Beating Classical Streaming Barriers with Noisy Predictions

Ali Vakilian

VIRGINIA TECH

Streaming Model

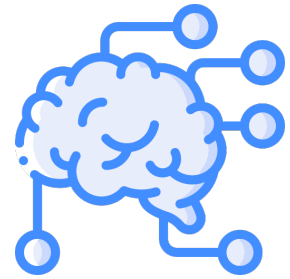
Available memory is much smaller than the size of the input stream

- **Input:** a massively long data stream $\sigma = \langle a_1, a_2, \dots, a_N \rangle$
  - I) **Sublinear storage**: N^{α} (for $\alpha < 1$) or $\log^c N$
  - II) **Small number of passes:** (ideally one pass) over the stream

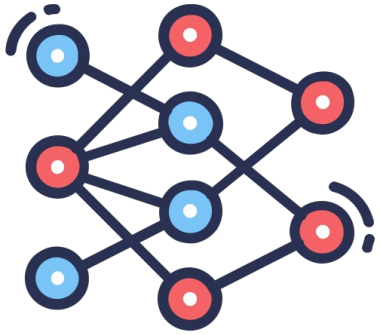
- **Goal**: compute $f(a_1, \dots, a_N)$ for a given function

Streaming Model (with predictions)

- What if we can obtain extra information about items in stream?
- Learning-Augmented Algorithms



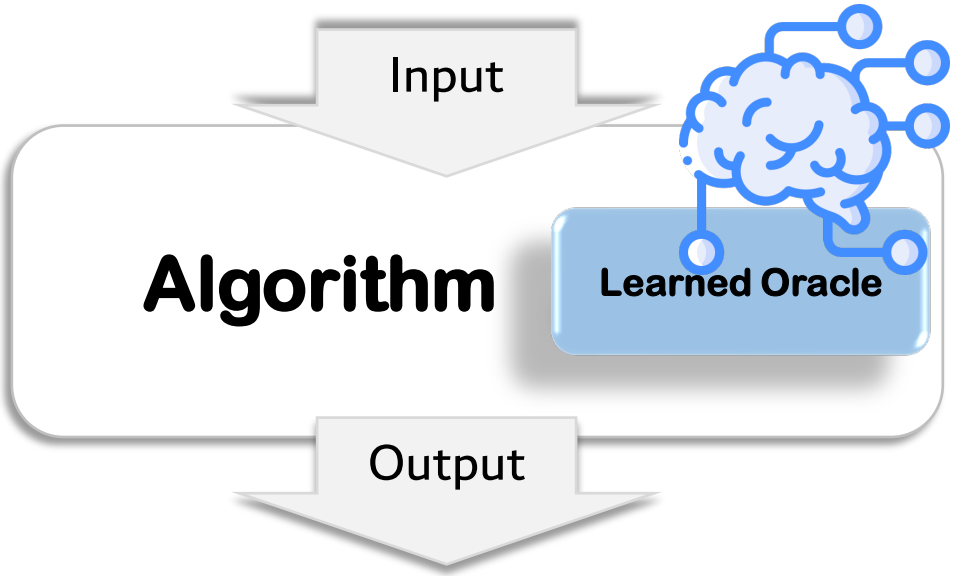
Streaming Algorithm



Learning-Augmented Algorithms

(aka **Algorithms with Predictions**)

I) **Better performance** when the input has some "learnable" pattern

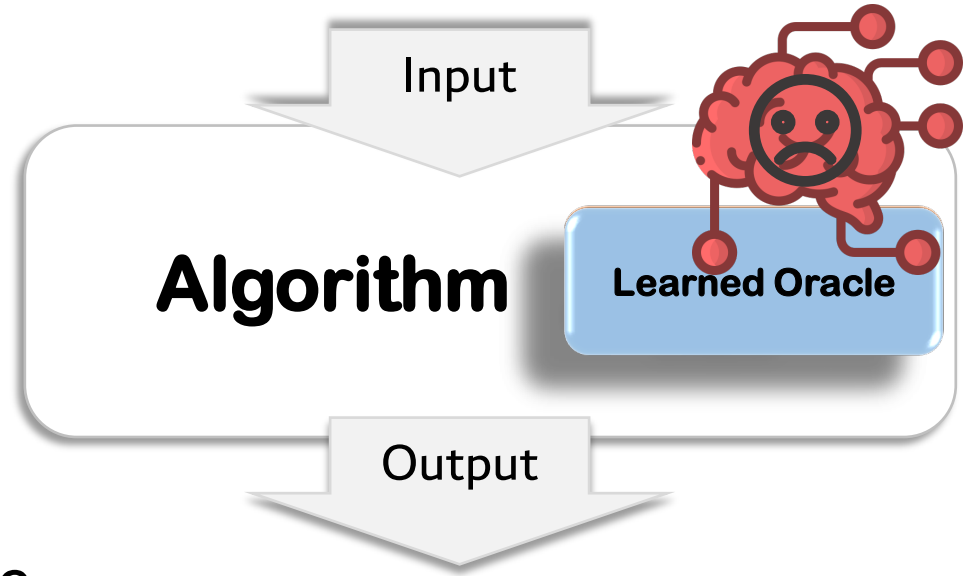


The algorithm has access to a **learned ORACLE** providing a certain type of **PREDICTIONS** about the instance in hand

Learning-Augmented Algorithms

(aka **Algorithms with Predictions**)

I) **Better performance** when the input has some "learnable" pattern



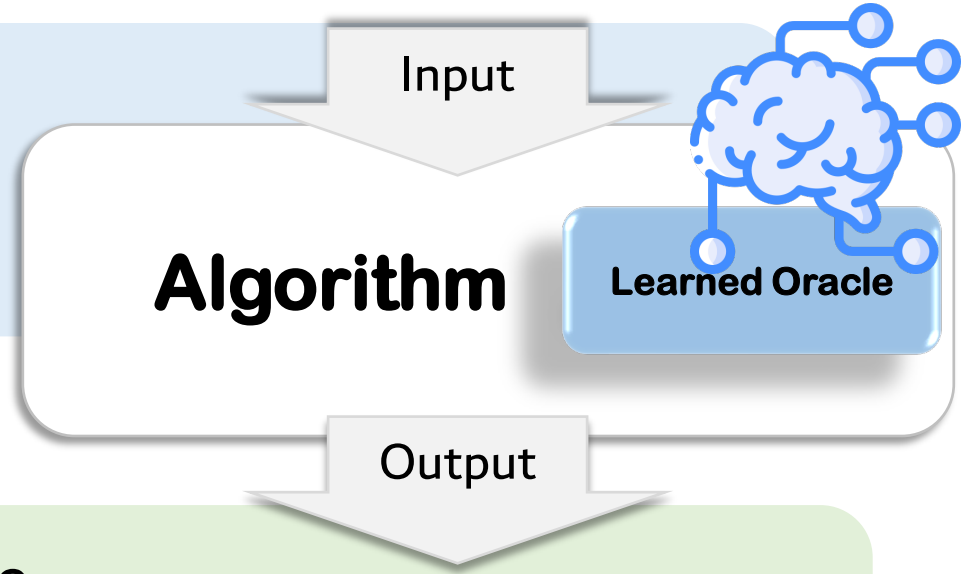
II) **Similar worst-case guarantee** as the best-known classical algorithms when oracle performs poorly

Learning-Augmented Algorithms

(aka **Algorithms with Predictions**)

I) **Better performance** when the input has some "learnable" pattern

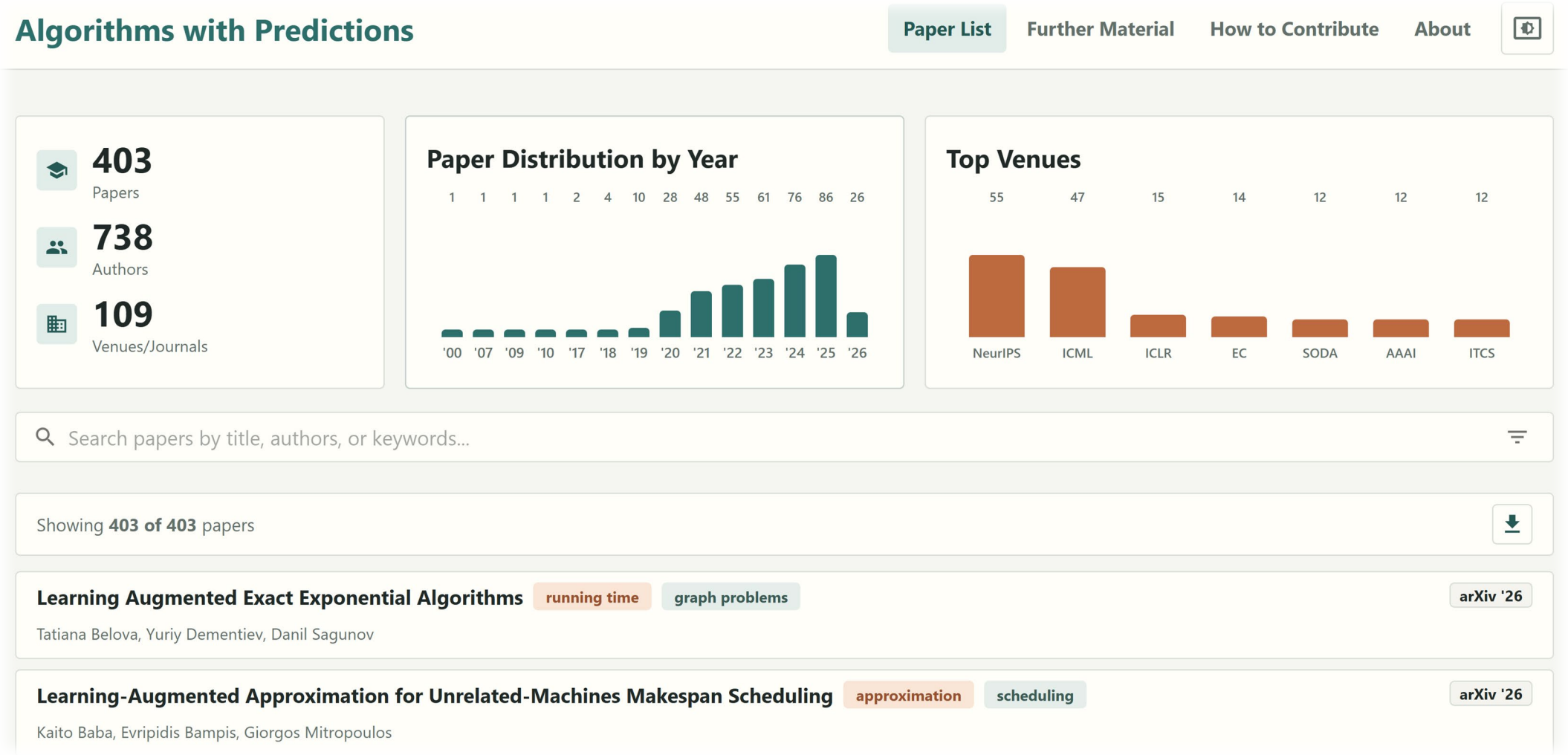
Consistency



II) **Similar worst-case guarantee** as the best-known classical algorithms when oracle performs poorly

Robustness

Useful Resource https://algorithms-with-predictions.github.io/



Outline of this talk

- **Streaming Algorithms with Learned Oracles**
  - Frequency-Vector Problems
- **Streaming Algorithms with Learned Structures**
  - Randomized Numerical Linear Algebra
- **Streaming Algorithms with ε-Accurate Predictions**
  - Graph Problems

Learned Oracles

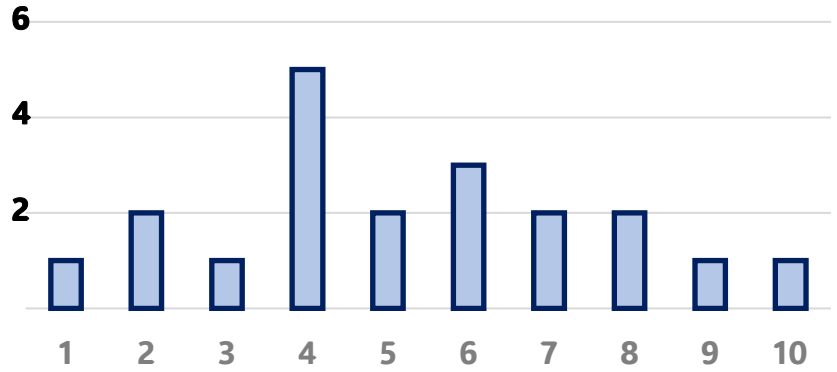
Heavy Hitter Oracle [HIK**V.**, ICLR19; ACNS**V.**, NeurIPS23]

Frequency Estimation Problem

INPUT: a stream of items that arrive one by one

Frequency Estimation Problem

8, 1, 7, 4, 6, 4, 10, 4, 4, 6, 8, 7, 5, 4, 2, 5, 6, 3, 9, 2



INPUT: a stream of items that arrive one by one

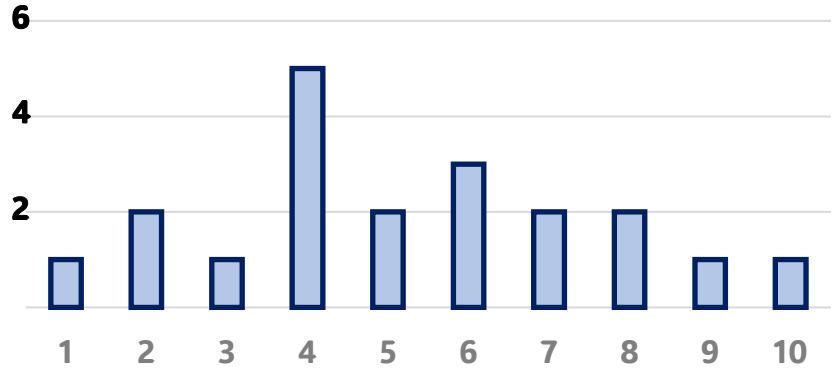
GOAL: compute the frequency of all items approximately

I) In **one pass** over the stream, and

II) Using **sublinear space** in the size of the stream (e.g., $\mathrm{poly}(\log n)$)

Frequency Estimation Problem

8, 1, 7, 4, 6, 4, 10, 4, 4, 6, 8, 7, 5, 4, 2, 5, 6, 3, 9, 2



INPUT: a stream of items that arrive one by one

GOAL: compute the frequency of all items approximately

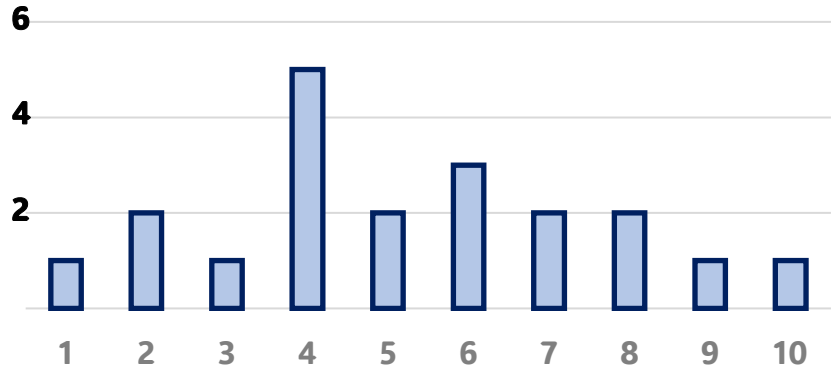
I) In **one pass** over the stream, and

II) Using **sublinear space** in the size of the stream (e.g., $\text{poly}(\log n)$)

- ❑ Fundamental subroutine in **Data Analysis**
- ❑ Applications in domains such as **NLP, Networking,** and **Database Optimization**

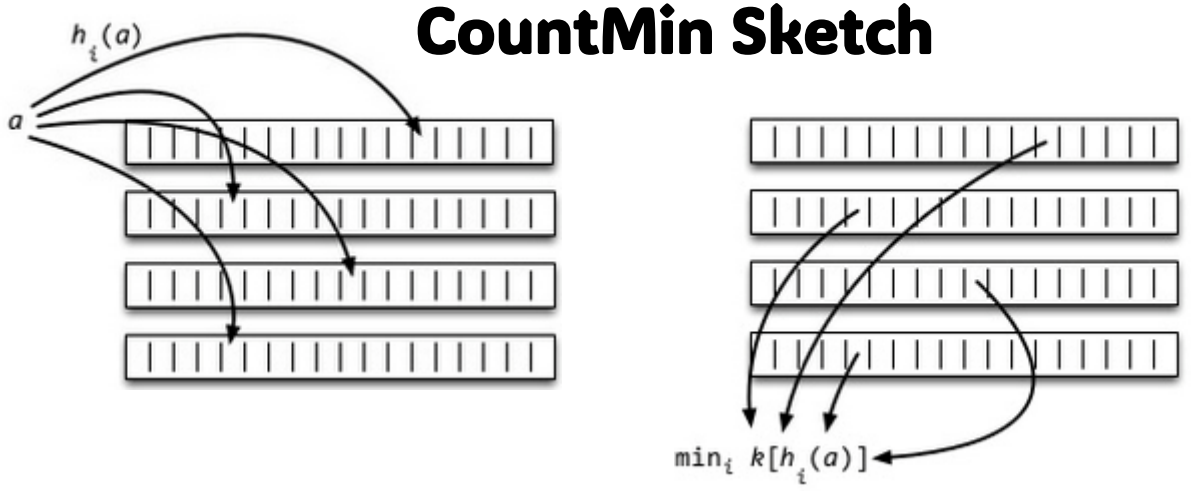
Frequency Estimation Problem: Hashing Techniques

8, 1, 7, 4, 6, 4, 10, 4, 4, 6, 8, 7, 5, 4, 2, 5, 6, 3, 9, 2



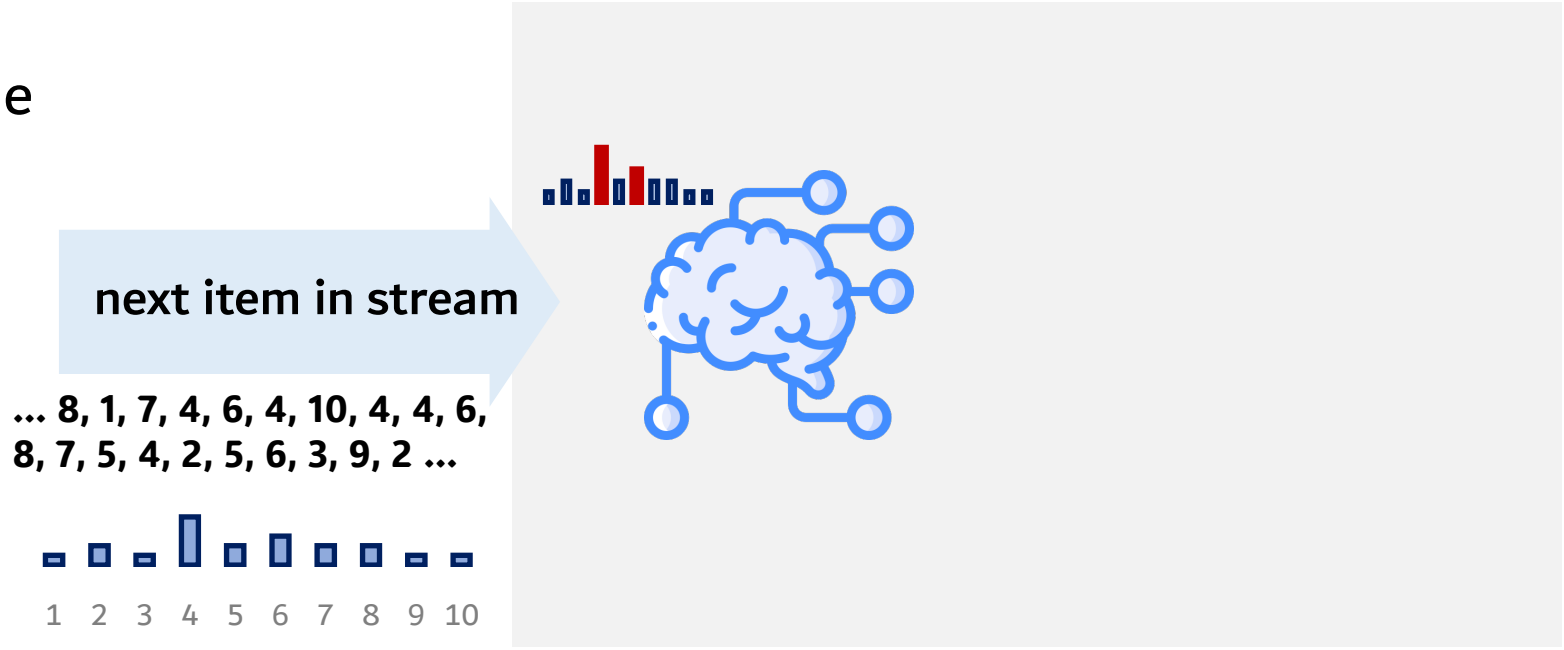
- CountMin
- CountSketch

CountMin Sketch



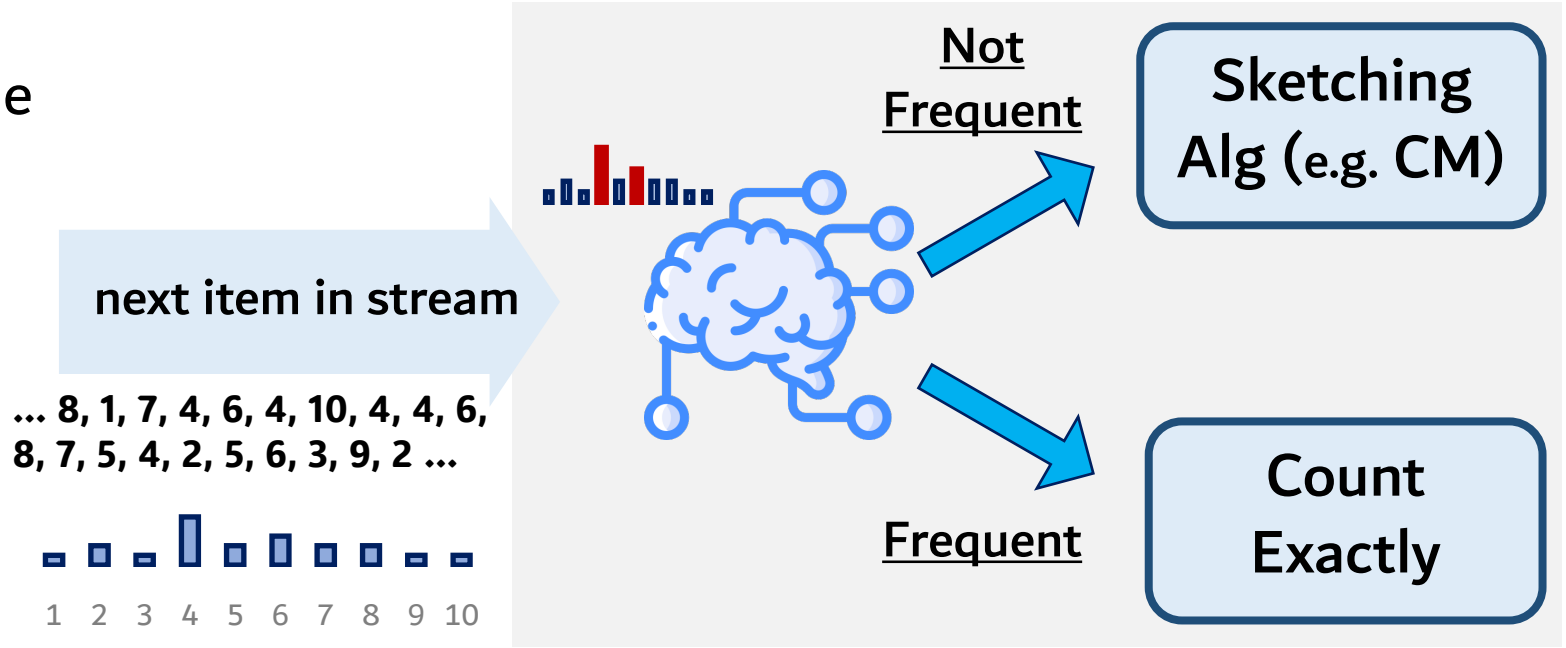
LEARNING-AUGMENTED Frequency Estimation

- **Learned Oracle:** Train an oracle to detect "**frequent**" elements (i.e., heavy hitters)



LEARNING-AUGMENTED Frequency Estimation

- **Learned Oracle:** Train an oracle to detect "**frequent**" elements (i.e., heavy hitters)
- Treat frequent items differently

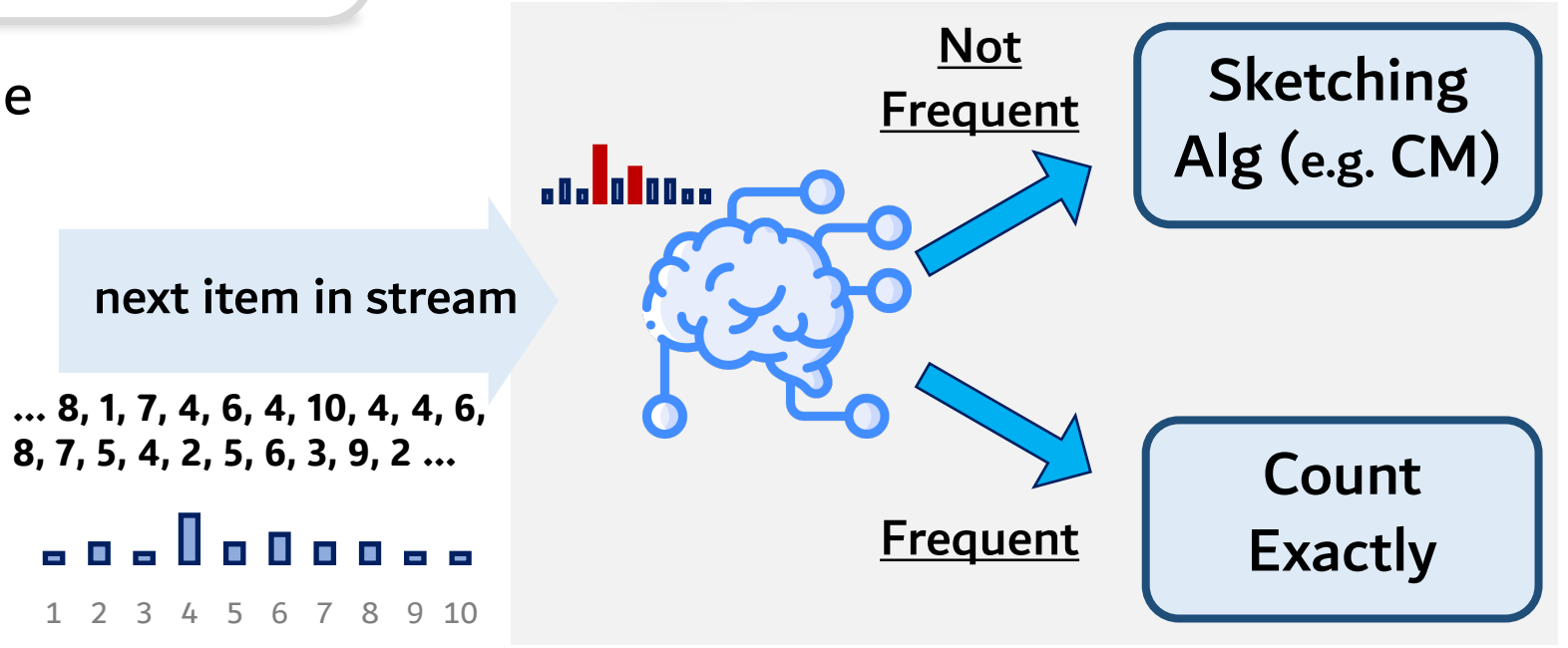


LEARNING-AUGMENTED Frequency Estimation

Why is this "different treatment" of items effective?

Collision with frequent items is **very costly**!

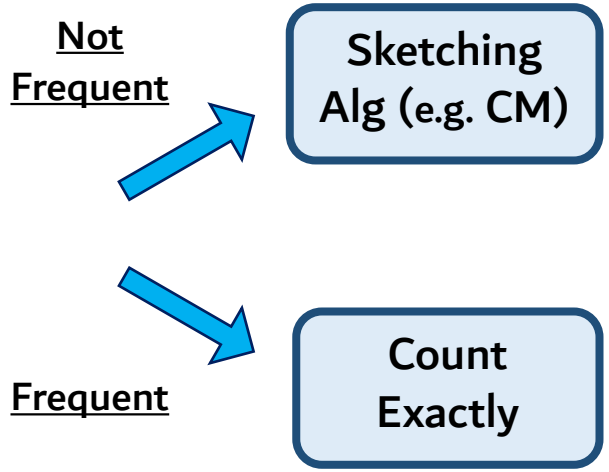
- **Learned Oracle:** Train an oracle to detect "**frequent**" elements (i.e., heavy hitters)



- Treat frequent items differently

Theoretical Results

[HIK**V**, **ICLR19**] & [ACNS**V**, **NeurIPS23**]



Zipfian Distribution ($f_i \propto 1/i$)

Method	Expected Err
CountMin (k rows)	$\Theta\left(\frac{k \cdot \ln n \cdot \ln(kn/B)}{B}\right)$
Learned CountMin	$\Theta\left(\frac{\ln^2(n/B)}{B}\right)$
Learned Sketch	$\Theta\left(\frac{1}{B}\right)$

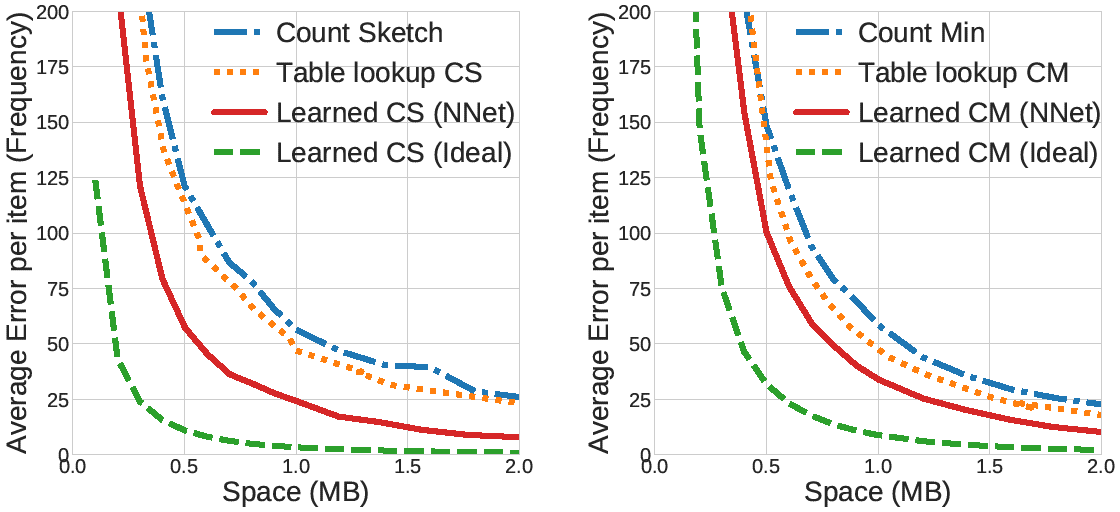
- ✓ Even if the oracle predicts poorly, asymptotically the same as CM
- ✓ Empirically, Learned CM reduces the error by $\sim 50\%$

n: number of items with non-zero frequency

B: amount of available space in words

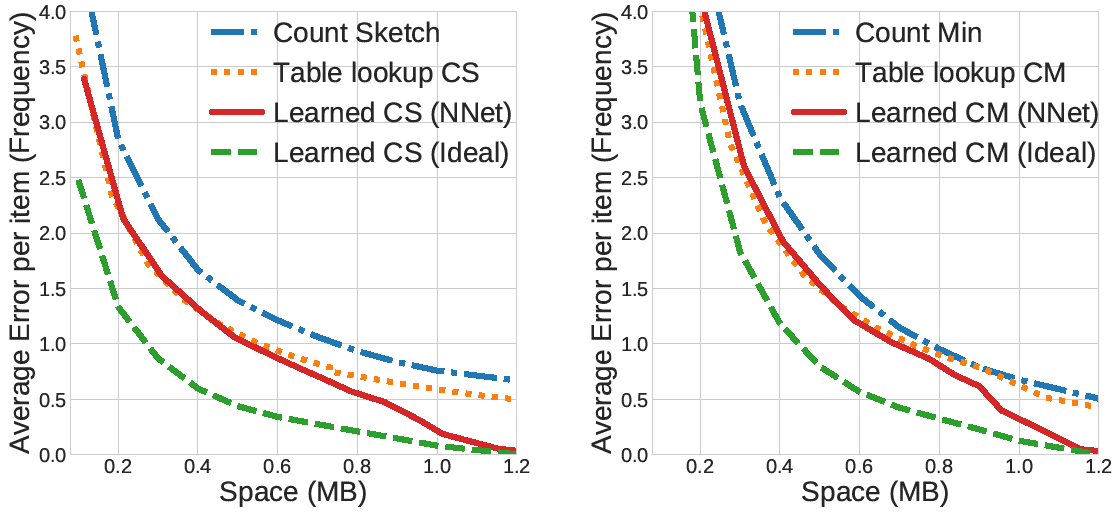
Empirical Evaluations

Network Traffic (CAIDA dataset)



HeavyHitter Oracle: RNN with **64** hidden units
+ two-layer NNs with total of **56** hidden units

Search Query (AOL dataset)



HeavyHitter Oracle: RNN with **256** hidden units
+ a fully-connected layer with **32** hidden units

Lookup table vs Learned Oracle. LT tends to memorize **HH**, while LO seems to learn their structure.

A GROWING LINE OF WORK

Subsequent Results

Better tail bounds

Extra sketch for low-frequency items [ACNSV, NeurIPS23]

Sliding-window streams

Learned sketches over recent windows [SSM, ICNP24]

Learned Misra–Gries

Deterministic & simpler, insertion-only [ACGSW, ICLR25]

Triangle & 4-cycle counting

Heavy-edge oracle; dynamic-stream extension [CEILNRSWWZ, ICLR22]

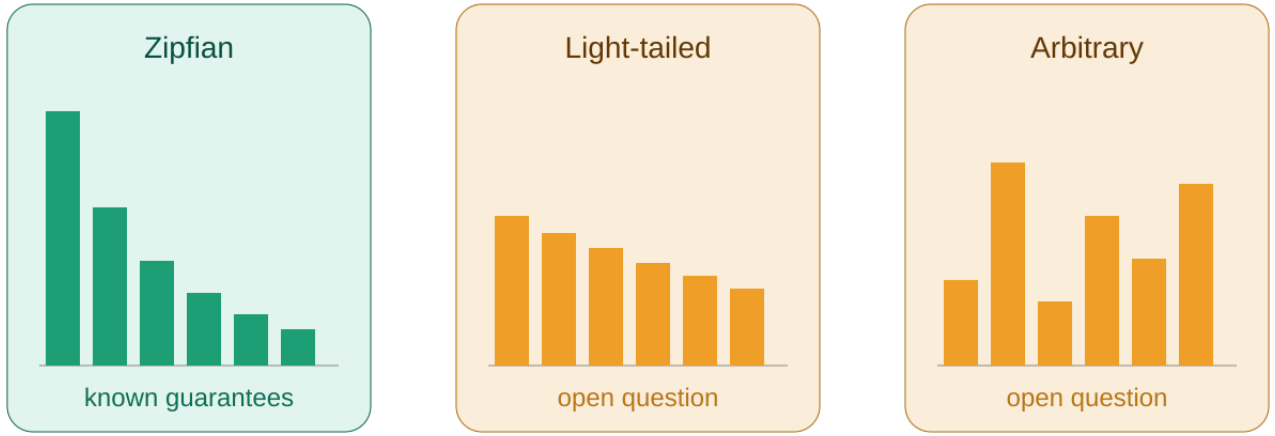
More on Frequency-Vector Problems

[Jiang, Li, Lin, Ruan, Woodruff, ICLR20]

Problem	Without Predictions		With Predictions
Distinct Elements $z \coloneqq (1 \pm \epsilon)\|x\|_0$	$O(\frac{1}{\epsilon^2}\log n\,(\log\frac{1}{\epsilon} + \log\log n))$ [Kane, Nelson, Woodruff'10]	$\Omega(\log n \log\log n)$ [Woodruff, Yang'19]	$O(\frac{1}{\epsilon^2}\log n \log\frac{1}{\epsilon})$
F_p Moment $(p > 2)$ $z \coloneqq (1 \pm \epsilon)\|x\|_p^p$	$\tilde{O}(n^{1-2/p})$ [Andoni, Krauthgamer, Onak'11]	$\Omega(n^{1-2/p})$ [Chakrabarti, Khot, Sun'03] [Bar-Yossef, Jayram, Kumar, Savikumar'04]	$\tilde{O}(n^{1/2-1/p})$
(k,p)-Cascaded Norm $(k \geq p \geq 2)$ $\|x\|_{p,q} \coloneqq \left(\sum_i \left(\sum_j \|x_{i,j}\|^q\right)^{\frac{p}{q}}\right)^q$	$\tilde{O}(n^{1-2/k}d^{1-2/p})$ [Jayram, Woodruff'09]	$\Omega(n^{1-2/k}d^{1-2/p})$ [Jayram, Woodruff'09]	$\tilde{O}(n^{1-1/k-p/(2k)}d^{1/2-1/p})$

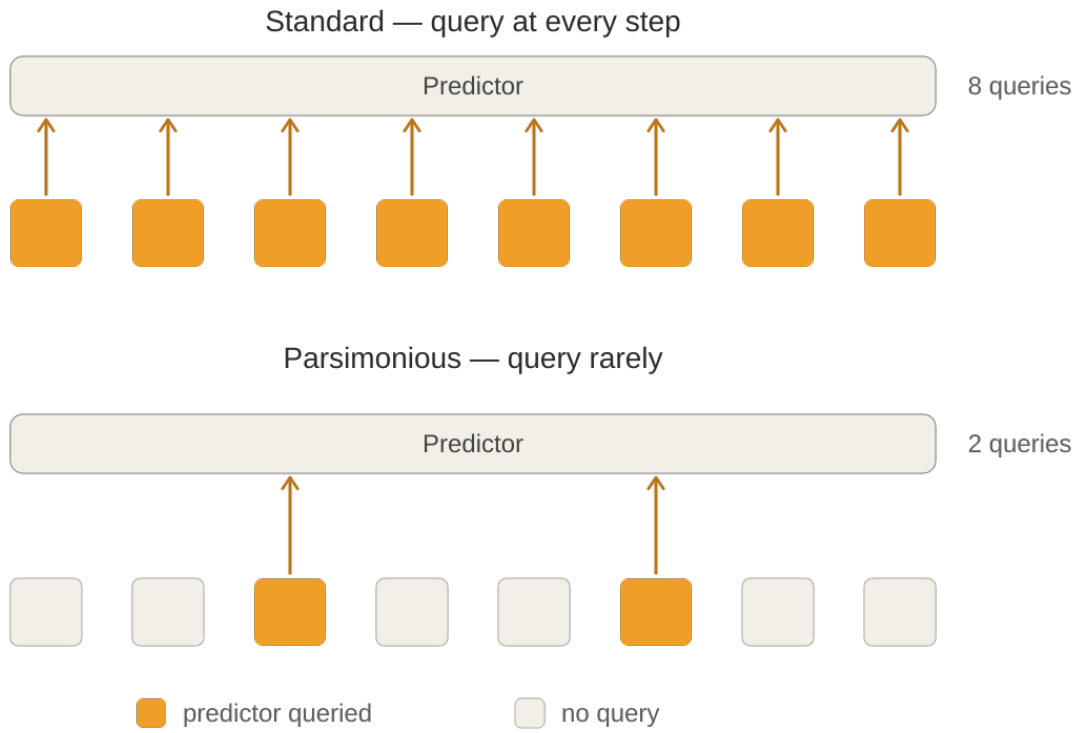
Questions

- What can be shown beyond Zipfian-like distribution?



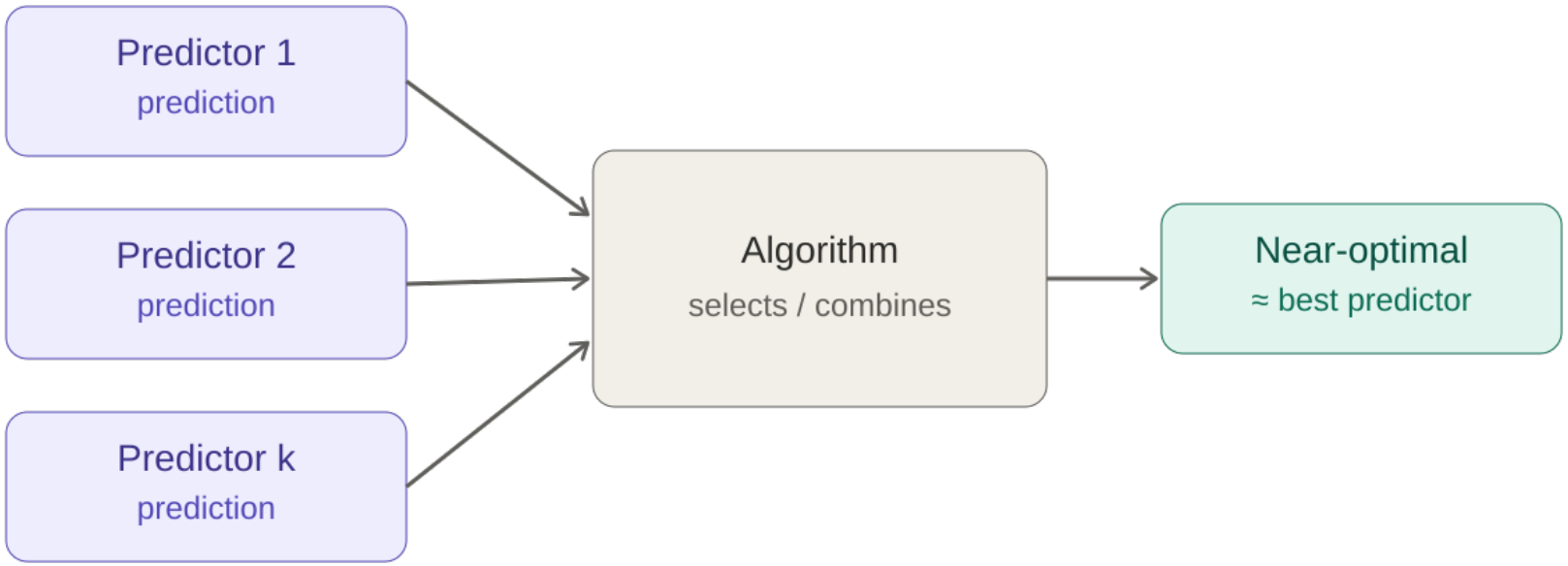
Questions

- What can be shown beyond Zipfian-like distribution?
- Parsimonious Setting



Questions

- What can be shown beyond Zipfian-like distribution?
- Parsimonious Setting
- Multiple Predictions



Learned Structure

Learning-Augmented Algorithms

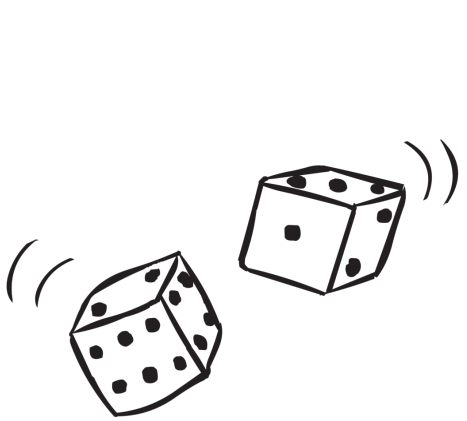
(aka Algorithms with Predictions)

I. Learned-Oracle Paradigm

- Identify a compact "**learnable**" prediction (e.g., **Heavy Hitter Oracle**)

II. Learned-Structure Paradigm

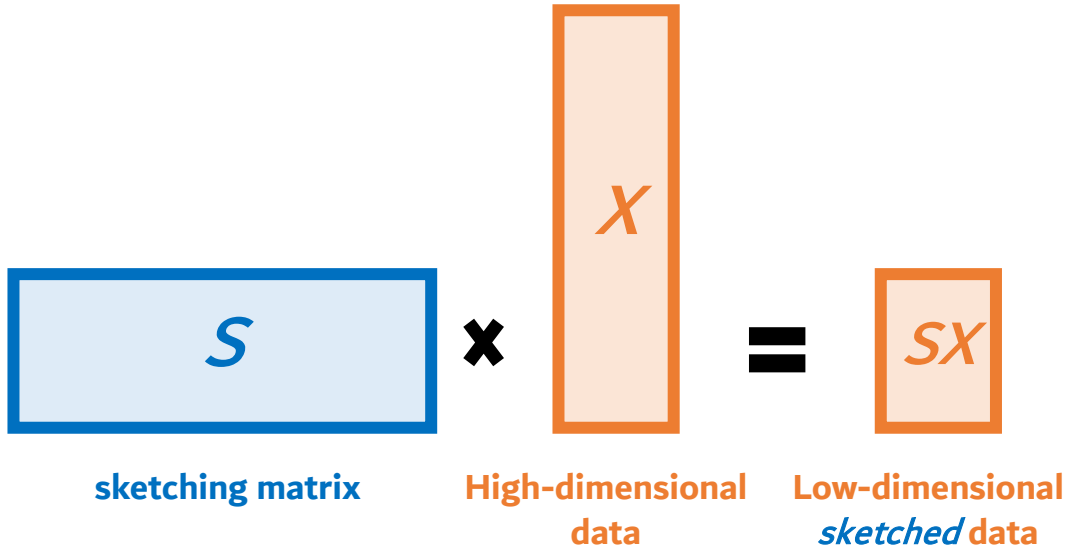
- Identify a (usually random) structure



Linear Sketches

General Sketching Framework

- **Input:** vector x (or matrix X)
- **Compression:** compress x into Sx
- **Output:** solve the problem on Sx



$x \in \mathbb{R}^d, Sx \in \mathbb{R}^s$ where $s \ll d$
e.g., $s \approx \log d$

Linear Sketches

General Sketching Framework

- ❑ **Input:** vector x (or matrix X)
- ❑ **Compression:** compress x into Sx
- ❑ **Output:** solve the problem on Sx

Examples

- **Dimensionality Reduction** (e.g., JL lemma)
- **Numerical Linear Algebra** (e.g., Regression, Low-Rank Approx.)
- **…**

Linear Sketches: APPLICATIONS

Low-Rank Approximation

Input: $A \in \mathbb{R}^{n \times d}$, rank parameter k

Output: rank-k matrix that best approximates A

Linear Sketches: APPLICATIONS

Low-Rank Approximation

Input: $A \in \mathbb{R}^{n \times d}$, rank parameter k

Output: rank-k matrix that best approximates A, $\boldsymbol{A_k = \operatorname{argmin}_{\operatorname{rank}(B)=k} \|A - B\|_F}$

Approximate Low-Rank Approximation

Singular Value Decomposition (SVD)

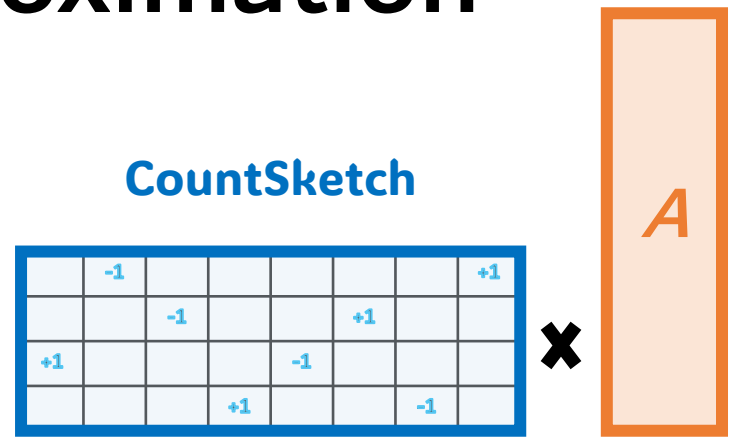
- Optimal, but inefficient
  - **Runtime:** $\min(nd^2, n^2d)$
  - **Space:** nd
- Instead, find an approximately optimal rank-k approximation

  A' such that, $\|A - A'\|_F \leq (1+\varepsilon)\|A - A_k\|_F$

Approximate Low-Rank Approximation

Singular Value Decomposition (SVD)

- Optimal, but inefficient
  - **Runtime:** $\min(nd^2, n^2d)$
  - **Space:** nd



- Instead, find an approximately optimal rank-k approximation

  A' such that, $\|A - A'\|_F \leq (1+\varepsilon)\|A - A_k\|_F$

- Sketching-based approach [Sarlos'06], [Clarkson, Woodruff'09, '13]

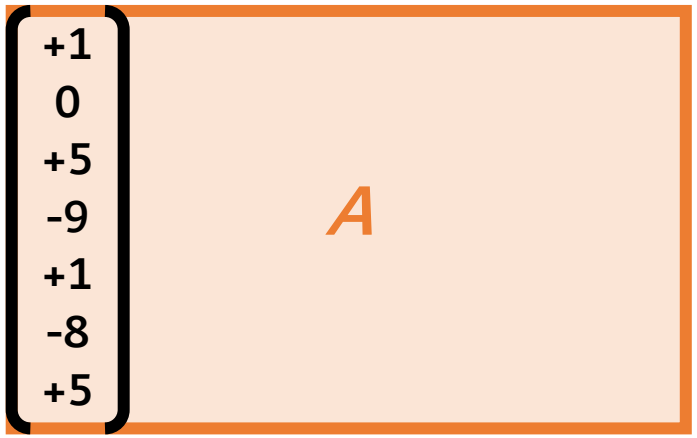
  ❑ Sketch matrix S with $r \ll n$ rows

  ❑ In most algorithms, S is independent of A

Sketching for Low-Rank Approximation

Streaming Setting:

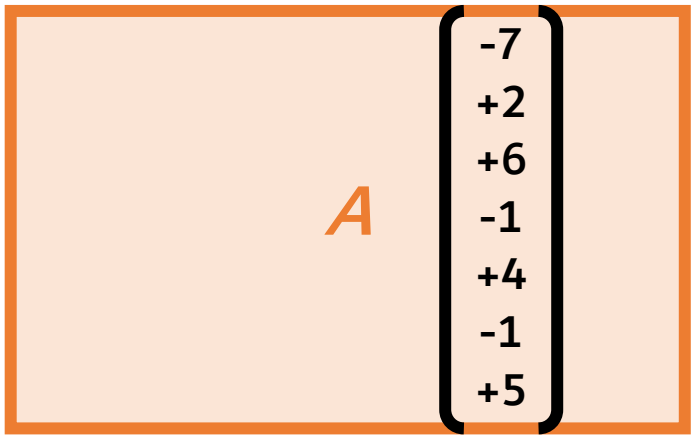
- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space



Sketching for Low-Rank Approximation

Streaming Setting:

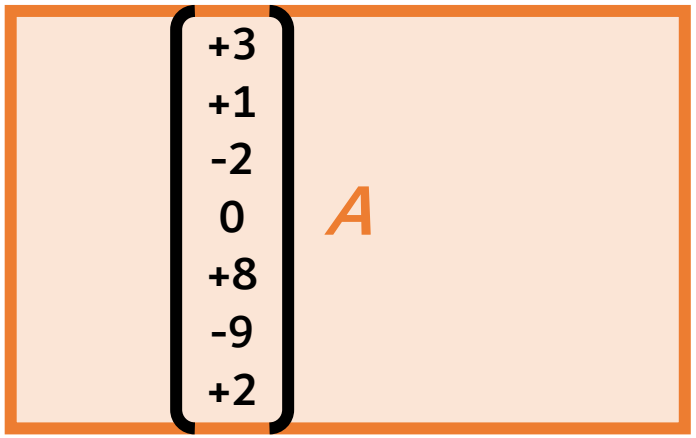
- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space



Sketching for Low-Rank Approximation

Streaming Setting:

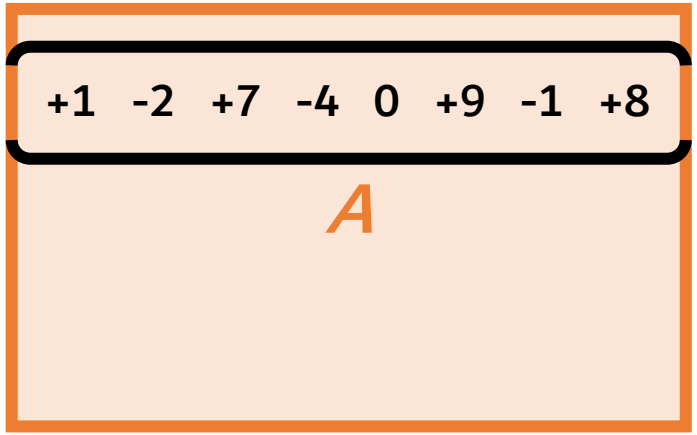
- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space



Sketching for Low-Rank Approximation

Streaming Setting:

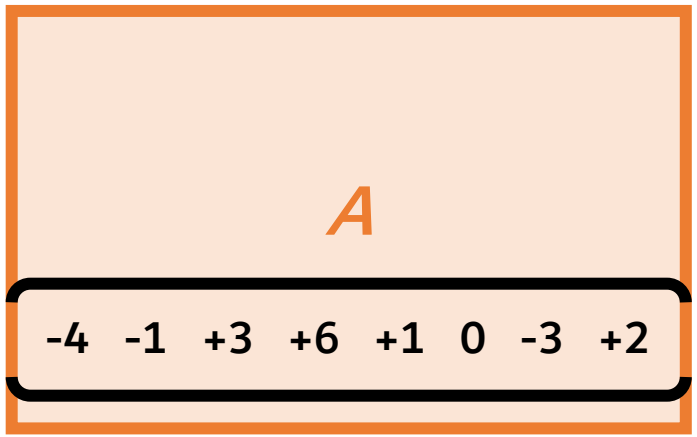
- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space



Sketching for Low-Rank Approximation

Streaming Setting:

- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space



Sketching for Low-Rank Approximation

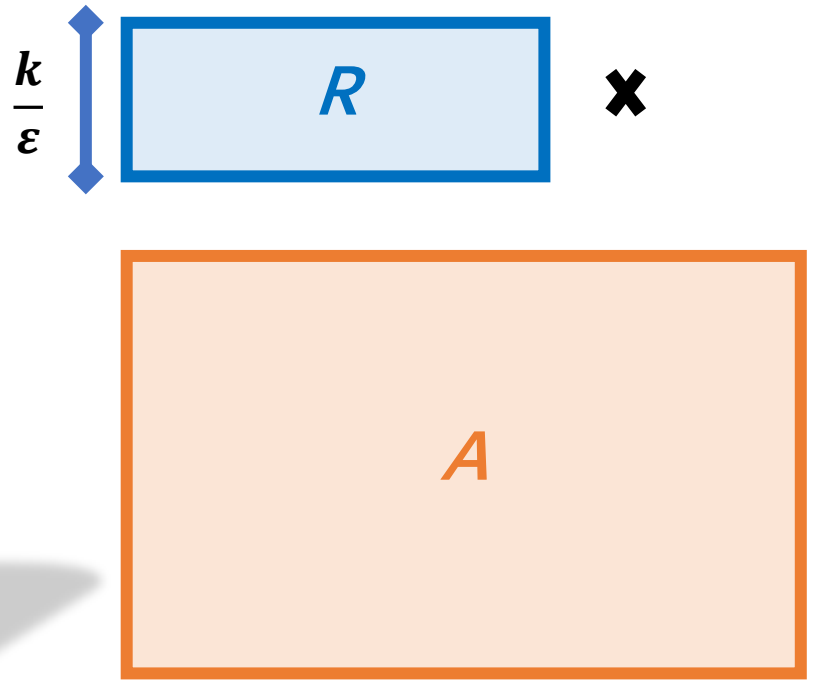
Streaming Setting:

- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space

[Sarlos'06] [Clarkson, Woodruff'09]

Random Sketching $m \times n$ matrix R with $m = \frac{k}{\varepsilon}$ rows

➢ Streaming friendly



Sketching for Low-Rank Approximation

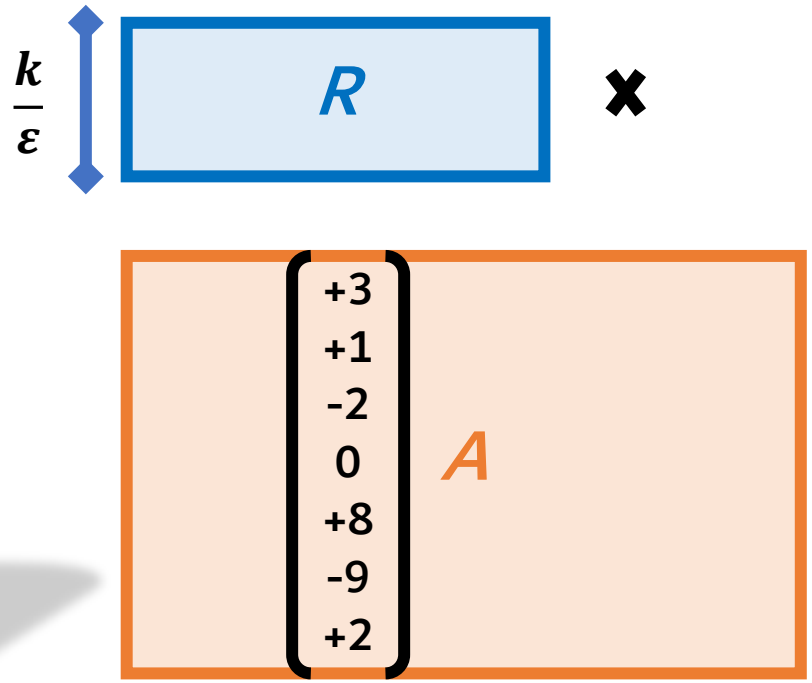
Streaming Setting:

- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space

[Sarlos'06] [Clarkson, Woodruff'09]

Random Sketching $m \times n$ matrix R with $m = \frac{k}{\varepsilon}$ rows

➢ Streaming friendly



Sketching for Low-Rank Approximation

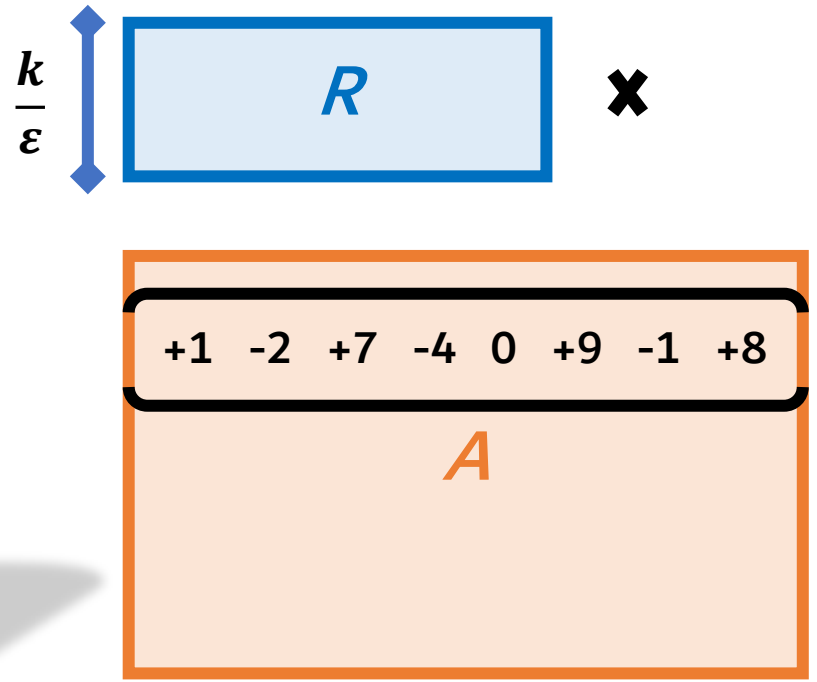
Streaming Setting:

- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space

[Sarlos'06] [Clarkson, Woodruff'09]

Random Sketching $m \times n$ matrix R with $m = \frac{k}{\varepsilon}$ rows

➢ Streaming friendly



Sketching for Low-Rank Approximation

Streaming Setting:

- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space

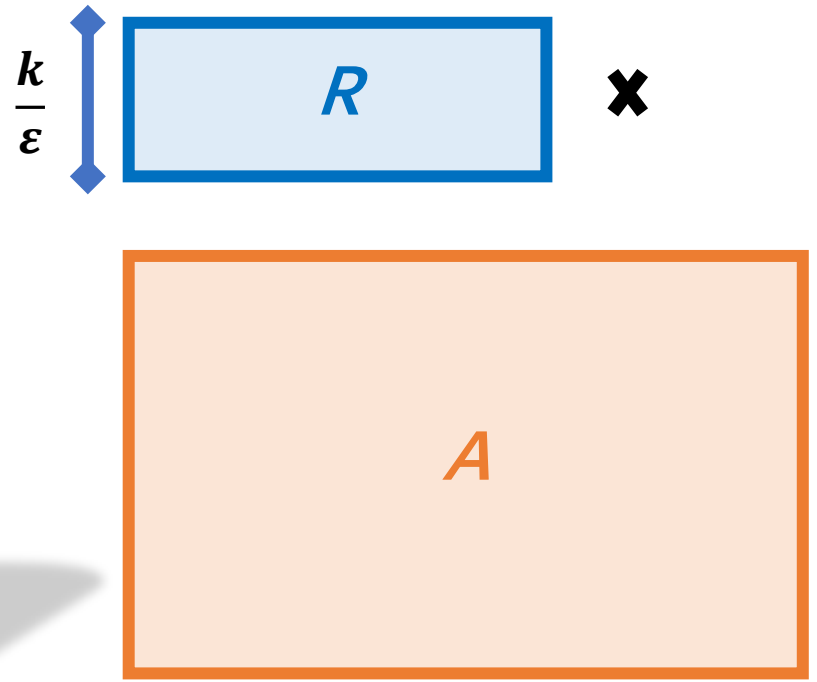
[Sarlos'06] [Clarkson, Woodruff'09]

Random Sketching $m \times n$ matrix R with $m = \frac{k}{\varepsilon}$ rows

➢ Streaming friendly

➢ $(1+\varepsilon)$-approximate rank-k $B = \mathbf{SCW}(A, R)$ in $(n + d) \cdot m$ space (to store RA and R)

✓ Improves over the $n \times d$ space of vanilla **SVD**



Sketching for Low-Rank Approximation

Streaming Setting:

- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space

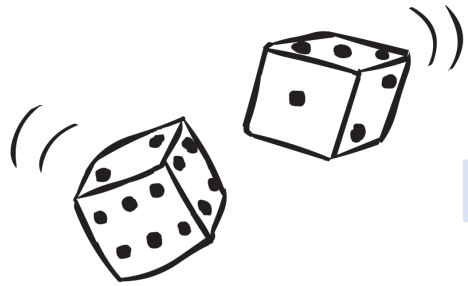
[Sarlos'06] [Clarkson, Woodruff'09]

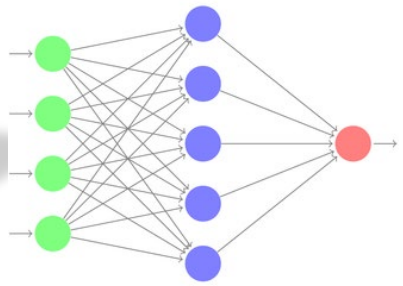
Random Sketching $m \times n$ matrix R with $m = \frac{k}{\varepsilon}$ rows

➢ Streaming friendly

➢ $(1+\varepsilon)$-approximate rank-k $B = \mathbf{SCW}(A, R)$ in $(n + d) \cdot m$ space (to store RA and R)

✓ Improves over the $n \times d$ space of vanilla **SVD**





Learning-Augmented Approach

Streaming Setting:

- **Input:** (row/column) updates to $n \times d$ matrix A arrives as a stream
- **Goal:** compute A_k with minimum amount of space

[Sarlos'06] [Clarkson, Woodruff'09]

Random Sketching matrix R with $m = \frac{k}{\varepsilon}$ rows

➢ $(1+\varepsilon)$-approximate rank-k $B = \mathbf{SCW}(A, R)$ in $(n + d) \cdot m$ space

SCW Algorithm

1. Compute RA
2. Run **SVD** on RA ($RA := \boldsymbol{U\,\Sigma\,V}$)
3. Compute $AV^\top$
4. Return $[AV^\top]_k\, V$

[IVY, NeurIPS19] [LLLVW, ICLR23]

Learned Sketch matrix S with $\bar{m} \ll \frac{k}{\varepsilon}$ rows

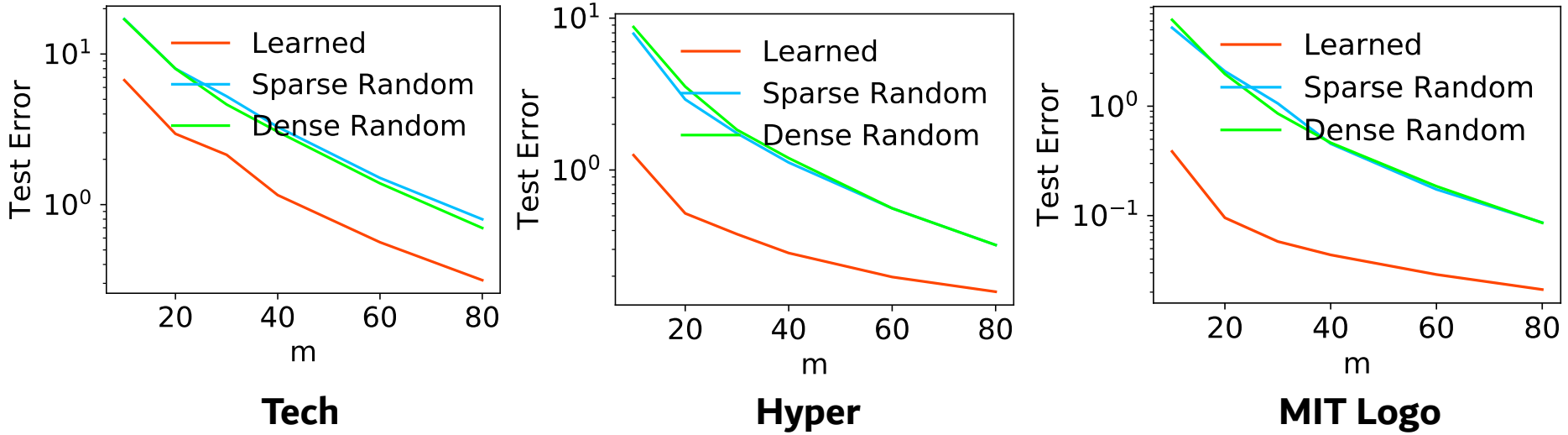
➢ $(1+\varepsilon)$-approximate rank-k $\bar{B} = \mathbf{SCW}(A, R)$ in $(n + d) \cdot \bar{m}$ space

- Training set: $A_1, \ldots, A_N$
- S minimizes $\sum_N \|A_i - \mathbf{SCW}(A_i, S)\|_F$
- **Neural Networks** to find S

Empirical Evaluations

- **Datasets: I.** Videos, **II.** Hyperspectral images (HS-SOD), and **III.** TechTC-300

Compare **empirical recovery error** $\sum_i ||A_i - \mathbf{SCW}(S, A_i)||_F$ - $||A_i - [A_i]_k||_F$ of our **learned matrix** S to sparse and dense **random matrices** R



Fallback Option: Learned + Random

Learned sketch has **better** empirical performance, but **no** worst guarantee

k	m	sketch	Logo	Hyper	Tech
10	20	Learned	0.1	0.52	2.95
10	20	Random	2.09	2.92	7.99

Fallback Option: Learned + Random

Learned sketch has **better** empirical performance, but **no** worst guarantee

k	m	sketch	Logo	Hyper	Tech
10	20	Learned	0.1	0.52	2.95
10	20	Mixed	0.2	0.78	3.73
10	20	Random	2.09	2.92	7.99

Idea. combine **learned** S with **random** rows R

Sketch monotonicity Lemma: Augmenting R with the (learned) matrix **S** may only reduce the error of **SCW**

$$err(SCW(.,\begin{bmatrix} R \\ S \end{bmatrix})) \leq min\left(err(SCW(.,[R]), err(SCW(.,[S])\right)$$

Why Learn the Sketch — Not the Whole Map

01

Valid by construction

Sketch-and-solve returns an exactly rank-k factorization in the right format. A black-box network might not output anything low-rank at all. No need for an extra test!

02

Linearity is preserved

S is a linear sketch: SA is mergeable, composable, and maintainable under streaming updates in one pass and small space. An end-to-end net is none of these.

03

Few parameters to learn

Learn only the entries of S — not an approximation of the entire SVD map. Sample-efficient, and easier to argue about its correctness.

ε-Accurate Predictions

[Gupta, Panigrahi, Subercaseaux, Sun, NeurIPS22]

[Braverman, Dharangutte, Shah, Wang, APPROX24]

[Cohen-Addad, d'Orsi, Gupta, Lee, Panigrahi, NeurIPS24]

[Ghoshal, Makarychev, Makarychev, SODA25]

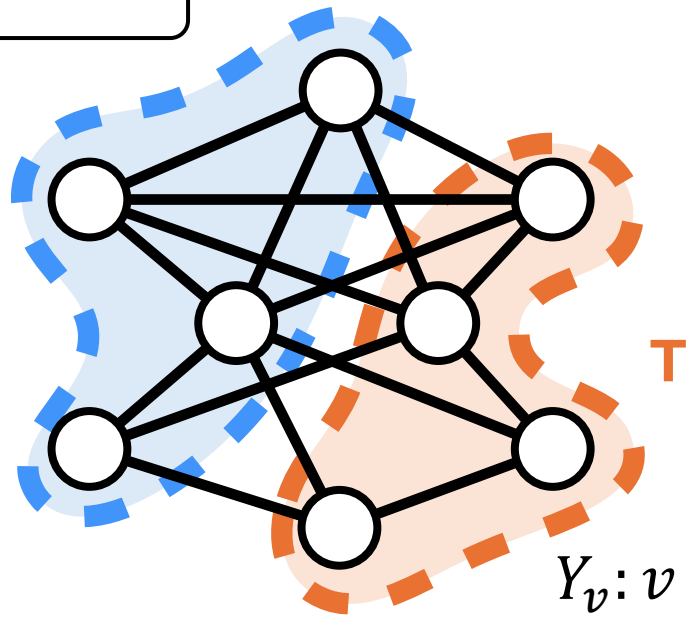
Definition

- $x^* \in \{-1,1\}^n$ denotes some fixed but unknown **optimal solution**
- Oracle access a **prediction vector** $Y \in \{-1,1\}^n$.
- Each entry Y_v is independently correct w.p. $0.5 + \varepsilon$

$$\forall v \in V, \quad Pr[Y_v = x_v^*] = \frac{1}{2} + \varepsilon$$

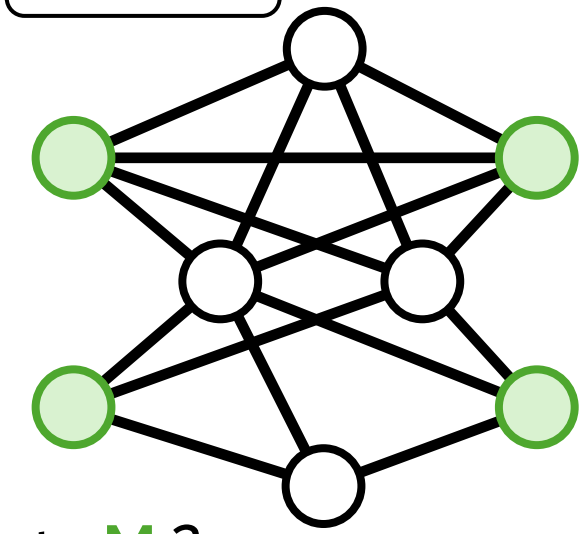
Only slightly better than a random guess

Max-Cut



Y_v: v belongs to **S** ?

MIS

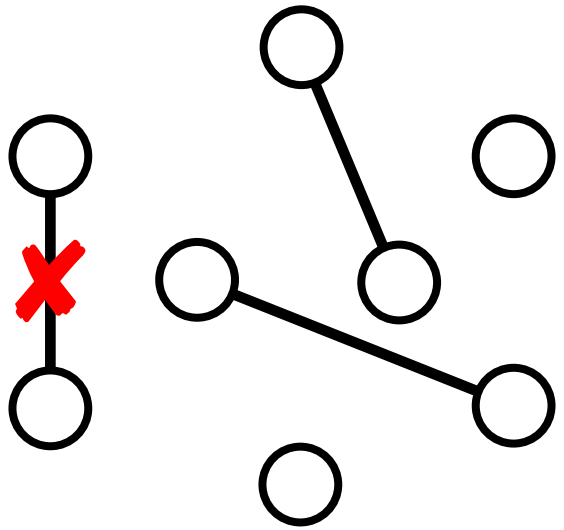


Y_v: v belongs to **M** ?

Graph Streaming Model

- Input graph is presented as a sequence of edge insertions and deletions
  - **Insertion-Only Streams:** consists of edge insertion only
  - **Dynamic Streams:** has both insertions and deletions

- **Goal:** in **one pass**, using **small space (**usually $\tilde{O}(|V|)$ space**)**, compute the solution



Streaming Max-Cut

Input: An undirected, unweighted graph $G = (V, E)$

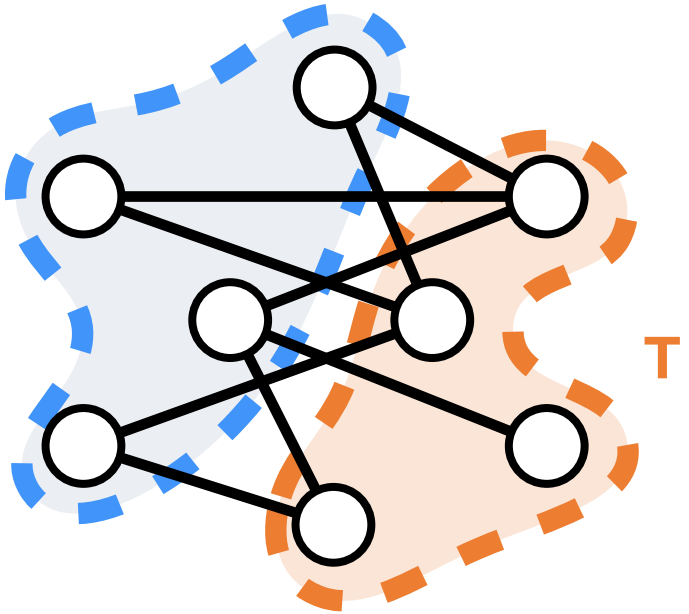
Max-Cut: a bipartition of V into S and T that maximizes the cut value

$$e(S,T) = |\{(u,v) \in E : u \in S \wedge v \in T\}|$$

Estimation Task: estimate the size of Max-Cut

Z is an α-approx. if, $\alpha \cdot \mathrm{OPT} \leq Z \leq \mathrm{OPT}$

- A trivial ½-approx. using $O(\log n)$ bits
- $(\frac{1}{2} + \varepsilon)$-approx. requires $\Omega_\varepsilon(n)$ bits of space [Kapralov, Krachun, STOC19]



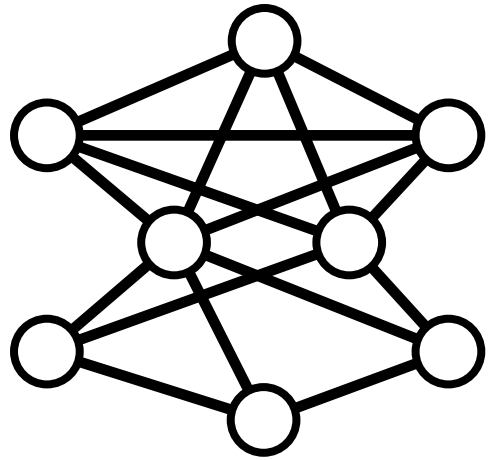
Streaming Max-Cut with Predictions

[Dong, Peng, **V.**, ITCS25] Given ε-accurate predictions, there exists a one-pass streaming algorithm with $\left(\frac{1}{2}+\varepsilon\right)$-approx. that uses

- $poly(1/\varepsilon)$ words of space in **insertion-only streams**, and
- $poly(\log n, 1/\varepsilon)$ words of space in **dynamic streams**.

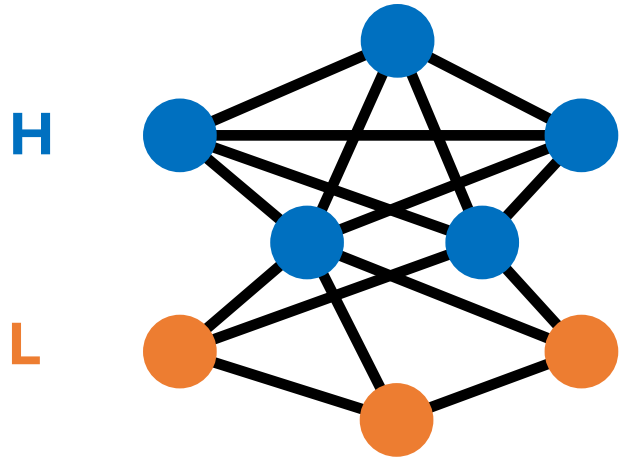
- Bypass the known $(1/2+\varepsilon)$-approx. vs $\Omega_\varepsilon(n)$ space trade-off
- No need to store predictions (only oracle access suffices)

High-Level Overview of the Algorithm



High-Level Overview of the Algorithm

- Divide vertices into **H** (high-deg) and **L** (low-deg) vertices
- Output one of the following Max-Cut estimates:

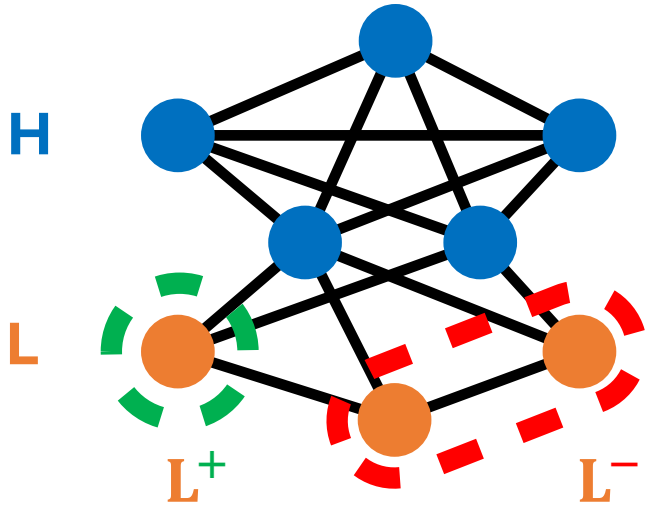


High-Level Overview of the Algorithm

- Divide vertices into **H** (high-deg) and **L** (low-deg) vertices
- Output one of the following Max-Cut estimates:

Candidate Estimate (1)

- Follow **predictions** on **L**, compute the size of (L^+, L^-)

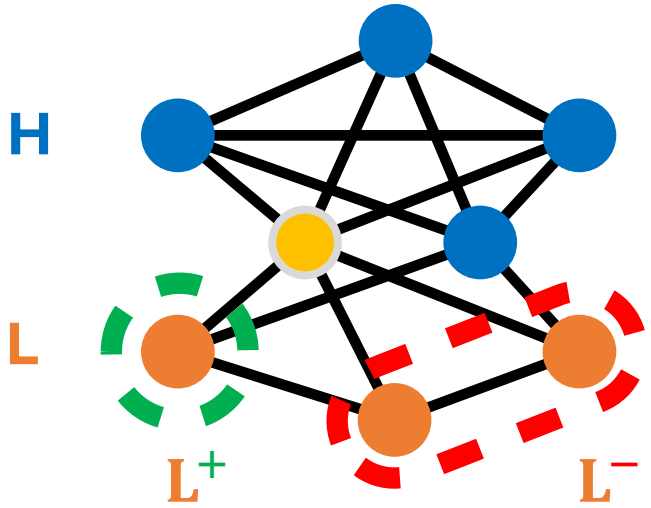


High-Level Overview of the Algorithm

- Divide vertices into **H** (high-deg) and **L** (low-deg) vertices
- Output one of the following Max-Cut estimates:

Candidate Estimate (1)

- Follow **predictions** on **L**, compute the size of (L^+, L^-)
- Extend (L^+, L^-) with vertices in **H** via a greedy approach

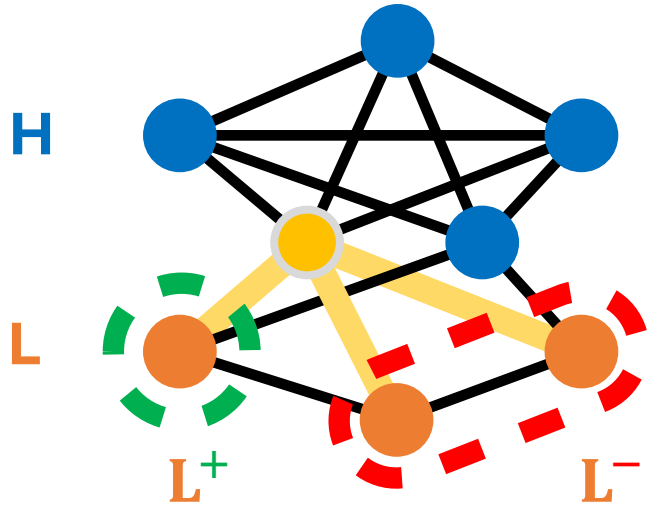


High-Level Overview of the Algorithm

- Divide vertices into **H** (high-deg) and **L** (low-deg) vertices
- Output one of the following Max-Cut estimates:

Candidate Estimate (1)

- Follow **predictions** on **L**, compute the size of (L^+, L^-)
- Extend (L^+, L^-) with vertices in **H** via a greedy approach

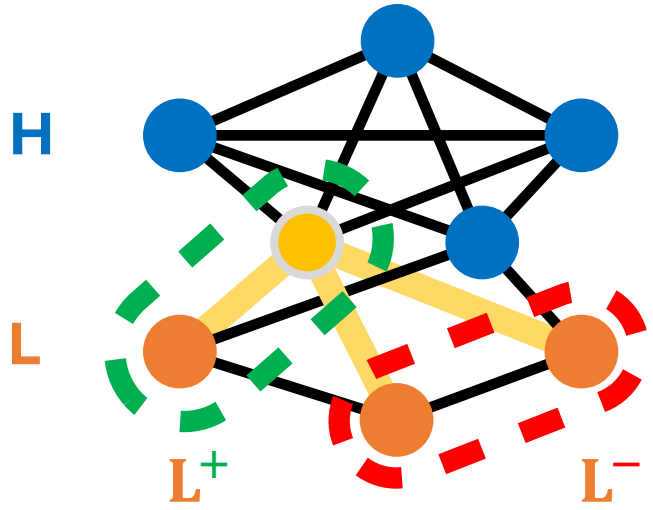


High-Level Overview of the Algorithm

- Divide vertices into **H** (high-deg) and **L** (low-deg) vertices
- Output one of the following Max-Cut estimates:

Candidate Estimate (1)

- Follow **predictions** on **L**, compute the size of (L^+, L^-)
- Extend (L^+, L^-) with vertices in **H** via a greedy approach

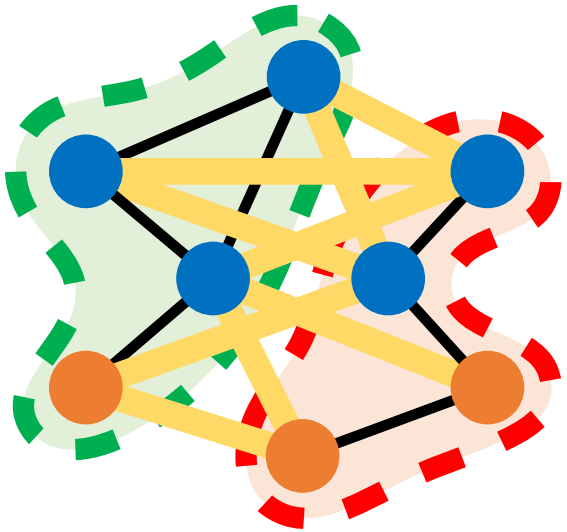


High-Level Overview of the Algorithm

- Divide vertices into **H** (high-deg) and **L** (low-deg) vertices
- Output one of the following Max-Cut estimates:

Candidate Estimate (1)

- Follow **predictions** on **L**, compute the size of (L^+, L^-)
- Extend (L^+, L^-) with vertices in **H** via a greedy approach



High-Level Overview of the Algorithm

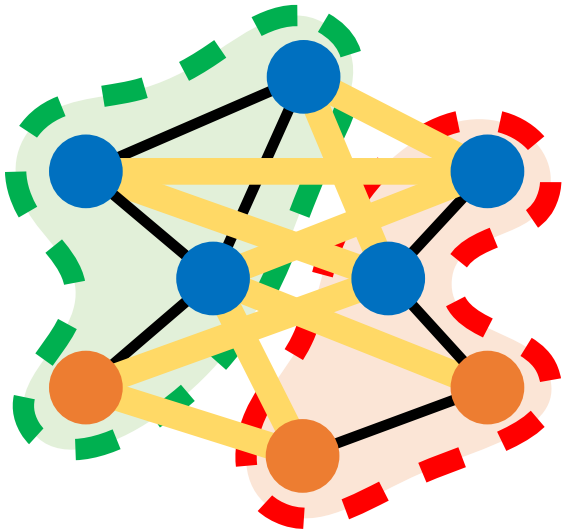
- Divide vertices into **H** (high-deg) and **L** (low-deg) vertices
- Output one of the following Max-Cut estimates:

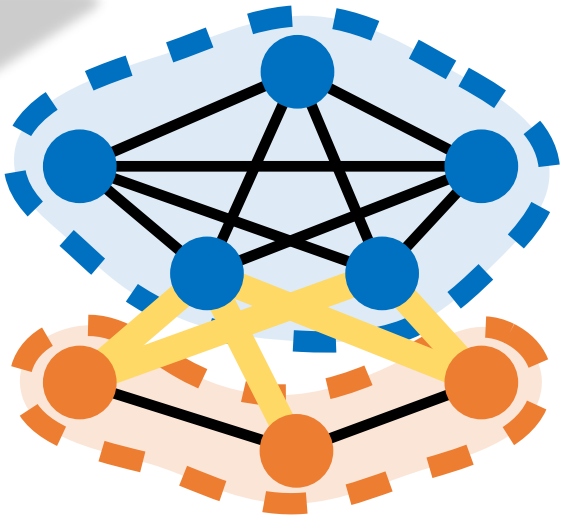
Candidate Estimate (1)

- Follow **predictions** on **L**, compute the size of $(\mathbf{L}^+, \mathbf{L}^-)$
- Extend $(\mathbf{L}^+, \mathbf{L}^-)$ with vertices in **H** via a greedy approach

Candidate Estimate (2)

- Compute the size of (**H**, **L**)





High-level Analysis:

- If $|E(\mathbf{H}, \mathbf{L})| \geq (.5 + \varepsilon)|\mathrm{E}|$: Done
- Else, start with $(.5 + \varepsilon)|E(\mathbf{L})|$ and augment with half of rest

Streaming Implementation

- We cannot compute $\mathbf{L}^+$, $\mathbf{L}^-$ and **H** exactly.
- However, w.h.p.,
  - We can compute a superset $\widetilde{H} \supset H$ via reservoir edge sampling
  - We can estimate all degrees via sketching techniques such as CountSketch
  - Then, we can compute $\mathbf{L}^+$, $\mathbf{L}^-$ and degree of $\widetilde{H}$ in $\mathbf{L}^+$, $\mathbf{L}^-$ approximately.

Extension to general CSP

- Natural approach following the prediction has a barrier (by expectation).
  - For Max-Cut, $.5 + \varepsilon$ is the barrier.
- What if we aim for $(1 - \eta)$-approx, for arbitrarily small values of η?

- In Boolean k-CSP, we're given
  - m constraints, represented by a function $f: \{-1,1\}^k \to \{0,1\}$ over k literals (either x or $-x$): e.g., k-AND: $-x_1 \wedge x_3 \wedge -x_6$
  - **Goal:** find an assignment of variables so that maximum number of constraints are satisfied (i.e., f obtains value 1 on those constraints)

Max k-CSP

- Max-Cut and Max-Dicut are special cases of Max 2-CSP

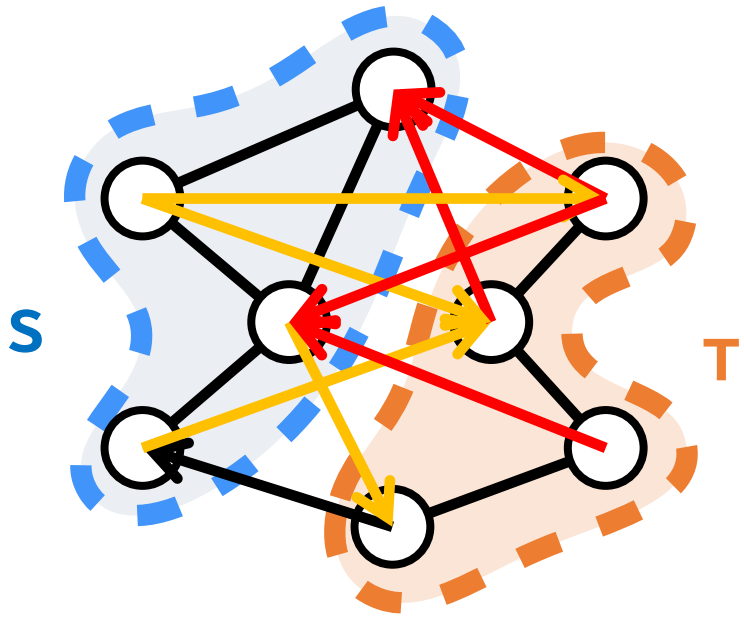
Input: A directed, unweighted graph $G = (V, E)$

Max-Dicut: a bipartition of V into S and T that maximizes the directed cut value

$$e(S,T) = |\{\overrightarrow{uv} \in E : u \in S \wedge v \in T\}|$$

- Follow the prediction approach has a limit of $1/4$.
- Best known (w/o prediction) is .5

[Azarmehr, Behnezhad, Ferrante, Saneian; **STOC'26**]



Max 2-CSP: High-level Idea

We follow a different approach: solve a **system of linear equations**

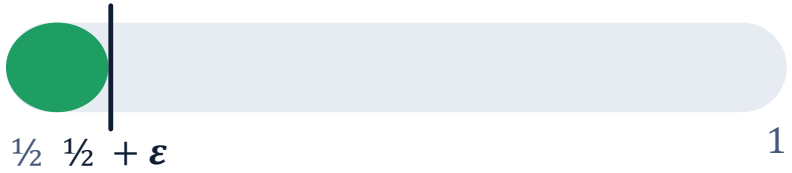
Ex: Max-Cut) Instead of only tracking edges in an optimal cut, we estimate count of

- Set of edges with one endpoint in each of S and T (X_1), and
- Set of edges with both endpoints in either S or T (X_2)

- Then, looking at predicted labels, in expectation:
  - $X_1 = \left(\frac{1}{2} + \varepsilon\right) k_1 + \left(\frac{1}{2} - \varepsilon\right) k_2$
  - $X_2 = \left(\frac{1}{2} + \varepsilon\right) k_2 + \left(\frac{1}{2} - \varepsilon\right) k_1$
- Still, need to deal with concentration challenges by separating high and low degree, and storing the required statistics in streaming setting

Are These Predictions Actually Learnable?

Probability a prediction is correct



The green edge is the **entire** advantage required — barely above a coin flip.

→

breaks the $\Omega(n)$ barrier

RESULT

Better-than-½ Max-Cut in one streaming pass, sublinear space.

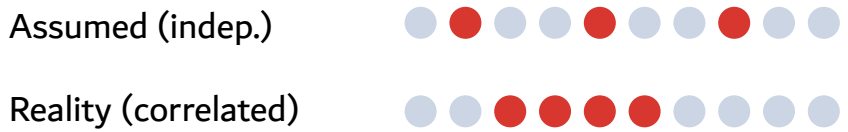
+ It's learnable

Such a weak signal is exactly what a model can pull from node features and historical structure.

node features | history → **model** → ½ + ε

− The catch: independence

Guarantees assume per-vertex predictions are independent, but real predictors err in correlated bursts.



Open Question: Generalizing to correlated predictions.

Questions

- Extending the results to weighted graphs
- Other problems in streaming, such as MIS

Thank you!