A COMPANION NOTE · PHYSICS.MAGFLOWMETERS.COM

How the geometry designed the quantum computer

A two-stage method that turned an almost-impossible search into a tractable one — and made the math *easier*, not harder.

← Back to the four papers and the rest

The same frozen shape that this project reads the forces and particle masses out of was also handed a very different, very practical problem: **design a fault-tolerant quantum computer.** What follows is the plain-language version of how that worked — and why, honestly, it would have been close to impossible without the geometry doing the heavy lifting.

The problem: a search space too big, and too tangled, to win

Protecting a quantum computer from errors means encoding each *logical* qubit (the one your algorithm uses) into many noisy *physical* qubits. The number you pay — physical qubits per logical qubit, the **overhead** — is the whole game. Driving it down means choosing, all at once:

- the physical encoding of a qubit,
- the error-correcting code's check structure and connectivity,
- its asymmetric code distances, and
- the noise bias the design is allowed to assume.

These choices are **coupled**: the three things that make a code efficient — a strong noise bias, the right code structure, and a protective spectral gap — normally fight each other. Improve one and you tend to spoil another. Searching that space directly is close to hopeless. It is high-dimensional, every candidate is expensive to evaluate, and the good regions are razor-thin.

The geometric prior: fewer knobs, not more

The counter-intuitive move is to *add* structure and end up with *less* to search. Instead of independently dialing a dozen hardware-and-code knobs, the entire architecture is generated from **four continuous numbers** — the geometry's moduli. The geometry's own structure — its curvature, its holonomy, its symmetry — supplies the couplings that a human designer would otherwise have to discover by trial and error.

> **The paradox.** Putting a rich piece of geometry in front of the problem made the math *simpler*, not more complicated. Four moduli replace a dozen conflicting dials; the relationships between bias, code, and gap that you would normally have to hunt for are *forced* by the shape. The geometry shrank the haystack *and* told us roughly where the needle had to be.

The two-stage method

The search was split into two stages, and the order is the entire point.

STAGE 1 — FIND THE REGIME WHERE THE COST STOPS GROWING WITH SIZE

The first subsystem we analyzed had a remarkable property: its overhead — physical qubits per logical qubit — became **independent of the size of the quantum computer**, i.e. of how many logical qubits it holds. Operated below the code's error-correction threshold, the per-logical cost settles onto a **floor** that does *not* climb as you add more logical qubits.

Stage 1 restricts the design search to exactly this **count-independent regime**, *before any specific code is chosen*. Its output is a structured, admissible **design space** in which every candidate already carries that floor-at-scale property. Count-independence becomes a built-in feature of the sub-space, established up front — not something you hope to recover at the end.

STAGE 2 — A TOPOLOGICAL SEARCH INSIDE THAT DESIGN SPACE

Within the count-independent sub-space that Stage 1 carved out, we run a **topological search** over the geometry-derived code structures: the protected-subspace, the check structure, the connectivity, and the asymmetric distances. Because the search can only

reach configurations *inside* the Stage-1 sub-space, every candidate it touches already has the scale-invariant property. The search only decides *where within* the count-independent band the design lands — never *whether* it is count-independent.

Why the order is load-bearing

Stage 1 *first* isolates the regime where overhead stops depending on qubit count; Stage 2 *then* searches there. Reverse the order — search first and hope the winner happens to be scale-invariant — and you are optimizing a number that grows with scale, with no guarantee the answer survives at ten thousand or a hundred thousand logical qubits. Doing it in this order is exactly what lets the result be reported as a **floor that holds at any size**.

Why it would be almost impossible without the geometry

Two reasons, and they compound:

1. **The search space.** Without the geometric prior, you face a high-dimensional, tightly coupled optimization with no map. The prior collapses that to four moduli and a small admissible region — turning an intractable search into one a laptop can run.
2. **Knowing where to look.** The count-independent regime that Stage 1 exploits is a structural property of the geometry's spectrum. Without the geometry there is no principled way to find — or even to know to look for — the subsystem in which error becomes independent of machine size. The geometry didn't just shrink the haystack; it told us the needle had to be there.

Put together: the geometry made a search that would otherwise be effectively unwinnable both **smaller** and **structured** — and the count-independence that makes the result worth reporting falls straight out of the shape, rather than being engineered in by hand.

The honest status (read this part too)

Every quantitative figure behind this note is a **design-stage, simulated projection** — not a measured property of any device, and not a runnable algorithm. The overhead is a *memory-overhead floor* (a qubit-count metric); a competitive ratio is a qubit-count win, not by itself a working cryptographic algorithm. The two-stage construction, the floor, the geometry, and every open gate — with their status labels — are recorded in full in the provisional patent. What this note claims is narrow and specific: **the geometry made**

the design search tractable, and made its scale-invariance a built-in property rather than a lucky outcome.

Chris Bergstrom · cbergstr@gmail.com · physics.magflowmeters.com