Past the Statevector Wall? Out-of-Core Relational Simulation and Spill Behaviour for Dense Quantum Circuits

Andrei Ilinescu (5945631)

Delft University of Technology · CSE3000 Research Project

Motivation

Statevector simulation hits a hard memory wall. A complex amplitude vector needs $16 \cdot 2^N$ bytes, about **16 GiB at 30 qubits**, doubling with every added qubit. Past that point the state does not fit in RAM and the simulator stops.

Out-of-core (OOC) relational simulation side-steps this. Amplitudes are stored as tuples; each tensor contraction becomes a join-and-aggregate step in a chain of Common Table Expressions (CTEs); the database spills intermediates to disk and continues *past* the wall.

The catch. This works well for *sparse* circuits, whose output has few non-zero amplitudes, so relational engines reach far larger qubit counts than statevector simulation. Whether it helps for *dense* circuits is open, and is the question of this work.

Research questions

- **RQ1 (boundary).** For a dense circuit (QFT), how far does OOC simulation push the qubit boundary under fixed memory caps, relative to statevector simulation?
- **RQ2 (failure modes & spill).** What binds each engine at the boundary (memory, host disk, wall-clock), and how does spill behaviour drive it?

Method

- **Dense family:** Quantum Fourier Transform, $N = 27 \ldots 32$, near-full-density output, the case that stresses the relational representation hardest.
- **Sparse family (comparison):** Expander circuits, Hadamards on h seed qubits then CNOT-only layers, giving exactly 2^h non-zero amplitudes regardless of n; evaluated $n = 40 \ldots 45$.
- **Backends:** Qiskit Aer (statevector baseline) plus PostgreSQL (server-class row store) and SQLite (embedded row store), executing the emitted SQL.
- **Caps:** 4 / 8 / 16 GiB (Docker cgroups, 1 CPU, swap off); 2-hour timeout; five timed runs per cell.
- **Measured:** status, wall time, spill (exact for PostgreSQL via `EXPLAIN ANALYZE`; OS proxy for SQLite), and largest / total CTE output-relation sizes along the chain.

The dense boundary barely moves

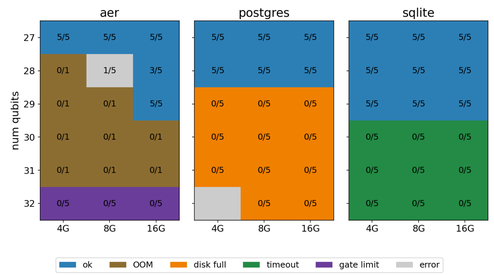

Figure 1. Dominant per-cell outcome (successes / total runs) across backends and caps.

Table 1. Boundary at the 16 GiB cap: median wall time, or the binding ceiling.

N	Qiskit Aer	PostgreSQL	SQLite
27	92 s	515 s	1306 s
28	183 s†	1012 s	2443 s
29	81 s	host disk full	5793 s
30	OOM	host disk full	timeout (>2h)
31	OOM	host disk full	timeout (>2h)
32	gate limit	host disk full	timeout (>2h)

†3/5 runs; reproducible worker error at $N = 28$.

Headline

No engine completes QFT beyond **29 qubits** under a 16 GiB cap, the *same* point at which Aer runs out of memory. For dense circuits the out-of-core advantage largely **disappears**.

The wall relocates, it does not vanish

The single memory wall is replaced by three engine-specific, spill-driven ceilings:

- **Aer** → out of **memory** ($N \geq 30$).
- **PostgreSQL** → out of **disk** ($N \geq 29$).
- **SQLite** → out of **time** ($N \geq 30$, >2h; still feasible).

OOC removes the *hard* feasibility wall; the rest is economic, not categorical.

The boundary is budget-set, not absolute

None of the ceilings is fundamental. Aer's wall scales with RAM (**30** q at 16 GiB, **33** q at 128 GiB, **≈40** q on MPI systems); more disk or time carries the relational engines further. At a matched budget the dense walls coincide.

Spill drives runtime and the boundary

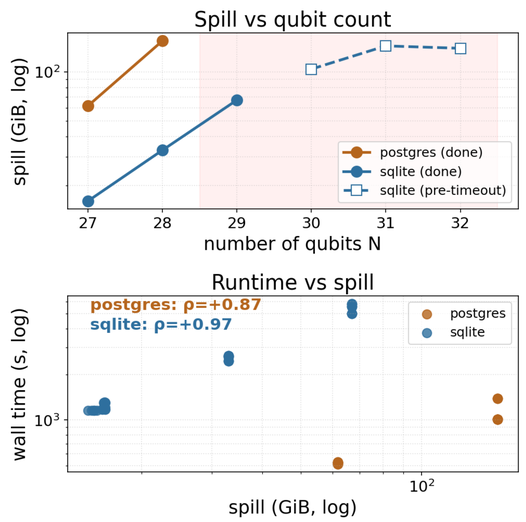

Figure 2. Top: spill grows geometrically in N; PostgreSQL spills 3–5× more than SQLite, crossing the host-disk capacity one qubit sooner. Bottom: runtime tracks spill ($\rho = 0.87$ / 0.97); externalisation, not arithmetic, dominates time.

Cumulative spill, not peak intermediate

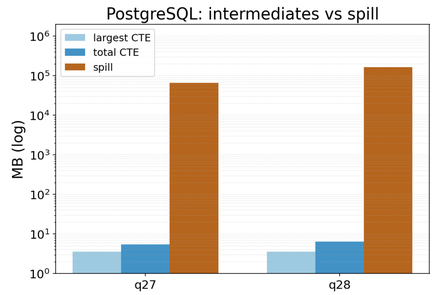

Table 2. Spill correlations, dense QFT (completed runs).

Engine	ρ(rt,spill)	r(lrg)	r(tot)
PostgreSQL	+0.87	+0.45	+0.90
SQLite	+0.97	+0.83	+0.94

At 28 qubits the largest CTE output relation is ~**3.6 MB**, yet PostgreSQL externalises ~**155 GiB** across ~130–140 steps. For both engines $r(\text{tot}) > r(\text{lrg})$: cumulative volume explains spill better than the peak.

Sparse vs dense: same mechanism, different scale

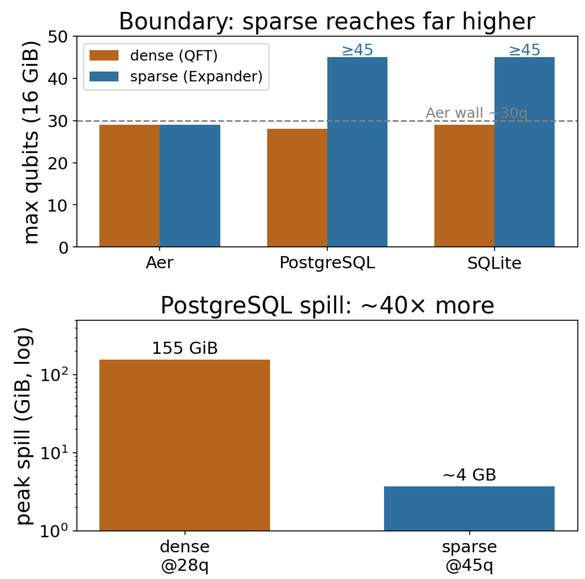

Figure 3. Top: max qubits completed; relational engines reach ≥ 45 on the sparse family but stop near Aer's wall on dense QFT. Bottom: PostgreSQL peak spill, dense externalises ~40× more, at 17 fewer qubits.

Table 3. Spill correlations, sparse family.

Engine	ρ(rt,spill)	r(lrg)	r(tot)
PostgreSQL	≈ +1.00	+0.46	+0.66
SQLite	+0.94	+0.38	+0.77

The same signatures hold in both regimes ($r(\text{tot}) > r(\text{lrg})$; runtime tracks spill). What differs is **spill magnitude**: sparse stays in the low-GB range and reaches 45 qubits; dense reaches hundreds of GiB and hits a ceiling at the wall. The sparse–dense gap is a difference in spill volume, not in kind.

Takeaways

- Dense circuits: OOC does **not** push the boundary past the statevector wall.
- The wall relocates from memory to **disk / time**, each engine-specific.
- The boundary is governed by **cumulative spill**, not peak state size.
- Lever: **spill reduction** (join ordering, materialisation, reuse); next, spill prediction for backend selection.