

Gaps in runtime-based quantum supremacy claims

Lessons from annealing, oracle algorithms, and hybrid optimization

Łukasz Paweła

Institute of Theoretical and Applied Informatics,
Polish Academy of Sciences

Quantumz.io



03.12.2025

Outline

- 1 Motivation and definitions
- 2 Simulated bifurcation machine (SBM)
- 3 Case atudy I: Ising/QUBO on D-Wave annealers
- 4 Case study II: restricted Simon's problem
- 5 Case study III: hybrid BF-DCQO optimization
- 6 General lessons from both papers
- 7 Summary

Motivation and definitions

From quantum advantage to runtime supremacy

- **Quantum advantage / supremacy:** outperforming classical computation for a well-defined task.
- Two worlds:
 - *Complexity-theoretic* speedups (e.g., Simon, Shor): asymptotic separation in query or gate complexity.
 - *Experimental* claims on NISQ hardware: “this device solved X faster than any classical machine.”
 - *Experimental scaling*: “this device’s runtime scales better than classical methods on problem family Y.”
- NISQ era: small, noisy devices, strong hardware constraints (topology, coherence, calibration).

Central question

Central question

Do recent experimental “quantum supremacy/advantage” claims actually translate into a shorter *wall-clock runtime* than strong classical baselines?

Why focus on runtime?

- In practice, users care about:
 - How long until I get a solution of quality ε ?
 - How much compute / energy did that require?

Why focus on runtime?

- In practice, users care about:
 - How long until I get a solution of quality ε ?
 - How much compute / energy did that require?
- Complexity separations may appear only for unrealistically large instances.
- Quantum and classical devices operate on very different timescales:
 - Clock rates, readout times, thermalization, queueing, compilation.
- A realistic “supremacy” statement should therefore be:

Runtime-based supremacy

For a clearly specified problem family and quality target, the best known quantum implementation achieves a shorter *end-to-end* runtime than any known classical method.

The time-to- ε metric

- For approximate optimization, both papers use the *time-to- ε* metric:

$$TT_{\varepsilon} = t_f \cdot \frac{\log(1 - 0.99)}{\log(1 - p_{E \leq E_0 + \varepsilon | E_0|})}$$

- t_f – time for one run of the solver (this is the contentious piece!).
- $p_{E \leq E_0 + \varepsilon | E_0|}$ – success probability of achieving energy within fractional gap ε of the ground state E_0 (or best-known energy).
- $TT_{\varepsilon} \approx$ expected time to obtain a “good enough” solution with 99% confidence.
- Core message: *the definition of t_f determines whether “speedup” survives.*

What counts as runtime?

Classical (CPU/GPU)

- Algorithm loop on CPU or GPU.
- Overheads:
 - Data loading, GPU transfers.
 - Hyperparameter search.
- Often: total time \approx compute time + small, weakly N -dependent constant.

What counts as runtime?

Classical (CPU/GPU)

- Algorithm loop on CPU or GPU.
- Overheads:
 - Data loading, GPU transfers.
 - Hyperparameter search.
- Often: total time \approx compute time + small, weakly N -dependent constant.

Quantum annealer

- Embedding + programming.
- **Annealing** (user-set duration).
- Readout per shot.
- Thermalization / reset.
- Cloud access overheads, queuing.

Gate-based device

- Transpilation.
- Circuit upload / scheduling.
- Shots (circuit repetitions).
- Readout and qubit reset.

What counts as runtime?

Classical (CPU/GPU)

- Algorithm loop on CPU or GPU.
- Overheads:
 - Data loading, GPU transfers.
 - Hyperparameter search.
- Often: total time \approx compute time + small, weakly N -dependent constant.

Pitfall

Many “supremacy” papers count only the nicest-looking piece (annealing time, number of oracle calls, shot count) and ignore everything else.

Quantum annealer

- Embedding + programming.
- **Annealing** (user-set duration).
- Readout per shot.
- Thermalization / reset.
- Cloud access overheads, queuing.

Gate-based device

- Transpilation.
- Circuit upload / scheduling.
- Shots (circuit repetitions).
- Readout and qubit reset.

Simulated bifurcation machine (SBM)

Simulated bifurcation machine (SBM) as a classical baseline

- SBM is a **quantum-inspired** classical solver based on nonlinear Hamiltonian dynamics that approximate networks of Kerr parametric oscillators (KPOs).
- Instead of thermal noise (as in SA / PT-ICM), SBM uses **deterministic but chaotic** trajectories with bifurcations that encode low-energy Ising states.
- Algorithm is **massively parallel**: many independent replicas can be integrated simultaneously on GPUs or other accelerators.
- Gives state-of-the-art performance on Ising benchmarks,

$$E = \sum_{i < j} J_{ij} s_i s_j + \sum_i h_i s_i,$$

SBM dynamics: from KPOs to classical ODEs

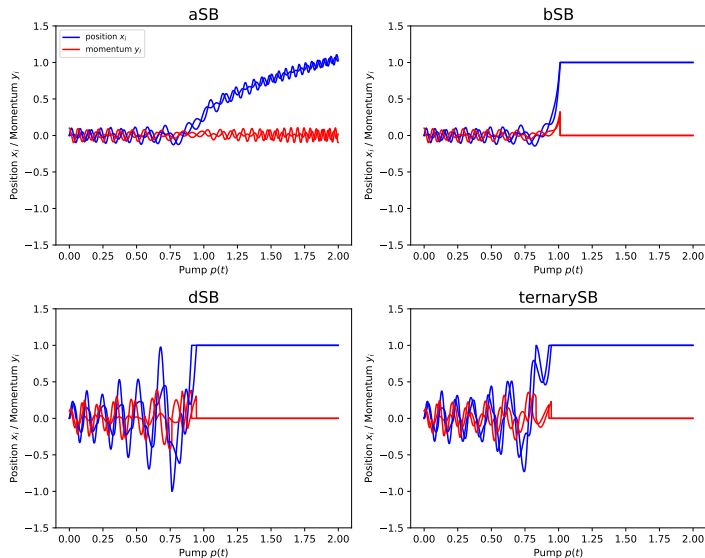
- Start from a network of Kerr nonlinear oscillators with parametric drive $p(t)$ and couplings J_{ij} that implement the Ising cost function.
- Under a coherent-state (mean-field) approximation and suitable dequantization, one obtains classical phase-space variables (x_i, y_i) obeying Hamiltonian-like dynamics:

$$\dot{x}_i = \Delta \cdot y_i, \quad \dot{y}_i = [-\Delta + p(t)]x_i + \xi_0 \left(\sum_j J_{ij} f(x_j) + h_i \right),$$

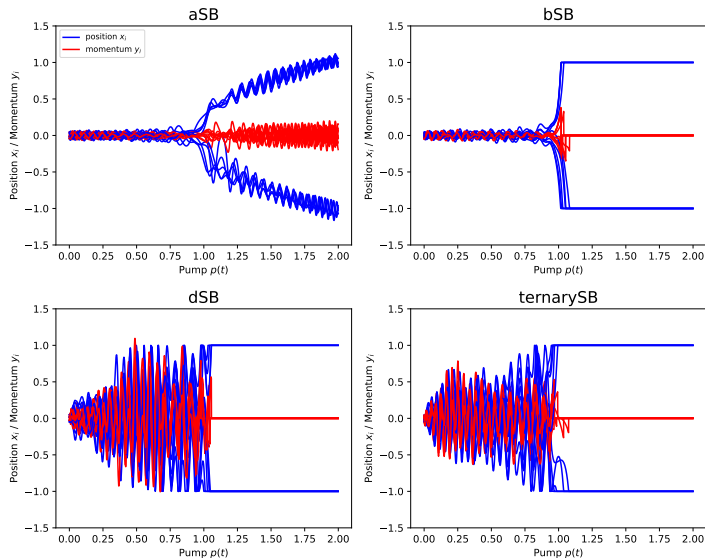
where:

- Δ – effective detuning, $p(t)$ – slowly ramped pump (annealing schedule),
- ξ_0 – coupling scale,
- $f(x)$ – discretization nonlinearity (see next slide).
- As $p(t)$ increases, the system undergoes a series of **bifurcations**; trajectories are attracted toward configurations whose signs $\text{sgn}(x_i)$ approximate low-energy Ising spin assignments.

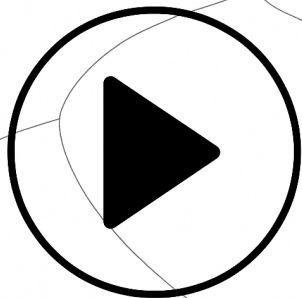
Trajectories for simple ferromagnetic model



Trajectories for simple spin glass model



Animation of simple dynamics



Click to play

Nonlinearity and discretization in SBM

- Two key ingredients make SBM numerically stable and efficient:
 - ① **Perfectly inelastic walls** at $|x_i| = 1$ (or $\sqrt{2}$ in some parametrizations):
 - If $|x_i|$ exceeds the wall, set $x_i \leftarrow \text{sgn}(x_i)$ and $y_i \leftarrow 0$.
 - Prevents trajectories from diverging and keeps the dynamics in a compact region.

Nonlinearity and discretization in SBM

- Two key ingredients make SBM numerically stable and efficient:
 - ① **Perfectly inelastic walls** at $|x_i| = 1$ (or $\sqrt{2}$ in some parametrizations):
 - If $|x_i|$ exceeds the wall, set $x_i \leftarrow \text{sgn}(x_i)$ and $y_i \leftarrow 0$.
 - Prevents trajectories from diverging and keeps the dynamics in a compact region.
 - ② **Ternary discretization** of the interaction term:

$$f(x) = \begin{cases} 0, & |x| \leq p(t), \\ \text{sgn}(x), & |x| > p(t), \end{cases}$$

with a threshold $\Delta(t)$ that shrinks during the evolution.

Nonlinearity and discretization in SBM

- Two key ingredients make SBM numerically stable and efficient:
 - ① **Perfectly inelastic walls** at $|x_i| = 1$ (or $\sqrt{2}$ in some parametrizations):
 - If $|x_i|$ exceeds the wall, set $x_i \leftarrow \text{sgn}(x_i)$ and $y_i \leftarrow 0$.
 - Prevents trajectories from diverging and keeps the dynamics in a compact region.
 - ② **Ternary discretization** of the interaction term:

$$f(x) = \begin{cases} 0, & |x| \leq p(t), \\ \text{sgn}(x), & |x| > p(t), \end{cases}$$

with a threshold $\Delta(t)$ that shrinks during the evolution.

- Early in the run, many components satisfy $|x_i| \leq p(t)$, so they are effectively switched off ($f(x_i) = 0$); only strong directions in the landscape drive the dynamics.

Nonlinearity and discretization in SBM

- Two key ingredients make SBM numerically stable and efficient:
 - ① **Perfectly inelastic walls** at $|x_i| = 1$ (or $\sqrt{2}$ in some parametrizations):
 - If $|x_i|$ exceeds the wall, set $x_i \leftarrow \text{sgn}(x_i)$ and $y_i \leftarrow 0$.
 - Prevents trajectories from diverging and keeps the dynamics in a compact region.
 - ② **Ternary discretization** of the interaction term:

$$f(x) = \begin{cases} 0, & |x| \leq p(t), \\ \text{sgn}(x), & |x| > p(t), \end{cases}$$

with a threshold $\Delta(t)$ that shrinks during the evolution.

- Early in the run, many components satisfy $|x_i| \leq p(t)$, so they are effectively switched off ($f(x_i) = 0$); only strong directions in the landscape drive the dynamics.
- As $\Delta(t)$ decreases, more variables “join” the dynamics, gradually refining the spin configuration.

Nonlinearity and discretization in SBM

- Two key ingredients make SBM numerically stable and efficient:
 - ① **Perfectly inelastic walls** at $|x_i| = 1$ (or $\sqrt{2}$ in some parametrizations):
 - If $|x_i|$ exceeds the wall, set $x_i \leftarrow \text{sgn}(x_i)$ and $y_i \leftarrow 0$.
 - Prevents trajectories from diverging and keeps the dynamics in a compact region.
 - ② **Ternary discretization** of the interaction term:

$$f(x) = \begin{cases} 0, & |x| \leq p(t), \\ \text{sgn}(x), & |x| > p(t), \end{cases}$$

with a threshold $\Delta(t)$ that shrinks during the evolution.

- Early in the run, many components satisfy $|x_i| \leq p(t)$, so they are effectively switched off ($f(x_i) = 0$); only strong directions in the landscape drive the dynamics.
- As $\Delta(t)$ decreases, more variables “join” the dynamics, gradually refining the spin configuration.
- Final spin assignment: $s_i = \text{sgn}(x_i)$; corresponding Ising / QUBO energy is evaluated and the best replica is kept.

SBM as an algorithm

Inputs: Ising instance (J_{ij}, h_i) and hyperparameters $(\Delta t, N_s, N_r)$.

- ① Good hyperparameter guess: $\Delta = 1$, $p(t)$ linearly ramped from 0 to 1, $\xi_0 = \frac{\sqrt{N-1}}{\sum_{ij} J_{ij}^2}$
- ② For each replica $r = 1, \dots, N_r$ (run in parallel on GPU):
 - ① Initialize $x_i^{(r)}, y_i^{(r)}$ randomly in $[-1, 1]$.
 - ② For $k = 1, \dots, N_s$ time steps:
 - Update $(x^{(r)}, y^{(r)})$ via a discretized step of the ODEs,
 - Apply walls $|x_i| > 1 \Rightarrow x_i \leftarrow \text{sgn}(x_i), y_i \leftarrow 0$,
 - Use current $p(t_k)$ and $\Delta(t_k)$ in the update.
 - ③ Decode spins $s_i^{(r)} = \text{sgn}(x_i^{(r)})$ and compute Ising energy $E^{(r)}$.
- ③ Return the best replica r^* with lowest $E^{(r^*)}$.
- Time-to- ε metric TT_ε is computed from the empirical success probability of reaching energy within an ε -gap of the ground state and the per-run time, exactly as for the quantum solvers.

SBM runtime vs quantum annealing

- For SBM on GPU, runtime naturally splits into:

$$t_f^{\text{tot}} = t_f^{\text{GPU}} + t_f^{\text{overhead}},$$

where t_f^{GPU} is pure integration time and t_f^{overhead} covers CPU–GPU transfers, memory management and bookkeeping.

- In practice:
 - t_f^{GPU} dominates for larger problem sizes; t_f^{overhead} is roughly a *weakly size-dependent offset*.
 - Therefore, changing the precise runtime definition (GPU-only vs total) does **not** qualitatively change TT_ϵ scaling.
- In contrast, for D-Wave QAC:
 - Readout time per sample $\sim 200 \mu\text{s}$ dominates over annealing time ($0.5\text{--}27 \mu\text{s}$).
 - Including programming, readout and thermalization makes the median TT_ϵ nearly flat in N over the studied range (no visible scaling advantage).

Case study I: Ising/QUBO on D-Wave annealers

Original claim: scaling advantage in Ising/QUBO optimization

- PRL 134, 160601 (Munoz-Bauza & Lidar):
 - Approximate QUBO/Ising problems on D-Wave Advantage 4.1 with QAC encoding.
 - Couplings from Sidon-28 set, logical patch sizes $L = 5 \dots 15$ ($N = 142 \dots 1322$ logical spins).
 - Figure of merit: TT_ε with relatively loose ε ($\sim 1\%$).
 - Runtime proxy: *annealing time* per sample, $\tau \in [0.5, 27] \mu\text{s}$.

Original claim: scaling advantage in Ising/QUBO optimization

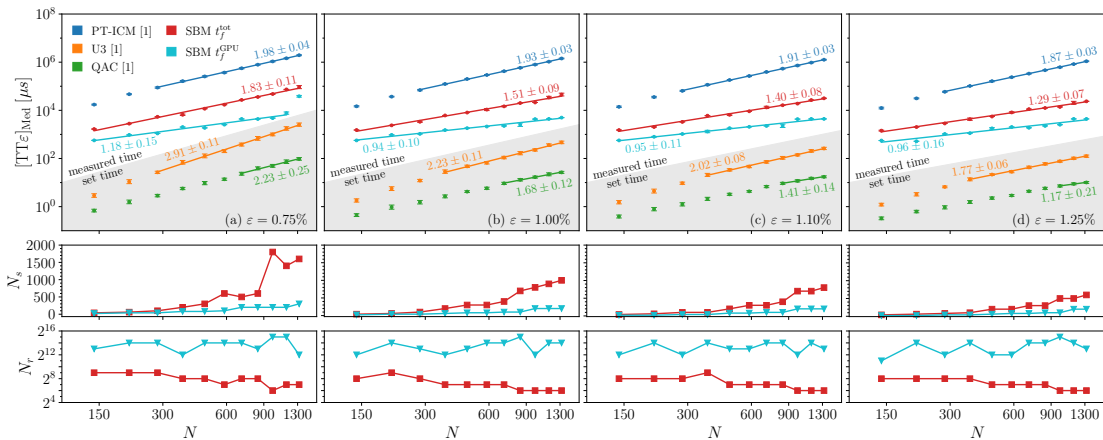
- PRL 134, 160601 (Munoz-Bauza & Lidar):
 - Approximate QUBO/Ising problems on D-Wave Advantage 4.1 with QAC encoding.
 - Couplings from Sidon-28 set, logical patch sizes $L = 5 \dots 15$ ($N = 142 \dots 1322$ logical spins).
 - Figure of merit: TT_ε with relatively loose ε ($\sim 1\%$).
 - Runtime proxy: *annealing time* per sample, $\tau \in [0.5, 27] \mu\text{s}$.
- Comparison against classical baselines:
 - PT-ICM (Parallel Tempering with Isoenergetic Cluster Moves).
 - U3 heuristic.

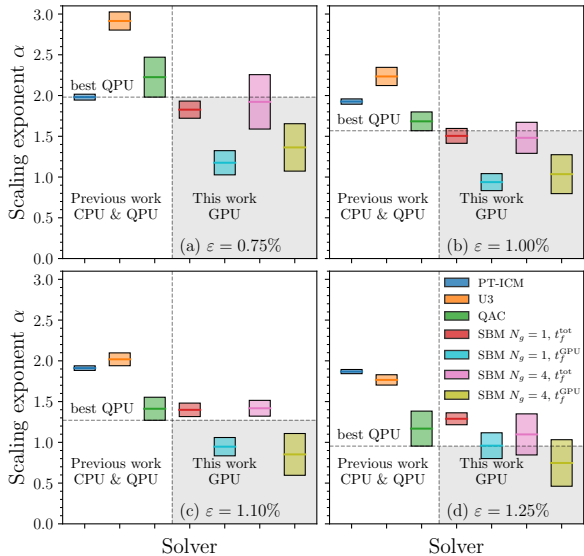
Original claim: scaling advantage in Ising/QUBO optimization

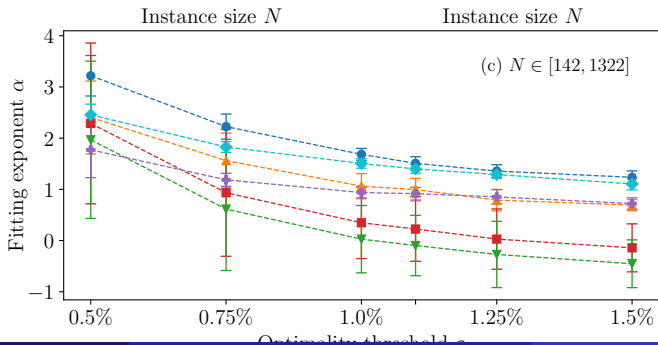
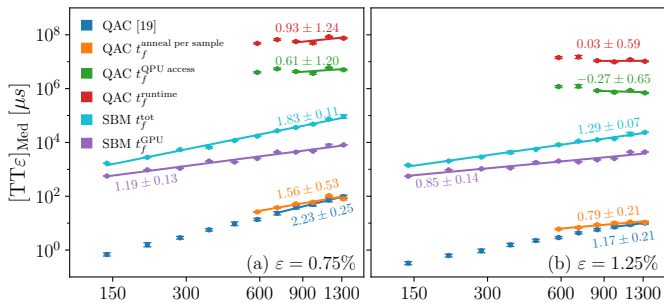
- PRL 134, 160601 (Munoz-Bauza & Lidar):
 - Approximate QUBO/Ising problems on D-Wave Advantage 4.1 with QAC encoding.
 - Couplings from Sidon-28 set, logical patch sizes $L = 5 \dots 15$ ($N = 142 \dots 1322$ logical spins).
 - Figure of merit: TT_ϵ with relatively loose ϵ ($\sim 1\%$).
 - Runtime proxy: *annealing time* per sample, $\tau \in [0.5, 27] \mu\text{s}$.
- Comparison against classical baselines:
 - PT-ICM (Parallel Tempering with Isoenergetic Cluster Moves).
 - U3 heuristic.
- Result: QAC showed better apparent scaling of TT_ϵ with N than PT-ICM, interpreted as a *quantum scaling advantage*.

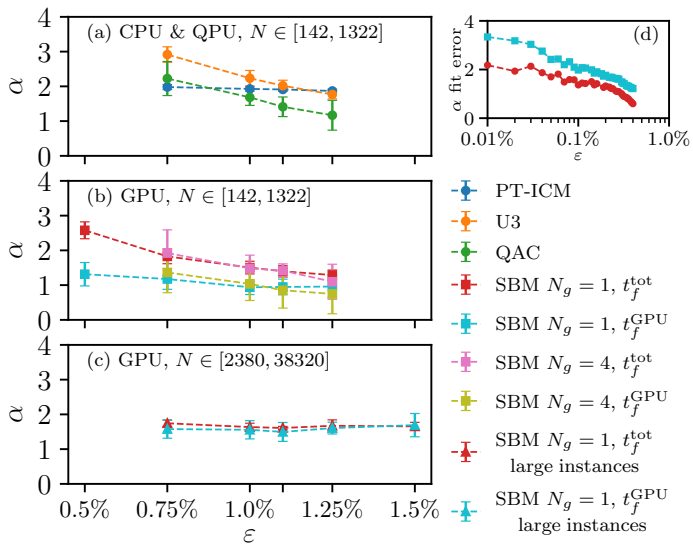
Re-analysis: measuring full quantum runtime

- Repeat experiments on the same Sidon-28 instances and define several runtime variants:
 - ① t_f^{anneal} – annealing time per sample (as in the PRL).
 - ② $t_f^{\text{QPU access}}$ – time reported by D-Wave for QPU access: programming + annealing + readout + thermalization.
 - ③ t_f^{runtime} – full cloud runtime including API overheads.
- At the same time they introduce a strong classical baseline:
 - **Simulated Bifurcation Machine (SBM)** – a chaotic, Hamiltonian-inspired algorithm, highly parallel on GPUs.
 - Measure both:
 - t_f^{GPU} – pure GPU compute time.
 - t_f^{tot} – total host + GPU runtime including data transfers and hyperparameter search.









Result: readout dominates, scaling flattens

Gap #1: Overhead-Only Scaling

The scaling advantage comes from plotting TT_ϵ vs N using a user-chosen control parameter (annealing time) instead of the actual runtime. Once full QPU time is used, no advantage remains.

Gap #2: Non-competitive classical baseline

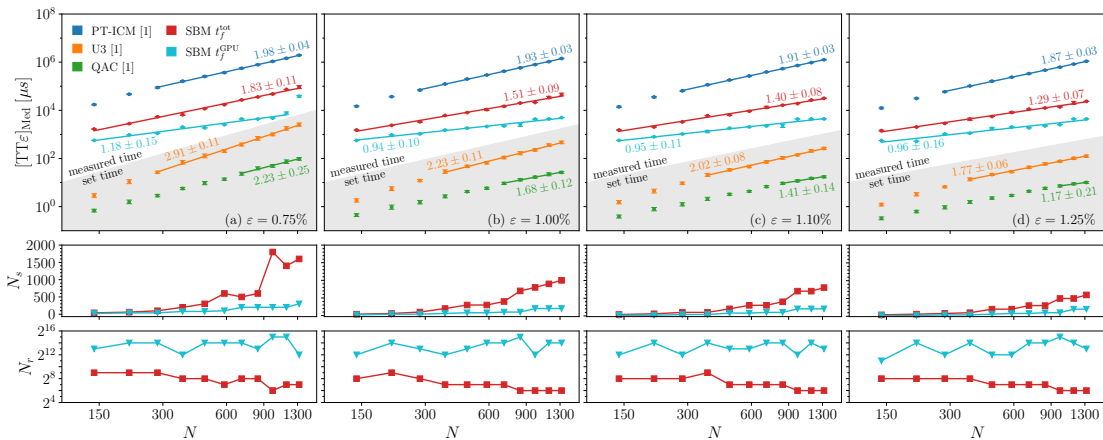
A quantum-inspired classical solver (SBM) running on commodity GPUs eliminates the claimed scaling gap once a fair runtime metric is used. In most cases, SBM outperforms QAC even when including classical overheads, giving unfair advantage to the quantum side.

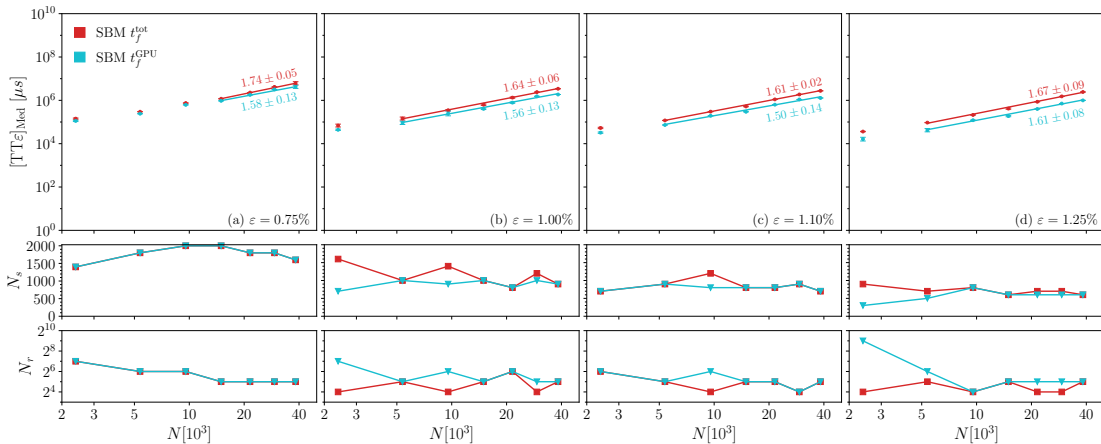
Finite-size effects and asymptotic scaling

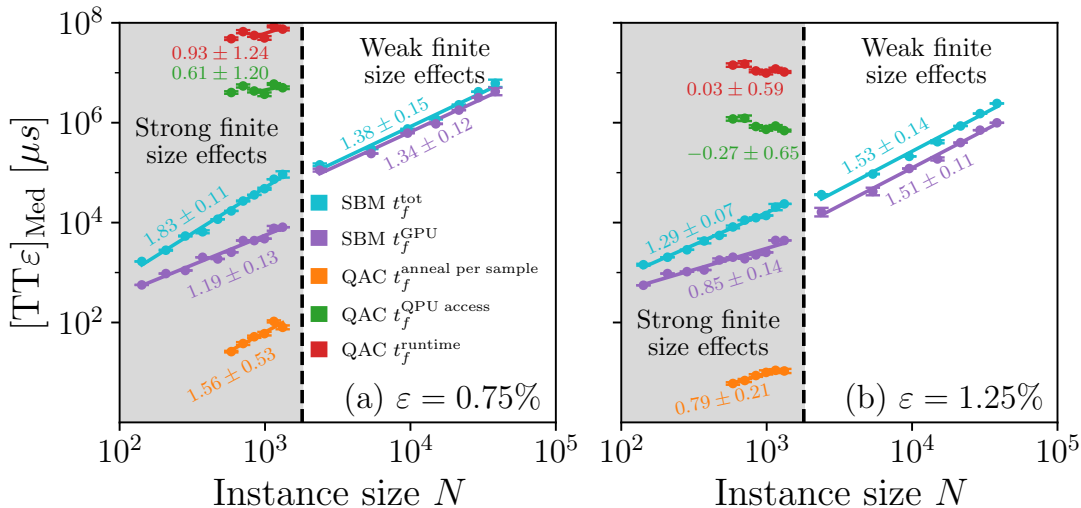
- For small N (hundreds to $\sim 10^3$ spins), overheads can dominate runtime on both quantum and classical accelerators.
- Studying SBM on much larger Sidon-28 instances:
 - N up to $\sim 4 \times 10^4$ logical spins.
 - Examine TT_ε scaling and fitted exponent α across different ε .
- Observations:
 - Overhead contribution becomes negligible; $t_f^{\text{GPU}} \approx t_f^{\text{tot}}$.
 - Scaling exponent α stabilizes around ~ 1.5 – 1.7 , with weak dependence on ε .
 - In contrast, current annealers cannot reach this size regime at all.
- Takeaway:

Gap #3: Small-system scaling is misleading

In the size window accessible to today's quantum hardware, we are far from the asymptotic regime; any “superior scaling” claim must be treated with extreme caution.







Case study II: restricted Simon's problem

Simon's problem and oracle query complexity

- **Simon's problem:** Given a 2-to-1 function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ with hidden XOR period $b \neq 0$, find b .

Simon's problem and oracle query complexity

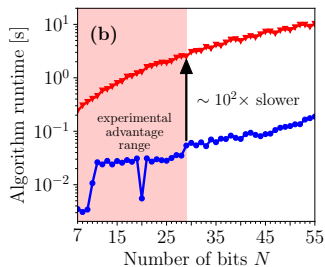
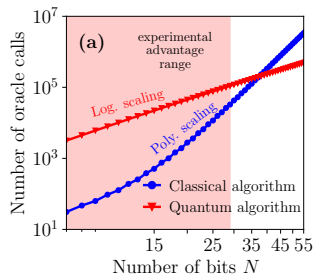
- **Simon's problem:** Given a 2-to-1 function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ with hidden XOR period $b \neq 0$, find b .
- In the oracle model:
 - Quantum algorithm needs $O(n)$ oracle queries.
 - Best classical randomized algorithms require $\Omega(2^{n/2})$ queries.
- **Restricted Simon's problem** (PRX 15, 021082): period b is promised to have Hamming weight $\leq w$.
 - Define a score function that penalizes random guessing.
 - Show provable *polylogarithmic* scaling of that score in the quantum case vs *polynomial* scaling classically.
- Experiment: implement the oracle on IBM hardware using a compiler that splits work between classical and quantum parts.

How we re-test the claim

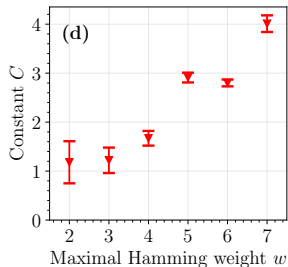
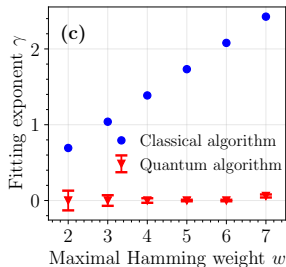
- Goal: check whether the *query complexity* advantage yields a *runtime* advantage.
- Steps:
 - Implement a classical brute-force solver on a single GPU.
 - Same family of oracle functions f_b as in the IBM experiment.
 - Time measurement includes only oracle evaluations (other classical work is the same in both cases).

How we re-test the claim

- Goal: check whether the *query complexity* advantage yields a *runtime* advantage.
- Steps:
 - Implement a classical brute-force solver on a single GPU.
 - Same family of oracle functions f_b as in the IBM experiment.
 - Time measurement includes only oracle evaluations (other classical work is the same in both cases).
 - Reconstruct quantum circuits with Qiskit:
 - Use the same compiler's oracle construction.
 - Transpile to IBM Brisbane topology (optimization level 3).
 - Choose number of shots from the analytic bounds in the PRX paper.
 - Use Qiskit timing estimates to get runtime per shot and total runtime.
- Only the oracle execution is attributed to runtime in both cases—favourable to the quantum side.



$$\text{NTS} \sim e^{C(\log_2 N_w)^\gamma [\log(1+\log_2 N_w)]^{(1-\gamma)}}$$



Oracle calls (query complexity)

- For $w = 7$ and N up to 29 bits:
 - Quantum algorithm: total number of oracle calls scales polylogarithmically in the number of possible periods N_w .
 - Classical brute-force: number of oracle calls grows roughly exponentially.
- This reproduces the expected *exponential separation in the oracle model*.

Runtime

- Despite needing more queries, the classical solver's oracle calls are cheap and massively parallel on GPU.
- Quantum oracle calls involve deep-ish circuits, slower gates, and substantial per-shot overhead.

Key empirical point

For $N = 29$, $w = 7$:

- Classical runtime ≈ 0.035 s.
- Quantum runtime ≈ 2 s.

So the quantum experiment is about two orders of magnitude **slower**.

When would the quantum algorithm win?

- Fit scaling of the score function vs total number of possible periods N_w :
 - Quantum: polylogarithmic behaviour.
 - Classical: effective exponential behaviour.
- Convert to runtime scaling using timing models for both implementations.
- Prediction:
 - Runtime crossover (where quantum beats classical) occurs around $N \approx 60$ bits for the explored w .
 - But according to the original PRX study, noise already destroys the algorithm's correctness before that point.
- Interpretation:

Gap #4: Asymptotic vs practical regime

The asymptotic query-complexity advantage does *not* manifest as a runtime advantage on current or near-term hardware, because the relevant instance sizes are unattainable in the noise-limited regime.

Case study III: hybrid BF-DCQO optimization

The BF-DCQO hybrid algorithm

- Recent preprint (Chandarana *et al.*, 2025): *Bias-field Digitized Counter-Diabatic Quantum Optimization* (BF-DCQO).
- Problem class: higher-order unconstrained binary optimization (HUBO) with cubic terms:

$$P(s) = \sum_{i_1, \dots, i_N; i_1 + \dots + i_N \leq 3} J_{i_1 \dots i_N} s_{i_1} \cdots s_{i_N}.$$

- Algorithm structure:
 - ① Run classical simulated annealing (SA) once to produce an initial bitstring.
 - ② Use a shallow quantum circuit (digitized counter-diabatic schedule) seeded with this bitstring to explore a local neighbourhood.
 - ③ Run SA again, starting from the quantum output, to mitigate readout errors.
- Original claim:
 - BF-DCQO achieves a *runtime advantage* over SA and CPLEX on carefully constructed HUBO instances.

Runtime Definitions: Where Things Go Wrong

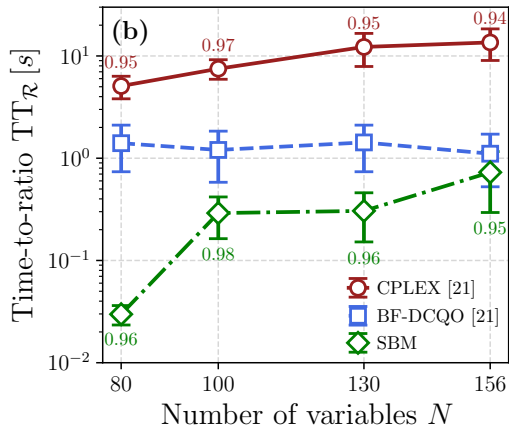
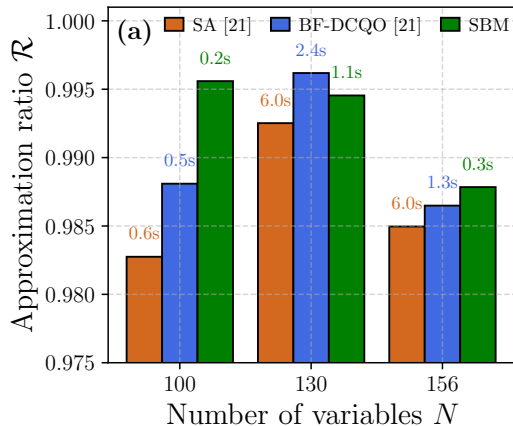
- Classical SA runtime:
 - Estimated from the time for a *single pass* multiplied by number of passes.
 - Overheads (initialization, memory, hyperparameter tuning) of order ~ 1.6 s *not* included in the reported times.
- Quantum runtime:
 - Assumed the device can execute 10^4 circuit repetitions per second.
 - Ignores:
 - Transpilation time.
 - Circuit upload / scheduling / queueing.
 - Readout and reset overheads.
- Hyperparameter tuning:
 - Algorithm is heuristic and hyperparameters are instance dependent.
 - Cost to tune them is not counted, neither for SA nor for the quantum part.
- Metric:
 - Use of “time-to-ratio” TTR, but without carefully accounting for stochastic variability and repetitions.

Gap #5: Incompatible Time Accounting

Classical and quantum components are timed in different ways, and major overheads are excluded from both. The resulting “runtime advantage” is therefore not robust.

Reproducing and extending the HUBO benchmarks

- We reconstruct the HUBO instances (aligned with IBM Heron's heavy-hex topology) from the published description and generate new random instances.
- We benchmark:
 - BF-DCQO (using data from the original paper).
 - SA tuned specifically for HUBO.
 - **Simulated Bifurcation Machine (SBM)** on GPU, after quadratizing HUBO to QUBO when needed.
- Metrics:
 - Approximation ratio $R = E/E_{GS}$.
 - TTR – time required to find an energy $\leq RE_{GS}$.
- Important subtlety:
 - Original BF-DCQO results often show performance for a *single best instance* or averages over only a few (e.g., 5) instances.
 - This enables manufacturing supremacy claims by cherry picking.

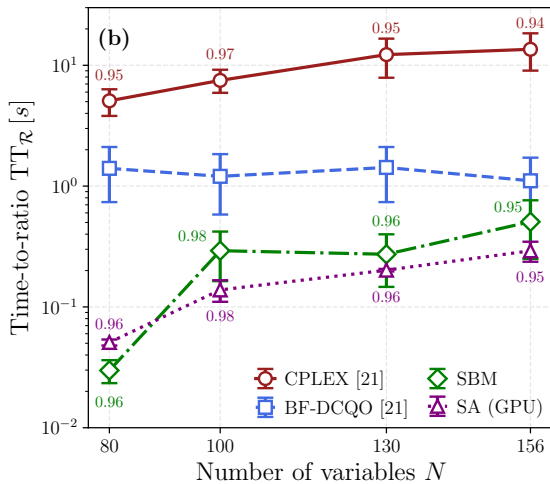
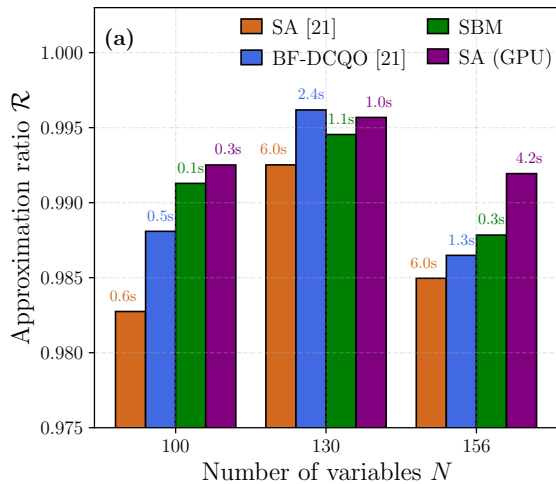


Results: SBM and optimized SA close the gap

- For the same HUBO instance families:
 - SBM consistently achieves equal or better approximation ratios in *less* time than BF-DCQO.
 - Optimized SA, when tuned fairly, also reaches comparable performance.
- Example (qualitative):
 - For $N = 100$, BF-DCQO's approximation ratio R is only $\Delta R \sim 0.002$ better than SBM,
 - but SBM's estimated TTR is about a factor of 2 smaller.
- Interpretation:

Gap #6: Classical Quantum-Inspired Solvers Were Ignored

Once a modern classical baseline leveraging GPUs and Hamiltonian dynamics is used, the BF-DCQO supremacy disappears, even under the original (optimistic) runtime model.



Embedding overheads on annealers

- As a further check, we embed the same HUBO-derived QUBO instances into D-Wave's Advantage2 1.6 (Zephyr topology).
- Steps:
 - Reduce HUBO to QUBO (introducing auxiliary variables).
 - Minor-embed QUBO onto Zephyr graph; this leads to chains of physical qubits representing each logical variable.
- Observations:
 - Chain-length distribution has a long tail—many very long chains.
 - Chain breaks and limited energy scales significantly degrade solution quality.
 - Even after cherry picking the best of many runs, D-Wave's approximation ratios are well below those of SBM and BF-DCQO.

Gap #7: Ignoring Embedding as a Cost

For non-native problem topologies, embedding overhead can completely wipe out a putative quantum advantage. Any realistic benchmarking must model this explicitly.

General lessons from both papers

Choosing a classical baseline

- A legitimate classical reference should:
 - ① Solve the *same* problem class natively (no artificial disadvantage through awkward reformulation).
 - ② Be competitive and up-to-date:
 - Exploit parallelism on GPUs, FPGAs, multi-core CPUs, wafer-scale chips etc.
 - Use problem-specific heuristics when appropriate.
 - ③ Be tuned and benchmarked using the *same* quality metric (TT_ϵ , TTR, approximation ratio).
 - ④ Have its own hyperparameter tuning costs included in the runtime budget.
- Examples from these studies:
 - PT-ICM is not embarrassingly parallel; SBM is.
 - CPLEX requires reformulation of HUBO into mixed-integer programs, inflating its runtime unfairly.

Small instances and overhead-dominated regimes

- For both annealers and gate-based devices, current qubit counts limit us to:
 - Hundreds of variables (gate-based).
 - A few thousand logical variables (annealers) for structured problems.
- In this regime:
 - Overheads (readout, thermalization, transpilation, queuing) can dominate runtime.
 - Fitted scaling exponents become extremely sensitive to the precise definition of t_f .
- Classical accelerators (GPUs, FPGAs) can also have nontrivial overhead, but:
 - For large enough problems, total time \approx compute time + nearly constant overhead, so scaling is robust.

Scaling claims must be anchored in an asymptotic regime

If the accessible instance sizes are too small to see stable scaling for state-of-the-art classical solvers, then they are too small to support a credible quantum scaling advantage.

A constructive outlook: where could quantum help?

- We look at 3D spin-glass instances where D-Wave exhibits good performance in a fast-annealing regime.
- When runtime is proxied by annealing time τ :
 - Quantum annealer shows better TT_ε scaling than SBM for $\varepsilon \lesssim 1\%$ on these structured instances.
- Once QPU access time or full runtime is used:
 - Scaling exponents become nearly zero; overheads erase the advantage.
- This suggests a potential path:

Hardware-level conditions for future advantage

- Reduce readout and thermalization times by (at least) one to two orders of magnitude.
- Increase qubit counts and connectivity to reach sizes where classical methods genuinely struggle.
- Maintain (or improve) solution quality in that regime.

Summary

What are the gaps in current supremacy claims?

- **Gap #1: Overhead-free runtimes.**
 - Using annealing time or ideal oracle calls instead of true end-to-end runtime.
- **Gap #2: Weak classical baselines.**
 - Ignoring modern, parallel classical algorithms (e.g., SBM) that match or beat quantum performance.
- **Gap #3: Small-instance extrapolation.**
 - Drawing scaling conclusions from size regimes where overhead dominates and exponents are unstable.
- **Gap #4: Oracle vs runtime mismatch.**
 - Exponential query-complexity separations that do not translate into runtime advantage on NISQ hardware.
- **Gap #5: Inconsistent metrics and cherry picking.**
 - Comparing TT_ϵ on the quantum side to under-counted, estimated times on the classical side; reporting best-of-few instances.






Practical guidelines for future claims

- Define a clear, operational metric:
 - TT_ε , TTR, or similar, with explicit ε and target confidence.
- Use *end-to-end* runtime:
 - Include transpilation, embedding, readout, thermalization, data transfer, hyperparameter tuning.
- Benchmark against the best available classical solvers:
 - And let them run on serious hardware (GPUs, FPGAs), not a single-threaded CPU baseline.
- Explore sufficiently large instances to probe asymptotic behaviour.
- Report full statistics:
 - Medians, distributions over many instances, error bars on scaling exponents.

Bottom line

On current NISQ hardware, credible runtime-based quantum supremacy remains unobserved in these case studies. The bar for future claims is now much clearer.

References

-  J. Tuziński, J. Pawłowski, P. Tarasiuk, Ł. Pawela, and B. Gardas, *Recent quantum runtime (dis)advantages*, arXiv:2510.06337 (2025).
-  J. Pawłowski, P. Tarasiuk, J. Tuziński, Ł. Pawela, and B. Gardas, *Toward Quantum Scaling Advantage in Approximate Optimization*, arXiv:2505.22514 (2025).
-  H. Munoz-Bauza and D. Lidar, *Scaling Advantage in Approximate Optimization with Quantum Annealing*, Phys. Rev. Lett. 134, 160601 (2025).
-  P. Singkanipa, V. Kasatkin, Z. Zhou, G. Quiroz, and D. A. Lidar, *Demonstration of Algorithmic Quantum Speedup for an Abelian Hidden Subgroup Problem*, Phys. Rev. X 15, 021082 (2025).
-  P. Chandarana *et al.*, *Runtime Quantum Advantage with Digital Quantum Optimization*, arXiv:2505.08663 (2025).