Quantum-Inspired Hamiltonian Descent

Theory, Implementation, and Real-World Applications

Yuxiang Peng

Department of Computer Science

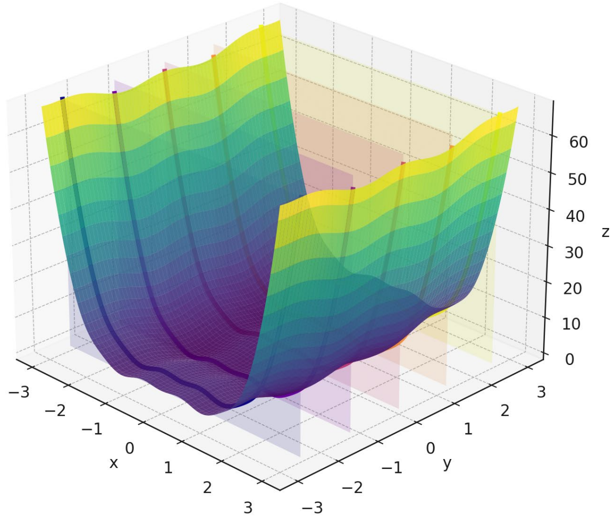
Purdue University

Mixed-Integer Non-Convex Programming

- Problem:

$$\min_x f(x)$$
$$\text{s.t.} \qquad Ax \leq b, \qquad L \leq x \leq R$$
$$x_i \in \mathbb{Z}, \qquad i \in \mathcal{I}$$

  - f: non-linear, non-convex, *(continuous)*
    - I.e., $f(x) = \frac{1}{2}x^T Q x + w^T x$ for indefinite Q.
  - Hardness: *NP-hard* $\rightarrow$ very hard in general



- Solvers
  - **Global solvers**: branch-and-bound, …
    - Certified global optimality
  - **Metaheuristics**: evolutionary algorithm, simulated annealing, …
    - Find high-quality solutions fast with many samples
  - **Designed for and engineered for CPU-based HPC.**

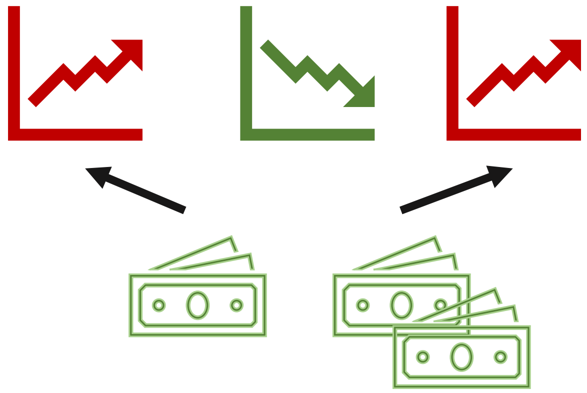
Practical Problems in Big Data Era

- Portfolio optimization in quantitative trading:
  - Invest in k stocks that maximize revenues and minimize risks.

$$\min_{x,z} \ \frac{\lambda}{2} x^T Q_{\mathrm{cov}} x - w^T x$$

$$\text{s.t.} \quad \Sigma_i z_i \le k, \quad \Sigma_i x_i \le S, \quad 0 \le x_i$$

$$l_i z_i \le x_i \le r_i z_i, \ z_i \in \mathbb{B}$$



  - "Small" size: SP500 has 500 stocks
    - Existing metaheuristics (even global solvers) work well.
  - Big data era:
    - Generally, $> 10^4$ investment assets possible
    - Mid-frequency automated trading: want solutions in seconds.
- Highly challenging for conventional solvers when size grows.

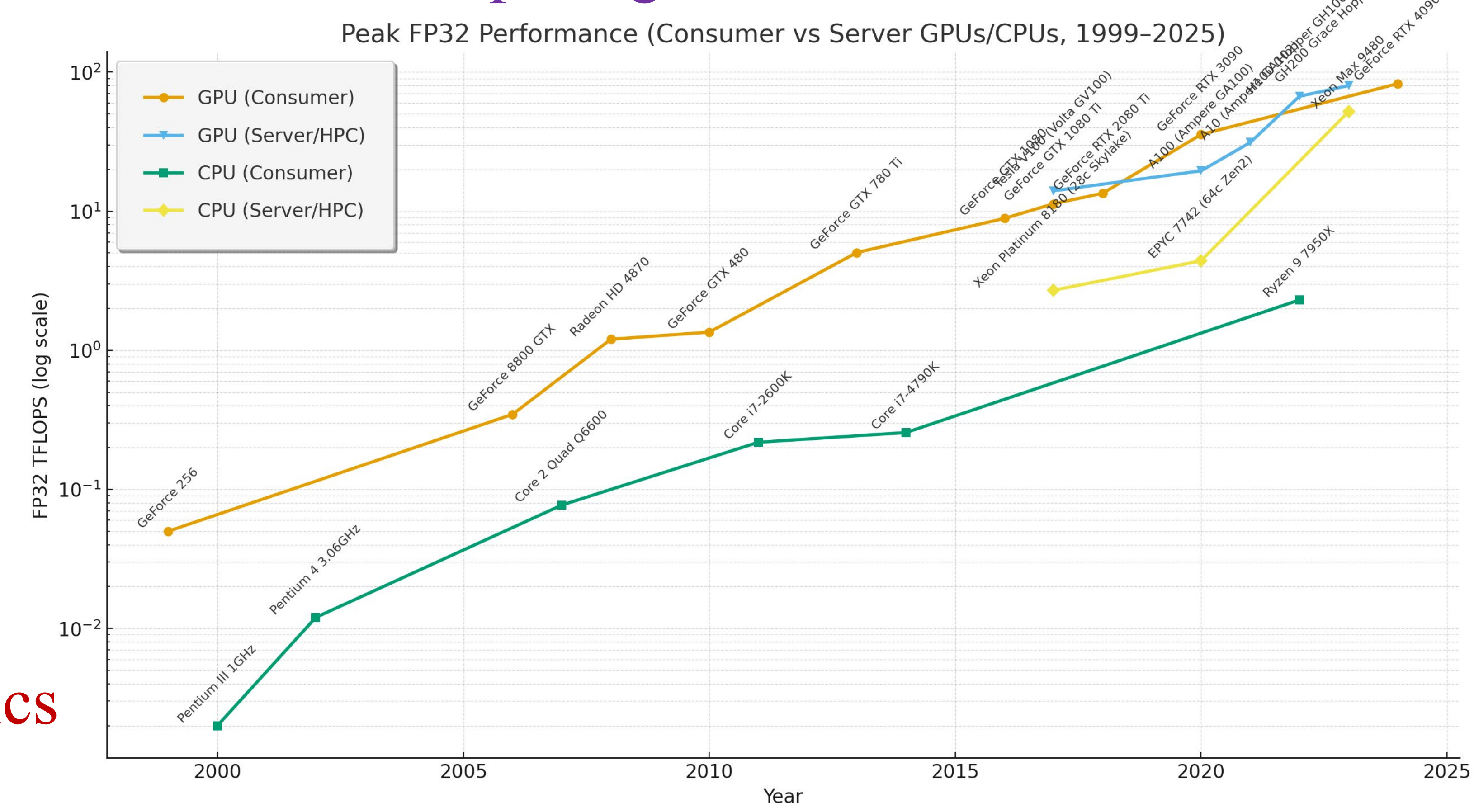
Hardware Acceleration with GPUs?

- GPUs have matured for scientific computing!

❑ Highly parallel computation for simple tasks

❑ Cheap per calculation

- Can we accelerate MINLP metaheuristics with modern GPUs?



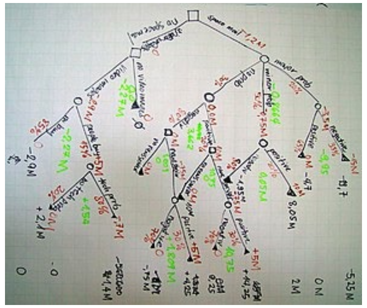
(Generated by ChatGPT deep research)

Engineering Challenges with GPUs

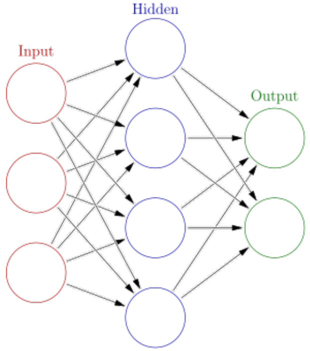
- GPU-hard operations in metaheuristics
  - Complicated logic operations and branching
    - Evolutionary, particle-swarm, (branch-and-bound), …
  - Random memory access
    - Simulated annealing, evolutionary, …
  - Communication between samples
    - Parallel tempering, particle-swarm, …
  - Generating numerous random numbers
    - MCMC-based, genetic, …
- Rarely any candidates are left… 😟

GPU-native algorithm design: **Quantum-inspired solvers!**

Analogy in ML



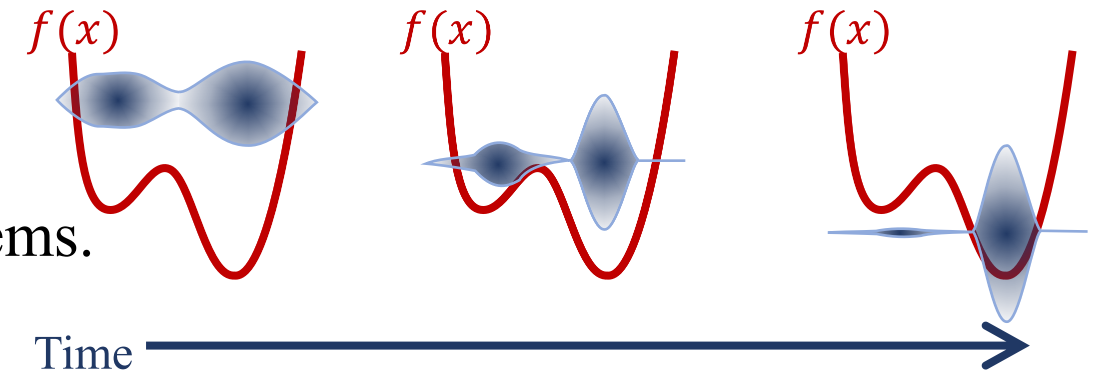
Decision-tree
CPU-friendly



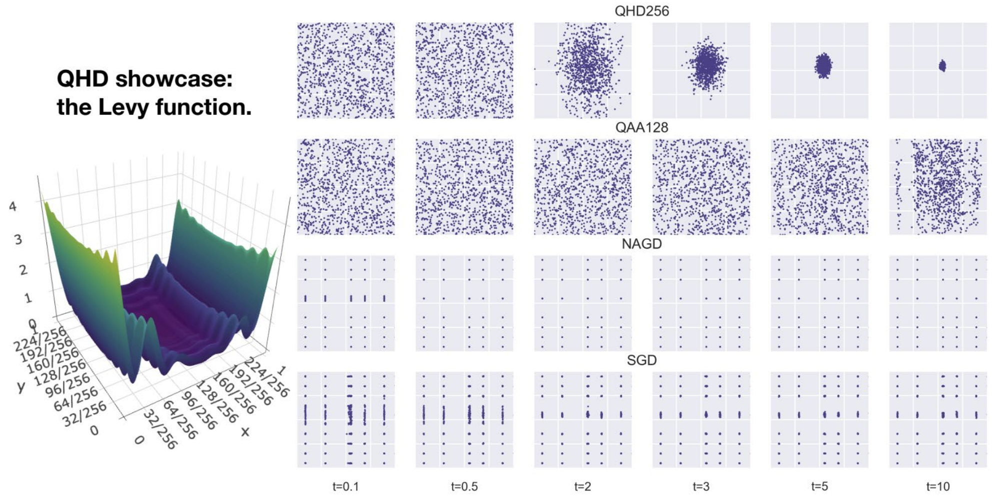
Neural networks
GPU-friendly

Quantum Hamiltonian Descent

- A quantum heavy-ball algorithm
  - A quantum particle falls to the minimum.
  - Target function embedded in quantum systems.
  - Exploits quantum tunneling effects.

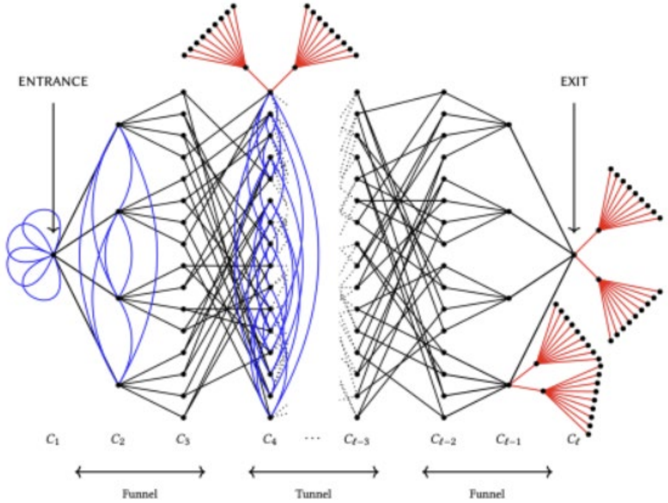


Empirical studies arXiv: 2303.01471



Often beats classical solvers on wall time

Theoretical analysis arXiv: 2504.14841



(Sub)exponential oracle separation

QHD main contributor: **Jiaqi Leng** (on the market)

More about QHD in A302, 1:45pm, Monday

From Quantum to Quantum Inspired

QHD dynamics

Quantum Hamiltonian

$$\hat{H}(t,x) = \frac{1}{\lambda(t)}\left(-\frac{1}{2}\Delta\right) + \lambda(t)V(x)$$

Schrödinger equation

$$i\hbar\partial_t|\Psi(t,x)\rangle = \hat{H}(t,x)|\Psi(t,x)\rangle$$

QIHD dynamics

Classical Hamiltonian

$$H(t,X,P) = \frac{1}{2\lambda(t)}\|P\|^2 + \lambda(t)V(X)$$

Hamilton's equations

$$\begin{cases} \dot{x} = \partial_p H \\ \dot{p} = -\partial_x H \end{cases}$$

- Hamiltonian dynamics
  - Bridge between quantum and classical mechanics
  - Different laws, similar forms

QIHD for MINLP

Classical Hamiltonian

$$H(t, X, P) = \frac{1}{2\lambda(t)} \|P\|^2 + \lambda(t)V(X) + V_{\text{pen}}(t, X)$$

- $\lambda(t)$: a monotonically increasing function.
- $V(x) = \begin{cases} f(x) & x \in \Omega \\ +\infty & x \notin \Omega \end{cases}$ is a potential function.
- $V_{\text{pen}}(t, X) \to M[X \in \Omega_{\text{feasible}}]$ when $t \to T$ for large M.

- Hamilton's equations (ODE): $\begin{cases} \dot{X}(t) = \frac{\partial H}{\partial P} = \frac{1}{\lambda(t)} P(t) \\ \dot{P}(t) = -\frac{\partial H}{\partial X} = -\lambda(t) f'(x) \end{cases}$
- Time discretization with symplectic Euler's method
- Randomness source: $\big(X(0), P(0)\big) \sim \rho\,(x, p)$

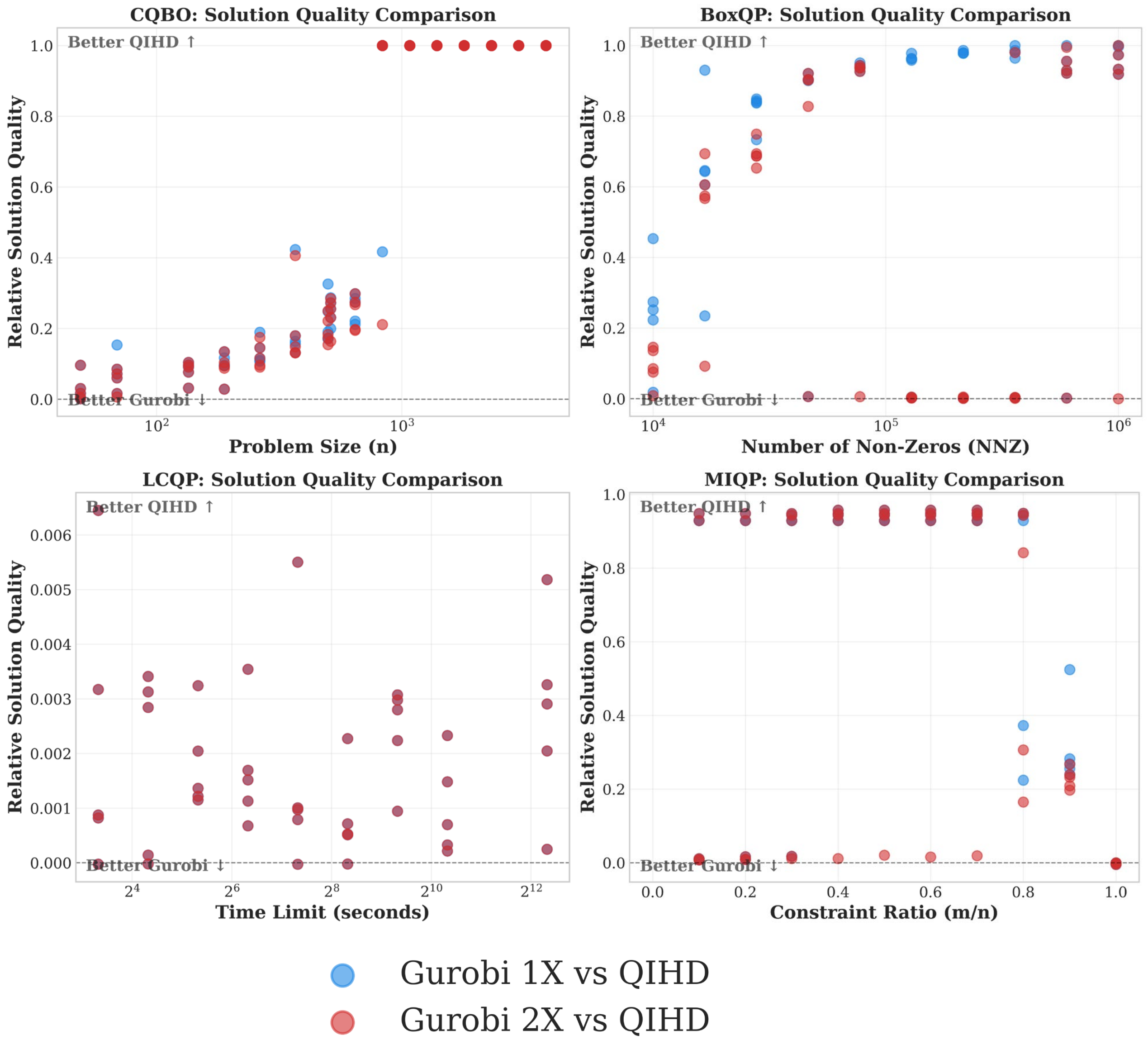
GPU-based Implementation

- QIHD: GPU-native metaheuristics for MINLP
  - Fully exploits GPU parallelism with samplings
  - Consists only matrix multiplication and vector operations
  - Minimal data transfer

- Open-source software: OpenPhiSolve
  - Based on JAX, a Python library for array computation
  - Post-processing with Optax / Mpax to boost precision
  - Features:
    - Easy deployment to CPUs, GPUs, and TPUs.
    - Automated distribution to multiple GPUs
    - Just-in-time compilation
    - Sparse matrix supports

Benchmarking

- Randomly synthesized problems
- Comparison against Gurobi
  - Gurobi uses limited time budget
    - 1x / 2x QIHD time
  - Relative gap $> 0 \iff$ QIHD is better
- Observations:
  - With short time budget, QIHD is almost always better than Gurobi
  - The advantage grows when problem is denser and larger.



Sweet Spot for QIHD

$$\min_{x\in\Omega} \frac{1}{2} x^T \begin{bmatrix} \cdot & \cdots & \cdot \\ \vdots & \ddots & \vdots \\ \cdot & \cdots & \cdot \end{bmatrix} x + w^T x$$

$$s.t. \begin{bmatrix} \cdot & \cdots & \cdot \\ \vdots & \ddots & \vdots \\ \cdot & \cdots & \cdot \end{bmatrix} x \leq b$$

- QIHD prefers the following properties:

Mathematically	Economically
• Dense data	• Light needs for global optimality
• Unstructured data	• Requires fast solvers
• Mild objective landscapes	• Requires frequent solving

- Two real-world applications
  - Satisfy these properties
  - Conventional solvers can hardly tackle
  - QIHD can find high-quality solutions with limited time-budget

Application: Model Sparsification

- Reduce LLM sizes
  - Smaller models, faster inference, less costs
  - SparseGPT [Frantar, 2023] uses sparse matrices to reconstruct outputs of dense ones
- Formulation:

$$\min_{\text{sparse } \widehat{M} \in \mathbb{R}^n} \sum_j \left\| M x_j - \widehat{M} x_j \right\|^2$$

  - Optimize masks (binary) and weights (continuous)
  - Better solution → higher model quality
  - 1.3B model optimization
    - ~16k binary and continuous variables
    - Dense & quadratic objectives
    - Constraints: sparse or semi-sparse masks
    - ~48k problem instances
- Existing methods: ad-hoc mask selection

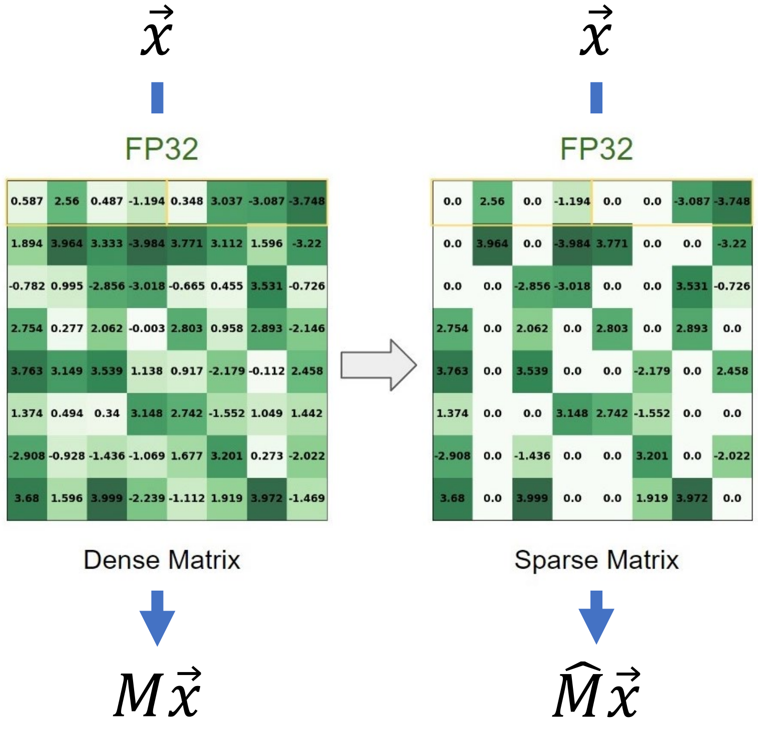
2024 United States Data Center Energy Usage Report

Arman Shehabi, Sarah J. Smith, Alex Hubbard, Alex Newkirk, Nuoa Lei, Md Abu Bakar Siddik, Billie Holecek, Jonathan Koomey, Eric Masanet, and Dale Sartor

Energy Analysis and Environmental Impacts Division, Lawrence Berkeley National Laboratory

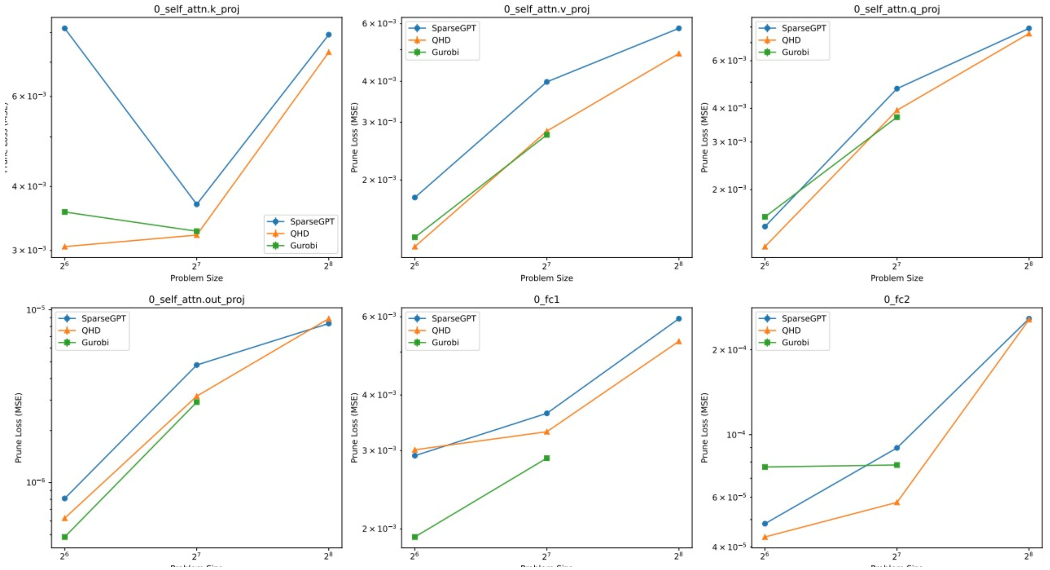
December 2024

Projected AI inference cost in 2028: \$82B

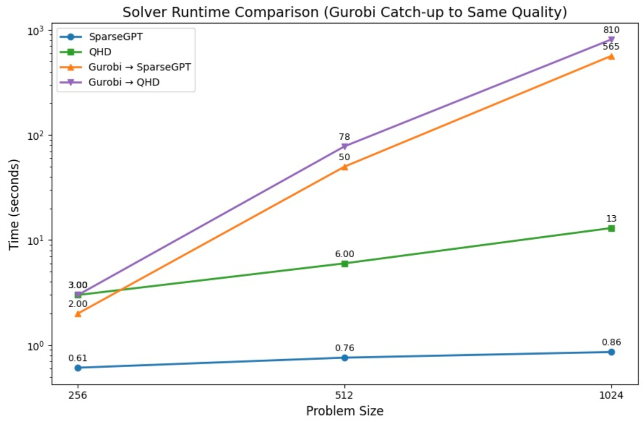


Application: Model Sparsification

- LLM experiments:
  - Solution quality:
    - QIHD is near-optimal: mostly matches Gurobi when Gurobi closes gap
    - Compared to SparseGPT, QIHD is 20% better for unstructured sparse and 10% better for semi-sparse
  - Time
    - QIHD: <1min for 16k vars
    - Gurobi: QIHD-quality solution needs 15 min for 1k vars (QIHD 13s)
    - QIHD is 50x-500x faster for practical size
- For 1.3B models, QIHD needs ~600 A10·hr to find near-optimal sparsified models

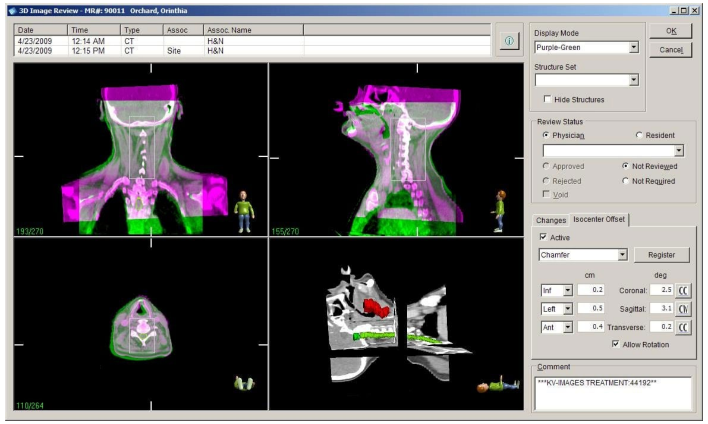


Prune loss: SparseGPT > QIHD ≈ Gurobi on avg.

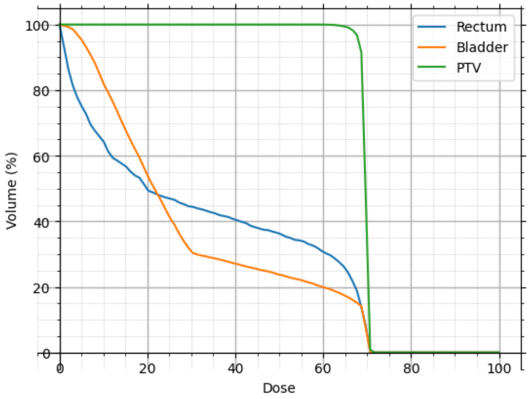


Solve time: Gurobi ≫ QIHD > SparseGPT on avg.

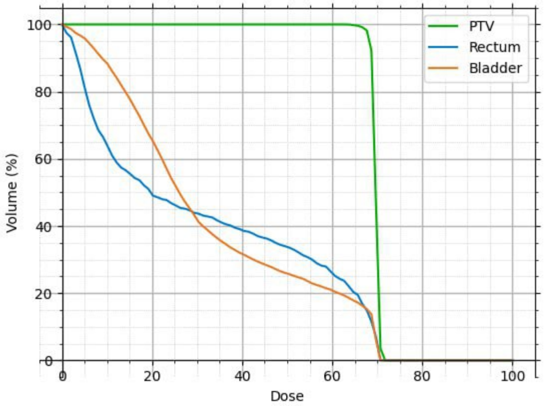
Application: Radiation Therapy



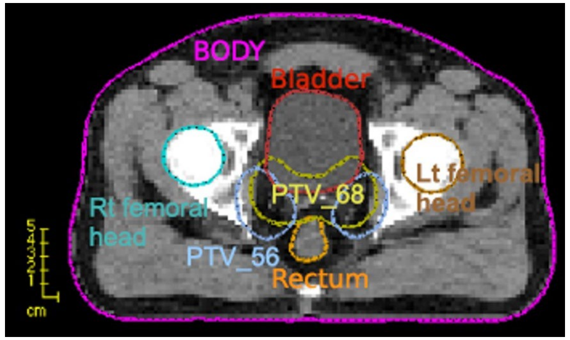
- Radiation therapy treatment planning
  - Damage tumors but protect organs-at-risks
  - Used in photon and proton therapy treatments
- Formulation:
  - MI LC linear/quadratic programming
  - Better solution → better patient outcomes
  - Size: $> 10^5$ binary and continuous variables.
  - Constraints: $\sim 10^5$; medical imaging data (“dense”)



Gurobi, 75 min



QIHD, 3 min



Prostate cancer case

Q & A

QHD dynamics

Quantum Hamiltonian

$$\hat{H}(t,x) = \frac{1}{\lambda(t)}\left(-\frac{1}{2}\Delta\right) + \lambda(t)V(x)$$

Schrödinger equation

$$i\hbar\partial_t|\Psi(t,x)\rangle = \hat{H}(t,x)|\Psi(t,x)\rangle$$

QIHD dynamics

Classical Hamiltonian

$$H(t,X,P) = \frac{1}{2\lambda(t)}\|P\|^2 + \lambda(t)V(X)$$

Hamilton's equations

$$\begin{cases} \dot{x} = \partial_p H \\ \dot{p} = -\partial_x H \end{cases}$$