Will a Quantum-Inspired Classical Algorithm Become the Default Algorithm for an Industrially Relevant Computational Task in the Next Decade?

Adam Godel and Egemen Tunca

December 5, 2025

Boston University Physics 536: Quantum Computing

Introduction

Why study quantum algorithms? A major reason: we believe that at some point in the future, they will be useful to solving real-world problems. This requires quantum hardware...

Quantum algorithms → “quantum-inspired” *classical* algorithms; made to show that a purported quantum speedup can often actually be just as efficient on a classical computer.

Are these quantum-inspired algorithms useful to solve problems in their own right?

The Argument In Favor

The key thing to note about these two algorithms is their *generality*.

Quantum-Inspired Algorithms for Linear Algebra and Recommendation Systems

Given an $m \times n$ matrix $A \in \mathbb{R}^{m \times n}$ with singular value decomposition (SVD)

$$A = \sum_{\ell=1}^{k} \sigma_\ell \boldsymbol{u}^{(\ell)} \boldsymbol{v}^{(\ell)^T},$$

the goal is to *sample* entries of the n-dimensional vector

$$\boldsymbol{x} = \sum_{\ell=1}^{k} \lambda_\ell \boldsymbol{v}^{(\ell)},$$

with respect to the length-square probability distribution $p_x(i) = x_i^2 / \|\boldsymbol{x}\|^2$.

Choosing Coefficients λ_ℓ

For linear systems of equations, we can get the solution to $A\boldsymbol{x} = \boldsymbol{b}$ for some vector $\boldsymbol{b}$ with the coefficients

$$\lambda_\ell = \frac{1}{\sigma_\ell^2} \langle \boldsymbol{v}^{(\ell)}, A^T \boldsymbol{b} \rangle.$$

For recommendation systems, A is a preference matrix where A_{ij} denotes the rating of user i to product j. We can get $\boldsymbol{x}$, the i-th row of the low-rank approximation of A, representing the preferences of user i, by

$$\lambda_\ell = \langle A_i^T, \boldsymbol{v}^{(\ell)} \rangle$$

where A_i is the i-th row of A.

Frieze-Kannan-Vempala (FKV) Algorithm

$$A = \begin{array}{c} \\ \mathrm{i}_1 \\ \mathrm{i}_3 \\ \\ \\ \mathrm{i}_4 \\ \\ \mathrm{i}_2 \\ \\ \\ \end{array} \begin{pmatrix} -2.04 & 0.02 & 0.79 & 0.3 & 2.04 & 0.68 & -0.37 & 0.4 & -0.08 & 0.66 \\ -3.44 & 1.5 & 2.68 & -0.93 & 4.04 & -0.02 & -0.78 & 0.96 & -0.58 & -1.99 \\ 5.25 & -2.78 & -3.3 & -1.97 & -4.99 & -5.27 & 1.16 & 1.38 & -3.31 & 2.16 \\ -0.03 & 3.14 & 2.49 & -0.33 & 1.04 & 0.81 & -0.21 & -0.84 & 1.8 & -3.86 \\ -2.51 & 2.45 & 3.35 & -0.48 & 3.87 & 0.17 & -0.59 & 0.62 & 0.48 & -1.63 \\ -0.89 & -0.79 & -1.5 & 1.41 & -0.97 & 3.14 & -0.19 & -1.58 & 1.37 & -0.68 \\ 0.42 & 1.31 & 1.36 & -0.12 & 0.69 & -0.47 & 0.06 & 0.14 & 0.55 & -0.18 \\ -0.22 & 2.63 & 1.94 & 0.78 & 0.85 & 2.08 & -0.18 & -1.3 & 2.58 & -2.31 \\ 0.0 & 1.29 & 1.09 & 0.84 & 0.6 & 1.17 & -0.02 & -0.6 & 1.6 & 0.06 \\ 3.69 & 1.48 & 1.27 & -1.99 & -1.2 & -4.43 & 0.69 & 1.24 & -1.13 & -0.39 \end{pmatrix}$$

$$\downarrow$$

$$R = \begin{array}{c} \mathrm{i}_1 \\ \mathrm{i}_2 \\ \mathrm{i}_3 \\ \mathrm{i}_4 \end{array} \begin{pmatrix} -94.51 & 41.21 & 73.63 & -25.55 & 111. & -0.55 & -21.43 & 26.38 & -15.93 & -54.67 \\ -7.33 & 87.64 & 64.64 & 25.99 & 28.32 & 69.31 & -6. & -43.32 & 85.97 & -76.97 \\ 87.04 & -46.09 & -54.71 & -32.66 & -82.73 & -87.37 & 19.23 & 22.88 & -54.87 & 35.81 \\ -35.23 & -31.27 & -59.38 & 55.82 & -38.4 & 124.31 & -7.52 & -62.55 & 54.24 & -26.92 \end{pmatrix}$$

$$\downarrow$$

$$R = \begin{array}{cccccccccc} \mathrm{j}_3 & & \mathrm{j}_1\ \mathrm{j}_4 & & & & & \mathrm{j}_2 & & \\ -94.51 & 41.21 & 73.63 & -25.55 & 111. & -0.55 & -21.43 & 26.38 & -15.93 & -54.67 \\ -7.33 & 87.64 & 64.64 & 25.99 & 28.32 & 69.31 & -6. & -43.32 & 85.97 & -76.97 \\ 87.04 & -46.09 & -54.71 & -32.66 & -82.73 & -87.37 & 19.23 & 22.88 & -54.87 & 35.81 \\ -35.23 & -31.27 & -59.38 & 55.82 & -38.4 & 124.31 & -7.52 & -62.55 & 54.24 & -26.92 \end{array}$$

$$\downarrow$$

$$C = \begin{array}{cccc} \mathrm{j}_1 & \mathrm{j}_2 & \mathrm{j}_3 & \mathrm{j}_4 \\ 106.02 & 57.6 & -129.5 & 106.02 \\ 93.08 & -94.6 & -10.04 & 93.08 \\ -78.78 & 49.96 & 119.26 & -78.78 \\ -85.51 & -136.6 & -48.28 & -85.51 \end{array}$$

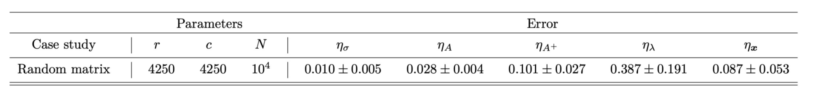
Getting Solution Vector x from C

Given singular values $\tilde{\boldsymbol{\sigma}}_\ell$ and left singular vectors $\boldsymbol{\omega}^{(\ell)}$ of C, we can pretty easily calculate the approximate solution vector $\tilde{\boldsymbol{x}}$.

For $\lambda = \langle \boldsymbol{y}, \boldsymbol{z} \rangle$, define a random variable $\chi_i = y_i z_i / p_y(i)$ sampled by $p_y(i) = y_i^2 / \|\boldsymbol{y}\|^2$. Take N samples → unbiased estimator $\hat{\lambda} \approx \lambda$.

We can implicitly compute the approximate solution vector $\tilde{\boldsymbol{x}} = \sum_{\ell=1}^{k} \lambda_\ell \boldsymbol{v}^{(\ell)} = R^T \boldsymbol{w}$, where $\boldsymbol{w} \equiv \sum_{\ell=1}^{k} \frac{\tilde{\lambda}_\ell}{\tilde{\sigma}_\ell} \boldsymbol{\omega}^{(\ell)}$, using rejection sampling to only *query* the entries we need of $\boldsymbol{w}$ and R to sample from $\tilde{\boldsymbol{x}}$ with respect to $p_x(i) = x_i^2 / \|\boldsymbol{x}\|^2$.

Experimental Results for Linear Systems



Parameters				Error				
Case study	r	c	N	η_σ	η_A	η_{A^+}	η_λ	$\eta_{\boldsymbol{x}}$
Random matrix	4250	4250	10^4	0.010 ± 0.005	0.028 ± 0.004	0.101 ± 0.027	0.387 ± 0.191	0.087 ± 0.053

For randomly generated $m \times n$ matrices A and length m vectors $\boldsymbol{b}$ with rank k and condition number κ, compute $\boldsymbol{x}$ such that $A\boldsymbol{x} = \boldsymbol{b}$ for $m = 40{,}000$, $n = 20{,}000$, and $k = \kappa = 5$:

- 5192.5 seconds for a direct calculation
- 2470.4 seconds for the quantum-inspired algorithm

The error was about 8.7% for the solution vector $\boldsymbol{x}$.

A Potential Application for Recommendation Systems

MovieLens 100K database (relatively small): direct calculation much faster than the quantum-inspired algorithm.

One potential candidate for a speedup for recommendation systems is patent indexing:

- Very large preference matrix (thousands of documents and tens of thousands of words)
- Seems to be approximable with rank as low as $k = 80$

Here, the vector $\boldsymbol{x}$ represents the alignment of term i for each document j.

“Opening the Black Box” of the Grover Oracle

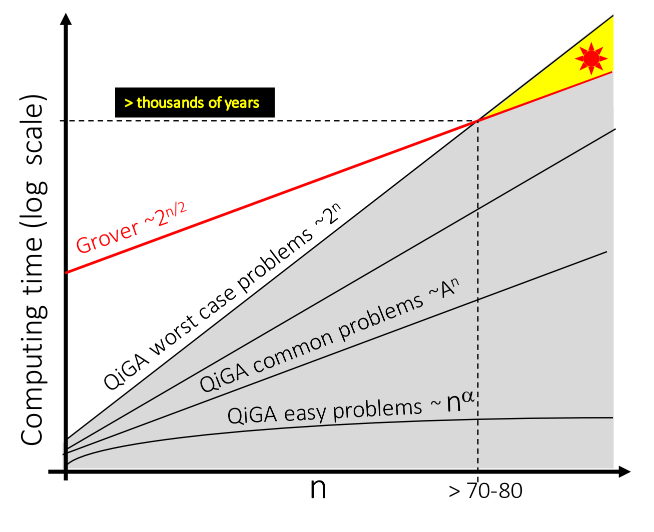
We know that Grover’s algorithm has a quadratic speedup, i.e. $O(2^{n/2})$ for searching length n bitstrings as opposed to $O(2^n)$.

However, remember that it relies on having access to a black-box oracle, which it assumes $O(1)$.

If we throw out this assumption and construct the oracle ourselves, is there still a quantum speedup in practice?

“Opening the Black Box” of the Grover Oracle

Short answer: it depends on the problem, but mostly no.



Quantum-inspired Grover's Algorithm (QiGA)

Consider the state

$$|\Psi_w\rangle = U_w|s\rangle = |s\rangle - \frac{2}{\sqrt{2^n}} \sum_{\alpha=1}^{S} |w^\alpha\rangle,$$

where $|s\rangle$ is the equal superposition state and $|w^1\rangle, \ldots, |w^S\rangle$ are the marked states.

How to extract the marked states? One solution: implement a mapping $|\Psi\rangle \otimes |s\rangle \mapsto (|\Psi\rangle - |s\rangle) \otimes |s\rangle$. But this is not unitary!

We could do this well on a classical computer using only one oracle call if we could represent $|\Psi_w\rangle$ (relatively) efficiently.

Using a Matrix Product State (MPS) to Represent $|\Psi_w\rangle$

A matrix product state (MPS) is represented as

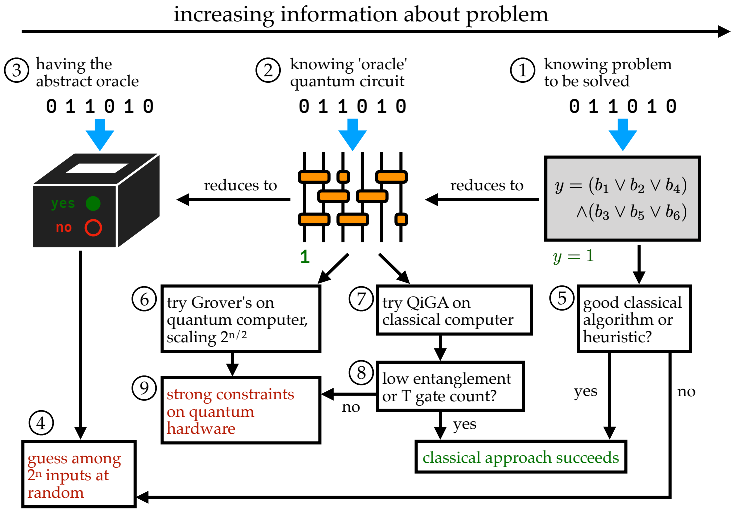
$$|\Psi\rangle = \sum_{x_{n-1}\cdots x_0 \in \{0,1\}^n} M_{n-1}(x_{n-1}) \cdots M_0(x_0) |x_{n-1} \cdots x_0\rangle$$

where the $M_i(x)$ are $\chi \times \chi$ matrices for $1 \leq i \leq n-2$; $M_0(x)$ is a $\chi \times 1$ matrix and $M_{n-1}(x)$ is a $1 \times \chi$ matrix.

$\chi =$ *bond dimension*. Crucially, it is dependent on the entanglement of the state; the entanglement of $|\Psi_w\rangle$ is usually low.

Note that exponential problems will still scale exponentially at worst → just a *relative* speedup.

When Can We Actually Use QiGA?



increasing information about problem

③ having the abstract oracle

② knowing 'oracle' quantum circuit

① knowing problem to be solved

$y = (b_1 \vee b_2 \vee b_4)$
$\wedge (b_3 \vee b_5 \vee b_6)$

$y = 1$

⑥ try Grover's on quantum computer, scaling $2^{n/2}$

⑦ try QiGA on classical computer

⑤ good classical algorithm or heuristic?

⑧ low entanglement or T gate count?

⑨ strong constraints on quantum hardware

④ guess among 2^n inputs at random

classical approach succeeds

Potential Applications for QiGA: *k*-SAT

The paper focuses on 3-SAT, which scales with $O(A^n)$ where $1 < A < 2$ in the general case but can actually scale polynomially in certain cases (e.g. "quasi-1D" 3-SAT).

QiGA could potentially be more effective for higher instances of k-SAT, especially when the number of satisfying inputs is low.

For comparison, Schöning's algorithm for k-SAT scales as $O((2(1-1/k))^n)$.

Potential Applications for QiGA: Subset Sum

Given positive integers $\boldsymbol{a} = (a_1, \dots, a_n)$ and M, find

$$\sum_{i=1}^{n} a_i x_i = M, \quad \forall i, x_i \in \{0, 1\}.$$

A quantum oracle for subset sum can be produced relatively efficiently, using n qubits and n classical "shadow registers", as well as using only CNOT and Toffoli gates.

The best classical algorithm is currently $O(2^{0.283n})$, while a proposed quantum algorithm that relies solely on Grover search is $O(2^{0.236n})$. Hence QiGA success = optimality!

Asymptotics beyond the hype

Theoretical worst-case bounds

- Linear systems:

$$\widetilde{O}\big(\kappa^{16} k^6 \, \|A\|_F^6 / \varepsilon^6\big)$$

- Recommendation:

$$\widetilde{O}\big(k^{12} / \varepsilon^{12}\big)$$

- Here $\widetilde{O}(\cdot)$ hides polylogarithmic factors in m, n, but *not* in k, κ, ε.

What these parameters mean in practice

- k: target rank / effective latent dimension (e.g. # topics in LSI).
- κ: condition number of A (sensitivity of the problem).
- ε: accuracy tolerance for the output (smaller ε means we want a more precise solution).

Hidden sampling cost N (I)

General Monte Carlo bound

- Estimating an inner product $\langle y, z \rangle$ from sampled entries:

$$N = O\left(\frac{1}{\varepsilon^2 \cos^2 \theta}\right),$$

  where θ is the angle between y and z and ε is the target precision of this estimator.

- Linear systems:

$$N = O\left(\frac{k^2 \kappa^2 \kappa_\beta^2}{\varepsilon^2}\right)$$

- Recommendation systems:

$$N = O\left(\frac{k\,\kappa_\nu^2}{\varepsilon^2}\right)$$

Takeaway: Sampling alone is polynomial in k, κ and $1/\varepsilon$.

Hidden sampling cost N (II)

Numerical example for the linear-systems bound

- Use "moderate" parameters:

$$k = 100, \quad \kappa = 100, \quad \kappa_\beta = 100, \quad \varepsilon = 10^{-2}.$$

- Plugging into $N = O\left(k^2\kappa^2\kappa_\beta^2/\varepsilon^2\right)$ gives

$$N \sim 10^{16} \text{ samples.}$$

Contrast with the experiments

- In the paper they *fix* $N = 10^4$ to keep the runtime reasonable.
- Increasing k or κ, or asking for smaller ε, would very quickly make N infeasible.

Two options after FKV

FKV step

- Get a small sketch C and approximate right singular vectors $v^{(\ell)}$ of A.
- Target vector (solution / recommendation):

$$\tilde{x} = \sum_{\ell=1}^{k} \lambda_\ell \, v^{(\ell)}.$$

Option 1: direct reconstruction (classical)

- Compute λ_ℓ and $v^{(\ell)}$ explicitly from the FKV output.
- Form $\tilde{x}$ explicitly; cost $O(kn)$ (linear in n for fixed k).

Option 2: QI sampling

- Keep $\tilde{x}$ implicit; store A in a length–square sampling data structure.
- Estimate λ_ℓ and sample entries of $\tilde{x}$ using Monte Carlo; runtime $\mathrm{poly}(k, \kappa, 1/\varepsilon, \log m, \log n)$.

When can sampling beat $O(kn)$?

What the paper says

- Direct reconstruction from the approximate SVD costs $O(kn)$.
- Authors: this "can be done extremely fast even for problems of large size" because it is linear in n.
- Sampling-based steps are preferable only in very structured cases.

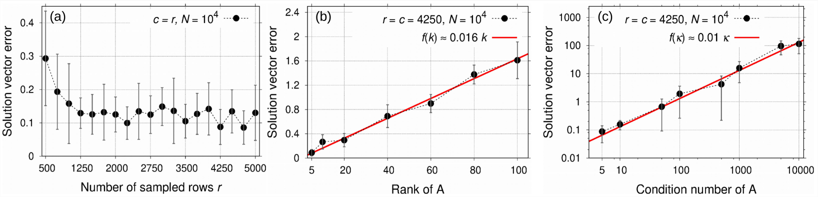
Conditions for a QI advantage

- Matrix dimensions m, n extremely large *and* rank k very small.
- Condition number κ small and accuracy requirement ε not too strict.
- Length-square sampling access to A available.

Otherwise

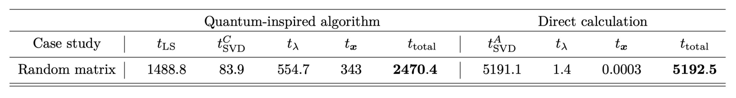
- Overheads in $k, \kappa, 1/\varepsilon$ and the sampling cost N dominate, so the simple $O(kn)$ reconstruction path is usually better.

Random-matrix case: error behaviour



- Setup: $m = 40{,}000$, $n = 20{,}000$, rank $k = 5$, condition number $\kappa = 5$, $r = c = 4250$, $N = 10^4$ samples.
- Best case (panel a, tuned r): solution-vector error $\eta_x \approx 8.7\%$.
- Panels b,c: error grows almost linearly with k and κ; around $k \approx 50$ or $\kappa \approx 10^2$ the error is already $O(1)$ (order 100%).

Random-matrix case: runtime and baseline



	Quantum-inspired algorithm					Direct calculation			
Case study	t_{LS}	t^{C}_{SVD}	t_{λ}	$t_{\boldsymbol{x}}$	t_{total}	t^{A}_{SVD}	t_{λ}	$t_{\boldsymbol{x}}$	t_{total}
Random matrix	1488.8	83.9	554.7	343	**2470.4**	5191.1	1.4	0.0003	**5192.5**

Random matrix, $k = \kappa = 5$ (best-case setup)

- Quantum-inspired total: $t_{\mathrm{QI}} \approx 2470$ s.
- Direct calculation total: $t_{\mathrm{direct}} \approx 5193$ s (exact SVD of A + exact solve).

Important caveat

- Both methods already run FKV on A and get an approximate SVD
- The direct classical algorithm could also use *that* low-rank SVD and reconstruct x in $O(kn)$ time, without Monte Carlo sampling.
- Then FKV error would be shared, and the only difference would be: *direct $O(kn)$ reconstruction vs. QI sampling overhead*.

Real-world dataset: MovieLens 100K

Setup

- $A \in \mathbb{R}^{943 \times 1682}$: user–movie ratings (MovieLens 100K).
- Low-rank model with moderate k (tens of latent factors).
- Same QI pipeline as for random matrices: FKV sketch + sampling with $N = 10^4$.

Accuracy

- QI algorithm: solution-vector error $\eta_x \approx 0.7$ (about 70%).
- Direct classical method: noticeably smaller error on the same task; within their tested parameters, QI never beats direct on error.

Runtime

- QI method is significantly slower than the direct method (sampling + data-structure overhead dominate).
- So on this first realistic recommendation benchmark, QI is both *less accurate* and *slower* than a standard classical baseline.

When is QiGA efficient?

- QiGA $=$ classical tensor-network (MPS/MPO) simulation of Grover.
- Runtime is dominated by the maximum MPS bond dimension χ_{max}:

$$T_{\mathsf{QiGA}} \;\propto\; \mathrm{poly}(n)\,\chi_{\mathsf{max}}^3.$$

- Bond dimension χ measures bipartite entanglement in the oracle circuit:
    - low entanglement $\Rightarrow$ small χ_{max} $\Rightarrow$ cheap;
    - volume-law entanglement $\Rightarrow$ huge χ_{max} $\Rightarrow$ exponential cost.
- So the only regime where QiGA can be competitive is when the Grover oracle admits a low-entanglement, “almost 1D” tensor-network representation.

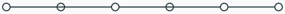
Random 3-SAT: entanglement and runtime blow up

QiGA runtime is dominated by the max bond dimension $\chi_{\max}$

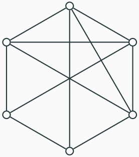
n	S	$\chi_{\max}$	time
30	4	467	21 s
32	2	954	1.8 min
34	48	1162	3.2 min
36	16	1994	8.3 min
38	8	5867	1.6 h
40	0	1402	4.2 min
40	28	2926	21 min
40	161	5690	1.65 h
40	174	10374	6.5 h

- Random 3-SAT near the phase transition: clause graph is highly connected and has large treewidth, so $\chi_{\max}$ is already in the 10^3 range.
- As instances get more "messy", $\chi_{\max}$ jumps to 10^4 and QiGA time grows from seconds to hours, while modern classical SAT solvers solve these $n \approx 40$ instances in $\ll 1$ second.

Structure: low treewidth ⇒ easy for both sides



quasi-1D SAT (low treewidth)



random SAT (high treewidth)

- **Treewidth:** how close the constraint graph is to a tree. Left: small treewidth; right: large treewidth.
- Bounded treewidth k ⇒ tree decomposition and DP in time $f(k)\,n$ (Courcelle / DP). QiGA's polynomial regime (quasi-1D SAT, structured subset sum) lives exactly in this low-treewidth region, where classical algorithms are already strong.

Rebuttal + Questions