviewed all variables that pertain solely to the hardware. Next, we examine two additional variables that influence the compilation process. Specifically, these variables can alter the routing methodology based on the underlying hardware communication constraints.

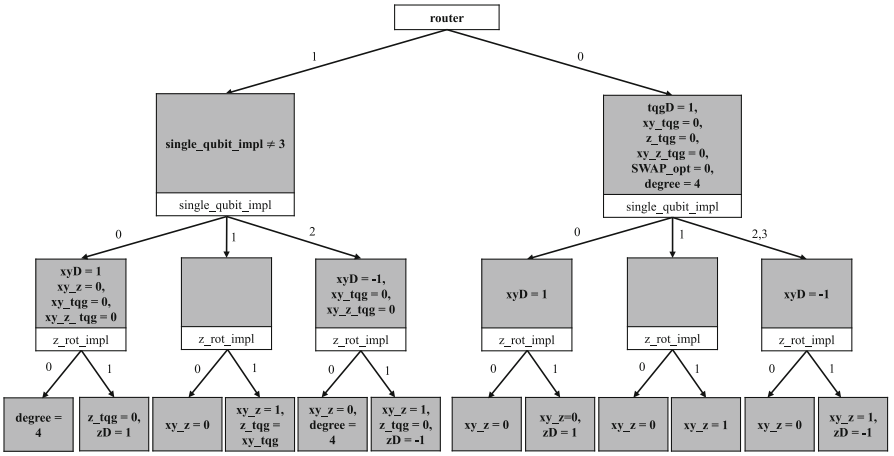

Fig. 2 Tree of valid architectural configurations that *SpinQ* handles

4.4 Routing methods via shuttling

(`router` = *0* → *Shuttle-based SWAP*, *1* → *beSnake*)

There are two available algorithmic options. The first one, introduced in [11, 52], is a *shuttle-based SWAP* routing algorithm. In particular, this algorithm was tailored around the unique constraints of the crossbar architecture [5], which necessitated the maintenance of the checkerboard physical qubit pattern to achieve a time-efficient compilation process. One of the advantages of this algorithm is, in fact, the maintenance of the checkerboard pattern, which in turn keeps the crosstalk interference low and exclusively enables the *Semi-Global* rotations scheme. The second routing option is called *beSnake* [54], and it is capable of freely shuttling qubits around any topology and in any direction, handling complex routing scenarios involving parallelized gates.

4.5 SWAP replacement

(`SWAP_opt` = *0, 1*)

This binary variable concerns an optional functionality of the *beSnake* routing algorithm, which can replace a sequence of shuttles with a SWAP gate under certain conditions [54]. These conditions are satisfied when the accumulated fidelity of a parallelized shuttle sequence exceeds the fidelity of only executing a SWAP on that particular location. This variable is useful for creating insights for architectures that support SWAPs compared to others that do not support them.

4.6 Design interdependencies and verification

All these variable values act as input into the upgraded *SpinQ* housing its new *configurable compiler*, making it the first compilation and DSE framework for spin-qubit architectures. After this upgrade, we are able to provide *SpinQ* with multiple combina-

tions of variable values, compile each input circuit iteratively for all valid architectures, and calculate and store multiple performance metrics, such as the ones used in [11]. These are gate overhead, circuit depth overhead, and ESP of the compiled quantum circuit(s).

As mentioned, not all combinations of variable assignments are valid, as there are interdependencies between them; hence, *SpinQ* automatically filters the design space to allow only valid combinations. If all combinations were allowed, presented in Table 1, there would be a total of 1, 105, 920 architectures, but the valid ones used in this work amount to 29, 312. Figure 2 shows the interdependences tree that *SpinQ* uses to filter valid architectures.

Another significant component of *SpinQ* is the compiler verification tool [11]. This tool is crucial, given the absence of real devices for testing. For this work, the verification functions have been enhanced to handle all valid architectures via our two-step method [11], which thoroughly scrutinizes circuits at each compilation stage.

5 Figure of merit

The selection of an appropriate figure of merit is a critical step to ensure fair and objective evaluation of quantum processor architectures. It must reflect how specific design features influence circuit performance, allowing each architecture to be characterized based solely on its inherent trade-offs and capabilities. To achieve this, each design variable must contribute directly to the final metric, such that the interplay between performance gains and associated costs (e.g., crosstalk or decoherence) is properly captured. This prevents the optimization tool ArtA (introduced in Sect. 6) from disproportionately favoring certain architectures by, for instance, maximizing the degree of gate parallelization without accounting for the negative effects it may introduce, such as increased crosstalk. To highlight an example of a design trade-off, we note that the higher the degree of parallelization, the shorter the circuit depth will be, and therefore, the shorter the algorithm execution time, but it will result in higher crosstalk. Such considerations also express more accurately design decisions in real experimental quantum processors.

When a quantum processor is not physically available, a common method for estimating its performance is to compute the Estimated Success Probability (ESP) [55–57], derived from the compiled quantum circuit. This method is significantly more computationally efficient than alternatives such as Schrödinger–Feynman simulation, tensor network contraction, or noise model simulations, all of which scale poorly with increasing circuit size and qubit connectivity [11, 57]. In this work, we build upon the ESP definition introduced in [11], shown in Eq. 1, and extend it to incorporate all previously discussed architectural considerations. The revised figure of merit, presented in Eq. 2, now consists of three components: a gate operations term, a crosstalk term, and a decoherence term.

$$ESP = \prod_{k} \prod_{i} F_{i,k} \tag{1}$$

where k represents the kth time step and i the ith gate in the kth time step. F is the fidelity of the corresponding gates.

$$ESP = \left(\prod_k \prod_i F_{i,k}\right) \cdot \left(\prod_k \prod_{i,j} C_{i,j,k}\right) \cdot \left(e^{(-t/T_2^*)}\right)^N \tag{2}$$

where C denotes the crosstalk between the ith and jth gates during the kth time step. The last term, adapted from [51], introduces the decoherence-induced errors, which represent the probability of all qubits staying coherent during the execution of the circuit's execution. T_2^* is the decoherence time, t the total duration time of the circuit, and N the total number of qubits. We will discuss each of the terms in detail in the next three subsections.

5.1 Gate operations term

The selection of typical operational fidelities for each gate, denoted by the $F_{i,k}$ term, plays a significant role in determining the success probability of quantum circuits. In our analysis, we consider these fidelities to be constants within the design space, rather than variables. This is because determining the optimal fidelity value for increasing the figure of merit (i.e., ESP) is straightforward; higher operational fidelity invariably leads to improved performance. Since we are conducting an exploration, we take a highly optimistic approach to the state-of-the-art [58–60] values for each gate type[1]:

$$F_{\text{single-qubit}} = 0.9999,\ F_{\text{two-qubit}} = 0.9998 \tag{3}$$

5.2 Crosstalk term

Crosstalk in spin-qubit systems has received limited attention in the literature, primarily due to the complexity of accurately modeling the effects that arise when multiple gates are executed simultaneously in close physical proximity [61–63]. Nonetheless, the presence of crosstalk is expected to have a large influence on quantum computer performance in the future, and current proposals are already suggesting techniques to mitigate it in simple occurrences [61]. For our purposes, crosstalk creates necessary trade-offs between architectural variables, and, as explained before, this is essential for a fair ESP calculation across architectures. For these reasons, we have devised a model to help us discover relative categorical differences between architectures, rather than accurate predictions of crosstalk effects. Our overall aim is not to approximate the actual performance of designs but to pronounce their differences by creating a performance order between them. We further assume that crosstalk should depend on local hardware characteristics, and since this information is already included in the

[1] Another reason we do so is to avoid *numeric underflows* where there is a loss of accuracy in numerical calculations if ESP becomes smaller than the smallest positive representable value in a programming language's floating-point arithmetic.

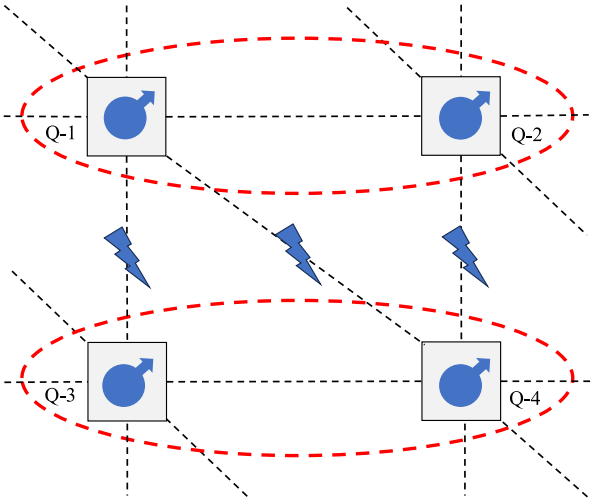

Fig. 3 Schematic overview of crosstalk during the execution of two two-qubit gates. Dashed black lines represent direct qubit connections, red ovals indicate active two-qubit gates, and blue lightning icons denote edges where crosstalk effects occur

operational fidelity of each gate type, our crosstalk definition is a function of those. Below, we summarize our assumptions:

- Nearest neighbor crosstalk only (i.e., a pair of qubits that are directly connected by an edge in the coupling graph)
- Crosstalk occurs for each edge connecting qubits, which are operated in parallel by different gates
- There should be some correlation between the fidelity of gates and crosstalk effects, as they are both influenced by the same quality of fabrication
- The crosstalk occurring between neighboring qubits on which two different quantum gates are applied should not be higher than the combined operational error of these two gates

Based on these assumptions, we calculate the crosstalk effect as follows:

$$C_{i,j,k} = \left(\frac{2}{\frac{1}{F_{i,k}} + \frac{1}{F_{j,k}}} \right)^{n} \tag{4}$$

where n is the number of the total direct links (i.e., nearest neighbors) on the topology between the ith and the jth operation. Similar to before, F corresponds to the operational fidelity.

A schematic overview of crosstalk effects is presented in Fig. 3. The diagram illustrates two two-qubit gates, represented by red ovals, acting between qubits Q-1 and Q-2, and Q-3 and Q-4, respectively. In this hypothetical architecture with `degree` = 4 connectivity, crosstalk arises along the three edges marked with a lightning symbol, as the involved qubits are directly connected to one another.

Table 2 Overview of all gate duration and the decoherence time used

Operation	Shuttle	X/Y/Z (Pulse)	Z (Shuttle)	$\sqrt{SWAP}$	T_2
Duration (ns)	10	100	20	200	1,000,000

5.3 Decoherence term

The last term of Eq. 2 represents the decoherence of quantum information, which creates a strong architectural trade-off between parallelization and crosstalk effects. Since N and T_2^* are given, we need to define the duration time of the circuit t:

$$t = \sum_{k=1}^{M} \max(D_{i,k}) \tag{5}$$

where $D_{i,k}$ is the duration of the ith operation in the kth time step. In Table 2, we present the gate duration and decoherence time used. The motivation behind this selection of durations, once again, is for purposes of architectural comparisons. We took reference for our two-qubit gate (i.e., $\sqrt{SWAP}$) from [59, 60], for the *shuttle* from [51] which is half the shuttle-based Z rotations [11], for the pulse-based single-qubit gates from [58], and for T_2^* from [64].

6 *ArtA*: Artificial Architect

From the aforementioned transformation of the *SpinQ* compilation framework to a fully functional DSE framework, we are able to brute force through all valid combinations of the input variable values. However, as we discussed, this can be a time-consuming process, especially when the defined design space is large. Therefore, a specialized tool is needed to automate the DSE process by traversing the space in a clever and faster way. As a result, fewer design iterations will be tried to achieve the same ESP (range) or other specified performance metrics.

To do that, we introduce **ArtA**, a tool integrated into the *SpinQ* framework that incorporates multiple optimization techniques. Its purpose is twofold: (a) to determine which optimization method is most effective at identifying architectures that achieve the desired ESP for each quantum circuit and (b) to identify the architecture that yields the highest ESP both on a per-circuit basis and across all circuits collectively (i.e., a universal architecture).

The abstracted ArtA process is depicted in Fig. 4. The initialization of ArtA starts with one or more quantum circuits, a design space such as the one described in Sect. 4, the optimization method(s) to traverse the space, and an (optional) termination condition. Inside the optimization stage, ArtA provides an architecture and circuit to *SpinQ* for compilation, gets the ESP value, and, based on a policy of the optimization method, selects the next architecture in the form of input variable values from the available "population" of architectures—a subset of design space selected by the optimization

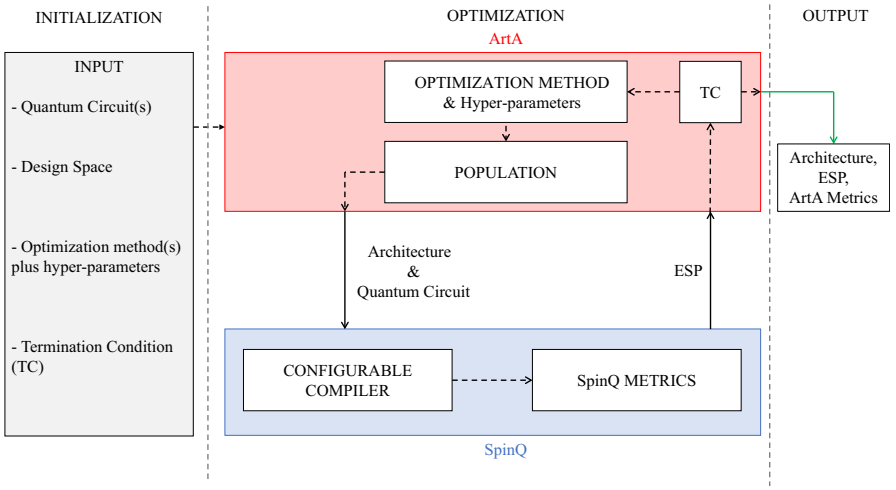

Fig. 4 Schematic overview of the initialization, optimization, and output cycle when using ArtA (red box) together with the upgraded *SpinQ* framework (blue box)

algorithm itself. This process repeats until a specific termination condition (TC) (e.g., elapsed time) is met or a desired range/value of a specific metric is reached (ESP in this work). Upon termination, the resulting architecture, its ESP calculation, and ArtA metrics are stored. In the following sections, we will describe the TCs and ArtA metrics in more detail.

In Appendix B, we analytically discuss the five optimization methods used and their hyper-parameter variations, totaling seventeen different optimization configurations. Also, in Appendix A, we present the seven benchmark quantum circuits and their different sizes in terms of qubit counts.

To answer the last two research questions, we will use ArtA's (a) and (b) capabilities. With the first one, we will answer the third research question, in Sect. 7.1, by testing all the implemented optimization methods to determine which finds the highest ESP the fastest for each quantum circuit. Therefore, first we need to compile each circuit for all 29, 312 architectures and store their ESP for later use. Then, we can run each method and determine the methods' performance based on two new metrics introduced in Sect. 6.2, evaluating ArtA's overall performance. With the second capability of ArtA, we will only use the best-performing optimization method, but expand our circuit selection to concretely answer the last research question, in Sect. 7.2.

6.1 Termination condition

During each optimization cycle, a specific TC is checked to reduce further computational time based on which way, (a) or (b), ArtA is used. Since we first obtain the ESP values for all architectures during the third question, as mentioned before, the highest ESP is known and can work as a TC. Additionally, we observed in practice that most runs reach the highest ESP sooner than 40 min; hence, this can be another TC. More specifically, our analysis revealed that over 95% of processes are completed earlier,

while a minority extends significantly longer. Given that our evaluation focuses on runtime, omitting these prolonged instances won't affect the overall assessment, as their inclusion would lead to unfavorable evaluations regardless.

Therefore, the TC when answering the third research question is:

- If the highest ESP value is found.
- If the method's run time exceeds 40 min.

For the second use of ArtA, which answers the last research question, we investigate the returned architectural designs by using the best optimization method derived from answering the third question. As discussed in Sect. 7.1, we find that the corresponding chosen optimization algorithm did not exceed 23.4% of the total 29, 312 architectures in the worst case before finding the one with the highest ESP.

Therefore, when answering the last research question, the TC triggers:

- When the number of evaluated architectures exceeds 23.4% of the total 29, 312.

6.2 Metrics evaluating ArtA

To determine the performance of ArtA, in the context of replying to the third research question, we need to define some relevant metrics. The optimization methods that will eventually determine ArtA's performance should return a solution faster compared to searching with the native brute force approach of *SpinQ*. Otherwise, ArtA will be of no practical interest. We thus use the relative *time to solution* as a metric in Eq. 6. The numerator includes the total runtime of both the optimization method and compilation until the desired result is reached (or the TC is triggered), and the denominator refers to the total runtime to brute force the entire design space.

$$\text{time-to-solution}_{\text{relative}} = \frac{\text{time-to-solution}}{\text{T}_{\text{Brute force}}} \cdot 100 \tag{6}$$

When the average compilation time increases due to larger quantum circuits, the execution time of the optimization method becomes relatively less significant in the total runtime. This happens because, as circuits grow in size, the compilation time per architecture dominates the total runtime, while the time taken by the optimization method remains approximately constant regardless of circuit size. Therefore, *time to solution* is not sufficient in evaluating ArtA's performance, as it becomes increasingly influenced by compilation time, rather than the efficiency of the optimization method itself. To address this, we also introduce a relative *calls-to-solution* metric in Eq. 7. With *calls to solution* in the numerator, we refer to the total number of times *SpinQ* is called to compile for each new architecture. In case the optimization method attempts the same architecture twice or more, the circuit compilation will be skipped since the results are already stored from the first call. Finally, the denominator represents the brute force approach and is the total number of architectures in the design space, which essentially equals 29, 312.

$$\text{calls-to-solution}_{\text{relative}} = \frac{\text{calls-to-solution}}{N_{\text{architectures}} = 29,312} \cdot 100 \tag{7}$$

7 Results and evaluation

In this section, we will attempt to completely answer the third and fourth research questions. In both questions, we will focus on finding the architecture that obtains the highest ESP among all within the design space. Since all optimization methods are inherently probabilistic, multiple runs are required to create robust conclusions. Due to limited computational resources,[2] we ran each method for each hyper-parameter combination ten times alongside the TCs. Besides the **mean** performance, we will also report the *worst* performing results for reference.

7.1 Comparison of optimization methods

In this first part of our results, we will address the third research question stated in Sect. 3, which is about finding the best performant optimization method according to our ArtA metrics defined in Sect. 6.2. The results of the relative *time-to-solution* metric for finding the architecture with the highest ESP are given in Table 3. This table also provides insights into matching different optimization method configurations (described in Appendix B) with specific quantum circuits. We observe overall that for each quantum circuit, on average, all optimization methods find the solution in less time than brute forcing (values under 100). For a better interpretation of the numbers on the table, a 50 means the particular optimization configuration of ArtA takes half the time of brute forcing all architecture, and 100 means that it takes the same time.

Multiple optimization methods greatly outperform the random sampling (RS column), as expected. The best performance is at **0**.**9**% on average for QV10 with SA [20, 3], meaning that ArtA, with this optimization method configuration and quantum circuit, takes only **0**.**9**% relative *time to solution*, on average, of the total brute force time to find the architecture with the highest ESP.

We calculated based on Table 3 that, on average, for all optimization methods, ArtA was able to decrease the exploration time by **80**.**6**%. Overall, different optimization methods clearly show varied performance across their respective columns. However, the performance differences among various hyper-parameter settings within a single optimization method are less distinct. In a performance order, we can observe GA being first, followed by BO, ACO, and, lastly, SA. More specifically, GA, with a population size of 100 and a mutation probability of 0.2, finds the highest ESP the fastest at **11**.**9**% relative *time-to-solution*, on average, with BO [EI, Ma] coming close at **12**.**1**%. Lastly, in **86**.**7**% of the cases, a top 5 optimization method configuration in the *worst* row is also a top 5 optimization method configuration in the **mean** row, which indicates the consistency of the results across the ten runs. To further support our findings, next, we will assess the relative *calls-to-solution* metric.

The results of the relative *calls-to-solution* metric for reaching the highest ESP are given in Table 4. We observe overall that for each quantum circuit, all optimization methods were able to find the highest ESP architecture in fewer compiler calls than brute forcing through all 29, 312 architectures. Once again, for a better interpretation

[2] All time metrics were retrieved by running ArtA with Python 3.10.8 single threaded on a 2017 MacBook Pro, using a 3.1 GHz Quad Core Intel i7 processor and 16 GB of RAM.

Table 3 Relative *time-to-solution* metric (the lower, the better) for finding the architecture with the highest ESP from the design space

Circuits\Methods			RS	SA				BO				GA				ACO			
				20 2	20 3	50 3	50 2	UCB Ma	EI Ma	EI Ga	UCB Ga	50 0.15	50 0.2	100 0.2	100 0.15	50 0.1	100 0.3	100 0.1	50 0.3
QFT	10	**mean**	**53.9**	**45.4**	**61.8**	**45.2**	**54.9**	**22.5**	**15.0**	**22.6**	**17.0**	**30.3**	**33.8**	**21.4**	**44.5**	**24.4**	**40.5**	**28.1**	**28.6**
		worst	*98.2*	*87.6*	*99.8*	*98.9*	*100.4*	*55.0*	*35.4*	*38.5*	-	-	*88.7*	*46.1*	*132.8*	*58.7*	*96.9*	*58.6*	*60.6*
	20	**mean**	**47.3**	**17.8**	**44.6**	**47.4**	**33.6**	**10.9**	**10.8**	**11.5**	**6.9**	**6.3**	**6.8**	**9.2**	**5.4**	**36.5**	**48.7**	**31.9**	**42.6**
		worst	*96.8*	*50.5*	*84.2*	*93.1*	*57.0*	*25.6*	*25.8*	*28.2*	*21.4*	*15.7*	*12.8*	*15.6*	*16.6*	*59.2*	*87.9*	*60.8*	*80.7*
cA	5b	**mean**	**35.7**	**38.0**	**41.8**	**36.5**	**27.0**	**16.2**	**10.0**	**14.3**	**7.1**	**8.4**	**6.8**	**4.8**	**5.2**	**15.5**	**11.5**	**18.2**	**14.9**
		worst	*58.1*	*67.1*	*93.0*	*71.8*	*64.6*	*61.0*	*27.4*	*33.9*	*22.1*	*16.9*	*14.5*	*9.2*	*12.6*	*33.5*	*26.6*	*73.1*	*43.0*
	10b	**mean**	**37.3**	**8.4**	**22.6**	**25.8**	**28.7**	**4.2**	**11.3**	**8.1**	**12.5**	**4.3**	**3.5**	**5.0**	**3.9**	**24.1**	**11.7**	**24.7**	**18.3**
		worst	*86.0*	*37.5*	*72.2*	*89.8*	*76.8*	*12.4*	*28.3*	*21.7*	*30.0*	*8.0*	*7.6*	*12.8*	*9.5*	*61.4*	*27.8*	*63.5*	*41.4*
vA	5b	**mean**	**52.3**	**10.0**	**24.9**	**18.3**	**11.8**	**10.4**	**9.2**	**9.0**	**7.4**	**3.1**	**5.8**	**5.9**	**4.5**	**15.6**	**12.6**	**18.0**	**22.8**
		worst	*80.2*	*23.6*	*56.5*	*31.2*	*29.9*	*21.7*	*22.3*	*19.9*	*29.1*	*8.5*	*12.2*	*13.8*	*10.6*	*38.5*	*42.8*	*61.9*	*44.7*
	10b	**mean**	**46.7**	**33.4**	**46.9**	**52.1**	**38.0**	**5.4**	**5.0**	**8.0**	**3.7**	**9.2**	**9.4**	**8.9**	**7.2**	**28.1**	**26.2**	**32.4**	**47.4**
		worst	*98.3*	*70.4*	*79.5*	*81.6*	*79.8*	*12.8*	*15.9*	*21.5*	*7.1*	*15.4*	*15.8*	*18.5*	*17.1*	*59.6*	*52.2*	*79.6*	*75.9*
QV	8	**mean**	**40.0**	**7.9**	**2.4**	**5.5**	**4.9**	**16.0**	**17.0**	**56.3**	**74.4**	**29.0**	**40.5**	**29.6**	**37.9**	**1.1**	**1.3**	**1.3**	**1.8**
		worst	*71.4*	*26.0*	*7.9*	*20.9*	*19.6*	-	*38.6*	-	-	-	*74.9*	*72.6*	-	*2.1*	*3.0*	*1.9*	*6.7*
	10	**mean**	**63.7**	**3.4**	**0.9**	**2.8**	**2.6**	**21.6**	**13.3**	**32.2**	**67.4**	**20.4**	**19.4**	**12.5**	**19.4**	**1.0**	**1.3**	**1.1**	**1.3**
		worst	*91.3*	*10.4*	*2.7*	*11.0*	*9.4*	-	*33.9*	*55.4*	-	*42.8*	*37.9*	*39.5*	*39.6*	*1.8*	*2.4*	*2.2*	*3.4*
G	10	**mean**	**43.2**	**56.8**	**46.0**	**51.8**	**46.6**	**42.1**	**30.6**	**30.6**	**45.5**	**27.0**	**14.2**	**26.8**	**32.1**	**23.3**	**18.1**	**34.2**	**27.8**
		worst	*80.4*	*98.3*	*97.4*	*101.1*	*91.6*	*81.7*	*70.4*	*59.4*	*103.6*	*58.3*	*33.0*	*42.6*	*74.2*	*79.2*	*41.8*	*65.1*	*52.5*
	20	**mean**	**62.1**	**46.1**	**59.6**	**34.3**	**46.5**	**11.5**	**12.1**	**11.8**	**7.1**	**7.5**	**8.9**	**11.7**	**13.0**	**30.4**	**24.2**	**26.9**	**32.4**
		worst	*96.1*	*85.6*	*92.2*	*84.6*	*85.7*	*34.4*	*33.4*	*25.3*	*17.4*	*17.4*	*23.2*	*21.9*	*25.9*	*57.6*	*52.6*	*81.5*	*80.6*
R	8	**mean**	**48.5**	**2.4**	**2.9**	**3.2**	**2.3**	**8.7**	**6.0**	**5.9**	**4.7**	**1.1**	**1.4**	**1.8**	**1.6**	**2.4**	**4.5**	**3.0**	**2.8**
		worst	*84.8*	*8.6*	*6.2*	*10.3*	*6.8*	*28.4*	*18.8*	*14.3*	*15.6*	*2.0*	*2.1*	*2.3*	*2.2*	*4.9*	*7.5*	*7.0*	*7.2*
	10	**mean**	**56.4**	**29.2**	**19.2**	**38.9**	**14.7**	**7.4**	**4.6**	**7.1**	**5.2**	**5.8**	**6.8**	**5.5**	**7.8**	**22.6**	**24.0**	**19.9**	**24.2**
		worst	*99.2*	*91.5*	*62.7*	*87.6*	*60.7*	*29.7*	*18.9*	*14.6*	*16.0*	*20.6*	*20.2*	*18.7*	*29.5*	*49.9*	*69.9*	*67.7*	*44.4*
	average mean		**48.9**	**24.9**	**31.1**	**30.2**	**26.0**	**14.7**	**12.1**	**18.1**	**21.6**	**12.7**	**13.1**	**11.9**	**15.2**	**18.8**	**18.7**	**20.0**	**22.1**
	average worst		*86.7*	*54.8*	*62.9*	*65.2*	*56.9*	*36.3*	*30.8*	*30.2*	*29.1*	*20.6*	*28.6*	*26.1*	*33.7*	*42.2*	*42.6*	*51.9*	*45.1*

A minus symbolizes that there was at least one run (out of the ten) in which the top ESP value was not reached in time (TC = 40 min). These results are colored red, as well as runs where the time taken exceeds the brute-force time (values over 100). The green cells indicate the top five configurations of optimization methods for a given quantum circuit. The **mean** and *worst* performing configurations from a sample of ten runs are shown for each quantum circuit (block of rows). **QFT**: Quantum Fourier Transformation, **cA**: Cuccaro Adder, **vA**: vbe Adder, **QV**: Quantum Volume, **G**: Grover's, **RC**: Random Circuit, **RS**: Random Sampling, **SA**: Simulated Annealing (Top: Start Temperature, bottom: Step Size), **BO**: Bayesian Optimization (Top: Acquisition function, bottom: Kernel), **GA**: Genetic Algorithm (Top: Population Size, bottom: Mutation), **ACO**: Ant Colony Optimization (Top: Population Size, bottom: Exploitation Rate)

of the numbers on the table, 50 means the particular optimization configuration of ArtA needs to go through half the architectures in the design space, and 100 means it needs to go through all the architectures.

The best optimization method only takes **1**% of the total compiler calls, on average, for QV8 and QV10 with ACO [50, 0.1]. We calculated based on Table 4 that, on

Table 4 Relative *calls-to-solution* metric (the lower, the better) for finding the architecture with the highest ESP from the design space

Circuits\Methods			RS	SA				BO				GA				ACO			
				20 2	20 3	50 3	50 2	UCB Ma	EI Ma	EI Ga	UCB Ga	50 0.15	50 0.2	100 0.2	100 0.15	50 0.1	100 0.3	100 0.1	50 0.3
QFT	10	**mean**	**53.9**	**46.1**	**61.8**	**45.2**	**55.3**	**9.0**	**6.4**	**9.0**	**7.6**	**23.1**	**28.9**	**23.1**	**22.0**	**24.9**	**38.3**	**29.3**	**26.8**
		worst	*98*	*87.2*	*98.5*	*97.5*	*98.9*	*18.6*	*12.8*	*13.8*	-	-	*63.1*	*43.5*	*51.2*	*56.8*	*79.5*	*58.6*	*54.4*
	20	**mean**	**47.3**	**18.7**	**44.7**	**47.7**	**35.2**	**9.7**	**9.6**	**10.6**	**6.4**	**8.1**	**8.8**	**11.8**	**6.9**	**38.4**	**49.3**	**34.0**	**42.6**
		worst	*96.8*	*51.6*	*84.2*	*92.9*	*59.1*	*21.3*	*22.5*	*24.2*	*18.8*	*19.6*	*16.3*	*19.4*	*20.2*	*61.8*	*84.7*	*63.9*	*77.1*
cA	5b	**mean**	**35.7**	**39.0**	**42.2**	**36.7**	**28.1**	**8.3**	**5.6**	**7.0**	**3.9**	**9.7**	**8.6**	**6.5**	**6.7**	**16.2**	**12.1**	**18.8**	**15.0**
		worst	*58.1*	*67.9*	*92.5*	*71.6*	*65.7*	*33*	*15*	*13.7*	*9.9*	*17.8*	*17.7*	*12.1*	*15.3*	*35.1*	*28*	*72.4*	*41*
	10b	**mean**	**37.4**	**9.0**	**22.8**	**26.0**	**29.6**	**3.8**	**8.6**	**6.5**	**9.7**	**5.8**	**4.7**	**6.7**	**5.2**	**24.5**	**12.0**	**25.5**	**18.7**
		worst	*85.9*	*39.2*	*72.4*	*89.8*	*77.3*	*10.4*	*21.9*	*15.9*	*24.2*	*10.6*	*10.1*	*16.5*	*12.3*	*62.8*	*29.4*	*66.1*	*42.1*
vA	5b	**mean**	**52.3**	**10.7**	**24.9**	**18.2**	**12.6**	**7.1**	**5.8**	**5.9**	**5.0**	**4.1**	**7.6**	**7.8**	**5.9**	**16.7**	**13.6**	**19.2**	**23.4**
		worst	*80.2*	*24.9*	*56.5*	*31.1*	*31.1*	*16.0*	*12.3*	*11.5*	*15.8*	*10.8*	*15.4*	*18.0*	*13.2*	*42.0*	*46.0*	*64.3*	*44.3*
	10b	**mean**	**46.7**	**34.6**	**46.5**	**51.7**	**39.0**	**5.5**	**4.9**	**7.9**	**4.3**	**12.0**	**12.5**	**11.5**	**9.3**	**29.2**	**27.1**	**33.5**	**47.5**
		worst	*98.3*	*71.5*	*79.1*	*81.1*	*80.5*	*12.1*	*14.7*	*19.5*	*7.5*	*18.9*	*20.1*	*23.0*	*20.9*	*61.5*	*53.9*	*80.8*	*73.4*
QV	8	**mean**	**40.0**	**8.1**	**2.4**	**5.6**	**5.0**	**11.0**	**12.0**	**35.2**	**48.3**	**21.1**	**37.3**	**28.6**	**28.1**	**1.0**	**1.3**	**1.3**	**2.0**
		worst	*71.3*	*26.4*	*8.2*	*21.4*	*19.8*	-	*30.0*	-	-	-	*66.1*	*65.9*	-	*2.0*	*3.3*	*2.1*	*7.7*
	10	**mean**	**63.7**	**3.5**	**0.9**	**2.9**	**2.7**	**18.6**	**11.1**	**25.9**	**55.4**	**20.8**	**20.6**	**13.4**	**19.9**	**1.0**	**1.2**	**1.0**	**1.2**
		worst	*91.1*	*10.8*	*2.8*	*11.3*	*9.7*	-	*28.5*	*43.3*	-	*40.5*	*39.5*	*41.3*	*38.9*	*2.0*	*2.3*	*2.1*	*2.9*
G	10	**mean**	**43.1**	**57.3**	**46.1**	**51.7**	**47.6**	**16.3**	**11.6**	**11.6**	**17.1**	**22.4**	**16.5**	**29.5**	**26.1**	**24.0**	**19.2**	**36.3**	**27.0**
		worst	*80.4*	*97.0*	*96.4*	*99.4*	*91.3*	*31.8*	*23.4*	*20.2*	*34.2*	*39.6*	*37.1*	*44.1*	*47.9*	*73.2*	*43.3*	*65.6*	*48.6*
	20	**mean**	**62.0**	**46.8**	**59.6**	**34.5**	**47.4**	**9.2**	**9.0**	**9.3**	**6.2**	**9.7**	**11.3**	**15.1**	**15.5**	**32.9**	**25.6**	**28.6**	**32.5**
		worst	*96.2*	*85.7*	*91.7*	*84.5*	*85.6*	*25.9*	*22.4*	*17.9*	*13.2*	*20.7*	*28.1*	*27.5*	*28.5*	*60.6*	*54.3*	*82.2*	*74.9*
R	8	**mean**	**48.4**	**2.8**	**3.2**	**3.4**	**2.7**	**7.4**	**5.5**	**5.3**	**4.4**	**1.5**	**1.9**	**2.4**	**2.1**	**3.4**	**6.2**	**4.2**	**3.7**
		worst	*84.7*	*9.4*	*6.3*	*11.0*	*7.6*	*21.7*	*17.2*	*11.3*	*13.6*	*2.9*	*2.9*	*3.1*	*3.1*	*6.9*	*10.4*	*10.1*	*9.8*
	10	**mean**	**56.4**	**29.9**	**19.7**	**39.1**	**15.5**	**7.2**	**4.4**	**6.8**	**5.3**	**6.7**	**8.5**	**6.8**	**8.9**	**24.9**	**25.6**	**21.5**	**25.9**
		worst	*99.2*	*91.6*	*63.2*	*87.6*	*61.9*	*26.6*	*17.1*	*12.9*	*14.9*	*23.1*	*23.9*	*22.5*	*31.0*	*53.5*	*70.6*	*71.0*	*46.2*
	average mean		**48.9**	**25.5**	**31.2**	**30.2**	**26.7**	**9.4**	**7.9**	**11.8**	**14.5**	**12.1**	**13.9**	**13.6**	**13.1**	**19.8**	**19.3**	**21.1**	**22.2**
	average worst		*86.7*	*55.3*	*62.7*	*64.9*	*57.4*	*21.7*	*19.8*	*18.6*	*16.9*	*20.5*	*28.4*	*28.1*	*25.7*	*43.2*	*42.1*	*53.3*	*43.5*

A minus with red color cells symbolizes that there was at least one run (out of the ten) in which the top ESP value was not reached in time (TC = 40 min). The green cells indicate the top five configurations of optimization methods for a given quantum circuit. The **mean** and *worst* performing methods from a sample of ten runs are shown for each quantum circuit (block of rows). **QFT**: Quantum Fourier Transformation, **cA**: Cuccaro Adder, **vA**: vbe Adder, **QV**: Quantum Volume, **G**: Grover's, **RC**: Random circuit, **RS**: Random Sampling, **SA**: Simulated Annealing (Top: Start Temperature, bottom: Step Size), **BO**: Bayesian Optimization (Top: Acquisition function, bottom: Kernel), **GA**: Genetic Algorithm (Top: Population Size, bottom: Mutation), **ACO**: Ant Colony Optimization (Top: Population Size, bottom: Exploitation Rate)

average, ArtA's optimization method configurations can decrease the compiler calls by **79.9**%. We can observe, once more, that BO, followed by GA, obtained the best performance compared to other optimization methods. Lastly, in **91.7**% of the cases, a top 5 optimization method configuration in the *worst* row is also a top 5 optimization method configuration in the **mean** row, showing results consistency again.

From different variants of the GA and BO used, we focus on GA [100, 0.2] and BO [EI, Ma], as they have the best performance and did not reach our TC in any of the ten runs. By comparing the two tables, we can observe that BO [EI, Ma] needs marginally more time to evaluate fewer architectures compared to GA [100, 0.2]. Since the compilation time per architecture is the same between methods, this points to a slightly (0.2% on average) longer execution time of BO [EI, Ma] compared to GA [100, 0.2]. This can be explained based on the algorithm's reliance on matrix multiplication and several other linear algebra operations, which are computationally more expensive than the more simple operations of GA (see Appendix B). However, the actual total runtime of ArtA will depend on the ratio between compilation and optimization time, which can change for large circuits. Based on their small relative *time-to-solution* difference, we can predict that BO [EI, Ma] will outperform GA [100, 0.2] for larger circuits as the latter has, on average, 58% higher relative *calls-to-solution*. Therefore, we conclude that BO [EI, Ma] is a better candidate for automating the DSE process for large-scale circuits, as it can traverse the design space faster.

Having said that, making a more detailed analysis for each circuit where internal circuit characteristics are correlated [11, 65] to optimization methods can improve the performance of ArtA by picking methods more accurately. This promising avenue is left for future work when more progress in circuit characterization is made.

7.2 Architecture analysis

We move on to address the last research question after having selected the best-performing optimization method in the previous section. Here, we use ArtA to create architectural insights into spin-qubit devices for each circuit category and suggest a universal architecture for all executed circuits.

In Table 5, we present the values of the architectural variables, introduced in Sect. 4, that were found by ArtA having the highest ESP with the BO [EI, Ma] optimization method alongside their respected ESP values. These circuits are organized by class and number of qubits within the rows of the table. Subsequent columns present the ESP of the best architecture discovered, followed by color-coded architectural parameter categories as delineated in Sect. 4. The use of color accents in the table serves to enhance visual clarity, with each corresponding to the values of each variable.

Before focusing on the architectural insights, we should comment on the low ESP values observed, especially in larger circuit sizes. Although such low values do not have any physical meaning after a certain small number, ESP still remains a reliable way to rank architectures. It is important to understand ESP is not inherently random, regardless of how low it gets. For instance, slight variations between two architectures causing a single gate difference will consistently and reliably be reflected through their ESPs. As long as numeric underflows are not caused, ESP remains a robust figure of merit for our purposes.

Observing the data presented in Table 5, it becomes evident that there is a discernible structure to the values of architectural variables. Notably, many or all circuits exhibit strong preferences for certain variables. Following an implications discussion on the

Table 5 Architectures returned by BO [EI, Ma] optimization method for a range of quantum circuits (refer to Appendix A)

Circuit	Gate count	ESP	xy_z	xy_tqg	z_tqg	xy_z_tqg	xyD	zD	tqgD	sD	sqi	zri	router	swap opt	degree
QFT10	205	58.02	1	1	1	1	100	50	75	100	1	1	1	1	8
QFT20	810	1.12	1	0	0	0	25	100	25	75	1	1	1	0	8
QFT40	4072	3.54E-15	1	0	0	1	100	25	1	25	1	1	1	0	8
QFT50	5025	1.50E-32	1	1	1	0	75	25	100	75	1	1	1	0	8
QFT60	7230	6.37E-59	1	1	1	0	100	50	25	50	1	1	1	0	8
QFT100	20050	6.80E-123	1	0	0	1	100	75	50	100	1	1	1	1	8
G10	278	64.37	1	0	0	1	25	50	1	25	1	1	1	1	8
G20	668	12.31	1	0	0	1	75	25	25	100	1	1	1	1	8
G50	1838	1.11E-05	1	1	1	1	75	50	100	25	1	1	1	1	8
G80	3008	9.55E-18	1	1	1	1	50	50	1	100	1	1	1	0	8
G90	3398	2.28E-23	1	1	1	1	100	100	1	25	1	1	1	0	8
G100	3788	3.88E-30	1	1	1	0	100	75	50	100	1	1	1	0	8
G150	5738	2.71E-75	1	1	1	1	25	25	75	50	1	1	1	0	8
G200	7688	1.56E-144	1	1	1	0	75	50	75	25	1	1	1	0	8
cA12	31	61.51	1	0	0	1	75	75	75	100	1	1	1	0	8
cA22	61	19.29	1	0	0	1	75	25	25	75	1	1	1	1	8
cA42	121	0.13	1	0	0	1	100	50	50	75	1	1	1	0	8
cA62	181	2.24E-05	1	0	0	1	75	100	100	100	1	1	1	0	8
cA82	241	1.54E-10	1	0	0	1	100	50	100	100	1	1	1	0	8
cA100	295	3.82E-16	1	0	0	1	50	100	75	100	1	1	1	0	8
cA130	385	4.67E-30	1	0	0	1	50	75	1	100	1	1	1	0	8
cA258	769	1.56E-137	1	0	0	1	100	75	100	75	1	1	1	0	8
vA16	36	41.06	1	0	0	1	75	75	1	50	1	1	1	1	8
vA31	76	3.03	1	1	1	0	100	100	25	25	1	1	1	1	8
vA61	156	2.90E-05	1	0	0	1	50	100	50	75	1	1	1	1	8
vA91	236	3.97E-13	1	0	0	1	50	100	100	25	1	1	1	0	8
vA121	316	1.46E-25	1	0	0	1	50	100	100	50	1	1	1	0	8
vA148	388	3.57E-40	1	0	0	1	100	100	75	25	1	1	1	0	8
vA193	508	5.42E-71	1	0	0	1	50	100	25	25	1	1	1	0	8
QV8	889	50.50	1	0	0	0	-1	-1	1	25	2	1	1	1	8
QV10	1111	38.55	1	0	0	0	-1	-1	1	25	2	1	1	1	8
QV20	4408	4.35	1	0	0	1	50	100	50	100	1	1	1	1	8
QV30	9939	1.42E-03	1	0	0	1	100	100	50	100	1	1	1	0	8
QV40	18863	3.87E-11	1	0	0	1	100	100	50	50	1	1	1	1	8
bV11	24	97.61	1	0	0	0	-1	-1	100	100	2	1	1	1	8
bV21	44	95.22	1	0	0	0	-1	-1	1	100	2	1	1	1	6
bV30	62	92.73	1	0	0	0	-1	-1	100	100	2	1	1	0	8
bV40	82	89.84	1	0	0	0	-1	-1	100	75	2	1	1	1	8
bV50	102	86.68	1	0	0	0	-1	-1	100	100	2	1	1	1	8
bV65	132	81.86	1	0	0	0	-1	-1	100	50	2	1	1	1	6
bV129	261	46.57	1	0	0	0	-1	-1	25	75	2	1	1	0	8
bV257	519	3.78	1	0	0	0	-1	-1	25	25	2	1	1	1	8
Average (universal architecture)			1.0	0.2	0.2	0.6	74.2	71.1	52.6	66.7	1.2	1.0	1.0	0.5	7.9

Results are color coded, with color schemes following the categories introduced in Sect. 4. Final-row averages omit -1 values

circuit performance of these strong architectural preferences, we will explore areas of weaker preference where there is more variability in value choices.

Strong preference

First, we observe a unanimous preference for the Z rotation implementation (`z_rot_impl`) in the **zri** column, with a pulse-based rotation being preferred. Note that previously, we have assumed in Sect. 5.1 that a shuttle and a pulse-based Z rotation share the same single-qubit gate fidelity, but because the latter requires two shuttles, it can benefit the ESP more than the other type. Moreover, as a shuttle operation involves two quantum dots while a pulse-based operation only involves one, the shuttle-based Z rotation will have a higher chance of incurring crosstalk errors based on our crosstalk

definition. Given these two reasons, BO has favored the pulse-based version. This indicates that shuttle-based single-qubit rotations are not sustainable compared to pulse-based ones for large-scale circuits as they progressively contribute more noise.

Moving on, since the X, Y, and Z gates are implemented the same way hardware-wise, `xy_z` automatically is assumed 1 based on the interdependencies detailed in Sect. 4.6. Consequently, `xy_tqg` and `z_tqg` are equivalent, taking either 0 or 1 from the optimization method of ArtA (i.e., BO [EI, Ma] in this case), as we can observe in the table. Additionally, `xy_z_tqg` can also be conceptually equivalent to `xy_tqg` and `z_tqg` since pulse-based Z rotations are selected. Therefore, whenever `xy_z_tqg` is 1 or `xy_tqg` and `z_tqg` is 1, it means the architecture(s) which allow(s) parallelization of all single- and two-qubit gates obtains the highest ESP. In that sense, one would expect `xy_z_tqg`, `xy_tqg` and `z_tqg` to be synchronized. However, since such exploration freedom was given in the design space, the reasons for the mismatches could be explained by the internal characteristics of the circuits themselves. For example, in QFT100, the parallelization of all gate types occurs more often than in QFT60, as observed from their gate parallelization constraints `xy_z_tqg` = 0, `xy_tqg` = 1, and `z_tqg` = 1.

Conversely, when applying the same reasoning to QV8, QV10, and QFT20, it appears that hardware support for parallel execution of single- and two-qubit gates is not essential. This highlights that the success of real quantum circuits can stem from different architectural "angles," even under reduced hardware capabilities [57]. This exemplifies the strength of our approach: it enables the identification of optimal architecture configurations without requiring the most complex or fabrication-intensive hardware designs.[3]

Another strong preference appears in the *beSnake* router, observed by the dark green 1 values in the **router** column. This result shows that the crosstalk implications of using the *beSnake* router (which gives more shuttling freedom) over the *shuttle-based SWAP* router (which minimizes crosstalk) are not as important as the other two components in Eq. 2. Similarly, the `degree` variable is mostly maximized, indicating again that the additional possibility of crosstalk does not outweigh other benefits, such as having more connections to perform two-qubit gates and shuttling operations.

Lastly, there is a strong preference for a local single-qubit gate implementation (`single_qubit_impl`). A local implementation, on the one hand, provides the highest parallelization possibilities, but on the other, it is the most demanding to achieve experimentally on real hardware.

Based on all the above conclusions, we can summarize that the crosstalk effect is not as strong even though ArtA suggests architectures with high crosstalk factors (the highest connectivity and parallelization and the most dense routing strategy). In fact, the decoherence-induced errors increase more severely (exponentially) with time, based on Eq. 4, than crosstalk, especially in circuits with high qubit and gate counts, which explains the prevailing preference for maximizing parallelization.

The same can be observed in our universal architecture recommendation in the last row when rounding the numbers, which performs best on average for all the

[3] Ultimately, each circuit profile has specific architectural traits that can be leveraged to achieve optimal performance without the need for excessively advanced hardware features (achieving a balance in architectural characteristics).

circuits examined. Additionally, the operational fidelity, which is directly linked to the gate overhead of compilation, can also severely impact the final ESP for circuits with high gate counts, which in turn explains the need to add the least additional gates possible during compilation through the use of *beSnake* [54] and local single-qubit implementation. Having said that, Bernstein Vazirani, QV8, and QV10 circuits preferred the global implementation, as these circuits consist of a significant number of single-qubit gates (with the same angles) being executed at the same time step.

Weak preference

Overall, we can observe that the single-qubit operations and shuttling prefer a higher degree of parallelization than the two-qubit gates. Considering that `xyD` and `zD` have the same meaning due to **zri** = 1, their combined averages from the last row of Table 5 are **72**.**7**%. This compares to an average of **52**.**6**% for the two-qubit gates and **66**.**7**% for `sD`. Therefore, prioritizing the simultaneous operation of single-qubit gates (and shuttling) over two-qubit gates is more important.

The `swap_opt` variable seems to show a preference for supporting SWAP gates for lower qubit counts. A possible explanation for this is that SWAP operations take a longer time than a parallelized sequence of shuttles [54]. As we explained before, the effects of having longer circuit execution times become more severe with larger circuits; hence, the extra SWAPs can outweigh the gained fidelity they initially offer. Combining this conclusion for large-scale circuits with the observed strong preference for shuttling flexibility, enabled by the beSnake router, suggests that shuttle operations constitute a key communication mechanism in spin-qubit architectures for the future.

8 Conclusion

In this paper, we propose a comprehensive exploration of the current and future architectural design space for quantum-dot spin-qubit quantum processors. Our research focuses on assessing the critical architectural characteristics that could be key in ensuring high performance on such devices. To this end, we present an upgraded version of the *SpinQ* compilation framework [11] for spin-qubit architectures in which architectural variables can configure internal compilation passes, resulting in the first Design Space Exploration (DSE) framework. After defining our design space, consisting of 29, 312 architectures, we propose a multi-optimization-based tool, ArtA (**Art**ificial **A**rchitect), to automate the DSE process. The goal of **ArtA** is twofold: (a) to identify which of the seventeen optimization configurations can most efficiently discover the architecture that achieves the highest ESP, compared to brute force exploration; and (b) to determine the architectural design characteristics that are critical for constructing high-performance spin-qubit devices across different circuit categories. We have shown up to **99.1%** improvement in computation times compared to brute forcing, showing that it is possible to easily explore a vast number of designs. We have also provided insights into matching different optimization methods with specific quantum circuit categories.

It should be noted that the underlying goal of our study was not to exhaustively benchmark all possible optimization methods, but rather to introduce the concept of applying Design Space Exploration to quantum computing systems—with a particular

focus on spin-qubit architectures—by employing a diverse and representative selection of methods from different optimization families. Bayesian optimization with an Expected Improvement acquisition function and a Matérn kernel was chosen as the best method due to its effectiveness in managing higher qubit counts. After that, we proceed to automate the DSE process for 44 quantum circuits of up to 258 qubits within the 29,312-architecture design space. Our findings reveal the importance of minimizing the circuit duration by maximizing parallelization whenever possible instead of selecting architectural characteristics that minimize crosstalk errors. Crosstalk is currently a significant concern in experiments, but our long-term considerations deem decoherence more crucial. Lastly, the shuttle operation is preferred over costly SWAP operations for communication purposes, especially for large-scale circuits. Contrary to that, shuttle-based single-qubit gates such as Z rotations are not preferred over pulse based. Furthermore, our universal architecture suggestion places higher priority on parallelizing single-qubit gates rather than two-qubit gates. Overall, our work demonstrates how the integration of DSE methodologies with optimization algorithms can uncover valuable architectural insights that would be difficult, if not impossible, to foresee while suggesting points of optimality with reduced hardware complexity.

However, it is important to acknowledge that while these findings highlight promising architectural directions, several of these design choices —such as high degrees of parallelization, connectivity, and flexible shuttling strategies—currently pose significant engineering challenges. Implementing fully local control, for instance, demands high fabrication uniformity and precise calibration, which is non-trivial at larger scales. Likewise, extensive shuttling flexibility requires precise individual gate addressing, which is experimentally demanding. Nonetheless, the purpose of this work is to chart a clear path forward, identifying architectural features that, while ambitious, can guide future efforts toward scalable and performant spin-qubit processors.

As for future improvements, a broader range of hyper-parameter combinations and circuit variations could enhance ArtA's effectiveness. Then, with advancements in spin-qubit device fabrication, new architectural possibilities could potentially expand the design space, and such integration will be relatively straightforward owing to the modularity of the internal functions of *SpinQ* and ArtA. Moreover, introducing non-architectural constraints such as development costs and time into ArtA's design space could help make real-world predictions regarding developing spin qubits. For instance, this would allow for operational fidelities to become variables.

Also, some of the assumptions made in this work could be fine-tuned as the field progresses. It is crucial to acknowledge the impact of these assumptions on the observations presented. For instance, we assumed the fidelity of pulse-based Z rotations to be equivalent to that of a single shuttle operation. This assumption should not be construed as either favorable or unfavorable; rather, it is a foundational premise upon which ArtA bases its architectural "recommendations." Should empirical evidence later demonstrate that shuttle operations exhibit significantly higher fidelity compared to pulse-based Z rotations or the crosstalk model depends on more factors, it may necessitate a reevaluation of preferences.

Further analysis of architectural patterns associated with specific quantum circuit characteristics may reveal additional trends and enable systematic categorization. For example, circuits with a high proportion of two-qubit gates (e.g., 90% or more) may

exhibit optimal performance only when the qubit connectivity exceeds a certain threshold. Insights like these, based on various circuit properties [11, 65], can inform the construction of a generalized lookup table that maps algorithmic profiles to architectural configurations. Such a resource would enable rapid prediction of high-performing architectures for new circuits, eliminating the need to re-run the full DSE process.

Appendix A: Quantum circuits

Quantum circuits are defined as a sequence of gates to be executed on qubits (single-qubit gates) or between qubits (two- or multiple-qubit gates). Such quantum circuits are often given in a hardware-agnostic form. This means there are no considerations for device-specific restrictions, such as connectivity of qubits or the set of executable gates. Therefore, a compiler [11, 66–74] is necessary to transform a hardware-agnostic circuit into a circuit that is executable on a given device architecture.

We have selected a wide range of representative quantum circuits of different qubit counts from Qlib [75] and qbench [65] libraries to thoroughly test *SpinQ* and ArtA. Table 6 summarizes all the quantum circuits used in this work.

Appendix B: Optimization methods

We introduce the five optimization methods, four of which have four different hyper-parameter configurations, totaling seventeen configurations. Our goal was not to exhaustively benchmark all possible optimization methods, but rather to introduce the overall concept of applying optimization to the DSE process for quantum computing systems, with a focus on spin-qubit architectures, using a diverse and representative selection of methods. More specifically, from the family of nature-inspired evolutionary algorithms, we selected the well-established Genetic Algorithms (GA); from the class of physics-inspired metaheuristics, we chose the widely used Simulated Annealing (SA); from surrogate-based optimization methods, we included Bayesian Optimization (BO); and from the Swarm Intelligence class, we incorporated Ant Colony Optimization (ACO). In the following sections, we will introduce each method, describe the policy it uses to select a new architecture in each iteration loop of ArtA, give additional details, and finally describe the hyper-parameters used. Several of the optimization methods rely on a notion of distance between architectures. Since our design parameters include categorical variables, we begin by outlining how a meaningful distance metric can be defined in this context

B.1 Distance measure

The notion of distance is hard to apply to values of categorical variables because a categorical variable is defined by having no order between the possible values. For example, the set $\phi = \{\text{apple, pear, banana}\}$ is categorical, as the "<" or ">" signs

Table 6 Summary of all quantum circuits analyzed to answer the third and fourth research questions

Circuit name	Description	Qubit count
QFT	Produces a highly entangled state including a phase factor [76]	10*, 20*, 40, 50, 60, 100
Grover's search	Used to perform database search [77]	10*, 20*, 50, 80, 90, 100, 150, 200
Quantum Volume	Created to study the Quantum Volume metric [78]	8*, 10*, 20, 30, 40
Random circuits	Circuit used for benchmarking quantum computers	8*!, 10*!
Cuccaro Adder	Used for the addition of two numbers [79]	12*, 22*, 42, 62, 82, 100, 130, 258
vbe Adder	Also used for addition, uses more ancilla qubits [80]	16*, 31*, 61, 91, 121, 148, 193
Bernstein Vazirani	Finds an unknown bitstring [81]	11, 21, 30, 40, 50, 65, 129, 257

All qubit counts, followed by a *, are used in answering both research questions, while an absence of a symbol indicates algorithms used in the fourth research question only. A ! indicates circuits *only* used in the third research question

hold no value when applied to this set. Consequently, no order means no distance can be introduced.

There is only one variable that is categorical: `single_qubit_ impl`. To solve this, we postulate an ordering of values by the maximum amount of gates that would be theoretically executed during the same time step. The ordering we used in this work is then *sequential* < *semi-global* < *local* < global. This means, for instance, the distance between *sequential* and *semi-global* is 1, but the distance between *sequential* and *local* is 2.

The second challenge is how to combine the distance measured between values of one variable with the distance measured between values of a different variable. To describe this, we define d as the distance between two values of the same variable and D as the distance between architectures such that the relation between D and d is given in Eq. B1. This implies that the distance between two architectures is the sum of the distances between the corresponding variable values. Finally, we decided to count all distances in units of 1 and have the distance equal to the difference in indices if we were to order the set of allowed values of one variable. This holds for all values, such that $d(\texttt{degree} = 4, \texttt{degree} = 6) = 1$, $d(\texttt{degree} = 4, \texttt{degree} = 8) = 2$, etc.

$$D(\text{arch}_1, \text{arch}_2) = \sum_{\text{arch\_variable}} d_{\text{arch\_variable}}(\text{arch}_1, \text{arch}_2) \tag{B1}$$

B.2 Random sampling

Random sampling is used as our baseline to compare other optimization methods. It traverses the design space by randomly selecting architectures to investigate. The policy comprises assigning one randomly sampled architecture as the population.

B.3 Simulated annealing

SA [82] is inspired by the metallurgical process of heating and then slowly cooling a material, which minimizes defects and thus approximates its lowest energy state. SA is a mix of a greedy strategy for selecting the best-performing architecture while trying to escape local optima to find the global optimum. A more detailed overview of simulated annealing can be found in [83]. The policy is as follows:

1. Start with a random architecture, called architecture$_i$, which has ESP$_i$, where i is the iteration number. It should be noted SA only works with a population size of one, which is the architecture$_i$.
2. Set a starting temperature $T_i = T_{\text{start}}$ (hyper-parameter set to either 50 or 20).
3. Find a random candidate architecture$_{c,i}$ which has distance D_i to architecture$_i$. We assume that D_i is randomized but remains greater than 0 to avoid picking the same architecture. Then, $D_i = \max(\text{Normal}(1, 1) \cdot step\_size, 1)$, with $step\_size$ used as a hyper-parameter equal to either 2 or 3, and Normal(1,1) indicates a normal distribution with mean and standards deviation equal to 1. In case no architecture$_{c,i}$ with distance D_i to architecture$_i$ is found, a new D_i is generated, and this step repeats until an architecture$_{c,i}$ is picked.

4. Calculate the selection probability: $C_i = \exp((\text{ESP}_{c,i} - \text{ESP}_i)/T_i)$.
5. Accept the candidate architecture as current architecture if $C_i > u$, where u is a random number between 0 and 1.
6. Update temperature T_i (linear cooling process) as $T_i = (T_{\text{start}}/i)$
7. Repeat steps 3–6 until the TC is met.

The temperature regulates the probability of accepting a candidate architecture with a lower ESP. Specifically, with higher temperatures, there are higher chances of accepting candidate architectures even if they have worse ESP values. However, as temperature decreases, the algorithm only accepts architectures with higher ESP. This mechanism prevents getting stuck in local maxima at the start of the optimization method while gradually becoming greedy at the end by selecting the top-performing architectures only.

B.4 Bayesian optimization

BO [84, 85] uses conditional probabilities based on previously sampled data points to predict the function value and uncertainty of subsequently sampled data points. It typically assumes a Gaussian Process (GP) as the statistical model that governs the relation between the function values of points. Our policy for BO is as follows:

- We assume the ESP function has been observed at n_o architectures at iteration i.
- The last $n_{\text{sample history}}$ (set to 300 in this work) of the sampled architectures is selected. We call them a_s.
- From the unsampled architectures, $n_{\text{candidate amount}}$ (set to 1000) architectures are selected. We call them a_c.
- a_s is used to build a surrogate function for a_c. The surrogate function predicts the mean and standard deviation for each architecture in a_c. This surrogate function is constructed with a *mean function* μ_0 and a covariance matrix by evaluating a *kernel* Σ_o.
- Given the surrogate function, we compute the acquisition function. This acquisition function uses each architecture's mean and standard deviation in a_c to determine whether an architecture should be evaluated. It balances a high expected value (exploitation) with a high uncertainty (exploration). As explained later, there are many acquisition functions, each with its own approach to making this exploitation/exploration trade-off.
- From a_c we choose $n_{\text{size pop}}$ (set to 50) architectures, namely those with the highest value of the acquisition function.

For discrete functions in the BO algorithm, all non-integer valued points are first rounded to integers before calculating their expected value [86]. Additionally, since not all combinations of architecture variables constitute valid architectures, we use a BO formulation for constrained problems [87]. Therefore, we only sample from valid architectures, which automatically satisfy the constraints and only contain the integer values for variables that are of interest to our problem.

For each architecture $a \in \mathrm{a}_c$, our GP assumption lets us compute the conditional distribution of ESP using Bayes' rule [88].

$$\mathrm{ESP(a)}|\mathrm{ESP}(\mathrm{a}_s) \sim \mathrm{Normal}(\mu_s(\mathrm{a}), \sigma_s^2(\mathrm{a})) \tag{B2}$$

where

$$\mu_s(\mathrm{a}) = \Sigma_0(\mathrm{a}, \mathrm{a}_s)\Sigma_0(\mathrm{a}_s, \mathrm{a}_s)^{-1}(\mathrm{ESP}(\mathrm{a}_s) - \mu_0(\mathrm{a}_s)) + \mu_0(\mathrm{a}) \tag{B3}$$

and

$$\sigma_s^2(\mathrm{a}) = \Sigma_0(\mathrm{a}, \mathrm{a}) - \Sigma_0(\mathrm{a}, \mathrm{a}_s)\Sigma_0(\mathrm{a}_s, \mathrm{a}_s)^{-1}\Sigma_0(\mathrm{a}_s, \mathrm{a}) \tag{B4}$$

For the mean function, we use the arithmetic mean as:

$$\mu_0(x)|\mathrm{a}_s = \frac{1}{s}\sum_{i=1}^{s}(\mathrm{ESP}_i) \tag{B5}$$

Two different Kernels are used, namely the Gaussian Kernel and the Matérn Kernel [88] with $\nu = 1.5$, both of which are commonly used for BO problems. They are defined as follows:

$$\Sigma_0(\mathrm{a}_a, \mathrm{a}_b) = \begin{cases} \text{Gaussian:} \\ \quad \exp(-||\mathrm{a}_a - \mathrm{a}_b||^2) \\ \text{Matérn:} \\ \quad (1 + \sqrt{3}||\mathrm{a}_a - \mathrm{a}_b||) \cdot \exp(-\sqrt{3}||\mathrm{a}_a - \mathrm{a}_b||) \end{cases} \tag{B6}$$

Given this surrogate function, an acquisition function (AF) is calculated to determine which points will be sampled next. To this end, we use Expected Improvement (EI) and Upper Confidence Bound (UCB) as AF, defined as:

$$\mathrm{AF(a)} = \begin{cases} \text{EI:} \\ \quad (\mu_s(\mathrm{a}) - \max(\mathrm{ESP}_s)) \cdot \mathrm{CDF}(Z) + \sigma_s(\mathrm{a}) \cdot \mathrm{PDF}(Z) \\ \text{UCB:} \\ \quad \mu_s(\mathrm{a}) + \sigma_s(\mathrm{a}) \end{cases} \tag{B7}$$

with $Z = (\mu_s(\mathrm{a}) - \max(\mathrm{ESP}_s))/(\sigma_s(\mathrm{a}))$. CDF and PDF are the Cumulative/Probability Density Functions, respectively. Variable $\max(\mathrm{ESP}_s)$ equals the highest ESP value found for all architectures present in a_s. ArtA then picks the new architectures with the highest acquisition function values to sample in the new iteration until the TC is met.

B.5 Genetic algorithm

GA [89] is modeled after the biological evolution of genes to arrive at a near-optimal solution for an optimization problem. The GA changes the architectural variables of

high-performing architectures to explore and discover even better-performing architectures with the use of three operations: selection, mutation, and cross-over. The GA policy works as follows:

- Every generation's iteration, i, considers a population of size n (hyper-parameter set to either 50 or 100). The population at generation $i+1$ consists of elite, cross-over, and mutation children. These are explained next.
- From these n architectures, n_{elite} individuals (set to 15% of n) with the highest ESP value are selected and automatically included in our population of the next generation.
- Additionally, n_{parent} (set to 50% of n) out of n architectures are selected iteratively by randomly considering $n_{\text{tournament}}$ architectures (set to 5) from the population and selecting the one with the highest ESP value. There are no duplicate parents.
- Two parents of this group are selected randomly. Cross-over is performed by picking a random cross-over point c with $1 \leq c \leq k$, where k is the total number of architectural variables of the design space. A new architecture is formed by combining the architectural variables of the two parents, with c indicating the index separating the two halves of the variable list. The process is repeated if this is not a valid architecture. $n_{\text{cross-over}}$ (set to 30% of n) architectures are created in this way.
- The remaining new generation is populated with n_{mut} architectures (set to 55% of n). For this operation, some children (from before) alter each of their architectural variable values with probability p_{mut} (hyper-parameter set to either 0.15 or 0.2). The process is repeated if this is not a valid architecture until one is found.
- The algorithm repeats until the TC is met.

Note that in GA (as in SA), it is possible to have the same architecture in multiple populations.

B.6 Ant colony optimization

The fundamental concept behind the ACO [90] is based on how ants find the shortest route from their colony to a food source. Ants lay down pheromones along their path, and the intensity of the pheromone trail guides other ants to the food source. Over time, the pheromone trail evaporates, reducing its strength. However, the shorter the path, the more frequently traveled by ants, and thus, more pheromone is deposited. This positive feedback eventually leads the colony to converge on the shortest path.

To implement ACO in the context of finding the highest ESP architecture, we need to explicitly pre-create the graph (in this case, a tree) on which ants are walking during the run. The four steps involved are as follows. (i) A "start" node is created. (ii) After a random valid architecture is selected, the next node after "start" is created with the first architectural variable value. The subsequent node includes the first and second variable values of this valid architecture. The tree continues to be constructed all the way until the last node contains the variable values. (iii) Sample the rest of the valid architectures of the design space and repeat the same process. Use the same node if the generated node with the same variable values already exists; otherwise, create a bifurcation to a new node. (iv) The ants traverse this tree, and the node at which they

end will always be a valid architecture. Having constructed this tree, the policy is as follows:

- Pheromone deposition is initialized on all edges of the tree.
- A population of n architectures (hyper-parameter set to either 50 or 100) is considered at every time step i.
- The tree is traversed n times (each time by an ant) from the top until the end for all n architectures. A selection rule is applied to a node with multiple outgoing nodes based on its pheromone concentration. With probability p_{exploit} (hyper-parameters set to either 0.1 or 0.3), the outgoing edge with the highest pheromone concentration is taken. Otherwise, exploration is performed with selection probability for each edge, which is the ratio of the pheromone concentration for the edge to the total pheromone amount on the entire tree.
- Pheromone concentration on each edge of an ant path is updated with Eq. B8 [91].
- Pheromones decay (set to 10% rate) for all edges.
- The new population at $i + 1$ consists of the n architectures these ants arrive at.
- The algorithm repeats until the TC is met.

$$ph_{ij} = \begin{cases} \text{if } \Delta ESP_i < 0: \ min\_ph \\ \text{else if } \Delta\text{ESP}\_avg < 0: \ max\_ph \\ \text{else}: \min(min\_ph + (max\_ph - min\_ph) \cdot \frac{\Delta\text{ESP}_i}{\Delta\text{ESP}\_avg}, max\_ph) \end{cases} \tag{B8}$$

Here, ph_{ij} is the pheromone deposited on the edge between node i and node j. max_ph is the maximum pheromone deposition (set to 6) and min_ph the minimum (set to 1). $\Delta\text{ESP}_i = \text{ESP}_{\text{ant}} - \max(\text{ESP})$, with max(ESP) being the maximum ESP observe so far. $\Delta\text{ESP\_avg} = \frac{1}{3}(\Delta\text{ESP}_{i-1} + \Delta\text{ESP}_{i-2} + \Delta\text{ESP}_{i-3})$. This was included to make large gains in the discovery of high ESP architectures relatively easy at an earlier stage, while at a later stage, increases in ESP will be smaller. Weighing the increase in ESP with this rolling average reflects this.

Acknowledgements This work is part of the research program OTP with project number 16278, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). CGA acknowledges funding from the Spanish Ministry of Science, Innovation and Universities through the Beatriz Galindo program 2020 (BG20-00023) and from the European ERDF under grant PID2021-123627OB-C51.

Author Contributions NP wrote the manuscript, conceptualized ArtA, the new ESP, the main ideas presented in this work, coded the upgraded SpinQ framework, and partially coded ArtA. DH coded most of ArtA, created drafts of the figures, and steered the research forward with NP's supervision. AS and SB provided ideas for the optimization methods and insights on best practices. SB and CGC reviewed the drafts of the manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest There is no conflict of interest between the authors.

References

1. Resch, S., Karpuzcu, U.R.: Quantum computing: an overview across the system stack. arXiv preprint arXiv:1905.07240 (2019)
2. Chatterjee, A., Stevenson, P., De Franceschi, S., Morello, A., Leon, N.P., Kuemmeth, F.: Semiconductor qubits in practice. Nat. Rev. Phys. **3**(3), 157–177 (2021)
3. Intel Corporation: Intel's New Chip to Advance Silicon Spin Qubit Research for Quantum Computing. https://www.intel.com/content/www/us/en/newsroom/news/quantum-computing-chip-to-advance-research.html (2023)
4. Vandersypen, L., Bluhm, H., Clarke, J., Dzurak, A., Ishihara, R., Morello, A., Reilly, D., Schreiber, L., Veldhorst, M.: Interfacing spin qubits in quantum dots and donors-hot, dense, and coherent. npj Quantum Inf. **3**(1), 1–10 (2017)
5. Li, R., Petit, L., Franke, D.P., Dehollain, J.P., Helsen, J., Steudtner, M., Thomas, N.K., Yoscovits, Z.R., Singh, K.J., Wehner, S., et al.: A crossbar network for silicon quantum dot qubits. Sci. Adv. **4**(7), 3960 (2018)
6. Hendrickx, N.W., Lawrie, W.I., Russ, M., Riggelen, F., Snoo, S.L., Schouten, R.N., Sammak, A., Scappucci, G., Veldhorst, M.: A four-qubit germanium quantum processor. Nature **591**(7851), 580–585 (2021)
7. Borsoi, F., Hendrickx, N.W., John, V., Meyer, M., Motz, S., Van Riggelen, F., Sammak, A., De Snoo, S.L., Scappucci, G., Veldhorst, M.: Shared control of a 16 semiconductor quantum dot crossbar array. Nat. Nanotechnol. **19**(1), 21–27 (2024)
8. Zhang, X., Morozova, E., Rimbach-Russ, M., Jirovec, D., Hsiao, T.-K., Fariña, P.C., Wang, C.-A., Oosterhout, S.D., Sammak, A., Scappucci, G., et al.: Universal control of four singlet-triplet qubits. Nat. Nanotechnol. **20**(2), 209–215 (2025)
9. John, V., Yu, C.X., Straaten, B., Rodríguez-Mena, E.A., Rodríguez, M., Oosterhout, S., Stehouwer, L.E., Scappucci, G., Bosco, S., Rimbach-Russ, M., et al.: A two-dimensional 10-qubit array in germanium with robust and localised qubit control. arXiv preprint arXiv:2412.16044 (2024)
10. Unseld, F.K., Undseth, B., Raymenants, E., Matsumoto, Y., Karwal, S., Pietx-Casas, O., Ivlev, A.S., Meyer, M., Sammak, A., Veldhorst, M., et al.: Baseband control of single-electron silicon spin qubits in two dimensions. arXiv preprint arXiv:2412.05171 (2024)
11. Paraskevopoulos, N., Sebastiano, F., Almudever, C.G., Feld, S.: Spinq: Compilation strategies for scalable spin-qubit architectures. ACM Trans. Quantum Comput. **5**(1) (2023). https://doi.org/10.1145/3624484
12. Pimentel, A.D.: Methodologies for design space exploration. In: Handbook of Computer Architecture, pp. 1–31. Springer (2022)
13. Mathew, D.M., Chinazzo, A.L., Weis, C., Jung, M., Giraud, B., Vivet, P., Levisse, A., Wehn, N.: Rramspec: A design space exploration framework for high density resistive ram. In: Embedded Computer Systems: Architectures, Modeling, and Simulation: 19th International Conference, SAMOS 2019, Samos, Greece, July 7–11, 2019, Proceedings 19, pp. 34–47 (2019). Springer
14. Yoneda, J., Takeda, K., Otsuka, T., Nakajima, T., Delbecq, M.R., Allison, G., Honda, T., Kodera, T., Oda, S., Hoshi, Y., et al.: A quantum-dot spin qubit with coherence limited by charge noise and fidelity higher than 99.9%. Nat. Nanotechnol. **13**(2), 102–106 (2018)
15. Camenzind, L.C., Geyer, S., Fuhrer, A., Warburton, R.J., Zumbühl, D.M., Kuhlmann, A.V.: A hole spin qubit in a fin field-effect transistor above 4 kelvin. Nat. Electron. **5**(3), 178–183 (2022)

16. Zwanenburg, F.A., Dzurak, A.S., Morello, A., Simmons, M.Y., Hollenberg, L.C.L., Klimeck, G., Rogge, S., Coppersmith, S.N., Eriksson, M.A.: Silicon quantum electronics. Rev. Mod. Phys. **85**, 961–1019 (2013). https://doi.org/10.1103/RevModPhys.85.961
17. Loss, D., DiVincenzo, D.P.: Quantum computation with quantum dots. Phys. Rev. A **57**, 120–126 (1998). https://doi.org/10.1103/PhysRevA.57.120
18. Veldhorst, M., Yang, C., Hwang, J., Huang, W., Dehollain, J., Muhonen, J., Simmons, S., Laucht, A., Hudson, F., Itoh, K.M., et al.: A two-qubit logic gate in silicon. Nature **526**(7573), 410–414 (2015)
19. Zajac, D., Hazard, T., Mi, X., Wang, K., Petta, J.R.: A reconfigurable gate architecture for si/sige quantum dots. Appl. Phys. Lett. **106**(22), 223507 (2015)
20. Watson, T., Philips, S., Kawakami, E., Ward, D., Scarlino, P., Veldhorst, M., Savage, D., Lagally, M., Friesen, M., Coppersmith, S., et al.: A programmable two-qubit quantum processor in silicon. Nature **555**(7698), 633–637 (2018)
21. Hanson, R., Kouwenhoven, L.P., Petta, J.R., Tarucha, S., Vandersypen, L.M.: Spins in few-electron quantum dots. Rev. Mod. Phys. **79**(4), 1217 (2007)
22. Burkard, G., Ladd, T.D., Pan, A., Nichol, J.M., Petta, J.R.: Semiconductor spin qubits. Rev. Mod. Phys. **95**(2), 025003 (2023)
23. Bluhm, H., Schreiber, L.R.: Semiconductor spin qubits-a scalable platform for quantum computing? In: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2019). IEEE
24. De Michielis, M., Ferraro, E., Prati, E., Hutin, L., Bertrand, B., Charbon, E., Ibberson, D.J., Gonzalez-Zalba, M.F.: Silicon spin qubits from laboratory to industry. J. Phys. D Appl. Phys. **56**(36), 363001 (2023)
25. De Leon, N.P., Itoh, K.M., Kim, D., Mehta, K.K., Northup, T.E., Paik, H., Palmer, B., Samarth, N., Sangtawesin, S., Steuerman, D.W.: Materials challenges and opportunities for quantum computing hardware. Science **372**(6539), 2823 (2021)
26. Boter, J.M., Dehollain, J.P., Van Dijk, J.P., Xu, Y., Hensgens, T., Versluis, R., Naus, H.W., Clarke, J.S., Veldhorst, M., Sebastiano, F., et al.: Phys. Rev. Appl. **18**(2), 024053 (2022)
27. Hill, C.D., Peretz, E., Hile, S.J., House, M.G., Fuechsle, M., Rogge, S., Simmons, M.Y., Hollenberg, L.C.: A surface code quantum computer in silicon. Sci. Adv. **1**(9), 1500707 (2015)
28. Franke, D.P., Clarke, J.S., Vandersypen, L.M., Veldhorst, M.: Rent's rule and extensibility in quantum computing. Microprocess. Microsyst. **67**, 1–7 (2019)
29. Paquelet Wuetz, B., Bavdaz, P., Yeoh, L., Schouten, R., Van Der Does, H., Tiggelman, M., Sabbagh, D., Sammak, A., Almudever, C.G., Sebastiano, F., et al.: Multiplexed quantum transport using commercial off-the-shelf cmos at sub-kelvin temperatures. npj Quantum Inf. **6**(1), 1–8 (2020)
30. Pauka, S., Das, K., Kalra, R., Moini, A., Yang, Y., Trainer, M., Bousquet, A., Cantaloube, C., Dick, N., Gardner, G., et al.: A cryogenic interface for controlling many qubits. arXiv preprint arXiv:1912.01299 (2019)
31. Veldhorst, M., Eenink, H., Yang, C.-H., Dzurak, A.S.: Silicon cmos architecture for a spin-based quantum computer. Nat. Commun. **8**(1), 1–8 (2017)
32. Ivlev, A.S., Crielaard, D.R., Meyer, M., Lawrie, W.I., Hendrickx, N.W., Sammak, A., Scappucci, G., Déprez, C., Veldhorst, M.: Operating semiconductor qubits without individual barrier gates. arXiv preprint arXiv:2501.03033 (2025)
33. Preskill, J.: Quantum computing in the nisq era and beyond. Quantum **2**, 79 (2018)
34. Almudever, C.G., Alarcon, E.: Structured optimized architecting of full-stack quantum systems in the nisq era. In: 2021 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 762–767 (2021). IEEE
35. Tomesh, T., Martonosi, M.: Quantum codesign. IEEE Micro **41**(5), 33–40 (2021)
36. Beverland, M.E., Murali, P., Troyer, M., Svore, K.M., Hoefler, T., Kliuchnikov, V., Low, G.H., Soeken, M., Sundaram, A., Vaschillo, A.: Assessing requirements to scale to practical quantum advantage (2022). arXiv preprint arXiv:2211.07629 (2022)
37. Murali, P., Debroy, D.M., Brown, K.R., Martonosi, M.: Architecting noisy intermediate-scale trapped ion quantum computers. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 529–542 (2020). IEEE
38. Quetschlich, N., Burgholzer, L., Wille, R.: Compiler optimization for quantum computing using reinforcement learning. In: 2023 60th ACM/IEEE Design Automation Conference (DAC), pp. 1–6 (2023). IEEE
39. Lin, W.-H., Tan, B., Niu, M.Y., Kimko, J., Cong, J.: Domain-specific quantum architecture optimization. IEEE J. Emerg. Sel. Topics Circuits Syst. **12**(3), 624–637 (2022)

40. Liang, C., Li, Z., Zhang, M., Han, L.: A superconducting quantum chip architecture design method for quantum programs. In: 2023 3rd International Symposium on Computer Technology and Information Science (ISCTIS), pp. 1115–1121 (2023). IEEE
41. Deb, A., Dueck, G.W., Wille, R.: Exploring the potential benefits of alternative quantum computing architectures. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **40**(9), 1825–1835 (2021). https://doi.org/10.1109/TCAD.2020.3032072
42. Theocharis, I., Krokidas, P., Gkatsis, V., Giannakopoulos, G.: Multi-fidelity bayesian optimization for efficiently sampling the design space of functionalized nanoporous materials. In: Proceedings of the 13th Hellenic Conference on Artificial Intelligence, pp. 1–4 (2024)
43. Chen, S., Bayrak, A.E., Sha, Z.: Distributed multi-agent Bayesian optimization for unknown design space exploration. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 88377, pp. 03–03040 (2024). American Society of Mechanical Engineers
44. Madsen, J., Stidsen, T.K., Kjaerulf, P., Mahadevan, S.: Multi-objective design space exploration of embedded system platforms. In: IFIP Working Conference on Distributed and Parallel Embedded Systems, pp. 185–194 (2006). Springer
45. Palesi, M., Givargis, T.: Multi-objective design space exploration using genetic algorithms. In: Proceedings of the Tenth International Symposium on Hardware/software Codesign, pp. 67–72 (2002)
46. Zhang, J., Chung, H.S.-H., Lo, A.W.-L., Huang, T.: Extended ant colony optimization algorithm for power electronic circuit design. IEEE Trans. Power Electron. **24**(1), 147–162 (2008)
47. Reagen, B., Hernández-Lobato, J.M., Adolf, R., Gelbart, M., Whatmough, P., Wei, G.-Y., Brooks, D.: A case for efficient accelerator design space exploration via bayesian optimization. In: 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 1–6 (2017). IEEE
48. Schafer, B.C., Takenaka, T., Wakabayashi, K.: Adaptive simulated annealer for high level synthesis design space exploration. In: 2009 International Symposium on VLSI Design, Automation and Test, pp. 106–109 (2009). IEEE
49. Wang, G., Gong, W., DeRenzi, B., Kastner, R.: Design space exploration using time and resource duality with the ant colony optimization. In: Proceedings of the 43rd Annual Design Automation Conference, pp. 451–454 (2006)
50. Nardi, L., Koeplinger, D., Olukotun, K.: Practical design space exploration. In: 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 347–358 (2019). IEEE
51. Helsen, J., Steudtner, M., Veldhorst, M., Wehner, S.: Quantum error correction in crossbar architectures. Quantum Sci. Technol. **3**(3), 035005 (2018)
52. Morais Tejerina, A.: Mapping quantum algorithms in a crossbar architecture (2019)
53. Wang, C.-A., John, V., Tidjani, H., Yu, C.X., Ivlev, A., Déprez, C., Riggelen-Doelman, F., Woods, B.D., Hendrickx, N.W., Lawrie, W.I., et al.: Operating semiconductor quantum processors with hopping spins. arXiv preprint arXiv:2402.18382 (2024)
54. Paraskevopoulos, N., Almudever, C.G., Feld, S.: besnake: A routing algorithm for scalable spin-qubit architectures. IEEE Trans. Quantum Eng. (2024)
55. Quetschlich, N., Burgholzer, L., Wille, R.: Predicting good quantum circuit compilation options. arXiv preprint arXiv:2210.08027 (2022)
56. Nishio, S., Pan, Y., Satoh, T., Amano, H., Meter, R.V.: Extracting success from ibm's 20-qubit machines using error-aware compilation. ACM J. Emerg. Technol. Comput. Syst. **16**(3), 1–25 (2020)
57. Paraskevopoulos, N., Steinberg, M., Undseth, B., Sarkar, A., Vandersypen, L.M.K., Xue, X., Feld, S.: Near-term spin-qubit architecture design via multipartite maximally entangled states. PRX Quantum **6**, 020307 (2025). https://doi.org/10.1103/PRXQuantum.6.020307
58. Philips, S.G., Madzik, M.T., Amitonov, S.V., Snoo, S.L., Russ, M., Kalhor, N., Volk, C., Lawrie, W.I., Brousse, D., Tryputen, L., et al.: Universal control of a six-qubit quantum processor in silicon. Nature **609**(7929), 919–924 (2022)
59. Xue, X., Russ, M., Samkharadze, N., Undseth, B., Sammak, A., Scappucci, G., Vandersypen, L.M.: Quantum logic with spin qubits crossing the surface code threshold. Nature **601**(7893), 343–347 (2022)
60. Noiri, A., Takeda, K., Nakajima, T., Kobayashi, T., Sammak, A., Scappucci, G., Tarucha, S.: Fast universal quantum gate above the fault-tolerance threshold in silicon. Nature **601**(7893), 338–342 (2022)

61. Undseth, B., Xue, X., Mehmandoost, M., Rimbach-Russ, M., Eendebak, P.T., Samkharadze, N., Sammak, A., Dobrovitski, V.V., Scappucci, G., Vandersypen, L.M.: Nonlinear response and crosstalk of electrically driven silicon spin qubits. Phys. Rev. Appl. **19**(4), 044078 (2023)
62. Heinz, I., Burkard, G.: Crosstalk analysis for single-qubit and two-qubit gates in spin qubit arrays. Phys. Rev. B **104**(4), 045420 (2021)
63. Jirovec, D., Fariña, P.C., Reale, S., Oosterhout, S.D., Zhang, X., Morozova, E., Snoo, S., Sammak, A., Scappucci, G., Veldhorst, M., et al.: Exchange cross-talk mitigation in dense quantum dot arrays. arXiv preprint arXiv:2503.23846 (2025)
64. Hendrickx, N.: Qubit arrays in germanium (2021)
65. Bandic, M., Almudever, C.G., Feld, S.: Interaction graph-based characterization of quantum benchmarks for improving quantum circuit mapping techniques. Quantum Mach. Intell. **5**(2), 40 (2023)
66. Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A., Duncan, R.: t| ket>: a retargetable compiler for nisq devices. Quantum Sci. Technol. **6**(1), 014003 (2020)
67. Qiskit contributors: Qiskit: An Open-source Framework for Quantum Computing (2023). https://doi.org/10.5281/zenodo.2573505
68. Khammassi, N., Ashraf, I., Someren, J., Nane, R., Krol, A., Rol, M.A., Lao, L., Bertels, K., Almudever, C.G.: Openql: a portable quantum programming framework for quantum accelerators. ACM J. Emerg. Technol. Comput. Syst. **18**(1), 1–24 (2021)
69. Salm, M., Barzen, J., Leymann, F., Weder, B., Wild, K.: Automating the comparison of quantum compilers for quantum circuits. In: Symposium and Summer School on Service-Oriented Computing, pp. 64–80 (2021). Springer
70. Developers, C.: Cirq. Zenodo (2023)
71. Computing, R.: Pyquil documentation. http://pyquil.readthedocs.io/en/latest, 64–65 (2019)
72. JavadiAbhari, A., Patil, S., Kudrow, D., Heckey, J., Lvov, A., Chong, F.T., Martonosi, M.: Scaffcc: A framework for compilation and analysis of quantum computing programs. In: Proceedings of the 11th ACM Conference on Computing Frontiers, pp. 1–10 (2014)
73. Chong, F.T., Franklin, D., Martonosi, M.: Programming languages and compiler design for realistic quantum hardware. Nature **549**(7671), 180–187 (2017)
74. Wu, X.-C., Khalate, P., Schmitz, A., Premaratne, S., Rasch, K., Daraeizadeh, S., Kotlyar, R., Ren, S., Paykin, J., Rose, F., et al.: Intel quantum sdk version 1.0: extended c++ compiler, runtime and quantum hardware simulators for hybrid quantum-classical applications. In: APS March Meeting Abstracts, vol. 2023, pp. 08–005 (2023)
75. Lin, C.-C., Chakrabarti, A., Jha, N.K.: Qlib: quantum module library. ACM J. Emerg. Technol. Comput. Syst. **11**(1), 1–20 (2014)
76. Coppersmith, D.: RC 19642 (07/12/94) Mathematics IBM Research Report An Approximate Fourier Transform Useful in Quan-tum Factoring. Technical report (2002)
77. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. STOC '96, pp. 212–219. Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237814.237866
78. Cross, A.W., Bishop, L.S., Sheldon, S., Nation, P.D., Gambetta, J.M.: Validating quantum computers using randomized model circuits. Phys. Rev. A **100**(3), 032328 (2019)
79. Cuccaro, S.A., Draper, T.G., Kutin, S.A., Petrie Moulton, D.: A new quantum ripple-carry addition circuit. Technical report (2004)
80. Vedral, V., Barenco, A., Ekert, A.: Quantum networks for elementary arithmetic operations. Phys. Rev. A **54**(1), 147–153 (1996). https://doi.org/10.1103/PhysRevA.54.147
81. Bernstein, E., Vazirani, U.: Quantum complexity theory. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing. STOC '93, pp. 11–20. Association for Computing Machinery, New York, NY, USA (1993). https://doi.org/10.1145/167088.167097
82. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Technical report (1983)
83. Suman, B., Kumar, P.: A survey of simulated annealing as a tool for single and multiobjective optimization. Nat. Publ. Group (2006). https://doi.org/10.1057/palgrave.jors.2602068
84. Frazier, P.I.: A tutorial on bayesian optimization. arXiv preprint arXiv:1807.02811 (2018)
85. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: a review of Bayesian optimization. Proc. IEEE **104**(1), 148–175 (2015)

86. Garrido-Merchán, E.C., Hernández-Lobato, D.: Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. Neurocomputing **380**, 20–35 (2020). https://doi.org/10.1016/j.neucom.2019.11.004
87. Ungredda, J., Branke, J.: Bayesian Optimisation for Constrained Problems (2021)
88. Frazier, P.I.: A Tutorial on Bayesian optimization (2018)
89. Zbigniew, M.: Genetic algorithms+ data structures= evolution programs. Comput. Stat. **24**, 372–373 (1996)
90. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006). https://doi.org/10.1109/MCI.2006.329691
91. Shobaki, G., Gordon, V.S., McHugh, P., Dubois, T., Kerbow, A.: Register-pressure-aware instruction scheduling using ant colony optimization. ACM Trans. Archit. Code Optim. **19**(2) (2022). https://doi.org/10.1145/3505558