MnnFast : A Fast and Scalable System Architecture for Memory-Augmented Neural Networks

Hanhwi Jang

46th International Symposium on Computer Architecture (ISCA), 2019

Presenter: Jaeyoon Lee

http://esoc.hanyang.ac.kr/people/<userlink>

March 26, 2020

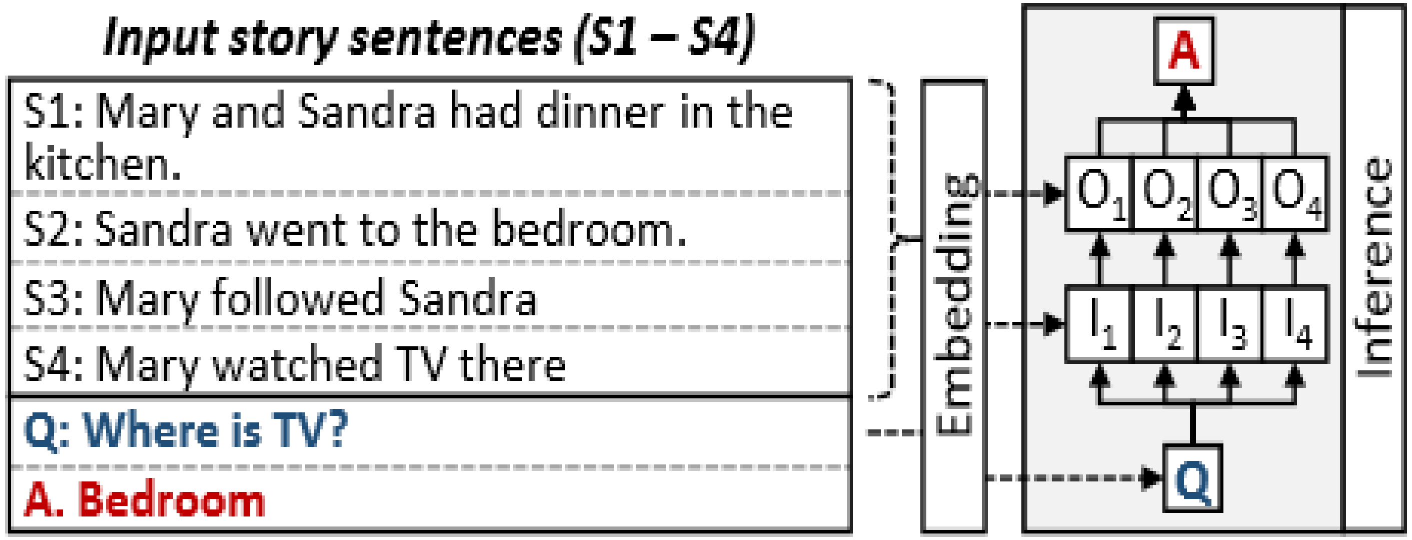
Neural Network Acceleration Study Season #2

Contents of presentation

- **Introduction**
  - Memory network(MemNNs)
  - Bag-of-Words(BoW) models
  - Computational steps of MemNN
- **Motivation**
  - Three major performance problems of MemNN
- **MnnFast**
  - Three ways to overcome performance problems of MemNN
- **Implementation & Evaluation**
- **Conclusion**

Introduction-Memory Network

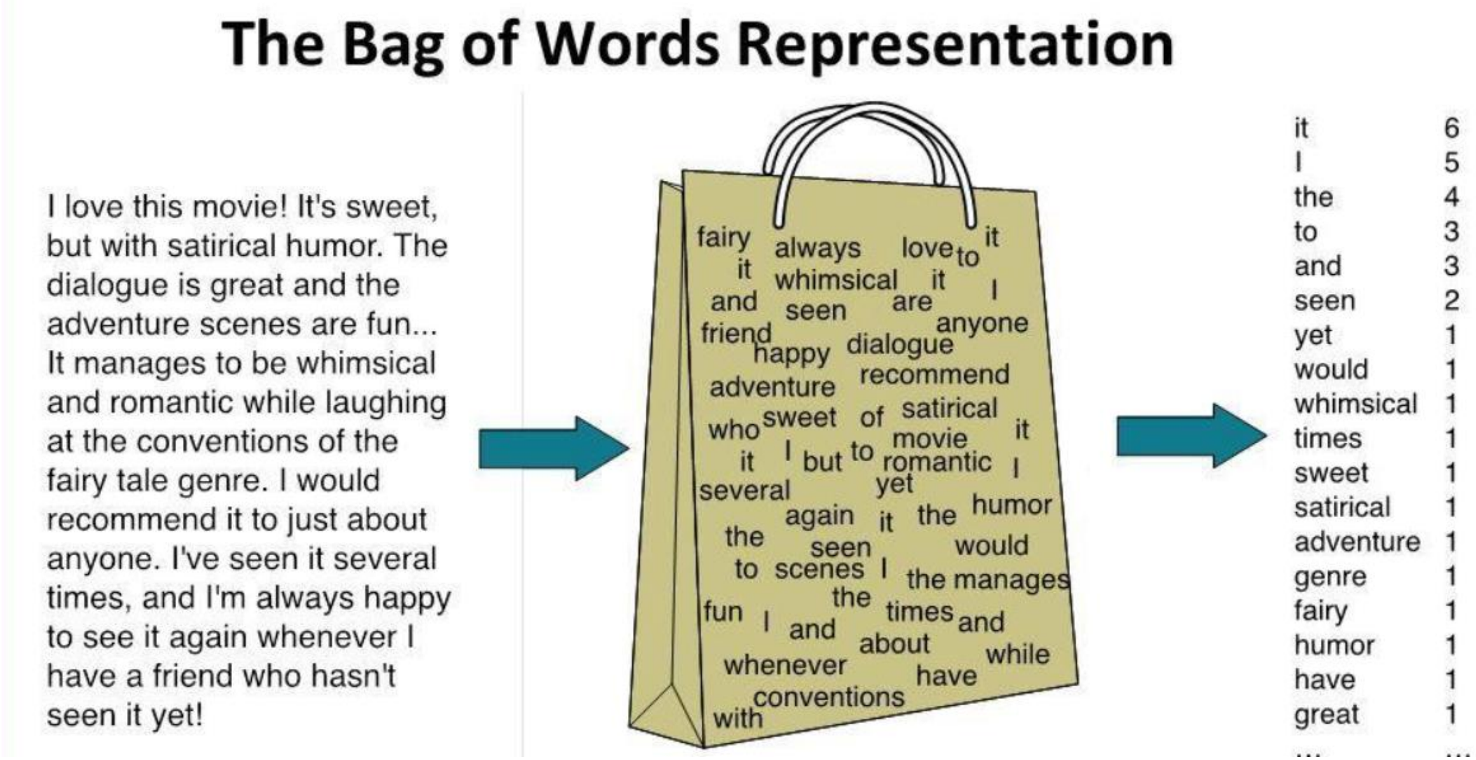
- **Context-aware information processing model, MemNNs.**
  - Similar to the human's working memory
  - Values are stored into **memory** components



(a) Example scenario (b) Memory network

Introduction-BoW models

- Bag-of-Words model enable text data to be represented in the form which can be processed by ML algorithms



Introduction-Word Embedding

- Word Embedding converts words into meaningful vectors

- After **BoW model**, sentence become ($1 \times V$) vector.

V : size of dictionary

- **Embedding operation** converts sentence vector ($1 \times V$) into internal states($1 \times ed$)

ed : embedding dimension

Introduction-Computational steps of MemNN

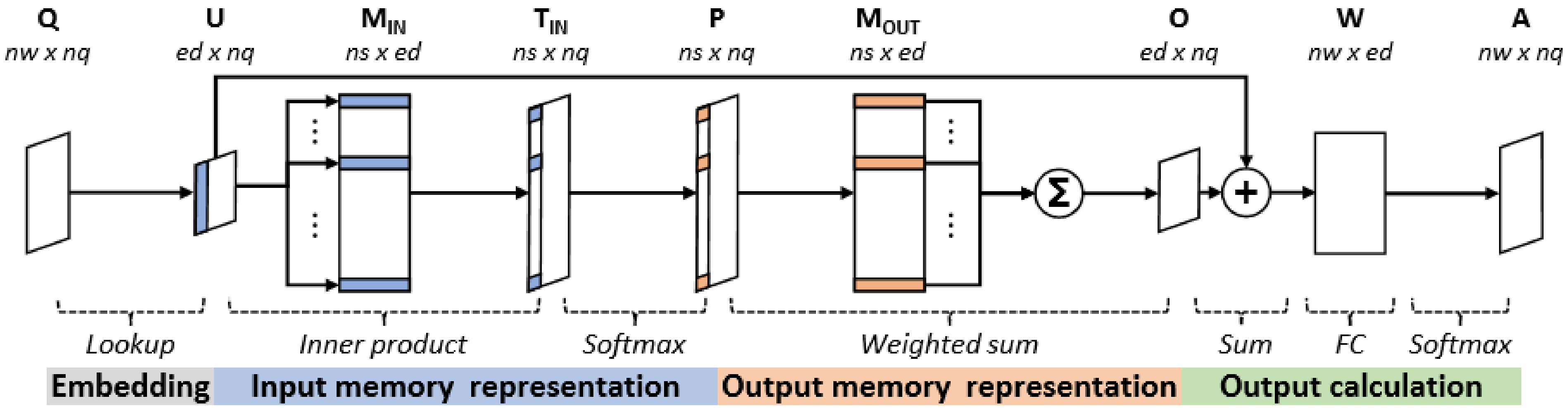
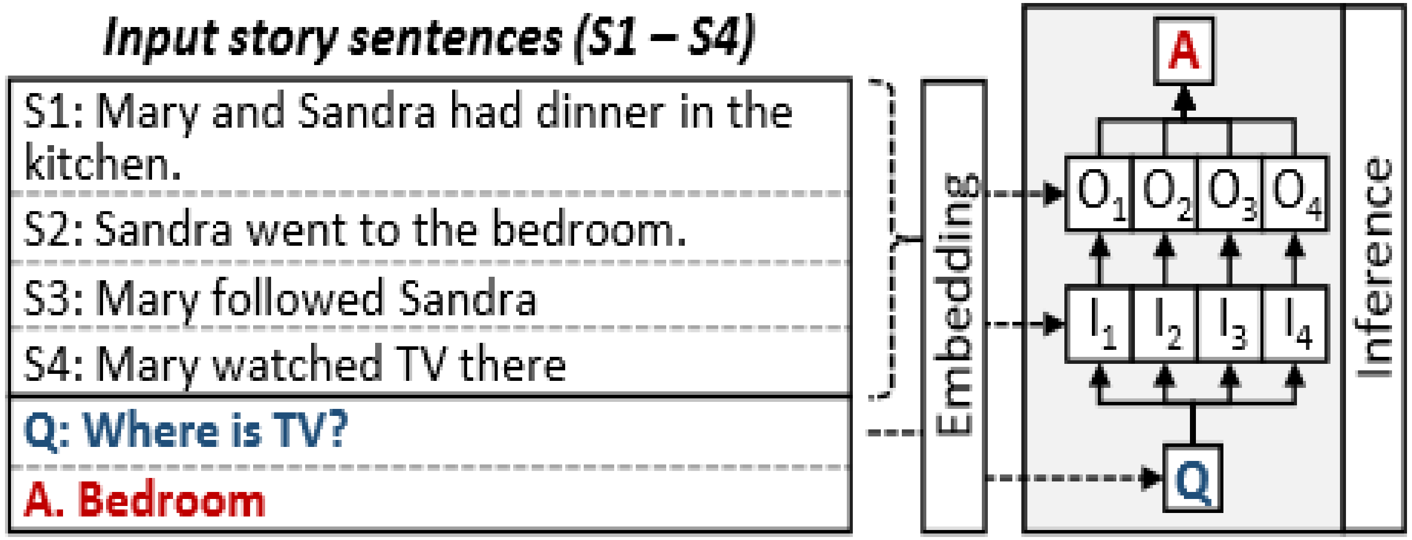


Figure 2: Computational steps of memory networks (MemNN). MemNN consists of embedding, input memory representation, output memory representation and output calculation. *nw* is the maximum number of words in a sentence. *nq* and *ns* are the number of questions and given story sentences, respectively. *ed* is the embedding dimension.

nw: max number of words in sentence
nq: number of questions
ed: embedding dimension
ns : number of story sentence

Motivation

- **To improve MemNNs, we have to increase the size of memory.**
  - Requires a fast and scalable computer infrastructure.
  - Current system **<u>does not provide enough scalability</u>**.



(a) Example scenario (b) Memory network

Motivation- performance problem of MemNN

1. High Memory BandWidth Comsumption

- Data do not fit into the cache

  →Data spills to DRAM memory

- Increased number of DRAM accesses degrades performance

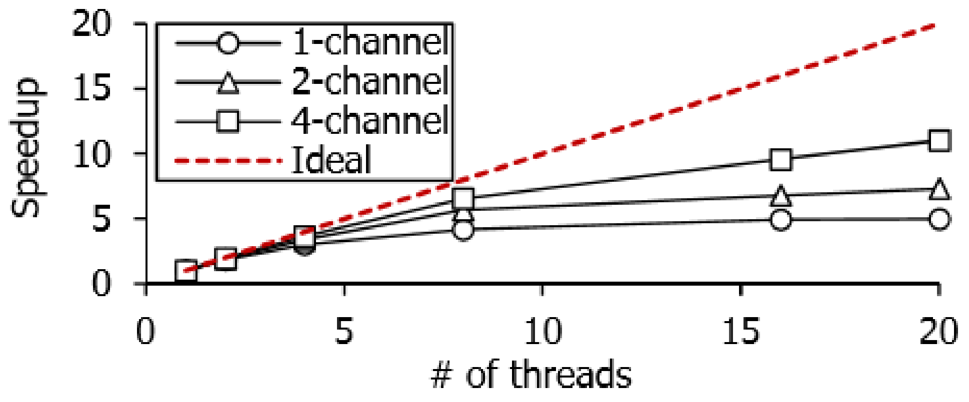


Figure 3: Limited scalability due to memory bandwidth bottleneck. The speedup results of each channel configuration are normalized to the corresponding single-thread results.

Motivation- performance problem of MemNN

2. Heavy Computation

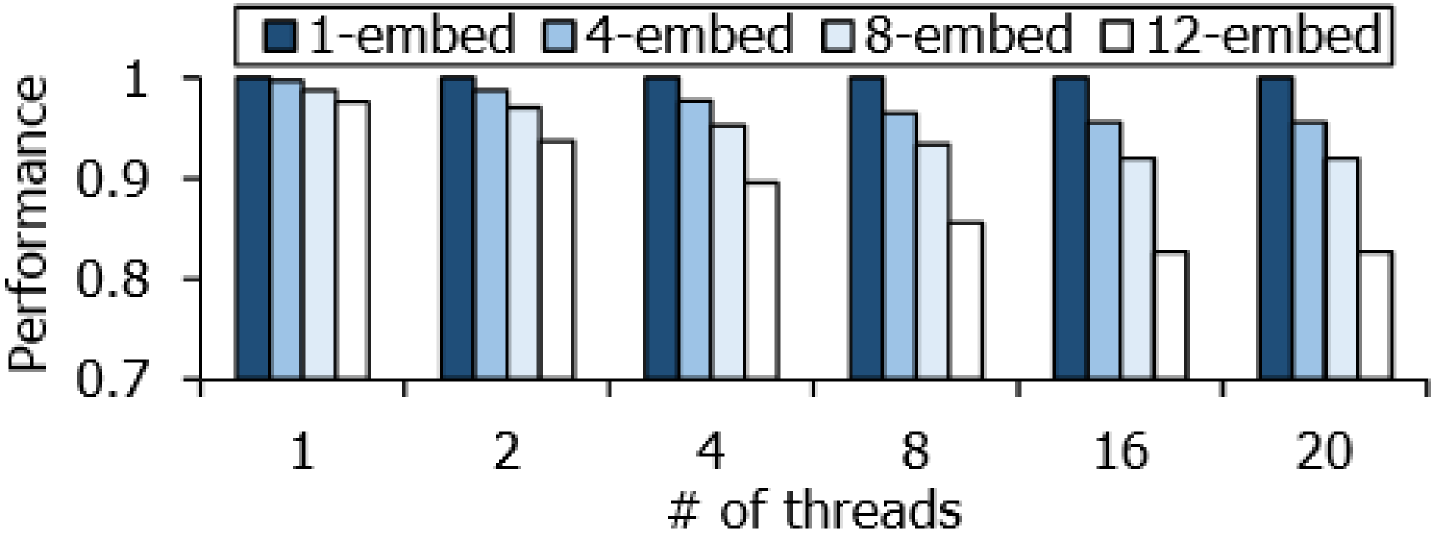
- Certain phases of MemNN consist of a large number of compute-intensive operation.

ex) Matrix inner product, weighted sum, softmax

Motivation- performance problem of MemNN

3. Cache Contention

- MemNN can suffer from **cache conflicts** because of shared cache
- *Inference operation* need to keep necessary data in shared cache
  ↔ *Embedding operation* results in polluting shared cache



MnnFast-Column based Algorithm

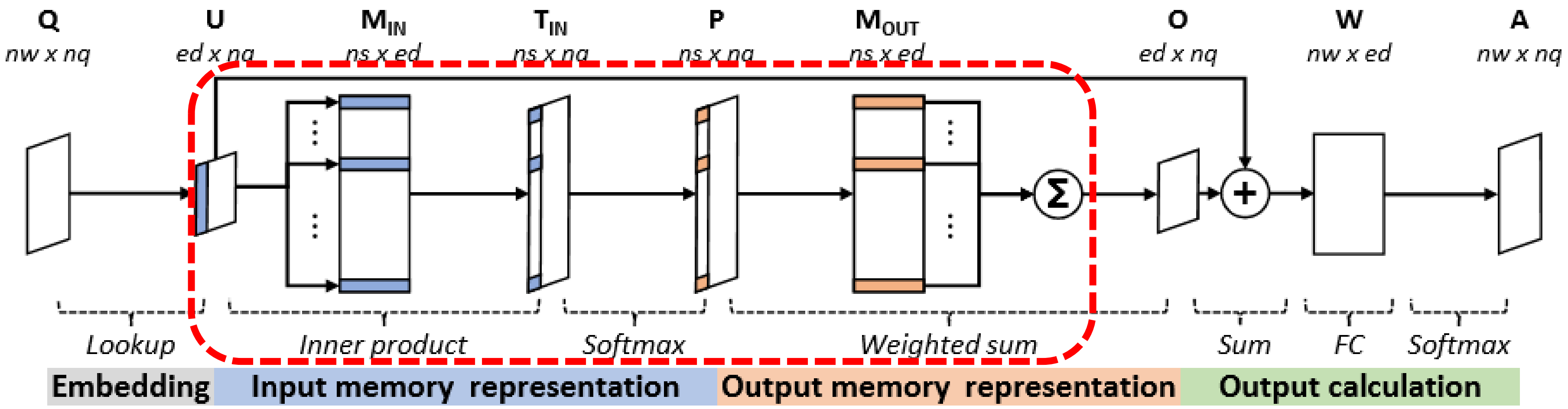


Figure 2: Computational steps of memory networks (MemNN). MemNN consists of embedding, input memory representation, output memory representation and output calculation. *nw* is the maximum number of words in a sentence. *nq* and *ns* are the number of questions and given story sentences, respectively. *ed* is the embedding dimension.

MnnFast-Column based Algorithm

1. Baseline MemNN

Intermediate vector(T_{IN}, P_{exp}, P)

-proportionate to *ns*

-each vector's size is **800MB**

-*spilled to DRAM*

$$o = \sum_i softmax(u \times m_i^{IN}) m_i^{OUT}$$

$$= \sum_i \frac{e^{u \times m_i^{IN}} m_i^{OUT}}{\sum_j e^{u \times m_j^{IN}}}$$

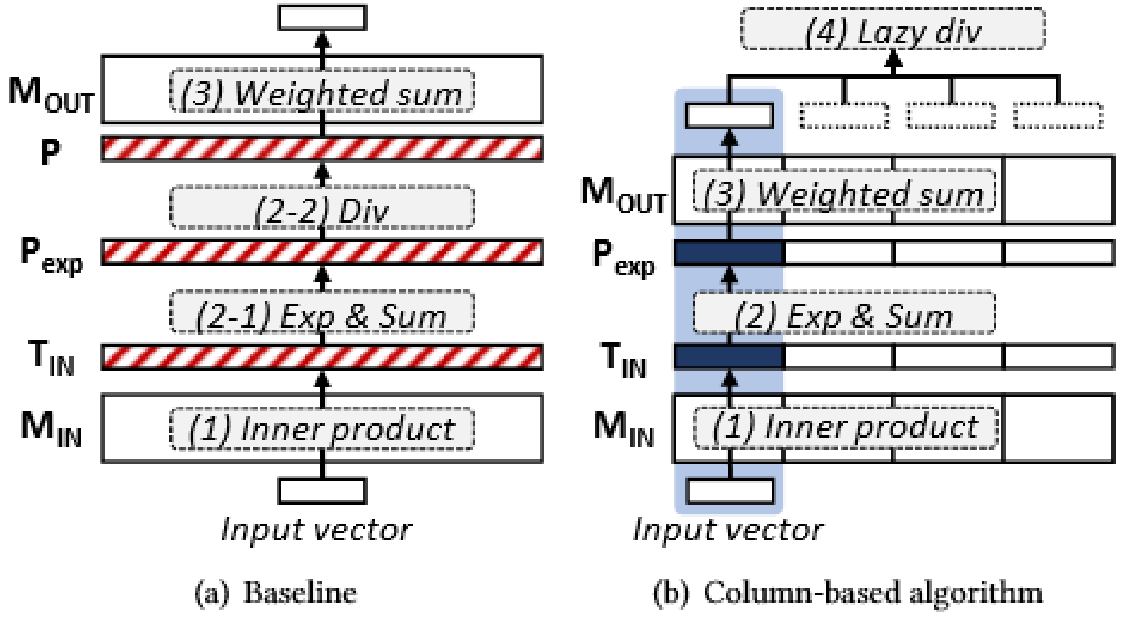


Figure 5: Dataflow comparison betweeen the baseline and the column-based algorithm.

MnnFast-Column based Algorithm

2. Column-based Algorithm

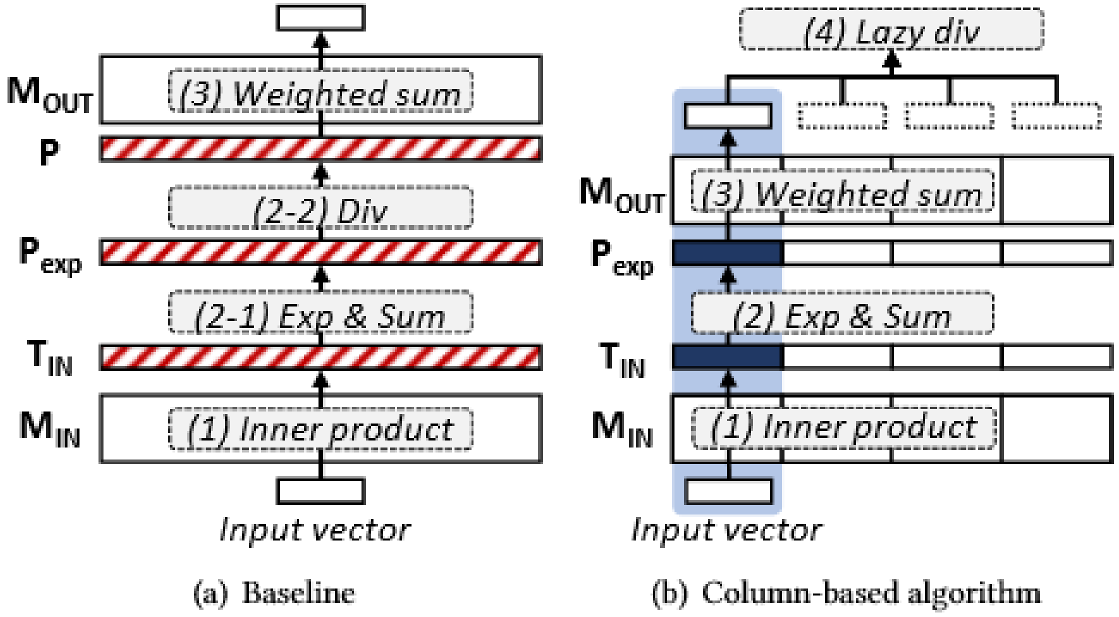


Figure 5: Dataflow comparison betweeen the baseline and the column-based algorithm.

Lazy softmax

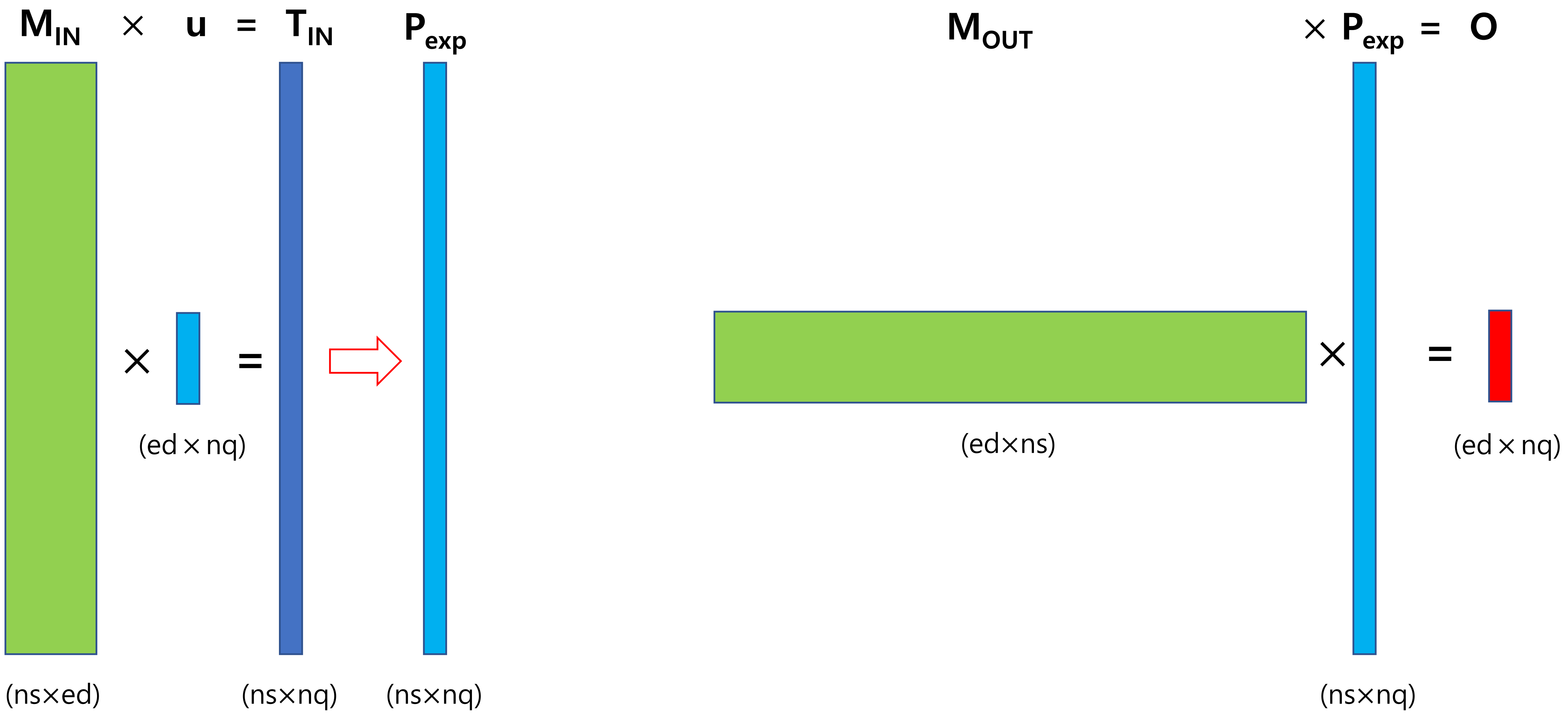
-compute Softmax's division at last

-same results as the baseline

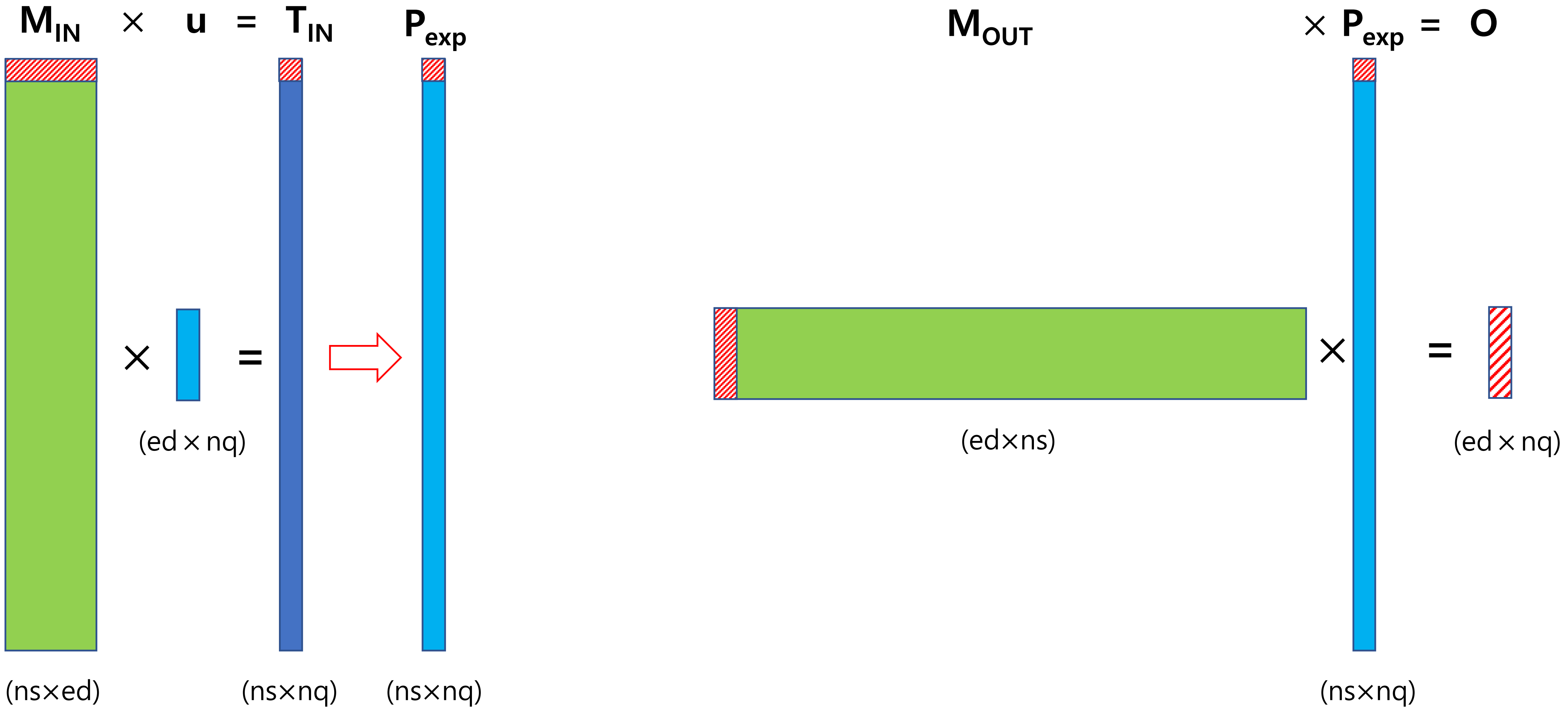
$$o = \sum_i softmax(u \times m_i^{IN}) m_i^{OUT}$$

$$= \frac{1}{\sum_j e^{u \times m_j^{IN}}} \sum_i e^{u \times m_i^{IN}} m_i^{OUT}$$

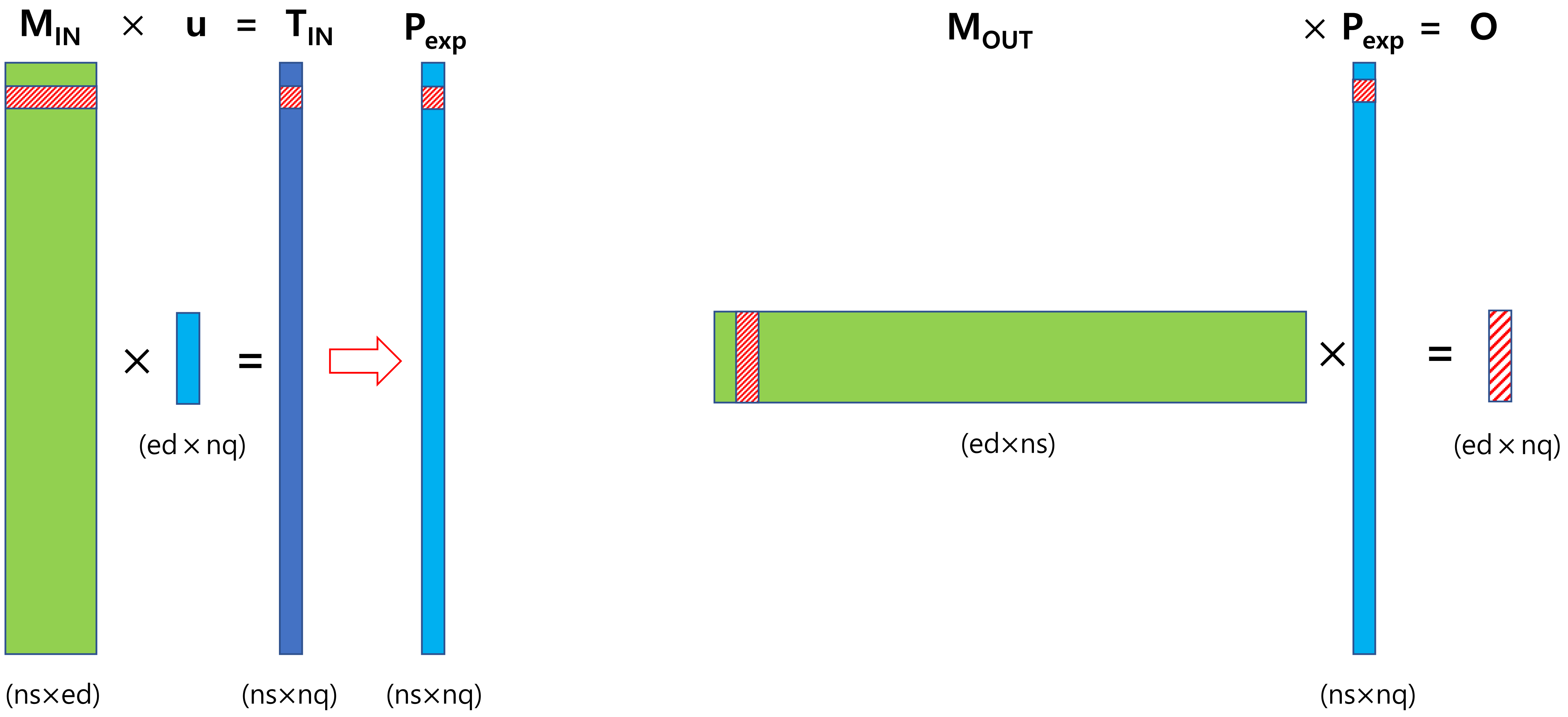
MnnFast-Column based Algorithm



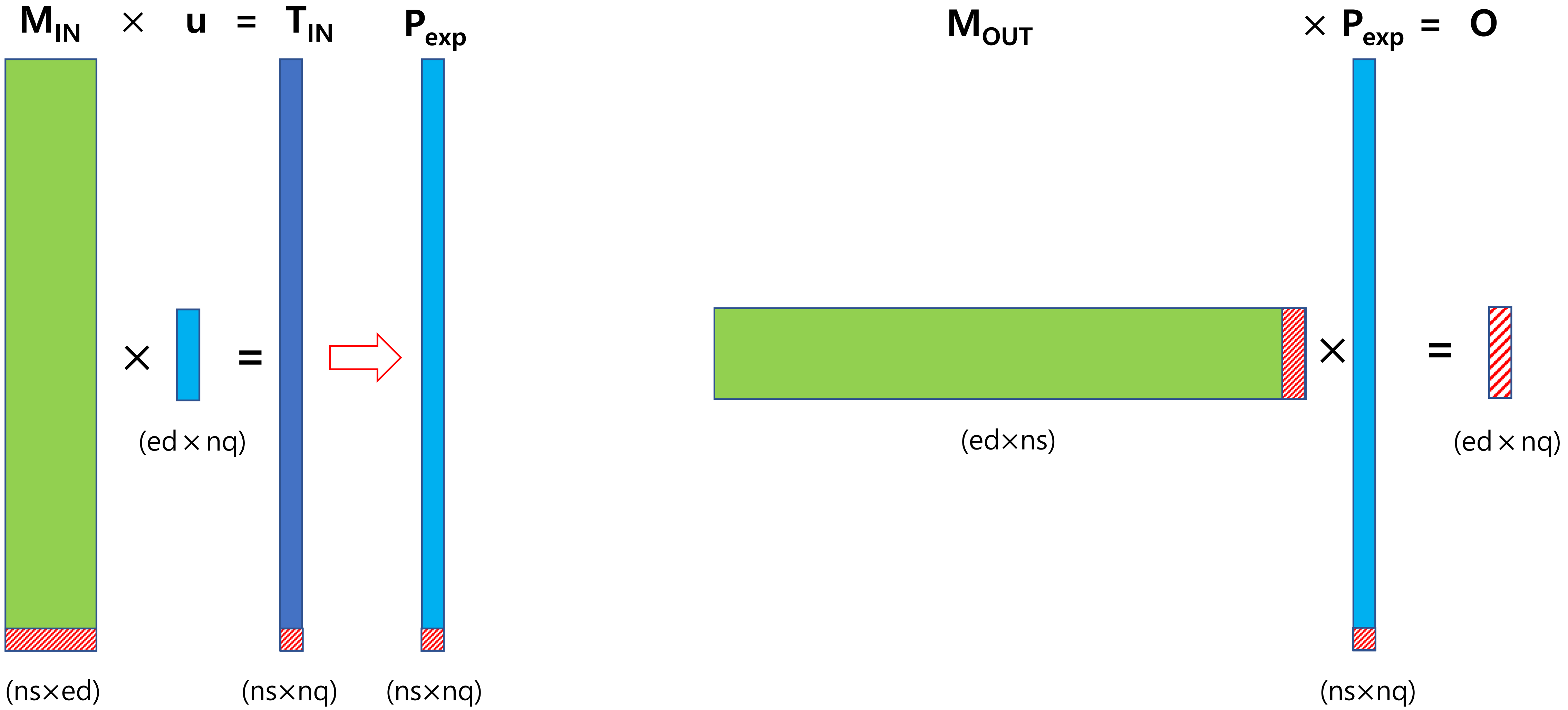
MnnFast-Column based Algorithm



MnnFast-Column based Algorithm



MnnFast-Column based Algorithm



MnnFast-Column based Algorithm

- By doing Column-based Algorithm,
  1) Reduce the size of temporary data to fit those into the on-chip cache

     Column-based MemNN can load those memory into the cache. This leads to the capability of streaming.

  2) Reduce the amount of computation(softmax's division operation)

  3) Column-based MemNN can partition each layer into multiple sub-layers.

MnnFast-Zero Skipping

- In vector p, **<u>only small number of</u>** words and sentences are correlated with given question.

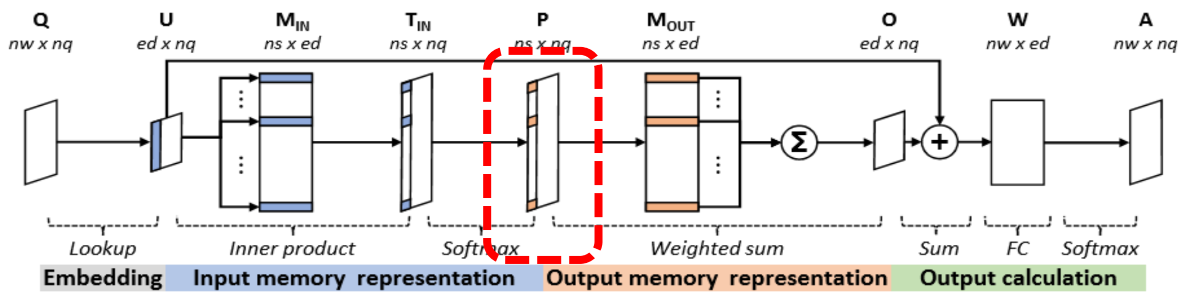


Figure 2: Computational steps of memory networks (MemNN). MemNN consists of embedding, input memory representation, output memory representation and output calculation. *nw* is the maximum number of words in a sentence. *nq* and *ns* are the number of questions and given story sentences, respectively. *ed* is the embedding dimension.

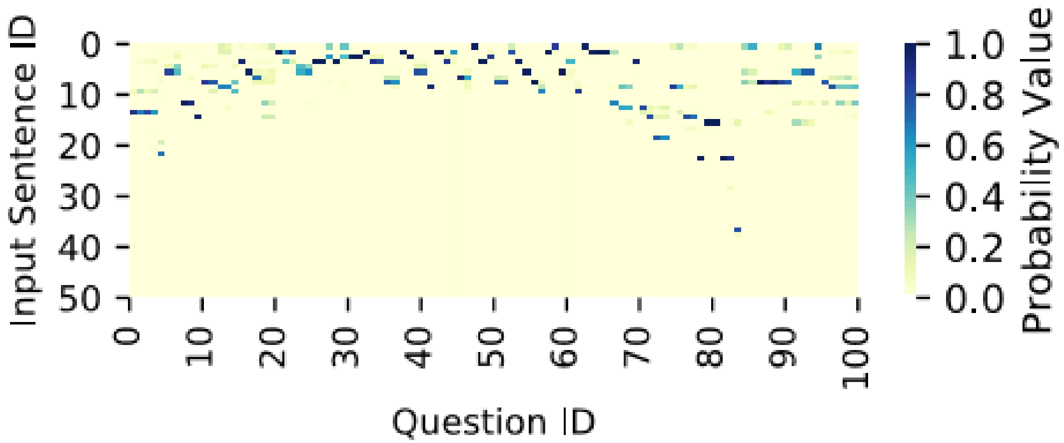


Figure 6: Probability value distribution. Each column represents the probability vector to each question. We use the Facebook bAbi dataset and testset [77].

MnnFast-Zero Skipping

- Algorithm 1 only computes the multiplication when the probability value is larger than a threshold value(th_{skip}).
- There are little trade-off between accuracy and skip ratio.

Algorithm 1: MnnFast's zero-skipping algorithm.

```
input  : The skip threshold th_skip
input  : The probability vector P
input  : The output memory M_IN
input  : The number of story sentences ns
output : The weighted sum O
/* Calculate the weighted sum of the output memory with the probability
   values.                                                              */
O = [0] /* Initialize the output vector.                                */
foreach i < ns do
    /* ns is the number of story sentences.                             */
    if p_i > th_skip then
        O = O + p_i m_i^OUT
end
return O
```

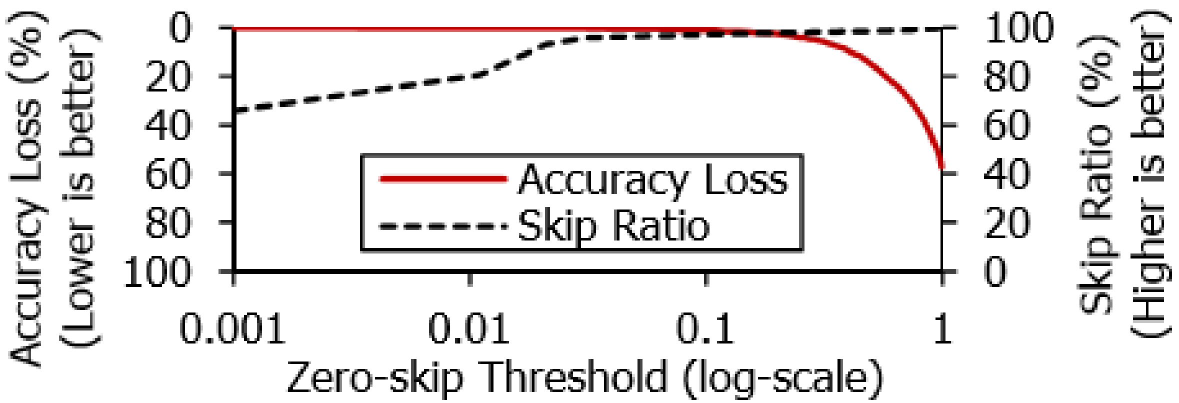


Figure 7: Tradeoffs between accuracy loss (relative loss in accuracy) and computation reduction according to the skip threshold.

MnnFast-Embedding Cache

- **Cache bypassing** has two major drawbacks
  1) increases execution latency of embedding operation
  2) raises the amount of memory pressure

- **Embedding cache** is a dedicated cache for storing internal state vectors during the embedding operation.

Implementation

- CPU
  - ❖Baseline Implementation
    - – MemNN is implemented in C++ with open-source BLAS library, OpenBLAS
    - – All input/output memory have already been converted into internal data format.
    - – Rely on **<u>OpenBLAS</u>** for efficient computation.
  - ❖MNNFast Implementation
    - – All operation except for softmax are implemented in the **<u>same way to the baseline</u>**.
- GPU
  - ❖GPU kernel implementation
    - – Rely on **<u>cuBLAS</u>** provided with CUDA toolkit 10.0.
    - – Only softmax operation is implemented as one custom kernel.
  - ❖MNNFast Implementation
    - – Each stream processes chunk and parallelized.
    - – Zero-skipping is ineffective

Implementation

- FPGA
  - ❖Baseline Implementation
    - – Omit the baseline implementation because its design is straightforward
  - ❖MNNFast Implementation
    - – During embedding, MNNFast converts a question and story sentences by passing them through embedding cache.
    - – Embedding cache is designed as a direct mapped cache

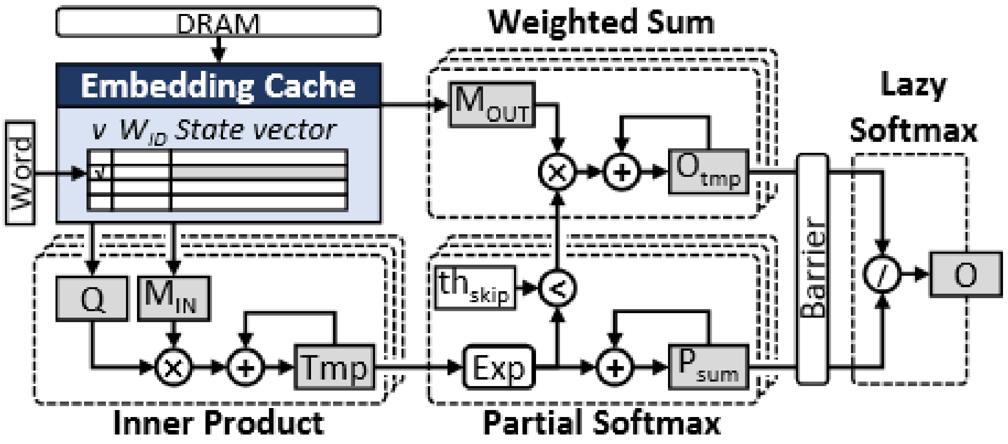
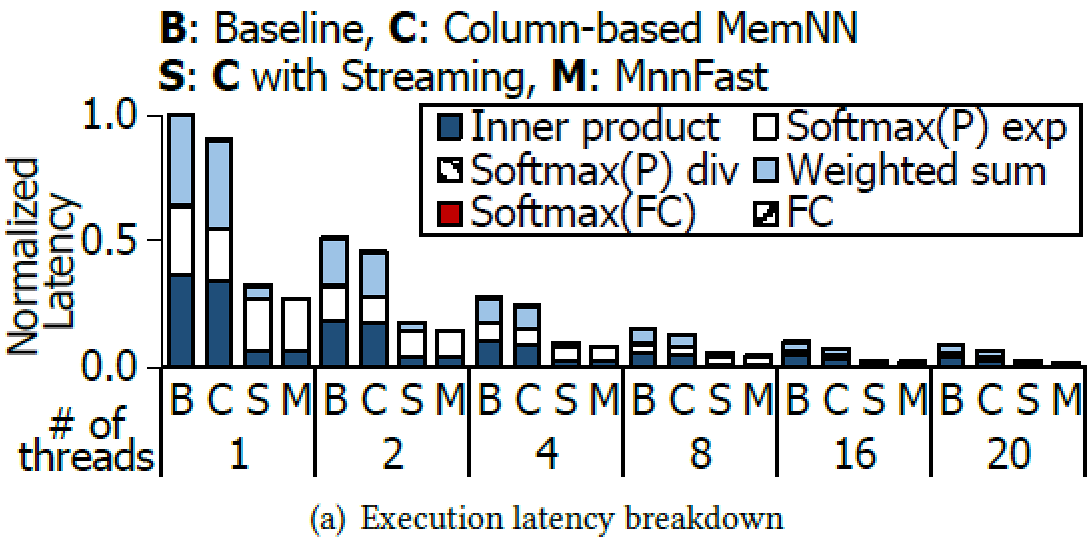


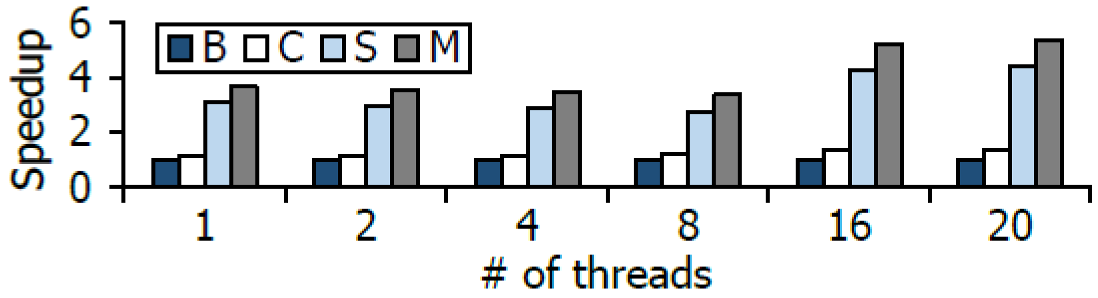
Figure 8: A high-level architecture of FPGA-based MnnFast.

Evaluation

-Performance on CPU



(a) Execution latency breakdown



(b) Performance speedup

Evaluation

- Cache efficiency on CPU

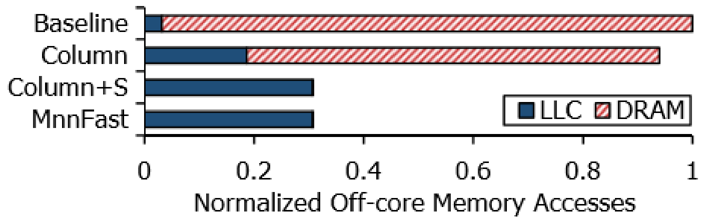
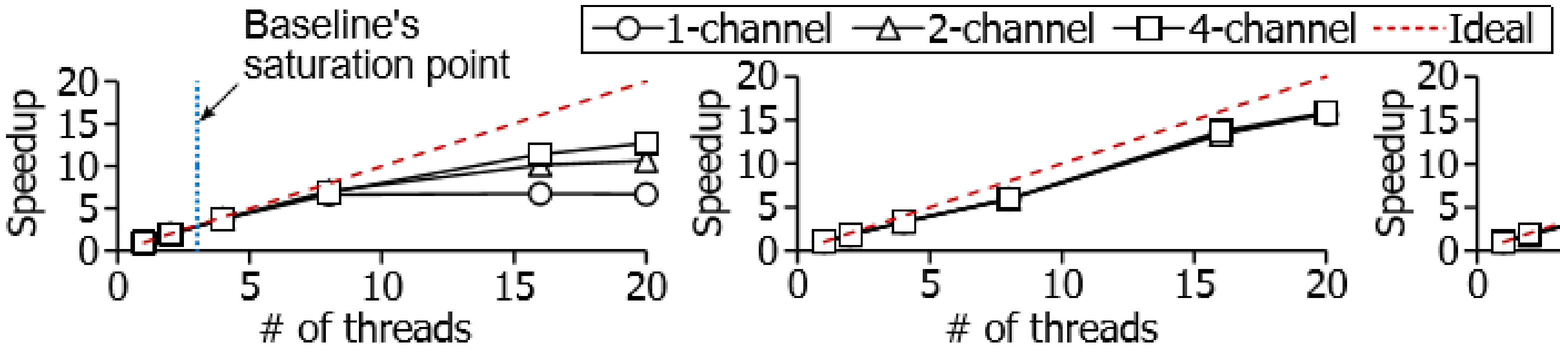


Figure 11: The number of off-chip memory accesses on CPU.



(a) Column

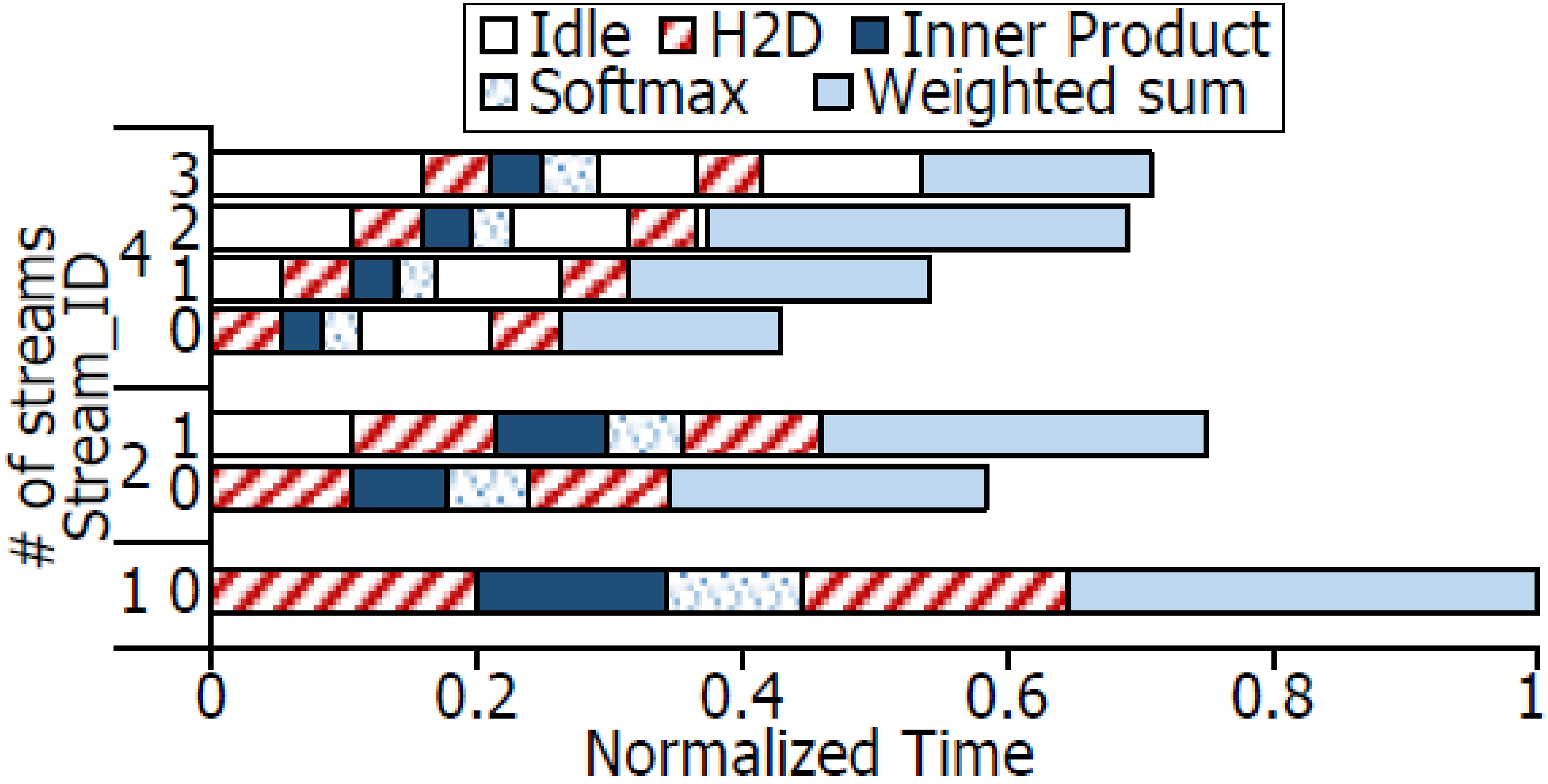
(b) Column-S

(c) MnnFast

Figure 10: Scalability of column-based algorithm on CPU.

Evaluation

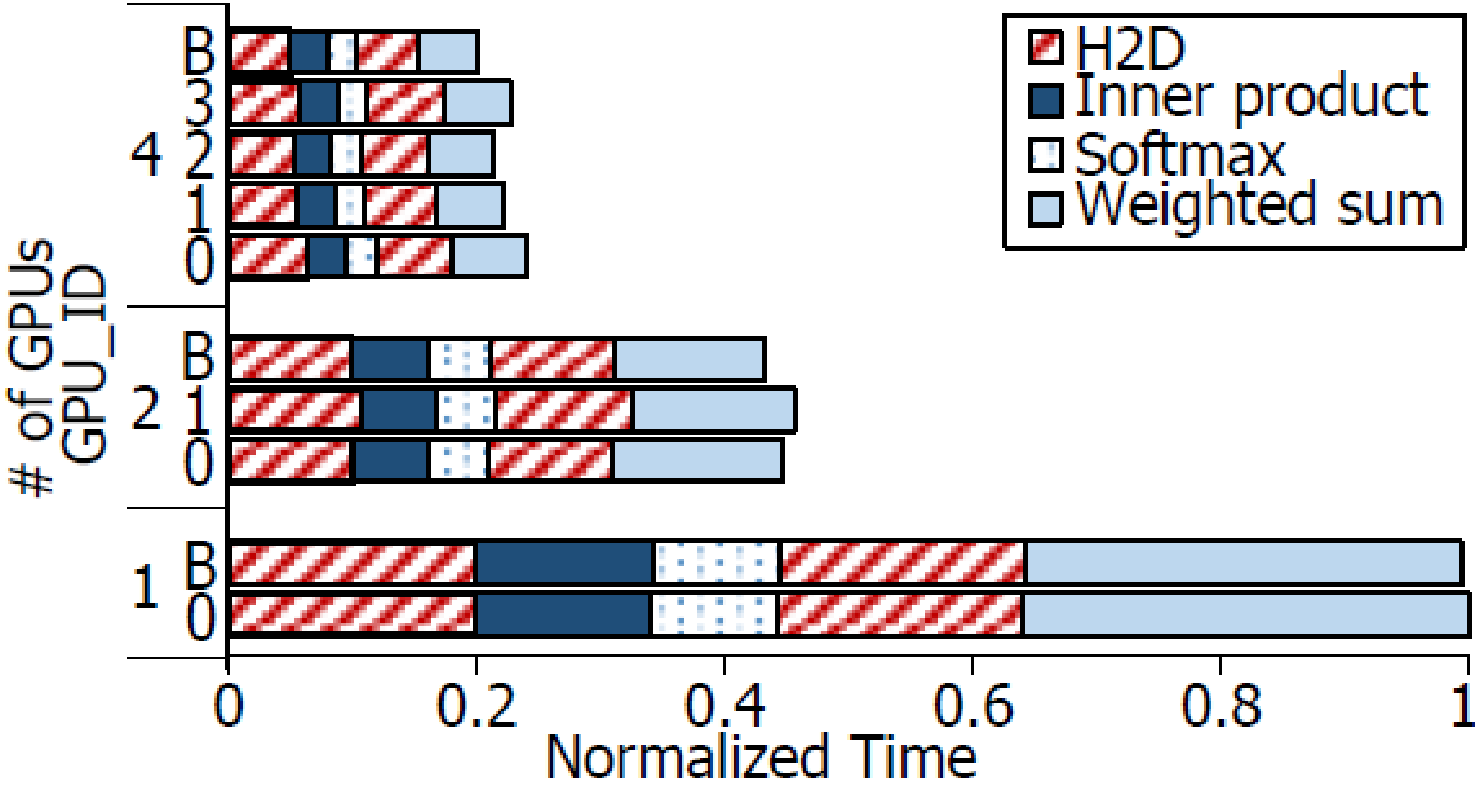
- Multiple CUDA streams



(a) Multiple CUDA streams

Evaluation

- Multiple GPUs



(b) Multiple GPUs (B: best run-alone)

Evaluation
- Performance of FPGA

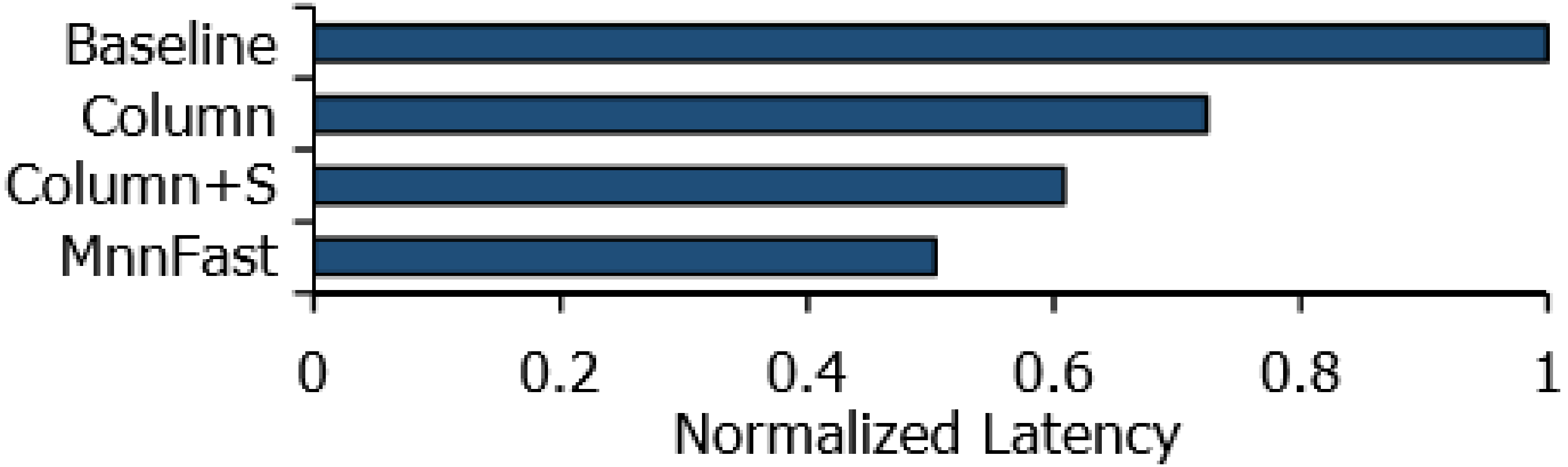


Figure 13: Latency reduction of FPGA-based MnnFast. Each latency is normalized to the baseline.

Evaluation
- Effectiveness of Embedding Cache in FPGA

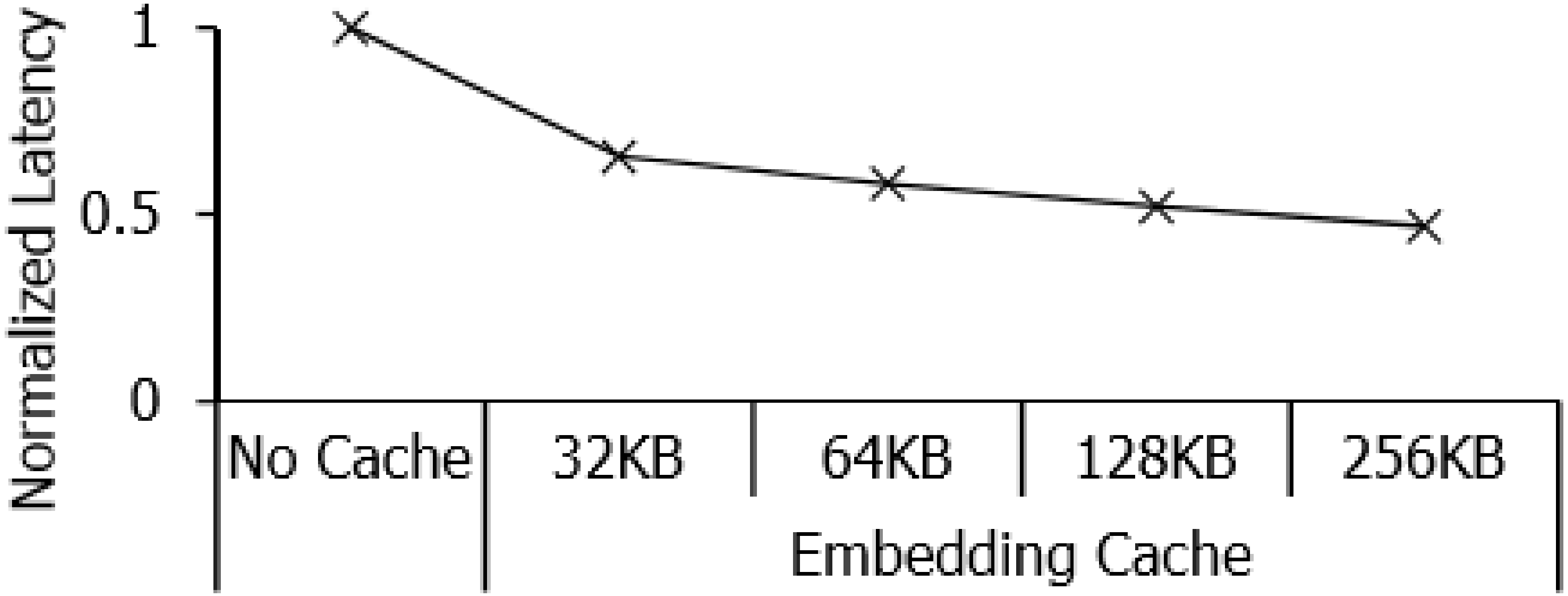


Figure 14: Effectiveness of embedding cache in FPGA-based MnnFast. Each latency result is normalized to the No Cache.

Conclusion

- **Three performance problems** of the current architecture :
  - *high memory bandwidth consumption, heavy computation, cache contention*.
- **Three key optimizations** proposed by MnnFast:
  - *column-based algorithm, zero-skipping, and embedding cache.*
- MnnFast solves problem and outperforms the baseline on various hardware : CPU, GPU, and FPGA

Thank you