Given stream of $n$ items $x_1, \ldots, x_n$ where each $x_i \in U$. Return a set $F$, such that for every $x \in U$:

1. If $f(x) \geq n/k$ then $x \in F$
2. If $f(x) < (1 - \epsilon)n/k$ then $x \notin F$

where $f(x)$ is the number of times $x$ appears in the stream.
(\(\epsilon, k\))-FREQUENT ITEMS PROBLEM

Given stream of \(n\) items \(x_1, \ldots, x_n\) where each \(x_i \in U\). Return a set \(F\), such that for every \(x \in U\):

1. If \(f(x) \geq n/k\) then \(x \in F\)
2. If \(f(x) < (1 - \epsilon)n/k\) then \(x \notin F\)

where \(f(x)\) is the number of times \(x\) appears in the stream.

**Relationship to Frequency Estimation.** Note that if you have an estimate \(\tilde{f}(x)\) for each each \(f(x)\) such that

\[
f(x) \leq \tilde{f}(x) \leq f(x) + \epsilon n/k
\]

then you can solve the above problem.
Count-Min Sketch: A random hashing based method closely related to bloom filters.
**FREQUENT ELEMENTS WITH COUNT-MIN SKETCH**

Count-Min Sketch: A random hashing based method closely related to bloom filters.

Random hash function $h$

$m$ length array $A$:  

$$
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
$$
Count-Min Sketch: A random hashing based method closely related to bloom filters.

Let $A[h(x)]$ estimate $f(x)$, the frequency of $x$ in the stream.

- Claim: $A[h(x)] \geq f(x)$.
- Claim: $A[h(x)] \leq f(x) + \frac{2n}{m}$ with probability at least $\frac{1}{2}$.

How can we increase this probability to $1 - \delta$ for arbitrary $\delta > 0$?
Count-Min Sketch: A random hashing based method closely related to bloom filters.

- **Claim:** \( A[h(x)] \geq f(x) \).
- **Claim:** \( A[h(x)] \leq f(x) + \frac{2n}{m} \) with probability at least \( \frac{1}{2} \).

How can we increase this probability to \( 1 - \delta \) for arbitrary \( \delta > 0 \)?
Count-Min Sketch: A random hashing based method closely related to bloom filters.

Count-Min Sketch:

- **Claim**: \( A[h(x)] \geq f(x) \).
- **Claim**: \( A[h(x)] \leq f(x) + 2n/m \) with probability at least \( \frac{1}{2} \).

How can we increase this probability to \( 1 - \delta \) for arbitrary \( \delta > 0 \)?
Count-Min Sketch: A random hashing based method closely related to bloom filters.

- **Claim:** \( A[h(x)] \geq f(x) \)
- **Claim:** \( A[h(x)] \leq f(x) + \frac{2n}{m} \) with probability at least \( \frac{1}{2} \).

How can we increase this probability to \( 1 - \delta \) for arbitrary \( \delta > 0 \)?
Count-Min Sketch: A random hashing based method closely related to bloom filters.

\[ A[h(x)] \geq f(x) \]

\[ A[h(x)] \leq f(x) + \frac{2n}{m} \] with probability at least \( \frac{1}{2} \).

How can we increase this probability to \( 1 - \delta \) for arbitrary \( \delta > 0 \)?
Count-Min Sketch: A random hashing based method closely related to bloom filters.

Use $A[h(x)]$ to estimate $f(x)$, the frequency of $x$ in the stream.

- **Claim:** $A[h(x)] \geq f(x)$.
- **Claim:** $A[h(x)] \leq f(x) + 2n/m$ with probability at least $1/2$. 

How can we increase this probability to $1 - \delta$ for arbitrary $\delta > 0$?
Count-Min Sketch: A random hashing based method closely related to bloom filters.

Use $A[h(x)]$ to estimate $f(x)$, the frequency of $x$ in the stream.

- **Claim:** $A[h(x)] \geq f(x)$.
- **Claim:** $A[h(x)] \leq f(x) + 2n/m$ with probability at least $1/2$.

How can we increase this probability to $1 - \delta$ for arbitrary $\delta > 0$?
COUNT-MIN SKETCH ACCURACY

Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$.

What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2n/m]$?

Answer: $\geq 1 - 1/2^t$.

Setting $t = \log(1/\delta)$ ensures probability is at least $1 - \delta$.

Setting $m = 2^k/\epsilon$ ensures the error $2n/m$ is $\epsilon n/k$ and this is enough to determine whether we need to output the element.
**Count-Min Sketch Accuracy**

- Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$.

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2n/m]$?

  Answer: $\geq 1 - 1/2^t$.

- Setting $t = \log(1/\delta)$ ensures probability is at least $1 - \delta$.

- Setting $m = 2^k/\epsilon$ ensures the error $2n/m$ is $\epsilon n/k$ and this is enough to determine whether we need to output the element.
• Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in \mathbb{T}} A_i[h_i(x)]$.

• What is $Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2n/m]$?

  Answer: $\geq 1 - \frac{1}{2^t}$.

  Setting $t = \log(1/\delta)$ ensures probability is at least $1 - \delta$.

  Setting $m = 2^k/\epsilon$ ensures the error $2n/m$ is $\epsilon n/k$ and this is enough to determine whether we need to output the element.
**COUNT-MIN SKETCH ACCURACY**

Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$.

What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{2n}{m}]$?

**Answer:** $\geq 1 - \frac{1}{2^t}$.

Setting $t = \log(\frac{1}{\delta})$ ensures probability is at least $1 - \delta$.

Setting $m = 2^k/\epsilon$ ensures the error $\frac{2n}{m}$ is $\epsilon n/k$ and this is enough to determine whether we need to output the element.

### Example

- **$t$ random hash functions $h_1, h_2, \ldots, h_t$**
- **$t$ length $m$ arrays**
  - $A_1$: 2 5 1 0 6 12 104 1 3 4
  - $A_2$: 1 6 1 10 78 80 4 11 3 5
  - $A_t$: 90 1 52 6 3 12 33 9 3 2

![Diagram](image)
• Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \).

• What is \( \Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2\frac{n}{m}] \)?

  \[
  \geq 1 - \frac{1}{2t}
  \]

• Setting \( t = \log(\frac{1}{\delta}) \) ensures probability is at least \( 1 - \delta \).

• Setting \( m = 2^k/\epsilon \) ensures the error \( 2\frac{n}{m} \) is \( \epsilon n/k \) and this is enough to determine whether we need to output the element.
• Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \).

• What is \( \Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{2n}{m}] \)?

Answer: \( \geq 1 - \frac{1}{2t} \).

• Setting \( t = \log(\frac{1}{\delta}) \) ensures probability is at least \( 1 - \delta \).

• Setting \( m = \frac{2k}{\epsilon} \) ensures the error \( \frac{2n}{m} \) is \( \frac{\epsilon n}{k} \) and this is enough to determine whether we need to output the element.
• Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$.
- Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$.
- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2n/m]$?
\section*{Count-Min Sketch Accuracy}

- Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \).
- What is \( \Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2n/m] \)? \textbf{Answer:} \( \geq 1 - 1/2^t \).
Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \).

What is \( \Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2n/m] \)? Answer: \( \geq 1 - 1/2^t \).

Setting \( t = \log(1/\delta) \) ensures probability is at least \( 1 - \delta \).
COUNT-MIN SKETCH ACCURACY

- Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \).
- What is \( \Pr[f(x) \leq \tilde{f}(x) \leq f(x) + 2n/m] \)? Answer: \( \geq 1 - 1/2^t \).
- Setting \( t = \log(1/\delta) \) ensures probability is at least \( 1 - \delta \).
- Setting \( m = 2k/\epsilon \) ensures the error \( 2n/m \) is \( \epsilon n/k \) and this is enough to determine whether we need to output the element.
**Upshot:** Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O(\log(1/\delta) \cdot k/\epsilon)$ space.
**Upshot:** Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O(\log(1/\delta) \cdot k/\epsilon)$ space.

- Accurate enough to solve the $(\epsilon, k)$-Frequent elements problem: Can distinguish between items with frequency $\frac{n}{k}$ and those with frequency $< (1 - \epsilon) \frac{n}{k}$. 

How should we set $\delta$ if we want a good estimate for all items at once, with 99% probability? $\delta = \frac{0.01}{|U|}$ ensures $\Pr[\text{there exists } x \in U \text{ with a bad estimate}] \leq \sum_{x \in U} \Pr[\text{estimate for } x \text{ is bad}] \leq \frac{0.01}{|U|} = 0.01$. 


**Upshot:** Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O(\log(1/\delta) \cdot k/\epsilon)$ space.

- Accurate enough to solve the $(\epsilon, k)$-Frequent elements problem: Can distinguish between items with frequency $\frac{n}{k}$ and those with frequency $< (1 - \epsilon) \frac{n}{k}$.
- How should we set $\delta$ if we want a good estimate for all items at once, with 99% probability?
**Count-Min Sketch**

**Upshot:** Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O(\log(1/\delta) \cdot k/\epsilon)$ space.

- Accurate enough to solve the $(\epsilon, k)$-Frequent elements problem: Can distinguish between items with frequency $\frac{n}{k}$ and those with frequency $< (1 - \epsilon)\frac{n}{k}$.
- How should we set $\delta$ if we want a good estimate for all items at once, with 99% probability? $\delta = 0.01/|U|$ ensures

  $$
  \Pr[\exists x \in U \text{ with a bad estimate}] 
  \leq \sum_{x \in U} \Pr[\text{estimate for } x \text{ is bad}] 
  \leq \sum_{x \in U} 0.01/|U| = 0.01
  $$
Count-min sketch gives an accurate frequency estimate for every item in the stream. But how do we identify the frequent items without having to look up the estimated frequency for $x \in U$?
Count-min sketch gives an accurate frequency estimate for every item in the stream. But how do we identify the frequent items without having to look up the estimated frequency for $x \in U$?

One approach:

• Maintain a set $F$ while processing the stream:
• At step $i$:
  • Add $i$th stream element to $F$ if it’s estimated frequency is $\geq i/k$ and it isn’t already in $F$.
  • Remove any element from $F$ whose estimated frequency is $< i/k$.
• Store at most $k$ items at once and have all items with frequency $\geq n/k$ stored at the end of the stream.
Questions on Frequent Elements?
‘Big Data’ means not just many data points, but many measurements per data point. I.e., very high dimensional data.
‘Big Data’ means not just many data points, but many measurements per data point. I.e., very high dimensional data.

- Twitter has 321 million active monthly users. Records (tens of) thousands of measurements per user: who they follow, who follows them, when they last visited the site, timestamps for specific interactions, how many tweets they have sent, the text of those tweets, etc.
'Big Data' means not just many data points, but many measurements per data point. I.e., very high dimensional data.

- Twitter has 321 million active monthly users. Records (tens of) thousands of measurements per user: who they follow, who follows them, when they last visited the site, timestamps for specific interactions, how many tweets they have sent, the text of those tweets, etc.

- A 3 minute Youtube clip with a resolution of $500 \times 500$ pixels at 15 frames/second with 3 color channels is a recording of $\geq 2$ billion pixel values. Even a $500 \times 500$ pixel color image has 750,000 pixel values.
‘Big Data’ means not just many data points, but many measurements per data point. I.e., very high dimensional data.

- Twitter has 321 million active monthly users. Records (tens of) thousands of measurements per user: who they follow, who follows them, when they last visited the site, timestamps for specific interactions, how many tweets they have sent, the text of those tweets, etc.

- A 3 minute Youtube clip with a resolution of $500 \times 500$ pixels at 15 frames/second with 3 color channels is a recording of $\geq 2$ billion pixel values. Even a $500 \times 500$ pixel color image has 750,000 pixel values.

- The human genome contains 3 billion+ base pairs. Genetic datasets often contain information on 100s of thousands+ mutations and genetic markers.
In data analysis and machine learning, data points with many attributes are often stored, processed, and interpreted as high dimensional vectors, with real valued entries.
In data analysis and machine learning, data points with many attributes are often stored, processed, and interpreted as high dimensional vectors, with real valued entries.

\[ \text{ATAGCCGTA}\overset{\rightarrow}{x} = [1 \ 2 \ 1 \ 3 \ 4 \ 4 \ 3 \ 2 \ 1 \ 3 \ 4] \]

\[ \text{5}\overset{\rightarrow}{x} = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1\ldots] \]
In data analysis and machine learning, data points with many attributes are often stored, processed, and interpreted as **high dimensional vectors**, with real valued entries.

Data as vectors and matrices

*Similarities/distances between vectors (e.g., $\langle x, y \rangle$, $\|x - y\|_2$) have meaning for underlying data points.*
DATASETS AS VECTORS AND MATRICES

Data points are interpreted as high dimensional vectors, with real valued entries. Data set is interpreted as a matrix.

Data Points: $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \mathbb{R}^d$.

Data Set: $X \in \mathbb{R}^{n \times d}$ with $i^{th}$ rows equal to $\vec{x}_i$. 
Data points are interpreted as high dimensional vectors, with real valued entries. Data set is interpreted as a matrix.

**Data Points:** \( \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \mathbb{R}^d \).

**Data Set:** \( X \in \mathbb{R}^{n \times d} \) with \( i^{th} \) rows equal to \( \vec{x}_i \).

\[ X \in \mathbb{R}^{n \times d} \]

n = 3000 images

d = 784 pixels
Datasets as Vectors and Matrices

Data points are interpreted as **high dimensional vectors**, with real valued entries. Data set is interpreted as a matrix.

**Data Points:** \( \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \mathbb{R}^d \).

**Data Set:** \( X \in \mathbb{R}^{n \times d} \) with \( i^{th} \) rows equal to \( \vec{x}_i \).

Many data points \( n \) \( \rightarrow \) tall. Many dimensions \( d \) \( \rightarrow \) wide.
**Dimensionality Reduction:** Compress data points so that they lie in many fewer dimensions.
**Dimensionality Reduction:** Compress data points so that they lie in many fewer dimensions.

\[ \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^d \rightarrow \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m \text{ for } m \ll d. \]

\[
\begin{array}{c}
5 \\
\end{array} 
\xrightarrow{\text{Dimensionality Reduction}} 
\begin{array}{c}
x = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ldots] \\
\tilde{x} = [-5.5 \ 4 \ 3.2 \ -1]
\end{array}
\]
**Dimensionality Reduction**: Compress data points so that they lie in many fewer dimensions.

\[
\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d \rightarrow \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m \text{ for } m \ll d.
\]

\[
x = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ldots] \rightarrow \tilde{x} = [-5.5 \ 4 \ 3.2 \ -1]
\]

‘Lossy compression’ that still preserves important information about the relationships between \(\vec{x}_1, \ldots, \vec{x}_n\).
**Dimensionality Reduction**: Compress data points so that they lie in many fewer dimensions.

\[ \bar{x}_1, \ldots, \bar{x}_n \in \mathbb{R}^d \rightarrow \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m \text{ for } m \ll d. \]

‘Lossy compression’ that still preserves important information about the relationships between \( \bar{x}_1, \ldots, \bar{x}_n \).

Generally will not consider directly how well \( \tilde{x}_i \) approximates \( \bar{x}_i \).
Low Distortion Embedding: Given $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, distance function $D$, and error parameter $\epsilon \geq 0$, find $\mathbf{\tilde{x}}_1, \ldots, \mathbf{\tilde{x}}_n \in \mathbb{R}^m$ (where $m \ll d$) and distance function $\tilde{D}$ such that for all $i, j \in [n]$:

$$(1 - \epsilon)D(\mathbf{x}_i, \mathbf{x}_j) \leq \tilde{D}(\mathbf{\tilde{x}}_i, \mathbf{\tilde{x}}_j) \leq (1 + \epsilon)D(\mathbf{x}_i, \mathbf{x}_j).$$

We'll focus on the case where $D$ and $\tilde{D}$ are Euclidean distances. I.e., the distance between two vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i} (x(i) - y(i))^2}.$$

This is related to the Euclidean norm, $\|\mathbf{z}\|_2 = \sqrt{\sum_{i=1}^{n} z(i)^2}$. 
**Johnson-Lindenstrauss Lemma**: For any set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and $\epsilon > 0$ there exists a linear map $M : \mathbb{R}^d \to \mathbb{R}^m$ such that $m = O\left(\frac{\log n}{\epsilon^2}\right)$ and letting $\tilde{x}_i = M\mathbf{x}_i$:

For all $i, j$:

$$(1 - \epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1 + \epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|_2.$$
Johnson-Lindenstrauss Lemma: For any set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and $\epsilon > 0$ there exists a linear map $M : \mathbb{R}^d \to \mathbb{R}^m$ such that $m = O\left(\frac{\log n}{\epsilon^2}\right)$ and letting $\tilde{x}_i = M\mathbf{x}_i$:

For all $i, j$:

$$\frac{1 - \epsilon}{2} \|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq \frac{1 + \epsilon}{2} \|\mathbf{x}_i - \mathbf{x}_j\|_2.$$ 

For $d = 1$ trillion, $\epsilon = .05$, and $n = 100,000$, $m \approx 6600$. 
**The Johnson-Lindenstrauss Lemma**

**Johnson-Lindenstrauss Lemma:** For any set of points $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and $\epsilon > 0$ there exists a linear map $M : \mathbb{R}^d \to \mathbb{R}^m$ such that $m = O\left(\frac{\log n}{\epsilon^2}\right)$ and letting $\tilde{x}_i = M\vec{x}_i$:

For all $i, j$:

$$(1 - \epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1 + \epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$ 

For $d = 1$ trillion, $\epsilon = .05$, and $n = 100,000$, $m \approx 6600$.

Very surprising! Powerful result with a simple construction: applying a random linear transformation to a set of points preserves distances between all those points with high probability.
For any $\vec{x}_1, \ldots, \vec{x}_n$ and $M \in \mathbb{R}^{m \times d}$ with each entry chosen independently from $\mathcal{N}(0, 1/m)$, with high probability, letting $\tilde{x}_i = M\vec{x}_i$:

For all $i, j$: $(1 - \epsilon) \|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1 + \epsilon) \|\vec{x}_i - \vec{x}_j\|_2$.

\[
m = O\left(\frac{\log n}{\epsilon^2}\right)
\]

random linear transformation (random projection)

compressed output point (low dimensions)

input point (high dimensions)
For any \( \vec{x}_1, \ldots, \vec{x}_n \) and \( M \in \mathbb{R}^{m \times d} \) with each entry chosen independently from \( \mathcal{N}(0, 1/m) \), with high probability, letting \( \tilde{x}_i = M \vec{x}_i \):

For all \( i, j \):

\[
(1 - \epsilon) \| \vec{x}_i - \vec{x}_j \|_2 \leq \| \tilde{x}_i - \tilde{x}_j \|_2 \leq (1 + \epsilon) \| \vec{x}_i - \vec{x}_j \|_2.
\]

- \( M \) is known as a random projection. It is a random linear function, mapping length \( d \) vectors to length \( m \) vectors.
For any $\vec{x}_1, \ldots, \vec{x}_n$ and $M \in \mathbb{R}^{m \times d}$ with each entry chosen independently from $\mathcal{N}(0, 1/m)$, with high probability, letting $\tilde{x}_i = M\vec{x}_i$:

For all $i, j$:

$$(1 - \epsilon) \|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1 + \epsilon) \|\vec{x}_i - \vec{x}_j\|_2.$$

- $M$ is known as a random projection. It is a random linear function, mapping length $d$ vectors to length $m$ vectors.
- $M$ is data oblivious. Stark contrast to methods like PCA.
• Alternative constructions: $\pm 1$ entries, sparse (most entries 0), Fourier structured, etc. $\implies$ efficient computation of $\tilde{x}_i = M\tilde{x}_i$. 
• Alternative constructions: ±1 entries, sparse (most entries 0), Fourier structured, etc. \(\Rightarrow\) efficient computation of \(\tilde{x}_i = M\tilde{x}_i\).

• Data oblivious property means that once \(M\) is chosen, \(\tilde{x}_1, \ldots, \tilde{x}_n\) can be computed in a stream with little memory.

• Storage is just \(O(nm)\) rather than \(O(nd)\).
• Alternative constructions: ±1 entries, sparse (most entries 0), Fourier structured, etc. $\Longrightarrow$ efficient computation of $\tilde{x}_i = M\tilde{x}_i$.

• Data oblivious property means that once $M$ is chosen, $\tilde{x}_1, \ldots, \tilde{x}_n$ can be computed in a stream with little memory.

• Storage is just $O(nm)$ rather than $O(nd)$.

• Compression can be performed in parallel on different servers.
• Alternative constructions: ±1 entries, sparse (most entries 0), Fourier structured, etc. \( \Rightarrow \) efficient computation of \( \tilde{x}_i = M\tilde{x}_i \).

• Data oblivious property means that once \( M \) is chosen, \( \tilde{x}_1, \ldots, \tilde{x}_n \) can be computed in a stream with little memory.

• Storage is just \( O(nm) \) rather than \( O(nd) \).

• Compression can be performed in parallel on different servers.

• When new data points are added, can be easily compressed, without updating existing points.