Mathematical Optimization in the Era of AI

INFORMS International 2025

Singapore

Yinyu Ye
SJTU and Stanford

Today’s Talk

- **Online Linear Programming: Learning and Decision Making in Real Time**

- **Online Hyper-Gradient Method: Theory and Practice**

- **Optimization Solvers on GPU: Preliminary Results**

Linear Programming and LP Giants

$$\max \quad \sum r_t x_t$$

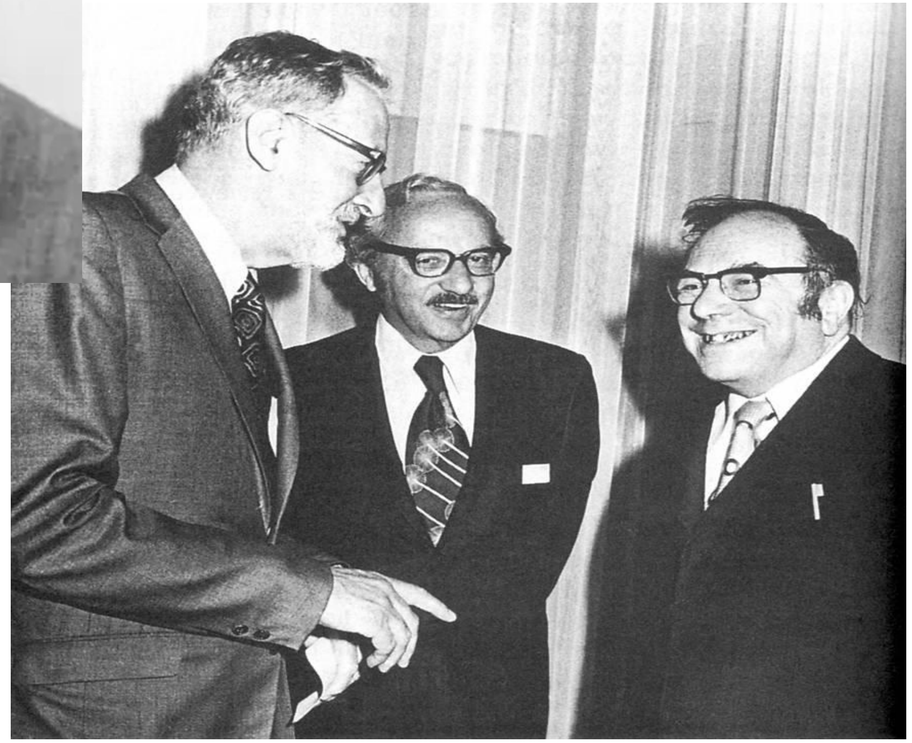
$$\text{s.t.} \quad \sum_t \boldsymbol{a}_t x_t \leq \boldsymbol{b}\,,$$

$$0 \leq x_t \leq 1 \quad \forall\ t = 1, \ldots, T$$

Von Neumann

Linear Objective Function
Set of linear constraints
x_t: decision variables
Data: r_t, $\mathbf{a}_t$, $\mathbf{b}$

Koopman
Dantzig
Kantorovich



A Toy Resource-Allocation Example

- **Jobs need resources and return rewards**
- **Each job request have to be decided: grant it or not**
- **The offline problem (hindsight performance) makes decisions after all requests received**

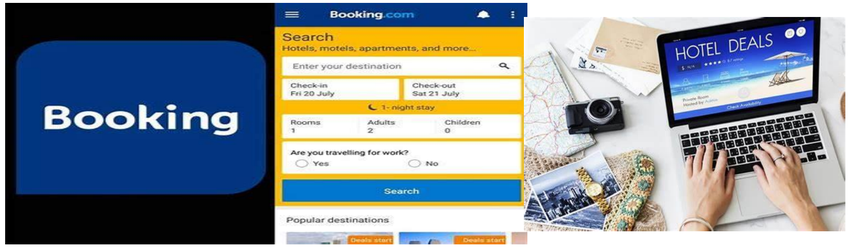
Job #	$100	$30	….	…	…	Inventory	
Decision	x1	x2					
R1	1	0	….	…	…	100	
R2	1	0				50	
R3	0	1				500	
R4	0	0				200	
R5	1	1	…	…	…	1000	

This can be modeled as a binary-integer LP problem

Online Linear Programming

Online setting:

- Requests arrive sequentially and the decision needs to be made instantly upon the the arrival (x_t can be a vector representing complex decisions)
- The decision is irrevocable
- The offline (MILP) total reward is an upper bound for online decisions



$$\max \sum_{t=1}^{T} r_t x_t$$

$$\text{s.t.} \sum_{t=1}^{T} a_{it} x_t \le b_i, \quad i = 1, \ldots, m$$

$$0 \le x_t \le 1 \quad \text{or} \quad x_t \in \{0, 1\}, \quad t = 1, \ldots, T$$

Performance of online algorithm measured with respect to (average) regret from the offline optimal objective, and there is an online algorithm and it is provably optimal when agents' arriving order is randomly permuted.

[Agrawal et al. 2010, 2014], [Kesselheim et al., 2014]
[Li/Y, 2019], [Li et al., 2020],

The Price-Based Decision Rule

- **The key is to learn "ideal" itemized-prices**
- **Use the prices to price each bid**
- **Accept if it is an over bid, and reject otherwis**

$$x_j^* \in \begin{cases} \{0\}, & c_j - a_j^\top y^* < 0 \\ [0,1] & c_j - a_j^\top y^* = 0 \\ \{1\} & c_j - a_j^\top y^* > 0 \end{cases}$$

Job #	$100	$30	….	…	…	Inventory	Price y*
Decision	x1	x2					
R1	1	0	….	…	…	100	45
R2	1	0				50	45
R3	0	1				500	10
R4	0	0				200	55
R5	1	1	…	…	…	1000	15

Such ideal prices exist and they are shadow/dual prices of the offline LP - How to Learn?

Learning-while-Acting: Periodically LP-Resolving

At time t, solve the sample LP at t=εT, 2εT, 4εT, …; and use the new shadow prices for the decision in the coming period (the resource allocations can be adaptively adjusted) where T is the total # of buyers.

$$\max \quad \sum_{k=1}^{t} \pi_k x_k$$

$$\text{s.t.} \quad \sum_{k=1}^{t} a_{ik} x_k \le (1-h_t)\frac{t}{T} b_i \quad \forall\ i$$

$$0 \le x_k \le 1 \qquad \forall\ k \in 1, \dots, t$$

b_i/T: **average resource per job**

The improvement is to adaptively adjust average resource per job

The absolute regret is $O(T^{1/2})$ or O(log T) when b_i=O(T)

O(1) Regret Result for Finite Job Support

- Now let's assume that $\{(\boldsymbol{a}_t, r_t)\}_{t=1}^{T}$ come from a distribution that has finite (with a total of J) categories, and $P\left((\boldsymbol{a}_t, r_t) = (\boldsymbol{c}_j, \mu_j)\right) = p_j$.
- For this case, we construct the fluid approximation LP

Primal LP

$$\begin{aligned} \max_{x} \quad & r^\top x \\ \text{subject to} \quad & Ax \le b \\ & 0 \le x \le 1 \end{aligned}$$

Fluid approximation LP Learning

$$\begin{aligned} \max \quad & \sum_{j=1}^{J} p_j \mu_j y_j \\ \text{s.t.} \quad & \sum p_j \boldsymbol{c}_j y_j \le \boldsymbol{b}/T \\ & 0 \le y_j \le 1 \ \text{ for all } j \end{aligned}$$

- The offline decision rule: if $(\boldsymbol{a}_t, r_t) = (\boldsymbol{c}_j, \mu_j)$, the optimal decision is $x_t = y_j^*$
- Online: at time t, we replace $\boldsymbol{b}$ and p_j by $\boldsymbol{b}_t$ (the remaining resource) and $\hat{p}_j$ (the **sample estimation** of p_j) and resolve the LP for $\boldsymbol{y}_t^*$, respectively
- Our decision at t will be based on the solution $\boldsymbol{y}_t^* = (y_{1,t}^*, \cdots, y_{J,t}^*)$.

Online Learning and Decision Making Algorithms

First LP-based resolving methods (PT)
[Agrawal et al. 10, 14]

- Solve increasingly large LPs
- $O(\sqrt{T})$ performance
- Minimal assumptions

Later LP-based resolving methods (PT)
[Li & Ye 19][Jiang et al. 22][Chen et al. 22][Bray 22][Ma et al. 22]

- Solve increasingly large LPs with adaptively adjusted inventories
- $O(\log T)$ or $O(1)$ performance
- Require non-degeneracy

LP-Resolving methods perform well, but they may take too much time

Solving an LP with T columns is often unacceptable

Recent first-order methods (FT)
[Li et al. 20][Lobos et al. 21][Balseiro et al. 22]
[Gao et al. 22][Ma et al. 22]

- No LP solving and very fast update
- $O(\sqrt{T})$ performance
- Minimal assumptions

First-order Online LP Methods: Online Gradient Descent

$$\min_{p\geq 0} b^T p + \sum_t [r_t - a_t^T p]_+ \quad \Longrightarrow \quad \min_{p\geq 0} \sum [(b/T)^T p + [r_t - a_t^T p]_+] =: \sum f_t(p)$$

Start with some p_1, at each t, the first-order based algorithms:

Decision-Making based on the current dual estimate: $\hat{x}_t = I\{r_t \geq a_t^T p_t\}$

Learning or updating the prices by **OGD** with step-size α_t: $p_{t+1} = [p_t - \alpha_t f_t'(p_t)]_+$

$O(\sqrt{T})$ performance under minimal assumptions
[Li et al. 2020][Lobos et al. 2021]
[Balseiro et al. 2022][Gao et al. 2022]

Recall LP-Resolving methods achieve beyond $O(\sqrt{T})$ under certain growth assumptions.

Can first-order methods match up?

Decoupling Learning and Decision-Making

A good ***L***earning algorithm: stationarity

$\mathcal{A}^L$

$\mathcal{A}^D$

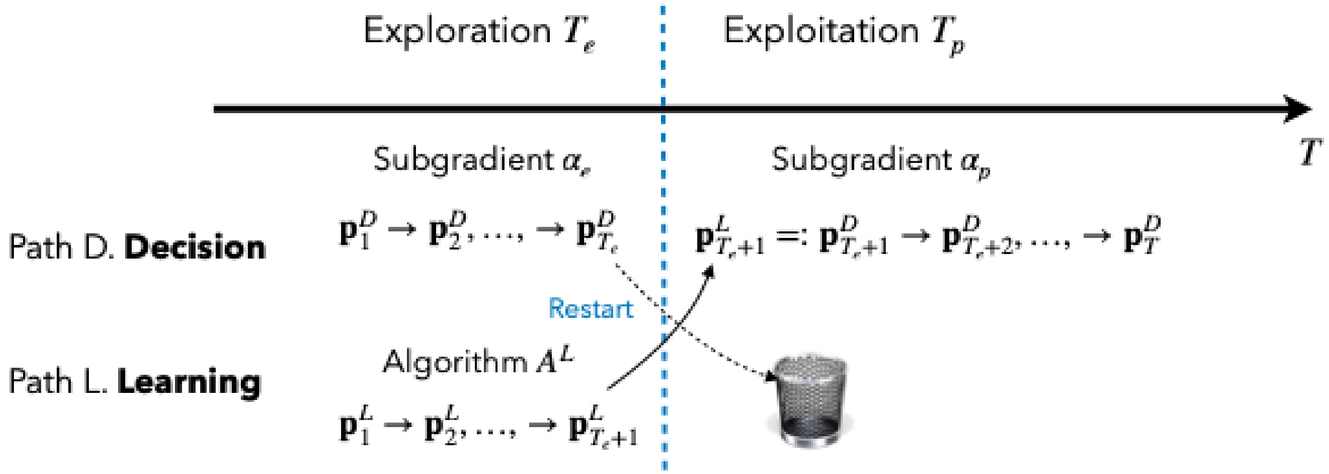
A good ***D***ecision algorithm: adaptivity

- Maintain two paths of dual price solutions

  Learning path: $\{p_t^L\}$ Decision path: $\{p_t^D\}$

- Both are updated via the OGD methods but with different step-sizes

- Periodically pass $\{p_t^L\}$ to the decision process as the new starting price solution



Main result (Gao et al. 2024, 2025)

Theorem: There is a first-order OLP method with

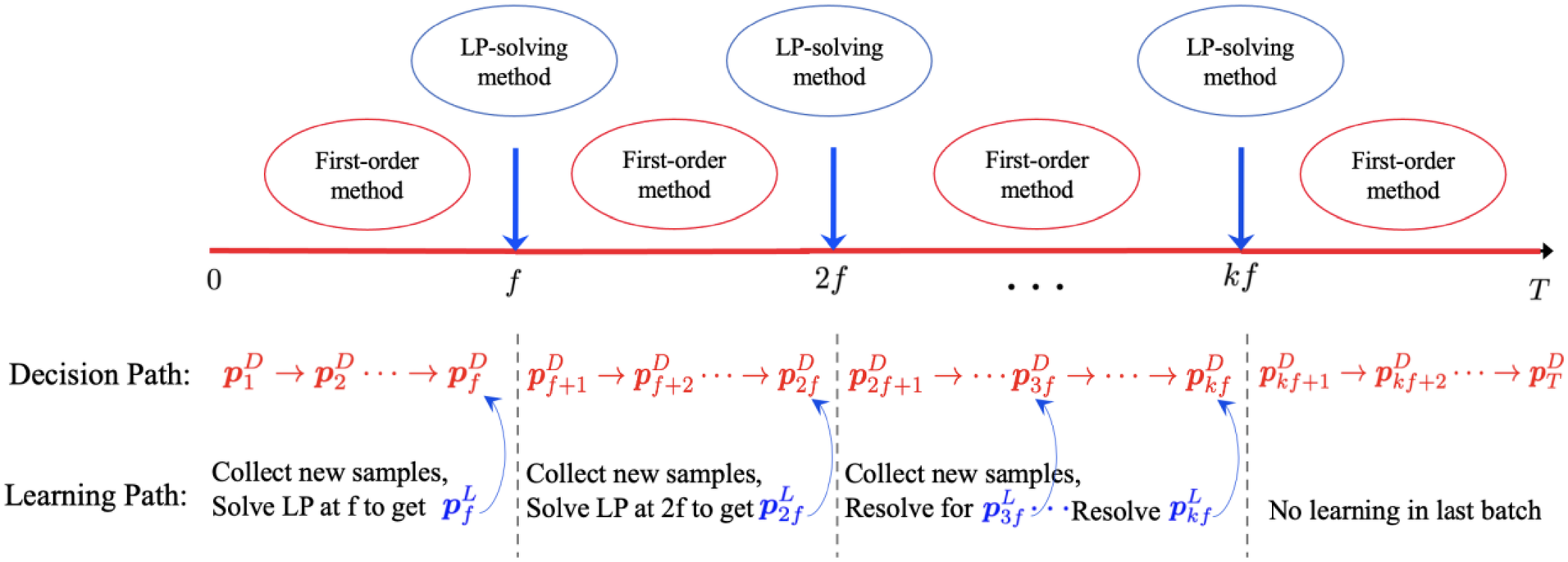
- $O(T^{1/3})$ regret for non-degenerate continuous support OLP
- $O(\log T)$ regret for non-degenerate finite support OLP
- $O\left(T^{\frac{\gamma-1}{2\gamma-1}}\right)$ for OLP with γ-dual error bound

$$f(p) - f(p^\star) \geq \mu||p - p^\star||^\gamma$$

Can we take the best-of-both-worlds for the first-order and LP-resolving methods?

Parallel Multi-Phase Algorithm (Sun et al. ICML 2025)

Multi-time restart strategy: solve LP every f customer, use first-order method in the middle



- Learning Path: collect new samples and **resolve** linear programs
- Decision Path: apply first order method to make instant decisions without wait
- **Learning Path** periodically updates dual solution for **Decision Path** to **"restart"**

Main Result

Theorem 2. *Under the assumption of non-degeneracy, denote f as the re-solving frequency, then the worst-case regret of the algorithm is bounded by:*

$$\mathbb{E}[\text{Regret} + \text{Violation}] \leq \mathcal{O}\left(\log\left(\frac{T}{f}\right) + \sqrt{f}\right)$$

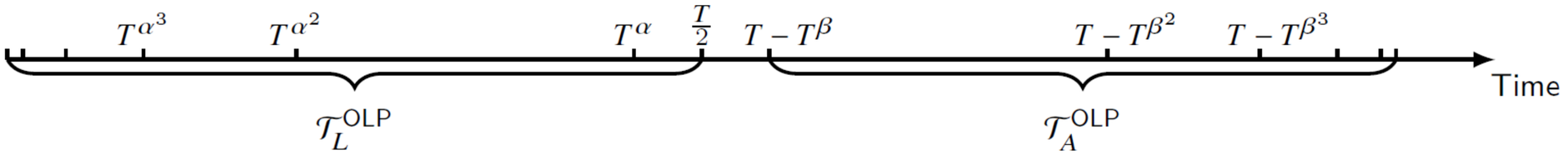
- Control f as the trade-off between decision regret and computation efficiency
- User's computation resource budget can determine the optimal f and best available regret

$$\min_{f \in \{1, \ldots, T\}} \quad \log\left(\frac{T}{f}\right) + \sqrt{f}$$

$$s.t. \qquad \text{LP cost} + \text{First order cost} \leq \text{Total Computation Resource}$$

More on LP-Resolving Frequency (Li et al. 2024)

LP resolving frequency



- First part is to learn the demand/reward, the second part is to balance inventory
- If the demand is known, then only second part is necessary
- Can achieve O(1) regret with O(loglogT) LP-resolvings

Online LP has been used in real applications, and extended to solve Bandit with Knapsack, Economic Equilibrium Markets, Resource Allocation with Supply Chain Network, Return Job with Markovian Behavior, Reusable Resource and Scheduling, etc. …

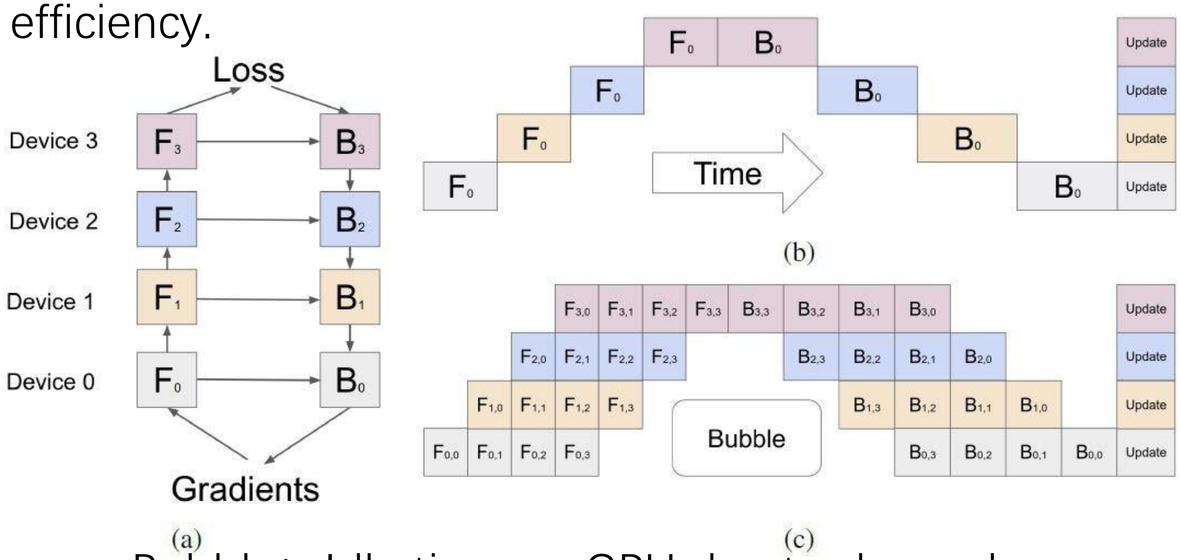
Application I: Resource Allocation in LLM Training

In order to train larger models and train faster, multiple GPUS and multiple nodes are utilized in parallel. There are various **parallelism strategies**, including

- Data Parallelism, DP
- Tensor Parallelism, TP
- Expert Parallelism, EP
- **Pipeline Parallelism, PP**

In practice, they are usually combined.

There is an inherent problem of PP, that due to the **dependency** between computations, there are always some devices **forced to be idle**, resulting a lower efficiency.

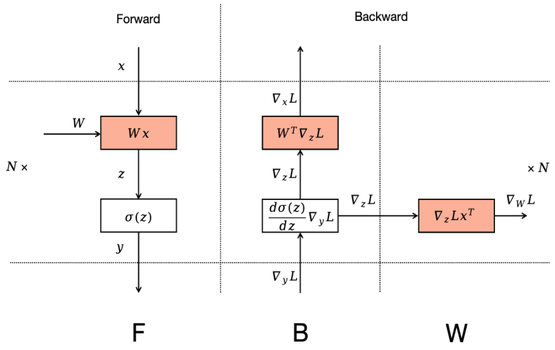


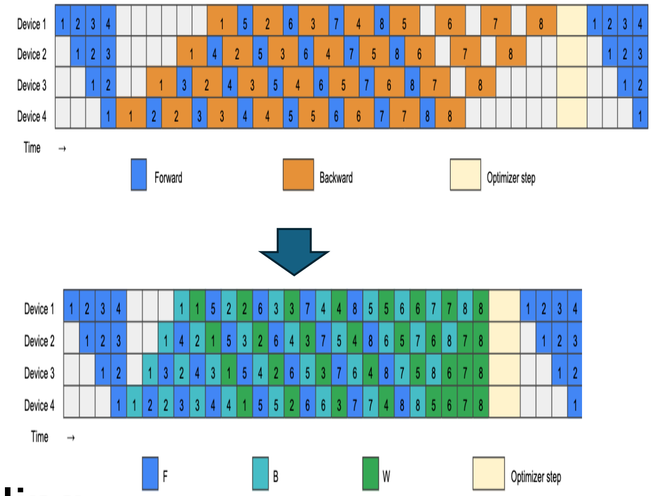
Bubble：Idle time on GPU due to dependency.

In order to improve efficiency, bubbles need to be reduced.

Zero Bubble

Found out that the backward stage could be spitted into W and B, so that bubbles are reduced.





Automatic Pipeline Scheduling

Starting from Zero Bubble, it sets up a MILP model considering parameters including time span of each stage, and solves the problem with MILP solver.

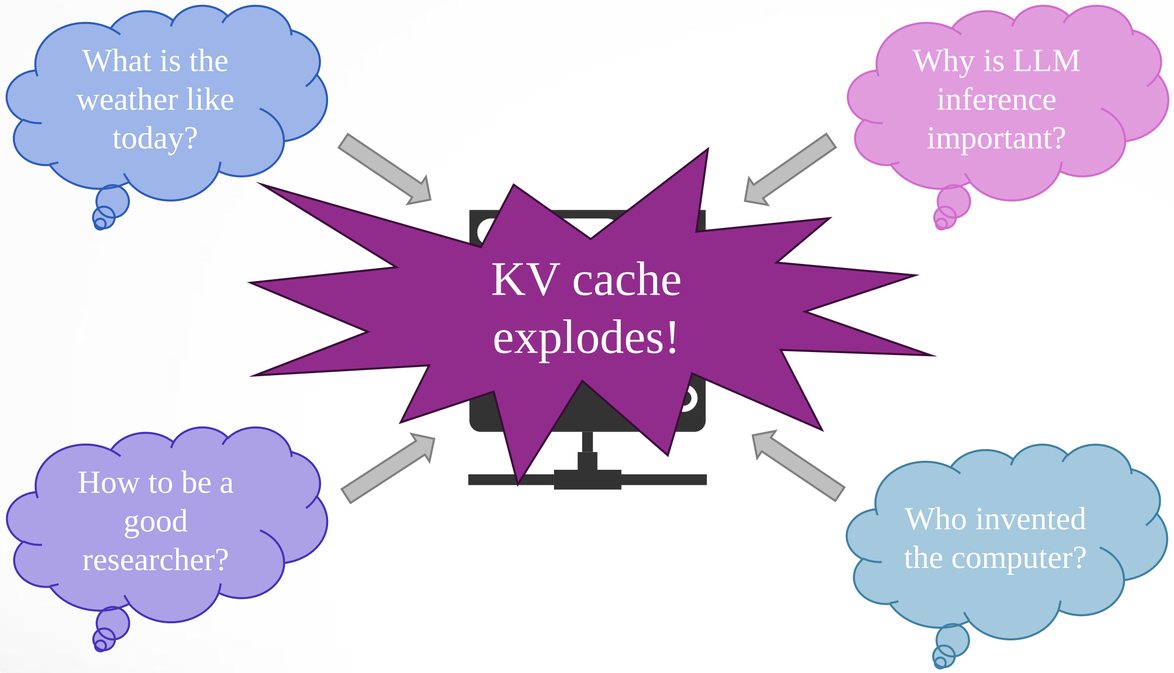
$$\min_{O,E} \quad \max_i E_{(i,m,W)} - E_{(i,1,F)} + T_{(i,1,F)}$$

$$\begin{aligned} s.t. \quad & E_{(i,j,F)} \geq E_{(i-1,j,F)} + T_{\text{comm}} + T_{(i,j,F)} \\ & E_{(i,j,B)} \geq E_{(i+1,j,B)} + T_{\text{comm}} + T_{(i,j,B)} \\ & E_{(i,j,c)} \geq E_{(i,j',c')} + T_{(i,j,c)} - O_{(i,j,c)\to(i,j',c')}\infty \\ & M_{\text{limit}} \geq \Delta M_{(i,j',c')} + \sum_{j,c} \Delta M_{(i,j,c)} O_{(i,j,c)\to(i,j',c')} \end{aligned}$$

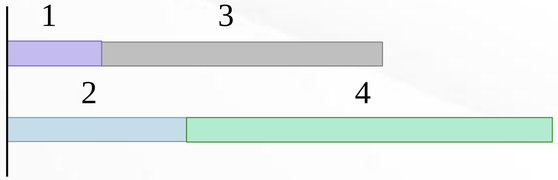
20-40% improvement!

Application II: LLM Inference Scheduling

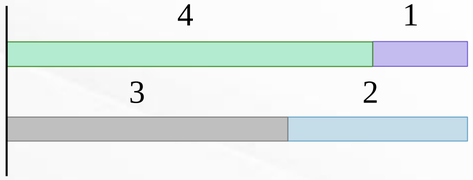
LLM Serving with Variable Prefill and Decode Lengths



Example: Different schedule brings different total end-to-end latency.



$$TEL_1 = 1 \cdot 2 + 2 \cdot 2 + 3 \cdot 1 + 4 \cdot 1 = 13$$



$$TEL_2 = 4 \cdot 2 + 3 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 17$$

User Experience:

- End-to-end latency.
- Time to the first token.
- Time between tokens.

Cost:

- Chatgpt receives 10 million prompts per day.
- Cost of LLM inference on each prompt is about $0.07.
- Daily LLM inference cost ~ $700000.

Energy and Sustainability:

- Chatgpt consumes > 500,000 kilowatts daily.
- Daily power usage ≈ 180000 households.
- A single conversation uses 50cL water, equivalent to one plastic bottle.

Traditional schedulers like FCFS fail to scale with LLM workloads - we need better approaches!

New Algorithm (Wang et al. 2025)

LLM Serving with Variable Prefill and Decode Lengths

Baseline Algorithm: When s_i is fixed, Shortest-First is proved to be optimal (Zhou 2024). However, when s_i is variable...

Theorem 2. Algorithm Shortest-First (`MC-SF`) achieves an unbounded competitive ratio.

New Algorithm:

At the core of the new algorithm `Sorted-F` is the quality metric $F(\mathcal{X})$ for any request set $\mathcal{X}$, defined as

$$F(\mathcal{X}) = \frac{\sum_{r_i \in \mathcal{X}} o_i}{|\mathcal{X}|^2}.$$

Smaller values of $F(\mathcal{X})$ indicate higher scheduling priority. This metric fundamentally improves upon `MC-SF` by jointly optimizing low average response lengths and high batch throughput.

while $\mathcal{I}$ **do**

$\mathcal{X} \leftarrow \arg\min_{\mathcal{X} \subseteq \mathcal{I}} (F(\mathcal{X}), -|\mathcal{X}|)$ **subject to** $M(\mathcal{X}, p_i + o_i) \leq M, \forall r_i \in \mathcal{X}$

Sort $\mathcal{X}$ in ascending order of o_i

$\mathcal{I}' \leftarrow \mathcal{I}' + \mathcal{X}, \mathcal{I} \leftarrow \mathcal{I} - \mathcal{X}$

end while

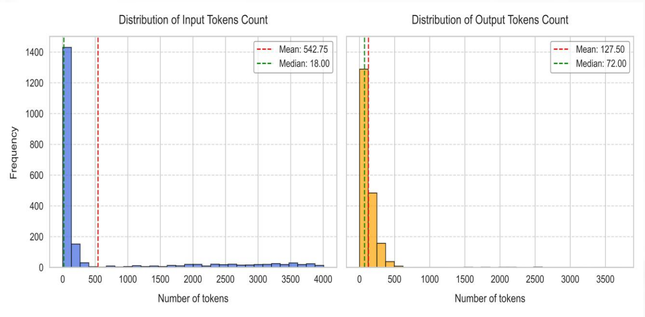
In the new algorithm `Sorted-F`, Phase 1 (shown above) sorts requests by iteratively selecting from the unsorted pool the sublist $\mathcal{X}$ that minimizes $F(\mathcal{X})$ and ordering requests in $\mathcal{X}$ by ascending o_i. Following this order, Phase 2 schedules requests sequentially at earliest feasible times while respecting memory constraints.

Theorem 3. Algorithm `Sorted-F` achieves a **constant competitive ratio** upper bounded by 48 against the optimal schedule, i.e.,

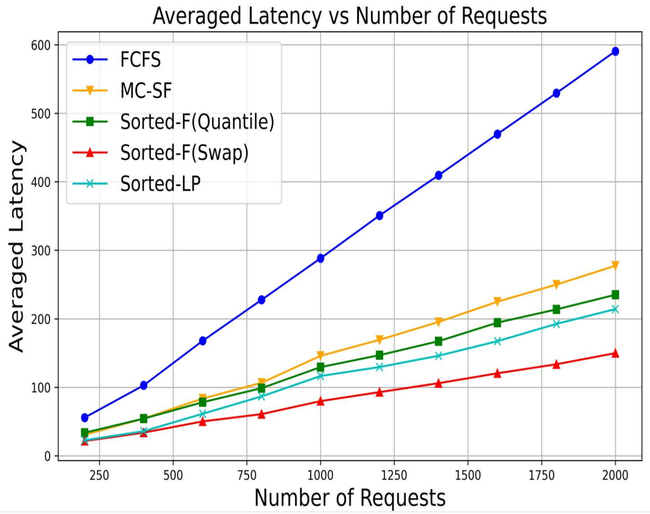
$$\mathrm{CR}(\texttt{Sorted-F}) < 48.$$

Numerical Experiments:

Distribution of the number of tokens of input prompt and output response respectively in the mixed dataset:



:



Robust LLM Inference Optimization under Uncertainty (Chen et al. 2025)

Setting

- Single inference worker with KV cache of size M
- n jobs with known input size s; unknown output length in $[l,u]$
- Define uncertainty: $\alpha=l/u$
- Jobs run in batches (1 time unit per batch)
- Memory usage must stay within M; cancellation and restart is allowed

Naive Benchmark Algorithm: A_{max}

- **Key Idea**: Assume worst-case output length $\boldsymbol{o_i = u}$ for all jobs.
- **Theorem**:

  The competitive ratio of $\boldsymbol{A_{max}}$ satisfies:

$$\frac{\alpha^{-1}\left(1+\alpha^{-1/2}\right)}{2} \leq CR(\boldsymbol{A_{max}}) \leq \frac{\alpha^{-1}(1+\alpha^{-1})}{2} + O(\frac{1}{M})$$

Robust Algorithm: A_{min}

- **Key Idea**: Use the lower bound ℓ to estimate memory demand more optimistically. Dynamically refine this bound: set $\tilde{\boldsymbol{o}}_{\boldsymbol{i}} = \ell$, then increase $\tilde{\boldsymbol{o}}_{\boldsymbol{i}}$ as output is generated.
- **Algorithm Outline**:

  1.Batch formation: Greedily add requests in order of increasing $\tilde{\boldsymbol{o}}_{\boldsymbol{i}}$.

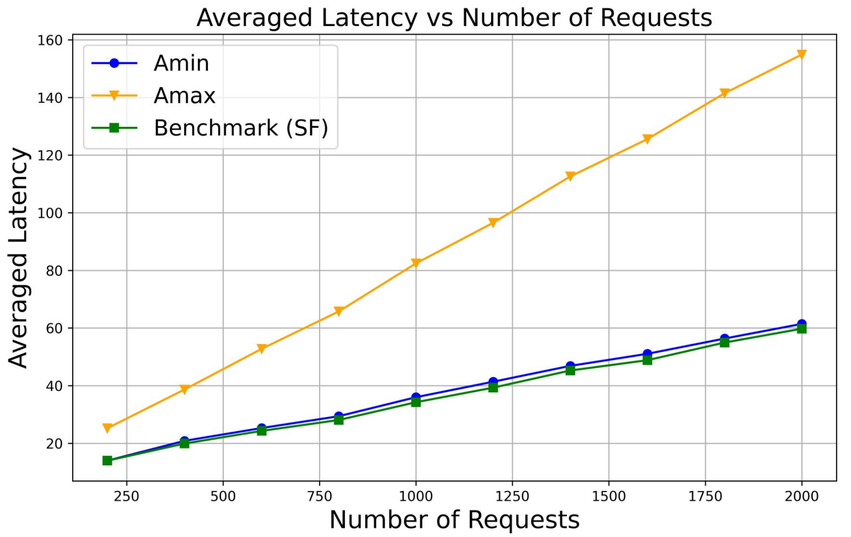
  2.Overflow resolution: Delete requests with smallest $\tilde{\boldsymbol{o}}_{\boldsymbol{i}}$ if memory overflows.

- **Theorem (Asymptotic Optimality):** For any online policy π without access to true output lengths, $CR(\pi) \geq CR(\boldsymbol{A_{min}})$ as $M \to \infty$.

- **Theorem (Asymptotic Upper Bound)**: Let $\vec{x} = (x_1, \ldots, x_u)$, under the assumption $s = 0, l = 1$, the competitive ratio of $\boldsymbol{A_{min}}$ simplifies to a Rayleigh quotient,

$$CR(\boldsymbol{A_{min}}; \boldsymbol{D}) = \frac{\vec{x}' \mathrm{A}_u \vec{x}}{\vec{x}' \mathrm{B}_u \vec{x}} + O(\frac{1}{M}).$$

And we prove that $CR(\boldsymbol{A_{min}}; \boldsymbol{D}) = \boldsymbol{O(log\, u)}$ as $\boldsymbol{u} \to \infty$.



$\boldsymbol{A_{min}}$: achieves average latency nearly identical to a selected benchmark with more information.

Today’s Talk

- **Online Linear Programming: Learning and Decision Making in Real Time**
- **Online Hyper-Gradient Method: Theory and Practice**
- **Optimization Solvers on GPU: Preliminary Results**

Two Dominating AI Training Methods: SGD and Adam

- **Consider** $\min_x f(x) := \sum_{i=1}^n f_i(x)$

  n: number of samples (or mini-batches of samples). x: trainable parameters

- **In the k-th iteration: Randomly sample τ_k from $\{1, 2, \dots, n\}$**

- SGD with momentum (SGD):
- $m_k = (1-\beta_1)\nabla f_{\tau_k}(x_k) + \beta_1 m_{k-1}$
- $x_{k+1} = x_k - \eta_k m_k$

- Adam (Kingma and Ba'15):
- $m_k = (1-\beta_1)\nabla f_{\tau_k}(x_k) + \beta_1 m_{k-1}$
- $v_k = (1-\beta_2)\nabla f_{\tau_k}(x_k) \circ \nabla f_{\tau_k}(x_k) + \beta_2 v_{k-1}$
- $x_{k+1} = x_k - \eta_k \frac{m_k}{\sqrt{v_k}}$

In practice: further consider weight decay: $\min_x f(x) := \sum_{i=1}^n f_i(x) + \frac{\lambda}{2}||x||_2^2$

- AdamW (Loshchilov and Huntter'17):
- $m_k = (1-\beta_1)\nabla f_{\tau_k}(x_k) + \beta_1 m_{k-1}$
- $v_k = (1-\beta_2)\nabla f_{\tau_k}(x_k) \circ \nabla f_{\tau_k}(x_k) + \beta_2 v_{k-1}$
- $x_{k+1} = x_k - \eta_k \left(\frac{m_k}{\sqrt{v_k}} + \lambda x\right)$

Stepsize/Learning rate

Adam is the Default for LLM

Scaling Decision in Gradient Descent

$$x^{t+1} = x^t - P_t \nabla f(x^t)$$

Hyper-parameter

such as step-size, precondition, quasi Newton,…

We write the optimality gap at step $T+1$ as

$$f(x^{T+1}) - f(x^\star) = [f(x^1) - f(x^\star)] \prod_{t=1}^{T} \frac{f(x^{t+1}) - f(x^\star)}{f(x^t) - f(x^\star)}$$

Function of P

The accumulated ratio product would be a function of *P*, which we like to minimize over *P*, but it is **not convex**…

Therefore we apply **arithmetic-geometric** mean inequality

$$f(x^{T+1}) - f(x^\star) \leq [f(x^1) - f(x^\star)] \left(\frac{1}{T} \sum_{t=1}^{T} \frac{f(x^{t+1}) - f(x^\star)}{f(x^t) - f(x^\star)} \right)^T$$

Online Scaling in Gradient Descent

$$f(x^{T+1}) - f(x^\star) \le [f(x^1) - f(x^\star)] \left(\frac{1}{T} \sum_{t=1}^{T} \frac{f(x^{t+1}) - f(x^\star)}{f(x^t) - f(x^\star)} \right)^T$$

To optimize the progress/trajectory, we now choose $\{P^t\}$ to minimize **the convex sum**

$$\sum_{t=1}^{T} \frac{f(x^{t+1}) - f(x^\star)}{f(x^t) - f(x^\star)} = \sum_{t=1}^{T} \frac{f(x^t - P^t \nabla f(x^t)) - f(x^\star)}{f(x^t) - f(x^\star)}$$

Define the **convergence rate as** $r_x(P) := \frac{f(x - P\nabla f(x)) - f(x^\star)}{f(x) - f(x^\star)}$, a convex function in P,

To minimize the cumulated ratios $\sum_{t=1}^{T} r_{x^t}(P^t)$, we again run **Online Gradient Descent (OGD)**

$$P^{t+1} = P^t - \eta \nabla r_{x^t}(P^t)$$

Online Scaling Gradient Method (OSGM)

Input: $x^0, P^0, \eta > 0$, $\mathbf{P}$ (not necessarily positive semidefinite)

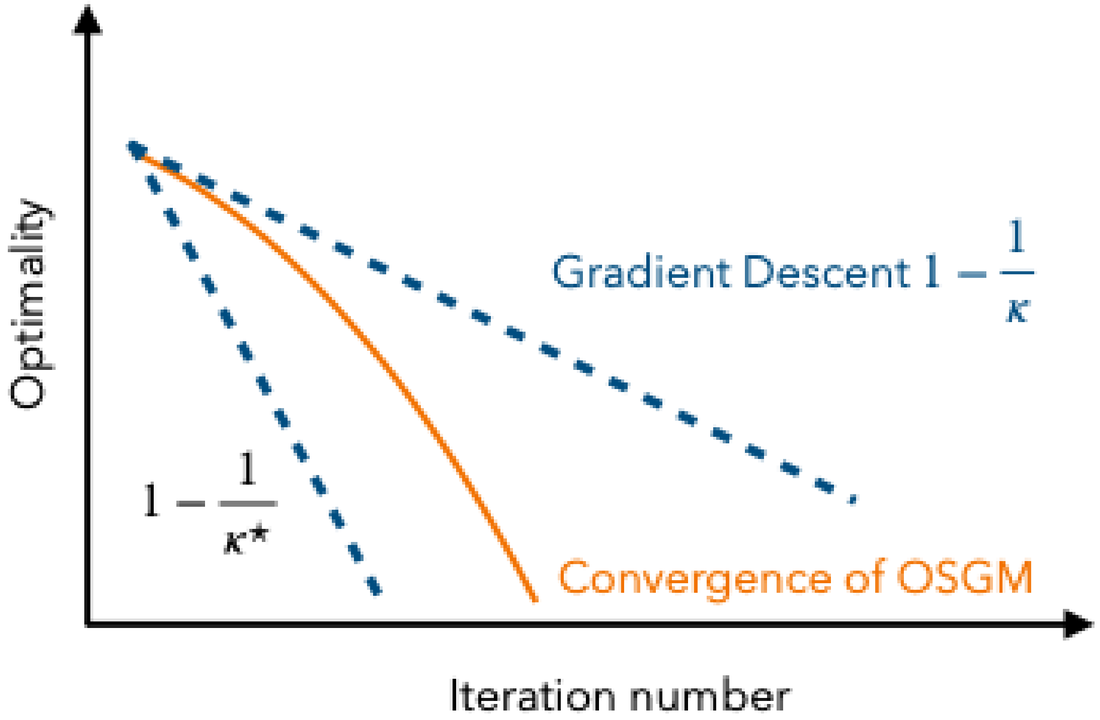
- $x^{t+1} = x^t - P^t \nabla f(x^t)$
- $P^{t+1} = \Pi_{\mathbf{P}}[P^t - \eta \nabla r_{x^t}(P^t)]$

$$\nabla r_x(P) = -\frac{\nabla f\big(x - P\nabla f(x)\big)\nabla f(x)^\top}{f(x) - f(x^\star)}$$

Theorem. $f(x^{T+1}) - f(x^\star) \le [f(x^1) - f(x^\star)]\left(1 - \frac{1}{\kappa^\star} + O\left(\frac{1}{\sqrt{T}}\right)\right)^T$ and the complexity of finding an ε-optimal point is $O\left(\kappa^\star \log\frac{1}{\varepsilon}\right)$. ***Here*** $1 - \frac{1}{\kappa^\star}$ ***is the contraction achievable by the optimal step-size***

- P can be taken to be sparse (e.g., diagonal), and we only need the corresponding elements of the rank-one outer product
- The same gradient oracle complexity as gradient descent and almost **no overhead**

Online Scaling Gradient Methos (OSGM)



The algorithm starts at the same rate as gradient descent

Online learning will learn and gradually achieve $1-\frac{1}{\kappa^\star}$ convergence

Learning-First and Acting-Second

In our setting, we can switch the order of updates:

- $x^{t+1} = \Pi_P[P^t - \nabla f(x^t)(P^t)]$
- $x^{t+1} = \Pi_P[P^{t+1} - \eta \nabla f(x^t)(P^t)]$

That is, we can learn first, and make the x-update second

Theorem. ***"Online learning" has*** $O(1)$ ***regret and***

$$f(x^{T+1}) - f(x^\star) \leq [f(x^1) - f(x^\star)]\left(1 - \frac{1}{\kappa^\star} + o\left(\frac{1}{T}\right)\right)^T$$

Additional Convergence Results

The online scaled gradient method shares several properties with the well-known quasi-Newton family

Theorem. ***Under standard regularity conditions and suppose the planner decision set*** $\mathbf{P} = \boldsymbol{R^{n\times n}}$***. Then online scaled gradient method with ratio feedback has local superlinear convergence*** $f(x^{K+1}) - f(x^\star) \leq [f(x^1) - f(x^\star)]O\left(\frac{1}{\sqrt{T}}\right)^{T}$.

Theorem. ***Under standard regularity conditions and suppose the planner decision set*** $\mathbf{P} = \boldsymbol{R^{n\times n}}$***. Then*** $P_k \to (\nabla^2 f(x^\star))^{-1}$.

Quasi-Newton works by learning $(\nabla^2 f(x^\star))^{-1}$. If $(\nabla^2 f(x^\star))^{-1}$ is a candidate decision of the planner, it works no worse than quasi-Newton on average

Compatibility with Other Acceleration Techniques

The method allows optimizing any learnable algorithm hyper-parameters

Gradient descent with **momentum** (Polyak 1967, Nexterov 1982)

$$x^{k+1} = x^k - P^k \nabla f\left(x^k\right) + \beta^k (x^k - x^{k-1})$$

Define $h_{x,x^-}(P,\beta) = \frac{f(x - P\nabla f(x) + \beta(x - x^-)) - f(x^\star)}{f(x) - f(x^\star)}$

- P-agent decides (P^k, β^k): $x^{k+1} = x^k - P^k \nabla f\left(x^k\right) + \beta^k (x^k - x^{k-1})$
- Executioner provides feedback $h_{x^k,x^{k-1}}\left(P^k, \beta^k\right)$
- P-agent learns from the feedback $(P^{k+1}, \beta^{k+1}) = \left(P^k, \beta^k\right) - \eta \nabla h_{x^k,x^{k-1}}\left(P^k, \beta^k\right)$

Also compatible with any other scaling GMs such as ADAM…

PDLP with Online Scaling (Lu and Zhang 2025)

Consider solving an offline LP:

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Saddle-point formulation:

$$\min_{x \geq 0} \max_{\lambda} L(x, \lambda) = c^T x - \lambda^T A x + b^T \lambda$$

Define online feedbacks as:

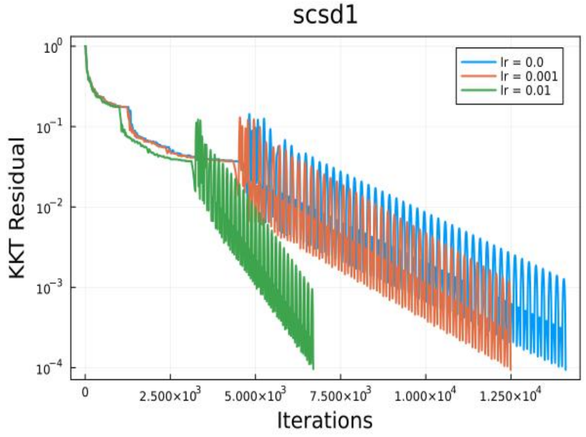
$$\ell_k^p(T_k) = L\left(x^{k+1}(T_k), \lambda^k\right) - L\left(x^k, \lambda^k\right)$$
$$\ell_k^d(\Sigma_k) = L\left(x^k, \lambda^k\right) - L\left(x^k, \lambda^{k+1}(\Sigma_k)\right)$$

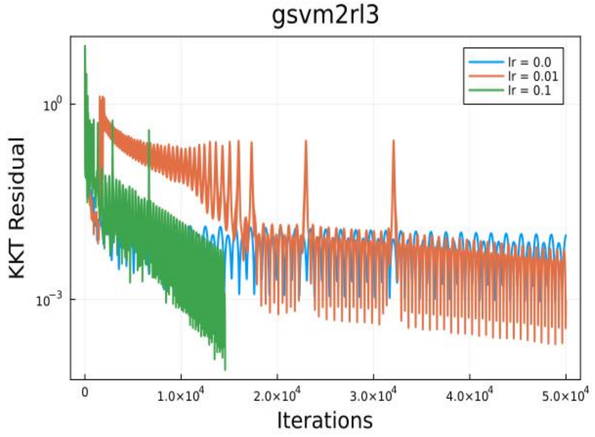
Preconditioned PDHG for LP:

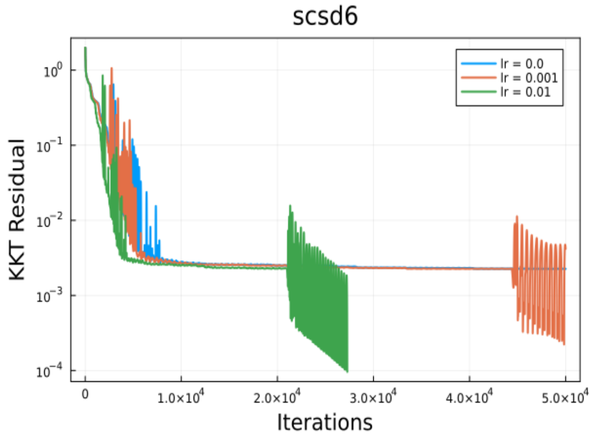
$$x^{k+1}(T_k) = \text{proj}_{\mathbb{R}^n_+}\left(x^k - T_k(c - A^T\lambda^k)\right)$$
$$\lambda^{k+1}(\Sigma_k) = \lambda^k + \Sigma_k(b - A(2x^{k+1} - x^k))$$

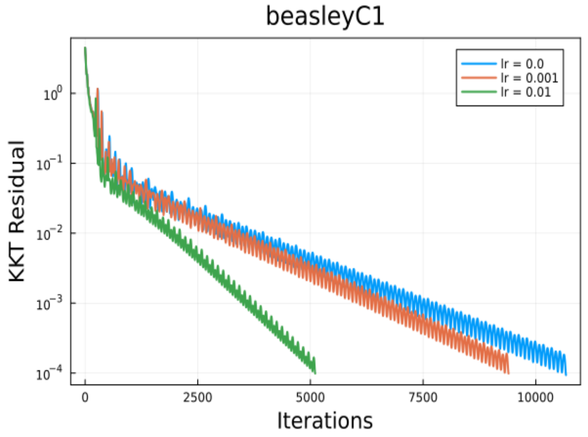
$$T_{k+1} = \Pi_P\left[T^k - \eta \nabla \ell_k^p(T_k)\right]$$
$$\Sigma_{k+1} = \Pi_D\left[\Sigma^k - \eta \nabla \ell_k^d(\Sigma_k)\right]$$

PDLP with Online Scaling Experiments







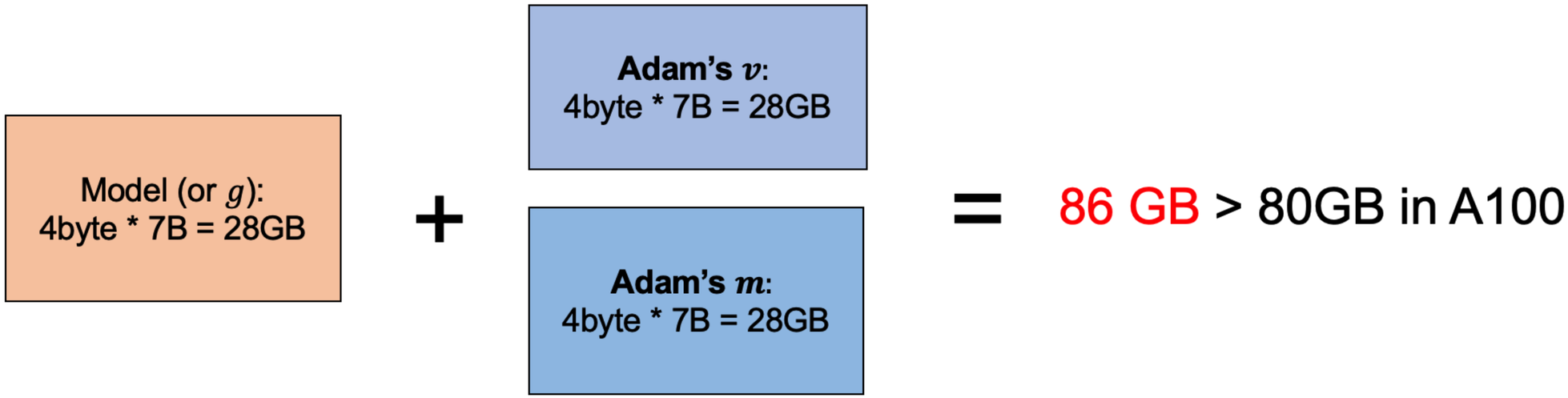


	MIPLIB		MIPLIB (large instances)	
	PDLP	+Online scaling	PDLP	+Online scaling
Iter (SGM10)	3693	3187	26120	20327
Iter (Mean)	25765	15985	66103	40195
Time (GM)	3.63	3.14	24.43	19.59
Time (Mean)	24.37	15.74	61.91	39.38

Scaling Decisions are Critical

- Adam needs memory for m, and v
  -- In total: 2x model size

  -- becomes a major overhead for LLMs: e.g., for 7B models



- 13B: Adam alone takes 80 A100 ($16M), 45 day to train
- To support training: CPU offload, model sharping… but cause latency!
- It is important to improve Adam!

Adam-mini: Sparse-Conditioning
(Zhang et al. https://arxiv.org/pdf/2406.16793 2024)

- **The paper proposes a simple and cheap way to find the "fewer but better" lrs:**

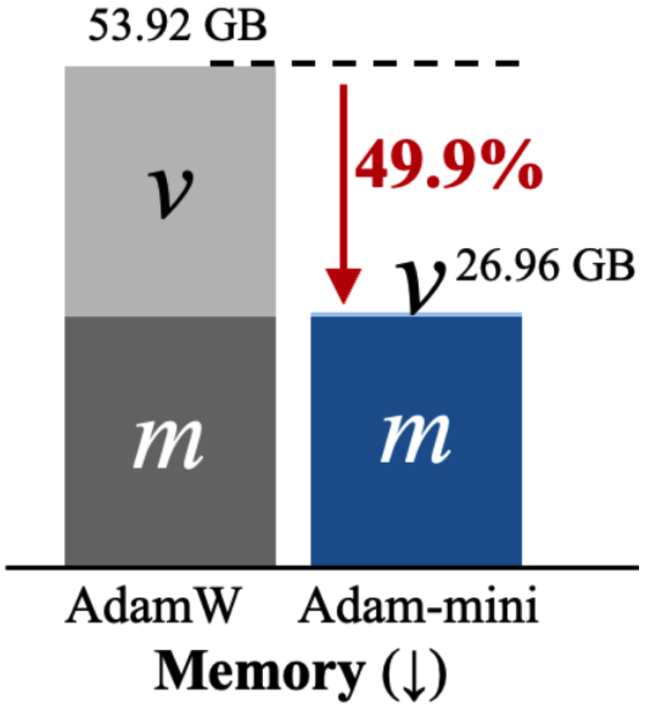
  - Step 1: partition the gradient g into B sub-vectors according to the dense Hessian sub-block $g_b, b = [B]$

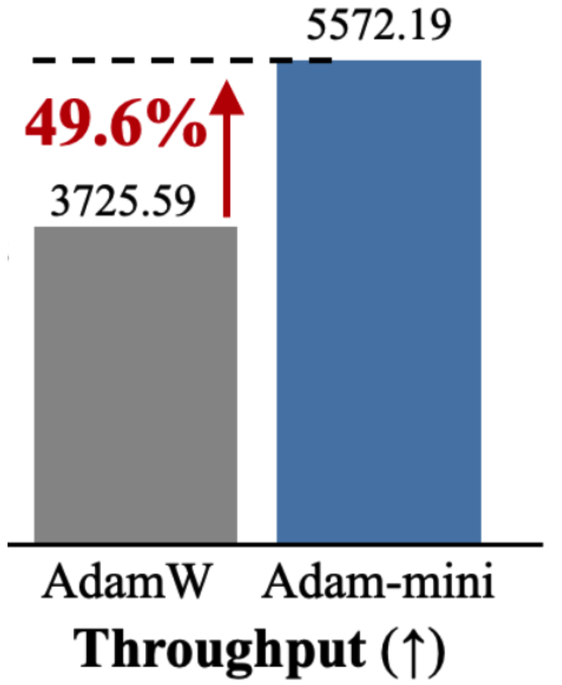
  - Step 2: for each g_b, calculate:

$$v_b = (1-\beta_2) * \mathbf{mean}(g_b \circ g_b) + \beta_2 * v_b, \quad b = 1, \cdots B$$

  - Step 3: then use $\frac{\eta}{\sqrt{v_b}}$ as the lr for all the parameters associated with g_b

Memory Cut-Down & Throughput Enhancement





Saves 50% memory of Adam

Can increase about 50% throughput of Adam (# processed data per second)

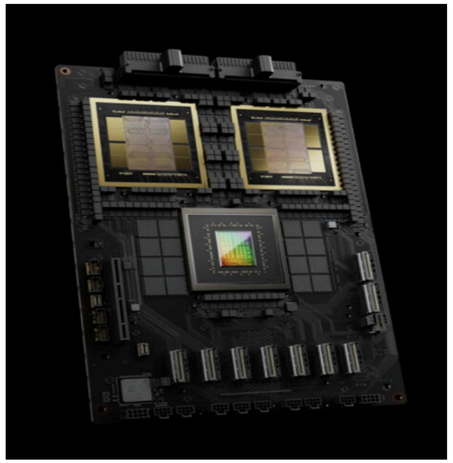
Why? Reduce communication + larger batch size per GPU

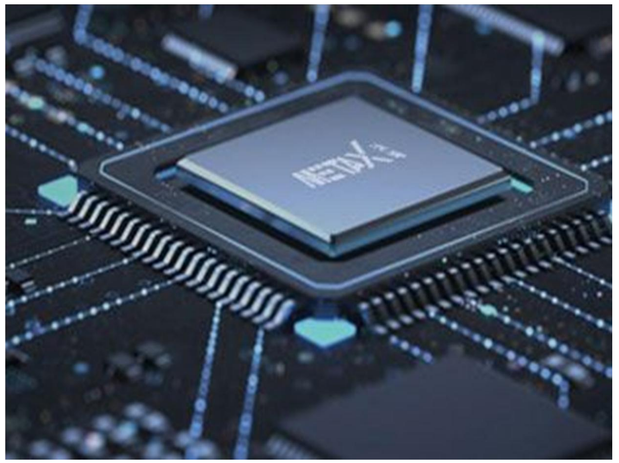
Today’s Talk

- **Online Linear Programming: Learning and Decision Making in Real Time**

- **Online Hyper-Gradient Method: Theory and Practice**

- **Optimization Solvers on GPU: Preliminary Results**





Primal-Dual Hybrid Gradient for Linear Programming

- **cuPDLP uses the saddle-point formulation of LP**

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & c^\top x \\ \text{s.t.} \quad & Gx \geq h \\ & Ax = b \\ & l \leq x \leq u \ , \end{aligned}$$

$$\min_{x\in X}\max_{y\in Y} \ L(x,y) := c^\top x - y^\top K x + q^\top y \ ,$$

$$\begin{cases} x^{t+1} \leftarrow \mathrm{proj}_X(x^t - \tau(c - K^\top y^t)) \\ y^{t+1} \leftarrow \mathrm{proj}_Y(y^t + \sigma(q - K(2x^{t+1} - x^t))) \end{cases} ,$$

An Iteration of PDHG [Esser at al. 2010]:

- Computing $Kx, K^T y$ by sparse matrix-vector product (spmv)
- Choosing step sizes: τ, σ
- PDLP Adaptive line-search and restart: Applegate et al. (21,23), Lu/Yang (23)
- All operations can be done on GPU!

CPU+GPU Heterogeneous Computing: Breakthrough for LP

杉数科技 Cardinal Operations

Can we develop new algorithms based on GPU to increase the scale and speed of solvable problems by a hundredfold or even ten thousandfold?

Type	Problem	Number of Constraints	Number of Variables	Non-zero Coefficients	CPU Solver Time (s) COPT	GPU Solver Time (s) cuPDLP-C
European Logistics Problem	zib03	19,731,970	29,128,799	104,422,573	59400	916
Google Pagerank	rand_1m_nodes	1,000,001	1,000,000	7,999,982	-	3.56
	rand_10m_nodes	10,000,001	10,000,000	79,999,982	-	44.22
	com-livejournal	3,997,963	3,997,962	77,358,302	-	21.07
	soc-livejournal1	4,847,572	4,847,571	78,170,533	-	22.26
Multi-layer Supply Chain Network Optimization	inv-10	4,035,449	3,758,458	15,264,380	-	1636
	inv-20	8,368,795	7,810,584	31,718,673	-	1157
	inv-40	13,186,756	12,066,105	49,528,729	-	6032
	inv-60	16,227,780	14,544,689	60,372,404	-	11102
China Southern Power Grid Clearing and Dispatching (Precision 1e-4)	model A	Sensitive Data			1951.35	254.93
	model B				1635.65	141.01

cuPDLP-C: A Strengthened Implementation of cuPDLP for Linear Programming by C language,
H Lu, J Yang, H Hu, Q Huangfu, J Liu, T Liu, Y Ye, C Zhang, D Ge, arXiv preprint arXiv:2312.14832, 2023

Milestones of Solving a Well-Known “Intractable” Instance

In a workshop in January 2008 on the *Perspectives in Interior Point Methods for Solving Linear Programs*, the instance `zib03` with 29,128,799 columns, 19,731,970 rows and 104,422,573 non-zeros was made public. As it turned out, the simplex algorithm was not suitable to solve it and barrier methods needed at least about 256 GB of memory, which was not easily available at that time. The first to solve it was Christian Bliek in April 2009, running CPLEX out-of-core with eight threads and converging in 12,035,375 seconds (139 days) to solve the LP without crossover. Each iteration took 56 hours! Using modern codes on a machine with 2 TB memory and 4 E7-8880v4 CPUs @ 2.20 GHz with a total of 88 cores, this instance can be solved in 59,432 seconds = 16.5 hours with just 10% of the available memory used. This is a speed-up of 200 within 10 years. However, when the instance was introduced in 2008, none of the codes was able to solve it. Therefore there was infinite progress in the first year. Furthermore, 2021 was the first time we were able to compute an optimal *basis* solution.

[1]Koch, Thorsten, et al. "Progress in mathematical programming solvers from 2001 to 2020." EURO Journal on Computational Optimization 10 (2022): 100031.

2008: Instance zib03[1]
29,128,799 variables
19,731,970 constraints

2009: Cplex Barrier (without crossover)
139 days (56 hours/IPM-iteration)

2019: IPM on a more advanced machine
16.5 hours

2023-24: cuPDLP-C (to 1e-6 tolerance)
1.7 hours on NVIDIA A6000
27 minutes on NVIDIA H100!

Excluding hardware improvement, LP (COPT and others) speed becomes 3.5x faster on average in the past 4 years
Future Directions: tackle the mix-ILP and solvers on GPU

The Factor-Representation of Semidefinite Programming

Consider the standard SDP: $\langle A_i, X\rangle = b_i,\ i=1,\ldots,m$

$$\min_{X\in\mathbb{S}^n} \ \langle C, X\rangle \quad \text{s.t.} \quad \mathcal{A}(X) = b, \quad X \succeq 0,$$

$$\min_{U\in\mathbb{R}^{n\times r}} \ \langle C, UU^\top\rangle \quad \text{s.t.} \quad \mathcal{A}(UU^\top) = b.$$

The low-rank factorization helps save:

- **storage**
- **computation efforts**

Theorem: Let *n* be the dimension of the matrix *X* and *m* be the number of the equality constraints. Then, if there is an optimal solution, there must exist an solution with rank $\sqrt{2m}$.

A low-rank factorization method (Burer and Monteiro, 2003) uses the Augmented Lagrangian method to solve the (**nonconvex**) problem above.

The Splitting Technique and ADMM (LoRADS, Han et al. 2024)

$$\min_{U \in \mathbb{R}^{n \times r}} \ \langle C, UU^\top \rangle \ \ \text{s.t.} \ \ \mathcal{A}(UU^\top) = b.$$

We "split" the variable U into

$$UU^T \rightarrow UV^T \quad s.t. \quad U = V$$

$$\min_{U,V \in \mathbb{R}^{n \times r}} \ \langle C, UV^\top \rangle + \frac{\gamma}{2}\|U - V\|_F^2 \ \ \text{s.t.} \ \ \mathcal{A}(UV^\top) = b.$$

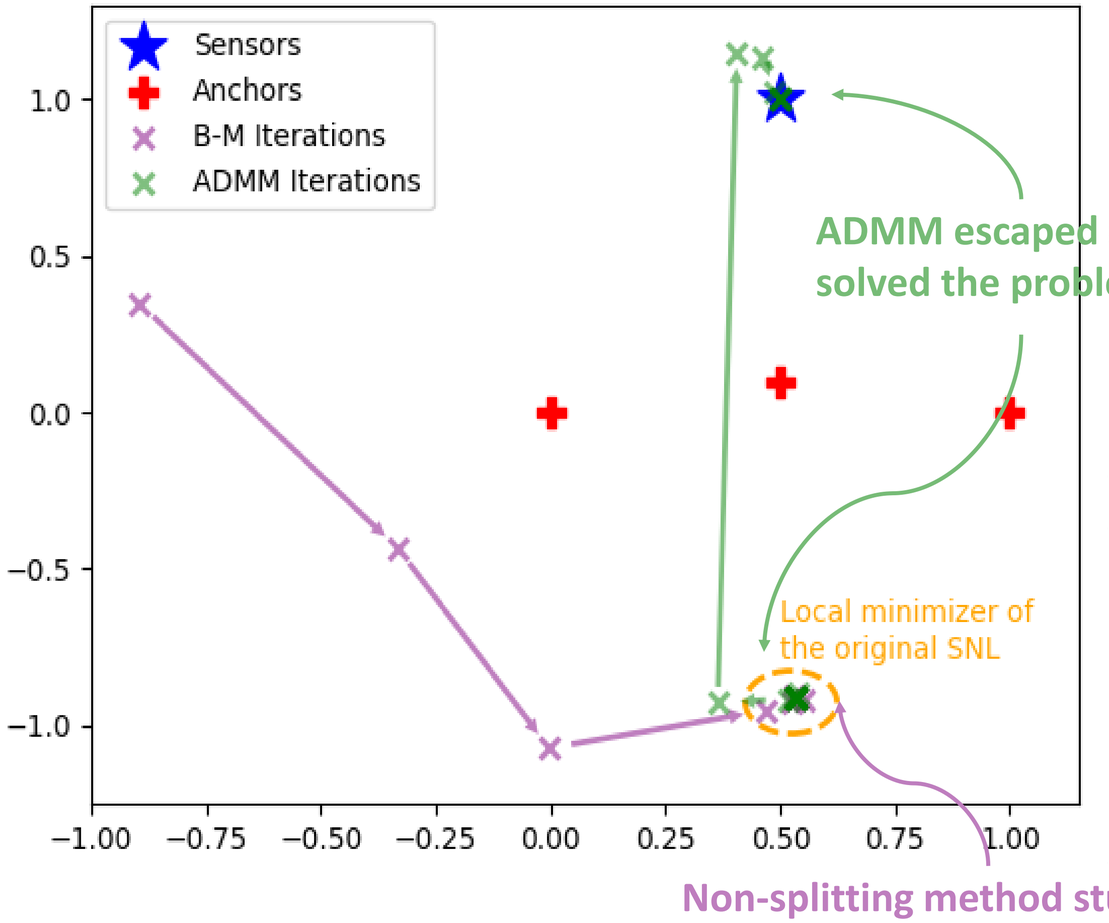
Then we apply the ADMM to solve the problem:

$$U^{k+1} \leftarrow \operatorname{argmin}_U \mathcal{L}_\rho(U, V^k, \lambda^k)$$
$$V^{k+1} \leftarrow \operatorname{argmin}_V \mathcal{L}_\rho(U^{k+1}, V, \lambda^k)$$
$$\lambda^{k+1} = \lambda^k + \rho\big(\mathcal{A}(U^{k+1}(V^{k+1})^\top) - b\big)$$

The Splitting Technique and ADMM

Benefits:

ADMM's "split search" may boost performance on difficult cases



The ADMM subproblem resulting in solving well conditioned linear system:

$$\left(\rho\sum_{i=1}^{m}\mathrm{vec}(A_iV^k)\mathrm{vec}(A_iV^k)^\top+\gamma I_{nr\times nr}\right)\mathrm{vec}(U^{k+1})$$
$$=-\mathrm{vec}\left(CV^k-\gamma V^k+\sum_{i=1}^{m}\lambda_i^kA_iV^k-\rho\sum_{i=1}^{m}b_iA_iV^k\right),$$

Typically, **warm-started CG** takes about $\mathbf{10^{-5}n}$ iterations for a $\mathbf{n}$-dimensional linear system:

Problem	$nr=n\sqrt{2m}$	ADMM Iterations	Total CG Iters	Avg CG Iters
G60	828251	100	845	8.45
MC_3000	4092176	17	9626	566.24
G40_mb	126586	104	2252	21.65
p_auss2_3.0	1230829	1656	11152	6.73
qap7	1338	1829	91128	49.82
qap10	4564	29452	226521	7.69
theta12	255011	1261	6938	5.50

Combining ALM and ADMM

Using (non-splitting) ALM method for warm-start (splitting) ADMM

Problem	warm-started LRADMM	LRADMM	LRALM (SDPLR)
G60	3.47	6.61	7.66
MC_3000	154	t	819
G40_mb	29.76	32.62	59.95
H3O	19.37	f	199
p_auss2_3.0	56.85	158	83.95
qap7	1.41	2.14	2.18
qap10	6.47	21.51	10.19
theta12	62.26	22.76	127

➢**The combined approach is both stable and fast**

Logarithmic Rank Selection: O(log(m)) rank

Typically, the factor representation method uses $\sqrt{2m}$ rank based on theoretical results. But it is too large in practice, can we do better?

The existence of O(log(m)) low-rank approx. solution!

Can be relaxed

Theorem 1.1 of (So et al. 2008) Let $A_1, \ldots, A_m \in \mathbb{R}^{n \times n}$ be symmetric positive semidefinite matrices, and let $b_1, \ldots, b_m \geq 0$. Suppose that there exists an $X \succeq 0$ such that $\langle A_i, X \rangle = b_i$ for $i = 1, 2, \ldots, m$. Let $r = \min\{\sqrt{2m}, n\}$. Then, for any $d \geq 1$, there exists an $X_0 \succeq 0$ with $\operatorname{rank}(X_0) \leq d$ such that:

$$\beta(m,n,d) \cdot b_i \leq \langle A_i, X_0 \rangle \leq \alpha(m,n,d) \cdot b_i \quad \text{for } i = 1, \ldots, m$$

where:

$$\alpha(m,n,d) = \begin{cases} 1 + \frac{12\ln(4mr)}{d} & \text{for } 1 \leq d \leq 12\ln(4mr) \\ 1 + \sqrt{\frac{12\ln(4mr)}{d}} & \text{for } d > 12\ln(4mr) \end{cases}$$

and

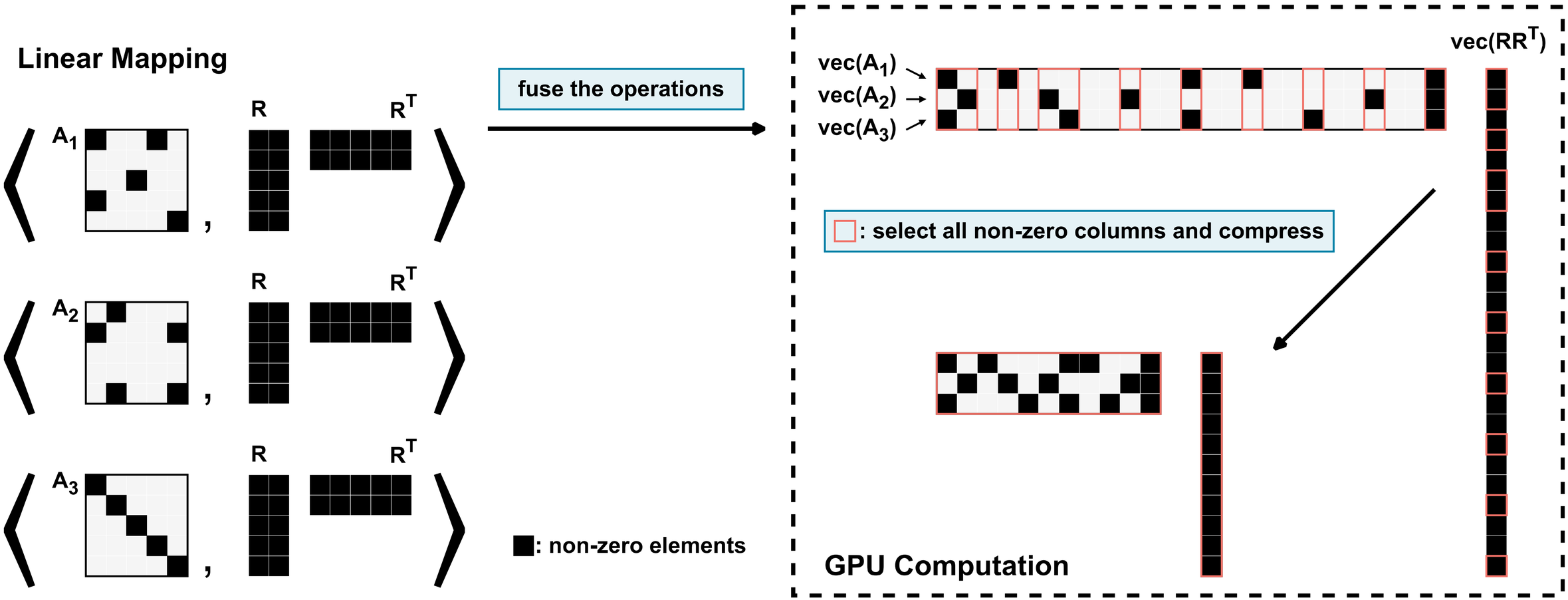
$$\beta(m,n,d) = \begin{cases} \frac{1}{e(2m)^{2/d}} & \text{for } 1 \leq d \leq 4\ln(2m) \\ \max\left\{ \frac{1}{e(2m)^{2/d}}, 1 - \sqrt{\frac{4\ln(2m)}{d}} \right\} & \text{for } d > 4\ln(2m) \end{cases}$$

Moreover, there exists an efficient randomized algorithm for finding such an X_0.

Move SDP Solving onto GPUs (cuLoRADS)

Efficient computational design for key operations on GPUs:

- **aligning with thread-level parallelism of GPUs and achieving memory efficiency**



The Quantum Ordered Search Problem: Computational Bottleneck

- The Ordered Search Problem (OSP) is a fundamental computational task: find a specific item in a sorted list of largest possible N elements with k queries, and it can be tackled by LP and SDP solutions

- The size of the SDP program scales **quadratically** with the list size N, and the maximum list size N for a given number of queries k grows **exponentially** with k.

- And the non-zeros of the problem comes to $O(N^2k^2)$

- For the 5-query case, the SDP was considered "out of reach of SDP solvers" for **over 18 years**.

- However, using the LP relaxation, a **5-query** algorithm with N=7265 was found that established the current best upper bound of $5log_{7265}N \approx 0.390log_2N$ (Carolan et al., 2025).

The Recent Breakthrough: Our GPU-based solver directly solved the 5-query case via the original SDP approach!

Certificates:

- **Feasibility for k=5, N=7265:**
  - **The SDP problem was solved to an absolute tolerance of 10^{-8} in about 21 hours.**

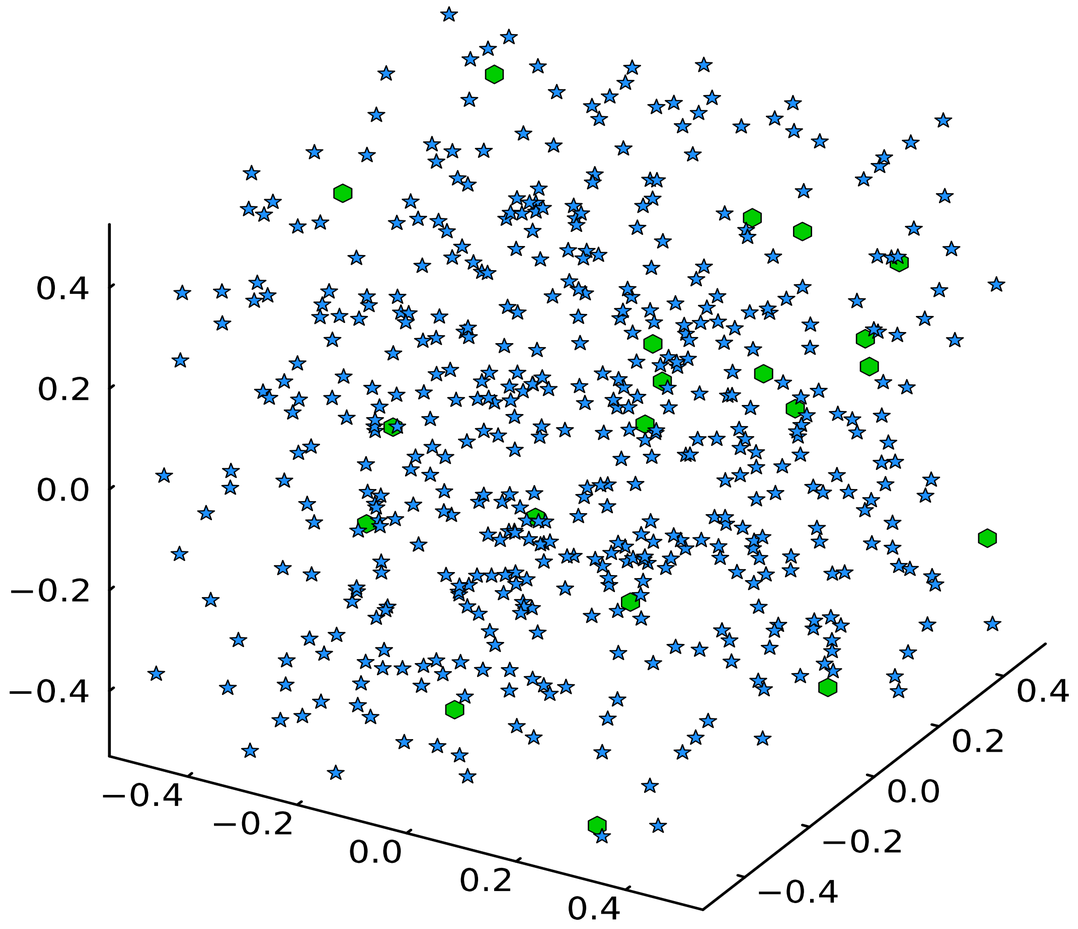
- **Infeasibility for k=5, N=7266:**
  - **Our solver produced a feasible dual solution with a positive dual objective value (1.92×10^{3}).**

This reproves that N=7265 is the maximal list size for the 5-query problem and gives hope for k=6…

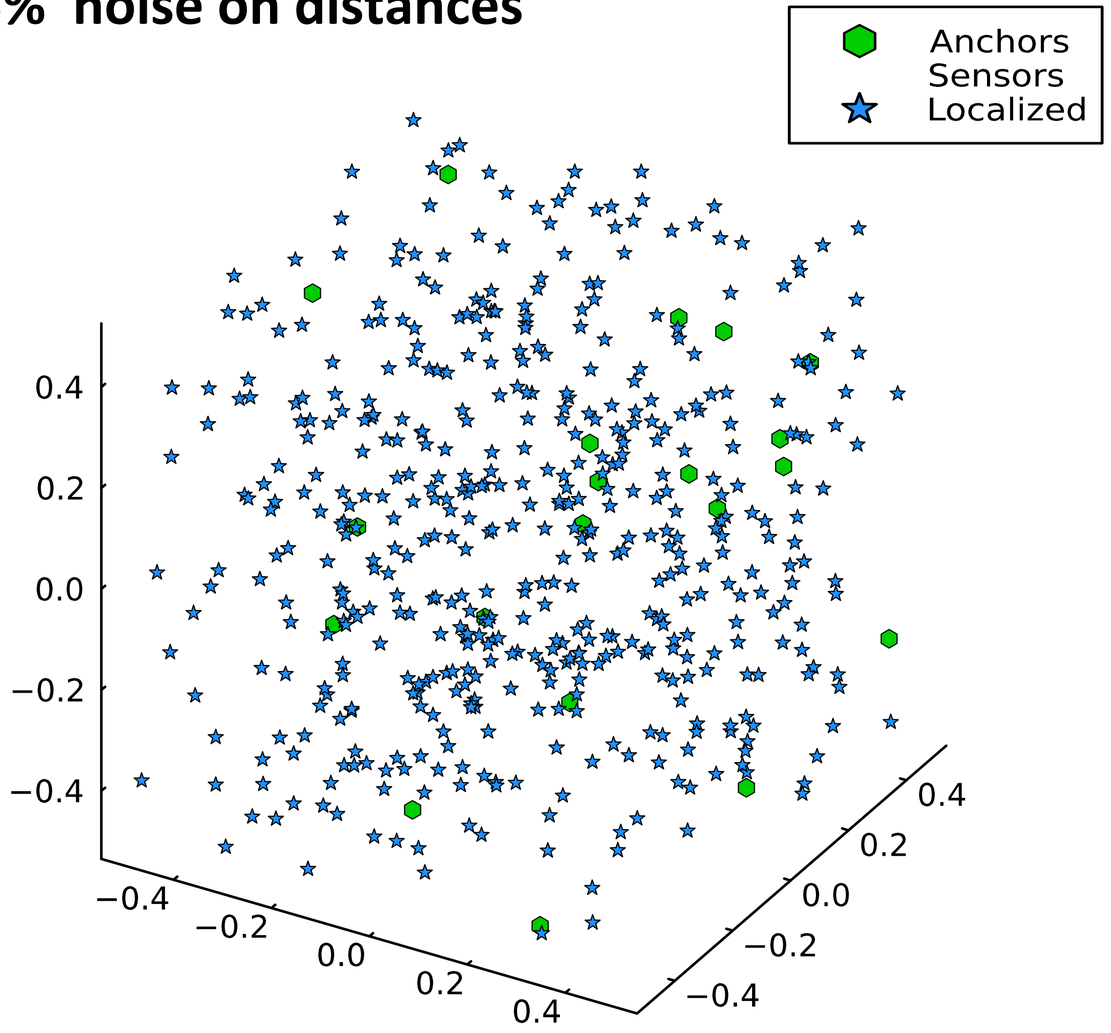
Computational Results on Sensor Network Localization

A system of quadratic equations wit performance of cuLoRADS: localize 10^4 points in seconds

0% noise on distances



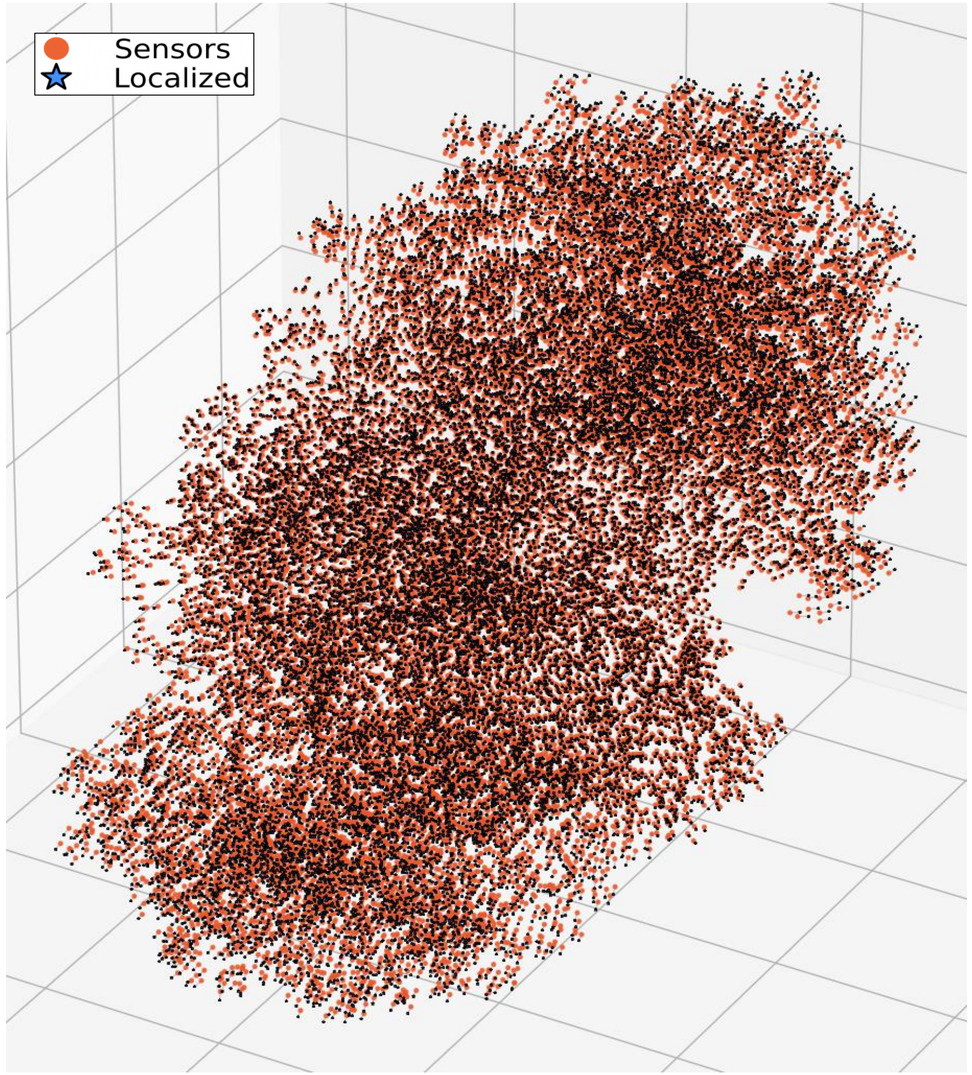
5% noise on distances

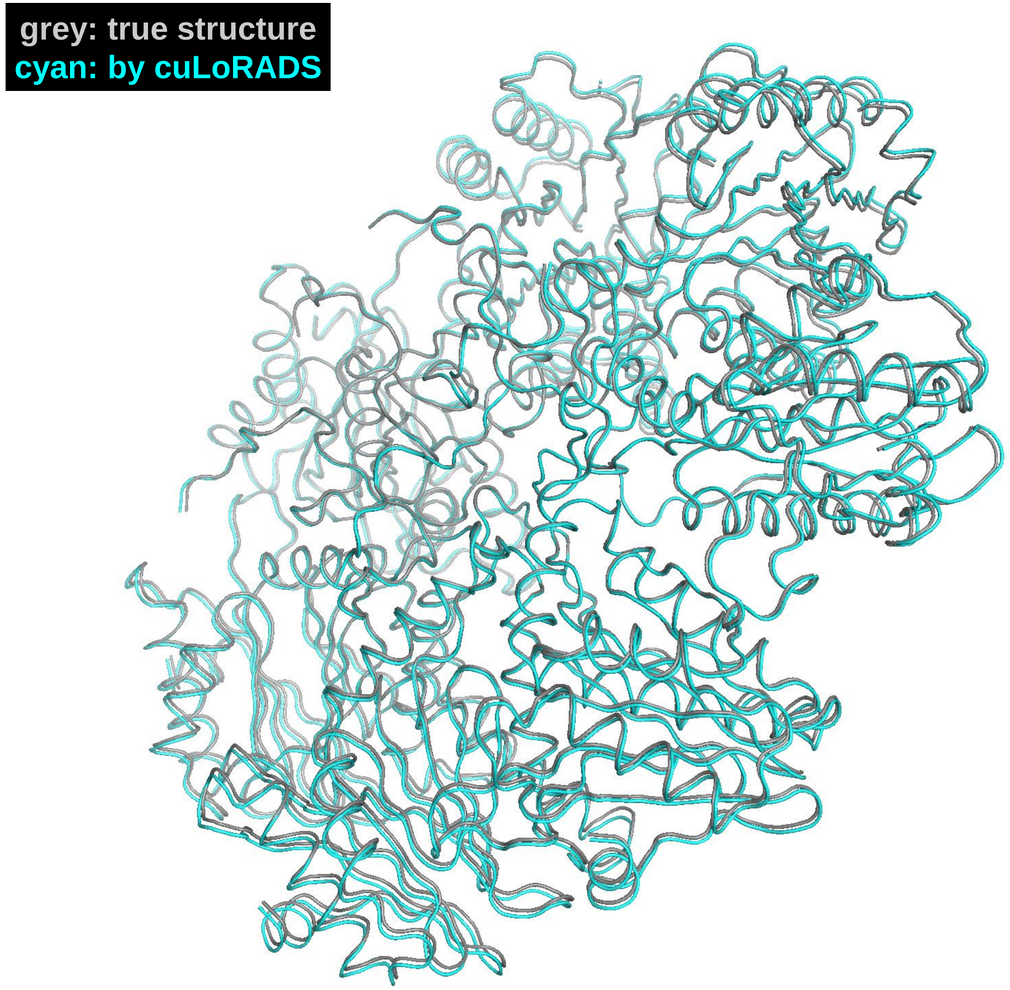


Computational Results on Molecular Confirmation

[2KU1], 25935 atoms, ~1.4 million constraints; 70% distances under 6Å with 5% noise

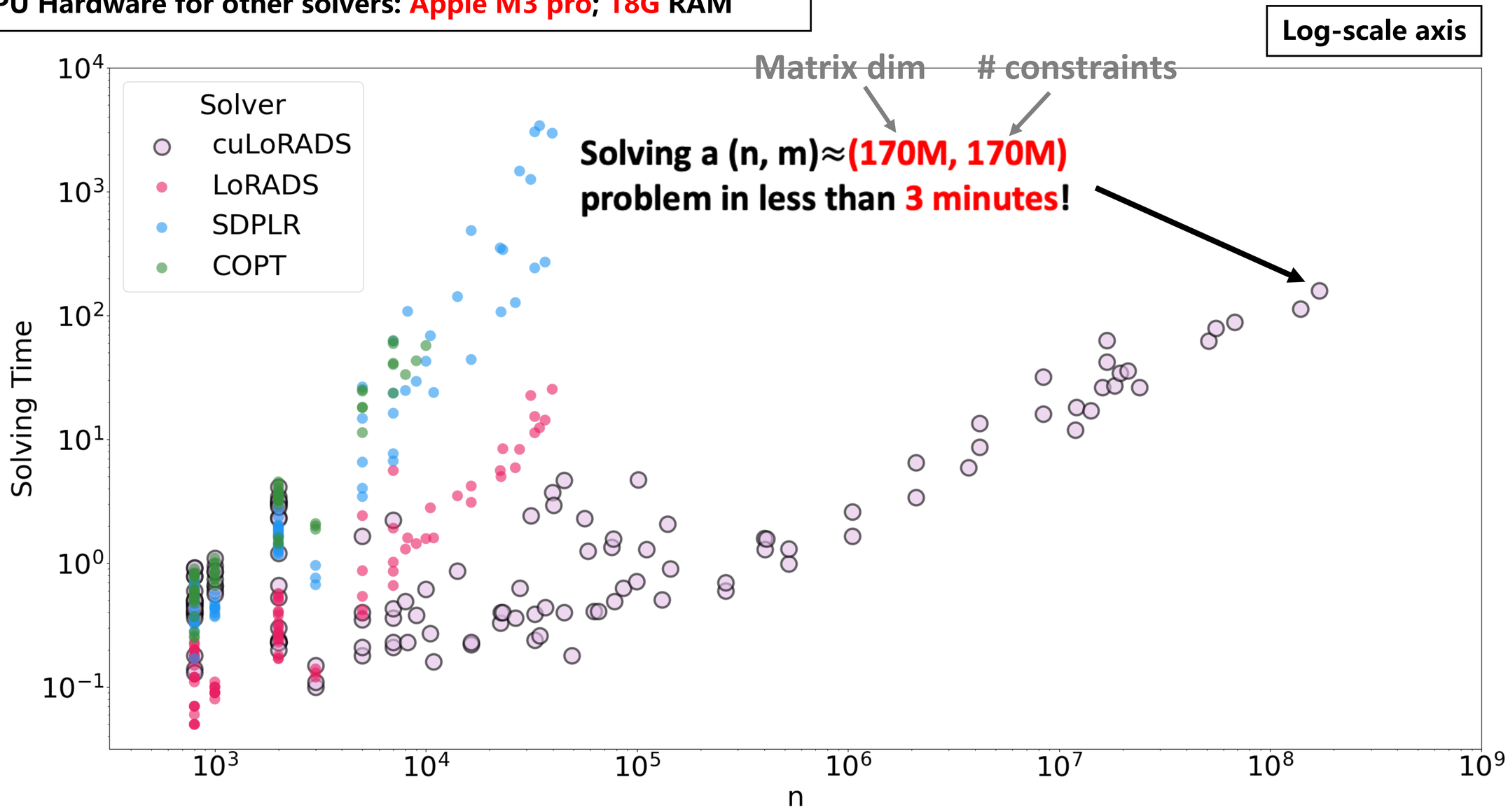
RMSD = 0.5947Å (atomic accuracy!), localized in 30 seconds





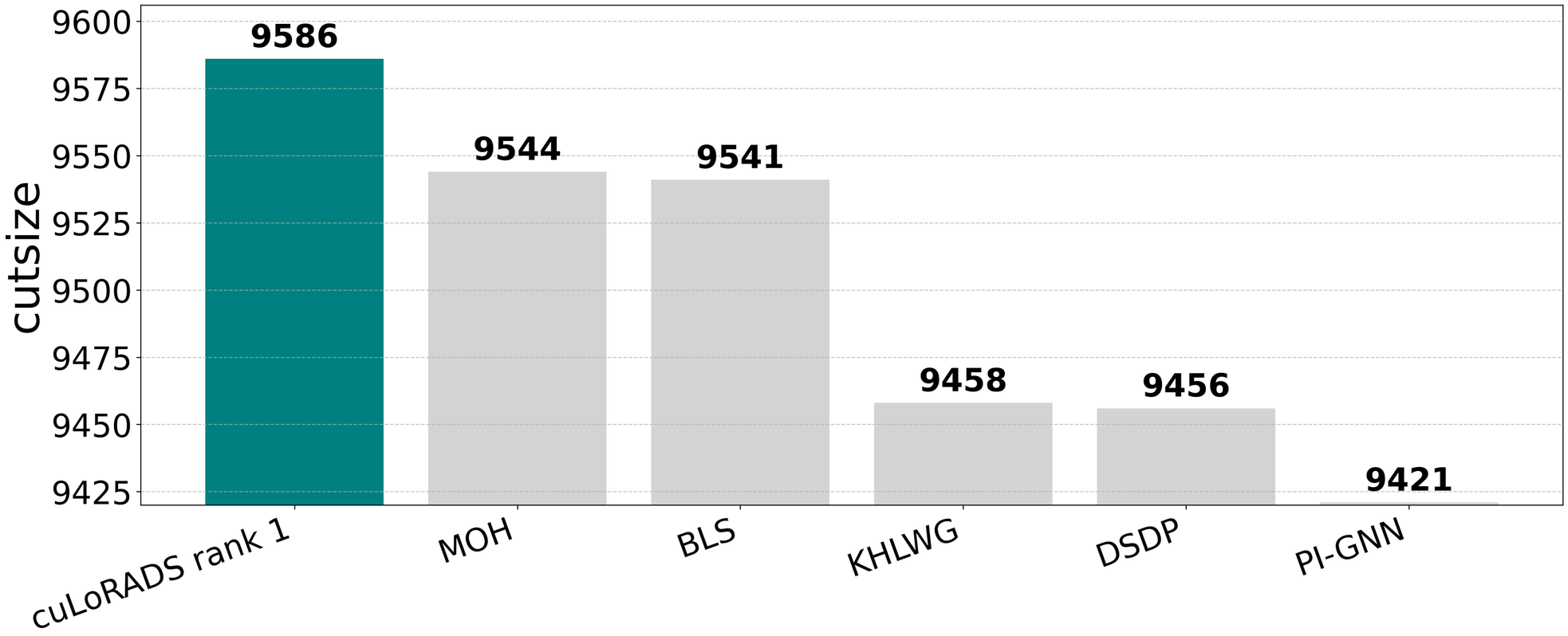
Computational Results on Max-Cut SDP Problems

GPU Hardware for cuLoRADS.jl: Nvidia H100; 80G VRAM
CPU Hardware for other solvers: Apple M3 pro; 18G RAM



Computational Results on the G70 Max-Cut Problem

- **Obtain a rank-1 solution from our SDP solver: comparing with existing state-of-the-art methods on Gset on G70 (10000 notes, 9999 edges**



- **Compared methods including learning methods (PI-GNN), heuristic Algorithms (BLS, MOH, KHLWG), and traditional SDP solvers (DSDP).**

Milestones of Solving SDP Instances

Solvers based on interior-point method:

Benchmark problem: Max Cut of size (n, m)

- **[1997]** SDPA (Fujisawa et al.),
  - a (1250, 1250) in 31 hours
- **[1998]** DSDP (Benson, Ye and Zhan),
  - a (1000, 1000) in ~ 5 minutes
- **[2008]** DSDP5 (Benson and Ye),
  - a (2000, 2000) in < 4 minutes
- **[2010]** According to Yamashita et al., 2010, a (5000, 5000)
  - SDPA 7.3.1 in ~ 10 minutes
  - CSDP 6.0.1 in ~ 10 minutes
  - SDPT3 4.0β in ~ 20 minutes
  - SeDuMi 1.21 > 1 hour
- **[2022]** HDSDP (Gao, Ge & Ye),
  - a (5000, 5000) in < 1 minutes
- **[Recent years]** Modern SOTA solvers,
  - can solver ~ $(10^4, 10^4)$ in minutes

Solvers based on First-Order type methods:

- **[2001]** The Burer-Monteiro low-rank method (Burer and Monteiro, 2003),
  - a (20000, 20000) Max Cut in ~ 8 minutes
- **[2021]** SketchyCGAL (Yurtsever et al., 2021),
  - ~ $(10^7, 10^7)$ Max Cuts in ~ 30 - 50 hours
- **[Mar 2024]** HALLaR (Monteiro et al., 2024),
  - a $(2 \times 10^5, 2 \times 10^7)$ Matrix Completion in 7.8 hours

[June 2024] cuLoRADS (Han et al.):

- Same ~ $(10^7, 10^7)$ Max Cuts in (Yurtsever et al., 2021) solved to high accuracy
  - in 10 seconds – 1 minutes (vs. 30 - 50 hours)
- A ~ $(4 \times 10^5, 1.6 \times 10^7)$ Matrix Completion similar to (Monteiro et al., 2024) solved to high accuracy
  - in ~ 2 seconds (vs. 7.8 hours)

SDP (COPT) speed is 2.5x faster on average in the past 3 years on a same machine

Long Live
Math Optimization

MO Pioneers (75Nobel)