

Complexity, Quantum, and AI: Deep Linkage?

Kenneth W. Regan¹
University at Buffalo (SUNY)

30 September, 2026

¹Recent Students: Chaowen Guan, Razieh Fathi, Marissa Dominijanni, Chen Xu (PhD), Yash Rathi (MS)

Computational Complexity

- The study of the time *needed* to solve computational problems, and how much memory and other resources computers require.
- Largely independent of the computer model, beyond a fundamental divide into **serial**, **parallel**, and **quantum**.
- Main technical achievement: the relation of computational problems by **reducibility**.
- Main scientific surprise:

The **many thousands** of computational problems that have been studied in many disciplines, some for centuries, cluster into **barely over a dozen** equivalence classes under reducibility.

- The biggest cluster is the class of **NP-complete** problems.

P=NP and Worse

- **P**: problems with algorithms that **solve** them in **polynomial time**:

As the size of the data doubles, the time needed goes up by at most a **linear** factor: $t(n) = n^k \implies t(2n) \leq Kt(n)$, $K = 2^k$.

- **NP**: “Nondeterministic” Polynomial Time: If you know a secret fact or guess a good answer, you can verify and **teach** it to someone in polynomial time.
- Example: Given a Boolean formula f like

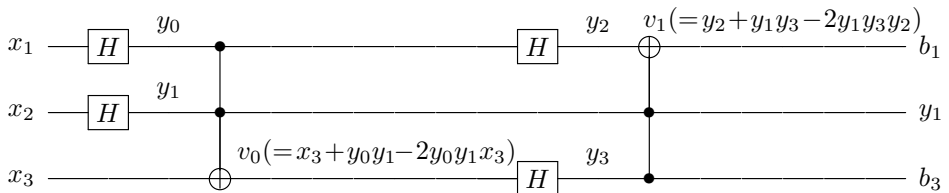
$$f = (x_1 \vee (\neg x_2)) \wedge ((\neg x_1) \vee x_2 \vee x_3) \wedge ((\neg x_2) \vee (\neg x_3)),$$

is there a way to make f true?

- Called *Satisfiability* (SAT).
- Equivalent to $\neg f$ *not* being a **tautology**.
- Is NP-complete, so $\text{NP} = \text{P} \iff \text{SAT belongs to P}$.
- We don't even know whether SAT can be solved in **linear** time!

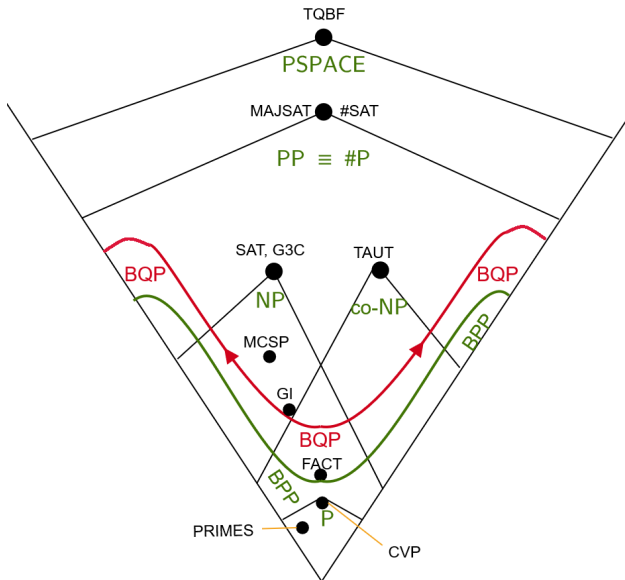
Enter Quantum

- **Factoring** is among a handful of problems in NP not known to be complete or in P.
- RSA security depends on it, so many want it to be *hard*.
- But solvable in polynomial time by a **quantum computer**: **BQP**.
- Textbook on quantum algorithms; blog series: Can QCs be Built?
- Research on simulating **quantum circuits** by logic and algebra:



$$\begin{array}{lll}
 p_1 = p_0 \text{ XOR } (y_0 \& x_1) & p_2 = p_1 \text{ XOR } (y_1 \& x_2) & v_0 = x_3 \text{ XOR } (y_0 \& y_1) \\
 p_3 = p_2 \text{ XOR } (y_2 \& y_0) & p_4 = p_3 \text{ XOR } (y_3 \& v_0) & v_1 = y_2 \text{ XOR } (y_3 \& y_1)
 \end{array}$$

The Complexity Class Neighborhood...



Structural Squeeze and Classical Heuristics...

- Between **P** and **NP**-complete is mostly deserted.
- Similar between **P** and **#P**, per “Dichotomy” results by Jin-Yi Cai and others.
- Except that **BQP** is in the latter desert. Is **BQP** squeezed out?
- **Shor’s Algorithm** for Factoring has precious few neighbors.
- **Grover’s Algorithm** is only quadratic savings, and for **SAT** and **#SAT**, saves only $\sqrt{\exp(n)} = \exp(n/2)$.
- Many **NP**-complete problems have adept heuristics.
- Also for **#SAT**: software **sharpSAT**, **Cachet**.
- However, **SAT**-encoded **cases** of Factoring remain hard for them.

Can we capture **quantum circuits** by combinatorial invariants that lead to new heuristics for *classically* simulating them?

...might pave the way for AI?

Progress on factoring n -bit RSA Challenge numbers:

- **RSA-250** (829 bits): Feb. 28, 2020—without AI
- **RSA-260** (862 bits): Sep. 3, 2026, by Eric Lu with Cognition AI (4,900 GPU-days at \$400,000 cost)
- **RSA-892**: Sep. 19, 2026, Stephen Weis + Claude (cost not publ.)

Still not close to deployed **2048**-bit numbers. Unclear how much Claude added to the general Number Field Sieve beyond lots of GPU compute.

Q: Does Factoring in **BQP** presage classical cracks that AI can exploit?

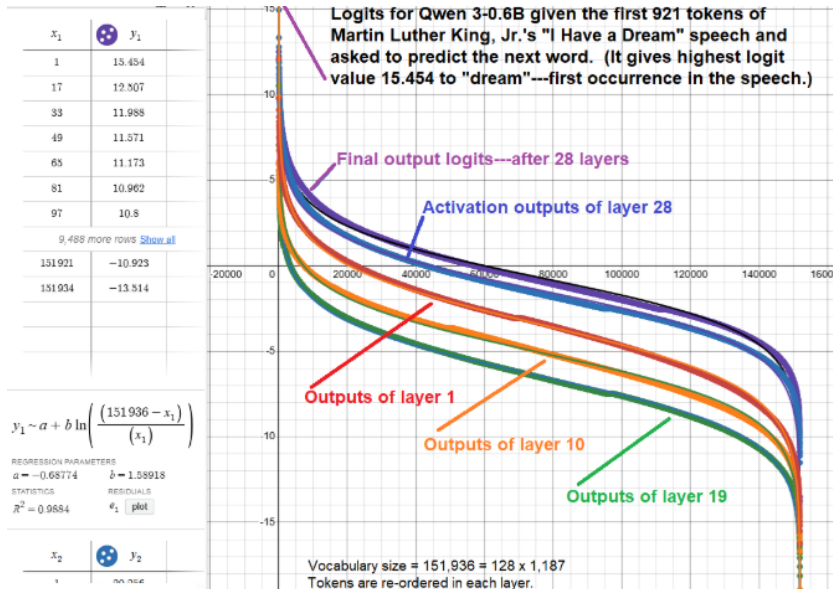
AI can now interpret QCs enough to solve John Preskill's HWs.

Can AI generally **de-quantize** real applications? The moniker “**Cargo Cult Algorithm**” is now far more apt than Scott Aaronson **realized**.

Segue to my other research...

- A **predictive analytic model** of human decision-making at chess.
- Distinguishes games played by (humans cheating with) computers.
- Can we solve the analogous “Turing Test” for LLM outputs?
- My C++ chess analyzer has been fitted to read LLM **logit** scores as though they are values of chess moves.
- Chess: find best next move(s). LLMs: generate best next word(s).
- LLM top-layer scores yield probabilities via **softmax**.
- **Key Q 1:** Can reading scores *from lower layers* improve prediction?
- Reason for *no*: LLM layer wts. already max predictive accuracy—?
- **Key Q 2:** Does giving human vs. AI text create different amounts of “struggle” in lower layers? (Quoting MS grad. Yash Rathi)
- Reason for *yes*: Training LLMs on AI-gen. text is less effective.
- Terms: **Logit Lens**, **Tuned Lens**, **EntropyLens**, **Pangram EditLens**.

Why Token Values Are Called “Logits”



Steep At Top Makes Sense, But Why At Bottom?

1. " dream" = 15.454
2. "strong" = 14.774
3. "heart" = 14.454
4. " heart" = 14.409 (this duplication/spacing issue needs to be resolved too)
5. "faith" = 14.280
6. " hope" = 14.155 ...

Then things settle down so that by 200 or so tokens in, consecutive logit values differ only in the 0.001 place. But at the end:

- 151,931. "Shu " = -12.717
- 151,932. "Die Dai " = -12.871
- 151,933. "Neng Hao " = -13.330
- 151,934. ".EventType" = -13.514
- 151,935. "<|endoftext|>" = -inf
- 151,936. "<|im_end|>" = -inf.

Granted, all the probabilities for these tokens obtained after softmax are microscopic. But why should the probability for "Die Dai " be $e^{0.459} = 1.58 \dots$ times higher than that for "Neng Hao "?