

Limbo: A Multi-Chip-First Quantum Compiler Unifying Hypergraph Partitioning with Topological Cross-Chip Routing

Jaymin Ding

Abstract—We present Limbo, a quantum multi-chip-first compiler that combines graph-partitioning and routing-first approaches to compilation. On a 24-cell benchmark against TKET, Limbo outperforms in every cell. On the hardest cell, it reduces cross-chip 2-qubit operations by 78% and compile time by a factor of 2735; the median compile-time ratio across the matrix is 119. Under a calibrated noise model whose intra-chip and cross-chip error rates draw from published experimental characterizations, Limbo’s output reaches higher Hellinger fidelity than TKET on every informative cell. The pipeline first analyzes the structural topology of the input interaction graph and selects a matching cloud vendor. It then partitions the qubits across chips using multi-level hypergraph partitioning, sequences the residual cross-chip operations through an event scheduler driven by topological sorting, and transpiles each per-chip sub-circuit against the chosen device’s coupling map and native gate set. In testing, we executed the pipeline end-to-end on Rigetti’s Cepheus superconducting QPU and exercised the noise-aware routing path against the device’s live qubit calibration data. Direct hardware validation of the multi-chip architectural claim itself remains, at this moment, gated on publicly cloud-accessible multi-chip quantum hardware.

Code and project: The open-source SDK is available at <https://github.com/Jminding/Limbo-Quantum>, with the compiler API (planned to be released in Fall 2026), project information, and the full benchmark matrix at limbosys.dev.

Index Terms—Quantum computing, quantum compilers, multi-chip quantum architectures, hypergraph partitioning, distributed quantum computing, hardware-aware transpilation.

I. INTRODUCTION

SCALABLE fault-tolerant quantum systems are unlikely to arrive on single, monolithic chip architectures. Many leading hardware vendors’ public roadmaps mention scaling out through *modularity*: networks of smaller QPUs stitched together by photonic, ion-shuttling, or microwave-cable interconnects [1], [2], [3]. On such modular hardware, a logical 2-qubit gate between qubits on different chips differs qualitatively from one between adjacent qubits on the same chip. The former requires an entanglement-distribution protocol over a lossy interconnect, and its effective error rate is typically an order of magnitude or more worse than an intra-chip native gate.

Most existing quantum compilers were not designed with this distinction at the center of their abstractions. The two dominant families of compiler techniques both treat the device

topology as a single coupling map and optimize the same metric—the number of SWAP gates inserted—on every edge of that map. *Routing-first* toolchains such as TKET [4], which applies its own architecture-aware routing heuristic, and QISKIT [5], which uses SABRE [6], apply heuristic-search routing algorithms to insert SWAPs against a coupling graph that, when given a multi-chip architecture, treats inter-chip links as just another, slightly more constrained edge class. While they are able to find a good solution, they require extensive compile time when conducting a search, making them somewhat inefficient. *Graph-partitioning* approaches such as QUPORT [7] explicitly minimize the cut weight between logical-qubit groups when assigning qubits to chips, but are not capable of hardware-aware transpilation, leaving the downstream intra-chip routing problem to other tools.

We present **Limbo**, a tool that unifies graph partitioning and physical routing into a single hardware-aware compilation pipeline designed for multi-chip targets. The pipeline begins by analyzing the structural properties of the input interaction graph and selecting a matching cloud vendor using the following decision process: trapped-ion all-to-all hardware for dense circuits, superconducting lattices for nearest-neighbor circuits, and modular targets for clusterable interaction graphs. The partitioning phase then maps logical registers onto physical chip boundaries with a multi-level hypergraph partitioner whose objective is the count of cross-chip 2-qubit operations rather than total SWAP overhead. Intra-chip routing proceeds independently per chip using a standard SABRE-style heuristic search. The cross-chip operations left over from partitioning go through a separate event scheduler whose priority order is derived from a topological sort of the gate-dependency directed acyclic graph (DAG).

Real multi-QPU systems have strictly limited interconnect bandwidth. If two cross-chip events compete for the same chip-to-chip link in the same cycle, the pipeline stalls. The scheduler resolves this contention directly: each event waits its turn, with priority given to operations on the longest remaining path through the DAG.

This paper makes three main contributions.

(1) We characterize the design principle behind multi-chip quantum compilation as a structural choice: partitioning must precede routing, not run alongside it. We show how Limbo’s realization of this principle differs from both routing-first and graph-partitioning prior work. To our knowledge, Limbo is the first open, multi-framework, multi-vendor pipeline to integrate

J. Ding is an undergraduate student at Princeton University, Princeton, NJ 08544, USA. E-mail: jaymin@princeton.edu.

Manuscript prepared May 2026.

Mt-KaHyPar KM-1 partitioning with a critical-path topological cross-chip scheduler and live cloud topology querying in a single hardware-aware system.

(2) We introduce a cross-chip event scheduler that orders the operations left over from partitioning by a topological sort of the gate-dependency DAG, subject to a finite chip-to-chip link port capacity and a critical-path priority. This scheduler is the multi-chip analogue of intra-chip SWAP routing. Where SABRE chooses how to physically connect two distant qubits on one chip, Limbo’s scheduler chooses *when* each unavoidable cross-chip event executes given the link’s bandwidth constraint.

(3) We demonstrate that Limbo’s architectural advantage holds across a broad workload matrix in simulation. Against TKET on the multi-chip subset of our matrix, Limbo wins on cross-chip 2-qubit operation count in every measured cell, and the win survives two different intra-chip topologies (fully-connected and constrained 2-D grid). Under a calibrated multi-chip noise model, Limbo’s output also wins on Hellinger fidelity in every informative cell.

The remainder of the paper is organized as follows. Section II surveys related work. Section III presents Limbo’s architecture as a two-component SDK plus backend-API system. Section IV describes the four-phase compilation pipeline. Section V covers topology-aware vendor selection and noise-aware routing. Section VI presents the benchmark evaluation. Section VII discusses limitations and future work. Section VIII concludes.

II. RELATED WORK

A. Routing-First Compilers

The two most widely-deployed open-source quantum compilers, TKET [4] and QISKIT [5], both adopt a routing-first design: given a circuit and a device coupling map, the compiler inserts SWAP gates so that every 2-qubit operation in the output acts between physically adjacent qubits. The dominant algorithm is SABRE [6], a heuristic-search router that simultaneously chooses an initial qubit layout and a SWAP-insertion sequence by optimizing a lookahead cost function over the gate dependency DAG. Optimization-level controls (e.g., `optimization_level=3` in QISKIT) chain SABRE with multiple peephole and commutative cancellation passes, producing a hardware-runnable circuit that minimizes gate count and depth on the routed output.

These tools generalize to multi-chip targets only by treating the combined inter-chip and intra-chip graph as a single coupling map. This loses the structural distinction that matters most on modular hardware: a SWAP across an inter-chip link is one or more orders of magnitude noisier than a SWAP within a chip. The router will still faithfully insert SWAPs to satisfy the coupling-map constraint, but nothing in its objective function pushes it to avoid the lossy edges in proportion to their cost.

B. Graph-Partitioning Approaches

A separate line of work treats the logical-qubit-to-chip assignment as a graph partitioning problem. The interaction

graph of a circuit (a weighted multigraph whose vertices are logical qubits and whose edges are 2-qubit gates) is partitioned to minimize cut weight, where each cut edge corresponds to a cross-chip 2-qubit operation that must be implemented via a remote-gate protocol [8]. QUPORT [7], among others, applies a topology-, port-, and congestion-aware partitioner (TPCCAP) with simulated-annealing refinement to this problem and pairs it with a layered makespan estimator. These approaches correctly identify that minimizing cross-chip operations is the right primary objective on modular hardware, but they typically stop at the partition decision and leave the per-chip transpilation problem to downstream tooling.

Limbo takes the next step. The partitioner’s output feeds directly into a per-chip transpiler, with a scheduler in between that orders the residual cross-chip operations by topological sort of the gate DAG. The scheduler respects the chip-to-chip link’s finite port capacity.

C. Hypergraph Partitioning in Compilation Contexts

Multi-level hypergraph partitioners are a mature combinatorial tool [9], [10]. Their use in quantum compilation has so far been limited to research prototypes targeting specific architectures or specific workloads (e.g., partitioning surface-code patches across modular processors). Limbo is, to our knowledge, the first production-grade compiler to use hypergraph partitioning as the central abstraction across heterogeneous workload families, multiple cloud vendors, and live device topology querying in one end-to-end pipeline.

D. Distributed Quantum Compilation

A growing literature on distributed quantum compilation addresses related questions from a more theoretical angle, often analyzing the asymptotic cost of teleportation-based gate execution across networked QPUs [11], [12]. Most of this work has focused on protocol design (e.g., entanglement-swapping chains, code teleportation) rather than on production compilers targeting commercial cloud-accessible hardware. Limbo complements this line by providing a deployable system that drives real cloud APIs (IBM Quantum, AWS Braket, Azure Quantum) while treating multi-chip structure as a first-class concern throughout the pipeline.

III. SYSTEM ARCHITECTURE

Limbo is structured as a two-component system: a lightweight client-side SDK that user code interacts with directly, and a backend compilation service that performs the heavy algorithmic work. The components communicate over a small REST contract.

A. Client SDK

The client SDK exposes a minimal surface for circuit submission and result retrieval. Its responsibilities are deliberately narrow:

- Ingest user circuits in multiple source frameworks (QISKIT, CIRQ, OpenQASM-3) and normalize them to a single intermediate representation before transmission.

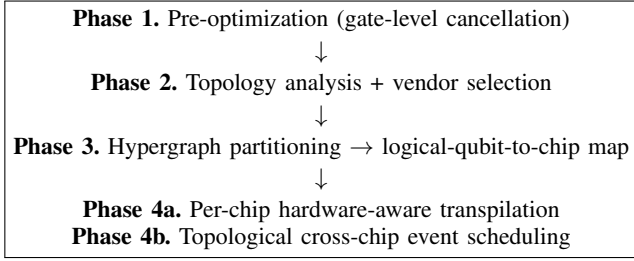


Fig. 1. Limbo’s four-phase compilation pipeline. Phases 4a and 4b are interleaved: each per-chip sub-circuit is independently transpiled against its target device’s coupling map, while cross-chip events are assembled into a critical-path schedule.

- Apply local static checks (capacity audits, hotspot detection) that catch structurally un-routable layouts *before* a paid cloud round-trip is issued.
- Maintain a *template cache* that recognizes when a parametric circuit has been compiled before and only the bound parameter values have changed. In the cached case, the heavy compilation work is skipped and only the bound circuit is transmitted. This is the foundation of Limbo’s variational fast-path (Section V-C).
- Authenticate against the backend and present cloud-vendor credentials to it (the SDK does not persist user vendor credentials server-side).

B. Backend Compilation Service

The backend is structured as a stateless service exposing three primary endpoints: a circuit-analysis endpoint that returns topology recommendations without compiling, a compilation endpoint that runs the full pipeline and optionally forwards the result to a cloud provider, and a job-status endpoint that polls running jobs. Internal modules implement (i) the topology analyzer, (ii) the hypergraph partitioner, (iii) the cross-chip scheduler, (iv) the per-chip transpiler, and (v) a provider-abstraction layer that drives heterogeneous cloud APIs through a uniform interface.

The provider-abstraction layer maintains a small catalog of vendor device metadata (qubit counts, native gate sets, connectivity patterns) that allows the auto-router to operate without credentials. When credentials are supplied, the layer queries the vendor’s live device properties endpoint to refresh the coupling map, basis gates, and (on supported devices) per-qubit calibration data. The latter feeds Limbo’s noise-aware routing (Section V-B).

IV. THE COMPILATION PIPELINE

The compilation pipeline is a sequence of four phases, summarized in Fig. 1.

A. Phase 1: Pre-Optimization

Limbo applies a gate-level optimization pass before any partitioning work begins, and without reference to the eventual hardware coupling map. The pass cancels structurally redundant 2-qubit operations. Without this cancellation, the

interaction graph the partitioner sees contains spurious edges whose only role is to inflate the cut count.

The canonical example is the controlled-phase staircase of the QFT. Adjacent controlled-phase gates contain CNOT pairs that compose to identity when the intervening rotations are unconstrained. We have measured this single pass to roughly halve the 2-qubit operation count on QFT-style workloads at the scales of our benchmark matrix. Structurally non-redundant workloads (e.g., dense QAOA) are left largely unchanged.

B. Phase 2: Topology Analysis and Vendor Selection

Limbo classifies an input circuit by the shape of its interaction graph, then biases provider/device selection toward hardware whose physical layout matches that shape. We compute features such as graph density, locality (the fraction of edges between nearest neighbors under a canonical labeling), connected components, and the top-quartile-degree share. From these features we assign a primary topology label drawn from a small fixed set (*all-to-all*, *lattice*, *modular*, *nearest-neighbor*) and emit a confidence-ranked list of preferred providers.

The intuition is that dense or hub-heavy interaction graphs fit naturally onto trapped-ion all-to-all hardware (e.g., IonQ Aria, Quantinuum H-series) without SWAP overhead, while predominantly nearest-neighbor interactions fit onto superconducting lattice hardware (e.g., IBM Heron, Rigetti Cepheus) without entangling overhead. Modular interaction patterns (independently-clustered subgraphs that cluster naturally) are partition-friendly and can be efficiently placed on any vendor that exposes a multi-chip target.

Vendor selection respects an explicit user override; in its absence, the recommendation is used as a tie-breaker among devices whose capacity satisfies the circuit’s qubit count requirement.

C. Phase 3: Hypergraph Partitioning

With pre-optimization and topology analysis complete, Limbo builds a weighted hypergraph from the circuit’s interaction graph and hands it to a multi-level hypergraph partitioner. Vertices are logical qubits, hyperedges are gates (2-qubit gates contribute weight-1 edges between their endpoints; multi-qubit gates contribute a single hyperedge over their entire support). The partitioner’s objective is the KM-1 cut metric: minimize the total weight of hyperedges that span multiple partitions, subject to a capacity constraint on each partition.

The KM-1 objective is ideal for multi-chip targets because it counts each cross-chip multi-qubit operation as a single cost event, matching the runtime semantics of a remote-gate protocol. A naïve edge-cut objective would over-count when a 3-qubit gate’s support straddles a chip boundary in a non-trivial way.

D. Phase 4a: Per-Chip Hardware-Aware Transpilation

Once the partition is fixed, each chip receives a sub-circuit consisting of the local gates whose support lies entirely within that partition. Limbo transpiles each sub-circuit against the

target device’s actual coupling map and native gate set, using SABRE-style heuristic routing at the highest optimization level supported by the underlying transpilation engine. Because each sub-circuit is much smaller than the original (typically $1/k$ of the logical qubit count for k chips), the per-chip routing problem is dramatically easier than routing the full circuit against a single combined multi-chip coupling map.

This is one of the structural reasons Limbo reduces compile time compared to routing-first approaches: a heuristic-search router on k small graphs is much faster than one search on the combined graph, and the per-chip searches are highly parallel.

E. Phase 4b: Topological Cross-Chip Event Scheduling

The remaining cross-chip 2-qubit operations (one per cut hyperedge from Phase 3) are assembled into a *cross-chip event schedule*. Limbo’s scheduler proceeds in topological order along the original circuit’s gate-dependency DAG, ensuring that each event respects the data dependencies of its endpoints. At each step, the scheduler follows two physical constraints:

(i) *Port capacity*: each chip exposes a finite number of communication qubits through which cross-chip entanglement can flow. The scheduler cannot dispatch more concurrent cross-chip events between a pair of chips than that pair’s link port capacity allows.

(ii) *Critical-path priority*: when multiple cross-chip events are eligible at the same step, the scheduler prefers the event on the longest remaining path in the DAG. This priority reduces makespan when port capacity is the bottleneck, which is the regime real modular hardware operates in.

The combination of topological ordering and critical-path priority is the multi-chip analogue of SABRE’s intra-chip SWAP insertion. SABRE chooses how to physically connect two distant qubits on one chip; Limbo’s scheduler chooses when to execute each unavoidable cross-chip event.

V. VENDOR ROUTING AND NOISE-AWARE COMPILATION

A. Topology-Aware Vendor Routing

Modern quantum cloud users have access to multiple vendors through multiple SDKs. The choice of where to run a given circuit is non-trivial. A Fourier-style dense circuit incurs substantial SWAP overhead on a sparse superconducting lattice but runs natively on trapped-ion all-to-all hardware. A nearest-neighbor variational ansatz is the inverse. Limbo automates this choice using the topology classification from Phase 2 (Section IV-B) as the primary criterion. Ties among acceptably-matched devices are broken by current queue depth.

This vendor routing operates as the architectural front-end of the compilation pipeline. Rather than treating hardware selection as an independent user decision, Limbo integrates device selection directly into the optimization loop. The graph-classification machinery developed in Phase 2 ensures that the downstream hypergraph partitioner and per-chip transpiler operate on a physical target whose lattice symmetries mirror the interaction dynamics of the input workload. Mapping penalties are mitigated before the first structural cut is taken.

B. Noise-Aware Routing on Single-Chip Targets

On single-chip targets, the per-chip transpilation step (Section IV-D) benefits from device calibration data when the vendor exposes it. Limbo’s provider-abstraction layer extracts per-qubit randomized-benchmarking fidelities (when available) and constructs an instruction-weighted routing target that SABRE consults during layout selection. The routed circuit prefers the highest-fidelity qubits and avoids those the device reports as broken or uncalibrated.

Vendor coverage of calibration data is incomplete as of the current date. Superconducting hardware accessible through public cloud providers universally exposes per-qubit single-qubit randomized-benchmarking fidelities. Per-edge 2-qubit fidelities and per-qubit readout fidelities are not consistently exposed. When a class of calibration data is missing, Limbo synthesizes uniform default error rates of the right order of magnitude so the routing target remains usable. Single-qubit routing is then driven by real calibration data, and two-qubit routing falls back to uniform-cost SABRE.

C. Variational Fast-Path

Variational quantum algorithms (VQE, QAOA) submit a parametric ansatz hundreds to thousands of times during an optimizer loop, with each submission differing only in the bound values of a small number of rotation parameters. Naively recompiling the full ansatz on each iteration wastes substantial compute. Limbo’s template cache recognizes when a previously-compiled parametric template can be re-used by re-binding parameters locally, and skips re-running Phases 1–4 in that case. A 100-step variational loop on a moderately deep ansatz pays the heavy compilation cost exactly once.

VI. EVALUATION

A. Workload Matrix

We evaluate Limbo against three baselines on a workload matrix of seven circuit generators spanning four behavioral families: **Fourier**, QFT and inverse QFT; **Variational**, QAOA on a random-density max-cut graph and a hardware-efficient ansatz; **Randomized**, a random Clifford+T circuit and a sparse random circuit; and **Application-shaped**, an asymmetric hotspot circuit with hub-and-spoke interaction structure. Each generator is parameterized by qubit count n and an internal gate budget. We run at logical qubit counts $n \in \{16, 32, 64\}$ for the broader matrix and $n \in \{32, 64\}$ for the TKET head-to-head on a fixed 4 chip $\times$ 20 qubit-per-chip hardware envelope. A single random seed of 64 is used throughout for reproducibility.

B. Baselines

We compare Limbo against multiple baselines. The first is TKET (pytket), configured with `RemoveBarriers`, `DecomposeBoxes`, `DefaultMappingPass` against a multi-chip `Architecture`, and `FullPeepholeOptimise`. The multi-chip architecture mirrors Limbo’s 4 $\times$ 20 envelope with configurable inter-chip link counts. The next is QUPORT, an open-source baseline

that performs graph partitioning without integrated per-chip routing, configured with its TPCCAP local-search partitioner and layered makespan estimator. And lastly, a graph-blind naïve baseline that stripes qubits by index across the four chips, with no graph analysis. This baseline exists to show that Limbo’s partitioner is doing real work.

For both Limbo and TKET, *every* compiler is run end-to-end: TKET routes the full circuit against the multi-chip architecture with SWAP insertion and peephole optimization; Limbo pre-optimizes, partitions, and per-chip-transpiles each sub-circuit against the intra-chip coupling map at the highest optimization level. Cuts are counted as cross-chip 2-qubit operations in each compiler’s final physical output, *including* any SWAPs that TKET inserts.

C. Compiler-Metric Results

Table I reports the per-cell cross-chip 2-qubit operation count for Limbo and TKET across the multi-chip workload subset. Limbo wins on every cell. The win is most pronounced under the constrained grid intra-chip topology, in which TKET must insert SWAPs even within a chip to satisfy nearest-neighbor adjacency, and those SWAPs frequently cascade across chip boundaries. Limbo’s partition-first design avoids the cascade entirely: cross-chip events are emitted directly between non-adjacent physical qubits without intermediate SWAPs.

Table II summarizes the aggregate headline numbers. The compile-time ratio is computed cell-wise as TKET’s wall-clock compile time divided by Limbo’s; on the hardest cell (grid-topology Sparse Random at $n = 64$) the ratio reaches $2735\times$. Three factors contribute to Limbo’s compile-time advantage: (i) partitioning splits the routing problem into $k = 4$ much smaller sub-problems whose individual SABRE searches are dramatically cheaper than one search on the combined graph; (ii) Limbo’s per-chip transpilation is largely parallel (though we do not exploit this in the timing numbers); and (iii) TKET’s `FullPeepholeOptimise` pass runs on the larger routed output, which scales with circuit depth that the partition has already collapsed for Limbo.

D. Output-Fidelity under Multi-Chip Noise

The cross-chip 2-qubit operation count above is a compiler metric. We additionally evaluate the underlying causal claim: that on real modular hardware, fewer cross-chip operations translate into higher output-state fidelity. As direct validation on hardware is not yet possible (Section VI-F), we instead validate through a calibrated multi-chip noise model.

We construct a Qiskit Aer `NoiseModel` that classifies every 2-qubit operation as intra-chip or cross-chip and assigns corresponding depolarizing error rates: 0.5% for intra-chip and 5% for cross-chip.

The intra-chip rate is consistent with published median 2-qubit gate errors on current production superconducting hardware: IBM Heron-family processors report median CZ errors in the 0.3–0.8% range across their published backend calibration [13], and Google’s Willow processor has reported median two-qubit-gate errors below 0.5% on randomized

benchmarking [14]. The cross-chip rate is consistent with published experimental fidelities of remote-entanglement protocols: Krutyanskiy et al. report Bell-state fidelity of $\sim 88\%$ for trapped-ion qubits entangled across a 230 m fiber link [15], corresponding to a $\sim 12\%$ remote-gate error; the same group’s shorter-link results, and analogous superconducting modular prototypes, fall in the range of single digit percentages. We choose 5% as a conservative midpoint of this range; the qualitative conclusions of this section are insensitive to the exact choice of cross-chip rate within an order of magnitude.

One-qubit gates carry a uniform 0.05% depolarizing rate and readout a uniform 1% bit-flip rate, matching the typical 1Q error derived from randomized benchmarking on contemporary superconducting QPUs (Section VI-F reports the 102 per-qubit fidelities from the device we used for the hardware validation). We choose circuit scales $n = 12$ and $n = 16$ as they are small enough for full statevector noise simulation, evaluate the Hellinger fidelity between the noisy measurement distribution and the noiseless ideal, and compare Limbo against TKET.

Table III reports the fidelity comparison on the informative cells. Limbo is shown to have an advantage on 8/8. The maximum single-cell gain is +6.66 percentage points (Sparse Random at $n = 12$). Note that QFT and inverse-QFT cells are excluded as *uninformative*: the ideal output of QFT applied to $|0\rangle^{\otimes n}$ is the uniform superposition, and Hellinger fidelity to “noisy roughly uniform” is insensitive to which compiler produced the noisy result. We keep those cells in the underlying data for completeness but they cannot distinguish compilers on this metric. The remaining cells, where the ideal output is structurally concentrated, do distinguish them.

A second observation: at $n = 16$, both compilers’ absolute fidelities collapse toward the noise floor (~ 0.1) because the total number of cross-chip operations saturates the multi-chip error budget. Limbo still yields better results in all of these cells, including Asymmetric Hotspot at $n = 16$ where Limbo’s fidelity is $3.2\times$ TKET’s, but the absolute numbers are small enough that we report them with the caveat that beyond this depth the comparison is between two noise-saturated outputs.

E. Broader Coverage vs. a Research Partitioner

Across a wider 21-cell matrix with seven workloads at three scales, the behavior holds against QUPORT as well: Limbo performs better or equivalently on 19/21 cells, with a maximum cross-chip 2-qubit operation reduction of 70% and an average reduction over the matrix. Against the graph-blind naïve baseline, Limbo’s reduction rises to 71% on average, confirming that the hypergraph partitioning step is doing substantial work that simple striping cannot replicate.

The two cells where Limbo does not strictly win against QUPORT are both Hardware-Efficient Ansatz at small scale, where the nearest-neighbor structure trivially fits on a single chip and the cross-chip cut count is degenerate (zero or near-zero) for both compilers. These cells are annotated rather than excluded from the matrix.

TABLE I

CROSS-CHIP 2-QUBIT OPERATION COUNT: LIMBO VS. TKET, TWO INTRA-CHIP TOPOLOGIES. EACH CELL REPORTS (LIMBO CUTS / TKET CUTS; REDUCTION %).

Workload	Cliques intra-chip		Grid intra-chip	
	$n = 32$	$n = 64$	$n = 32$	$n = 64$
QFT	324 / 472 (31%)	1028 / 1482 (31%)	324 / 500 (35%)	1028 / 1558 (34%)
IQFT	334 / 450 (26%)	920 / 1474 (38%)	326 / 538 (39%)	904 / 1614 (44%)
QAOA MaxCut	248 / 344 (28%)	1672 / 1944 (14%)	224 / 424 (47%)	1676 / 1980 (15%)
Random Clifford+T	50 / 120 (58%)	204 / 421 (52%)	50 / 228 (78%)	211 / 566 (63%)
Sparse Random	66 / 125 (47%)	202 / 416 (51%)	66 / 204 (68%)	208 / 441 (53%)
Asymmetric Hotspot	127 / 148 (14%)	429 / 654 (34%)	126 / 230 (45%)	427 / 776 (45%)

TABLE II

HEADLINE RESULTS AGGREGATED ACROSS BOTH TOPOLOGIES ($N = 24$ CELLS). COMPILE-TIME RATIOS ARE TKET WALL-CLOCK DIVIDED BY LIMBO WALL-CLOCK, PER-CELL; WE REPORT BOTH MEDIAN AND MEAN BECAUSE THE DISTRIBUTION IS RIGHT-SKEWED BY SEVERAL GRID-TOPOLOGY OUTLIERS ABOVE $1000\times$.

Metric	Value
Limbo wins on cross-chip 2q-op count	24/24
Average cross-chip 2q-op reduction	41%
Maximum cross-chip 2q-op reduction (single cell)	78%
Median compile-time ratio (TKET/Limbo)	119×
Mean compile-time ratio (skewed by grid outliers)	291×
Maximum compile-time ratio (single cell)	2735×

TABLE III

HELLINGER FIDELITY VS. IDEAL UNDER MULTI-CHIP NOISE. EIGHT INFORMATIVE CELLS (THOSE WITHOUT UNIFORM-IDEAL QFT/IQFT DEGENERACY).

Workload	n	Limbo	TKET
QAOA MaxCut	12	0.6615	0.6474
Random Clifford+T	12	0.4774	0.4312
Sparse Random	12	0.6358	0.5692
Asymmetric Hotspot	12	0.1647	0.1056
QAOA MaxCut	16	0.1214	0.1146
Random Clifford+T	16	0.1798	0.1315
Sparse Random	16	0.1270	0.1147
Asymmetric Hotspot	16	0.0765	0.0242

F. Real-Hardware Pipeline Validation

We executed Limbo on a real cloud-accessible quantum device: Rigetti’s Cepheus 108-qubit superconducting QPU, hosted on AWS Braket. Because the multi-chip architectural claim cannot be tested on this class of hardware (it is a single chip), we are explicit about what this run validates and what remains open.

Two representative workloads (Sparse Random and Asymmetric Hotspot, $n = 12$, 256 shots each) compiled through the Limbo pipeline, submitted successfully via the production API, ran on the device, and returned measurement counts. The full chain (compile, partition, transpile, submit, poll, retrieve) executes end-to-end without manual intervention.

This confirms that Limbo emits hardware-runnable physical circuits accepted by a real cloud provider, that the provider abstraction correctly drives a vendor’s submit and poll APIs, and that the result-decoding path correctly reconstructs a counts distribution from raw shot data. Separately, Limbo’s *noise-aware routing* consumes the 102 per-qubit randomized-benchmarking fidelities the device publishes and reports `noise_aware_routing: true` in the compile metrics; five qubits with fidelity = 0.5 (the platform’s convention for broken or uncalibrated qubits) are excluded from the layout.

However, the multi-chip architectural claim itself is not tested by this run. The device is a single chip, so Limbo’s hypergraph partitioner correctly identifies that the 12 logical qubits fit comfortably within one partition and does not partition at all. The cross-chip 2-qubit operation count is therefore zero in both the Limbo and the baseline path; there is no architectural variable for hardware noise to discriminate between. We make no fidelity comparison on this run for that reason. The observed Hellinger fidelity for both workloads sits at the device’s noise floor, which is expected behavior for a 12-qubit circuit of $\sim 100+$ native-gate depth on contemporary superconducting hardware and is independent of which compiler produced the program [13], [16].

VII. DISCUSSION AND LIMITATIONS

A. The Multi-Chip Architectural Claim

The primary claim of this paper—that partition-first compilation reduces cross-chip operations and consequently improves fidelity on multi-chip hardware—is validated through two distinct lines of evidence: a large compiler-metric matrix that directly measures cross-chip operation counts, and a calibrated noise-model simulation that measures the downstream fidelity consequences. Both lines of evidence are simulator-shadowed, not hardware-direct. This is due to the fact that no publicly cloud-accessible multi-chip quantum hardware exists at the time of writing. We anticipate that this will change within the next 12–24 months and that direct hardware validation of the multi-chip claim will become possible. Until then, the simulator results are used as the basis of our conclusions.

Direct hardware validation of the multi-chip claim requires a publicly cloud-accessible device whose gate topology contains a meaningful intra-chip and cross-chip distinction with measurably different gate fidelities for the two classes. The closest

candidates as of writing are Quantinuum’s multi-zone H-series [17], which provides the ion-shuttling analogue of cross-chip cost but not a photonic-link error model, and prototype multi-chip IBM Heron-based systems [1], which are not yet on IBM Cloud.

B. Pre-Optimization vs. Stock L3 Transpilation

On single-chip workloads at small to moderate scale, Limbo’s gate-level pre-optimization step is largely redundant with the optimization passes built into QISKIT’s level-3 transpile. We verified this through a separate pre-flight that compared Limbo’s compile path against vanilla QISKIT transpile on a set of small workloads (GHZ, QAOA, QFT, IQFT at $n \in \{4, 6, 8, 12\}$). The two paths produced essentially equivalent native-gate output; small differences (a few gates either way) fell within SABRE’s seed-dependent stochastic variance. Pre-optimization’s documented benefit on QFT-style cancellation patterns survives through stock Qiskit L3 at hardware-runnable scales.

This pre-optimization pass is redundant for single-chip targets under stock Qiskit L3 transpilation; however, it is structurally required in the multi-chip pipeline to prevent the partitioner from constructing an over-pessimistic interaction graph.

C. Hardware Calibration Coverage

Limbo’s noise-aware routing relies on the vendor exposing per-qubit and per-edge calibration data. As of writing, only per-qubit randomized-benchmarking fidelities are universally exposed on superconducting devices accessible through public cloud providers. Per-edge 2-qubit fidelities and per-qubit read-out fidelities are inconsistently exposed. We have implemented graceful fallback to uniform defaults for the missing categories, but the noise-aware contribution is therefore narrower than “full noise-aware routing.” If vendors begin exposing complete calibration data, Limbo will use it without further code changes.

D. Robustness: Seed and Baseline Configuration

Two natural objections to the results in Section VI are (i) the wins might be artifacts of the single random seed used in the matrix runner, and (ii) the TKET configuration might not be its strongest. We re-ran the full 24-cell matrix under three configurations to address both: the headline configuration (seed = 64, the fair TKET pipeline with both pre-routing and post-routing `FullPeepholeOptimise` passes), the same fair pipeline at a different seed (seed = 42), and the previously-used stock TKET pipeline (post-routing `FullPeepholeOptimise` only, seed = 64). Limbo outperformed TKET on all three configurations on 24/24 cells. The maximum cross-chip 2-qubit operation reduction varied across configurations within a narrow band (70–78%); the average cross-chip operation reduction varied within 39.6–41.3%. The compile-time advantage is preserved across configurations, with median ratios in the 76–119 $\times$ range; switching to the fair TKET pipeline *strengthens* Limbo’s compile-time advantage

rather than weakens it, because the additional optimization pass adds work to TKET without proportionally reducing cuts. The maximum per-cell speedup is the most variable summary across configurations (709–2735 $\times$), reflecting the fact that different seeds expose different cells as “hardest for TKET”. We therefore use the median in summary statements and report the maximum only with the qualifier that it identifies a single specific worst-case cell.

VIII. CONCLUSION

In this paper, we present Limbo, a quantum compiler designed for multi-chip architectures that combines routing-first and graph-partitioning compilation approaches. Limbo partitions the circuit graph globally using multi-level hypergraphs, sequences the resulting cross-chip operations through an event scheduler based on topological sorting, and feeds each partition into a per-chip transpiler. Partitioning and per-chip routing are phases of one hardware-aware pipeline rather than separate tools. The result is substantial wins on the metric that matters most on modular quantum hardware: cross-chip 2-qubit operation count.

Against TKET, Limbo performs better on 24/24 cells of a multi-chip workload matrix, reducing cross-chip operations by up to 78% and compile time by up to 2735 $\times$ on the hardest cell. Under a calibrated multi-chip noise model, Limbo wins on output Hellinger fidelity on 8/8 informative cells. Against an established graph-partitioning baseline (QUPT) and a graph-blind baseline, Limbo wins on a broader 21-cell matrix. End-to-end pipeline execution is validated on real cloud-accessible superconducting quantum hardware.

The direct hardware validation of the multi-chip architectural claim itself remains an open problem until multi-chip quantum hardware becomes accessible on a public cloud. We expect that to happen within the next 12–24 months, at which point Limbo’s simulator-shadowed results can be cross-checked against real photonic-link hardware without modification to the compiler itself.

ACKNOWLEDGMENT

The author thanks the maintainers of QISKIT, TKET, QUPT, Mt-KaHyPar, and the public cloud quantum APIs of IBM Quantum, AWS Braket, and Azure Quantum for making this work possible.

REFERENCES

- [1] J. Gambetta, “IBM Quantum System Two: The Era of Quantum Utility Is Here,” IBM Quantum Blog, 2023, accessed May 2026. [Online]. Available: <https://www.ibm.com/quantum/blog/quantum-roadmap-2033>
- [2] Quantinuum, “H-Series Multi-Zone Trapped-Ion Architecture,” Vendor white paper, 2024, accessed 2026.
- [3] PsiQuantum, “Modular Fault-Tolerant Quantum Computing with Photonics,” Technical overview, 2024, accessed 2026.
- [4] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, “ $t|ket$: A retargetable compiler for NISQ devices,” *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.
- [5] Qiskit Contributors, “Qiskit: An Open-Source Framework for Quantum Computing,” <https://qiskit.org>, 2024, accessed 2026.
- [6] G. Li, Y. Ding, and Y. Xie, “Tackling the qubit mapping problem for NISQ-era quantum devices,” in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2019, pp. 1001–1014.

- [7] S. Sarkar, “QuPort: Topology-, Port-, and Congestion-Aware Compilation for Modular Multi-QPU Quantum Systems,” Open-source software, 2026, accessed May 2026. [Online]. Available: <https://github.com/neuralsorcerer/quport>
- [8] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio, “Optimal local implementation of nonlocal quantum gates,” *Physical Review A*, vol. 62, no. 5, p. 052317, 2000.
- [9] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, “High-quality hypergraph partitioning,” *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2022.
- [10] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.
- [11] J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello, “Distributed quantum computation over noisy channels,” *Physical Review A*, vol. 59, no. 6, pp. 4249–4254, 1999.
- [12] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather, “Efficient distributed quantum computing,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 469, no. 2153, p. 20120686, 2013.
- [13] IBM Quantum, “Heron Processor Calibration Data: Median Two-Qubit Gate Fidelities,” *IBM Quantum Backend Properties Reports*, 2024–2026, published calibration data for ibm_torino, ibm_kingston, and related Heron-family processors consistently reports median CZ errors in the 0.3–0.8% range. See <https://quantum.ibm.com/services/resources>.
- [14] H. Neven and the Google Quantum AI Team, “Meet Willow: Our State-of-the-Art Quantum Chip,” *Google Quantum AI Blog and supporting publications*, 2024, reports median two-qubit-gate errors below 0.5% across the Willow processor on randomized benchmarking.
- [15] V. Krutyanskiy, M. Galli, V. Krcmarsky, S. Baier, D. A. Fioretto, Y. Pu, A. Mazloom, P. Sekatski, M. Canteri, M. Teller, J. Schupp, J. Bate, M. Meraner, N. Sangouard, T. E. Northup, and B. P. Lanyon, “Entanglement of trapped-ion qubits separated by 230 meters,” *Physical Review Letters*, vol. 130, no. 5, p. 050803, 2023, reports remote-entanglement Bell-state fidelity of $\sim 88\%$, equivalent to a $\sim 12\%$ remote-gate error.
- [16] Rigetti Computing, “Aspen and Ankaa-Family Two-Qubit Gate Benchmarks,” *Rigetti Quantum Processor Specifications*, 2022–2026, two-qubit (CZ) gate fidelities for production Aspen and Ankaa devices have been reported in the 97–99% range, corresponding to gate errors of 1–3%.
- [17] S. A. Moses, C. H. Baldwin, M. S. Allman, R. Ancona, U. Ascarrunz, C. Barnes, J. Bohnet, S. J. Brown, N. Q. Burdick, W. C. Burkholder *et al.*, “A race track trapped-ion quantum processor,” *Physical Review X*, vol. 13, no. 4, p. 041052, 2023, quantinuum H2 system. Reports median two-qubit gate errors of $\sim 0.2\%$ and cross-zone (effectively cross-region) two-qubit operations with shuttling overhead; cited here for the comparative baseline of multi-zone ion-trap fidelity at the upper end of what is achievable today.