НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

БПМИ-191: Абрамов Арсений

Multiplying Matrices without Multiplying

Москва, 2022

Введение

- Матричное умножение лежит в основе многих алгоритмов и возникает почти всюду в ТИ и МО.
- Честное матричное умножение имеет кубическую сложность (плохо).
- Approximating Matrices Multiplication (AMM) ищет баланс между точностью и эффективностью по времени / памяти.
- Активно разрабатываются новые методы AMM, обладающие ещё большей эффективностью. Много методов AMM работают при допущениях.

Традиционный Подход

Линейный оператор

$$\mathbf{A} \in R^{N\times D},\ \mathbf{B} \in R^{D\times M},\ \mathrm{N} >> \mathrm{D} \geq \mathrm{M}$$

Data

$$V_A, V_B \in R^{D\times d},\ \mathrm{d} << \mathrm{D}$$

sparse

$$\boldsymbol{AB} \sim (\boldsymbol{AV_A})(\boldsymbol{V_B^T}\mathbf{B})$$

(Если коротко, то проблема сводится к честному произведению в меньшей размерности через линейные преобразования A и B.)

НЕТрадиционный Подход

Assumptions:

Given a matrix $A \in \mathbb{R}^{m \times n}$, let us define:

ядро

- A is a **fat matrix** if $m \leq n$ and $\text{null}(A^T) = \{0\}$
- A is a **tall matrix** is $m \geq n$ and $\text{range}(A) = \mathbb{R}^n$

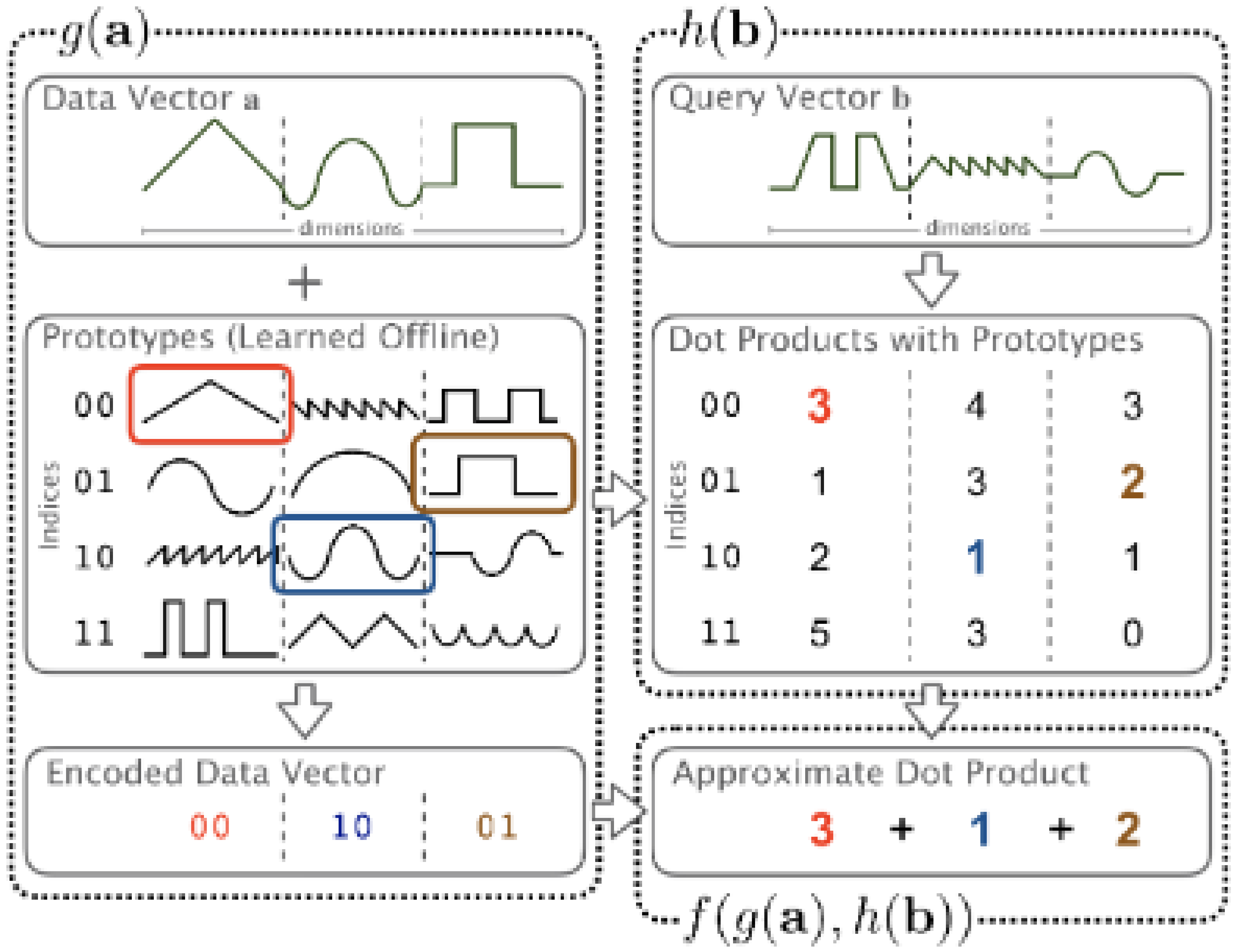
образ

- Matrices are tall
- Matrices are relatively dense
- Resident in a single machine's memory
- **Exists a training set Ã.**

MADDNESS employs a nonlinear preprocessing function and reduces the problem to table lookups.

If B is *known ahead of time,* MADDNESS **requires no multiply-add operations.**

Product Quantization



1. **Prototype Learning** - In an initial, offline training phase, cluster the rows of $\boldsymbol{A}$ (or a training set $\tilde{\boldsymbol{A}}$) using K-means to create prototypes. A separate K-means is run in each of C disjoint subspaces to produce C sets of K prototypes.
2. **Encoding Function**, $g(\boldsymbol{a})$ - Determine the most similar prototype to $\boldsymbol{a}$ in each subspace. Store these assignments as integer indices using $C \log_2(K)$ bits.
3. **Table Construction**, $h(\boldsymbol{B})$ - Precompute the dot products between $\boldsymbol{b}$ and each prototype in each subspace. Store these partial dot products in C lookup tables of size K.
4. **Aggregation**, $f(\cdot,\cdot)$ - Use the indices and tables to *lookup* the estimated partial $\boldsymbol{a}^\top \boldsymbol{b}$ in each subspace, then sum the results across all C subspaces.

НЕТрадиционный Подход, Сам Метод

Hash function family

Algorithm 1 MaddnessHash

```
Input: vector x, split indices j^1, ..., j^4, split thresholds v^1, ..., v^4
i ← 1                                  // node index within level of tree
for t ← 1 to 4 do
    v ← v_i^t        // lookup split threshold for node i at level t
    b ← x_{j^t} ≥ v ? 1 : 0                 // above split threshold?
    i ← 2i − 1 + b                    // assign to left or right child
end for
return i
```

Learning the hash function parameters

Algorithm 2 Adding The Next Level to the Hashing Tree

Input: buckets $\mathcal{B}_1^{t-1}, \ldots, \mathcal{B}_{2^{t-1}}^{t-1}$, training matrix $\tilde{A}$
// greedily choose next split index and thresholds
$\hat{\mathcal{J}} \leftarrow \texttt{heuristic\_select\_idxs}(\mathcal{B}_1^{t-1}, \ldots, \mathcal{B}_{2^{t-1}}^{t-1})$
$l^{min}, j^{min}, \boldsymbol{v}^{min} \leftarrow \infty$, NaN, NaN
for $j \in \hat{\mathcal{J}}$ **do**
  $l \leftarrow 0$ // initialize loss for this index to 0
  $\boldsymbol{v} \leftarrow [\,]$ // empty list of split thresholds
  for $i \leftarrow 1$ **to** 2^{t-1} **do**
    $v_i, l_i \leftarrow \texttt{optimal\_split\_threshold}(j, \mathcal{B}_i^{t-1})$
    $\texttt{append}(\boldsymbol{v}, v_i)$ // append threshold for bucket i
    $l \leftarrow l + l_i$ // accumulate loss from bucket i
  end for
  if $l < l^{min}$ **then**
    $l^{min} \leftarrow l, j^{min} \leftarrow j, \boldsymbol{v}^{min} \leftarrow \boldsymbol{v}$ // new best split
  end if
end for
// create new buckets using chosen split
$\boldsymbol{\mathcal{B}} \leftarrow [\,]$
for $i \leftarrow 1$ **to** 2^{t-1} **do**
  $\mathcal{B}_{below}, \mathcal{B}_{above} \leftarrow \texttt{apply\_split}(v_i^{min}, \mathcal{B}_i^{t-1})$
  $\texttt{append}(\boldsymbol{\mathcal{B}}, \mathcal{B}_{below})$
  $\texttt{append}(\boldsymbol{\mathcal{B}}, \mathcal{B}_{above})$
end for
return $\boldsymbol{\mathcal{B}}, l^{min}, j^{min}, v^{min}$

$$\mathcal{L}(j, \mathcal{B}) \triangleq \sum_{\boldsymbol{x} \in \mathcal{B}} \left(x_j - \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}' \in \mathcal{B}} x'_j \right)^2$$

$$\mathcal{L}(\mathcal{B}) \triangleq \sum_j \mathcal{L}(j, \mathcal{B}).$$

Optimizing prototypes

$$\boldsymbol{P} \triangleq (\boldsymbol{G}^\top \boldsymbol{G} + \lambda \boldsymbol{I})^{-1} \boldsymbol{G}^\top \tilde{\boldsymbol{A}}.$$

$\boldsymbol{P} \in \mathbb{R}^{KC \times D}$

Param = 1

$\tilde{\boldsymbol{A}} \approx \bar{\boldsymbol{G}} \boldsymbol{P}$

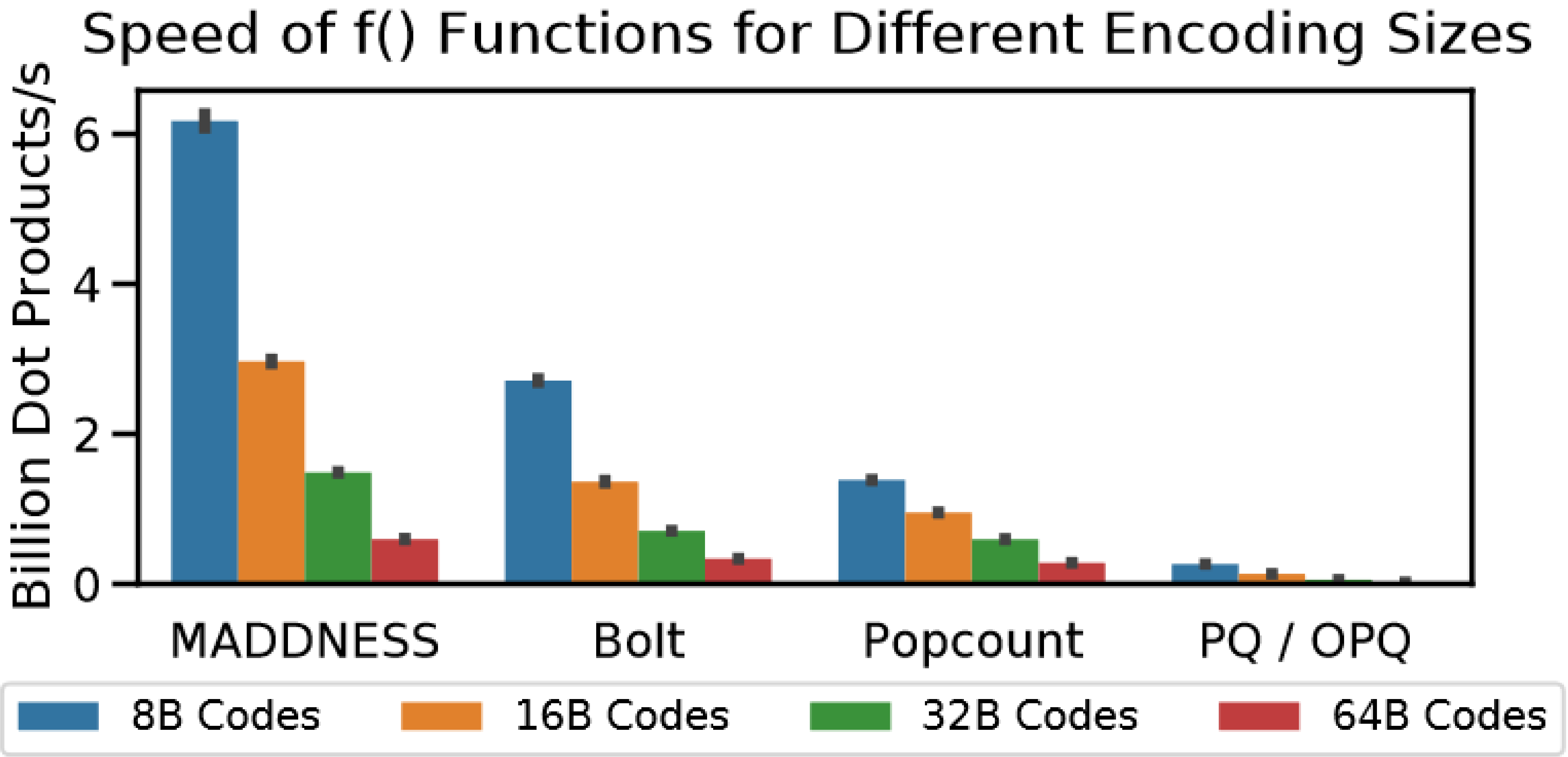
Fast 8-bit aggregation

Let $\boldsymbol{T} \in \mathbb{R}^{M \times C \times K}$ be the tensor of lookup tables for all M columns of $\boldsymbol{B}$. Given the encodings $\boldsymbol{G}$, the function $f(\cdot, \cdot)$ is defined as

$$f(g(\boldsymbol{A}), h(\boldsymbol{B}))_{n,m} \triangleq \sum_{c=1}^{C} \boldsymbol{T}_{m,c,k}, \;\; k = g^{(c)}(\boldsymbol{a}_n). \qquad (9)$$

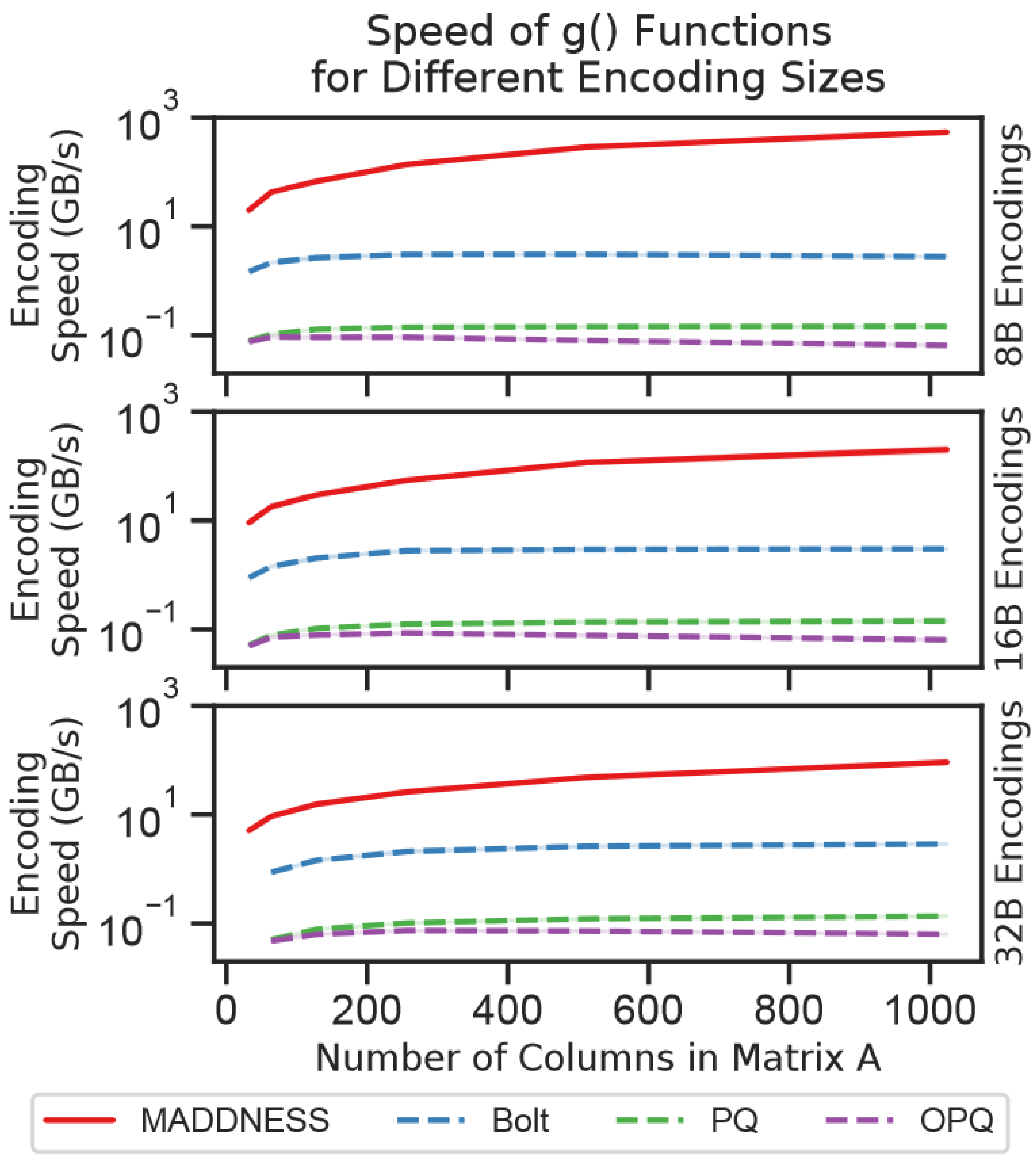
~~Addition~~ $\longrightarrow$ Averaging

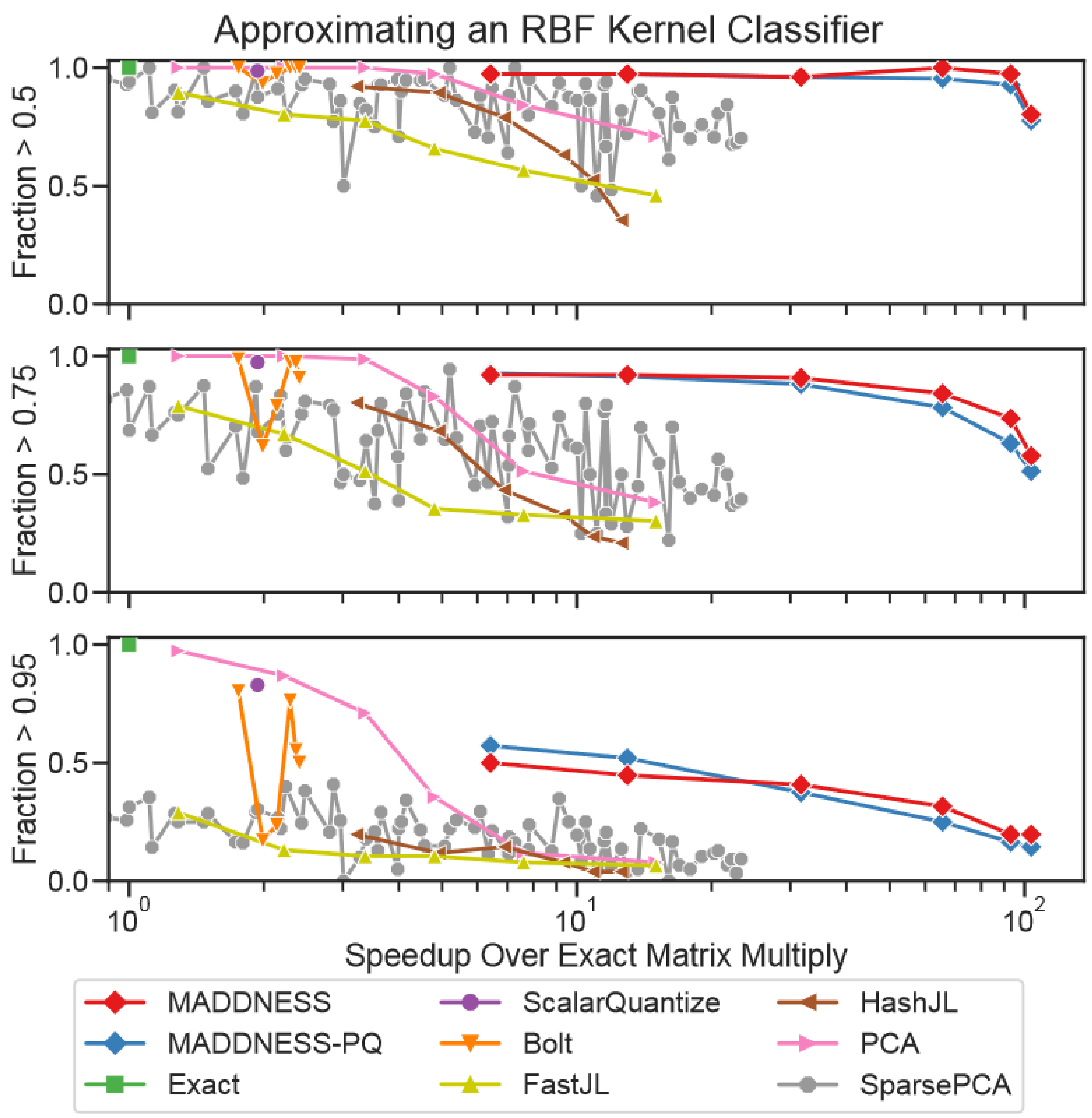
Я СКОРОСТЬ


Speed of f() Functions for Different Encoding Sizes
Billion Dot Products/s
0
2
4
6
MADDNESS
Bolt
Popcount
PQ / OPQ
8B Codes
16B Codes
32B Codes
64B Codes

Я СКОРОСТЬ

БПМИ-191, Абрамов Арсений





Наглядное сравнение подходов

$$AB \approx (AV_A)(V_B^\top B).$$

Learning Hash Function

САМОЕ ГЛАВНОЕ

https://smarturl.it/Maddness

ИСТОЧНИКИ

(Источник)

ИСТОЧНИК

Источник

- *Davis Blalock, John Guttag:* **Multiplying Matrices Without Multiplying**
  URL: https://arxiv.org/pdf/2106.10860.pdf

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ