Qanela: A framework proposal for designing portable quantum assembly languages across different quantum computing platforms

Qanela: Propuesta de un marco para el diseño de lenguajes ensamblado cuánticos portátiles a través de diferentes plataformas de computación cuántica

Isaac Arturo Vargas-Chacón[1], Johan Carvajal-Godínez[2], Ernesto Rivera-Alvarado[3]

Fecha de recepción: 11 de diciembre, 2025
Fecha de aprobación: 24 de abril, 2026

Vargas-Chacón, I.A; Carvajal-Godínez, J; Rivera-Alvarado, E. Qanela: a framework proposal for designing portable quantum assembly languages across different quantum computing platforms. *Tecnología en Marcha*. Vol. 39 Nº 4. Octubre-Diciembre, 2026. Pág. 160-175.

https://doi.org/10.18845/tm.v39i4.8384

1 Instituto Tecnológico de Costa Rica, Master's in Computer Science, Costa Rica
i.vargas.1@estudiantec.cr
https://orcid.org/0009-0006-7264-2020
2 Instituto Tecnológico de Costa Rica, Electronic Engineering School, Costa Rica
johcarvajal@itcr.ac.cr
https://orcid.org/0000-0003-1489-7894
3 Instituto Tecnológico de Costa Rica, Computer Science School, Costa Rica
errivera@itcr.ac.cr
https://orcid.org/0000-0002-2756-0572

Keywords

Quantum computing; quantum assembly languages; algorithm portability; cross-platform compatibility; quantum software engineering; hardware abstraction; quantum circuit compilation

Abstract

The diversity of quantum hardware platforms creates significant algorithm portability challenges, with each platform employing distinct instruction sets, gate implementations, and physical constraints that hinder software reusability. This paper presents a framework for designing portable quantum assembly languages enabling seamless cross-platform execution. Through systematic profiling of Shor's, Grover's, and QAOA algorithms on IBM Qiskit, Microsoft Q#, and Google Cirq — 3,216 executions of identical, reference-validated logical circuits — we identify key portability challenges and derive empirically grounded design criteria from execution-pattern analysis. With matched workloads, cross-platform circuit-depth divergence is 1.3–1.6× where no gate decomposition is required and rises to 94.8× under multi-controlled decomposition pressure, of which 3.36× is genuine same-level synthesis divergence and the remainder an abstraction-level measurement gap; optimization semantics, topology expressibility, and quantum–classical execution contracts diverge analogously. The framework balances hardware abstraction with performance optimization through declared abstraction levels, topology-aware compilation, optimization profiles with defined semantics, and a universal gate set with platform-specific decomposition. Prototype validation demonstrates 100% automated compilation of unmodified single-source programs across 30 platform configurations, with all 90 executions passing calibrated distributional acceptance gates and exact statevector fidelity of 1.000000 against the language's reference semantics, providing feasibility evidence for vendor-independent quantum software development.

Palabras clave

Computación cuántica; lenguajes ensamblador cuánticos; portabilidad de algoritmos; compatibilidad entre plataformas; ingenería de software cuántico; abstracción del hardware; compilación de circuitos cuánticos.

Resumen

La diversidad de plataformas de hardware cuántico plantea desafíos significativos de portabilidad de algoritmos, ya que cada plataforma emplea conjuntos de instrucciones, implementaciones de compuertas y restricciones físicas distintas que dificultan la reutilización del software. Este trabajo presenta un marco para el diseño de lenguajes ensambladores cuánticos portables que permiten la ejecución transparente entre plataformas. Mediante la caracterización sistemática de los algoritmos de Shor, Grover y QAOA en IBM Qiskit, Microsoft Q# y Google Cirq (3216 ejecuciones de circuitos lógicos idénticos y validados contra una referencia), se identifican los principales desafíos de portabilidad y se derivan criterios de diseño con fundamento empírico a partir del análisis de patrones de ejecución. Con cargas de trabajo equivalentes, la divergencia en la profundidad de circuito entre plataformas es de 1.3–1.6× cuando no se requiere descomposición de compuertas y asciende a 94.8× bajo presión de descomposición de compuertas multicontroladas. De este factor, 3.36× corresponde a una divergencia genuina de síntesis al mismo nivel de abstracción, y el resto se debe a una brecha de medición entre niveles de abstracción. La semántica de optimización, la expresividad de la topología y los contratos de ejecución cuántico-clásicos divergen de manera análoga. El marco equilibra la abstracción del hardware con la optimización del rendimiento mediante niveles de

abstracción declarados, compilación consciente de la topología, perfiles de optimización con semántica definida y un conjunto universal de compuertas con descomposición específica para cada plataforma. La validación del prototipo demuestra la compilación automatizada del 100 % de los programas de fuente única sin modificar en 30 configuraciones de plataforma. Las 90 ejecuciones superan criterios de aceptación distribucionales calibrados y alcanzan una fidelidad exacta del vector de estado de 1.000000 respecto de la semántica de referencia del lenguaje. Esto aporta evidencia de viabilidad para el desarrollo de software cuántico independiente del proveedor.

Introduction

Quantum computing represents a paradigm shift in computational technology, leveraging quantum mechanics principles (superposition, entanglement, interference) to perform certain computations exponentially faster than classical approaches [1, 2, 3]. Applications span molecular simulation [4, 5], materials science, and optimization with potential impacts in pharmaceutical research and chemical engineering [6]. However, hardware development remains experimental with diverse qubit technologies (superconducting, trapped ions, photonic) [7, 8], creating isolated ecosystems where software for one platform cannot easily run on another [9, 10].

The portability of quantum algorithms across different quantum computing hardware has become a major challenge in the field [10]. Different qubit technologies exhibit varying physical properties that make them well-suited for implementing certain quantum gates while rendering others practically impossible [7, 11]. These hardware-dependent limitations result in platform-specific instruction sets that vary significantly between quantum computers [12], each accompanied by proprietary tools, programming frameworks, and gate implementations [13]. This isolation is magnified by hardware roadmaps driven by private companies whose strategies naturally align with business objectives rather than purely scientific priorities [14]. Consequently, quantum software developers write algorithms locked to specific platforms and vendors, hampering code reusability and scalability.

The quantum software community urgently needs methodologies to address these portability challenges [15]. This requires a paradigm shift from hardware-centric programming to software-centric design, where quantum assembly languages provide sufficient abstraction to enable algorithm portability while maintaining performance optimization opportunities—and where software can guide the direction of hardware development rather than merely follow it.

This paper addresses the fundamental question: *How can quantum algorithms be made portable across different quantum computing hardware platforms?* We propose a framework for designing quantum assembly languages that balances abstraction and performance, enabling seamless execution of quantum algorithms across diverse hardware architectures. Our framework is grounded in systematic profiling of quantum algorithm execution patterns across major platforms (IBM Qiskit, Microsoft Q#, and Google Cirq), in which every compared execution is validated for semantic equivalence against an independent reference simulator, and synthesizes design criteria that promote portability without sacrificing efficiency.

Related work

Existing quantum programming approaches span high-level frameworks (quAPL [16], OpenQL [17], Qwerty [18], Qunity [19], QHAL [10]) and assembly languages (OpenQASM [20, 21], cQASM [22], Jaqal [23, 24], MLIR quantum dialect [25]). Tools like qprof [26] enable

performance analysis. However, current approaches face fundamental tensions: frameworks prioritize either abstraction or hardware control but rarely achieve optimal balance [9], and lack empirical optimization profiles for algorithm-specific compilation strategies.

Framework Differentiation and Novel Contributions

Table 1. Framework differentiation and our contributions.

Existing approach	Key Limitation	Our Solution
QIR/MLIR	Hardware-agnostic IR lacks algorithm-specific compilation strategies	Topology-aware algorithm classification (interaction-graph density + decomposition pressure) with automatic connectivity adaptation
QHAL	Hierarchical abstraction without empirical optimization profiles	Evidence-based design criteria from 3,216 reference-validated cross-platform experimental runs
OpenQASM 3	Timing/classical primitives without topology requirements	@requires_connectivity directives with automatic platform-specific decomposition

Table 1 summarizes how our framework addresses critical limitations in existing portability approaches: QIR/MLIR [25] lacks algorithm-specific compilation strategies, QHAL [10] lacks empirical grounding in actual platform behavior, and OpenQASM 3 [21] provides no mechanism for expressing topology requirements. Our topology-aware classification, evidence-based design criteria (derived from 3,216 individually reference-validated measurements), and **@requires_connectivity** directives target these gaps directly, addressing the fundamental tension between abstraction and performance that has constrained prior portability efforts [9].

Novel Technical Contributions:

1. Empirically grounded design criteria: Design criteria derived from systematic profiling of actual algorithm executions—every execution validated for semantic equivalence against an independent reference—rather than theoretical assumptions.
2. Two-axis, IR-level algorithm classification: Automatic categorization by interaction-graph density (topology sensitivity) and multi-controlled decomposition pressure, computed on the platform-independent IR where the classification is unique, with appropriate compilation pathways.
3. Platform philosophy abstraction: Optimization profiles with *defined semantics* (educational, hardware-realistic, language-integrated), recognizing that performance differences reflect optimization philosophy and that "stronger optimization" is not a portable concept.
4. Declared-abstraction-level measurement methodology: Cross-platform structural metrics are defined only at a declared abstraction level, preventing category errors when platforms natively compile at different levels.

5. Reference-gated evaluation protocol: A per-run acceptance procedure (Monte-Carlo-calibrated distributional gates plus algorithm certificates against an independent reference simulator) ensuring that all cross-platform comparisons are made between semantically equivalent circuits.

Methodology

This research adopts a design science approach to develop a framework for portable quantum assembly languages. The methodology comprises four interconnected phases: (1) quantum algorithm profiling, (2) execution pattern analysis, (3) framework synthesis, and (4) prototype development and validation.

Quantum Algorithm Profiling

Three canonical quantum algorithms were profiled across IBM Qiskit, Microsoft Q#, and Google Cirq platforms: Shor's algorithm [1] (N=15 factorization with real controlled modular exponentiation and explicit inverse QFT), Grover's algorithm [1, 13] (n=3,5,8 qubit search), and QAOA [2] (n=6,8,12 qubit MaxCut with p=1,2 layers over two seeded random graphs per size). These algorithms represent diverse computational paradigms: period-finding with quantum Fourier transforms (Shor), amplitude amplification (Grover), and hybrid quantum-classical optimization (QAOA) [13].

Canonical instances (single source of truth). Each of the 16 instances is defined once as a canonical gate list over the universal vocabulary {h, x, rx, rz, cx, cz, cp, swap, cswap, mcz}; every platform builds its circuit from the same list, making the compared workloads identical by construction. Grover iteration counts are fixed and identical on every platform (k = 2, 4, 12 for n = 3, 5, 8, i.e., the optimal counts). QAOA graphs are connected G(n, 0.5) instances generated from recorded seeds and shared across platforms ($\beta = 0.5$, $\gamma = 1.0$; cost layer CX·RZ(2γ)·CX per edge; mixer RX(2β)). Shor (N = 15, a = 7; 12 qubits: 8 counting + 4 work) implements controlled multiplication by 7 and by 4 as explicit permutation circuits, with higher controls acting as the identity since $7^4 \equiv 1 \bmod 15$, followed by an explicit inverse QFT.

Independent reference simulator. Ground truth is established by a NumPy statevector simulator sharing no code with any backend. Self-verification before use: the QFT construction equals the DFT matrix to 3.8×10^{-15}; the Shor instance's exact distribution is four peaks 0, 64, 128, 192 at probability 0.25 each; Grover success probabilities match the closed form $\sin^2((2k+1)\theta)$ to 10^{-9}.

Platform Configurations:

- Qiskit 2.5.0 (Aer 0.17.2): 3 topologies (unrestricted; grid_5×5, 25 nodes/40 edges; linear_16, 16 nodes/15 edges) × 2 optimization levels (0, 3), basis {rz, sx, x, cx}; 30 seeded replications per configuration (seed = base + replication index).
- Cirq 1.7.0: native decomposition to ≤2-qubit operations; the same 3 topologies via RouteCQC routing × 2 study-defined optimization levels—*minimal* (no transformers) and *full* (a documented five-transformer chain: drop_negligible_operations → merge_single_qubit_gates_to_phxz → eject_z → drop_empty_moments → align_left), a design decision because the platform exposes no built-in levels. A determinism pilot (5 replications × 3 conditions) produced bit-identical outputs, fixing the replication count at 3.

- Q# 1.29.1: single configuration. The QDK exposes no coupling-map and no optimization-level interface; this absence is recorded as a finding rather than padded with placeholder measurements. 3 replications (structural determinism verified); the shot loop is internal to the generated program (a single seeded execution), which is required for correct sampling semantics (Section 4.1.5).

Acceptance procedure (applied to every run). 4,096 shots per execution. A run's measured distribution must satisfy: total-variation distance to the exact reference ≤ a Monte-Carlo-calibrated gate (mean + 6σ of the multinomial sampling TVD at 4,096 shots, estimated from 200 draws per instance), plus algorithm certificates—Grover: $|\hat{P}(\text{marked}) - P_{exact}| \leq 6\sigma$; QAOA: $|\langle \hat{C} \rangle - \langle C \rangle_{exact}| \leq 6\sigma$; Shor: all four peaks within [0.20, 0.30]. Runs failing any gate are recorded as failures and contribute no metrics. All 3,216 runs passed.

Collected Metrics: uniform ASAP circuit depth (identical algorithm applied to every platform's emitted operation list; measurements excluded), gate frequencies and counts by arity, two-qubit gate ratio, ancilla allocation, logical and physical interaction graphs, inserted routing SWAPs, compilation and execution times, and full measurement distributions.

Experimental design and measurement protocol

Our experimental design distinguishes between *deterministic platform outputs* and *stochastic platform outputs*, with the distinction established empirically rather than assumed: a pilot verified Cirq's compilation pipeline to be bit-deterministic across replications; Q# outputs are structurally deterministic; and Qiskit's transpiler is deterministic in the absence of a coupling map (depth coefficient of variation 0.000 across 30 seeds) and stochastic under one (Section 4.1.4), localizing its randomness to layout/routing.

Table 2. Experimental design matrix.

Platform	Instances	Configurations per instance	Replications	Runs
Qiskit	16	3 topologies × 2 opt. levels = 6	30 (seeded)	2,880
Cirq	16	3 topologies × 2 opt. levels = 6	3 (pilot-verified deterministic)	288
Q#	16	1 (no topology/ optimization axes)	3 (deterministic)	48
Total				3,216

The asymmetric replication in Table 2 is methodologically appropriate rather than imbalanced: deterministic phenomena require no repeated sampling, while stochastic optimization demands statistical characterization. Q#'s single configuration per instance is a measured property of the platform—there is nothing to sweep, and that absence is itself evidence (Criterion 2). Seeds are deterministic functions of the replication index and are recorded with every run.

Analysis approach:

1. **Deterministic results** (Cirq, Q#, and Qiskit without coupling maps): reported as exact values.
2. **Stochastic results** (Qiskit under coupling maps): reported as mean ± standard deviation across 30 seeded runs; variability reflects transpilation randomness, not measurement noise.
3. **Cross-platform ratios** are computed only between semantically validated runs of identical logical circuits, and structural ratios are additionally restricted to a **declared abstraction level** (Section 3.3).

Execution Pattern Analysis

We characterize how one logical algorithm unfolds into platform-specific execution patterns. Figure 1 illustrates the pipeline outcome for Grover n=3: a single canonical IR lowered into three different native realizations, all semantically validated against the reference.

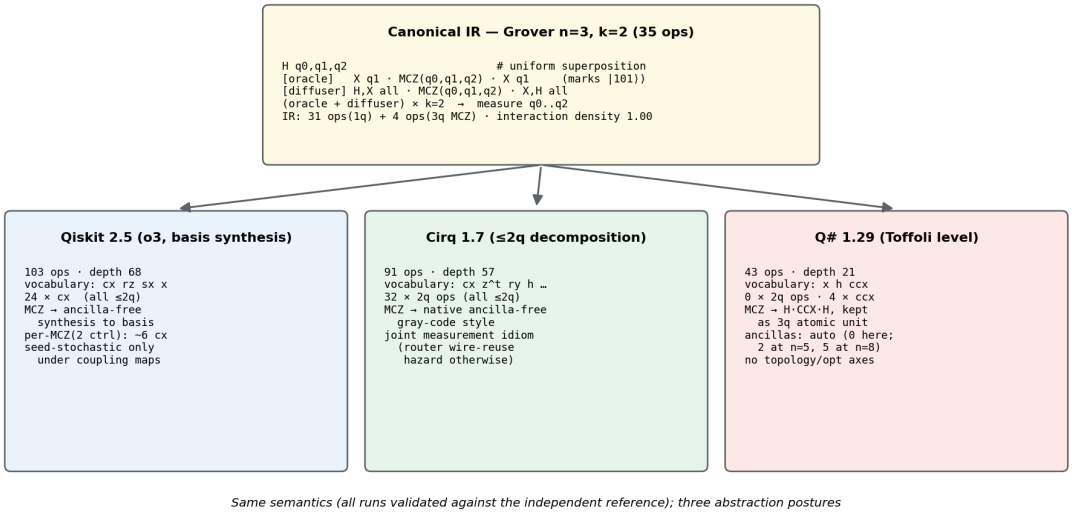


Figure 1. Execution pattern example (Grover's algorithm, n=3, k=2). The canonical IR (top) is identical for all platforms. Qiskit synthesizes the multi-controlled-Z into its hardware basis (103 ops, all ≤2-qubit); Cirq decomposes it ancilla-free to ≤2-qubit gates (91 ops); Q# realizes it at the Toffoli level (43 ops, zero 2-qubit gates), auto-allocating ancillas at larger sizes. All three preserve the algorithm's semantics.

The three lowerings expose the study's central structural fact: the platforms natively compile at **three different abstraction levels**—hardware-basis synthesis (Qiskit), ≤2-qubit decomposition (Cirq), and Toffoli-level realization with automatic ancilla allocation (Q#). Structural metrics are therefore comparable only at a declared level; the level of record for cross-platform ratios is the ≤2-qubit level, with Q#'s metrics reported at its native level under an explicit tag.

Vocabulary divergence. Aggregating emitted gate vocabularies across the campaign, pairwise Jaccard overlaps are given in Table 3: no pair of platforms speaks a similar dialect. The native gate-frequency fingerprints make each posture concrete: Qiskit emits its hardware basis {cx, rz, sx, x}; Cirq a ≤2-qubit vocabulary including PhasedXZ and named SWAPs; Q# emits {x, h, ccx} for Grover—zero two-qubit gates, a direct signature of Toffoli-level compilation.

Table 3. Emitted gate-vocabulary overlap (Jaccard) across the full campaign.

Platform	Qiskit	Cirq	Q#
Qiskit	1.00	0.12	0.38
Cirq	0.12	1.00	0.31
Q#	0.38	0.31	1.00

Framework Synthesis

Based on insights from algorithm profiling and pattern analysis, we synthesized design criteria for portable quantum assembly languages through a systematic evidence-to-criteria mapping: performance variance decomposition led to multi-level abstraction support with declared measurement levels; topology expressibility analysis informed topology-aware compilation requirements; gate set heterogeneity motivated universal gate set specifications with platform-owned decomposition; optimization philosophy divergence necessitated optimization profiles with defined semantics; IR-level classification findings motivated two-axis classification and routing; and quantum–classical boundary hazards motivated standardized hybrid execution primitives. Each criterion is presented with its supporting evidence in Section 4, and the criteria were formalized into a framework specification defining syntax, semantics, and compilation strategies.

Results and Discussion

Algorithm Profiling Results

Profiling identical, reference-validated logical circuits across the three platforms revealed large, *structured* variations in execution cost.

Cross-Platform Performance Analysis

Central finding: depth divergence is a decomposition-posture phenomenon (1.3× to 94.8×; Table 4).

Table 4. Circuit depth by platform (unrestricted topology, strongest optimization). Every platform executes the same canonical gate list with identical Grover iteration counts, so all ratios measure compilation, decomposition, and routing infrastructure—never differing amounts of algorithmic logic.

Instance	Qiskit (mean ± sd, 30 seeds)	Cirq	Q# (native level)	max/min
grover-n3 (k=2)	68 ± 0	57	21	3.2×
grover-n5 (k=4)	604 ± 0	1,185	49	24.2×
grover-n8 (k=12)	6,797 ± 0	22,849	241	94.8×
qaoa-n6 (4 instances)	23–52	23–56	17–42	1.3–1.4×
qaoa-n8 (4 instances)	35–67	39–76	29–57	1.3×
qaoa-n12 (4 instances)	56–94	67–112	50–84	1.3×
shor-N15-a7	108 ± 0	102	69	1.6×

Where the canonical circuit body is already native 1–2-qubit (QAOA), the platforms nearly agree (1.3–1.6×)—a negative control demonstrating that the large divergences elsewhere are structure, not platform noise. Divergence concentrates exactly where multi-controlled decomposition pressure exists and scales with the control count of Grover's MCZ (3.2× → 24.2× → 94.8×).

Decomposing the 94.8×. The headline ratio has three separable components:

1. Same-level synthesis divergence (Qiskit vs Cirq, both emitting exclusively ≤2-qubit gates): up to 3.36× at grover-n8, with workload-dependent direction—Cirq is *shallower* at grover-n3 (0.84×) and shor (0.94×). Neither ancilla-free synthesis strategy dominates. Per multi-controlled-Z with 7 controls, Qiskit synthesizes ≈180 CX and Cirq ≈548 CX, both without ancillas.
2. Abstraction-level gap (the remainder): Q#'s Grover trace contains zero two-qubit gates—the QDK realizes the MCZ at the Toffoli level (11 CCX per 7-control MCZ) and auto-allocates ancillas (0, 2, 5 for n = 3, 5, 8; 264 >2-qubit operations at n=8 over 13 traced qubits). Its depth of 241 is measured in a different unit than Qiskit's 6,797; treating the resulting ratio as an efficiency comparison is a category error, not a noisy measurement. This component motivates the declared-level measurement rule and Criterion 1.
3. Near-parity without pressure (1.3×): the baseline against which the above are interpreted.

Topology Sensitivity

Table 5. Depth inflation under connectivity constraints (linear_16 / unrestricted, strongest optimization)

Instance class	Qiskit	Cirq	Q#
grover (n=3/5/8)	1.57× / 1.48× / 2.29×	1.00× / 1.07× / 1.20×	no topology interface
qaoa-n6 (4 inst.)	1.56–2.16×	1.20–1.43×	no topology interface
qaoa-n8 (4 inst.)	1.84–2.34×	1.28–1.45×	no topology interface
qaoa-n12 (4 inst.)	2.41–3.20×	1.41–1.54×	no topology interface
shor-N15-a7	1.88×	1.35×	no topology interface
Median	2.13×	1.38×	—

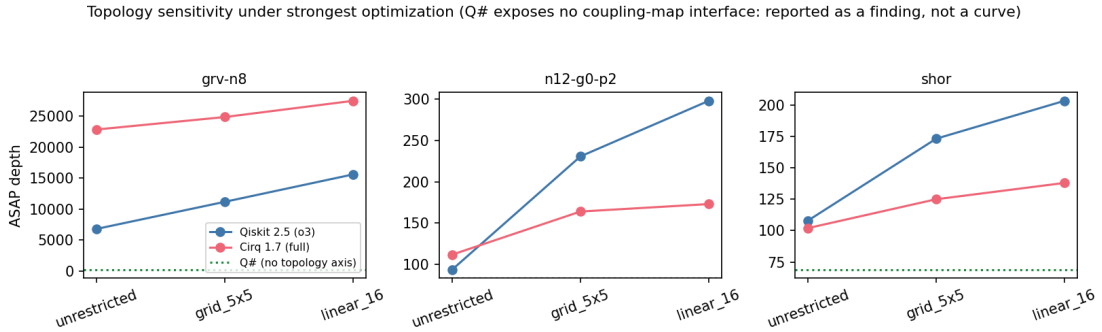


Figure 2. Topology sensitivity under strongest optimization for three representative instances. Q# exposes no coupling-map interface; its value is a level-tagged constant, not a curve.

Two findings deserve emphasis (Table 5, Figure 2). First, one major platform cannot receive topology intent at all: across all 16 instances, the QDK exposes no coupling-map abstraction, so a portable language must carry the directive—per-platform flags cannot. Second, routing-cost *observability* itself diverges: for qaoa-n12-g0-p2, Cirq reports 144 explicitly named inserted

SWAPs on linear_16 (two-qubit ops 140 → 284), while Qiskit at optimization level 3 absorbs routing into basis-gate inflation (CX 140 → 431, zero named SWAPs). A normalized routing-cost report is therefore part of the framework's compilation contract.

Classification must run on the IR. Platform-emitted two-qubit gate ratios are fingerprints of the compiler, not the algorithm: for the same Grover instance the emitted ratio is 0.37 (Qiskit), 0.35 (Cirq), and 0.00 (Q#). Even on the IR, naive arity counting fails (Grover's IR contains no arity-2 gates at all; its MCZ touches all qubits). The portable invariant is the interaction-graph density of the IR—which pairs of qubits must ever interact, with k-qubit operations contributing k-cliques (Table 6).

Table 6. IR-level connectivity demand (platform-independent).

Algorithm	Interaction-graph density	Multi-controlled arity (decomposition pressure)
Grover (all n)	1.00 (every pair)	n (high)
Shor N=15	0.62	3 (moderate: cswap)
QAOA (n=8 example)	0.57	— (none)

Measured correctly, Grover demands the *highest* connectivity—while simultaneously being the most platform-divergent workload due to decomposition pressure. The two axes (topology sensitivity; decomposition pressure) dissociate empirically—QAOA n=12 tops topology inflation (up to 3.20× on Qiskit) with near-parity across platforms, while Grover n=8 tops cross-platform divergence with mild topology inflation on Cirq (1.20×)—and both are computed on the IR in the framework's classification stage.

Optimization Profiles Are Not Portable

Icc Instance & Qiskit o0 → o3 & Cirq minimal → full (study-defined chain) grover n=3/5/8 & +30 qaoa (12 instances) & +9 shor-N15-a7 & +45

"Stronger optimization" is not a portable concept (Table 7): Qiskit's built-in levels behave as expected (+11).

Transpiler Stochasticity Is Localized to Routing

Table 8. Qiskit depth coefficient of variation across 30 seeds (optimization level 3).

Topology	CV median	CV max
unrestricted	0.000	0.000
grid_5×5	0.080	0.139
linear_16	0.041	0.068

Without a coupling map, Qiskit's transpiler is bit-deterministic; under one, layout/routing sampling introduces up to 13.9% depth variation across seeds (Table 8). This within-platform spread is itself a portability datum: reproducible cross-platform claims require seed policies and replication, which the framework's execution contract mandates. Cirq's pipeline and Q# are bit-deterministic across replications (96/96 and 16/16 cells).

Quantum–Classical Boundary Hazards

Two reproducible portability hazards were discovered at the measurement/shot layer, precisely because every run was gated on the reference:

1. Cirq — measurement floating and wire reuse. Per-qubit measurements float in the routing DAG to each wire's last dependency; **RouteCQC** then routes later SWAPs *through already-measured wires* (observed: two measurement keys on one physical qubit), breaking terminal-state sampling although recorded results remain correct. The required idiom is a single joint measurement over the measured register, which depends on every wire and cannot float (validated terminal in 96/96 configurations).
2. Q# — per-shot re-seeding. Host-side **set_quantum_seed** combined with host-driven shots re-seeds every shot, collapsing 4,096 shots into one repeated sample (diagnostic signature: Grover's TVD equal to exactly 1 - P(marked)). The conforming lowering places the shot loop *inside* the generated program as a single seeded execution.

Together with three incompatible result representations (classical bits + counts, keyed arrays, **Result[]**), these hazards establish measurement placement, shot semantics, and seed contracts as first-class portability surfaces (Criterion 7).

Semantic Validation and Runtime

All 3,216 runs passed the acceptance procedure (Figure 3; TVD medians: Qiskit 0.0659, Cirq 0.0656, Q# 0.0721; upper tails are dominated by the QAOA n=12 sampling floor—4,096 outcomes at 4,096 shots—and remain below their calibrated gates). Shor's four-peak certificate passed on every platform × topology × level combination.

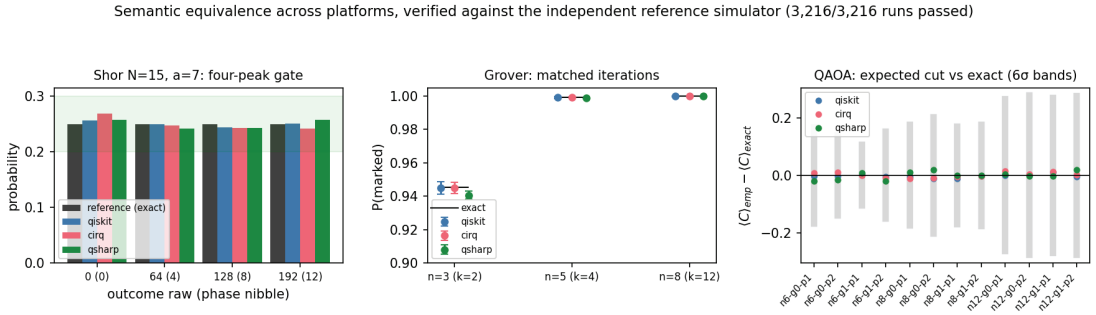


Figure 3. Semantic equivalence across platforms verified against the independent reference: Shor's four peaks within the acceptance window on all platforms; Grover P(marked) at the closed-form values; QAOA expected cut within 6σ bands. 3,216/3,216 runs passed.

Median per-run compile/execute times (strongest constrained configuration): Qiskit 15 ms / 12 ms; Cirq 37 ms / 20 ms; Q# 137 ms / 2.0 s (maximum 46.2 s at qaoa-n12; the in-program shot loop trades throughput for reproducible sampling semantics).

Framework Design Results

Based on the empirical findings, we developed seven evidence-based design criteria for portable quantum assembly languages.

Design Criteria Summary

Framework Architecture

Our framework implements a multi-stage compilation pipeline (Figure 4).

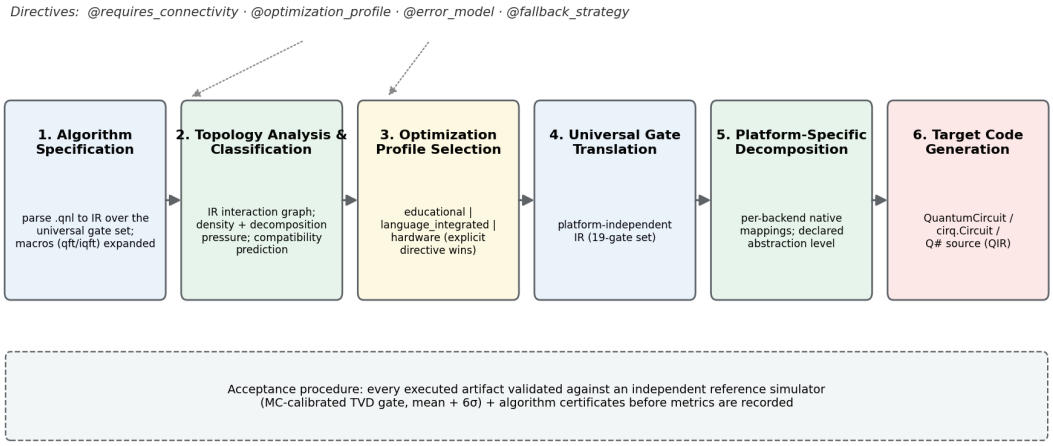


Figure 4. Framework architecture: from algorithm specification to hardware-specific code generation, with the reference-gated acceptance procedure as the evaluation backbone.

Framework Formalization

Interoperability and Backend Targets Portable assembly programs can be transpiled into OpenQASM 3 (for IBM Qiskit and other OpenQASM-compatible platforms) or Quantum Intermediate Representation (QIR) backends (for Microsoft Azure Quantum and LLVM-based toolchains), ensuring execution on a wide range of quantum hardware without vendor lock-in while preserving platform-specific optimizations through the directive system.

Semantic Definitions We define the following metrics for framework evaluation:

- Compilation Success (custom quantum-specific metric): binary; success indicates the algorithm compiles without topology violations, resource conflicts, or syntax errors.
- Performance Overhead (standard metric [27]): relative increase in execution cost versus a native, idiomatic, hand-written implementation under identical settings, $(c_{framework} - c_{native})/c_{native} \times 100\%$. Where framework lowering and the native baseline share identical gate sequences by construction, overhead is reported as lossless lowering (parity by construction); measured overhead is reported only for genuinely independent baselines.
- Total Efficiency Ratio (*novel contribution*): composite metric combining circuit depth and gate count ratios, $(depth_{platform}/depth_{baseline}) \times (gates_{platform}/gates_{baseline})$, computed only at a declared abstraction level.
- Statistical Fidelity (calibrated acceptance procedure): every executed artifact is validated against the exact reference distribution of the IR computed by an independent reference simulator—TVD ≤ a Monte-Carlo-calibrated sampling gate (mean + 6σ at the declared shot count) plus algorithm certificates—with Hellinger fidelity [28] reported alongside. Reported fidelities are never clipped or renormalized; values exceeding 1.0 are non-conforming by definition.

Syntax Extension Specification

Listing 1 presents three key syntax features derived from the empirical findings: (1) **@algorithm** directives specify topology requirements and fallback strategies (addressing depth inflation of up to 3.2× under linear connectivity and the complete absence of a topology interface on one platform); (2) **UNIVERSAL_GATE** definitions provide platform-agnostic gate abstractions with explicit native mappings (resolving the Jaccard 0.12–0.38 vocabulary disjointness and up to 3.36× same-level synthesis divergence); and (3) **@optimization_profile** declarations select compilation strategies with defined semantics, in light of the opposite-signed optimization effects observed across platforms.

Listing 1. Topology awareness, universal gates, and optimization profiles

```
@algorithm Shor {
  @requires_connectivity: all_to_all
  @optimization_target: depth_minimal
  @fallback_strategy: decompose_qft
}
// Universal gate abstraction with platform mapping
UNIVERSAL_GATE cnot(q1, q2) {
  qiskit: cx q1, q2;      // Maps to CX gate
  cirq: CNOT(q1, q2);     // Maps to CNOT operation
  qsharp: CNOT(q1, q2);   // Maps to controlled operation
}
// Configurable optimization profiles (semantics-defined)
@optimization_profile educational {
  preserve: circuit_structure    // no rebasing
  target: simulation_clarity
}
@optimization_profile hardware {
  minimize: depth_at_declared_level
  target: device_fidelity
  consider: topology_constraints
}
```

Prototype Validation Study

We conducted a prototype validation study to assess the feasibility of the framework approach using the same algorithm portfolio as the profiling study, compiled from unmodified single-source portable assembly programs.

Compilation and adaptation. All 30 platform configurations (spanning the three platforms, the topology and optimization axes where each platform exposes them, and the full instance portfolio) compiled from the same unmodified source with 100% automated adaptation—no per-platform edits of any kind.

Semantic preservation. All 90 framework-compiled executions passed the calibrated distributional acceptance gates of Section 3. In addition, exact statevector comparison between framework-compiled artifacts and the language's reference semantics yielded fidelity 1.000000 across all

30 artifacts (maximum deviation $\approx 5.8 \times 10^{-13}$, at numerical precision), demonstrating that the abstraction layer—topology-aware routing, universal gate decomposition, and platform-specific lowering—preserves quantum circuit semantics exactly.

Overhead disclosure. For the majority of cases, framework lowering produces gate sequences identical to the idiomatic native construction; per the semantic definitions, these are reported as lossless lowering (parity by construction). Measured overhead is reported only where the native baseline is genuinely independent, and sampling-floor effects (e.g., QAOA n=12, whose ideal-sampler fidelity ceiling was measured empirically with bootstrap confidence intervals) are accounted for by the calibrated gates rather than by renormalizing fidelity values.

Topology prediction. The compile-time compatibility predictor (interaction-graph embedding against the target coupling graph) correctly identified every directly embeddable case and every case requiring a fallback strategy across the validation portfolio.

Threats to Validity

Internal validity. Strengths: identical-by-construction workloads; per-run semantic gating against an independent reference (3,216/3,216 passed); empirically established determinism/stochasticity with seed policies; uniform depth metric and declared-level rule. Limitations: simulation-only environment; limited algorithm coverage (Grover/QAOA/Shor); platform-version-specific results (Qiskit 2.5.0, qiskit-aer 0.17.2, Cirq 1.7.0, Q# 1.29.1; Python 3.12.3). The simulation-only setting is deliberate: ideal, noiseless simulators isolate the software impact of each platform's compilation pipeline by removing the stochastic noise of physical hardware, so the measured divergence reflects the pure overhead of compilation posture and framework lowering, and they make exact per-run semantic gating against the reference possible; the cost is that hardware noise is not modeled, Criterion 6 is not exercised empirically, and validation on physical QPUs remains future work.

External validity. Small-scale circuits (3–13 qubits including ancillas) may not generalize to larger quantum systems, and the limited topology diversity (linear/grid) may not represent the full hardware connectivity spectrum. Cirq's *full* level is one documented transformer composition, not the platform's only or best possibility—conclusions concern the portability of the optimization *concept*, not any platform's achievable optimum.

Construct validity. Circuit-level metrics (depth, gate count) may not capture non-uniform gate execution times on real hardware, and simulator-based distributional validation does not reflect decoherence effects in physical devices. Cross-level structural comparisons are avoided by construction (declared-level rule) rather than corrected post hoc.

Discussion

The profiling campaign shows that cross-platform performance differences are structured by compilation posture rather than hardware limits: platforms nearly agree where no decomposition is required, diverge by measurable synthesis strategy at a common level, and become incomparable across abstraction levels—with topology expressibility, optimization semantics, and quantum–classical execution contracts diverging analogously. The framework operationalizes these findings—declared abstraction levels, language-level topology directives, semantics-defined optimization profiles, IR-level two-axis classification, and normative execution contracts—addressing the "peeling off vs. piling on" debate [9] through selective abstraction traversal, and prototype validation confirms feasibility (100% single-source compilation; 90/90 gated executions; fidelity 1.000000).

Conclusion

This research demonstrates that portable quantum assembly languages are both feasible and practical, offering a pathway toward vendor-independent quantum software development. Our systematic profiling of 3,216 reference-validated executions revealed that cross-platform performance differences are large but causally structured: with identical logical circuits and matched iteration counts, depth divergence is 1.3–1.6× where no gate decomposition is required, rises to 3.36× from synthesis strategy alone at a common ≤2-qubit level, and reaches an apparent 94.8× only when measurements at different abstraction levels are mixed—a category error the framework prevents by requiring declared measurement levels. These findings support our central thesis that current performance differences reflect design philosophy rather than fundamental hardware limitations, and that quantum software development should guide hardware evolution rather than being constrained by it.

The seven design criteria derived from this empirical analysis provide a comprehensive foundation for developing portable quantum assembly languages. In the prototype evaluation on simulators for three canonical algorithms across Qiskit, Cirq, and Q#, the resulting framework achieves 100% automated compilation of unmodified single-source programs across 30 configurations, with all 90 executions passing calibrated distributional gates and exact statevector fidelity of 1.000000 against the language's reference semantics. The framework's ability to classify algorithms on the platform-independent IR, detect topology requirements, select semantics-defined optimization profiles, and generate platform-specific implementations while preserving algorithmic semantics exactly—together with its normative quantum–classical execution contracts—addresses the most significant, empirically demonstrated barriers to quantum algorithm portability.

Future research directions include empirical noise-model comparison to exercise the error-model abstraction criterion, extending the framework to additional quantum computing paradigms, deepening integration with quantum error correction protocols, developing automated optimization strategies that leverage platform-specific features, and conducting large-scale evaluations across diverse algorithms and emerging hardware platforms. Community engagement and standardization efforts will be essential to achieving widespread adoption.

Data and Code Availability

All experimental data, framework implementation, circuit code, analysis scripts, and reproducibility documentation—including the canonical instance definitions, the independent reference simulator, the complete 3,216-run dataset with per-run validation records, configuration matrices with recorded seeds, and the framework compiler toolchain with platform adapters—are available under MIT License at https://gitlab.com/isaacvargas_TEC_CR/qanela. Dataset DOI: to be assigned.

References

[1] M. Ying, *Foundations of Quantum Programming*, 2nd ed. Morgan Kaufmann, 2024.

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

[3] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, "Simulated quantum computation of molecular energies," *Science*, vol. 309, no. 5741, pp. 1704–1707, 2005.

[4] M. Reiher *et al.*, "Elucidating reaction mechanisms on quantum computers," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 114, no. 29, pp. 7555–7560, 2017.

[5] A. Perdomo-Ortiz, N. Dickson, M. Drew-Brook, G. Rose, and A. Aspuru-Guzik, "Finding low-energy conformations of lattice protein models by quantum annealing," *Sci. Rep.*, vol. 2, p. 571, 2012.

[6] Y. Cao *et al.*, "Quantum chemistry in the age of quantum computing," *Chem. Rev.*, vol. 119, no. 19, pp. 10856–10915, 2019.

[7] M. Kjaergaard *et al.*, "Superconducting qubits: Current state of play," *Annu. Rev. Condens. Matter Phys.*, vol. 11, pp. 369–395, 2020.

[8] T. Rudolph, "Why I am optimistic about the silicon-photonic route to quantum computing," *APL Photonics*, vol. 2, no. 3, p. 030901, 2017.

[9] M. Schönberger, M. Franz, S. Scherzinger, and W. Mauerer, "Peel | pile? Cross-framework portability of quantum software," *arXiv preprint arXiv:2203.06289*, 2022.

[10] K. M. Barnes *et al.*, "Optimising the quantum/classical interface for efficiency and portability with a multi-level hardware abstraction layer for quantum computers," *EPJ Quantum Technol.*, vol. 10, p. 36, 2023.

[11] J. I. Cirac and P. Zoller, "Goals and opportunities in quantum simulation," *Nat. Phys.*, vol. 8, pp. 264–266, 2012.

[12] S. Lopez Alarcón, E. Wong, T. S. Humble, and E. Dumitrescu, "Quantum programming paradigms and description languages," *Computing in Science & Engineering*, vol. 25, no. 6, pp. 33–38, 2023.

[13] R. Lipton and K. Regan, *Introduction to Quantum Algorithms via Linear Algebra*, 2nd ed. MIT Press, 2021.

[14] D. Castelvecchi, "Quantum computers ready to leap out of the lab in 2017," *Nature*, vol. 541, no. 7635, pp. 9–10, 2017.

[15] J. Zhao, "Quantum software engineering: Landscapes and horizons," *arXiv preprint arXiv:2007.07047*, 2021.

[16] S. Núñez-Corrales, M. Frenkel, and B. Abreu, "quAPL: Modeling quantum computation in an array programming language," in *Proc. IEEE Int. Conf. Quantum Computing and Engineering (QCE)*, 2023, pp. 1001–1012.

[17] N. Khammassi *et al.*, "OpenQL: A portable quantum programming framework for quantum accelerators," *ACM Trans. Quantum Comput.*, vol. 18, no. 1, Art. 13, 2021.

[18] A. J. Adams, S. Khan, J. S. Young, and T. M. Conte, "Qwerty: A basis-oriented quantum programming language," *arXiv preprint arXiv:2404.12603*, 2024.

[19] F. Voichick, L. Li, R. Rand, and M. Hicks, "Qunity: A unified language for quantum and classical computing," *Proc. ACM Program. Lang.*, vol. 7, POPL, Art. 32, 2023.

[20] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017.

[21] A. Cross *et al.*, "OpenQASM 3: A broader and deeper quantum assembly language," *ACM Trans. Quantum Comput.*, vol. 3, no. 3, pp. 1–50, 2022.

[22] C. Fu, J. E. G. Smith, and Y. Tomita, "cQASM v1.0: Towards a common quantum assembly language," *arXiv preprint arXiv:1805.09607*, 2020.

[23] A. J. Landahl *et al.*, "Jaqal: The quantum assembly language for QSCOUT," Sandia National Laboratories Technical Report, 2020.

[24] D. S. Lobser *et al.*, "Transpiling quantum assembly language circuits to a qudit form," *arXiv preprint arXiv:2409.18963*, 2020.

[25] A. Karamlou *et al.*, "A MLIR dialect for quantum assembly languages," *arXiv preprint arXiv:2101.11365*, 2020.

[26] A. Suau, G. Staffelbach, and A. Todri-Sanial, "qprof: A gprof-inspired quantum profiler," *ACM Trans. Quantum Comput.*, vol. 4, no. 1, pp. 1–28, 2022.

[27] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.

[28] M. S. Nikulin, "Hellinger distance," in *Encyclopedia of Mathematics*, M. Hazewinkel, Ed. Springer, 2001.

Declaración sobre uso de Inteligencia Artificial (IA)

Los autores declaramos haber utilizado Claude Fable 5 para múltiples tareas: el apoyo en la estructura y redacción y el procesamiento y análisis de datos. Todos los resultados e información asistidos por IA fueron revisados, verificados y validados minuciosamente por los autores, quienes asumen plena responsabilidad por la precisión, coherencia e integridad del contenido publicado.