Routing Techniques for Error-Corrected Silicon Spin Qubit Quantum Architectures

Julian Shen*, Ludwig Schmid*, Robert Wille*†,

*Chair for Design Automation, Technical University of Munich, Munich, Germany
†MQSC, Garching near Munich, Germany
{julian.shen, ludwig.s.schmid, robert.wille}@tum.de
https://www.cda.cit.tum.de/research/quantum

***Abstract*—Silicon spin qubits have emerged as a promising qubit technology due to their favorable scaling and fabrication properties. However, efficiently compiling quantum circuits onto spin qubit platforms remains challenging, particularly when accounting for hardware constraints and the high sensitivity to static defects. Existing compilation approaches for spin qubits either largely ignore error correction, despite its critical role for large-scale quantum computation, or focus on low-level schedule constructions, missing a high-level compilation and routing for logical, error-corrected algorithms. To address this gap, we introduce a compilation framework for spin qubits based on the recent snakes on a plane model, which utilizes a 2D surface code and qubit teleportation to mitigate errors. Building on this model, we propose shortest-path and rotation-based algorithms as two novel classes of qubit-routing techniques, along with additional defect-handling and initial-mapping strategies. We evaluate both algorithms across diverse architectural settings and problem sizes, demonstrating that shortest-path methods excel in sparse, low-defect scenarios, while rotation-based approaches perform better in high-density environments. An open-source implementation of our framework is publicly available on GitHub as part of the Munich Quantum Toolkit (MQT) at https://github.com/munich-quantum-toolkit/spin-qubit-routing.**

I. Introduction

Quantum computing is steadily progressing towards practical realization. As the field continues to advance, research into scalable quantum architectures has explored a diverse range of different qubit technologies [1]–[5]. Among these, *silicon spin qubits* [6] stand out as a particularly promising candidate due to their favorable manufacturing and scaling properties [7], [8], as well as their long coherence times [9] and increasing gate fidelities [10]–[12].

However, in order to utilize these technologies, quantum circuits must first be compiled for the physical hardware before execution. This process involves several steps, including initial qubit mapping and routing operations to ensure that all two-qubit interactions are realized in the correct order. In recent years, compilation approaches for spin qubit platforms have emerged, largely tailored to specific hardware models, such as the crossbar architecture [13], [14] or shuttling busses [15], [16]. A key limitation of these proposed models is that they focus on compiling physical circuits directly onto the hardware, targeting the NISQ era and neglecting recent developments in *Quantum Error Correction* (QEC). Pataki et al. [17] pioneered QEC compilation for spin qubits, focusing on surface code stabilizer extraction cycles on the crossbar architecture. Nevertheless, their work still misses a high-level compilation and routing for logical, error-corrected algorithms. Despite the importance of QEC for scalability, there is currently no compilation framework for spin qubits that explicitly targets routing for error-corrected architectures.

To close this research gap, we explore compilation strategies to optimize qubit routing in spin qubit architectures based on the *snakes on a plane* model recently introduced by Siegel et al. [18], which provides an intrinsic mechanism for error correction referred to as *snake surgery*. Based on this framework, we first identify important routing challenges of spin qubits and formalize them using a graph-based abstraction of the original model. This formalization transforms the physical routing problem into a classical path-finding problem which can then be tackled with well-known graph algorithms, resulting in two novel routing approaches for spin qubits: *path algorithms* and *rotation algorithms*. Additionally, we propose two complementary defect-handling mechanisms for robust routing under noise, and we incorporate initial-mapping techniques to reduce the routing overhead. These components form the proposed framework that compiles quantum circuits into executable schedules for the snakes on a plane model.

In an empirical evaluation, we examine how the proposed algorithms perform across different architectural settings and problem sizes, revealing that path algorithms are better suited for scenarios with sparse qubit density and low-defect architectures, whereas rotation algorithms emerge as the more effective choice in high-qubit-density environments.

In summary, we make the following contributions:

- We formalize physical routing for spin qubit architectures as a graph-based path-finding problem, bridging hardware constraints with classical computer science methods.
- We propose two new qubit routing strategies: path algorithms and rotation algorithms.
- Based on this, we introduce the first high-level routing framework for spin qubit architectures that explicitly incorporates error correction mechanisms.
- We empirically evaluate the proposed framework across varying architectural conditions and problem sizes.

This paper is structured as follows: Sec. II reviews spin qubit quantum computing and the snakes on a plane model. Sec. III identifies key routing challenges and formulates the core problem based on a graph formalization. Sec. IV outlines the proposed solution strategies with technical implementation details provided in Sec. V. The compilation algorithms are then evaluated in Sec. VI, and Sec. VII concludes this paper.

II. Background

We begin with an overview of spin qubit quantum computers, followed by a discussion of the main error sources in silicon spin qubit devices and a brief review of quantum error correction techniques. After that, we introduce the snakes on a plane architecture, which serves as the central computational model considered in this work.

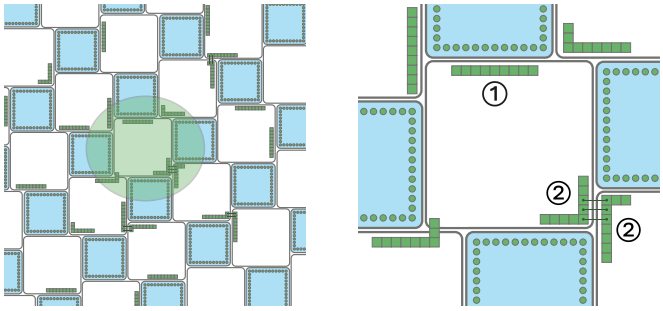
A. Spin Qubit Quantum Computers and Compilation

Spin qubit quantum computers store information in the *spin state* of single electrons confined in a silicon quantum dot [19]. The spin can either be in a spin-up or spin-down state, corresponding to the quantum states $|0\rangle$ and $|1\rangle$, respectively. Spin qubits are typically realized in silicon-germanium heterostructures or silicon metal-oxide-semiconductor devices, where precise control of the electron's local environment leads to long coherence times and high-fidelity operations [11]. Qubit manipulation is achieved using magnetic or electric fields, allowing the realization of both single-qubit and two-qubit gate operations [11]. Recent experimental results have demonstrated gate fidelities exceeding the thresholds required for quantum error correction in silicon spin qubit platforms [12]. Measurement in spin qubit systems is typically performed via spin-to-charge conversion [20], [21]. In this approach, the quantum dot containing the spin qubit is tunnel-coupled to an electron reservoir. By tuning the gate voltages, only one spin state is energetically allowed to tunnel out of the quantum dot. As a result, the spin state is converted into a detectable charge configuration corresponding to the presence or absence of an electron. This technique enables high-fidelity readout and is well suited for scalable silicon-based quantum dot architectures [21].

However, performing two-qubit gates in spin qubit architectures requires the participating qubits to be physically close to one another, making qubit shuttling necessary. One promising shuttling approach is the conveyor belt technique, which creates a moving confinement potential that can trap and continuously transport single electrons across the device [22]. Beyond the physical realization of qubit transport, several abstract models for spin qubit quantum computers have been proposed [13]–[15], [23], [24]. These models enable the development of compilation techniques that compute efficient schedules for qubit movements, ensuring that required qubit interactions occur in the correct order while minimizing routing overhead. However, none of the referenced models address errors or provide mechanisms for detecting or correcting them, leaving the compilation methods equally unaware.

B. Errors in Spin Qubit Quantum Computers

The most detrimental source of errors to be considered in silicon spin qubit devices are *charge noises* [11], [25], [26]. Charge noise arises from random fluctuations of charges between defects spread across the device and effectively induces uncontrolled variations in the magnetic field landscape, leading to dephasing errors. In general, charge noise can be divided into the following two categories:



(a) General layout. (b) Close-up of the green region.

Fig. 1: Sketch of the snakes on a plane architecture [18].

1) **Abrupt Charge Disturbance**: Local charge rearrangements that cause sudden and severe phase errors.
2) **Small-Amplitude Fluctuation**: Slow and gradual variation in the magnetic field landscape that leads to mild but persistent dephasing.

One common approach to mitigate such errors is the use of *Quantum Error-Correcting Codes* (QECCs). QECCs protect quantum information by encoding the qubits of a quantum circuit, referred to as *logical qubits*, into entangled states of multiple *physical qubits*. This redundancy allows certain errors affecting individual physical qubits to be detected and corrected without directly measuring or disturbing the encoded logical quantum state. Many QECCs are formulated within the *stabilizer formalism* [27], where a set of commuting operators, called stabilizers, define the valid subspace of the encoded logical qubit. By repeatedly measuring these stabilizers, one obtains an error syndrome that reveals the presence and location of errors [28] while preserving the logical information. A prominent example of such a code is the *surface code* [29], which arranges qubits on a two-dimensional lattice and uses local stabilizer measurements to detect both bit-flip and phase-flip errors. Surface codes are particularly attractive for near-term quantum hardware, including spin qubit platforms, because they only require nearest-neighbor interactions and exhibit relatively high error thresholds.

Recently, Siegel et al. [18] have introduced a new architectural model for spin qubit quantum computers called *snakes on a plane*. This model explicitly incorporates surface-code-based error correction to mitigate the aforementioned noise sources and employs a dedicated protocol for fault-tolerant qubit transport, called *snake surgery*, as discussed in the following.

C. Snakes on a Plane Model

In the snakes on a plane model [18], the device architecture is modeled as a 2D latticework formed of low-dimensional array strands that occasionally meet at junctions. A schematic overview of the architecture is provided in Fig. 1. Error correction based on surface codes is implemented using static ancilla qubits (green dots) located in blue squares, which facilitate the required stabilizer measurements. Logical qubits are built from multiple physical qubits (green squares), creating snakes that can be shuttled along the inner edges of the white squares in the latticework. When a snake reaches a junction, it can either

remain in its current square or move to a neighboring square. In general, snakes move for two main reasons in this model:

- *Error Correction*: When a snake moves along the outer edge of a blue square, referred to as a *stabilizer edge*, its physical qubits face the ancilla qubits located along the edge. Stabilizer measurements can then be performed between pairs of qubits that are opposite to each other [18]. By repeatedly shuttling the snake back and forth along a stabilizer edge and performing the stabilizer measurements in an appropriate sequence, a surface code can be implemented and the logical qubit represented by the snake remains error-free.

  Example 1. *The snake denoted with ① in Fig. 1b is being shuttled along a stabilizer edge while performing stabilizer measurements using the green ancilla qubits.*

- *Interaction*: Two snakes can be moved to an edge where they face each other, enabling them to interact and realize a logical (transversal) two-qubit gate. Edges, where snakes can be brought into parallel alignment, are therefore referred to as *interaction edges*.

  Example 2. *The two snakes denoted with ② in Fig. 1b interact with each other at an interaction edge. Qubit interactions (black connections) can be performed whenever the respective qubits are opposite to each other.*

If a snake permanently remains at a stabilizer edge, errors can be mitigated through repeated stabilizer measurements. However, when a snake travels over a larger distance, additional mechanisms are required to handle the two noise types described previously in Sec. II-B.

Abrupt charge disturbances are modeled as defective edges that can occur within the snakes on a plane architecture. Each functioning edge may become defective with probability p_{defect}, while each defective edge recovers with probability p_{recovery}, which models the natural decay of dephasing effects on shuttling links. If a snake traverses a defective edge, it acquires an error and, therefore, does not arrive at its destination in an error-free state. In this case, the shuttling operation can be reversed using a dedicated defect-handling protocol called *snake surgery* [18]. In this protocol, the length of the snake is first doubled before shuttling using lattice surgery [30], creating an entangled pair consisting of two parts:

- *Tail*: The original snake, which remains at the initial stabilizer edge and continues to be stabilized and protected against errors.
- *Head*: A newly created extension of the snake that is shuttled across the device towards the destination.

Once the head reaches its destination, a measurement of either the head or the tail determines where the logical state is teleported to. If the route was error-free, the state is teleported to the head, otherwise, it is teleported back to the tail, effectively reversing the shuttling process.

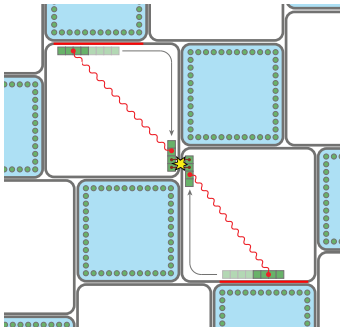


Fig. 2: Illustration of the snake surgery protocol [18].

Example 3. *Consider the scenario illustrated in Fig. 2. To enable the interaction between the two snakes, snake surgery is performed on each snake, splitting it into an entangled head-tail pair, indicated by squiggly lines. The heads are then shuttled to the designated interaction edge, while the tails remain stabilized at their original positions.*

Since snake surgery requires the detection of errors in order to determine when the shuttling process must be reversed, the snakes on a plane model further incorporates two dedicated mechanisms for detecting defects [18]:

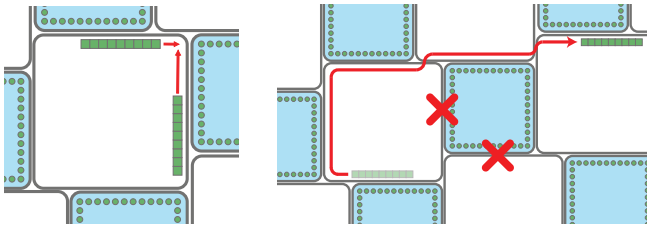
- *Monitor Qubits*: These are additional qubits which are prepared in the $|+\rangle$ state and then interlaced with the data qubits of a snake. The qubits are then shuttled together, ideally leaving the monitor qubits unchanged, but defects on the device may cause them to accumulate a phase during the process. In this case, measuring these monitor qubits in the X-basis allows us to detect whether the snake has encountered a defect or not.
- *Complementary Gap*: This leverages the fact that ancilla qubits continuously gather information about potential errors by measuring stabilizers throughout the shuttling process. The complementary gap [31] characterizes the confidence of a surface code decoder that it can determine the cause of an observed error. In case of a high-risk event, snake surgery can be used to repeat the shuttling process and to ensure logical correctness.

Small-amplitude fluctuations can be mitigated using the *singlet-triplet qubit encoding* [32], where a logical qubit is encoded into two electron spins. Hereby, the logical states depend on the relative spin orientation rather than the absolute spin directions of the electrons, making singlet-triplet qubits far less sensitive to uniform magnetic-field fluctuations [20].

In this contribution, we adopt the snakes on a plane model to represent an error-correcting architecture and focus on the routing problem arising from snake surgery.

III. The Snake Routing Problem

Following the snakes on a plane model reviewed above, which incorporates error correction as an intrinsic feature, the problem of efficient compilation still remains open. In this work, we focus on two-qubit gates that require the participating snakes to be physically close to each other. Other QEC-related compilation tasks, e.g., magic-state creation and



(a) Snake collision. (b) Correct handling of defective edges.

Fig. 3: Routing challenges in the snakes on a plane model [18].

distribution [33], are not the focus of this work but are interesting follow-up questions that can be addressed using the results presented here. Consequently, the compilation in this setting reduces to a routing problem. This section begins by outlining physical routing challenges and then introduces a graph-based formalization, enabling the problem to be formulated as a path-finding problem. This formalization allows the routing task to be tackled using established algorithms.

A. Routing Challenges

1) Snake Collision: Since snakes are not permitted to overtake one another, they may collide during shuttling operations. Preventing such collisions requires careful coordination of their movements, particularly in densely populated grids with a large number of snakes.

Example 4. *Consider the scenario illustrated in Fig. 3a. If both snakes move simultaneously in the indicated directions, they will collide with each other. To avoid this conflict, one of the snakes must wait until the other one has cleared the intended path before proceeding.*

2) Tail Blockage of Stabilizer Edges: When a snake is shuttled over a larger distance, its tail remains anchored at the originating stabilizer edge. This allows continuous stabilizer measurements to be performed on the tail, ensuring that the shuttling process can be repeated if the head encounters any error. Consequently, the originating stabilizer edge remains occupied and cannot be used to route other snakes until the tail is teleported away.

Example 5. *Consider the scenario illustrated previously in Fig. 2. As a result of the snake surgery protocol, the red edges remain occupied by the stationary tails and are therefore blocked for routing other snakes until the tails are teleported away.*

3) Defect Handling: Edges that are known to be defect should be avoided during shuttling in order to reduce the need for snake surgery teleportations and to minimize the required routing overhead.

Example 6. *Consider the scenario illustrated in Fig. 3b. The red crosses indicate defective shuttling links that must be avoided. This can be achieved by routing the snake along the red path.*

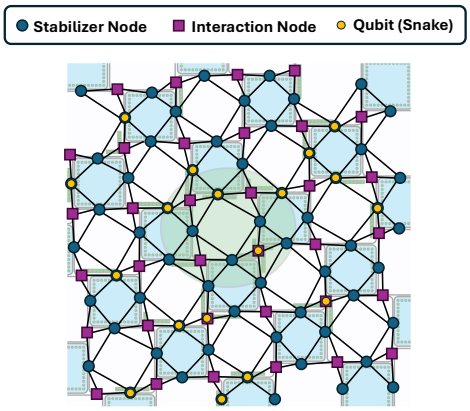


Fig. 4: The graph formalization process starting from the snakes on a plane model as shown in Fig. 1a.

B. Graph Formalization and Problem Formulation

To systematically address the snake routing problem, we formalize the underlying architecture into a graph representation, allowing the physical routing problem to be formulated as a path-finding problem. This formalization process is illustrated in Fig. 4. Based on the physical architecture shown in Fig. 1, the main elements of the snakes on a plane model are mapped to graph elements as follows:

- Snakes are represented as single qubits (yellow circles) that can occupy exactly one node at any given time.
- Stabilizer edges become blue circular nodes, called *stabilizer nodes*. Stabilizer nodes can be occupied by at most one snake at a time. Occupying a stabilizer node corresponds to performing stabilizer measurements.
- Interaction edges become purple square nodes, called *interaction nodes*. These nodes are only occupied when two snakes intend to perform an interaction. Accordingly, interaction nodes have a capacity of two and must be occupied by exactly two snakes simultaneously to realize a two-qubit interaction.

The edges of the graph are defined in accordance with the snakes on a plane model and each edge can only be traversed by at most one qubit at each point in time. In particular, edges between stabilizer nodes represent the option to shuttle a snake along stabilizer edges within a square of the original architecture without engaging in an interaction.

Similar to the original snakes on a plane model, defects on the device are modeled by defective edges that can occur with a certain probability p_{defect} during the shuttling process and recover with probability p_{recovery}. If a qubit traverses a defective edge along its route, it collects an error and is teleported back to its original starting position via snake surgery upon reaching its destination. Therefore, the starting positions of shuttled qubits may only be occupied by other qubits if the shuttled qubit undergoes a snake surgery teleportation to its destination. Otherwise, these starting nodes remain occupied by the respective tails.

Based on this formalization and the physical challenges reviewed in Sec. III-A, we derive the following routing problem:

Snake Routing Problem

Given an initial placement of qubits on stabilizer nodes and a sequence of required qubit interactions, determine a routing schedule that realizes these interactions while optimizing routing performance and satisfying the following properties:

1) The capacities of nodes and edges are respected.
2) Qubits do not collide with other qubits or their respective tails.
3) Qubits remain on stabilizer nodes whenever possible to enable error correction through stabilizer measurements. They may only leave stabilizer nodes for the time span of their qubit interaction.
4) Only qubits that are error-free may interact. Defective qubits are routed again using snake surgery. Noisy edges have to be avoided.
5) The logical order of interactions is respected.

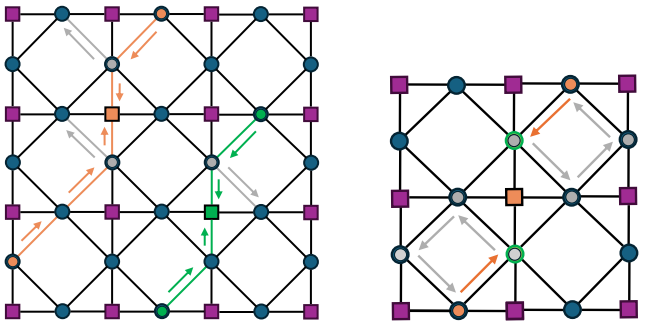
IV. Proposed Solution

Having established a graph-based formalization of the snake routing problem, it can now be tackled using well-known graph algorithms. We propose two classes of algorithms suitable for this problem: *path algorithms* and *rotation algorithms*. Their robustness and efficiency are further improved through the use of defect-handling mechanisms and initial-mapping techniques. Together, these components form the compilation framework proposed in this work, enabling the automatic translation of quantum circuit instructions into executable schedules for the snakes on a plane architecture. As discussed above, we assume that magic-state preparation and decomposition are already resolved and focus on the routing task for the corresponding logical qubits. As a result, the input to our compilation flow is a sequence of interactions between given logical qubits.

A. Path and Rotation Algorithms

This section describes the workflow of the proposed path and rotation algorithms used to perform qubit routing on the snakes on a plane model. Both algorithms share a common structure, consisting of five major building blocks that transform a given sequence of qubit interactions into an executable shuttling schedule. The main differences between the two approaches lie in how they resolve path obstructions and handle defects. The general idea of both algorithms can be summarized into the following steps:

1) **Layer Creation**: To enable the simultaneous shuttling of qubits without violating the prescribed interaction logic, the algorithms first partition the given sequence of interactions into layers.
2) **Path Computation**: For each interaction in a layer, a Breadth-First Search (BFS) is executed to identify the interaction node that minimizes the summed distance to both qubits. This node is called the *meeting node* of that qubit pair. Furthermore, qubits are classified either as *active qubits* or *idle qubits*, depending on whether they participate in an interaction in the current layer or not. During this step, idle qubits are ignored in order to compute the most efficient paths for active qubits to their meeting nodes using an A^* search.



(a) Path algorithm scheme. (b) Rotation algorithm scheme.

Fig. 5: Illustration of path and rotation schemes.

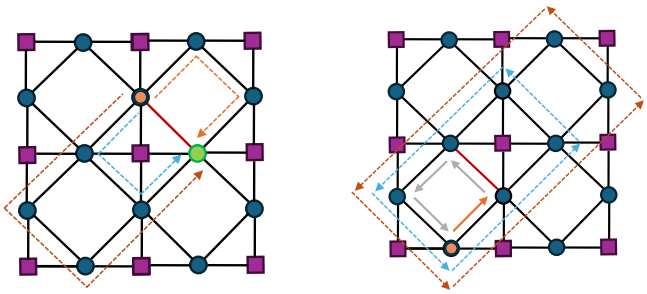
3) **Relocation of Idle Qubits**: Since idle qubits were not considered during path computation in the previous step, some of the paths may be obstructed by idle qubits. To resolve this, these *blocking idle qubits* have to be relocated to clear the paths. This is where the two algorithms differ from each other:
   - Path algorithms first relocate blocking idle qubits to the nearest free stabilizer nodes that are not on any other paths, before any active qubits are moved.

     Example 7. *Consider the scenario illustrated in Fig. 5a. Idle qubits are shown in gray, while the two pairs of active qubits and their respective paths to the meeting nodes are highlighted in green and orange. Idle qubits located along these paths are relocated to the nearest available stabilizer nodes.*

   - Rotation algorithms relocate idle qubits simultaneously by rotating them along diamond-shaped cycles.

     Example 8. *Consider the scenario illustrated in Fig. 5b. The orange qubits are scheduled to interact at the central interaction node and must first move onto the green nodes to reach it. Idle qubits blocking these paths are rotated away along diamonds, simultaneously to the movements of the orange qubits.*

     This simultaneous relocation is only possible if snake surgery teleportation is performed at every hop. Without teleporting the tail at each step, previously occupied nodes would remain blocked, preventing other qubits from moving onto them. Continuously teleporting the tail removes tail blockage entirely, at the cost of an increased number of teleportation operations.
4) **Defect Handling**: Before the qubits are physically shuttled, the defect model samples all edges. Edges that become defect along the paths of active or idle qubits are handled by the following defect-handling strategies:
   - **Waiting**: The blocked qubit remains at its current node and waits until the defective edge has recovered. This strategy can be used by both algorithms, referred to as the *path* and *rotation algorithm with waiting*.



(a) Example for local circumvention used by path algorithms.

(b) Example for dynamic cycles used by rotation algorithms.

Fig. 6: Illustration of dynamic path adaption schemes.

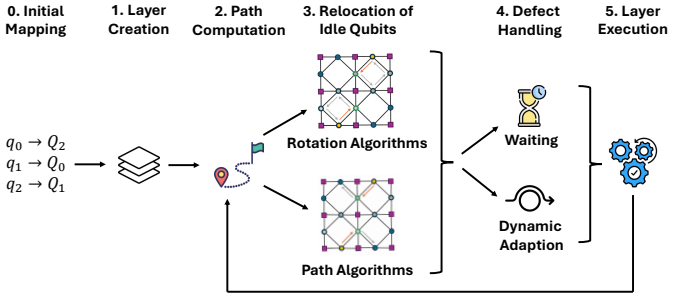


Fig. 7: Overview of the complete compilation flow.

- **Dynamic Adaption**: Path algorithms can try to reroute blocked qubits using a local circumvention, referred to as the *path algorithm with rerouting*.

  Example 9. *In Fig. 6a, the orange qubit tries to reach the green node but is blocked by the red defective edge. Path algorithms can try to reroute the qubit around the defect along the illustrated circumvention paths.*

  Rotation algorithms can try to avoid defective links by shuttling qubits along larger cycles rather than the default diamond-shaped paths, referred to as the *rotation algorithm with dynamic cycles*.

  Example 10. *In Fig. 6b, the orange qubit is scheduled to move along the highlighted cycle, which contains a defective edge (red). Rotation algorithms may instead route the qubit along a larger cycle, e.g. blue or brown, to bypass this faulty edge.*

  However, if these dynamic path adoptions also fail, the waiting mechanism has to be used as a fallback.

5) **Layer Execution**: In the last step, the shuttling plan is executed for the current layer. The procedure then continues with the next layer, restarting from Step 2.

Once a complete routing schedule has been determined on the graph formalization, the corresponding qubit movements can be directly mapped back to snake movements in the snakes on a plane model. Hence, this translation yields a complete routing plan for the physical architecture.

B. Initial Qubit Mapping

It has been shown that the initial qubit mapping can have a substantial impact on the efficiency of the final routing schedule [34], [35]. Consequently, the proposed framework offers the following three initial-mapping techniques:

- **Random Mapping (Baseline)**: The logical qubits from the quantum circuit are randomly assigned to physical qubits on the device.
- **Reverse Traversal Mapping** [36]: Start with a random mapping and execute the reversed quantum circuit on that mapping. The resulting qubit placement of the execution is then used as the initial mapping for the original quantum circuit.
- **Interaction-based Mapping** [37]: First, the qubit with the highest number of interactions is placed at the center of the graph. Qubits that interact with it are then placed around it recursively, ensuring that frequently interacting qubits are mapped to nearby locations.

C. Resulting Compilation Flow

By combining the presented initial-mapping techniques (as Step 0) with the proposed routing algorithms, we obtain a complete compilation framework for the snakes on a plane architecture. This framework automatically translates quantum circuits into executable shuttling schedules for spin qubit devices while respecting the physical constraints of the hardware. The complete compilation flow of the proposed framework is shown in Fig. 7.

V. Implementation Details

After presenting the core structure and fundamental ideas of the proposed compilation algorithms, we now examine in greater detail how the key concepts are implemented at a technical level. In particular, computing collision-free paths for active qubits to their meeting nodes and determining relocation paths for idle qubits require careful algorithmic design to ensure efficient and collision-free routing under the architectural constraints of the snakes on a plane model.

A. Path Constraints

A primary objective of the routing algorithms is to prevent collisions between qubits during shuttling operations. Furthermore, highly congested configurations should be avoided, as they increase the complexity of qubit relocation and reduce the efficiency of the routing process. To address these challenges, we introduce the following path constraints that limit potential collision scenarios and reduce congestion during routing:

1) *Unique Meeting Node*: Each active qubit pair within a layer is assigned a distinct meeting node. This prevents congestion and potential collisions in the vicinity of interaction nodes.
2) *Synchronization of Interactions*: Since the distances from the two qubits of a pair to their meeting node may differ, their arrival times may be misaligned. To synchronize the interaction, the qubit that arrives earlier waits at the stabilizer node immediately preceding the meeting node. We define this waiting point as its *target node*.

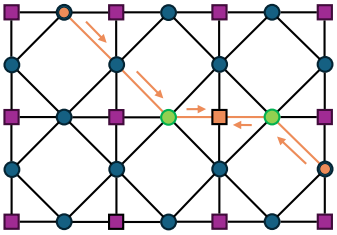


Fig. 8: Example for interaction synchronization.

Example 11. *Consider the scenario illustrated in Fig. 8. The path of the right orange qubit to the meeting node is shorter than the path of the left orange qubit. Consequently, the right qubit must wait at its target node (right green node) until the left qubit reaches its corresponding target node (left green node). Once both qubits have arrived, they can proceed to the interaction node to perform the interaction.*

3) *Unique Target Node*: Each active qubit is assigned a distinct target node to reduce congestion. Additionally, qubits are moved back to their target nodes after interaction. If two qubits had the same target node, this would lead to a collision.

The above constraints must be satisfied by both path and rotation algorithms. However, due to the differences in qubit relocation, the last constraint is algorithm-specific. Path algorithms teleport tails only when the corresponding heads reach their targets, in order to reduce the number of teleportations. Consequently, the following constraint must be satisfied:

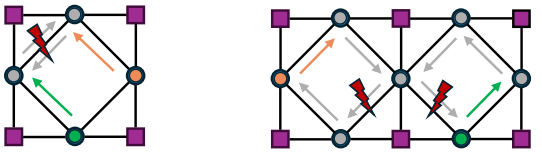
4) (Path Alg.) *Tail Blockage*: A routing path must not pass through the initial positions of other qubits that are being routed simultaneously, as these nodes remain occupied by stationary tails during the shuttling process.

In contrast, rotation algorithms employ a cyclic relocation scheme which requires snake surgery teleportation to be performed continuously to free occupied nodes. As a result, tail blockage is eliminated entirely. However, this introduces a different constraint:

4) (Rotation Alg.) *Rotation Conflicts*: Qubits routed simultaneously must not be assigned to the same rotation cycle, as this would lead to conflicts. Furthermore, neighboring cycles must also be rotated sequentially.

Example 12. *Consider the scenario illustrated in Fig. 9a. The orange and green active qubits rotate simultaneously within the same cycle, leaving no opportunity for the gray qubits to relocate. In such cases, the active qubits must be shuttled sequentially.*

Example 13. *Consider the scenario illustrated in Fig. 9b. The orange and green active qubits rotate simultaneously in neighboring cycles, forcing the central gray idle qubit to relocate in two conflicting directions at once. This conflict is resolved by executing the rotations sequentially rather than in parallel.*



(a) Single cycle conflict. (b) Neighboring cycles conflict.

Fig. 9: Examples for rotation conflicts.

The path computation procedure described in the following section is implemented in a way that it adheres to all of these path constraints.

B. Collision-Free Path Computation

To compute collision-free paths to the meeting nodes in Step 2 of the workflow illustrated in Sec. IV-A, both algorithms utilize an adapted A^* search in combination with a reservation table that maintains the current routing state. Each qubit is represented by a state (v, t), indicating that the qubit occupies node v at timestep t. The reservation table records which nodes and edges are occupied at each timestep. It is updated whenever a qubit changes its state, ensuring that conflicts between qubits are avoided during path planning. The cost of a path is defined as the tuple $(moves, time)$, capturing both the number of qubit moves and the elapsed time. Since physical qubit movement is more expensive than waiting, the number of moves is prioritized over time when comparing costs. To fully specify the adapted A^* search, we define the following components, in accordance with the standard formulation from Ref. [38]:

- *Successor Function*: From a qubit state (v, t), a qubit can either wait or move to a neighboring node. This leads to the following successor transitions that are possible:
  - Wait: $(v, t) \to (v, t+1)$ with cost $(0, 1)$
  - Move: $(v, t) \to (u, t+1)$ for $u \in N(v)$ with cost $(1, 1)$

  Here, $N(v)$ denotes the neighborhood of v. A successor is only valid, if it can be reached without collision according to the reservation table and if the traversed edge is not defective.
- *Cost Function*: Each node n maintains the accumulated cost $g(n) = (moves, time)$ where $g(n)$ represents the total cost incurred from the start node up to node n. Costs are compared lexicographically, prioritizing fewer moves over shorter time. This allows the search to prefer waiting over taking unnecessary detours.
- *Heuristic*: By labeling each node in the graph with a coordinate, we choose the Chebyshev distance [39] as an heuristic for assessing the distance from a node n to the target node: $h(n) = \max(|x_n - x_{\text{target}}|, |y_n - y_{\text{target}}|)$.
- *Evaluation Function*: Assembling these components, the evaluation key of the A^* search is:

$$f(n) = (g_{\text{moves}}(n) + h(n), g_{\text{time}}(n) + h(n))$$

All interactions for which no collision-free paths can be found, that satisfy the rules from Sec. V-A, are deferred to a new layer, which is inserted immediately after the current layer. This allows the algorithms to retry routing these

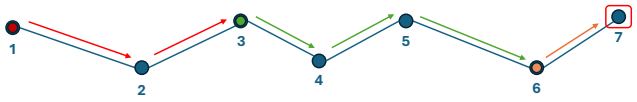


Fig. 10: The recursive relocation process.

interactions in a reduced interaction set. We refer to this newly created layer as the *spillover layer*. If routing fails for all interactions in a spillover layer, the algorithm repeatedly explores alternative, possibly non-optimal meeting and target node assignments until either a feasible path is found or a predefined retry threshold is exceeded.

C. Qubit Relocation

For rotation algorithms, relocating blocking idle qubits is straightforward, since they are simply rotated along the diamonds according to a predefined pattern. For path algorithms, however, relocation is more complex because idle qubits must be moved in a way that avoids collisions with other qubits. This issue can be addressed by reusing the path-computation procedure introduced in the previous section, with suitable constraints to account for the occupied nodes. For each idle qubit, the nearest available stabilizer node is chosen such that it is neither the start nor the target node of any active qubit and has not already been assigned to another idle qubit. Collision-free relocation paths are then computed using the same algorithm as described in Sec. V-B, while excluding all nodes currently occupied by active qubits. In the special case where a blocking idle qubit is itself obstructed by other non-blocking idle qubits, an additional recursive relocation procedure is required to clear the way for the active qubits. This process is illustrated in the following example.

Example 14. *Consider the scenario illustrated in Fig. 10. Here, nodes 1, 2, 4, and 5 are part of the paths computed for active qubits. Along the relocation path of the red blocking idle qubit to its destination (node 7), additional non-blocking idle qubits (green and orange) obstruct its routing. To resolve this situation, the idle qubits are relocated recursively, indicated by the colored arrows, until node 1 becomes available.*

If the path of some active qubits cannot be freed, e.g., due to a lack of available stabilizer nodes, the blocked qubits are placed into a spillover layer and executed separately.

For further technical details, we refer to the open-source implementation of our framework, which is publicly available as part of the *Munich Quantum Toolkit* (MQT) [40] at https://github.com/munich-quantum-toolkit/spin-qubit-routing.

VI. Evaluation

With the compilation framework for the snakes on a plane model in place, we now evaluate its practical performance. Although the previous sections described the framework's structure and algorithmic components, its behavior under different architectural constraints and problem sizes has not yet been quantified. The two routing paradigms introduced in Sec. IV-A, combined with different strategies for defect handling and initial mapping, lead to markedly different trade-offs in *execution time* and *routing overhead*. We therefore assess the framework using two complementary metrics: required timesteps, which capture execution time, and qubit movements, which reflect routing overhead. Based on these metrics, the evaluation is guided by the following research questions:

RQ1 (Qubit Density): How do path and rotation algorithms compare in terms of execution time and routing overhead as qubit density increases?

RQ2 (Adaptive Defect Handling): To what extent do dynamic adaption strategies improve robustness and execution time, and what drawbacks do they introduce in defective architectures?

RQ3 (Initial Qubit Mapping): How strongly does the choice of initial qubit mapping influence the routing performance of the proposed algorithms?

RQ4 (Scalability): How do the computational complexities of path and rotation algorithms scale with increasing input size, measured by the number of qubits?

A. Experimental Setup

To answer these research questions, we evaluated the proposed compilation framework on randomly generated quantum circuits composed exclusively of a given sequence of two-qubit interactions, with a fixed depth of 10 layers. Unless stated otherwise, the number of qubits is chosen as 9, which turns out to be the densest possible occupancy of the grid where all algorithms remain reliable (see Sec. VI-B). All experiments are conducted on an architecture grid with 24 stabilizer nodes as shown in Fig. 5a. This grid is large enough to provide a non-trivial routing problem while still being small enough to support extensive repeated evaluation. The default hardware parameters are set to a defect probability of $p_{\text{defect}} = 1\%$ and a recovery probability of $p_{\text{recovery}} = 25\%$. The values are chosen to represent a low but non-negligible fault rate characteristic of realistic near-term quantum hardware. The moderate recovery rate allows defects to persist long enough to affect execution, while still enabling gradual mitigation over time. To reduce the impact of measurement variability, results are averaged over 100 randomly generated sequences for each experiment. In addition, every computed schedule is verified to be collision-free and consistent with the routing constraints introduced in Sec. III-B. To prevent unbounded waiting in defective architectures, all algorithms use a retry threshold of 50 consecutive routing attempts. Runs that exceed this threshold are counted as failures. We conduct a series of four experiments, each designed to address one of the research questions above.

B. RQ 1: Impact of Qubit Density

To analyze how path and rotation algorithms perform under increasing congestion, we vary the number of qubits while keeping all other parameters to their default values. Since defective edges are not the limiting factor in this experiment, we compare the path algorithm with waiting and the rotation algorithm with waiting, as no specialized defect-handling mechanisms are required. The results of this experiment are shown in Fig. 11. Blue curves represent the path algorithm

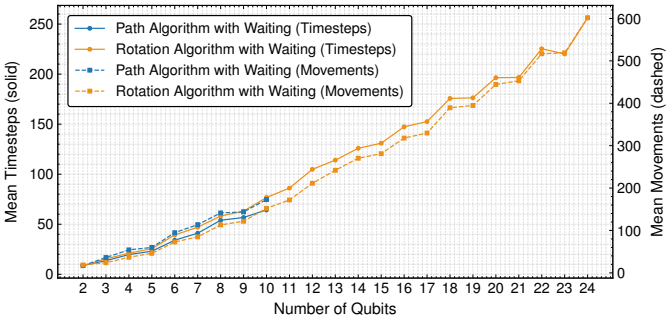


Fig. 11: Comparison of path and rotation schemes as qubit density increases.

with waiting, while orange curves correspond to the rotation algorithm with waiting. Timesteps and movements are shown in the same plot, with timesteps being displayed on the left y-axis (solid lines) and movements on the right y-axis (dashed lines). The two main observations are:

- The path algorithm requires more qubit movements, whereas the rotation algorithm incurs more timesteps.
- The rotation algorithm is more scalable than the path-based approach, maintaining functionality even under full grid occupancy, whereas the path algorithm yields results only for systems with up to 10 qubits within the retry limit.

These differences can be explained by the underlying mechanisms of the two approaches. The path algorithm exhibits a consistently higher movement count because it explicitly relocates idle qubits to clear routing paths. While this enables parallel execution once paths are established, resulting in relatively low timesteps, its dependence on available free stabilizer nodes makes it increasingly susceptible to congestion, ultimately leading to routing failures as the number of qubits grows. In contrast, the rotation algorithm avoids explicit relocation by leveraging cyclic rotations to dynamically free nodes. This allows for more direct routing and eliminates the costly detours typically associated with idle qubit relocation in path algorithms. As a result, the approach remains functional even under full occupancy, making it inherently more scalable. However, this advantage comes at the cost of frequent snake surgery teleportations at each movement step, which might introduce a technical overhead in practical implementations due to the required measurements. Apart from that, the reliance on cyclic rotations creates sequential dependencies (see Sec. V-A), increasing the total number of timesteps and limiting parallelism.

C. RQ 2: Impact of Adaptive Defect Handling

To assess the benefits and limitations of dynamic adaption strategies, we vary the amount of defects occurring on the device and evaluate the performance for both waiting-based and dynamically adaptive variants across both algorithm classes. Instead of considering p_{defect} and p_{recovery} independently, we express their combined effect through a single parameter E, denoting the expected fraction of functional edges. The results for path algorithms are shown in Fig. 12 and the results

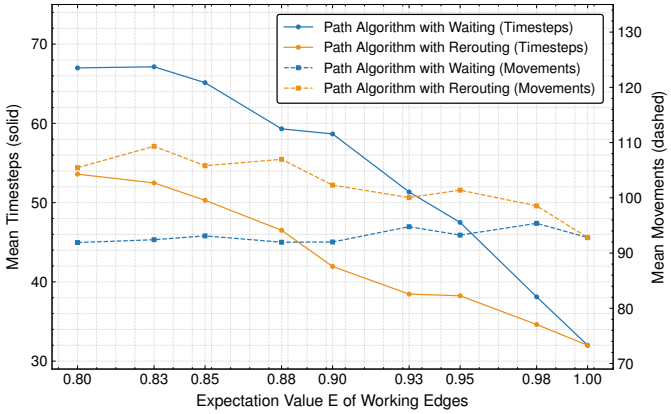


Fig. 12: Effects of dynamic adaption on path algorithms.

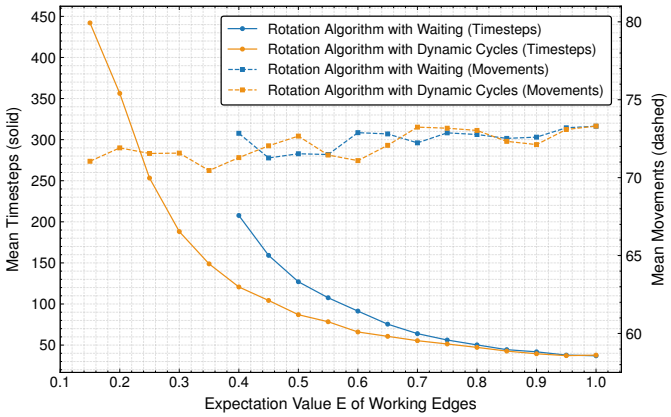


Fig. 13: Effects of dynamic adaption on rotation algorithms.

for rotation algorithms are illustrated in Fig. 13. From these results, the following key messages can be derived:

- Dynamic adaption reduces the number of required timesteps but increases the number of qubit movements for path algorithms.
- For rotation algorithms, the number of timesteps is also reduced by dynamic adaption but the differences in routing overhead are minimal.
- Dynamic adaptation further improves the robustness of rotation algorithms, which are already more robust than path algorithms.

These observations can be attributed to how each algorithm employs adaptive strategies in the presence of defects. In general, dynamic adaptation enhances robustness by actively exploiting alternative routing options, rather than passively waiting for defects to resolve. For rotation algorithms, this adaptability is realized through dynamic cycle extension. By forming larger cycles, qubits can bypass defective edges without requiring a large amount of additional movements in most cases. As a result, valid routing schedules can still be computed even at higher defect rates, which explains the improved robustness. Furthermore, reducing idle waiting contributes to fewer overall timesteps. In contrast, path algorithms rely on rerouting, which provides more limited benefits in

Algorithm	Mapping Strategy	∅ Timesteps	∅ Movements
Path Algorithm	Random Mapping	64.28 (baseline)	172.75 (baseline)
	Reverse Traversal Mapping	62.35 (↓ 3.00%)	161.35 (↓ 6.60%)
	Interaction-based Mapping	63.76 (↓ 0.81%)	166.53 (↓ 3.60%)
Rotation Algorithm	Random Mapping	256.28 (baseline)	601.58 (baseline)
	Reverse Traversal Mapping	246.31 (↓ 3.89%)	576.01 (↓ 4.25%)
	Interaction-based Mapping	247.08 (↓ 3.59%)	568.13 (↓ 5.56%)

Tab. 1: Evaluation of initial-mapping strategies.

terms of robustness, as the number of feasible alternative paths is typically smaller than the number of possible cycle adaptations. Additionally, when rerouting is feasible, it often introduces detours from optimal paths, leading to an increase in qubit movements. Nevertheless, similar to rotation-based approaches, actively exploring alternative routes helps to reduce idle waiting and thus lowers the total execution time.

D. RQ 3: Impact of Initial Qubit Mapping

Tab. 1 summarizes the average number of timesteps and movements for path and rotation algorithms with waiting under default conditions using the three mapping strategies included in our framework. Path algorithms were evaluated on 9 qubits, while rotation algorithms used a fully occupied grid.

Reverse traversal mapping is especially effective because it propagates the impact of earlier gates in reverse execution order, resulting in initial qubit placements that better anticipate future routing requirements. In contrast, interaction-based mapping places frequently interacting qubits close together, thereby reducing routing distances and minimizing unnecessary shuttling. Together, these approaches highlight how incorporating the interaction sequence can significantly improve the benefits of initial mapping.

E. RQ 4: Scalability Analysis

To analyze the impact of qubit density on runtime, we systematically vary the number of qubits and measure the average runtime per configuration across all runs for each of the four algorithms. The results in Fig. 14 show the following:

- The runtime of path algorithms increases with qubit density, while the runtime of rotation algorithms remains nearly constant at a very low level.
- The path algorithm with waiting exhibits a slightly higher runtime than the rerouting variant.

The higher runtime complexity of path algorithms stems from the need to explicitly compute collision-free paths for both active and idle qubits. In contrast, the rotation algorithms only compute paths for active qubits and subsequently handle idle qubits by applying rotations according to a predefined pattern. Furthermore, the slightly increased runtime of the path algorithm with waiting is due to its greater number of waiting cycles compared to the path algorithm with rerouting. Since it does not consider alternative paths, it requires additional algorithm iterations.

F. Implications and Practical Trade-Offs

This evaluation shows that path algorithms and their implementations might not scale to denser architectures, especially at high qubit densities. For such architectures, one should consider using the rotation-based approach or a modified path algorithm that keeps the path-computation complexity within certain bounds, at the cost of optimality. The development of such large-scale algorithms will be future work, based on the insights from the routing methods studied in this work.

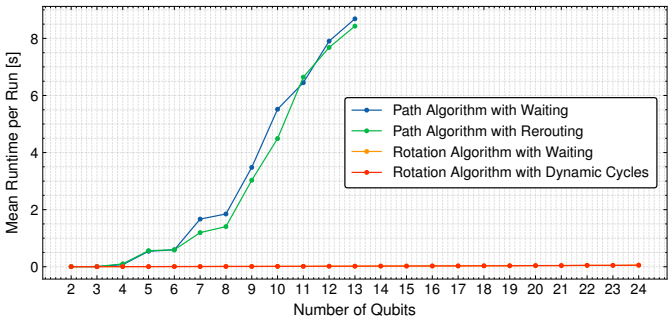


Fig. 14: Impact of qubit count on CPU runtime.

VII. Conclusion

In this work, we presented a compilation framework for spin qubit quantum computers based on the snakes on a plane architecture, modeling qubit shuttling as a constrained routing problem on a graph. Within this framework, we developed two routing strategies: path algorithms, which rely on explicit collision-free paths, and rotation algorithms, which use cyclic rotations to resolve congestion. Our evaluation shows complementary trade-offs between both approaches. Path algorithms achieve lower execution times in well-connected, low-defect settings but are sensitive to congestion and defects. In contrast, rotation algorithms remain robust under high qubit density and limited connectivity, at the cost of longer execution times, while requiring fewer qubit movements. We further demonstrated that adaptive defect-handling improves performance by reducing execution time, and that initial qubit mapping also has a large influence on routing efficiency. Together, this work constitutes one of the first compilation methods for error-corrected spin qubit architectures. Future research will focus on developing path-based approaches for dense large-scale architectures that require a reduced number of snake teleportations compared to the rotation algorithms.

Acknowledgements & AI Usage Disclosure

This work received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318) and was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. Furthermore, it was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation, No. 563402549).

During the preparation of this manuscript, AI-based language models were used to improve readability, spelling, grammar, and overall clarity. All outputs generated by these tools were carefully reviewed and edited manually as needed. The authors take full responsibility for the final content.

REFERENCES

[1] P. Kok, W. J. Munro, K. Nemoto *et al.*, "Linear optical quantum computing with photonic qubits," *Reviews of Modern Physics*, 2007.

[2] A. Imamoglu, D. D. Awschalom, G. Burkard *et al.*, "Quantum Information Processing Using Quantum Dot Spins and Cavity QED," *Physical Review Letters*, 1999.

[3] J. I. Cirac and P. Zoller, "Quantum Computations with Cold Trapped Ions," *Physical Review Letters*, 1995.

[4] L. Henriet, L. Beguin, A. Signoles *et al.*, "Quantum computing with neutral atoms," *Quantum*, 2020.

[5] M. Kjaergaard, M. E. Schwartz, J. Braumüller *et al.*, "Superconducting Qubits: Current State of Play," *Annual Review of Condensed Matter Physics*, 2020.

[6] A. Wild, J. Kierig, J. Sailer *et al.*, "Few electron double quantum dot in an isotopically purified ^{28}Si quantum well," *Applied Physics Letters*, 2012.

[7] F. A. Zwanenburg, A. S. Dzurak, A. Morello *et al.*, "Silicon quantum electronics," *Reviews of Modern Physics*, 2013.

[8] Members of the HRL Quantum Team and Collaborators, "A digitally controlled silicon quantum processing unit," 2026, arXiv: 2604.16216 [quant-ph].

[9] M. Veldhorst, J. C. C. Hwang, C. H. Yang *et al.*, "An addressable quantum dot qubit with fault-tolerant control-fidelity," *Nature Nanotechnology*, 2014.

[10] E. Kawakami, T. Jullien, P. Scarlino *et al.*, "Gate fidelity and coherence of an electron spin in an Si/SiGe quantum dot with micromagnet," *Proceedings of the National Academy of Sciences*, 2016.

[11] J. Yoneda, K. Takeda, T. Otsuka *et al.*, "A quantum-dot spin qubit with coherence limited by charge noise and fidelity higher than 99.9%," *Nature Nanotechnology*, 2017.

[12] X. Xue, M. Russ, N. Samkharadze *et al.*, "Quantum logic with spin qubits crossing the surface code threshold," *Nature*, 2022.

[13] N. Paraskevopoulos, F. Sebastiano, C. G. Almudever, and S. Feld, "SpinQ: Compilation Strategies for Scalable Spin-Qubit Architectures," *ACM Transactions on Quantum Computing*, 2023.

[14] N. Paraskevopoulos, C. G. Almudever, and S. Feld, "BeSnake: A Routing Algorithm for Scalable Spin-Qubit Architectures," *IEEE Transactions on Quantum Engineering*, 2024.

[15] P. Escofet, A. Semenov, N. Murphy *et al.*, "Compilation Techniques for Spin Qubits in a Shuttling Bus Architecture," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2025.

[16] P. Escofet, E. Alarcón, S. Abadal *et al.*, "Synthesizing an optimal spin-qubit shuttling-bus architecture for the surface code," *Physical Review A*, 2026.

[17] D. Pataki and A. Pályi, "Compiling the Surface Code to Crossbar Spin Qubit Architectures," *Physical Review B*, 2025.

[18] A. Siegel, Z. Cai, H. Jnane *et al.*, "Quantum Snakes on a Plane: Mobile, Low-Dimensional Logical Qubits on a 2D Surface," *PRX Quantum*, 2026.

[19] C. J. Murphy and J. L. Coffer, "Quantum Dots: A Primer," *Applied Spectroscopy*, 2002.

[20] D. Loss and D. P. DiVincenzo, "Quantum computation with quantum dots," *Physical Review A*, 1998.

[21] R. Hanson, L. P. Kouwenhoven, J. R. Petta *et al.*, "Spins in few-electron quantum dots," *Reviews of Modern Physics*, 2007.

[22] I. Seidler, T. Struck, R. Xue *et al.*, "Conveyor-mode single-electron shuttling in Si/SiGe for a scalable quantum computing architecture," *npj Quantum Information*, 2022.

[23] O. Crawford, J. R. Cruise, N. Mertig, and M. F. Gonzalez-Zalba, "Compilation and scaling strategies for a silicon quantum processor with sparse two-dimensional connectivity," *npj Quantum Information*, 2023.

[24] C.-Y. Huang and W.-K. Mak, "An Efficient Routing Optimization Framework for Silicon-Based Spin-Qubit Devices," in *IEEE International Conference On Computer Aided Design (ICCAD)*, 2025.

[25] D. Culcer, X. Hu, and S. Das Sarma, "Dephasing of Si spin qubits due to charge noise," *Applied Physics Letters*, 2009.

[26] M. M. E. K. Shehata, G. Simion, R. Li *et al.*, "Modeling semiconductor spin qubits and their charge noise environment for quantum gate fidelity estimation," *Physical Review B*, 2023.

[27] D. Gottesman, "Stabilizer Codes and Quantum Error Correction," 1997, arXiv: quant-ph/9705052 [quant-ph].

[28] V. Kolmogorov, "Blossom V: A new implementation of a minimum cost perfect matching algorithm," *Mathematical Programming Computation*, 2009.

[29] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, 2012.

[30] D. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, 2012.

[31] C. Gidney, M. Newman, P. Brooks, and C. Jones, "Yoked surface codes," *Nature Communications*, 2025.

[32] J. Levy, "Universal Quantum Computation with Spin-$1/2$ Pairs and Heisenberg Exchange," *Physical Review Letters*, 2002.

[33] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal Clifford gates and noisy ancillas," *Physical Review A*, 2005.

[34] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit Allocation," in *IEEE International Symposium on Code Generation and Optimization (CGO)*, 2018.

[35] A. Zulehner, A. Paler, and R. Wille, "Efficient Mapping of Quantum Circuits to the IBM QX Architectures," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2018.

[36] G. Li, Y. Ding, and Y. Xie, "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[37] S. Park, D. Kim, M. Kweon *et al.*, "A Fast and Scalable Qubit-Mapping Method for Noisy Intermediate-Scale Quantum Computers," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[38] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[39] C. D. Cantrell, *Modern Mathematical Methods for Physicists and Engineers*. Cambridge University Press, 2000.

[40] R. Wille, L. Berent, T. Forster *et al.*, "The MQT Handbook : A Summary of Design Automation Tools and Software for Quantum Computing," in *IEEE International Conference on Quantum Software (QSW)*, 2024.