A Rust-accelerated, topology-informed workflow for Kuramoto–XY quantum simulation and NISQ benchmarking

Miroslav Šotek[1, 2, *]

[1] *ANULUM, Marbach SG, Switzerland*

[2] *ORCID: 0009-0009-3560-0851*

(Dated: May 25, 2026)

(Nodo DOI: 10.5281/zenodo.20382138)

Abstract

We describe the software and experimental workflow behind `scpn-quantum-control`, an open-source framework for mapping heterogeneous Kuramoto-type oscillator networks to XY spin Hamiltonians, compiling topology-informed variational ansatze, benchmarking classical and quantum simulation paths, and executing reproducible NISQ hardware experiments. The framework combines Python/Qiskit orchestration with a Rust/PyO3 acceleration layer for hot-path kernels such as coupling-matrix construction, Hamiltonian assembly, trajectory features, hybrid digital–analog partitioning, and protected-subspace diagnostics. We report artefact-generated timings on three CPU sources (local workstation, ML350, and Vertex AI `n1-standard-4`) and one Vertex AI Tesla T4 GPU run. Fresh benchmark artefacts generated on 2026-05-05 show direct Rust coupling-matrix speedups of 15.5–44.1× at $n = 16$ across the three self-contained CPU runs, exact checksum parity across Python, Rust, and Go kernels, and a three-seed $n = 4$ VQE comparison in which the topology-informed ansatz achieves a mean relative energy error of 0.455 %, compared with 6.136 % for `TwoLocal` and 7.522 % for `EfficientSU2` at the same iteration budget. A follow-up convergence artefact extends the same comparison to 320 COBYLA evaluations and records seed-wise uncertainty: at $n = 4$ and one ansatz repetition the topology-informed family reaches a mean relative error of 3.08×10^{-8} %, compared with 6.76×10^{-4} % for `TwoLocal` and 1.92×10^{-3} % for `EfficientSU2`. A batched state-vector expectation benchmark on Vertex T4 gives CUDA median times of 0.176–0.905 ms for $n = 8$–11, compared with Vertex CPU medians of 3.557–12.094 ms for the same workload. IBM ansatz-validation lanes were then executed at $n = 4, 6, 8$ with full computational-basis readout calibration. The original $n = 4$ Marrakesh lane preserves the topology-informed ordering, but the larger $n = 6, 8$ Fez/Marrakesh lanes rank `EfficientSU2` lowest in measured energy. Additional $n = 8$ seed-sensitivity lanes with seed 23 preserve that larger-width ordering on both backends. The hardware evidence is therefore a reproducibility and ansatz-sensitivity check, not a backend-general topology-informed superiority claim. The paper is methodological rather than a quantum-advantage claim: small-system exact simulation remains classically tractable, and hardware results are used to validate compilation, noise budgeting, and experiment reproducibility. All code, benchmark commands, hardware raw counts, and derived artefacts are public at `anulum/scpn-quantum-control`.

* protoscience@anulum.li

I. INTRODUCTION

Quantum simulation papers often underreport the engineering layer that connects a model Hamiltonian to executable circuits, raw-count archives, analysis scripts, and reproducible performance claims. This omission is especially problematic for NISQ studies, where small changes in ansatz topology, transpilation, hardware calibration, readout processing, or classical baseline implementation can dominate the advertised physics.

This manuscript presents the computational methodology used in the SCPN Kuramoto–XY campaign [1]. The contribution is not a broad quantum-advantage result. Instead, it is a reproducible pipeline for:

1. compiling heterogeneous oscillator couplings into XY Hamiltonians,

2. generating coupling-topology-informed ansatze and Trotter circuits,

3. dispatching hot kernels to Rust while preserving Python/Qiskit usability,

4. benchmarking classical, simulator, and hardware paths with command-level provenance, and

5. archiving raw IBM hardware counts with integrity hashes and reproduction scripts.

The immediate use case is Kuramoto-type synchronisation mapped to the XY Hamiltonian

$$H_{XY} = \sum_{i<j} K_{ij} \left(X_i X_j + Y_i Y_j \right) + \sum_i \omega_i Z_i, \tag{1}$$

where K_{ij} encodes the oscillator coupling topology and ω_i are heterogeneous natural frequencies. The same workflow also supports VQE, Trotterised dynamics, readout/post-processing studies, DLA-parity diagnostics, and hardware experiment packaging.

The paper is intentionally separated from the companion IBM hardware preprint. The hardware preprint asks whether a parity-sector and excitation-number correlated leakage signal is reproducible on real Heron r2 hardware [2, 3]. The present manuscript asks a different question: whether the software stack that generated, validated, and analysed those experiments is itself reproducible, benchmarked, and reusable.

Layer	Representative modules	Role
Bridge	`bridge/knm_hamiltonian.py`	$K_{ij} \to H_{XY}$
Ansatz	`control/structured_ansatz.py`	topology-informed circuits
Phase	`phase/phase_vqe.py`	VQE and dynamics workflows
Hardware	`hardware/runner.py`	simulator / IBM execution
Rust	`scpn_quantum_engine`	hot-path kernels

TABLE I. High-level workflow components. The paper reports only auditable functions with command-level provenance or committed test coverage.

II. SOFTWARE ARCHITECTURE AND ARTEFACT DISCIPLINE

The framework is organised as a Python front-end with a Rust/PyO3 engine for hot paths. The Python layer provides model construction, Qiskit circuits, hardware runners, analysis scripts, and documentation. The Rust layer provides deterministic kernels for coupling construction, trajectory features, Hamiltonian-adjacent operations, and selected diagnostics. This division keeps the public API accessible while reducing overhead in loops where Python object allocation would dominate.

The central reproducibility rule in this manuscript is that numerical tables are not hand-authored. Benchmarks are generated by scripts under `scripts/` and written to JSON/CSV artefacts under `data/rust_vqe_methods/`. The manuscript cites those artefacts directly. This prevents stale performance claims and makes later reruns auditable when compilers, CPUs, cloud machines, or Python dependencies change.

III. MINIMAL WORKING EXAMPLE

The core user workflow is intentionally short: construct a Kuramoto–XY coupling matrix, convert it into an operator, generate a topology-informed ansatz, and then pass the same artefacts to a VQE or benchmark harness. The following sketch is representative of the public API used by the benchmark scripts:

```
from scpn_quantum_control.bridge.knm_hamiltonian import (
    OMEGA_N_16,
```

```
    build_knm_paper27,
    knm_to_ansatz,
    knm_to_hamiltonian,
)


n = 4
k_matrix = build_knm_paper27(n)
hamiltonian = knm_to_hamiltonian(k_matrix, OMEGA_N_16[:n])
ansatz = knm_to_ansatz(k_matrix, reps=2)
```

The benchmark artefacts in `data/rust_vqe_methods/` are produced by extending this workflow with fixed seeds, fixed optimiser budgets, explicit machine metadata, and JSON/CSV output. This keeps the examples close to the submitted hardware and methods data rather than presenting toy APIs unrelated to the reported numbers.

IV. RUST KERNEL BOUNDARY

The Rust layer is deliberately narrow. It accelerates deterministic kernels that are called repeatedly from Python workflows, while Qiskit remains the circuit, operator, and transpilation layer. For the coupling-matrix benchmark, the accelerated operation has the conceptual signature

```
pub fn build_knm(n: usize, k0: f64, alpha: f64) -> Vec<f64>
```

and returns a row-major dense representation of the exponentially decaying coupling matrix. The benchmark harness compares this path against matched Python and Go implementations using checksum parity before reporting timing statistics. The reported speedups therefore refer to this kernel boundary, not to the entire Qiskit workflow.

This distinction is an Amdahl's-law caveat rather than a weakness. For example, reducing a K_{ij} construction call from roughly 10^{-2} ms to 10^{-3} ms is useful when the kernel is called many times inside scans or packaging loops, but it does not remove the dominant costs of an end-to-end VQE or hardware workflow. Qiskit circuit construction and transpilation, optimiser evaluations, state-vector expectation calculations, GPU/CPU dense linear algebra, queue latency, and QPU execution can each dominate the wall time in different regimes.

Ansatz	Params	Raw depth	2q gates	Median VQE error
K_{ij}-informed	16	13	12	0.372 %
`TwoLocal`	24	15	12	7.164 %
`EfficientSU2`	24	11	6	7.708 %

TABLE II. Generated $n = 4$, two-repetition ansatz/VQE benchmark. Circuit columns come from `ansatz_benchmark_summary_2026-05-05.json`; VQE errors come from `vqe_benchmark_summary_2026-05-05.json` with three seeds and 80 COBYLA iterations per seed.

The software claim is therefore bounded to reducing specific deterministic bottlenecks and improving reproducibility, not to accelerating every workflow stage by the reported kernel factor.

V. TOPOLOGY-INFORMED ANSATZ DESIGN

The topology-informed ansatz uses the support of K_{ij} to decide which qubit pairs receive entangling gates. This is a simple design principle: do not spend variational capacity on edges absent from the physical coupling graph. For sparse or structured oscillator networks, this can reduce parameter count and circuit clutter relative to generic linear or all-to-all ansatze.

The current benchmark harness regenerates ansatz construction and VQE tables from source. For the representative $n = 4$, two-repetition case, the topology-informed ansatz uses fewer parameters than the generic alternatives while retaining the same raw two-qubit-gate count as `TwoLocal`. The generated VQE artefact then compares all three ansatze over three random seeds with identical COBYLA iteration budgets. The original comparison is deliberately reported as a fixed-budget score, not as a converged variational optimum: the May 5 artefact records the terminal energies after 80 COBYLA iterations per seed, but did not publish a full energy-versus-iteration convergence trace. The 2026-05-20 extension closes that specific documentation gap for the representative $n = 4$ lane by publishing checkpointed best-so-far energies, final parameters, elapsed-time spread, and standard-error columns. It does not remove the broader small-system limitation, but it distinguishes an 80-evaluation screening result from a longer optimiser-trace result.

Ansatz	Mean error	Std. err.	Best error	Mean time
K_{ij}-informed	$3.08 \times 10^{-8}\,\%$	$1.36 \times 10^{-8}\,\%$	$1.07 \times 10^{-8}\,\%$	187.0 ms
`TwoLocal`	$6.76 \times 10^{-4}\,\%$	$6.76 \times 10^{-4}\,\%$	$5.59 \times 10^{-8}\,\%$	612.8 ms
`EfficientSU2`	$1.92 \times 10^{-3}\,\%$	$1.69 \times 10^{-3}\,\%$	$2.93 \times 10^{-5}\,\%$	618.7 ms

TABLE III. Generated $n = 4$, one-repetition VQE convergence extension from `vqe_convergence_methods_2026-05-20.json`. Each row uses three seeds and 320 COBYLA evaluations. The table reports best-so-far relative energy error against exact diagonalisation; the standard error is across the three seeds.

Ansatz	n	Params	Transpiled depth	2q gates
K_{ij}-informed	16	32	68	115
`TwoLocal`	16	64	72	120
`EfficientSU2`	16	64	25	15
K_{ij}-informed	20	40	84	163
`TwoLocal`	20	80	88	190
`EfficientSU2`	20	80	29	19

TABLE IV. Generated construction/transpilation scaling extension from `ansatz_scaling_error_bars_2026-05-20.csv`. This table is not a dense VQE energy claim at $n = 16$–20; it is the memory-safe large-n resource lane requested by the methods review.

A follow-up scaling artefact extends the circuit-size and truncation diagnostics to $n = 4, 6, 8, 10, 12$. Dense exact diagonalisation is used up to $n = 8$, and sparse `eigsh` ground-state references are used for $n = 10, 12$. The generated sparse residual norms are 1.12×10^{-11} at $n = 10$ and 8.70×10^{-10} at $n = 12$. The corresponding MPS truncation diagnostic gives mid-chain entropies of 0.9999 bits at $n = 10$ and 1.3143 bits at $n = 12$ for the canonical Kuramoto–XY coupling matrix. These rows strengthen the scaling diagnostic, but they remain ground-state reference data rather than a large-system VQE optimisation theorem.

Task	n	Median time	Note
`build_knm`, Python	4	0.00627 ms	reference
`build_knm`, Rust	4	0.00106 ms	5.89×
`build_knm`, Python	16	0.01028 ms	reference
`build_knm`, Rust	16	0.00277 ms	3.71×
`build_knm`, Python	32	0.01148 ms	reference
`build_knm`, Rust	32	0.00828 ms	1.39×
Dense H_{XY}	8	0.11705 ms	Hermitian error 0
Timing context	–	–	shared workstation; non-isolated

TABLE V. Generated opportunistic shared-workstation core benchmark table from `rust_core_benchmark_summary_2026-05-05.json`. Python/Rust coupling matrices match to the configured tolerance for all tested sizes $n \in \{4, 8, 16, 32, 64\}$. The dense H_{XY} row reports the maximum absolute Hermiticity residual, which is zero for the generated matrix.

VI. GENERATED CPU BENCHMARK ARTEFACTS

Table V lists conservative local performance facts generated by the new benchmark harnesses. These are opportunistic timings on a shared workstation, not final publication-grade timing claims; CPU load from other processes was not pinned or isolated. The older internal claim of a 5401× Hamiltonian-construction speedup is not used here because it has not yet been reproduced by the present harness with a matched baseline. The fresh coupling-matrix benchmark shows useful speedups at $n = 4$–32, but not at $n = 64$; the manuscript therefore reports bounded kernel-level acceleration rather than a universal speedup.

To make the comparison extensible, we added a multi-language coupling-matrix harness. It currently reports Python/NumPy, direct Rust/PyO3, Julia, and Go timings where local runtimes are available. Runtimes without a stable matched K_{ij} kernel are intentionally omitted from the result table rather than listed as failed benchmarks.

The self-contained cross-machine harness was then executed on three independent CPU sources: the local workstation `aaarthuus`, the ML350 server `god-of-the-math`, and a Vertex AI `n1-standard-4` worker. This harness deliberately avoids importing the project package so that a secondary machine can run the same Python/Rust/Go kernels before the full

Language path	$n = 16$ median	Status
Python/NumPy	0.01339 ms	ok
Rust/PyO3	0.00330 ms	ok
Julia	0.00509 ms	ok
Go	0.00402 ms	ok

TABLE VI. Generated multi-language K_{ij} construction comparison at $n = 16$ from `multilang_knm_benchmark_summary_2026-05-05.json`. The same shared-workstation caveat applies; this table is a comparator framework, not yet a final language shootout.

project environment is installed. All three runs report matching checksums for the three language kernels, which makes the timings meaningful as implementation comparisons rather than unrelated algorithms.

Cross-machine K_{ij} construction comparison. The generated 16-oscillator cross-machine benchmark is kept inline rather than as a floating table so the machine rows remain adjacent to the explanatory text.

Machine	Python median	Rust median	Rust/Python
Local	0.06631 ms	0.00150 ms	44.1×
ML350	0.07620 ms	0.00493 ms	15.5×
Vertex CPU	0.04705 ms	0.00260 ms	18.1×

The rows are generated from the remote K_{ij} benchmark JSONs and the combined methods benchmark summary JSON dated 2026-05-05. Each row uses the self-contained benchmark harness with 1000 iterations and matching Python/Rust/Go checksums; the Go timings are retained in the JSON artefacts but omitted from this compact manuscript table. The reported machines had non-zero load averages during measurement, so the table supports robustness of the ordering, not isolated microbenchmark claims.

VII. GPU-ASSISTED VALIDATION BENCHMARK

GPU acceleration is useful for a different part of the workflow than the Rust kernels. The K_{ij} construction kernel is small and allocation sensitive; moving it to a GPU would

n	Hilbert dim.	Vertex CPU median	T4 CUDA median
8	256	11.121 ms	0.176 ms
10	1024	3.557 ms	0.290 ms
11	2048	12.094 ms	0.905 ms

TABLE VII. Generated batched expectation benchmark from `gpu_benchmark_summary_vertex_t4_2026-05-05.json`. The workload uses batch size 64 and 20 timing repeats. The CPU column is the NumPy path inside the same Vertex worker; the CUDA column is the PyTorch CUDA path on one Tesla T4 using the `pytorch/pytorch:2.5.1-cuda12.4-cudnn9-runtime` container.

mostly benchmark transfer overhead. The GPU-relevant workload is batched classical validation: evaluating many state-vector energies or VQE candidate states against the same dense Hamiltonian. We therefore added a separate GPU harness for batched expectation values,

$$E_b = \langle \psi_b | H_{XY} | \psi_b \rangle, \qquad b = 1, \ldots, B, \tag{2}$$

with $B = 64$ random normalised complex state vectors.

Local GPU timings are not reported as valid because the local NVIDIA stack was inconsistent during this session: `nvidia-smi` failed with an NVML driver/library mismatch, and CUDA initialisation in the local CUDA-enabled PyTorch environment failed with error 804. Instead, the valid GPU artefact is the Vertex AI Tesla T4 run using `pytorch/pytorch:2.5.1-cuda12.4-cudnn9-runtime`.

These results support a hybrid implementation strategy. Rust is used for low-latency CPU kernels and deterministic orchestration hot paths; GPU acceleration is used for batched dense linear algebra in classical validation and VQE scoring. QPU time is reserved for hardware-noise questions that cannot be answered by classical simulation alone.

VIII. HARDWARE WORKFLOW AND REPRODUCIBILITY

The same repository contains the raw-count hardware workflow used for the DLA-parity paper. That campaign is not the central result of the present methodology paper, but it demonstrates why the software architecture is useful: each promoted hardware dataset

Ansatz	Raw energy	Full-readout mitigated energy
K_{ij}-informed	−6.1938	−6.2021
`TwoLocal`	−6.0733	−6.0805
`EfficientSU2`	−6.0461	−6.0539

TABLE VIII. IBM Marrakesh ansatz-validation lane from the 2026-05-20 analysis JSON listed in the data-availability section. The result is a small hardware ordering check, not a hardware convergence proof. It supports the narrower statement that the topology-informed ansatz energy ordering survived one pinned IBM backend/layout execution and full 16-state readout-mitigation replay.

carries raw counts, circuit metadata, IBM job identifiers, SHA256 hashes, and reproducer scripts. This paper should cite the DLA-parity preprint as an application case study, while keeping the main methodological claim focused on the toolchain and benchmarking discipline.

For the methods ansatz claim itself, the 2026-05-20 revision adds a bounded hardware validation lane on `ibm_marrakesh`. The lane used the same $n = 4$ ansatz families and fixed seed as the convergence artefact, measured the Z, X, and Y global bases for each ansatz, and included a full 16-state computational-basis readout calibration. The job `d86u48op0eas73dlfqs0` executed 25 circuits at 1024 shots each on physical qubits $[7, 17, 6, 8]$. The calibration assignment matrix was well-conditioned, with condition number 1.059.

To address the small-width limitation directly, we also submitted the same ansatz-validation machinery at $n = 6$ and $n = 8$ on `ibm_fez` and `ibm_marrakesh`. Each larger lane measured the same three ansatz families and included the complete 2^n computational-basis readout calibration. These runs validate the packaging, retrieval, and readout inversion path at larger width, but they do not preserve the $n = 4$ topology-informed ranking. Table IX reports the full-readout-mitigated energies. In all four larger lanes, `EfficientSU2` has the lowest measured energy. This is a useful negative methods result: the topology-informed ansatz remains a good small-system design signal, but the larger hardware lanes show that ansatz ranking is sensitive to width, backend, layout, and noise.

The larger-width hardware lanes also give a useful negative design rule. The topology-

Backend	n	Seed	K_{ij}-informed	`TwoLocal`	`EfficientSU2`
`ibm_fez`	6	11	−9.1343	−7.9625	−10.0840
`ibm_fez`	8	11	−6.6274	−3.6932	−9.7473
`ibm_fez`	8	23	−6.7392	−5.1422	−10.0636
`ibm_marrakesh`	6	11	−10.0442	−9.9373	−10.3118
`ibm_marrakesh`	8	11	−11.3559	−10.2083	−11.9303
`ibm_marrakesh`	8	23	−11.4752	−10.6883	−11.9551

TABLE IX. Larger-width IBM ansatz-validation lanes submitted on 2026-05-20. Entries are full-readout-mitigated energy estimates from the retrieved raw counts. The seed-23 repeat at $n = 8$ preserves the larger-width `EfficientSU2`-lowest ordering on both backends. The result is deliberately reported as ansatz-sensitivity evidence: the larger hardware lanes validate the workflow at $n = 6, 8$ but do not promote the topology-informed ansatz as hardware-general superior.

informed ansatz is the best family in the original small $n = 4$ screening, but `EfficientSU2` gives the lowest measured energy in the $n = 6, 8$ hardware lanes and in the seed-23 $n = 8$ repeats. A plausible mechanism is that the topology-informed circuit encodes non-native graph structure whose routed two-qubit gates and calibration-window sensitivity outweigh its variational inductive bias on current heavy-hex hardware. The paper therefore promotes the ansatz path as a reproducible compiler and benchmarking method, not as a universal hardware ansatz-selection rule.

IX. LIMITATIONS

The benchmark results are deliberately bounded. First, the CPU timings are opportunistic shared-machine timings. They record machine metadata and load averages, but do not pin CPU affinity, clock governor, cache state, or thermal state. They are sufficient to document the pipeline and compare implementation ordering; they are not a final language shootout.

Second, the VQE comparison is small. The $n = 4$ topology-informed ansatz result is a useful methods signal, but it should not be interpreted as a universal optimisation theorem. The original fixed 80-iteration COBYLA budget is a controlled screening budget, not proof

Workflow component	Interpretation
K_{ij} construction	Rust/PyO3 kernel acceleration reduces repeated coupling-graph construction cost during sweeps and artefact regeneration.
Hamiltonian/ansatz export	Python/Qiskit object construction remains part of the cost and limits end-to-end speedup.
Optimiser evaluations	COBYLA and state-vector evaluations dominate VQE screening once the Hamiltonian is built.
Transpilation/provider execution	Hardware lanes are governed by circuit depth, backend routing, shot budget, queueing, and calibration windows.

TABLE X. End-to-end placement of the Rust kernel in the methods workflow. The kernel speedups are real and useful, but they are deliberately reported as workflow-component gains rather than full-stack VQE speedups.

of convergence; the 320-evaluation extension improves traceability for the representative case but still covers only a small Hamiltonian and three seeds. Larger ansatz matrices, more seeds, and additional Hamiltonian families remain necessary before promoting a broad ansatz-selection rule. The IBM observable comparisons in Tables VIII and IX are hardware sanity checks for the methods workflow, not backend-general ansatz-superiority evidence.

Third, the GPU benchmark is a classical-validation workload, not a claim that the full Qiskit circuit-construction path is GPU accelerated. Circuit construction and transpilation remain mostly Python/object-graph workloads. The GPU contribution is batched dense linear algebra.

Finally, the hardware experiments that motivated this pipeline remain small-N NISQ experiments. The software stack improves reproducibility and throughput, but it does not by itself establish quantum advantage. At the time of the reported hardware campaigns, authenticated live QPU access for the programme was limited to IBM Quantum hardware. The package includes provider-neutral execution and artefact-packaging surfaces for additional gate-model, trapped-ion, neutral-atom, photonic, annealing, analogue, and simulator targets, but these routes should be read as readiness infrastructure, not as live non-IBM hardware evidence. The intended next use of the methods stack is to rerun the same pre-

registered payloads on additional hardware families when compute allocations and provider access become available.

X. DATA AND CODE AVAILABILITY

The public repository is https://github.com/anulum/scpn-quantum-control. The methods-paper benchmark artefacts are under `data/rust_vqe_methods/`. The primary generated summaries used in this manuscript are:

- `rust_core_benchmark_summary_2026-05-05.json`,
- `ansatz_benchmark_summary_2026-05-05.json`,
- `vqe_benchmark_summary_2026-05-05.json`,
- `multilang_knm_benchmark_summary_2026-05-05.json`,
- `remote_knm_benchmark_aaarthuus_local_2026-05-05.json`,
- `remote_knm_benchmark_ml350_2026-05-05.json`,
- `remote_knm_benchmark_vertex_n1_standard_4_2026-05-05.json`,
- `gpu_benchmark_summary_vertex_t4_2026-05-05.json`,
- `ansatz_scaling_tn_summary_2026-05-05.json`,
- `ansatz_scaling_summary_2026-05-05.csv`,
- `tn_truncation_summary_2026-05-05.csv`,
- `combined_methods_benchmark_summary_2026-05-05.json`,
- `vqe_convergence_methods_2026-05-20.json`,
- `vqe_convergence_aggregate_2026-05-20.csv`,
- `ansatz_scaling_error_bars_2026-05-20.csv`,
- `ansatz_ibm_validation_readiness_ibm_marrakesh_20260520T163234Z.json`,

- `ansatz_ibm_validation_submission_ibm_marrakesh_20260520T163234Z.json`,
- `ansatz_ibm_validation_raw_counts_ibm_marrakesh_20260520T163812Z.json`,
- `ansatz_ibm_validation_analysis_ibm_marrakesh_20260520T163812Z.json`,
- `ansatz_ibm_validation_analysis_ibm_fez_20260520T195028Z.json`,
- `ansatz_ibm_validation_analysis_ibm_fez_20260520T195040Z.json`,
- `ansatz_ibm_validation_analysis_ibm_marrakesh_20260520T195051Z.json`,
- `ansatz_ibm_validation_analysis_ibm_marrakesh_20260520T195102Z.json`,
- `ansatz_ibm_validation_analysis_ibm_fez_20260520T201721Z.json`, and
- `ansatz_ibm_validation_analysis_ibm_marrakesh_20260520T201741Z.json`.

The latest combined summary hashes at the time of this draft are `593330a1dd19f495b899be1031ebe3dd4caa07171053aa376c2f761e557c1428` for JSON and `e69b94df590ff06708b3b21245864f74c3df630b514254526dc6c4af3fe24c2f` for CSV. The corresponding generator scripts are `scripts/benchmark_rust_core_methods.py`, `scripts/benchmark_ansatz_methods.py`, `scripts/benchmark_vqe_methods.py`, `scripts/benchmark_multilang_knm_methods.py`, `scripts/benchmark_remote_knm_machine.py`, `scripts/benchmark_gpu_methods.py`, and `scripts/summarise_rust_vqe_method_artifacts.py`. The 2026-05-20 extension scripts are `scripts/benchmark_vqe_convergence_methods.py`, `scripts/submit_methods_ansatz_ibm_validation.py`, and `scripts/retrieve_methods_ansatz_ibm_validation.py`.

XI. CONCLUSION

We have presented a conservative methodology paper around `scpn-quantum-control`: a Rust-accelerated, topology-informed workflow for Kuramoto–XY quantum simulation, VQE benchmarking, and NISQ hardware reproducibility. The strongest near-term contribution is not an unqualified speedup headline, but a transparent engineering stack that connects physical coupling graphs to circuits, benchmark artefacts, hardware raw counts, and reproducible analysis. The generated artefacts show that Rust gives consistent low-latency CPU

benefits for the coupling-matrix kernel across three CPU sources, that topology-informed ansatz design improves the tested small VQE case, and that Vertex T4 GPU acceleration is useful for batched classical expectation evaluation.

---

[1] M. Šotek, *`scpn-quantum-control`: Quantum-Native SCPN Phase Dynamics and Control*, version 0.9.7, Zenodo DOI: 10.5281/zenodo.18821929 (2026).

[2] M. Šotek, "Replicated hardware observation of parity-sector and excitation-number correlated decoherence asymmetry in the XY Hamiltonian," Zenodo preprint DOI: 10.5281/zenodo.20382000 (2026).

[3] M. Šotek, *DLA parity hardware validation package for heterogeneous Kuramoto–XY circuits*, Zenodo dataset and public repository artefacts (2026), Zenodo record 19098792, https://zenodo.org/records/19098792.

[4] Qiskit contributors, *Qiskit: An Open-source Framework for Quantum Computing*.

[5] A. Peruzzo *et al.*, "A variational eigenvalue solver on a photonic quantum processor," Nature Communications **5**, 4213 (2014).

[6] A. Kandala *et al.*, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," Nature **549**, 242–246 (2017).

[7] Y. Kuramoto, *Chemical Oscillations, Waves, and Turbulence* (Springer, Berlin, 1984).