Élivágar: Efficient Quantum Circuit Search for Classification

Sashwat Anagolum
Pennsylvania State University

Narges Alavisamani
Georgia Institute of Technology

Poulami Das
University of Texas, Austin

Moinuddin Qureshi
Georgia Institute of Technology

Yunong Shi
AWS Quantum Technologies

Abstract

Designing performant and noise-robust circuits for Quantum Machine Learning (QML) is challenging — the design space scales exponentially with circuit size, and there are few well-supported guiding principles for QML circuit design. Although recent Quantum Circuit Search (QCS) methods attempt to search for performant QML circuits that are also robust to hardware noise, they directly adopt designs from classical Neural Architecture Search (NAS) that are misaligned with the unique constraints of quantum hardware, resulting in high search overheads and severe performance bottlenecks.

We present Élivágar, a novel resource-efficient, noise-guided QCS framework. Élivágar innovates in all three major aspects of QCS — search space, search algorithm and candidate evaluation strategy — to address the design flaws in current classically-inspired QCS methods. Élivágar achieves hardware-efficiency and avoids an expensive circuit-mapping co-search via noise- and device topology-aware candidate generation. By introducing two cheap-to-compute predictors, Clifford noise resilience and representational capacity, Élivágar decouples the evaluation of noise robustness and performance, enabling early rejection of low-fidelity circuits and reducing circuit evaluation costs. Due to its resource-efficiency, Élivágar can further search for data embeddings, significantly improving performance.

Based on a comprehensive evaluation of Élivágar on 12 real quantum devices and 9 QML applications, Élivágar achieves 5.3% higher accuracy and a $271\times$ speedup compared to state-of-the-art QCS methods.

1. Introduction

Quantum Machine Learning (QML) is an important class of quantum algorithms for the Noisy-Intermediate Scale Quantum (NISQ) [65] era, due to its applicability to real-world problems such as classification [20, 28, 42, 80], generative modeling [26, 54, 70, 96, 97] and learning physical systems [27, 41]. QML uses variational quantum circuits to perform machine learning tasks. The parameters in these variational circuits are iteratively updated (i.e., trained) using a classical optimizer until the circuit learns the input problem to high accuracy.

*ss517@psu.edu
We explain these mismatches using QuantumNAS [80], a state-of-the-art QCS framework. At the compilation level, mapping classical designs to quantum circuits via automatic differentiation as intermediate states cannot be expensive due to efficient data movement. However, due to the limited connectivity of NISQ devices, large routing costs are incurred when QML circuits do not match device topology.

Existing QCS methods naively adopt classical ML designs, precluding compatibility with the unique constraints of QML. We explain these mismatches using QuantumNAS [80], a state-of-the-art QCS method. As shown in Fig. 2, QuantumNAS first trains a large device-agnostic SuperCircuit with fixed data embeddings. Next, it performs an evolutionary co-search to identify a performant and noise-robust circuit-qubit mapping pair, using the trained SuperCircuit parameters to estimate the performance of candidate circuits on the target device. Finally, it returns the best circuit-qubit mapping pair found. Other frameworks such as QuantumSupernet [18] follow similar designs, and thus also experience the following issues:

1. They use device-agnostic search spaces, necessitating an additional search for logical-to-physical qubit mappings. The found mappings are also hardware-inefficient, since selected circuits usually do not match the device topology and SWAP gates must be inserted before circuit execution.

2. They do not assess the impact of data embeddings on circuit performance as they use fixed, dataset-agnostic data embeddings. This leads to circuits with poor performance.

3. They adopt strategies from classical ML relying heavily on gradient computation, which is expensive on quantum computers and scales poorly with circuit size.

4. They perform expensive performance evaluation for all circuits. This is inefficient, since many circuits have extremely low fidelity, and will perform poorly since their outputs will be severely corrupted by to device noise. Existing methods do not have any mechanisms to identify and eliminate the performance evaluation for such circuits.

Consequently, existing QCS frameworks incur significant search overheads even for small circuits and problem sizes, and often perform poorly on QML tasks.

To address the problems above, we propose Élivágar, a resource-efficient, high-performance QCS framework tailored to QML systems. First, Élivágar uses a device topology- and noise-aware circuit generation process to generate noise robust circuits that are hardware efficient, eliminating the costly circuit-mapping co-search (issue 1). Second, to mitigate performance bottlenecks caused by fixed, data-agnostic data embeddings (issue 2), Élivágar generates circuits with different data embeddings and variational gates, allowing Élivágar to search for the optimal data embedding for a QML task. Third, to reduce the overheads of training-based circuit evaluations (issue 3), Élivágar introduces a novel, cheap-to-compute performance predictor, representational capacity, for circuit performance evaluation. By using representational capacity, Élivágar eliminates the expensive training step in the QCS workflow while still effectively predicting circuit performance. Last, Élivágar makes use of the insight that evaluating noise robustness is much simpler than evaluating performance. By introducing Clifford noise resilience, a predictor of circuit noise robustness, it decouples these two evaluations to enable early rejection of low-fidelity circuits (issue 4).

Our evaluations using 9 near-term QML benchmarks show that Élivágar finds circuits with 5.3% higher accuracy while being 271× faster than prior state-of-the-art QCS methods. Moreover, we show that the speedup of Élivágar increases with problem size. In summary, our contributions are:

1. We propose a novel QCS workflow, Élivágar, that is tailored to QML systems and addresses the design flaws in current classically-inspired QCS methods.

2. We show that noise-guided, topology-aware and data embedding-aware circuit search, on average, leads to a 5.3% improvement in circuit performance compared to current methods.

3. We show that by using representational capacity to predict circuit performance, the number of circuit executions in Élivágar can be improved by 22×- 523× relative to training-based evaluation strategies.

4. We show that by performing early rejection of low-fidelity circuits using Clifford noise resilience, the resource efficiency of Élivágar can be further improved by 2×-20× depending on noise level.

We open source Élivágar to facilitate future QCS research at https://github.com/SashwatAnagolum/Elivagar.
2. Background and Motivation

2.1. Executing quantum programs on NISQ computers

NISQ-era devices with limited qubit connectivity and imperfect operations make program executions error-prone. Operations on non-adjacent qubits are enabled using SWAPs, as shown in Fig. 3a, which are usually implemented using three 2-qubit gates, making them noisy and expensive. Prior works [15, 35, 48, 67, 78] show that circuit fidelity can be improved through circuit compilation techniques such as qubit mapping and routing, which minimize SWAPs, making it desirable to eliminate SWAPs to make circuits hardware-efficient and boost circuit fidelity.

2.2. Variational circuits for QML

Variational circuits in QML consist of data embedding gates, trainable variational gates, and measurement operations. The data embedding gates are used to transform classical data into quantum states by using these classical data as rotation angles for the data embedding gates. Once the data has been embedded, the variational gates in the circuit are used to manipulate the representations of the data, and measurements are applied by using measurement operators to extract classical data for post-processing, such as for computing parameter updates during training, or for making predictions during inference.

After training, a QML circuit can be used for inference. To do so, we embed into the circuit the data to make predictions based on, and use the learned parameter values for the variational gates. The circuit is executed on a quantum device, and measurements are performed to extract classical data which can be post-processed to obtain predictions. The post-processing can potentially involve statistical analysis, decoding, or other task-specific operations transforming measurement outputs into meaningful results.

Different circuits can be constructed by varying the number, type, and placement of gates used. Due to the large search space, designing performant QML circuits is challenging. As a result, practitioners often rely on a set of commonly used variational templates, such as the hardware efficient ansatz [45]. However, numerous works [9, 12, 28, 45] show that these templates tend to perform poorly on QML tasks as circuit size increases, motivating the search for better circuit structures.

2.2.1. Data embedding gates

The choice of data embedding in a QML circuit significantly impacts its performance, as highlighted in prior studies [10, 62, 72, 73]. To illustrate, consider the example in Fig. 3b, where two circuits share the same variational gates, but differ in data embeddings (RX(x) and RY(x), versus RX(x) alone), resulting in markedly different performance outcomes. However, choosing a suitable data embedding for a given QML task is challenging due to the vast search space, leading many QML circuits to use a fixed embedding such as an angle [7] or Instantaneous Quantum Polynomial-time (IQP) embedding [24], regardless of the nature of the QML task. This often results in suboptimal performance due to the mismatch between the embedding used and the QML task.

2.2.2. Training variational circuits

Variational circuits are trained using gradient-based optimization in a classical-quantum feedback loop, as shown in Fig. 3c. However, training variational circuits is costlier than training classical ML models. In classical ML, gradients are efficiently computed via Automatic Differentiation (AD) [4], which takes constant time. However, this approach is not feasible on quantum computers since intermediate quantum states cannot be manipulated or copied due to the no-cloning theorem [52, 89]. Thus, alternative methods such as parameter-shift rules [71, 87] are used. Parameter-shift methods compute the gradient of a single parameter \( \theta \) by running two circuits...
Table 1: Key differences between Élivágar and existing QCS methods. ES, RL, and MCTS refer to evolutionary search (ES), reinforcement learning (RL), and Monte Carlo tree search (MCTS), respectively.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Search Space</th>
<th>Search Algorithm</th>
<th>Candidate Evaluation Strategy</th>
<th>Runtime Bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCTS-QAOA [91]</td>
<td>Pauli strings</td>
<td>RL + MCTS</td>
<td>Train circuit</td>
<td>Sample-inefficient RL</td>
</tr>
<tr>
<td>QCEAT [30]</td>
<td>Random circuits</td>
<td>ES</td>
<td>Train circuit</td>
<td>Gradient computation</td>
</tr>
<tr>
<td>QuantumSupernet [18]</td>
<td>SuperCircuit</td>
<td>Random search</td>
<td>SuperCircuit loss</td>
<td>Gradient computation</td>
</tr>
<tr>
<td>QuantumNAS [80]</td>
<td>SuperCircuit</td>
<td>ES</td>
<td>SuperCircuit loss</td>
<td>Gradient computation</td>
</tr>
<tr>
<td>Élivágar</td>
<td>Device- and noise-aware circuits</td>
<td>Noise-guided random search (Sec. 4) and RepCap (Sec. 6)</td>
<td>CNR (Sec. 5)</td>
<td>–</td>
</tr>
</tbody>
</table>

with shifted parameters, $\theta + s$ and $\theta - s$ (where $s$ is a constant), and then computing the difference. Unfortunately, this method requires circuit executions scaling linearly with the number of circuit parameters, in contrast to the constant scaling of AD methods; thus, QML gradient computation scales poorly with the number of parameters.

2.3. Limitations of existing QCS works

QCS works that use reinforcement learning (RL), such as MCTS-QAOA [91], converge slowly due to the large action and state spaces involved in quantum circuit design, and incur large circuit evaluation costs during training of the RL model. SuperCircuit-based works such as QuantumSupernet [18] and QuantumNAS [80] also result in impractical runtime overheads due to the high cost of SuperCircuit training (Section 2.2.2). These methods additionally require a circuit-mapping co-search which is often unable to make circuits hardware-efficient, leading to SWAP insertions and reduced circuit noise robustness (Section 2.1). Moreover, SuperCircuit-based works cannot search for optimal data embeddings for a QML task, leading to lowered performance due to data embedding-QML task mismatch (Section 2.2.1). Together, these issues hinder the ability of existing works to find performant circuits while maintaining low search costs.

2.4. The necessity of a new QCS pipeline

It is infeasible to improve significantly on existing QCS works since many of the prerequisites for performant, low-cost QCS are fundamentally incompatible with SuperCircuit-based methods. For example, SuperCircuit training is necessary for accurate circuit performance prediction, preventing the elimination of training and gradient computation. However, gradient computation on quantum hardware is costly, leading to high search overheads. Additionally, SuperCircuit performance prediction is only accurate when using a fixed data embedding for all candidate circuits, precluding the possibility of searching for optimal data embeddings to boost performance.

Therefore, to enable efficient QCS, it is necessary to develop a novel QCS pipeline that accounts for the constraints of QML systems and NISQ-era quantum hardware. To this end, we propose Élivágar, a QCS framework which eschews using a SuperCircuit in favor of training-free predictors of circuit performance, avoiding the training-induced runtime bottlenecks of prior work. Moreover, since Élivágar does not use a SuperCircuit, it can search for optimal data embeddings and perform efficient noise-guided candidate sampling, allowing it to generate hardware-efficient circuits at negligible cost and outperform prior QCS works consistently. Élivágar’s key differences with prior QCS works are presented in Table 1.

3. Overview of Élivágar

As illustrated in Fig. 4, Élivágar consists of five steps:

1. **Topology- and noise-aware candidate circuit generation:** First, Élivágar generates candidate circuits and data embeddings in a device- and noise-aware manner.

2. **Evaluation of noise robustness:** Élivágar then computes the Clifford noise resilience score for circuits to estimate circuit noise robustness.

3. **Early rejection of low-fidelity circuits:** Élivágar ranks candidate circuits based on their Clifford noise resilience and eliminates low-fidelity circuits.

4. **Evaluation of circuit performance:** Élivágar predicts the performance of the remaining circuits on the target QML task using representational capacity.

5. **Final circuit selection:** Finally, Élivágar computes composite scores combining representational capacity and Clifford noise resilience for the remaining candidate circuits, and returns the best circuit.
We break Élivágar down into three parts — candidate circuit generation (step 1, Section 4), noise-guided candidate rejection (step 2-3, Section 5), and circuit performance estimation and circuit selection (step 4-5, Section 6).

4. Candidate circuit generation

To overcome the issues of SuperCircuit-based circuit generation discussed in Section 1, Élivágar directly generates candidate circuits and data embeddings based on target device topology via a noise-guided sampling policy shown in Algorithm 1. This approach has the following advantages over SuperCircuit-based methods: 1) generating circuits based on device topology ensures hardware-efficiency and improves circuit fidelity; 2) circuits are generated along with optimal qubit mappings, eliminating the need for an expensive circuit-mapping co-search; 3) noise-guided sampling further boosts the average fidelity of candidate circuits; 4) generating varied data embeddings for candidate circuits improves selected circuit performance.

As a result, circuits generated by Élivágar have 18.9% higher fidelity than device-unaware circuits optimized using SABRE [35] (Section 9.1). Furthermore, Élivágar obtains 6% higher accuracy on average when searching for data embeddings than when using a fixed data embedding (Section 9.3).

4.1. Noise-guided candidate circuit generation

As shown in Algorithm 1, Élivágar first chooses a connected subgraph from the device topology, and then samples a list of gates to form the circuit. Gate placement choices are guided by subgraph connectivity, noise information such as qubit coherence times, 1- and 2-qubit gate fidelities, readout fidelities, and the current circuit structure. Following prior works in classical NAS [17, 38], Élivágar samples qubits and gate placements from probability distributions instead of directly choosing the best options to encourage candidate diversity.

Choosing a subgraph \( S(V^{(S)}, E^{(S)}) \) for every circuit allows Élivágar to trivially obtain a qubit mapping for every generated circuit by using \( V^{(S)} \). This way, Élivágar avoids the expensive circuit-mapping co-search required by other QCS frameworks. In contrast, the evolutionary co-search used by QuantumNAS [80] cannot guarantee hardware-efficiency, and is very expensive: eliminating it results in Élivágar becoming 1.4×-33.4× faster (Section 9.4). Thus, rather than use an alternative co-search mechanism, Élivágar adopts a novel approach that completely eliminates the need for a co-search, resulting in higher circuit fidelity and search efficiency.

**Insight 1: Generating device-aware circuits**

Generating device- and noise-aware circuits eliminates the need for an expensive circuit-mapping co-search, improves circuit fidelity, and ensures hardware-efficiency.
A key insight of Élivágar is that predicting circuit noise is directly related to circuit fidelity. First, we elaborate on Clifford replicas and CNR, and then use Clifford circuits to identify noise-robust circuits for QML.

5. Noise-guided candidate rejection

A key insight of Élivágar is that predicting circuit noise robustness is much simpler than predicting circuit performance. Leveraging this, Élivágar reduces the number of performance estimations required by identifying and discarding low-fidelity circuits. To do so, Élivágar defines a predictor, Clifford noise resilience (CNR), that accurately predicts circuit fidelity.

While simulating a quantum circuit is exponentially costly in general, Clifford circuits are a class of efficiently simulable quantum circuits [8]. Thus, the fidelity of Clifford circuits can be calculated efficiently by comparing their outputs with noiseless classical simulation results. This enables efficient fidelity estimation for a circuit by using the fidelity of its Clifford replicas as a proxy [32], which is expensive to compute directly for large circuits due to the high simulation cost. While prior works use Clifford replicas to create circuit compilation passes [14, 15, 68] and characterize device noise [67], Élivágar uses Clifford circuits to identify noise-robust circuits for QML.

First, we elaborate on Clifford replicas and CNR, and then show the strong correlation between CNR and circuit fidelity.

4.2. Generating data embeddings

Élivágar employs a simple strategy to generate data embeddings: a few gates in each circuit are used as data embedding gates, with each gate randomly assigned one dimension of the input data to embed (line 14 in Algorithm 1). Despite the random generation of data embeddings, Élivágar leverages representational capacity (Section 6) to predict circuit performance with different data embeddings and select circuits that contain the most suitable data embeddings for the QML task of interest. Consequently, Élivágar optimizes both the data embedding and the variational gates in the selected circuit, resulting in improved performance. By co-searching for data embeddings and variational gates, Élivágar unlocks 6% higher accuracy than when using a fixed data embedding (Section 9.3).

5.1. Creating Clifford replicas

Clifford replicas of a circuit are created by substituting all of the non-Clifford gates in the circuit with Clifford gates. Previous works such as [14, 15] generate a Clifford replica for a circuit by substituting non-Clifford gates with the closest Clifford gate as measured by the diamond norm. In contrast, Élivágar uses multiple Clifford replicas with randomly chosen Clifford gates. The primary reason for this is that prior works deal with circuits at the compilation stage, where the parameter values to be used with each gate are known. In contrast, since Élivágar generates Clifford replicas for circuits before training, it is impossible to know what parameter values will be used with the circuit. Moreover, since gate angles will be changed due to training or different input data, using multiple Clifford replicas, each with randomly chosen Clifford gates, provides a reliable measure of circuit noise robustness over the course of training and prediction.

Figure 5a and 5b show a circuit and one of its Clifford replicas. Since Clifford replicas use the same structure as the original circuit, circuit properties such as depth, are preserved. Élivágar creates multiple Clifford replicas and averages the fidelities of all the constructed replicas. We find that using as few as 16 Clifford replicas can accurately characterize circuit noise robustness.

5.2. Computing CNR

To compute the CNR for a circuit, Élivágar first collects the noisy outputs $P_{\text{noisy}}$ obtained by executing the generated Clifford replicas on the target quantum device. Then, it obtains the ideal Clifford replica outputs $P_{\text{noiseless}}$ via simulation. Élivágar then calculates the Total Variation Distance (TVD) between the noisy and noiseless outputs and computes the fidelity as

$$\text{TVD} = \frac{1}{2} \sum_{i \in |\psi\rangle} |P_{\text{noiseless}}(i) - P_{\text{noisy}}(i)|.$$  

(1)

CNR($C$), the Clifford noise resilience for circuit $C$, is defined as the average fidelity of the $M$ Clifford replicas $C_R(i)$:

$$\text{CNR}(C) = \frac{1}{M} \sum_{i=1}^{M} \text{Fid}(P_{\text{noiseless}}^{C_R(i)}, P_{\text{noisy}}^{C_R(i)}).$$  

(2)

Figure 5: (a) A circuit and (b) a Clifford replica of the circuit. CNR is strongly correlated with circuit fidelity, as demonstrated using circuits run on (c) IBMQ-Kolkata and IBMQ-Guadeloupe, and (d) a noise model of Rigetti Aspen-M-2.
5.3. Candidate circuit rejection

After computing the CNR of candidate circuits, Élivágar ranks circuits by CNR and by default rejects circuits with CNR values less than a threshold of 0.7, or outside the top 50% of all candidates. However, these values can be set by users of Élivágar as hyperparameters. All rejected circuits require no performance evaluation, leading to reduced resource requirements. Depending on device noise levels and threshold values, this rejection step can speed up Élivágar significantly. For example, when searching for a 250-parameter circuit on IBMQ-Manila with a CNR threshold of 0.9, Élivágar can reject 95% of circuits, achieving an almost 20x reduction in circuit executions.

In comparison, SuperCircuit-based QCS works conflate the evaluation of circuit noise robustness and circuit performance, making the identification of low-fidelity circuits unnecessarily expensive. In these methods, the performance of a candidate circuits is evaluated using a validation data set \( \mathcal{X}_{\text{valid}} \) on the target device. This requires executing \( |\mathcal{X}_{\text{valid}}| \) circuits to evaluate every candidate circuit. This cost is often significantly larger than the constant number of circuits required to compute CNR, making identifying low performing circuits needlessly costly in SuperCircuit-based frameworks.

Figure 5c and 5d show the correlation between the CNR and the fidelity of circuits executed on IBMQ-Guadalupe and IBMQ-Kolkata, and a noise model of Rigetti Aspen-M-2. In all cases, CNR is highly predictive of circuit fidelity.

Insight 3: Early rejection of low-fidelity circuits

Noise robustness evaluation is simpler than performance evaluation. Evaluating noise robustness first using Clifford noise resilience enables early rejection of low-fidelity circuits, reducing evaluation costs.

6. Circuit performance evaluation

Our calculations indicate that over 90% of the circuit executions for SuperCircuit-based QCS methods are performed during SuperCircuit training, due to the high cost of computing gradients on quantum hardware via the parameter-shift rule. Thus, eliminating the initial training phase of QCS is crucial for resource efficiency.

Élivágar addresses this by introducing a novel, cheap-to-compute circuit performance predictor, representational capacity (RepCap). RepCap requires far fewer circuit executions than training-based evaluation strategies, but is just as effective at predicting circuit performance. Fig. 6 shows the correlations of SuperCircuit predicted losses and RepCap with trained circuit performance on the FMNIST-2 task. Despite not requiring any training, RepCap is as strong a predictor of performance as a trained SuperCircuit. RepCap is a strong predictor of circuit performance on other QML benchmarks as well, as shown in Fig. 7, and has a Spearman R correlation of 0.632 with circuit performance over all the QML benchmarks used for evaluation.

First, we define RepCap and provide some intuition about it, and then elaborate on how to compute RepCap. Then, we discuss how we combine CNR and RepCap to rank the circuits remaining after the noise-guided circuit elimination process.

6.1. Predicting circuit performance using RepCap

Intuitively, RepCap measures the intra-class similarities and inter-class separation of the final quantum states for a variational circuit. As illustrated in Fig. 6a, the larger the inter-cluster distance and the smaller the intra-cluster distance in the final qubit state space, the higher the RepCap of a variational circuit is. A higher RepCap indicates that the circuit has a larger “capacity” for representing the input data, and likely will perform better.

RepCap for circuit \( C \) is defined as:

\[
\text{RepCap}(C) = 1 - \frac{\| R_C - R_{\text{ref}} \|_2^2}{2 \cdot n_d \cdot d^2},
\]

where \( d_c \) and \( n_c \) are the number of samples selected from each class and the number of classes in \( \mathcal{X}_{\text{train}} \), respectively. \( \| \cdot \|_2 \) is the Frobenius norm. \( R_{\text{ref}} \) is a \( d \times d \) reference matrix that captures the ideal circuit behaviour, i.e., \( R_{\text{ref}}(i,j) = 1 \) iff \( y_i = y_j \), and \( R_{\text{ref}}(i,j) = 0 \) otherwise. \( R_C \) is a \( d \times d \) matrix that captures pairwise similarities between the representations of data points \( x_i \) and \( x_j \) created by circuit \( C \), with \( R_C(i,j) \in [0,1] \).

The entries of \( R_C \) used in Eq. (3) are the induced similarity \( IS_C \) between data points \( x_i \) and \( x_j \), i.e.,

\[
R_C(i,j) = IS_C(x_i,x_j).
\]

\( IS_C(x_i,x_j) \) is defined as the averaged similarity of the output quantum states \( \rho_C(x_i,\theta) \) and \( \rho_C(x_j,\theta) \) of circuit \( C \) when fed input data \( x_i \) and \( x_j \) over random parameters \( \theta \). In practice, we use classical approximations \( \hat{\rho}_C(x_i,\theta) \), computed via Algorithm 2, instead of states \( \rho_C(x_i,\theta) \) in order to avoid executing circuits for every pair of samples \( x_i \) and \( x_j \):

\[
IS_C(x_i,x_j) = \frac{1}{n_\theta} \sum_{\theta=\theta_1}^{\theta_n} \text{Sim}(\hat{\rho}_C(x_i,\theta),\hat{\rho}_C(x_j,\theta)).
\]

The angles \( \{ \theta \} \) can be uniformly sampled. Then the similarity \( \text{Sim}(x_i,x_j) \) between \( x_i \) and \( x_j \) can be defined as the one minus Total Variational Distance (TVD) between their output states by \( C \) using a randomized measurement protocol [19, 23, 29].

\[
\text{Sim}(\hat{\rho}_C(x_i,\theta),\hat{\rho}_C(x_j,\theta)) = \frac{1}{n_\text{bases}} \sum_{k=1}^{n_\text{bases}} 1 - \text{TVD}(\hat{\rho}_C(x_i,\theta)_k,\hat{\rho}_C(x_j,\theta)_k).
\]
In practice, we take very poorly with problem size due to the high cost of gradient circuits. The initial training overhead of a SuperCircuit scales each on average, and then evaluate the performance of \( N \) on various QML tasks.

Figure 7: Representational capacity is a strong predictor of performance for various QML tasks.

Algorithm 2: Constructing the classical approximation \( \hat{\rho}_C(x, \theta) \) of a representation \( \rho_C(x, \theta) \)

**Input:** Circuit \( C \): Input data \( x \); variational parameters \( \theta \); array of random rotation angles \( \alpha \) of size \( n_{\text{bases}} \times n_{\text{meas}} \times 3 \).

**Output:** Classical approximation \( \hat{\rho}_C(x, \theta) \) of \( \rho_C(x, \theta) \)

1. Append a set of \( n_{\text{meas}} \) U3 gates to \( C \) acting on the \( n_{\text{meas}} \) qubits being measured;
2. Initialize \( \hat{\rho}_C(x, \theta) \) as an empty list of length \( n_{\text{bases}} \);
3. for \( i = 1 \) to \( n_{\text{bases}} \) do
4. Execute circuit \( C(x, [\theta, \alpha_i]) \);
5. Construct probability distribution \( P_{(\alpha, i)}(x, \theta) \) using the outputs of \( C(x, [\theta, \alpha_i]) \);
6. Set \( \hat{\rho}_C(x, \theta) \leftarrow P \);
7. return \( \hat{\rho}_C(x, \theta) \);

Computing RepCap requires only \( n_c \cdot d_c \cdot n_p \) circuit executions, where \( n_p \) is the number of parameter initializations we average over. In practice, we take \( d_c = 16 \), \( n_p = 32 \). For \( N_C \) circuits, Ólivágar then requires \( 512 \cdot N_C \cdot n_c \) circuit executions. In contrast, SuperCircuit-based methods require \( 2 \cdot r \cdot |\mathcal{X}_{\text{train}}| \cdot \bar{p} + N \cdot |\mathcal{X}_{\text{valid}}| \) circuit executions to train a SuperCircuit for \( r \) epochs via sampling subcircuits with \( \bar{p} \) parameters each on average, and then evaluate the performance of \( N \) subcircuits. The initial training overhead of a SuperCircuit scales very poorly with problem size due to the high cost of gradient computation, increasing the speedup of Ólivágar with problem size. For example, Ólivágar is only 44 \( \times \) faster than QuantumNAS on the 16-parameter Moons task (Section 7); for the 72-parameter MNIST-10 task, the speedup is over 5000 \( \times \).

**Insight 4:** Predict circuit performance without training

By analyzing the output states of a circuit for different input data, we can cheaply estimate circuit performance as well as prior work.

### 6.2. Final evaluation of remaining circuits

Ólivágar combines \( \text{CNR}(C) \) and \( \text{RepCap}(C) \) to accurately predict performance of the remaining circuits on the target device. The composite score \( \text{Score}(C) \) for circuit \( C \) is given by

\[
\text{Score}(C) = \text{CNR}(C)^{\alpha_{\text{CNR}}} \times \text{RepCap}(C). \tag{7}
\]

\( \alpha_{\text{CNR}} \) is a hyperparameter controlling the relative importance of \( \text{CNR}(C) \), with \( \text{RepCap}(C) \) and \( \text{CNR}(C) \) being equally important at \( \alpha_{\text{CNR}} = 1 \). For all experiments, we set \( \alpha_{\text{CNR}} = 0.5 \). Finally, Ólivágar returns the circuit with the highest \( \text{Score}(C) \).

### 7. Experimental Setup

#### 7.1. Benchmarks

We conduct experiments on 9 different QML benchmarks. A summary of the benchmarks is shown in Table 2. Train and Test in Table 2 refer to the number of samples used for training and testing, respectively. Params is the number of parameters in the circuits we search for in Section 8. Moons is a synthetic dataset generated using scikit-learn. We select a balanced subset of the original Bank dataset [43]. We use classes \{0, 1\} from the MNIST dataset for MNIST-2, \{0, 1, 4, 8\} for MNIST-4, \{T-shirt, Trousers\} from the FMNIST dataset for FMNIST-2, and \{T-shirt, Trousers, Bag, Ankle Boot\} for FMNIST-4. For the MNIST and FMNIST benchmarks, we center crop the original images to 24x24 and downsample them to 4x4 using mean pooling, except for MNIST-10, which is downsampled to 6x6.
Table 2: Summary of the 9 QML benchmarks used for evaluations.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Classes</th>
<th>Data Dim.</th>
<th>Train</th>
<th>Test</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moons</td>
<td>2</td>
<td>2</td>
<td>0.6K</td>
<td>0.12K</td>
<td>16</td>
</tr>
<tr>
<td>Bank</td>
<td>2</td>
<td>4</td>
<td>1.1K</td>
<td>0.12K</td>
<td>20</td>
</tr>
<tr>
<td>MNIST-2</td>
<td>2</td>
<td>4x4</td>
<td>1.6K</td>
<td>0.4K</td>
<td>20</td>
</tr>
<tr>
<td>MNIST-4</td>
<td>4</td>
<td>4x4</td>
<td>8K</td>
<td>2K</td>
<td>40</td>
</tr>
<tr>
<td>FMNIST-2</td>
<td>2</td>
<td>4x4</td>
<td>1.6K</td>
<td>0.2K</td>
<td>32</td>
</tr>
<tr>
<td>FMNIST-4</td>
<td>4</td>
<td>4x4</td>
<td>8K</td>
<td>2K</td>
<td>24</td>
</tr>
<tr>
<td>Vowel-2</td>
<td>2</td>
<td>10</td>
<td>0.6K</td>
<td>0.12K</td>
<td>32</td>
</tr>
<tr>
<td>Vowel-4</td>
<td>4</td>
<td>10</td>
<td>0.6K</td>
<td>0.12K</td>
<td>40</td>
</tr>
<tr>
<td>MNIST-10</td>
<td>10</td>
<td>6x6</td>
<td>60K</td>
<td>10K</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 3: Summary of the real quantum hardware used.

<table>
<thead>
<tr>
<th>Device</th>
<th>Qubits</th>
<th>QV</th>
<th>Readout</th>
<th>1Q</th>
<th>2Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>OQC Lucy</td>
<td>8</td>
<td>–</td>
<td>1.3e-1</td>
<td>6.2e-4</td>
<td>4.4e-2</td>
</tr>
<tr>
<td>Rigetti Aspen M-3</td>
<td>79</td>
<td>–</td>
<td>8.0e-2</td>
<td>1.5e-3</td>
<td>9.3e-2</td>
</tr>
<tr>
<td>IBMQ Jakarta</td>
<td>7</td>
<td>32</td>
<td>2.6e-2</td>
<td>2.2e-4</td>
<td>8.5e-3</td>
</tr>
<tr>
<td>IBM Nairobi</td>
<td>7</td>
<td>32</td>
<td>2.4e-2</td>
<td>2.7e-4</td>
<td>9.6e-3</td>
</tr>
<tr>
<td>IBM Lagos</td>
<td>7</td>
<td>32</td>
<td>1.9e-2</td>
<td>2.1e-4</td>
<td>9.8e-3</td>
</tr>
<tr>
<td>IBM Perth</td>
<td>7</td>
<td>32</td>
<td>2.8e-2</td>
<td>2.8e-4</td>
<td>8.7e-3</td>
</tr>
<tr>
<td>IBM Geneva</td>
<td>16</td>
<td>32</td>
<td>2.7e-2</td>
<td>2.2e-4</td>
<td>1.1e-2</td>
</tr>
<tr>
<td>IBM Guadalupe</td>
<td>16</td>
<td>32</td>
<td>2.0e-2</td>
<td>2.9e-4</td>
<td>8.9e-3</td>
</tr>
<tr>
<td>IBMK Kolkata</td>
<td>27</td>
<td>128</td>
<td>1.2e-2</td>
<td>2.3e-4</td>
<td>9.0e-3</td>
</tr>
<tr>
<td>IBM Mumbai</td>
<td>27</td>
<td>128</td>
<td>1.9e-2</td>
<td>2.0e-4</td>
<td>9.6e-3</td>
</tr>
<tr>
<td>IBM Kyoto</td>
<td>127</td>
<td>–</td>
<td>1.4e-2</td>
<td>2.5e-4</td>
<td>9.1e-3</td>
</tr>
<tr>
<td>IBM Osaka</td>
<td>127</td>
<td>–</td>
<td>1.7e-2</td>
<td>2.2e-4</td>
<td>1.0e-2</td>
</tr>
</tbody>
</table>

The Vowel-2 and Vowel-4 datasets are constructed by merging 8 out of the 9 classes in the original Vowel dataset, and selecting the 10 most significant PCA dimensions.

7.2. Backends and compiler configurations

The hardware devices used are listed in Table 3. We also use multiple noisy simulators use noise models based on IBM Nairobi, IBM Lagos, IBM Perth, IBMQ Jakarta, IBM Guadalupe, Rigetti Aspen-M-2, Rigetti Aspen-M-3, and OQC Lucy in order to perform evaluations, compute circuit fidelity and CNR. To compute RepCap and perform circuit training, we use noiseless simulators from Pennylane [6] and TorchQuantum [80]. Before running circuits on any hardware device or noisy simulator, we compile the circuit using the default Qiskit compiler with optimization level set to 3 for all competing methods except for QuantumNAS, for which we use level 2, and Elivagar, for which we use level 0.

7.3. Training methodology

All circuits are trained using the same methodology to ensure fair comparisons. We train circuits for 200 epochs using a batch size of 128 and optimize parameters using the Adam optimizer with learning rate 0.01. No weight decay or learning rate scheduling is used. The training objective is to minimize the classification loss between circuit predictions and the labels of the training set. We perform training on noiseless simulators using the TorchQuantum package [80], and use an AWS ml.g4dn.xlarge instance with 16GB of memory and an NVIDIA T4 GPU.

7.4. Competing methods

We compare Elivagar to four competing methods. Every circuit compared to, including those chosen by Elivagar, all contain the same number of parameters in order to ensure a fair comparison. The number of parameters used in the circuits for each benchmark is given in Table 2.

Random: we randomly generate 25 circuits using the RXYZ + CZ gateset from [80], and report the average performance of the circuits.

Human designed (baseline): We use three data embedding schemes (angle, amplitude, and IQP embedding) paired with the commonly used BasicEntanglerLayers template from the Pennylane [6] library. We train all three circuits and report the average performance.

QuantumSupernet [18]: We modify the SuperCircuit training procedure proposed in [18] and use mini-batch gradient descent with batch size 32 instead of full-batch gradient descent to ensure all SuperCircuit parameters are updated sufficiently. All other hyperparameters are taken from [18].

QuantumNAS [80]: We use the same hyperparameters for SuperCircuit training and evolutionary search as [80], and use the RXYZ + CZ gateset since it performs the best out of the 6 gatesets in [80] before pruning. We do not perform iterative pruning for trained QuantumNAS circuits as the pruning phase does not contribute towards circuit search.

7.5. Hyperparameters for Elivagar

We randomly sample $d_c = 16$ data points from each class of $X_{\text{train}}$ and $n_p = 32$ parameter initializations to compute RepCap. We use $M = 32$ Clifford replicas to compute CNR, and reject circuits with CNR scores less than 0.7 or outside the top 50%. We construct composite scores using $\alpha_{\text{CNR}} = 0.5$. To mitigate the effects of random circuit sampling, we repeat our workflow 25 times, and report the average performance.

7.6. Figure of merit

We measure the performance of a circuit using the classification accuracy it obtains over a dedicated test set for each benchmark. We highlight the difference between circuit accuracy and circuit fidelity in this work: the classification accuracy of a circuit is the fraction of predictions made by the circuit that are correct, i.e. match the ground truth label of the associated sample for which the prediction was made; while fidelity measures the degree to which the outputs of a quantum circuit remain unaffected by hardware noise.
8. Results

8.1. Performance on QML benchmarks

Fig. 8a shows performance on 9 different QML benchmarks, using noisy simulators with noise models of Rigetti Aspen-M-3, OQC Lucy, IBM Lagos, IBM Perth, IBM Nairobi, IBMQ Jakarta. Fig. 8b shows the results of evaluations performed on real quantum hardware. Élivágar is competitive with or outperforms QuantumNAS on all benchmarks except Vowel-4. Circuit performance on the Rigetti Aspen-M-3 and OQC Lucy devices is worse than on the IBM devices due to their higher noise levels (see Table 3). On average, Élivágar achieves 5.3% higher accuracy than QuantumNAS, and 22.6% higher accuracy than the human designed baseline.

8.2. Runtime speedups

We consider two hardware setups when measuring the speedup of Élivágar over QuantumNAS, which we elaborate on below.

8.2.1. Using classical simulators

Using classical simulators results in reduced training costs as gradients can be computed efficiently using backpropagation. Additionally, training and inference on noiseless simulators can be performed in a batched manner, further speeding up the QCS process. We provide absolute runtime values for both QuantumNAS and Élivágar in Table 4. Even when using classical simulators and backpropagation, which disproportionately benefits training-heavy frameworks such as QuantumNAS, Élivágar is $11.7 \times$ faster than QuantumNAS on average. Moreover, the speedup achieved by Élivágar increases with benchmark size despite the reduced cost of gradient computation, highlighting Élivágar’s resource-efficiency.

8.2.2. Using quantum hardware

In this scenario, all training is done via the parameter-shift rule [87], which results in drastically increased training costs, as explained in Section 2. Unfortunately, the high variance in the times required to run circuits on quantum hardware via a cloud computing platform make it near impossible to reliably estimate runtimes via wall-clock time. Consequently, the most reliable way to estimate the speedup achieved by Élivágar is to compare the number of circuit executions required by both methods for each benchmark. Table 4 shows the speedups achieved by Élivágar, which is $271 \times$ times faster than QuantumNAS on average. We provide a detailed breakdown of this speedup in Section 9.4.
9. Performance Analysis

9.1. Device-aware circuit generation

To analyze Élivágar’s circuit generation strategy, we compare device-aware circuits generated via Algorithm 1 with randomly generated, device-unaware circuits. We generate pairs of device-aware and -unaware circuits with the same number of 1- and 2-qubit gates before compilation to ensure a fair comparison. Device-aware circuits are run unoptimized, but device-unaware circuits are optimized using level 3 of the Qiskit compiler and SABRE [35]. Table 5 shows that device-aware circuits have 18.9% higher fidelity on average. Thus, Élivágar’s circuit generation strategy not only eliminates the need for a circuit-mapping co-search, but also boosts circuit noise robustness, reducing performance degradation.

9.2. Circuit statistics

The statistics for the compiled and optimized circuits used for the Bank and Vowel-2 benchmarks are shown in Table 6. Random and Human Design are noise and device unaware, resulting in large, deep circuits (depths 163, 260 for MNIST-4, and 211, 344 for MNIST-10) with many two-qubit gates even after optimization using level 3 of the Qiskit compiler. As a result, these circuits experience a large reduction in accuracy when run on real devices compared to their noise-free performance. QuantumSupernet is noise and device-aware, but uses a deep embedding subcircuit that requires multiple layers of entangling CRY gates, and thus undergoes significant accuracy reduction as well. The circuits found by QuantumNAS and Élivágar are much shallower due to the evolutionary search and CNR incentivizing the selection of shallow, noise-robust circuits, respectively.

However, despite being similarly noise-robust and shallow, the circuits found by Élivágar perform significantly better than those found by QuantumNAS, likely due to the difference in the embeddings used by the circuits. We explore the effect of searching for data embeddings in detail in Section 9.3.
9.3. Breakdown of performance improvement

We break down Élivágar’s performance improvement into three parts: device-aware circuit generation (Section 4), RepCap and data embeddings (Section 6), and CNR (Section 5) in Fig. 9. We use 8 benchmarks from Table 2, and compare Élivágar to three baselines: (1) device- and noise-unaware circuits, (2) hardware-efficient device- and noise-aware random circuits generated via Algorithm 1, and (3) circuits found by Élivágar using only RepCap, i.e. without using CNR to rank circuits.

Using device- and noise-aware circuits increases accuracy over device-unaware circuits by 5%, further demonstrating the improved noise-robustness of circuits generated using Algorithm 1. The biggest accuracy improvement (6%) is obtained by using RepCap to select circuits instead of choosing randomly, highlighting the importance of strong predictors of circuit performance.

Using RepCap to select circuits allows Élivágar to search for both data embeddings and variational gates. To isolate the effect of searching for data embeddings, we compare Élivágar with versions of Élivágar that use fixed angle [7] and IQP embeddings [24]. We evaluate the three versions of Œlivágar using a noiseless simulator to eliminate the effects of noise. As shown in Fig. 10, Élivágar obtains 5.5% and 20% higher accuracy when searching for data embeddings than when using a fixed angle and IQP embedding, respectively. This is because searching for embeddings allows Élivágar to co-search for suitable combinations of embeddings and variational gates, allowing it to find higher performance circuits than when using a fixed embedding. Thus, almost all of the accuracy gains achieved by using RepCap is due to the data embedding search, reinforcing the significance of data embeddings in determining performance on QML tasks.

Finally, using CNR in addition to RepCap further increases accuracy by 2% on average. Using both predictors results in better performance than only using RepCap as while using RepCap leads to Élivágar choosing circuits with high noiseless accuracy, RepCap is not device- and noise-aware and thus may choose circuits that are not noise-robust, leading to large accuracy degradation when run on real hardware. Using both predictors allows Élivágar to balance circuit learning capability and expressivity with circuit noise robustness to find a circuit that can accurately learn the target dataset while also being robust to hardware noise.

9.4. Breakdown of runtime speedup

When training on quantum hardware, the speedup of Élivágar over QuantumNAS comes from (1) using representational capacity for performance evaluation, (2) early rejection of low-fidelity circuits, and (3) eliminating the circuit-mapping co-search by generating device-aware circuits. Using representational capacity speeds up Élivágar by $16 \times - 78 \times$ for the benchmarks in Section 8, growing with problem size. Since we use conservative CNR hyperparameter values, the speedup due to early rejection for all tasks is $2 \times$. Eliminating the circuit-mapping co-search provides a speedup of $1.4 \times - 33.4 \times$, growing with problem size.

9.5. Compatibility with other QML frameworks

Élivágar is compatible with frameworks targeting the data preprocessing and training stages of the QML pipeline, such as QTN-VQC [66] and QuantumNAT [82], and makes no assumptions about how these tasks are performed. These frameworks can be combined with Élivágar to further boost circuit performance. Fig. 11a compares the performance of Élivágar and QuantumNAS with and without QuantumNAT [82], a framework that aims to increase the noise robustness of circuits during training and inference. Élivágar obtains 2.2% higher accuracy than QuantumNAS + QuantumNAT, and 5.5% higher accuracy when paired with QuantumNAT.

![Figure 10: Performance of Élivágar when searching for data embeddings versus using a fixed data embedding. All values are absolute classification accuracies obtained on noiseless simulators. Each bar shows the mean of 25 runs.](image)

![Figure 11: Results on real quantum hardware when performing training and inference with and without (a) QuantumNAT [82] and (b) QTN-VQC [66]. All values are absolute classification accuracies.](image)
We further combine Élivágar and QuantumNAS with QTN-VQC, a framework that performs classical preprocessing of input data via a trainable tensor network, with results shown in Fig. 11b. Even when using QTN-VQC, which significantly boosts performance due to the added classical trainable parameters in the tensor network used for preprocessing, Élivágar outperforms QuantumNAS by 2.4% on average.

10. Related works

10.1. Quantum Machine Learning

Several theoretical works have illustrated potential quantum advantage in tasks such as classification [1, 28] and regression [62]. Various approaches to QML have been proposed, including using quantum circuits as kernel methods [9, 21, 23, 24, 28, 33, 41, 72, 85], and as quantum neural networks [5, 20, 34, 51, 57, 66, 75, 76, 81, 97]. Multiple works [1, 76] introduced metrics that estimate circuit performance, but they are unsuitable for QCS due to their high cost.

10.2. Noise-aware quantum compilation and training

Recent compilation works explore different circuit transformations to improve noise robustness, including gate cancellation [31, 49, 90], qubit mapping and routing [3, 39, 40, 44, 47, 50, 55, 56, 77, 79, 88], error mitigation [13, 16, 58, 59], circuit synthesis [60, 61, 63, 86, 93], and pulse-level optimizations [22, 37, 46, 74]. Some of these works [13, 14, 67] use Clifford replicas to characterize device noise or create enhanced compilation passes. Élivágar, in contrast, applies Clifford replicas to QML, and uses them to perform a noise-guided search for QML circuits. Other works [82, 83] focus on increasing noise robustness for QML circuits through improved training procedures, an approach complementary to Élivágar, as these techniques can be combined with Élivágar to improve the training of circuits selected by Élivágar.

10.3. QCS for Variational Quantum Algorithms (VQAs)

A few studies have explored QCS for VQAs [11], such as QAQA [91], VQE [30], and state preparation [25, 92, 94]. These works largely adopt techniques from classical NAS. For example, QCEAT [30] is similar to evolutionary search-based NAS [69], and MCTS-QAOA [91] uses a reinforcement learning-based search algorithm inspired by [2, 95]. These frameworks can be extended to perform QCS for QML tasks, despite being originally designed for VQAs. However, since they rely on classically-inspired search and candidate evaluation mechanisms, these methods face the same issues as QCS works for QML, and are extremely slow even for small QML tasks. We compare Élivágar with [30, 91] in Table 1.

10.4. Data embeddings

Several works [10, 62, 73] have theoretically investigated the effect of data embeddings on circuit performance through the lens of generalization bounds and Fourier series. These works focus on developing theory that relates the data embeddings used to trained circuit performance. As such, they use simplified settings that do not account for hardware noise or limited device connectivity, and place strict constraints of the structures of the circuits used. Thus, they cannot be used as tools for QCS directly. In contrast, Élivágar solves the near-term problem of identifying performant circuits for QML applications on NISQ hardware, which is a fundamentally different task. Élivágar, however, applies these theoretical insights on data embeddings to QCS by searching for both optimal data embeddings and variational gates for a QML task, which allows Élivágar to outperform prior QCS works.

10.5. QCS for QML

A few studies have focused on QCS for QML tasks, such as QuantumNAS [80] and QuantumSupernet [18]. These methods are both based on the classical Supernet [64] framework, and have been extensively discussed in Section 1, and compared with Élivágar in Section 8. Élivágar eschews the SuperCircuit that [18, 80] use in favor of a novel QCS pipeline that leverages training- and gradient-free circuit performance and noise robustness predictors to avoid expensive gradient computation. Thus, Élivágar avoids the runtime bottlenecks of SuperCircuit-based QCS frameworks, and is able to perform QCS with low overheads. In Table 1, we present a summary of the differences between Élivágar and prior works.

11. Conclusion

In this work, we present Élivágar, a noise-guided, resource-efficient QCS framework that searches for high-performance variational quantum circuits for QML tasks. Élivágar proposes that QCS methods should be tailored to QML tasks and carefully consider realistic constraints of QML algorithms and NISQ quantum hardware. Based on this proposal, Élivágar addresses the issues in current classically-inspired QCS methods and innovates in all important aspects of QCS with its topology-aware and data embedding-aware search space, noise-guided search algorithm, and two-stage circuit evaluation strategy. Comprehensive experimental results show that Élivágar significantly outperforms leading QCS methods while achieving much more favorable resource efficiency and scalability. Due to its resource efficiency and improved search space, Élivágar provides a powerful tool for enabling new research in QML and noise-aware quantum compilation.

Acknowledgements

S.A. thanks Sidharth Ramanan for fruitful discussions and comments on various drafts. Y.S. thanks Gushu Li and Eric Kessler for helpful reviews on the paper draft. We thank Prof. Devesh Tiwari for shepherding the paper and for providing detailed feedback. This work was funded in part by EPiQC, an NSF Expedition in Computing, under grant CCF-173044.
A. Artifact Appendix

A.1. Abstract

Élivágar introduces a novel pipeline for performing Quantum Circuit Search (QCS), which involves systematically searching for performant and noise-robust circuits to deploy on quantum hardware for machine learning applications.

The associated artifacts can be used to validate the main results presented in our paper, specifically the performance of Élivágar on various QML datasets with different target quantum hardware backends. We provide scripts to reproduce each step of Élivágar’s pipeline (see Fig. 4), from device-aware circuit generation to the training of the candidate circuits selected by Élivágar. We additionally provide scripts to generate results for the random and human-designed baselines, as well as to integrate Élivágar with other works aiming to boost QML circuit performance, such as QuantumNAT [82] and QTN-VQC [66]. Software requirements and dependencies include a Python installation, and various Python packages such as TorchQuantum, Qiskit, and PyTorch: a full list of requirements can be found at https://github.com/SashwatAnagolum/Elivagar.

A.2. Artifact check-list (meta-information)

• **Algorithm:** Training-free quantum circuit search for QML classification tasks containing the following steps:
  – Early rejection of low-fidelity candidate circuits.
  – Composite score computation and circuit selection.

• **Program:** We use the following benchmarks to evaluate Élivágar:
  – Moons: available through the scikit-learn Python package.

• **Model:** Randomly generated, human-designed, and automatically searched parameterized quantum circuits used for machine learning tasks.

• **Data set:**
  – MNIST: approximately 15MB.
  – FMNIST: approximately 15MB.
  – Bank: approximately 50KB.
  – Moons: approximately 50KB.
  – Vowel: approximately 100KB.

All of the listed datasets are provided as part of the artifact, and no downloads are required to perform evaluations.

• **Run-time environment:**
  – The main software dependencies include:
    * Python
    * TorchQuantum
    * Qiskit

• **Hardware:** Performing evaluations on real quantum devices requires access to IBM quantum hardware, which is publicly available at https://quantum.ibm.com/, and for which we provide a token as part of the artifact.

• **Run-time state:** Due to drift in quantum hardware, the performance of Élivágar on the same benchmark and device may vary over time.

• **Execution:** It is important that users perform all steps of Élivágar’s pipeline within a relatively short timespan to prevent device drift from occurring in between the candidate generation, fidelity prediction, and circuit training stages, which will result in lowered performance. The runtime required for an experiment depends on the circuit size, as well as how many candidates are evaluated. Since we evaluate Élivágar on each benchmark-device combination 25 times to mitigate any randomness, the estimated total time required for completing experiments using the default hyperparameters for Élivágar (Section 7.5) on any of the 4-qubit benchmarks is 1 day, dominated by the classical noisy simulation of circuits via Qiskit.

• **Metrics:** Classification accuracy on the test portion of different benchmarks.

• **Output:** The classification accuracy of the circuits being trained and evaluated will both printed to the console and saved to disk for later retrieval and comparison to results presented in Fig. 8.

• **Experiments:** Shell scripts are provided to perform each step of Élivágar’s pipeline, and detailed instructions are presented in https://github.com/SashwatAnagolum/Elivagar/blob/main/README.md. The results obtained are expected to be similar to those in Fig. 8, with a maximum allowed variation in classification accuracy of 10% due to randomness in the circuit generation and selection process, as well as drift in the quantum hardware being used.

• How much disk space required (approximately)?: < 10 GB.

• How much time is needed to prepare workflow (approximately)?: < 1 hour.

• How much time is needed to complete experiments (approximately)?: 72 hours.

• Publicly available?: Yes, at https://github.com/SashwatAnagolum/Elivagar

• Code licenses (if publicly available)?: MIT License.

• Data licenses (if publicly available)?:
  – MNIST: Creative Commons Attribution-Share Alike 3.0.
  – FMNIST: MIT License.
  – Bank: Creative Commons Public available.
  – Moons: BSD 3-Clause License.

• **Workflow framework used?:** The artifacts depends on the following frameworks:
  – PyTorch
  – TorchQuantum

• **Archived (provide DOI)?:** The artifact is currently available at https://github.com/SashwatAnagolum/Elivagar - we will provide a DOI after evaluation.
A.3. Description

A.3.1. How to access

Users can access the artifact by visiting https://github.com/SashwatAnagolum/Elivagar and cloning or downloading the repository.

A.3.2. Hardware dependencies

Remote access to IBM quantum devices is required to perform evaluation of Élivágar on real quantum hardware. We provide an access token along with the artifact that enables access to public IBM devices; a separate token will be required to use devices which are accessible via reservation only, but is not required to replicate the main results of the paper. Performing circuit training on GPUs will speedup training, although the total experiment runtime is dominated by real circuit execution on the cloud and noisy simulation of circuits.

A.3.3. Software dependencies

Élivágar is implemented in Python, and has several dependencies, including the PyTorch, TorchQuantum, and Qiskit Python packages. For a full list of dependencies, see the requirements.txt file.

A.3.4. Data sets

The datasets used include MNIST, FMNIST, Vowel, Bank, and Moons. Details on the preprocessing performed for each dataset can be found in Table 2, and all of the datasets are included as part of the artifact as downloaded from https://github.com/SashwatAnagolum/Elivagar - no further downloads are required.

A.3.5. Models

The artifact enables generation of various QML models via running Élivágar, random generation, and through implementation of human-designed architectures.

A.4. Installation

In order to install the artifact, please download the GitHub repository on your local machine. Once the repository has been downloaded, following the installation instructions presented in the README.md file will allow users to perform all the setup necessary to perform evaluations.

A.5. Experiment workflow

Once setup has been completed as outlined in https://github.com/SashwatAnagolum/Elivagar/blob/main/README.md, executing the following scripts from the root directory will allow users to perform QCS on the Moons dataset targeting the IBM Osaka hardware backend:

- Generate candidate circuits: ./scripts/moons/elivagar/circ_gen_osaka.sh
- Compute CNR for candidate circuits: ./scripts/moons/elivagar/cnr_osaka.sh
- Compute RepCap for candidate circuits: ./scripts/moons/elivagar/repcap_osaka.sh
- Compute composite scores and train selected circuits: ./scripts/moons/elivagar/train_osaka.sh
- Evaluate trained circuits: ./scripts/moons/elivagar/eval_osaka.sh

The following scripts will generate, train, and evaluate random circuits for the same benchmark-device combination:

- Generate candidate circuits: ./scripts/moons/random/circ_gen_osaka.sh
- Train generated circuits: ./scripts/moons/random/train_osaka.sh
- Evaluate trained circuits: ./scripts/moons/random/eval_osaka.sh

For detailed instructions and for steps on how to run evaluations for other benchmark-device combinations, please see the README.

A.6. Evaluation and expected results

The functionality of Élivágar can be verified by performing the steps above and observing the creation and training of various quantum circuits. Over the course of training, the circuit loss should decrease, and circuit classification accuracy should increase. To verify the performance of Élivágar, the accuracy of circuits found by Élivágar can be obtained by performing inference using various benchmarks on noiseless and noisy simulators, as well as real quantum hardware. Élivágar is expected to achieve higher accuracy than the random and human-designed baselines, corroborating the results presented in Fig. 8, and demonstrating the efficacy of Élivágar’s device- and noise-aware circuit search pipeline. The maximum allowed variance in results is 10% due to device drift, as well as due to randomness in the circuit sampling and selection processes.

A.7. Experiment customization

The artifacts provided can also be used to perform QCS for arbitrary classification datasets and currently supports QCS for any device accessible through IBM Quantum. Élivágar also provides scripts to easily train circuits of any architecture, and enables generation of a wide range of circuits, such as random, human-designed, and more.

A.8. Notes

Multiple devices that were used to generate the results in Fig. 8, such as IBM-Lagos, IBM-Perth, IBM-Jakarta, and IBM-Nairobi, are currently retired. As a result, we cannot reproduce results on these devices - however, running evaluations on the same benchmarks using currently available IBM devices should provide similar performance.
A.9. Methodology

Submission, reviewing and badging methodology:
- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

References


Xinhao Wang, Yuxuan Du, Yong Luo, and Dacheng Tao. Towards understanding the power of quantum kernels in the NISQ era. Quantum, 5:531, aug 2021.


