Approximating Matrix Multiplication using Randomised Numerical Linear Algebra

Shreshtha Modi (202411015), Dhairya Patel (202411082)

***Abstract*—Matrix multiplication, particularly for large datasets, is a computationally intensive task.**

In this project, we try to approximate the matrix multiplication using the sketch and solve paradigm and use techniques such as priority sampling, coordinated sampling, uniform sampling, JL lemma (Gaussian, PHD and Count sketch), leverage scores, and square root of leverage scores.

We apply these techniques to approximate $A^T B$, where $A \in \mathbb{R}^{n \times d}$ and $B \in \mathbb{R}^{n \times d}$.

The matrices here are generated synthetically and are mainly divided into four parts *NG, NB (Matrices with Bad Leverage scores), UG and UB (Matrices with Good Leverage scores)*

***Index Terms*—RandNLA, Matrix Multiplication, Sampling, and Sketching**

I. Introduction

Matrix multiplication is a fundamental problem in linear algebra and machine learning. Most of the real-world data today can be written as matrices, and performing operations on the matrices reveals insights about the data. Be it neural networks or a simple linear regression algorithm, all of these techniques have one thing in common: matrix multiplication forms the core of these algorithms.

Multiplying two matrices of the size$n \times d$ usually takes around $O(nd^2)$ time; this is fine for matrices with smaller sizes but as the size increases, the quadratic factor becomes very bothersome

With Language Models and dense models quickly gaining popularity these days, optimization of matrix multiplication is a field of great importance and relevance in current context

The goal of this project is to explore sampling and sketching methods to approximate matrix multiplication and find a decent solution for a variety of data.

For our purposes we have generated four different types of synthetic matrices: ***UG*** *(Uniform Good: well conditioned with good leverage scores),* ***UB*** *(Uniform Bad: good leverage scores but poor condition number),* ***NG*** *(Nonuniform Good: well conditioned with bad leverage scores), and* ***NB*** *(Nonuniform Bad: bad leverage scores and poorly conditioned)* [1]

Randomized Numerical Linear Algebra is a field of mathematics that provides the readers with toolkits to approximate several matrix operations, Matrix multiplication being one of them

Randomized Numerical Linear Algebra (RandNLA) methods offer several key advantages compared to deterministic methods, such as

- Speed and efficiency, as sometimes they make samples or sketches of the actual data. i.e., they give us an approximate answer instead of calculating the original answer
- Reduced Communication Cost for distributed settings: RandNLA algorithms require fewer passes over the data as compared to traditional methods
- Relative Error Guarantees in the sense they provide answers that are good enough till a certain threshold
- RandNLA methods are well-suited for modern large-scale problems where traditional methods become infeasible due to memory and processing constraints.

These approaches work by creating compressed versions—often called *sketches*—of the original matrices. Instead of multiplying $\boldsymbol{A^T B}$ directly, we compute something like

$$\boldsymbol{\tilde{A}^T \tilde{B} \approx A^T B},$$

such that $\boldsymbol{\tilde{A}} \in \mathbb{R}^{k \times d} and \boldsymbol{\tilde{B}} \in \mathbb{R}^{k \times d}$ where $k << n$

the lecture notes [2] also talk more about them

The popular sketch-and-solve paradigm [3] is one such problem of linear algebra that we will be talking about here. This paradigm involves solving large-scale computation problems by reducing the dimensionality via sketches. These approaches are mainly divided into two parts:

- **Sketching-based approaches** : Sketching is a technique that involves getting a smaller, more compressed version of the original matrix, often by multipliying the original matrix with another matrix with certain properties (random or otherwise) [4]. Here, the goal is to compute $\boldsymbol{\tilde{A}^T \tilde{B} \approx A^T B}$, where $\tilde{A}$ and $\tilde{B}$ are compact "sketches" of A and B. Overtime, researchers have developed a lot of

sketches which give "good" results. We will explore some of them below but before we do that, it is important to understand how can we say that the results will always be good

To quantify the results for sketching-based approaches, researchers refer to a popular theoretical guarantee known as the Johnson-Lindenstrauss (JL) lemma.

[5] This lemma guarantees that if we project high-dimensional data onto a lower-dimensional subspace using a carefully chosen random matrix, we can preserve distances between vectors with high probability.

A random matrix $\boldsymbol{S} \in \mathbb{R}^{r\times d}$ forms a Johnson-Lindenstrauss transform with parameters $\varepsilon, \delta \in (0, 1/2)$ and a positive integer n, or an (ε, δ, n)-JLT for short, if with probability at least $1-\delta$, for any fixed set $V \subseteq \mathbb{R}^d$ with $|V| = n$, it holds that

$$(1-\varepsilon)\|\mathbf{v}\|_2^2 \leq \|\boldsymbol{S}\mathbf{v}\|_2^2 \leq (1+\varepsilon)\|\mathbf{v}\|_2^2$$

for all $\mathbf{v} \in V$.

The random matrix here also has special properties. Given below are some of the most prevalent matrices that are used to generate a sketch for JL lemma:

- Gaussian matrix: This matrix $\boldsymbol{S}$ is generated such that $\boldsymbol{S} \in R^{kn}$, where each entry$\boldsymbol{S_{ij}} \sim \mathcal{N}\left(0, \frac{1}{k}\right)$ meaning it is sampled independently from a normal distribution with mean 0 and variance $1/k$.
- [6] Count Sketch Matrix is a sparse, hashing-based transform that uses a random sign vector and a hash function to compress the matrix efficiently by choosing a hash value from the size of the sample for each column and then randomly flipping the signs of the column
- [6] PHD matrix is a structured random projection technique that combines a randomized Hadamard transform with uniform subsampling. For a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the transformation is defined as:

$$\tilde{\boldsymbol{A}} = \boldsymbol{PHDA}$$

where:

  - $\boldsymbol{D} \in \mathbb{R}^{m\times m}$ is a diagonal matrix with entries drawn independently from $\{-1, +1\}$.
  - $\boldsymbol{H} \in \mathbb{R}^{m\times m}$ is the normalized Hadamard matrix.
  - $\boldsymbol{P} \in \mathbb{R}^{k\times m}$ is a sampling matrix that selects k rows uniformly at random.

- **Sampling-based approaches** : Sampling-based approaches usually use some form of "scoring" measure to decide how important an entry/row/column of the matrix is. Based on the calculation of the importance of the entries of matrices, we have several different types of sampling techniques. One such technique is known as leverage score sampling.
- Leverage score sampling selects rows of a matrix with probabilities proportional to their statistical leverage scores.

For a matrix $\boldsymbol{A}$ with singular value decomposition (SVD) $\boldsymbol{A} = \boldsymbol{U\Sigma V^T}$, the leverage score of the i-th row is given by: [7]

$$h_{ii} = \|\boldsymbol{U(i,:)}\|_2^2.$$

Instead of taking the absolute values for leverage scores, their square root can also be taken and it is known as square of leverage score sampling

we also talk about priority and threshold sampling introduced here [8]

The main idea behind these methods is to select the rows that "contribute" the most towards the entire matrix and use them to approximate matrix multiplication. How do they go about selecting it and the subtle nuances are discussed below

Threshold Sampling: Let $\boldsymbol{A} \in \mathbb{R}^{n\times d}$ be a matrix. The frobenius norm for the matrix is given as the square sum of rows for the matrix

$$\|\boldsymbol{A}\|_F^2 = \sum_{i=1}^{n} \|\boldsymbol{A_i}\|_2^2,$$

The threshold here is calculated using sample size k and the frobenius norm which is then used to select the rows

$$\tau = \frac{k}{\|\boldsymbol{A}\|_F^2}.$$

For each row i, a random value $u_i \sim \mathcal{U}(0, 1)$. The row is selected if

$$u_i \leq \tau \|\boldsymbol{A_i}\|_2^2.$$

So essentially, threshold sampling selects the rows if the value of their hash function u_i is below the threshold

Priority Sampling: For priority sampling, instead of just using the threshold criteria directly, the rank r_i is calculated as follows:

$$r_i = \frac{u_i}{\|\boldsymbol{A_i}\|_2^2},$$

The k rows with the smallest values of r_i are retained, and the threshold τ is defined as the $(k+1)$-th smallest value among all r_i.

Approximate Product: To approximate $\boldsymbol{A^T B}$, both $\boldsymbol{A}$ and $\boldsymbol{B}$ are sampled independently, sharing the same random seed where $\boldsymbol{S_A}$ and $\boldsymbol{S_B}$ are the

sampled matrices and τ_A, τ_B are the corresponding thresholds.

$$\boldsymbol{A^T B} \approx \sum_{i \in S_A \cap S_B} \frac{\boldsymbol{A_i^T B_i}}{\min\Big(1,\ \|\boldsymbol{A_i}\|_2^2\,\tau_A,\ \|\boldsymbol{B_i}\|_2^2\,\tau_B\Big)},$$

These methods talk about how to select rows which represent the original matrix the best. However, an older and much simpler form of sampling called uniform sampling assigns the same "weight" to each row in the matrix and selects them on a random "chance" More formally, for a given matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and a desired sample size k, each row is selected uniformly at random with probability

$$p = \frac{k}{n}.$$

Thus, this sampling gives every row an "equal" chance of being selected.

II. Notations and Preliminaries

Throughout the course of this entire report, we will follow the following convention:

- Matrices are denoted in bold capital letters. E.g. $\boldsymbol{A} \in R^{n \times d}$ is a matrix
- Scalar values are denoted by small letters. E.g a is a scalar
- Sets are denoted by capital letters. E.g S is a set
- Samples or sketches of the original matrices are denoted by the letter $\boldsymbol{S}$
- By default, all vectors are column vectors unless specified otherwise
- Key norms:
  - Frobenius norm: $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} a_{ij}^2}$
  - Spectral norm: $\|\boldsymbol{A}\|_2 = \sigma_{\max}(\boldsymbol{A})$

III. Literature Review

[1] explores randomized matrix algorithms for solving large-scale ℓ_1 and ℓ_2 regression problems in distributed environments, with implementations evaluated on datasets of up to terabyte size using Apache Spark. The work addresses computational challenges inherent in parallel settings, particularly where communication costs are often the primary bottleneck, by utilizing randomization to either create smaller subproblems or construct efficient preconditioners.

Synthetic test matrices with controlled properties were generated to assess the algorithm's performance under various conditions. Two categories of matrices were created: (1) matrices with uniform leverage scores (UG and UB), constructed via SVD with adjustable condition numbers, and (2) matrices with nonuniform leverage scores (NG and NB), designed using a block structure to intentionally introduce leverage score imbalances. Both well-conditioned ($\kappa \approx 5$) and ill-conditioned ($\kappa \approx 10^6$) variants were produced, along with corresponding regression targets by perturbing the linear systems with controlled noise.

The experiments compared low-precision techniques (which directly solve randomly projected subproblems) with high-precision methods (which use randomization to precondition iterative solvers). Various embedding techniques were evaluated, including Gaussian projections, subsampled randomized Hadamard transforms (SRHT), and sampling based on approximate leverage scores. The findings indicated that no single method outperformed the others across all scenarios—performance was highly dependent on matrix properties and precision requirements. Gaussian projections demonstrated strong performance as preconditioners, while methods such as CountSketch (CW) offered computational efficiency at the potential cost of requiring larger embedding dimensions. Leverage score sampling was particularly effective when accurate approximations were available, especially for matrices with nonuniform leverage structures.

[8] investigates the approximation of large-scale matrix products $\boldsymbol{A}^\top \boldsymbol{B}$ via coordinated sampling methods—specifically priority sampling and threshold sampling—that exploit shared randomness to jointly select informative rows from both matrices. In priority sampling, rows are ranked and chosen based on hash-derived values proportional to their norms, whereas threshold sampling determines inclusion probabilities using global norm thresholds. Each approach produces compressed sketches $\boldsymbol{S}_A$ and $\boldsymbol{S}_B$ that retain the essential structural relationships of $\boldsymbol{A}$ and $\boldsymbol{B}$, with provable bounds on the Frobenius-norm approximation error. This framework contrasts with uniform sampling, which neglects row significance and often yields suboptimal results on sparse data.

Empirical evaluations on synthetic sparse matrices—varying sparsity levels from 10% to 80% nonzeros—demonstrated that both priority and threshold sampling substantially outperform classical Johnson–Lindenstrauss projections in terms of product approximation accuracy. In a downstream regression experiment using the IMDB movie-review dataset (represented as sparse TF–IDF feature matrices), priority sampling delivered superior solution accuracy compared to projection-based sketches at equivalent compression ratios, underscoring its practical effectiveness for real-world sparse-data analytics.

[2] presents scalable techniques for approximating the product $\boldsymbol{AB}$ by sampling a subset of columns from $\boldsymbol{A}$

and the corresponding rows from $\boldsymbol{B}$. A straightforward implementation forms reduced matrices $\boldsymbol{C}$ and $\boldsymbol{R}$ via uniform sampling over c column–row pairs.

The resulting approximation satisfies both expectation and high-probability bounds:

$$\|\boldsymbol{AB} - \boldsymbol{CR}\|_F \;\le\; \mathcal{O}\big(c^{-1/2}\big)\,\|\boldsymbol{A}\|_F\,\|\boldsymbol{B}\|_F,$$

where c denotes the number of samples. These guarantees rely on classical concentration tools:

- **Markov's inequality:** Offers a loose bound on tail probabilities from the expectation, e.g.,

$$\mathbb{P}\big(\|\boldsymbol{AB} - \boldsymbol{CR}\|_F \ge a\big) \;\le\; \frac{\mathbb{E}\big[\|\boldsymbol{AB} - \boldsymbol{CR}\|_F\big]}{a}.$$

- **Chernoff bounds:** Provide exponentially decaying bounds on deviations for sums of independent sampling errors, yielding probabilities that shrink as $\exp(-\Theta(c))$ with increasing c.

[7] proposes a fast randomized algorithm to estimate the statistical leverage scores of a tall matrix $\boldsymbol{A} \in \mathbb{R}^{n\times d}$ with $n \gg d$. The leverage score ℓ_i for the i-th row is defined as the squared ℓ_2-norm of that row in the matrix $\boldsymbol{U}$ containing the top d left singular vectors of $\boldsymbol{A}$:

$$\ell_i \;=\; \|\boldsymbol{U}_{(i)}\|_2^2.$$

Equivalently, ℓ_i is the i-th diagonal entry of the projection matrix onto the column space of $\boldsymbol{A}$:

$$\ell_i \;=\; (\boldsymbol{U}\boldsymbol{U}^\top)_{ii} \;=\; (\boldsymbol{P}_A)_{ii}.$$

[9], [5] examine practical realizations of the Johnson–Lindenstrauss (JL) lemma, which ensures that any set of n points in $\mathbb{R}^d$ can be embedded into $\mathbb{R}^k$ with

$$k \;=\; \mathcal{O}\big(\epsilon^{-2}\log n\big)$$

such that, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$,

$$\Pr\left[(1-\epsilon)\|\boldsymbol{x}-\boldsymbol{y}\|^2 \le \|\boldsymbol{\Phi x} - \boldsymbol{\Phi y}\|^2 \le (1+\epsilon)\|\boldsymbol{x}-\boldsymbol{y}\|^2\right] \ge 1-\delta.$$

Key variants include:

- **Dense Gaussian JL:** $\boldsymbol{\Phi} \in \mathbb{R}^{k\times d}$ with entries $\Phi_{ij} \sim \mathcal{N}(0, 1/k)$.
- **Sparse JL** [9]: Reduces the number of nonzeros per column to $\mathcal{O}(\epsilon^{-1}\log n)$, enabling faster matrix multiplication while retaining the JL guarantee.
- **Approximate Structured JL** [5]: Uses structured random matrices (e.g., truncated randomized Hadamard or CountSketch variants) to trade off slightly weaker distortion bounds for significant computational savings.

[10] investigates uniform random sampling as a simple strategy for approximating large matrices. In this approach, each row of the matrix is selected independently with equal probability, avoiding the need to compute statistical leverage scores—which can be as costly as solving the original problem. Although uniform sampling eliminates this preprocessing overhead, it treats all rows as equally informative and thus may overlook critical high-leverage rows. Under assumptions of row incoherence, however, uniform sampling can still provide provable bounds on the approximation error in both expectation and with high probability.

[4] describes low-rank approximation as a means of capturing the essential information of a matrix $\boldsymbol{A}$ using fewer dimensions. A key example is the CUR decomposition:

$$\boldsymbol{A} \;\approx\; \boldsymbol{C}\,\boldsymbol{U}\,\boldsymbol{R},$$

where:

- $\boldsymbol{C} \in \mathbb{R}^{m\times c}$ contains c actual columns of $\boldsymbol{A}$,
- $\boldsymbol{R} \in \mathbb{R}^{r\times n}$ contains r actual rows of $\boldsymbol{A}$,
- $\boldsymbol{U} \in \mathbb{R}^{c\times r}$ is a small linking matrix chosen to minimize the reconstruction error.

This factorization maintains interpretability—since the basis elements are drawn directly from $\boldsymbol{A}$—while achieving an approximation close to the optimal rank-k representation under standard norms.

IV. Experiment Setup

A. Data Generation

For this project, we are working with four different types of data generated synthetically, which closely follow [1]

- **UG (Uniform Good)**: These matrices are constructed so that they have a good condition number and good leverage scores. The condition number, defined as,

$$\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})},$$

tells us how sensitive the original matrix is to the errors. The UG matrix ($\mathbf{A}_{ug}$), which has uniform leverage scores (i.e., no significant outliers and the data is fairly closely distributed) along with a good condition number which means that the system is stable

Algorithm 1 UG(m, n, κ)

Input: m (rows), n (columns), condition number κ
Output: Matrix $\mathbf{A}_{ug} \in \mathbb{R}^{m \times n}$
Generate an orthonormal matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ (e.g., via QR decomposition of a random matrix)
Generate an orthonormal matrix $\mathbf{V} \in \mathbb{R}^{n \times n}$
Define $\mathbf{S} = \text{diag}(s_1, s_2, \ldots, s_n)$, where s_i are linearly spaced from 1 to $1/\kappa$
Set $\mathbf{A}_{ug} \leftarrow \mathbf{U}\,\mathbf{S}\,\mathbf{V}^\top$
return $\mathbf{A}_{ug}$

- **UB (Uniform Bad)**: These are the matrices ($\mathbf{A}_{ub}$) that have uniform leverage scores, but they have an ill-conditioned number.

Algorithm 2 UB(m, n, κ)

Input: m, n, κ
Output: Matrix $\mathbf{A}_{ub} \in \mathbb{R}^{m \times n}$
Generate orthonormal matrices $\mathbf{U} \in \mathbb{R}^{m \times n}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ (as above)
Define the diagonal matrix $\mathbf{S} = \text{diag}(s_1, s_2, \ldots, s_n)$, where each s_i decays exponentially from 1 to $1/\kappa$
Set $\mathbf{A}_{ub} \leftarrow \mathbf{U}\,\mathbf{S}\,\mathbf{V}^\top$
return $\mathbf{A}_{ub}$

- **NG (Nonuniform Good)**: In these matrices ($\mathbf{A}_{ng}$), the leverage scores vary significantly, but the matrix remains well-conditioned. Thus, a few rows are more influential than others, yet the ratio of the largest to smallest singular values remains controlled.

Algorithm 3 NG(m, n)

Input: m, n
Output: Matrix $\mathbf{A}_{ng} \in \mathbb{R}^{m \times n}$
Let $p \leftarrow n/2$ and $q \leftarrow m - p$
Generate a Gaussian block $\mathbf{G} \in \mathbb{R}^{q \times p}$ with entries $G_{ij} \sim \mathcal{N}(0, 1)$
Generate a noise block $\mathbf{N} \in \mathbb{R}^{q \times p}$ with entries on the order of 10^{-4}
Define the identity matrix $\mathbf{I} \in \mathbb{R}^{p \times p}$
Define Top block $\mathbf{T} \leftarrow \begin{bmatrix} \mathbf{G} \mid \mathbf{N} \end{bmatrix}$
Define Bottom block $\mathbf{B} \leftarrow \begin{bmatrix} \mathbf{0}_{p \times p} \mid \mathbf{I} \end{bmatrix}$
Stack vertically to obtain

$$\mathbf{A}_{ng} \leftarrow \begin{bmatrix} \mathbf{T} \\ \mathbf{B} \end{bmatrix}.$$

return $\mathbf{A}_{ng}$

- **NB (Nonuniform Bad)**: These matrices ($\mathbf{A}_{nb}$) exhibit both nonuniform leverage scores and a poor condition number. They are generated to reflect the most challenging scenario, where a small set of rows dominate the structure and the matrix is ill-conditioned.

Algorithm 4 NB(m, n, κ)

Input: m, n, κ
Output: Matrix $\mathbf{A}_{nb} \in \mathbb{R}^{m \times n}$
Let $p \leftarrow n/2$ and $q \leftarrow m - p$
Generate a Gaussian block $\mathbf{G} \in \mathbb{R}^{q \times p}$ with entries $G_{ij} \sim \mathcal{N}(0, 1)$
Generate a tiny noise block $\mathbf{N} \in \mathbb{R}^{q \times p}$ with entries on the order of 10^{-8}
Define the identity matrix $\mathbf{I} \in \mathbb{R}^{p \times p}$
Define Top block $\mathbf{T} \leftarrow \begin{bmatrix} \kappa\,\mathbf{G} \mid \mathbf{N} \end{bmatrix}$
Define Bottom block $\mathbf{B} \leftarrow \begin{bmatrix} \mathbf{0}_{p \times p} \mid \mathbf{I} \end{bmatrix}$
Stack vertically to obtain

$$\mathbf{A}_{nb} \leftarrow \begin{bmatrix} \mathbf{T} \\ \mathbf{B} \end{bmatrix}.$$

return $\mathbf{A}_{nb}$

Using these definitions, we generate paired matrices (labeled $\mathbf{A}$ and $\mathbf{B}$) for each type, all of size $10{,}000 \times 100$, across a range of sample sizes (250, 500, 750, 1000, 1250, 1500, 1750, 2000).

Sparse Matrices: Sparse matrices are the matrices which have mostly zero values. i.e. a very small chunk of these matrices contains useful information. These matrices are very prevelant in domains such as social networks and NLP. Here we have generated this matrix by filling it up wiht zeroes, some random non zero entries drawn from the normal distribution and a certain percentage of the matrix has outliers

Algorithm 5 GenerateSparseMatrices(m, n, ρ, η)

Input: Dimensions m, n, sparsity ρ, outlier ratio η
Output: Sparse matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$
Calculate non-zero count $t \leftarrow \lfloor (1 - \rho)mn \rfloor$
Generate values:
  $\mathbf{A}$: t samples from $\mathcal{N}(0, 1)$
  $\mathbf{B}$: t samples from $\mathcal{U}(-1, 1)$
Inject ηt outliers in each matrix
Choose distinct non-zero positions for $\mathbf{A}$ and $\mathbf{B}$
Map positions to $(row, column)$ coordinates
Build $\mathbf{A}, \mathbf{B}$ in COO format
return $\mathbf{A}, \mathbf{B}$

B. Algorithm design

For the scope of this project, we have picked several sampling and sketching techniques and applied them to approximate matrix multiplication across regression setting and on real world dataset. The methods picked are a combination of traditional methods such as uniform sampling, Gaussian matrix for JL lemma or Sampling based on the leverage scores. We have also picked newer methods of sampling such as count sketch matrix, Priority sampling, Threshold sampling, Fast JL transform etc. The implementation of these algortihms is described in greater detail in the parts below

Method 1: Uniform Sampling
Idea: Rows of matrices $\mathbf{A}$ and $\mathbf{B}$ are sampled uniformly at random i.e. each row has equal chance of being selected at random

```
For t in {1, ..., s}:
   Select index i uniformly from {1, ...,
       n}
   C[:, t] = sqrt(n/s) * A[i, :]
   R[t, :] = sqrt(n/s) * B[i, :]
Return: Approximate = C @ R
```

Method 2: Priority Sampling
Idea: The idea behind priority sampling is that we use a random uniform hash function to select rows with the highest row norms. The idea behind this is that the rows with the highest row norms have the most to contribute to the matrix. When approximating a matrix product, we only select the rows which have common indices across both the matrices

```
For i in {1, ..., n}:
   r[i] = hash[i] / ||A[i]||^2
Select rows with the smallest s values of
    r
Return: Sketched matrix of rows and their
    indices
```

Method 3: Threshold Sampling
Idea: A global threshold τ is computed based on the Frobenius norm of $\mathbf{A}$ and that is used as a selection measure to choose rows. Basically, if the norm of the row exceeds the threshold, we pick it

```
Compute tau = s / ||A||_F^2
For i in {1, ..., n}:
   If hash[i] <= tau * ||A[i]||^2:
      Include row i
Return: Sketched matrix of selected rows &
     indices
```

Method 4: Approx. Multiplication Using Sketches
Idea: Helper function, used to approximate matrix multiplication by selecting the common indices

```
$common\_indices = indices\_A \cap indices
    \_B$
Initialize result = zero matrix
For each idx in common_indices:
   rowA = sketchA[idx]
   rowB = sketchB[idx]
   normA_sq = ||rowA||^2
   normB_sq = ||rowB||^2
   denom = min(1, normA_sq * tauA,
       normB_sq * tauB)
   result += outer(rowA, rowB) / denom
Return: result
```

Method 5: Leverage Score Sampling
Idea: Rows from A and B are sampled based on their leverage scores. The idea is that, higher leverage score for a row, means higher contribution to the matrix

```
Input: A (m x n), B (n x p), sample size s
Compute SVD: A = U  V^T, B = Ub b Vb^T
For each row i in U:
   L_A[i] = sum_j U[i,j]^2
For each row i in Ub:
   L_B[i] = sum_j Ub[i,j]^2
Select top_A, top_B (largest s in L_A, L_B
    )
SampledA = A[top_A, :]
SampledB = B[top_B, :]
scaledA = SampledA * sqrt(m/s)
scaledB = SampledB * sqrt(p/s)
Return: scaledA, scaledB
```

Method 6: Sqrt Leverage Score Sampling
Idea: Similar to leverage score sampling, but it takes the square root of the leverage scores first, used to flatten extreme values

```
Input: A (m x n), B (n x p), sample size s
Compute SVD: A = U  V^T, B = Ub b Vb^T
For each row i in U:
   L_A[i] = sqrt(sum_j U[i,j]^2)
For each row i in Ub:
   L_B[i] = sqrt(sum_j Ub[i,j]^2)
Select top_A, top_B (largest s in L_A, L_B
    )
SampledA = A[top_A, :]
SampledB = B[top_B, :]
scaledA = SampledA * sqrt(m/s)
scaledB = SampledB * sqrt(p/s)
Return: scaledA, scaledB
```

Method 8: PHD JL (Hadamard-Based)
Idea: A fast JL transform is achieved using a randomized Hadamard transform followed by subsampling. Matrices are optionally padded so that the row dimension is a power of 2, and random sign flipping is applied.

```
Input: A (m x n), B (m x p), sketch size s
M = next power of 2 >= m
Pad A, B with zeros if needed
Generate d_vec in {-1, 1}^M
A_scaled = A_padded * d_vec
B_scaled = B_padded * d_vec
A_transformed = FWHT(A_scaled) / sqrt(M)
B_transformed = FWHT(B_scaled) / sqrt(M)
Select s rows randomly:
   A_proj = A_transformed[selected_rows]
   B_proj = B_transformed[selected_rows]
Approx_product = A_proj^T @ B_proj
Return: Approx_product
```

Method 9: Count-Sketch JL Transform

Idea: A sparse projection is implemented by hashing the m rows into s buckets and flipping their signs. The contributions are accumulated into sketch matrices, which are then used to project the matrices into lower dimensions.

```
Input: A (m x n), B (m x p), sketch size s
Generate hash h: {1,...,m} -> {0,...,s-1}
Generate sigma in {-1,1}^m
Initialize A_sketch in R^(n x s), B_sketch
     in R^(p x s) as zero
For i in {1,...,m}:
   bucket = h(i)
   A_sketch[:, bucket] += sigma[i] * A[i,
       :]^T
   B_sketch[:, bucket] += sigma[i] * B[i,
       :]^T
Approx_product = A_sketch @ B_sketch^T
Return: Approx_product
```

V. Experiments

A. Regression on IMDB dataset

The popular IMDB dataset, used for sentiment analysis is the perfect use case for the experiment here as we have the matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ generated using TF-IDF vectorization which gives a sparse matrix, where:

- $n = 10{,}000$ reviews
- $d \in \{128, 256, 512\}$ are the top n features that we will analyze for
- Target vector $b \in \mathbb{R}^n$

The regression task solves:

$$\min_{x \in \mathbb{R}^d} \|\mathbf{A}x - b\|_2$$

using approximations of $\mathbf{A}$ and b generated using the sketching and sampling algorithms mentioned in the algortihms section.

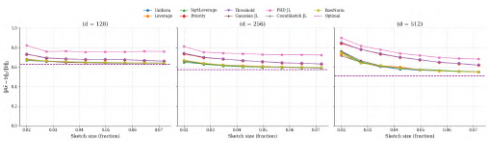

Figure 1: Plots for relative error comparison of different sketching and sampling techniques to approximate regression on IMDB dataset for top n features, in the increasing order

As it is shown in the figure, there are methods such as priority sampling, threshold sampling, count sketch and row-norm sampling which perform better on sparse data. Which again goes in-line with the fact that when we have data which is not distributed uniformly the uniform methods are less likely to work.

B. Matrix multiplication across varying sparsity

We also generated matrices $\mathbf{A} \in R^{1000 \times 1000}$ and $\mathbf{B} \in R^{1000 \times 1000}$ with 10 percent, 40 percent and 80 percent sparsity for the sample sizes 50, 100, 200, 400 for 5 iterations. Taking these matrices, we approximated the matrix product $\mathbf{A}^\top \mathbf{B}$ using the aforementioned algorithms.

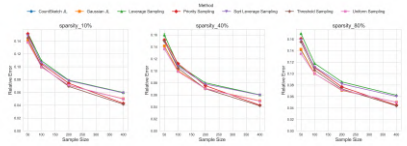

Figure 2: Relative error for matrix multiplication across matrices of varying sparsity for different sketch sizes

Looking at the figure, it is visible that some methods are naturally performing better even across the sparser matrices as compared to data independent methods of sketching and sampling.

C. Matrix multiplication across different types of matrices

For our experiment, we also generated four different types of matrices such as NG, NB, UG and UB respectively. Following up on the work at [1] we implemented newer methods along with the traditional methods mentioned in the paper.

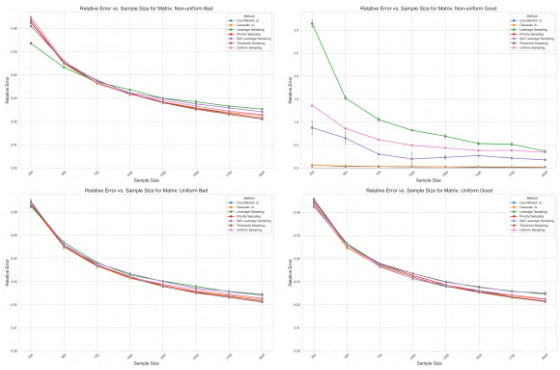

Figure 3: Comparing relative errors across varying sample sizes for different matrix types: (top-left) Non-uniform bad condition, (top-right) Non-uniform good condition, (bottom-left) Uniform bad condition, (bottom-right) Uniform good condition.

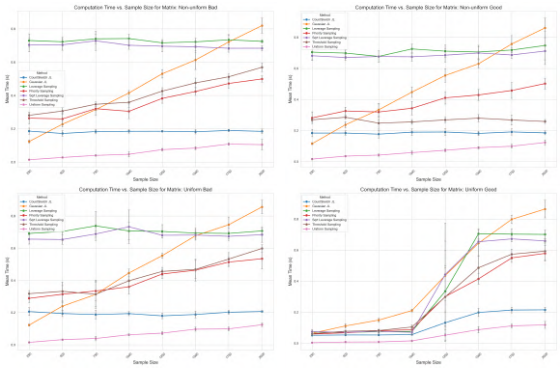

Figure 4: Comparing time taken to approximate matrix multiplication across varying sample sizes: (top-left) Non-uniform bad condition, (top-right) Non-uniform good condition, (bottom-left) Uniform bad condition, (bottom-right) Uniform good condition.

D. Approximating Linear regression

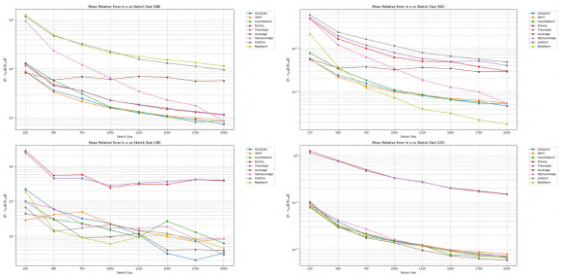

Figure 5: Comparing relative error for approximating linear regression across varying sample sizes: (top-left) Non-uniform bad condition, (top-right) Non-uniform good condition, (bottom-left) Uniform bad condition, (bottom-right) Uniform good condition.

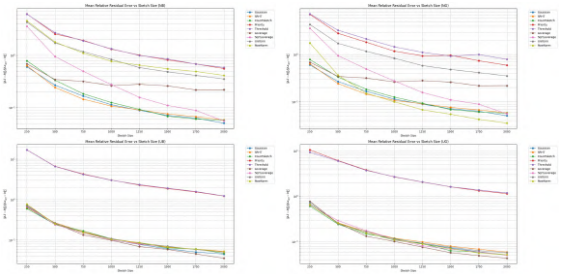

Figure 6: Comparing residual error for approximating linear regression across varying sample sizes: (top-left) Non-uniform bad condition, (top-right) Non-uniform good condition, (bottom-left) Uniform bad condition, (bottom-right) Uniform good condition.

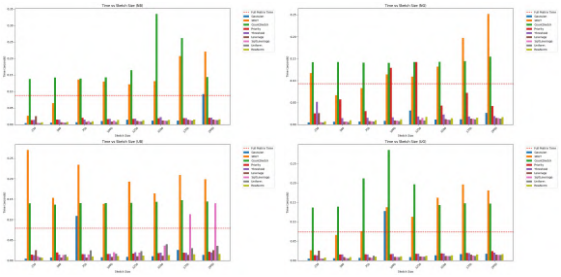

Figure 7: Comparing time taken to approximate linear regression across varying sample sizes: (top-left) Non-uniform bad condition, (top-right) Non-uniform good condition, (bottom-left) Uniform bad condition, (bottom-right) Uniform good condition.

VI. Results

Table I: Results for 10% Sparsity Matrix

Method	Relative Error	Time (s)
CountSketch JL	0.049 684	0.039 572
Gaussian JL	0.050 069	0.048 942
Leverage Sampling	0.059 799	1.057 340
Priority Sampling	0.042 880	3.133 510
Sqrt Leverage Sampling	0.059 200	1.070 900
Threshold Sampling	0.041 486	3.174 380
Uniform Sampling	0.049 640	0.007 936

Table II: Results for 40% Sparsity Matrix

Method	Relative Error	Time (s)
CountSketch JL	0.049 558	0.062 431
Gaussian JL	0.050 035	0.065 496
Leverage Sampling	0.060 113	1.952 850
Priority Sampling	0.042 768	4.599 940
Sqrt Leverage Sampling	0.059 754	1.811 420
Threshold Sampling	0.041 765	4.719 640
Uniform Sampling	0.049 385	0.023 827

Table III: Results for 80% Sparsity Matrix

Method	Relative Error	Time (s)
CountSketch JL	0.049 874	0.060 071
Gaussian JL	0.049 985	0.070 773
Leverage Sampling	0.062 235	1.942 250
Priority Sampling	0.044 697	4.322 670
Sqrt Leverage Sampling	0.060 196	1.884 160
Threshold Sampling	0.043 686	4.360 450
Uniform Sampling	0.049 776	0.030 381

Table IV: Results for Uniform Good Matrix

Method	Relative Error	Time (s)
Threshold Sampling	0.021 110	0.163 708
Priority Sampling	0.021 355	0.155 558
CountSketch JL	0.022 412	0.095 340
Uniform Sampling	0.022 419	0.038 821
Gaussian JL	0.022 480	0.463 576
Leverage Sampling	0.024 677	0.218 094
Sqrt Leverage Sampling	0.024 699	0.080 779

Table V: Results for Uniform Bad Matrix

Method	Relative Error	Time (s)
Threshold Sampling	0.021 157	0.181 009
Priority Sampling	0.021 538	0.150 317
Gaussian JL	0.022 214	0.504 210
CountSketch JL	0.022 381	0.091 798
Uniform Sampling	0.022 831	0.035 862
Leverage Sampling	0.024 093	0.177 088
Sqrt Leverage Sampling	0.024 430	0.098 007 9

Table VI: Results for Non-uniform Good Matrix

Method	Relative Error	Time (s)
Priority Sampling	0.000 000	0.160 147
Threshold Sampling	0.000 000	0.099 757
CountSketch JL	0.005 657	0.095 713
Gaussian JL	0.022 669	0.575 134
Sqrt Leverage Sampling	0.178 564	0.108 404
Uniform Sampling	0.345 311	0.043 445
Leverage Sampling	0.367 438	0.139 248

Table VII: Results for Non-uniform Bad Matrix

Method	Relative Error	Time (s)
Threshold Sampling	0.021 016	0.182 439
Priority Sampling	0.021 466	0.155 563
Gaussian JL	0.022 424	0.530 467
CountSketch JL	0.022 530	0.097 074
Uniform Sampling	0.023 017	0.038 171
Sqrt Leverage Sampling	0.024 170	0.098 221
Leverage Sampling	0.025 329	0.142 600

VII. Conclusion

The aim of this project was to explore different algortihms for approximating matrix multiplication. We used real and synthetic dataset for the same. We compared traditional and novel algorithms to see how they compare against each other and found that some methods, such as importance sampling, are better for sparse data, and for data which is uniform without significant outliers, data independent sampling methods such as uniform sampling are a better choice.

VIII. Future work

Through the course of our experimentation, we found that the PHD matrix for projection is not living up to the theoretical and empirical guarantees. We plan to dig deeper on the same during our major and apply the same to real-world datasets on complex architectures in the future.

Contributions

The individual contributions to this work are outlined as follows :

- **Drafting & Final Editing of Report** **SM**
- **Feedback** **DP**
- **Matrix Multiplication Experiments** **SM**
  approximation of matrix multiplication for synthetic and sparse data
- **IMDB Dataset Analysis** **SM**
  Implementation and validation of experiments on the IMDB dataset.
- **Linear Regression on Synthetic Data** **DP**
  solving the least squares problem using sketch and solve paradigm for synthetic data.

REFERENCES

[1] J. Yang, X. Meng, and M. W. Mahoney, "Implementing randomized matrix algorithms in parallel and distributed environments," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 58–92, 2015.

[2] M. W. Mahoney *et al.*, "Randomized algorithms for matrices and data," *Foundations and Trends® in Machine Learning*, vol. 3, no. 2, pp. 123–224, 2011.

[3] A. Lavaee, "Sketch'n solve: An efficient python package for large-scale least squares using randomized numerical linear algebra," *arXiv preprint arXiv:2409.14309*, 2024.

[4] D. P. Woodruff *et al.*, "Sketching as a tool for numerical linear algebra," *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.

[5] A. Sobczyk and M. Luisier, "Approximate euclidean lengths and distances beyond johnson-lindenstrauss," *Advances in Neural Information Processing Systems*, vol. 35, pp. 19 357–19 369, 2022.

[6] K. L. Clarkson and D. P. Woodruff, "Low-rank approximation and regression in input sparsity time," *Journal of the ACM (JACM)*, vol. 63, no. 6, pp. 1–45, 2017.

[7] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff, "Fast approximation of matrix coherence and statistical leverage," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3475–3506, 2012.

[8] M. Daliri, J. Freire, D. Li, and C. Musco, "Matrix product sketching via coordinated sampling," *arXiv preprint arXiv:2501.17836*, 2025.

[9] D. M. Kane and J. Nelson, "Sparser johnson-lindenstrauss transforms," *Journal of the ACM (JACM)*, vol. 61, no. 1, pp. 1–23, 2014.

[10] M. B. Cohen, Y. T. Lee, C. Musco, C. Musco, R. Peng, and A. Sidford, "Uniform sampling for matrix approximation," in *Proceedings of the 2015 conference on innovations in theoretical computer science*, 2015, pp. 181–190.