SELEQTOR: Intelligent Quantum Backend Selection and Runtime Estimation

Srikar Chundury*, Seongmin Kim†, Murali Gopalakrishnan Meena†, Chao Lu†, In-Saeng Suh†, Frank Mueller*
* Department of Computer Science, North Carolina State University, Raleigh, NC, USA
† National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN, USA
{schundu3, fmuelle}@ncsu.edu, {kims, gopalakrishm, luc1, suhi}@ornl.gov

Abstract*—Hybrid quantum–high-performance computing (Q-HPC) workflows are increasingly central to scaling quantum applications on noisy intermediate-scale quantum (NISQ) devices. Yet no single simulator or hardware backend performs best across all workloads, making it difficult for users to pick the right execution backend and for HPC schedulers to allocate resources effectively. This paper introduces SELEQTOR, which develops *meta-backend-driven benchmarking*, a technique that leverages workload-sensitive metrics to select the best backend per workload and provide runtime estimates for intelligent Q-HPC resource scheduling. SELEQTOR contributes two framework-agnostic tools. QBacMet (Quantum Backend Metrics) wraps the transpile–execute cycle as a context manager, instrumenting six execution-stack layers to produce 35-metric, 57-feature per-run vectors. MQBac (Meta Quantum Backend) trains dual Random Forest models on QBacMet data, a classifier that selects the best backend and a regressor that predicts runtime. Experiments across various benchmarks and backends confirm workload-specific strengths and uncover hidden performance crossovers opaque to traditional benchmarking. On a held-out workload split, MQBac selects the best backend with 0.950 accuracy (macro-F1 $= 0.906$), outperforming static baselines (heuristic: 0.650, always-NWQSim: 0.550), and delivers a $2.57\times$ mean wall-clock speedup over the always-NWQSim default. MQBac also predicts wall-clock runtime with $R^2 = 0.928$ across more than four orders of magnitude of execution time. To the best of our knowledge, SELEQTOR is among the first to combine automated, learned backend selection with runtime estimation across both classical simulators and hardware QPUs. By automating backend selection and runtime estimation, SELEQTOR improves resource allocation in Q-HPC systems.

***Index Terms*—Hybrid quantum-classical workflows, Quantum frameworks, Backend selection, Workload-sensitive metrics, Machine learning, Distributed architectures, Performance analysis**

I. INTRODUCTION

Many scientific and engineering workloads still exhibit exponential scaling on classical hardware, limiting practical problem size [1]–[4]. Quantum computing offers a promising paradigm by exploiting quantum parallelism, entanglement, and superposition [5], [6]. Progress in quantum hardware [1], [7]–[9] has delivered mid-scale quantum processing units (QPUs) that can act as accelerators alongside CPUs and GPUs in high-performance computing (HPC) environments, but limited qubit counts, device noise, and the diversity of backends complicate practical deployment [10]. Long-term directions such as topological quantum computing further highlight the breadth of hardware and architectural pathways under active study [11].

Crucially, no single simulator or hardware backend provides the best performance across all workloads [12]. Here and throughout, "best" denotes the backend achieving the lowest mean wall-clock execution time for a given workload. At the same time, HPC schedulers and resource managers require accurate runtime predictions to allocate nodes, GPUs, and wallclock-time to quantum jobs; without such estimates, quantum workloads either over-provision expensive accelerators or stall in queues due to insufficient reservations. Static decision rules, such as those of Alexeev [13], route circuits by qubit count, depth, and Clifford content. They provide useful teaching heuristics but cannot capture the nuanced crossovers revealed by real execution data. Recent systems, such as Maestro [14] and Sharma *et al.* [15], begin to automate selection via circuit analysis and predictive models, yet none leverage full-stack execution traces from HPC-scale runs. Neither do they frame automated selection with runtime prediction to inform resource scheduling decisions. Even simpler automatic selection already exists within a single simulator (e.g., Qiskit Aer's `automatic` method), but it does not scale to the many, constantly shifting factors that determine backend performance in a Q-HPC setting, motivating SELEQTOR's joint treatment of benchmarking, selection, and runtime estimation as one dependent pipeline.

The Quantum Framework (QFw) [12], [16]–[18] provides a backend-agnostic orchestration layer that integrates multiple local simulators (NWQ-Sim [19], Qiskit Aer, TN-QVM, QTensor) and cloud/hardware backends (ion traps from IonQ and superconducting devices from IBM-Q) under a unified interface on HPC systems. Prior work [12] benchmarked seven workloads across these backends at scale, confirming workload-dependent performance but with manual selection.

This paper introduces SELEQTOR, which contributes *meta-backend-driven benchmarking*, a technique that leverages workload-sensitive metrics to automatically orchestrate execution across multiple quantum backends to provide runtime estimates for intelligent Q-HPC resource scheduling. SELEQTOR contributes two components:

1) **QBacMet**: a vendor-agnostic, zero-change instrumentation tool that wraps the transpile–execute cycle as a context manager, capturing 35 metrics across six layers of the Q-HPC stack. Metrics range from Slurm

job metadata through circuit topology over transpilation overhead to backend-specific runtime characteristics.
2) **MQBac**: an ML-driven backend selector trained on QBacMet data that (a) classifies the best backend and (b) predicts expected runtime
3) We evaluate QBacMet and MQBac at scale on Frontier across eight benchmarks, validating selector accuracy and uncovering hidden performance crossovers, demonstrating meta-backend-driven benchmarking as a practical methodology for resource-aware scheduling.

II. Background

A. Quantum Framework (QFw)

QFw [12], [16]–[18] is a scalable, backend-agnostic orchestration platform for hybrid quantum–classical execution on HPC systems. It integrates state-vector, tensor-network, multi-method, cloud, and real-QPU backends under a unified Python interface compatible with Qiskit [20] and PennyLane [21]; the same application code targets any backend without modification [12]. The central dispatcher (Quantum Platform Manager) is the entry point for both QBacMet metric collection and MQBac intelligent selection introduced in this work.

B. Workloads

SELEQTOR targets both non-variational and variational workloads. Non-variational circuits, such as Greenberger-Horne-Zeilinger (GHZ) [22], Hamiltonian simulation (HAM) [23], Mermin-Bell, Bit Code [24], the transverse-field Ising model (TFIM) [25], [26], and Harrow-Hassidim-Lloyd (HHL) [27], execute fixed circuits without a classical parameter-update loop, exposing raw execution and communication characteristics. Phase-estimation-style kernels and implementations, e.g., LuGo, further expand this non-variational class [28]. Variational workloads, such as QAOA [29], and DQAOA [30], [31], couple parameterized ansatze with classical optimizers and amplify orchestration demands through iterative circuit evaluation. Together, these eight benchmarks span a wide range of circuit depth, qubit count, entanglement structure, and backend sensitivity, making them suitable for training ML models.

C. The Backend Selection Challenge

Simulators and QPUs differ fundamentally in architecture, gate support, parallelization strategy, and tolerance of circuit depth, such that choosing the most suitable for a given workload is non-trivial. Alexeev [13] proposed a rule-based decision tree (reproduced in Fig. 1) that routes circuits based on qubit count, depth, and Clifford content through state-vector, tensor-network, and Clifford simulators. While useful as a teaching heuristic, this static tree does not account for hardware resources, compilation overhead, or backend-specific optimizations. Practical experience [12] shows that the best backend can change at specific qubit or depth thresholds motivating SELEQTOR's data-driven approach.

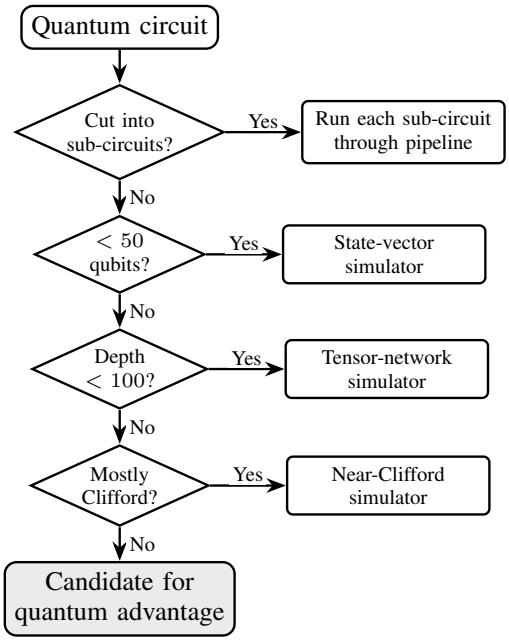


Fig. 1: Simulator selection heuristic [13].

III. Related Work

a) HPC–quantum Integration: Integrating quantum accelerators into classical HPC workflows is an active research area. Esposito and Haus [32] use Slurm heterogeneous jobs and MPI to co-schedule hybrid workloads. SCIM MILQ [33] adds circuit-cutting-aware scheduling to minimize makespan. XACC [34] provides a service-oriented framework for heterogeneous backends. MQSS [35] offers a full-stack platform at LRZ with dynamic compilation. Cloud aggregators (Amazon Braket [36], Azure Quantum [37], qBraid [38], Conqure [39]) demonstrate backend flexibility in remote settings. QFw [12] targets tightly coupled HPC–quantum execution with multi-backend support. Recent software-tool overviews and national-scale ecosystem efforts also emphasize backend interoperability and design automation, e.g., MQT [40], Q-Exa [41], and domain-specific toolchains such as D-Wave Ocean [42]. SELEQTOR adds automated metric collection and learned backend selection to this landscape.

b) Backend Selection and Intelligent Routing: Recent work studies how to *automatically* match circuits to the most suitable backend. Alexeev [13] sketches a manual decision tree based on qubit count, depth, and Clifford content, while subsequent surveys [43], [44] advocate ML-driven simulator selection. Maestro [14] proposes a predictive runtime model that routes circuits across state-vector, matrix product states (MPS), tensor-network, and stabilizer simulators, demonstrating gains in HPC batched workloads. Sharma *et al.* [15] introduce a three-module architecture (circuit analysis, capability mapping, multi-criteria ranking) for hardware-agnostic backend adaptation. EQC [45] partitions variational workloads across devices to mitigate noise. Qoncord [46] extends multi-device scheduling to variational algorithms.

QOS [47] and Qonductor [48] add cloud orchestration primitives. Nguyen [49] demonstrates reinforcement-learning-based selection in a serverless context. Ma and Li [50] instead predict a circuit's execution time on a single backend via a graph-transformer model trained on static circuit structure.

Automatic method selection also exists within a single simulator: Qiskit Aer's `automatic` mode picks among state-vector, MPS, and tensor-network methods from static circuit properties. This form of automation is narrower than the backend orchestration problem addressed here in two ways: It never considers hardware QPUs, and it is blind to the Slurm placement, calibration drift, and queuing effects that only appear once a job leaves a single process.

These closest prior works [14], [15], [49] share the motivation for ML-driven selection but differ from SELEQTOR in three critical ways: (1) *training data*: real HPC execution traces spanning MPI ranks, GPU placements, and cloud QPU queue delays on Frontier [51] are collected here, rather than synthetic benchmarks or limited local runs; (2) *scope*: the full HPC–cloud spectrum is covered here, trained jointly on local simulators and remote QPU services (IonQ, IBM), whereas [14] targets simulators only and [15] relies on circuit analysis alone; and (3) *dual objective*: runtime is predicted alongside backend choice, enabling HPC schedulers to right-size Slurm allocations before submission.

c) Middleware and Benchmarking: Pilot-Quantum [52], [53] provides resource and task management bridging HPC and quantum systems. Rallis *et al.* [54] survey the HPC–quantum interface design space. Murillo *et al.* [55] identify backend selection as a key open challenge. Li *et al.* [56] contribute QASMBench for NISQ evaluation. Chundury *et al.* [12] provide initial QFw benchmarks. SELEQTOR adds workload-sensitive metric collection and a learned selector on top of this middleware layer.

d) ML for Quantum Computing: Machine learning has been applied extensively to quantum computing tasks: variational algorithm optimization [57], [58], noise characterization and mitigation [59], [60], gate synthesis and compilation. However, these applications focus on *circuit* optimization, i.e., tuning parameters, decomposing gates, or characterizing errors, rather than the *backend orchestration* problem addressed here: Given a workload and a pool of heterogeneous backends (local simulators, cloud QPUs, GPUs), automatically predict which backend(s) will execute fastest and estimate the quantum kernel's runtime for resource scheduling when embedded in an HPC job. SELEQTOR targets exactly this backend-orchestration gap.

IV. METHODOLOGY

This section presents SELEQTOR's two tools: QBacMet and MQBac. Both are standalone Python packages compatible with any supported quantum framework backend. They are demonstrated within QFw, whose execution model is summarized first for context.

A. QFw Execution Model

A QFw job is submitted via Slurm with two heterogeneous groups: *hetgroup-0* for the application and *hetgroup-1* for QFw's Quantum Platform Manager (QPM) services and backend workers. The QPM dispatches circuit batches to local MPI workers or cloud REST endpoints and streams results back asynchronously. Backends and sub-backends are selected at runtime via a lightweight property dictionary, enabling zero-code-change substitution. Fig. 2 shows the framework-agnostic meta-backend pipeline that is the focus of this paper.

B. QBacMet: Workload-Sensitive Metric Collection

QBacMet (Quantum Backend Metrics) is a vendor-agnostic Python tool that attaches to any quantum job (both simulator and hardware device) without rewriting experiments. It distinguishes *features*, e.g., configurable/descriptive properties of a run, from *metrics*, i.e., quantitative values produced by execution.

TABLE I: QBacMet's six-layer metric hierarchy.

Layer	Name	Key items
0	Slurm/HPC	job ID, partition, nodelist, CPUs-on-node, scheduling policy, cluster name
1	Algorithm	width, depth, 2-qubit gates, Clifford/non-Clifford counts, critical-path length, connected components, variational parameters
2	Transpilation	optimization level, basis gates, routing method; *post-transpile*: hw_depth, hw_width, connectivity degree, SWAP count, 2-qubit gates
3	Backend	simulator flag, Aer method (SV/MPS/TN), bond dimension, device (CPU/GPU), precision; *hardware*: CLOPS (Circuit Layer Operations Per Second), QV (Quantum Volume), T_1/T_2, gate/readout errors, calibration age
4	Execution	queue-time, execution-time, MPI rank, OpenMP threads, HPC job ID
5	Post-process	mitigation overhead, quantum error correction (QEC) syndrome time, result marshaling time

a) Six-layer hierarchy: QBacMet organizes metric collection into six layers that mirror the Q-HPC execution stack (Tab. I): Layer 0 (Slurm orchestration) determines node affinity and MPI rank assignment, which drive communication patterns invisible to circuit-only analysis. Layer 1 (algorithmic circuit structure) defines entanglement and gate topology. Layer 2 (transpilation) introduces hardware-specific compilation overhead that can exceed algorithmic depth by $5\times$. Layer 3 (backend characteristics) encodes simulator method (MPS vs. tensor-network contractibility) or QPU error rates. Layer 4 (runtime execution) captures wall-clock time and queuing dynamics. Layer 5 (post-processing) adds error-mitigation and marshaling latency. This vertical decomposition enables MQBac to identify which layer's features dominate

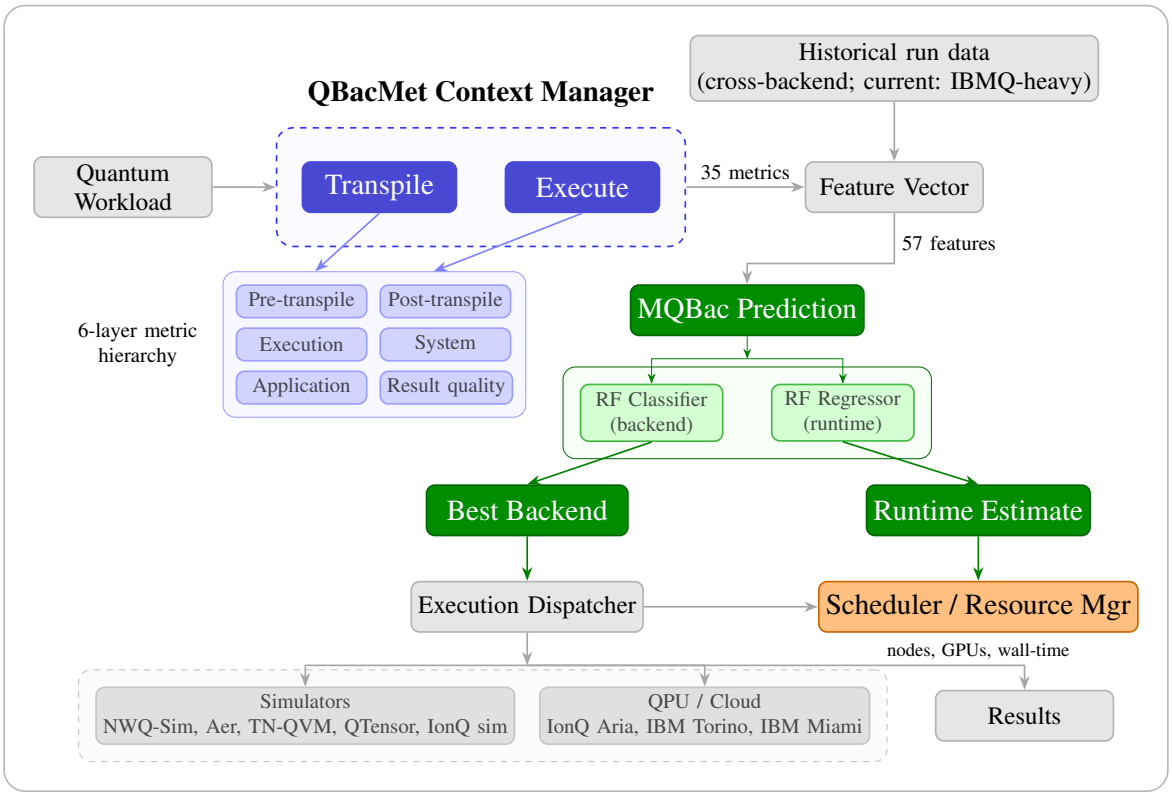


Fig. 2: Meta-backend-driven benchmarking pipeline: QBacMet collects full-stack metrics and MQBac selects the best backend and estimates runtime, enabling cross-platform quantum resource scheduling with no changes to user code.

the selection decision (features from Layers 2 and 3 rank highest, see Sec. VI-B), and future extensions can selectively re-collect only modified layers when infrastructure changes. These six layers and their 35 metrics were chosen manually, by mapping the Q-HPC execution stack layer by layer, since no existing full-stack taxonomy was available to adapt (hence making our approach more novel). Their predictive value was then confirmed empirically via feature importance.

b) Zero-change instrumentation: Many profiling tools require users to wrap executions or add import-time hooks. QBacMet instead exposes a Python context manager that intercepts standard transpile and execution entry points of Qiskit and PennyLane at the library level, eliminating the need for user code changes. This design choice reflects two practical considerations: (i) *adoption*, where users can add a single `with collector.collect():` block around existing code and reap benefits without refactoring applications; and (ii) *correctness*, where the context-manager pattern ensures the collection lifecycle (setup, activate, execute user code, collect, deactivate, flush) is well-defined and exception-safe (metrics are collected even if the circuit execution throws an error). On exit, it collects all six layers and restores the original call paths, leaving user circuits and orchestration scripts unchanged. The typical usage pattern is given in Fig. 3.

c) Rich feature extraction: Beyond the six-layer runtime metrics, QBacMet extracts 40+ static circuit features from any

```
from qbacmet import collector

with collector.collect():
    qc = QuantumCircuit(n)
    # ... build circuit ...
    tqc = transpile(qc, backend,
     opt_level=1)
    result = backend.run(tqc,
    shots=1024).result()

metrics = collector.metrics
```

Fig. 3: QBacMet context-manager usage: `collect()` installs hooks around the transpile–execute cycle and records timing and circuit metrics as user code runs inside the block; on exit, it finalizes all six layers into `collector.metrics`. No change to the user's quantum code is required.

compiled circuit: gate-arity counts (1Q–4Q), directed acyclic graph (DAG) statistics (edge count, longest path, serial layers), topology metrics (connected components, tensor factors, distinct 2Q pairs), per-qubit touch counts, Clifford/non-Clifford classification, parameterization statistics, and per-gate-type counts. Feature extraction proceeds in two phases. First, static circuit analysis on the compiled DAG yields gate counts (1-4 qubits) and DAG graph metrics (edge count, longest

path, number of serial layers for parallelization potential). Second, *Clifford-content* classification leverages Qiskit's CliffordSpan decomposition to partition the circuit into Clifford-native subsets (executable on stabilizer simulators via tableau algorithms in O(n^3) time) versus non-Clifford parts (requiring exponential simulation). Per-qubit touch counts (how many gates operate on each qubit) correlate with routing overhead, as frequently-accessed qubits may require more SWAP insertions on constrained topologies. These features are extracted *post-transpile*, after basis-gate substitution and routing, i.e., they capture the actual execution complexity facing the backend rather than the idealized algorithmic circuit. These features form the input to MQBac.

d) Persistent metric store: Each collection block writes a single JSON snapshot keyed by the tuple *(benchmark, size, backend, sub-backend, device, run mode, nodes, processes, timestamp)* into a shared store. Snapshots are append-only and schema-tolerant: Layers that do not apply to a given backend (e.g., hardware error rates on a simulator) are recorded as null rather than rejected, and new layer-keys added by new backends do not invalidate old snapshots. This hierarchical schema design enables two critical capabilities: (i) *schema evolution*, which ensures new keys are added when a new backend introduces additional Layer-3 metrics (e.g., a QPU with exotic calibration parameters), while missing keys are gracefully skipped; and (ii) *incremental collection* with a persistent store that is append-only, i.e., each new job run appends one snapshot without requiring schema migration or re-processing of prior runs. This design is critical for a tool running continuously on a supercomputer for months, where new hardware or backend versions arrive during non-stop operation. A separate flattening step joins the snapshots with the static circuit-feature table and emits two compact tables, one per workload tuple for classification and another per execution for regression. These tables are the direct inputs to MQBac. The decoupling in separate tables lets QBacMet collection and MQBac training proceed independently on different cadences and on different machines.

C. MQBac: Intelligent Backend Selector

MQBac (Meta Quantum Backend) is a standalone ML-based backend selector. Given a set of QBacMet features for an incoming circuit batch, MQBac predicts the best-suited backend and an expected runtime. In the QFw deployment evaluated here, MQBac replaces the manual backend property (Fig. 2) so that the same artifact can be loaded into any other Python orchestration layer unchanged. MQBac addresses two complementary prediction tasks.

a) Need for historical data: MQBac is deliberately trained on the accumulated QBacMet snapshot store rather than on freshly generated benchmarks for each prediction. On-demand benchmarking is impractical at Q-HPC scheduler scale: QPU runs are queue-bound, QPU access is metered, and calibration drift changes absolute runtimes between windows. The *relative* ordering of backends is more stable than absolute runtimes, so a model trained on the historical sweep captures this ordering and only requires periodic top-up snapshots to track drift. The persistent store described above is therefore the natural training substrate: Each new job that runs through QFw appends one snapshot, and MQBac is retrained off-line against the growing corpus.

b) Task 1: Backend selection (classification): For each workload tuple (benchmark, qubit count, node count, process count), the backend with the lowest mean wallclock time is labeled as the target class. A random-forest classifier is trained on the QBacMet feature vectors under a stratified train/validation/test split. Label-leaking columns (the oracle runtime and the winning sub-backend identifier) and columns collinear with retained features (e.g., qubit count vs. size) are dropped before fitting. Cross-validation groups are formed from workload identifiers to reduce near-duplicate leakage. Random Forests were chosen over alternatives (Support Vector Machines (SVM), gradient-boosted trees, neural networks) based on: (i) its natural handling of mixed categorical (backend identifiers) and numeric features without explicit encoding; (ii) its robustness to feature scaling differences (e.g., Slurm node counts in the range of 1–32 vs. circuit depths in the range of 1–1000) and to collinear features common in circuit descriptors; (iii) interpretable impurity-based feature importance (Sec. VI-B), validating that the model learns sensible heuristics rather than spurious correlations; and (iv) rapid training (seconds on a single core), enabling frequent retraining without disrupting the scheduler. Initial XGBoost experiments showed marginal gains ($< 2\%$ accuracy) at substantially higher cost.

c) Task 2: Runtime estimation (regression): A random-forest regressor predicts mean wall-clock time given the same feature set augmented with the backend identifier. To avoid double-counting repeated executions of the same workload, rows are grouped by the workload tuple introduced above and the target is taken as the per-group mean wall-clock time, with numeric features collapsed by group mean. This makes the held-out R^2 comparable across benchmarks with different numbers of repeats, using a random train/validation/test split with grouped cross-validation. The regression target is transformed as $\log(1 + \text{mean\_ms})$, and we report the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) on the back-transformed millisecond scale while reporting R^2 on log-scale.

d) Shared pipeline and feature engineering: Both models share a single preprocessing pipeline: Categorical columns are one-hot encoded with unknown-category tolerance, numeric columns are median-imputed, and the resulting vector is passed to the random-forest estimator. Starting from the processed circuit and system feature set used by MQBac, highly correlated pairs (Pearson $r > 0.95$) are greedily pruned separately for each model. Feature importance is assessed via impurity-based importance. Principal Component Analysis (PCA) is used as a diagnostic visualization after redundancy pruning rather than as a training-stage dimensionality reduction step. In the current regression table, Pearson pruning at $r > 0.95$ reduces the PCA input from 104 numeric features to 44, and the first two components explain 32.1% of the re-

maining variance, confirming that redundancy is concentrated in a few large feature clusters rather than uniformly across all descriptors. Here, the counts refer to different stages: QBacMet logs 35 metrics, MQBac trains on a curated 57-feature modeling view, and the correlation/PCA diagnostic operates on the 104 numeric columns present after preprocessing.

e) Hyperparameter selection and reproducibility: The classifier uses 300 estimators with max-depth 16 and minimum leaf size 2, and the regressor uses 500 estimators; feature importance is computed via Gini impurity. Categorical features are one-hot encoded with unknown-category tolerance, where unseen categories are handled by filling a zero vector. Numeric features are imputed with column-wise median, robust to outliers (e.g., a single QPU run queued for hours does not skew the distribution). The random seed is fixed at 42 for all sklearn operations to ensure deterministic, easily reproducible results.

f) Deployment: MQBac exposes a single `predict` entry point that returns the recommended backend label and an estimated runtime. As a standalone library it can be invoked from any Python application before circuit submission, and within QFw the trained artifact is loaded into QPM at startup and consulted by the dispatcher in place of the manual backend property (Fig. 2). User override and confidence-based fallback are retained in both modes. Inference is negligible, seen by a single forest evaluation requiring sub-millisecond time on a dispatcher CPU, which is well below the per-job Slurm-launch overhead. Retraining both forests on the full corpus completes in seconds on a single login-node core, allowing the artifact to be regenerated periodically as the snapshot store grows without disrupting the running scheduler.

Five simulator backends are integrated: NWQ-Sim (state-vector, MPI-distributed), TN-QVM [61] and QTensor [62] (tensor-network), Qiskit Aer (multi-method with state-vector, MPS, and automatic modes), and IonQ (cloud REST simulator). QPU backends include IonQ Aria (trapped-ion), IBM Torino, and IBM Miami (superconducting). Each backend implements a standardized API. Backends are selected at runtime via a lightweight property dictionary, enabling zero-code-change substitution [12]. Applications interact with the framework through a lightweight backend client. Non-variational workloads are batched across available cores, whereas variational workloads issue asynchronous calls per optimizer iteration. DQAOA decomposes a large Quadratic Unconstrained Binary Optimization (QUBO) problem into sub-QUBOs that are solved concurrently via threaded RPC calls [12]. This modular model enables QBacMet and MQBac to attach transparently without modifying application logic.

V. Experimental Setup

a) Compute Platform: Experiments for SELEQTOR were conducted on a 32-node Frontier test cluster at the Oak Ridge Leadership Computing Facility (OLCF) [51]. Each node has a 64-core AMD EPYC CPU (512 GiB DDR4) and four AMD MI250X GPUs (8 Graphics Compute Dies (GCDs), 64 GiB HBM2e each), connected via HPE Slingshot 200. With one core per Last-Level Cache (LLC) reserved for system processes, 56 application cores per node are available.

b) Job Orchestration: QPM spawns eight worker threads and dispatches tasks round-robin via the PMIx Reference RunTime Environment (PRTE) for local jobs or REST for IonQ cloud jobs [63]; QBacMet and MQBac attach to this pipeline (Sec. IV-A) without depending on it. Each experiment is repeated three times, mean and standard deviation are reported.

c) Backends and Benchmarks: Experiments span the five simulator and three QPU backends integrated into QFw (Sec. IV-C); IonQ Aria is transpile-only here, used for backend-characterization features rather than execution runtime, and IBM Torino has 133 qubits. Eight benchmarks span non-variational kernels (SupermarQ GHZ, SupermarQ HAM, SupermarQ Mermin-Bell, SupermarQ Bit Code, TFIM, and HHL) and variational workloads (QAOA of QUBO sizes 4–30 and DQAOA of QUBO size 30 and 40 with varying sub-problem splits).

d) QBacMet Data Collection: QBacMet (Sec. IV-B) collected a benchmark corpus across benchmark$\times$backend$\times$size configurations, including real-QPU runs on IBM Torino and IBM Miami. The resulting flattened feature tables serve as the training set for MQBac.

e) MQBac Training and Evaluation: The two MQBac models, namely the backend-selection classifier and the runtime regressor, are trained and evaluated on stratified/random 8:1:1 train/validation/test splits described in Sec. IV-C. We report classification accuracy, macro-F1, per-class precision/recall, and the confusion matrix. We also report MAE, RMSE, and R^2 for regression. For robustness, classification additionally uses GroupKFold ($k = 3$) cross-validation on the training partition and regression uses KFold ($k = 3$) cross-validation. Backend-selection performance is contrasted against the static heuristic of Fig. 1 and the always-NWQSim default.

f) Baselines and speedup computation: The static baseline reproduces the simulator-selection flowchart of Fig. 1 [13]: Small-width circuits are routed to the state-vector family, low-depth circuits to the tensor-network family, near-Clifford circuits to the stabilizer path, and the residual "quantum-advantage" branch is treated as an abstention. The heuristic is scored *family-correct*: A prediction counts if the true winning backend belongs to the predicted family. Accuracy is reported both on the simulator-only subset (where the heuristic has applicable rules) and on the full test split (treating abstentions as incorrect, so that the number is directly comparable to MQBac). The wall-clock speedup of MQBac over the always-NWQSim default is computed by looking up per-tuple mean runtime in the regression table for both the predicted and the default backends on every test row and taking the mean of the per-row ratios.

VI. Results and Discussion

A. QBacMet Analysis

a) Depth inflation: Fig. 4 plots the ratio of post-transpilation hardware depth to pre-transpilation algorithmic depth for each benchmark×backend combination. The median inflation is 2.9×, with observed maxima above 30× on challenging mappings. This metric distinguishes backends that preserve shallow circuits from those that incur large routing overhead, and enters MQBac's top-ranked features for both backend selection and runtime prediction (Fig. 6).

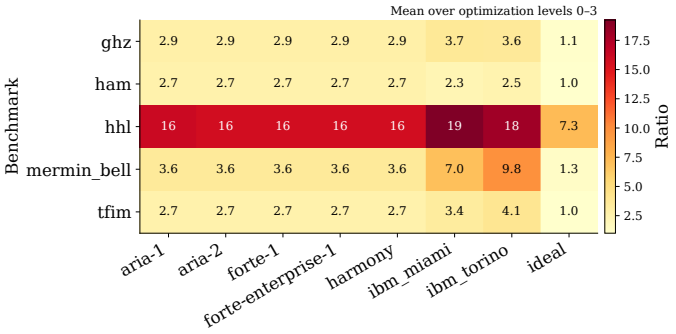


Fig. 4: Mean depth inflation (HW depth / algorithmic depth) per benchmark × backend. Depth can inflate by up to 30×, revealing that compilation overhead, invisible to pre-transpilation heuristics, is a primary driver of backend performance differences.

b) QBacMet exposes hidden complexity: Depth inflation ratios reveal that backend sensitivity is not solely a function of algorithmic circuit structure; compilation routing, basis-gate decomposition, and device topology can inflate depth by up to 30×. Framework choice further complicates this landscape: Qiskit's `transpile()` applies iterative optimization (levels 0–3) with Sabre or Stochastic routing, while PennyLane delegates compilation to underlying device APIs with less control. Different frameworks may produce post-transpile depths differing by 20–30% for the same circuit and backend, yet SELEQTOR's post-transpile metrics capture these framework artifacts and integrate them into a single decision model. Features invisible to pre-transpilation analysis, such as SWAP insertion count and post-transpile connectivity degree, carry significant predictive weight, validating the multi-layer design of QBacMet.

c) Hidden performance crossovers: Fig. 5 plots mean wall-clock runtime vs. qubit count with separate curves per backend for the two benchmark families (GHZ and TFIM) supported by multi-backend coverage across a sufficient qubit-size range. For both GHZ (Fig. 5a) and TFIM (Fig. 5b), backend-curve intersections and rank-order changes with qubit growth confirm the crossover behavior that motivates automated selection, which cannot be captured by a single static heuristic. These intersections also show the hardware–simulator crossover: Hardware curves are comparatively flatter with qubit growth, while state-vector simulator cost rises rapidly, making static backend choices unreliable. Before the crossover, i.e., at smaller qubit counts, simulators are clearly faster: NWQ-Sim leads on both GHZ and TFIM in this region. The crossover to hardware occurs (barring MPS) around 25 qubits for GHZ and around 28 qubits for TFIM, with IonQ/IBM rank-order reversals visible where both hardware curves are present. Missing points in these crossover plots are not measurement omissions; they correspond to runs that exceeded user Slurm wall-time limits and were terminated before completion.

B. SELEQTOR Selector Accuracy

a) Backend selection (classification): SELEQTOR's MQBac achieves **0.950** accuracy (macro-F1 = 0.906) on the held-out workload split, outperforming static baselines (heuristic: 0.650, always-NWQSim: 0.550; Fig. 7). Translated into runtime, SELEQTOR-recommended backends are on average **2.57**× faster than the always-NWQSim default.

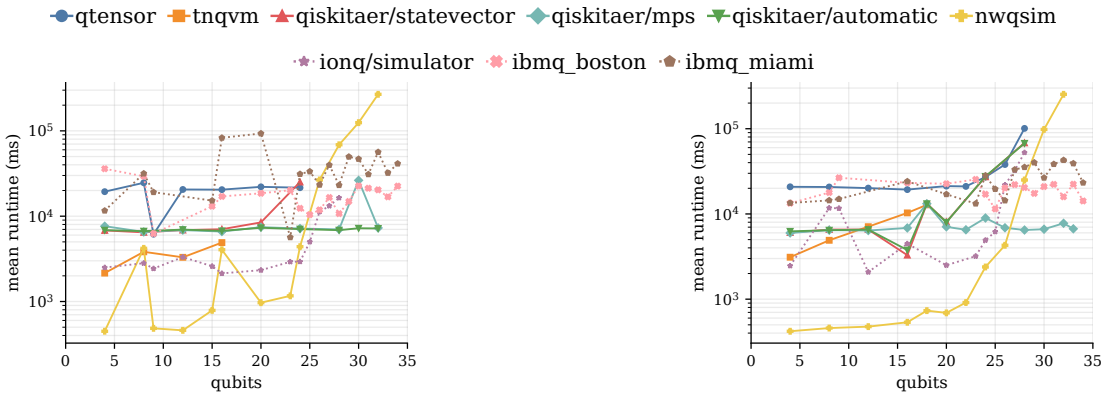
TABLE II: Regression performance.

Metric	Value
MAE (ms)	15301.86
RMSE (ms)	59094.24
R^2	0.928

b) Runtime estimation (regression): The runtime regressor achieves strong predictive accuracy across four orders of magnitude in wall-clock runtime (Fig. 8), with detailed metrics in Tab. II. The largest residuals occur on hardware backends, where queuing variability inflates measured time beyond what circuit-only features can predict.

c) Error analysis and failure modes: While achieving 95.0% accuracy on held-out workloads, SELEQTOR's MQBac chooses an inferior backend in a small fraction of cases (∼5.0%). Analysis reveals two primary failure modes: (i) *Hardware queuing variability*. Wall-clock runtime can fluctuate by 30–50% on IBM Torino and Miami depending on queue depth at submission time, which degrades prediction stability. (ii) *Imbalanced training data* in sparsely sampled workload regimes lowers per-class precision. On the simulator-only subset (no queue noise), MQBac maintains high accuracy, indicating the core selection logic is sound, i.e., it is the external variability that drives the failures, not a flawed model.

d) Generalization: To assess generalization beyond the training distribution, we perform leave-one-backend-out evaluation: For each backend, its rows are removed from training only, and the resulting classifier is re-evaluated on the original held-out test set. The accuracy drop scales with each backend's share of the training data, reaching -0.50 when the majority-class backend (NWQ-Sim) is withheld and averaging 0.18 absolute across all backends, with markedly smaller impact for less-represented classes. This indicates that the classifier depends on seeing sufficient examples of the dominant backends rather than on backend-idiosyncratic signatures, i.e., fine-tuning will likely be needed to transfer to new platforms or hardware.



(a) GHZ. (b) TFIM.

Fig. 5: Mean runtime (log scale) versus qubit count per backend. Backend-curve intersections and rank-order changes across qubit counts reveal hidden crossovers that static heuristics struggle to capture, motivating automated selection. Missing data points indicate runs terminated after exceeding user Slurm wall-time limits.

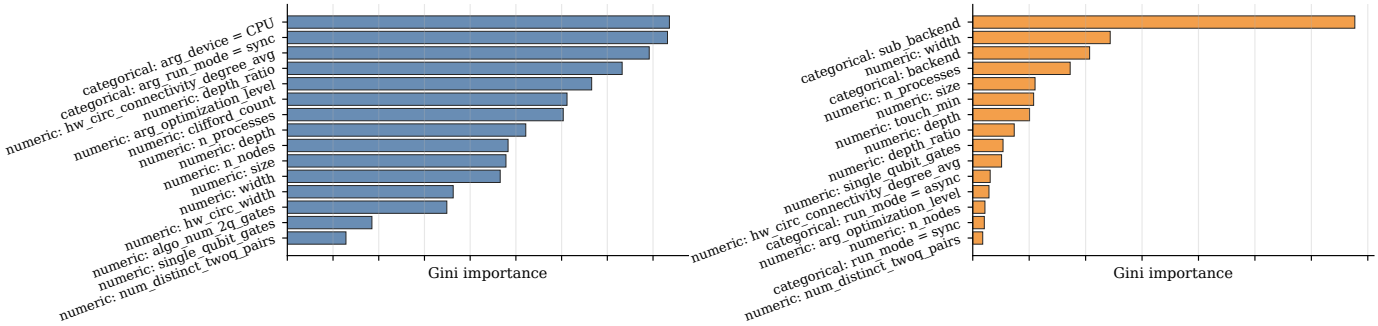


Fig. 6: Top-15 Gini importance features for SELEQTOR's MQBac classifier (left) and regressor (right), visualizing the distinct feature hierarchies. Hardware-connectivity and execution-configuration features (`hw_circ_connectivity_degree_avg`, `arg_device`, `arg_run_mode`) dominate backend selection, while backend identity and circuit-scale features dominate runtime prediction, confirming that full-stack metrics are essential for accurate cross-platform decisions.

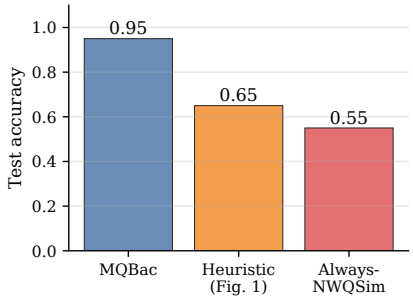


Fig. 7: SELEQTOR's MQBac test accuracy vs. static baselines. MQBac achieves 95.0% accuracy, clearly exceeding the heuristic baseline (65.0%) and always-NWQSim (55.0%), showing the advantage of learned cross-platform selection.

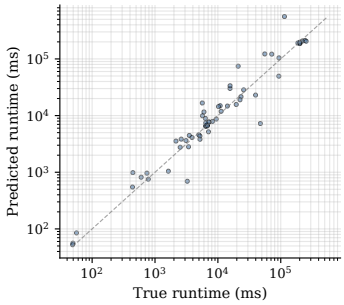


Fig. 8: Predicted vs. true wall-clock runtime per workload tuple on the held-out test split (log–log). Strong agreement across four orders of magnitude ($R^2 = 0.928$, log-scale) enables HPC schedulers to right-size resource allocations.

e) Feature importance: The choice of Random Forest over a single decision tree (cf. the Alexeev heuristic in Fig. 1) is deliberate. Alexeev's flowchart encodes one static path through a few hand-picked thresholds, while a Random Forest grows many trees in parallel, each sampling a random subset of features. In practice, each tree tends to specialize on a different layer of the QBacMet hierarchy, i.e., one tree may split on qubit count (Layer 1), another on post-transpile SWAP count (Layer 2), a third on MPI process count (Layer 4). Additionally, the ensemble vote aggregates these layer-specific

signals into a single robust decision.

This is reflected in Fig. 6: The top classifier features are execution-configuration signals (`arg_device=CPU`, 0.084; `arg_run_mode=sync`, 0.083), followed by hardware-connectivity and compiled-depth structure (`hw_circ_connectivity_degree_avg`, 0.079; `depth_ratio`, 0.073; `arg_optimization_level`, 0.067). System-scale parameters (`n_nodes` + `n_processes`, ≈0.109 combined) also rank, because backends differ fundamentally in MPI scaling, e.g., NWQ-Sim benefits from many nodes while a QPU is a single-shot device indifferent to node count. The expected state-vector scaling of NWQ-Sim still dominates at larger qubit sizes: Execution time becomes increasingly expensive and then practically infeasible beyond roughly 35 qubits in our setting. The runtime regressor is dominated by backend identity (`sub_backend`, 0.339; `backend`, 0.104), as each sub-backend exposes a qualitatively different runtime profile, with Qiskit Aer automatic mode also showing workload-dependent behavior, aligning with tensor-network-like scaling on GHZ but following a distinct trajectory on TFIM. It also draws additional signal from circuit width and parallelism (`width`, 0.122; `n_processes`, 0.086), followed by size/depth locality terms (`size`, 0.055; `touch_min`, 0.054; `depth`, 0.050). The regressor's largest residuals occur on hardware backends where queue depth (invisible to any circuit feature) inflates measured time unpredictably. Among classical simulators, MPS most closely tracks hardware scaling trends in our benchmarks while remaining faster than hardware across the tested sizes.

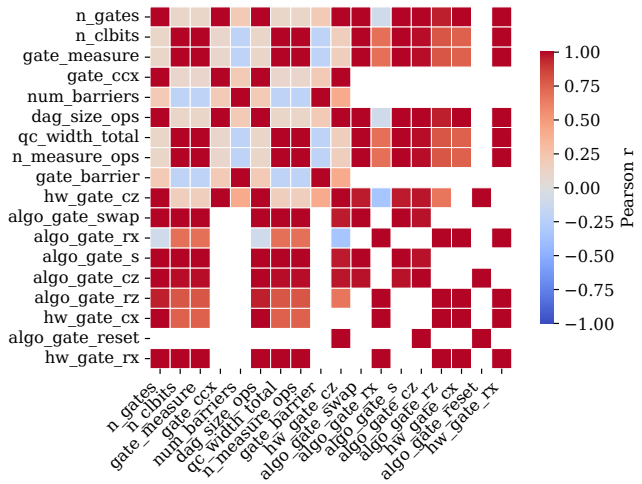


Fig. 9: Pearson-correlation structure among numeric QBacMet-derived features. Large blocks of nearly interchangeable descriptors reveal structural redundancy: 200 pairs perfectly correlated, 1532 satisfy $|r| \geq 0.80$.

f) Feature correlation: Fig. 9 shows two complementary views of redundancy in the training table. On the left, the correlation matrix exposes large blocks of nearly interchangeable descriptors: size, width, classical-bit count, and measurement count move together because they all scale with circuit width; several gate-family counters track the same basis-decomposition decisions; and hardware-facing gate counts align tightly with controlled-gate and routing-heavy workload features. For clarity, this 104-column matrix is the post-preprocessing numeric analysis view, distinct from the 35 raw QBacMet metrics and the 57-feature MQBac training view. Across the 104 numeric features analyzed, 200 pairs are perfectly correlated and 1532 pairs satisfy $|r| \geq 0.80$, so the redundancy is structural rather than incidental.

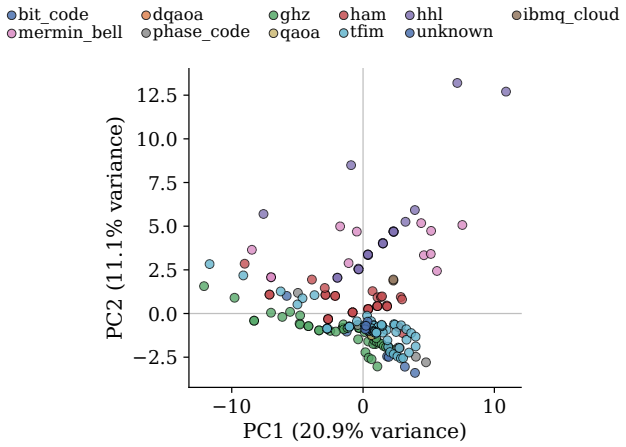


Fig. 10: PCA projection of the 44 features remaining after pruning Pearson $r > 0.95$ pairs. Each point is one (benchmark, size, backend) aggregate. Benchmark families occupy distinct regions (PC1 = 20.9%, PC2 = 11.1% explained variance), confirming that meaningful variation persists despite feature redundancy.

Fig. 10 asks a complementary question: After removing features with Pearson $r > 0.95$, does meaningful variation remain? The answer is yes. Pruning reduces the PCA input to 44 features, and the first two components still capture 32.1% of the remaining variance (PC1 = 20.9%, PC2 = 11.1%). PC1 is dominated by two-qubit interaction structure and width-related terms (`algo_gate_cx`, `algo_num_2q_gates`), while PC2 carries parametric-gate count, register structure, and depth effects that are not purely size-driven. Despite feature redundancy, the Random Forest achieves **95.0%** accuracy by selecting the most informative feature per split.

g) From runtime estimation to resource scheduling: MQBac's regression model predicts wall-clock execution time before a circuit batch is submitted. HPC schedulers can use this to right-size Slurm allocations (nodes, GPUs, and wall-time limits) before quantum jobs enter the queue. Over-provisioning wastes scarce accelerator hours; under-provisioning triggers job failures or re-queuing. On our held-out split, the regressor attains $R^2 = 0.928$ (log-scale) with MAE = 15.30 s and RMSE = 59.09 s across runtimes spanning more than four orders of magnitude, accurate enough for Slurm wall-time requests to track actual job duration within a small constant factor.

h) Comparison with concurrent work: Maestro [14] and Sharma *et al.* [15] remain the closest concurrent efforts. PilotQuantum [52] offers complementary resource management primitives that could be paired with SELEQTOR's workload-aware selection.

C. Deployment Scenarios

1) *One-shot backend choice:* When a workflow must pick one backend before launch, the classifier maps the workload descriptor directly to the fastest (or "best") admissible target.
2) *Scheduler-facing planning:* When a user or scheduler must request wall time, node counts, or GPU resources before submission, the regressor predicts the runtime consequences of candidate choices.
3) *Limited backend menus:* When a deployment exposes only a subset of simulators or one cloud provider, SELEQTOR uses the classifier to narrow the admissible set and the regressor to compare remaining options under local resource constraints.

VII. FUTURE WORK

This work opens several directions. *Robust ML*: Online learning and ensemble approaches can mitigate hardware queuing variance and calibration drift, while active sampling addresses under-represented workload–backend pairs. *Hardware expansion*: transfer learning can extend coverage to trapped-ion and photonic QPUs and additional HPC platforms (Perlmutter, Aurora) with minimal retraining. *Metric hierarchy*: QBacMet's schema already defines Layer 3 and 5 fields for device error rates (T1/T2, CX infidelity) and post-execution fidelity; populating them consistently as backend APIs standardize will further improve feature ranking. *Production integration*: Scheduler plugins may support efforts to right-size wall-time and compute requests at submission, and support online model adaptation when mid-execution metrics diverge from predictions. *Co-Scheduling*: MQBac can be extended to co-schedule sub-circuits post circuit cutting across simulator backends and QPU devices, leveraging SELEQTOR's runtime estimates to choose the best partition and route sub-circuits for the shortest makespan. *Multi-criteria backend definitions*: The current notion of "best" backend is defined by mean wall-clock time; extending MQBac's objective to jointly weigh result fidelity, monetary cost, and queue policy would better reflect practical Q-HPC scheduling trade-offs. SELEQTOR's methodology is agnostic to the criterion used to label the "best" backend, so fidelity-, cost-, or queue-aware objectives can be substituted for wall-clock time without changing QBacMet's metric collection or MQBac's learning pipeline.

VIII. CONCLUSION

This paper contributed SELEQTOR, a *meta-backend-driven benchmarking* approach that combines workload-aware metric collection with learned backend selection and runtime estimation to support intelligent Q-HPC resource scheduling. SELEQTOR's two standalone tools, QBacMet and MQBac, work with any Qiskit- or PennyLane-compatible backend and were evaluated here within an HPC quantum framework.

Evaluations on a Frontier-collected corpus spanning eight benchmarks expose performance crossovers hidden from traditional benchmarking. QBacMet's depth-inflation analysis reveals that compilation overhead alone can shift the best backend choice, a factor invisible to pre-transpilation heuristics. MQBac achieves **95.0%** backend-selection accuracy (macro-F1 $= 0.906$), a **2.57**$\times$ mean wall-clock speedup over the always-NWQSim default, and runtime prediction with $R^2 =$ **0.928** (log-scale).

To the best of our knowledge, SELEQTOR is among the first to unify automated backend selection with data-driven runtime estimation across classical simulators and hardware QPUs in a single learned decision framework. By turning backend selection and runtime estimation into an automated, data-driven process, this work improves backend selection for quantum kernels and helps HPC schedulers right-size job allocations for both simulator and hardware execution.

ACKNOWLEDGMENT

This work was supported in part by NSF awards OSI-2531350, OSI-2410675, PHY-2325080, CISE-2316201, OMA-2120757, CCF-2217020, and PHY-1818914 as well as DOE DE-SC0025384. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Advanced Scientific Computing Research programs in the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Quantum Science Center. Notice: This manuscript has been co-authored by UT-Battelle LLC under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

GENERATIVE AI DISCLOSURE

In accordance with IEEE policy, we disclose the use of generative AI assistants during manuscript and code development. GitHub Copilot [64], OpenAI's ChatGPT (GPT-4) [65], and Anthropic's Claude [66] were utilized as coding and drafting assistants. These tools aided in writing Slurm scripts, collecting statistics, developing Python plotting scripts, implementing scikit-learn random forest models, and cleaning Sections I, III, IV, V, and VI, tables. All generated content was reviewed and validated by the authors before use.

REFERENCES

[1] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 2019.

[2] S. Kim, W. Shang, S. Moon, T. Pastega, E. Lee, and T. Luo, "High-performance transparent radiative cooler designed by quantum computing," *ACS Energy Letters*, vol. 7, no. 12, pp. 4134–4141, 2022.

[3] S. Kim, S. Jung, A. Bobbitt, E. Lee, and T. Luo, "Wide-angle spectral filter for energy-saving windows designed by quantum annealing-enhanced active learning," *Cell Reports Physical Science*, vol. 5, no. 3, 2024.

[4] S. Kim, S.-W. Ahn, I.-S. Suh, A. W. Dowling, E. Lee, and T. Luo, "Quantum annealing for combinatorial optimization: a benchmarking study," *npj Quantum Information*, vol. 11, no. 1, p. 77, 2025.

[5] G. Q. AI and Collaborators, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, no. 8052, pp. 920–926, 2025.

[6] A. Montanaro, "Quantum algorithms: An overview," *npj Quantum Information*, vol. 2, no. 1, p. 15023, 2016.

[7] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042319, 2007.

[8] C. Monroe and J. Kim, "Scaling the ion trap quantum processor," *Science*, vol. 339, no. 6124, pp. 1164–1169, 2013.

[9] Y. Alexeev, D. Bacon, K. R. Brown, R. Calderbank, L. D. Carr, F. T. Chong, B. DeMarco, D. Englund, E. Farhi, B. Fefferman *et al.*, "Quantum computer systems for scientific discovery," *PRX Quantum*, vol. 2, no. 1, p. 017001, 2021.

[10] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[11] M. Freedman, A. Kitaev, M. Larsen, and Z. Wang, "Topological quantum computation," *Bulletin of the American Mathematical Society*, vol. 40, no. 1, pp. 31–38, 2003.

[12] S. Chundury, A. Shehata, S. Kim *et al.*, "Scaling hybrid quantum–HPC applications with the quantum framework," in *Proceedings of the SC'25 Workshops*, 2025.

[13] Y. Alexeev, "ASPLOS: Quantum applications," Invited talk, Planning Workshop on Quantum Computing (PlanQ 2024), co-located with ASPLOS 2024, San Diego, CA, USA, Apr. 2024, slide 3, "Quantum benchmarks". [Online]. Available: https://palnqc24-fmuelle-dfadda480377b1fee71e373dcf0f6fae139ca245338441.gitlab.io/yuri.pdf

[14] O. Bertomeu, H. Ghayas, A. Roman, and S. DiAdamo, "Maestro: Intelligent execution for quantum circuit simulation," 2025. [Online]. Available: https://arxiv.org/abs/2512.04216

[15] A. Sharma, S. Kailash, and D. Ethirajan, "Hardware agnostic backend adaptation through quantum circuit evaluation," *EPJ Web of Conferences*, vol. 360, p. 01009, 2026.

[16] S. Chundury, A. Shehata, T. Naughton III, S. Kim, F. Mueller, and I.-S. Suh, "QFw: A quantum framework for large-scale hpc ecosystems," Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), Tech. Rep., 11 2024. [Online]. Available: https://www.osti.gov/biblio/2498439

[17] Z. Xu, S. Chundury, S. Kim, A. Shehata, X. Li, A. Li, T. Luo, F. Mueller, and I.-S. Suh, "Gpu-accelerated distributed qaoa on large-scale hpc ecosystems," 2025. [Online]. Available: https://arxiv.org/abs/2506.10531

[18] A. Shehata, T. Naughton, and I.-S. Suh, "A framework for integrating quantum simulation and high performance computing," 2024. [Online]. Available: https://arxiv.org/abs/2408.08098

[19] A. Li, B. Fang, C. Granade, G. Prawiroatmodjo, B. Hein, M. Rotteler, and S. Krishnamoorthy, "Sv-sim: Scalable pgas-based state vector simulation of quantum circuits," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.

[20] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross *et al.*, "Quantum computing with qiskit," *arXiv preprint arXiv:2405.08810*, 2024.

[21] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[22] D. M. Greenberger, M. A. Horne, and A. Zeilinger, "Going beyond bell's theorem," 2007. [Online]. Available: https://arxiv.org/abs/0712.0921

[23] E. Ising, "Beitrag zur theorie des ferromagnetismus," *Zeitschrift für Physik*, vol. 31, no. 1, pp. 253–258, 1925.

[24] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong, "Supermarq: A scalable quantum benchmark suite," 2022. [Online]. Available: https://arxiv.org/abs/2202.11045

[25] E. Fradkin, *Field Theories of Condensed Matter Physics*, 2nd ed. Cambridge: Cambridge University Press, 2013.

[26] G. H. Low and I. L. Chuang, "Hamiltonian simulation by qubitization," *Quantum*, vol. 3, p. 163, jul 2019. [Online]. Available: https://doi.org/10.22331%2Fq-2019-07-12-163

[27] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical Review Letters*, vol. 103, no. 15, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.103.150502

[28] C. Lu, M. G. Meena, and K. C. Gottiparthi, "Lugo: an enhanced quantum phase estimation implementation," *arXiv preprint arXiv:2503.15439*, 2025.

[29] E. Farhi, J. Goldstone, S. Gutmann, and L. Zhou, "The quantum approximate optimization algorithm and the sherrington-kirkpatrick model at infinite size," *Quantum*, vol. 6, p. 759, Jul. 2022. [Online]. Available: http://dx.doi.org/10.22331/q-2022-07-07-759

[30] S. Kim, V. R. Pascuzzi, Z. Xu, T. Luo, E. Lee, and I.-S. Suh, "Distributed quantum approximate optimization algorithm on a quantum-centric supercomputing architecture," 2024. [Online]. Available: https://arxiv.org/abs/2407.20212

[31] S. Kim and I.-S. Suh, "Distributed variational quantum algorithm with many-qubit for optimization challenges," *arXiv preprint arXiv:2503.00221*, 2025.

[32] A. Esposito and U.-U. Haus, "Slurm heterogeneous jobs for hybrid classical-quantum workflows," 2025. [Online]. Available: https://arxiv.org/abs/2506.03846

[33] P. Seitz, M. Geiger, C. Ufrecht, A. Plinge, C. Mutschler, D. D. Scherer, and C. B. Mendl, "Scim milq: An hpc quantum scheduler," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Sep. 2024, p. 292–298. [Online]. Available: http://dx.doi.org/10.1109/QCE60285.2024.10294

[34] A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, and T. S. Humble, "Xacc: A system-level software infrastructure for heterogeneous quantum-classical computing," 2019. [Online]. Available: https://arxiv.org/abs/1911.02452

[35] Munich Quantum Valley, "Q-dessi: Quantum design environment for software, systems, and integration," https://www.munich-quantum-valley.de/research/consortia/q-dessi, 2024, accessed: 2024-06-13.

[36] Amazon Web Services, "Amazon Braket," 2020. [Online]. Available: https://aws.amazon.com/braket/

[37] Microsoft, "Azure quantum," https://azure.microsoft.com/en-us/products/quantum, 2025, accessed: 2025-08-11.

[38] qBraid, "qbraid: Quantum computing platform," https://www.qbraid.com, 2024, accessed: 2024-06-13.

[39] A. Mahesh, S. Mittal, and F. Mueller, "CONQURE: A co-execution environment for quantum and classical resources," in *International Workshop on Integrating High-Performance and Quantum Computing*, Oct. 2025. [Online]. Available: https://arxiv.org/abs/2505.02241

[40] R. Wille, L. Berent, T. Forster, J. Kunasaikaran, K. Mato, T. Peham, N. Quetschlich, D. Rovara, A. Sander, L. Schmid, D. Schoenberger, Y. Stade, and L. Burgholzer, "The MQT handbook: A summary of design automation tools and software for quantum computing," in *IEEE International Conference on Quantum Software (QSW)*, 2024.

[41] Q-Exa Consortium, "Q-exa: Quantencomputer-demonstrationsaufbauten," https://www.quantentechnologien.de/forschung/foerderung/quantencomputer-demonstrationsaufbauten/q-exa.html, 2024, accessed: 2024-06-13.

[42] D-Wave Systems Inc., "Ocean samplers api reference," https://docs.dwavequantum.com/en/latest/ocean/api_ref_samplers/index.html, 2024, accessed: 2024-06-13.

[43] Y. Alexeev, M. H. Farag, T. L. Patti, M. E. Wolf, N. Ares *et al.*, "Artificial intelligence for quantum computing," *Nature Communications*, 2025.

[44] A. Berezutskii, M. Liu, A. Acharya, R. Ellerbrock, J. Gray, R. Haghshenas, Z. He, A. Khan, M. Pflitsch, M. Pistoia, V. Vinokur, and Y. Alexeev, "Tensor networks for quantum computing," *Nature Reviews Physics*, vol. 7, pp. 581–593, 2025.

[45] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, "EQC: Ensembled quantum computing for variational quantum algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 59–71.

[46] M. Wang, P. Das, and P. J. Nair, "Qoncord: A multi-device job scheduling framework for variational quantum algorithms," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Nov. 2024, p. 735–749. [Online]. Available: http://dx.doi.org/10.1109/MICRO61859.2024.00060

[47] E. Giortamis, F. Romão, N. Tornow *et al.*, "QOS: Quantum operating system," in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, 2025.

[48] E. Giortamis, F. Romao, N. Tornow, D. Lugovoy *et al.*, "Qonductor: A cloud orchestrator for quantum computing," in *Proceedings of the ACM Symposium on Cloud Computing*, 2025.

[49] T. H. Nguyen, "Serverless and reinforcement learning-based resource management for quantum computing," 2025, phD Thesis, University of Melbourne.

[50] N. Ma and H. Li, "Understanding and estimating the execution time of quantum circuits," *ACM Trans. Softw. Eng. Methodol.*, Nov. 2025, just Accepted. [Online]. Available: https://doi.org/10.1145/3778031

[51] S. Atchley, C. Zimmer, J. Lange, D. Bernholdt, V. Melesse Vergara, T. Beck, M. Brim, R. Budiardja, S. Chandrasekaran, M. Eisenbach, T. Evans, M. Ezell, N. Frontiere, A. Georgiadou, J. Glenski, P. Grete, S. Hamilton, J. Holmen, A. Huebl, D. Jacobson, W. Joubert, K. Mcmahon, E. Merzari, S. Moore, A. Myers, S. Nichols, S. Oral, T. Papatheodore, D. Perez, D. M. Rogers, E. Schneider, J.-L. Vay, and P. K. Yeung, "Frontier: Exploring exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3581784.3607089

[52] P. Mantha, F. J. Kiwit, N. Saurabh, S. Jha *et al.*, "Pilot-quantum: A quantum-hpc middleware for resource, workload and task management," 2024. [Online]. Available: https://arxiv.org/abs/2412.18519

[53] ——, "Hybrid quantum-hpc middleware systems for adaptive resource, workload and task management," 2026. [Online]. Available: https://arxiv.org/abs/2604.03445

[54] K. Rallis, I. Liliopoulos, G. D. Varsamis, E. Tsipas *et al.*, "Interfacing quantum computing systems with high-performance computing systems: An overview," 2025. [Online]. Available: https://arxiv.org/abs/2509.06205

[55] J. M. Murillo, J. Garcia-Alonso, E. Moguel *et al.*, "Quantum software engineering: Roadmap and challenges ahead," *ACM Transactions on Software Engineering and Methodology*, 2025.

[56] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for NISQ evaluation and simulation," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–26, 2023.

[57] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, p. 4213, 2014.

[58] E. R. Anschuetz, J. P. Olson, A. Aspuru-Guzik, and Y. Cao, "Variational quantum factoring," 2018.

[59] H. P. Patil, D. Baron, and H. Zhou, "Q-cluster: Quantum error mitigation through noise-aware unsupervised learning," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2025, pp. 849–860.

[60] A. V. Uvarov, A. S. Kardashin, and J. D. Biamonte, "Machine learning phase transitions with a quantum processor," *Physical Review A*, vol. 102, no. 1, jul 2020. [Online]. Available: https://doi.org/10.1103%2Fphysreva.102.012415

[61] A. J. McCaskey, "Tensor Network Quantum Virtual Machine (TNQVM)," 11 2016. [Online]. Available: https://www.osti.gov//servlets/purl/1340180

[62] D. Lykov, A. Chen, H. Chen, K. Keipert, Z. Zhang, T. Gibbs, and Y. Alexeev, "Performance evaluation and acceleration of the qtensor quantum circuit simulator on gpus," in *2021 IEEE International Conference on Quantum Computing and Engineering*, 2021. [Online]. Available: http://dx.doi.org/10.1109/QCS54837.2021.00007

[63] IonQ Quantum Cloud, "Jobs page," https://cloud.ionq.com/jobs, 2025, accessed: 2025-08-11.

[64] GitHub, "Github copilot," https://github.com/features/copilot, 2024.

[65] OpenAI, "Chatgpt," 2025, large language model. Available at https://chat.openai.com.

[66] Anthropic, "Claude: An ai assistant by anthropic," 2024. [Online]. Available: https://www.anthropic.com/claude