# GeneralSparse: Bridging the Gap in SpMM for Pruned Large Language Model Inference on GPUs

SKLP, Institute of Computing Technology, CAS
University of Chinese Academy of Sciences

Reporter: Yaoyu Wang

# content


01 Introduction

02 Background & Motivation

03 GeneralSparse Design

04 Experiment

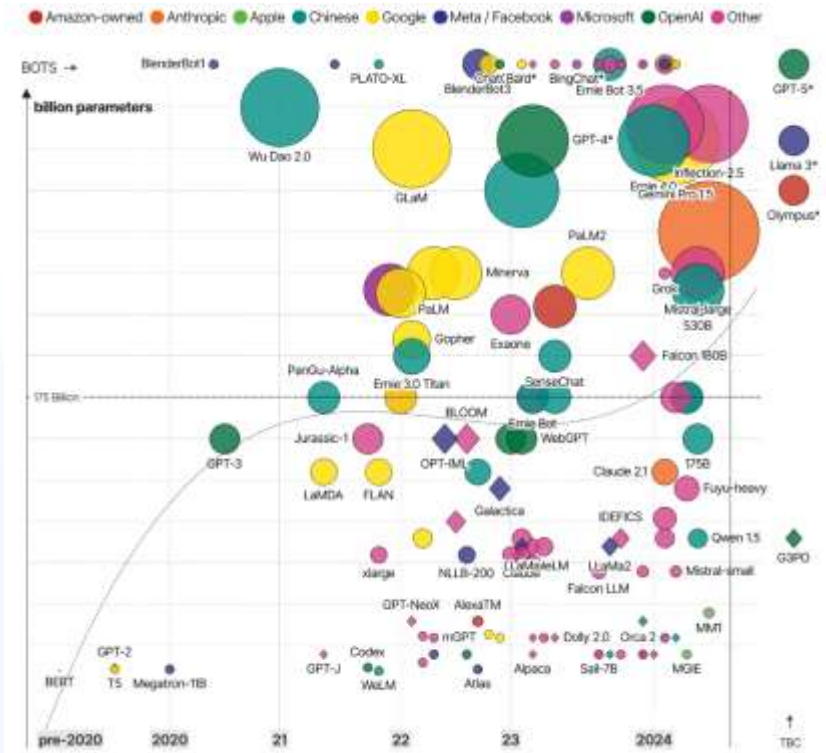
05 Related Work

06 Conclusion & Future Work

# Introduction

01

# Deployment challenges of LLMs are brought by the growth of parameter size (storage, inference latency)




LLM performance and timeline comparison (Source: information is beautiful)

## Storage

The number of parameters in LLMs has surged, doubling storage requirements and putting huge pressure on hardware and memory.

## Inference Latency

Large-scale parameters lead to prolonged inference times, directly impairing the user experience.

## Deployment Challenges

High storage demands and long latency pose challenges to the limits of edge devices and real-time applications.

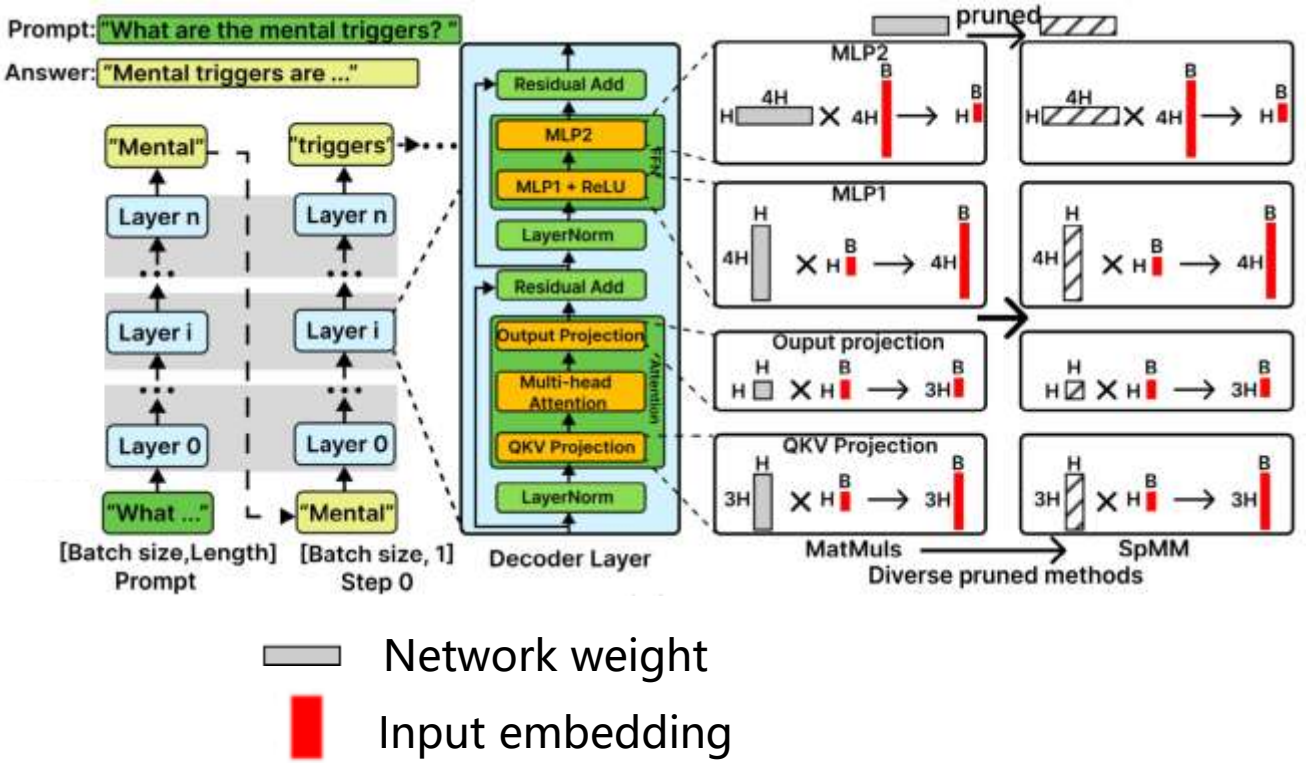
# The matrix-multiplication operations account for the majority of the inference time.

**Core Computations in the Decoding Layer**

- The four major matrix-multiplication operations in the decoding layer consume a significant amount of time, accounting for as much as 80%.

**Pruning methods are key ways to solve this challenge (storage and inference latency)**

- Pruning reduces network weights and maintains the accuracy of the results.
- It reduces both the storage of weights and the amount of calculation.

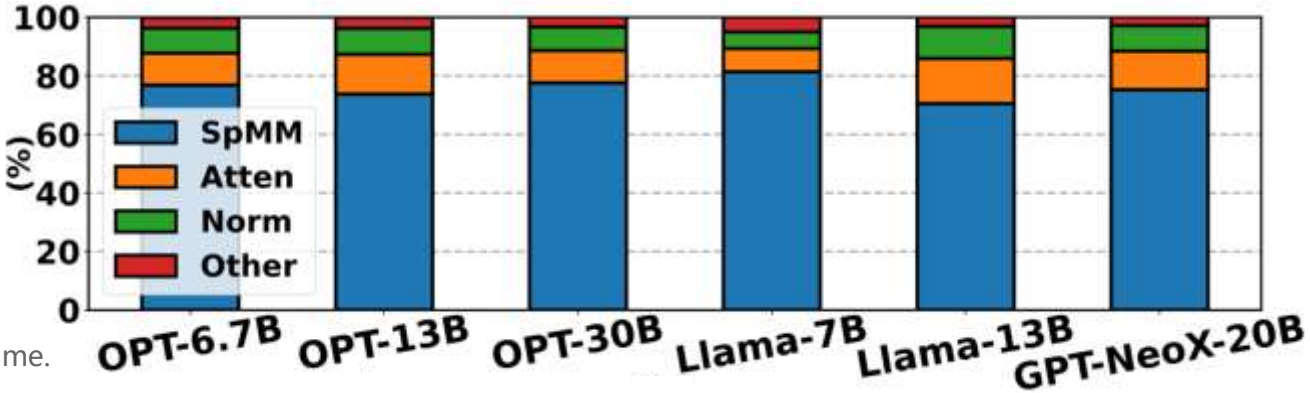


# However, SpMM remains a performance bottleneck

It still occupies the major part in the inference process.

**Bottleneck Analysis of SpMM**

- Although SpMM has the theoretical potential for high efficiency.
- In practical applications, it has become a bottleneck restricting performance, occupying about 70% of the execution time.



# Background & Motivation

02

# Weight pruning reduces computational and memory overhead, but introduces the sparse diversity.

### Weight Pruning

By removing redundant weights, it significantly reduces the model size and computational complexity, improving efficiency.

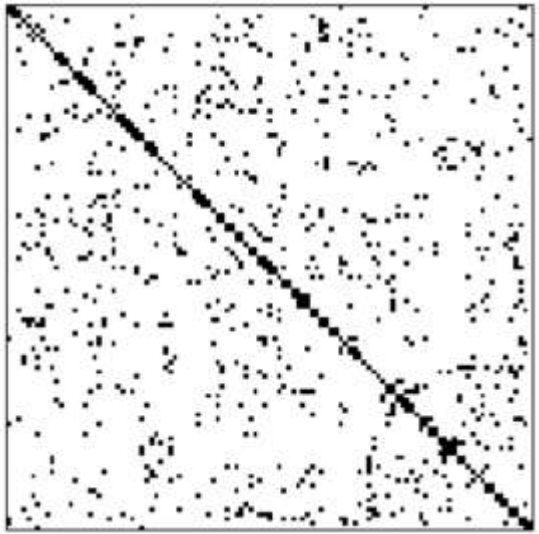
### Two Sparse Diversity Aspects

**1. Sparse Patterns**

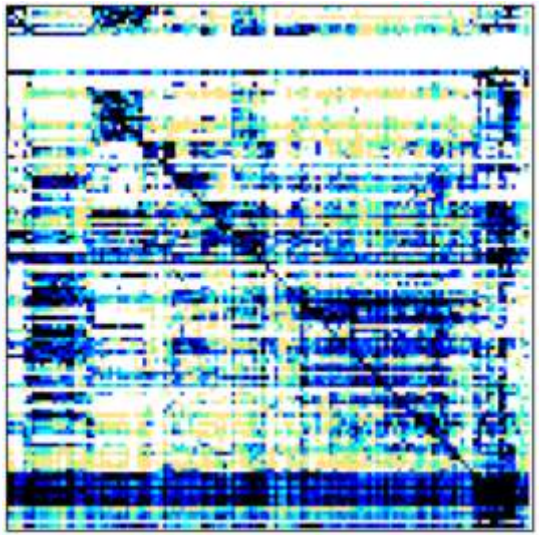
**2. Sparse Levels**

After pruning, the weight distribution is uneven, forming complex sparse matrices, which poses new challenges to the computing architecture.

## Pruned Weight Matrices (e.g.)



Magnitude[1] & 90%



Random & 60%

**Sparse Patterns & Sparse Levels**

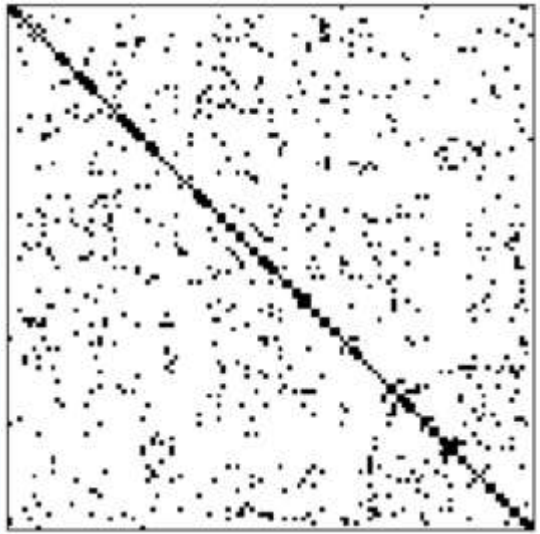
[1] Han S, Pool J, Tran J, et al. Learning both weights and connections for efficient neural network[J]. Advances in neural information processing systems, 2015, 28.

# 1. Sparse Patterns

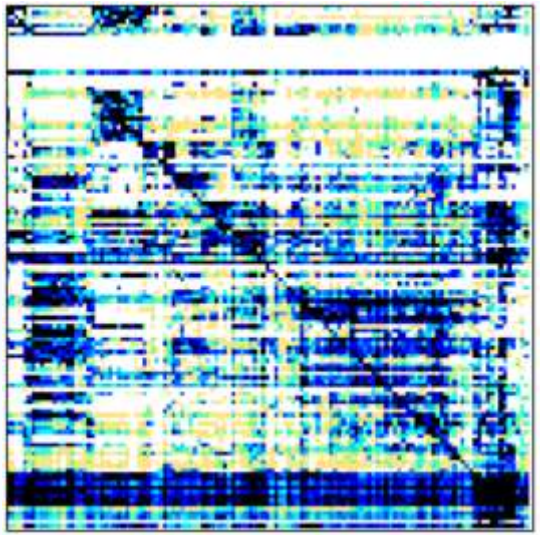
**For example,** two well-known pruning methods: random and magnitude.

- Random prunes values randomly and results in a uniform non-zero element distribution per row.
- Magnitude prunes values with smaller absolute values of matrix weight and leads to an unpredictable and irregular distribution.

**Pruned Weight Matrices (e.g.)**



Magnitude[1] & 90%



Random & 60%

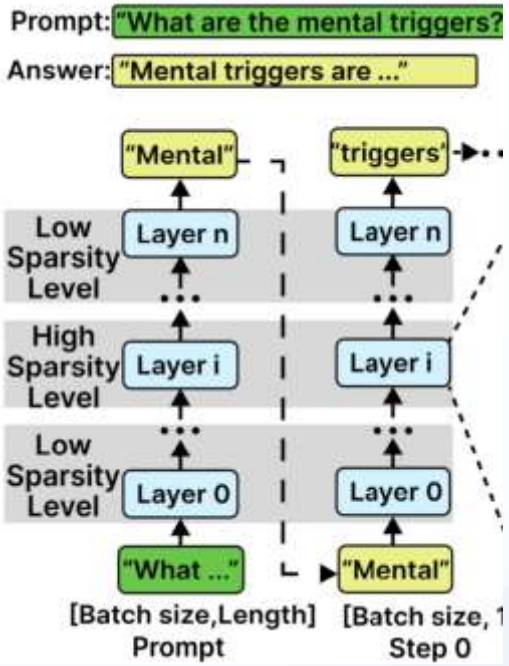
**Sparse Patterns & Sparse Levels**

[1] Han S, Pool J, Tran J, et al. Learning both weights and connections for efficient neural network[J]. Advances in neural information processing systems, 2015, 28.

## 2. Sparsity level

**For example, different depth layers have different sparsity level requirements.**

- With different network depths, the sparsity level after pruning varies. [1]
- The bottom and top layers show higher sensitivities (low sparsity level).
- The intermediate layers have lower sensitivities (high sparsity level)




[1] Shao H, Liu B, Qian Y. One-shot sensitivity-aware mixed sparsity pruning for large language models[C]//ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2024: 11296-11300.

# Existing SpMM solutions have poor adaptability and cannot efficiently handle diverse sparsity.

### Limitations of Traditional SpMM

- The traditional SpMM algorithm has the relative single-design.
- Difficult to cope with complex and variable sparsity, resulting in low efficiency.

### Diversity of Sparsity

The pruned model weights exhibit irregular sparsity, which sets high requirements for algorithm flexibility.

### Performance Bottlenecks Emerge

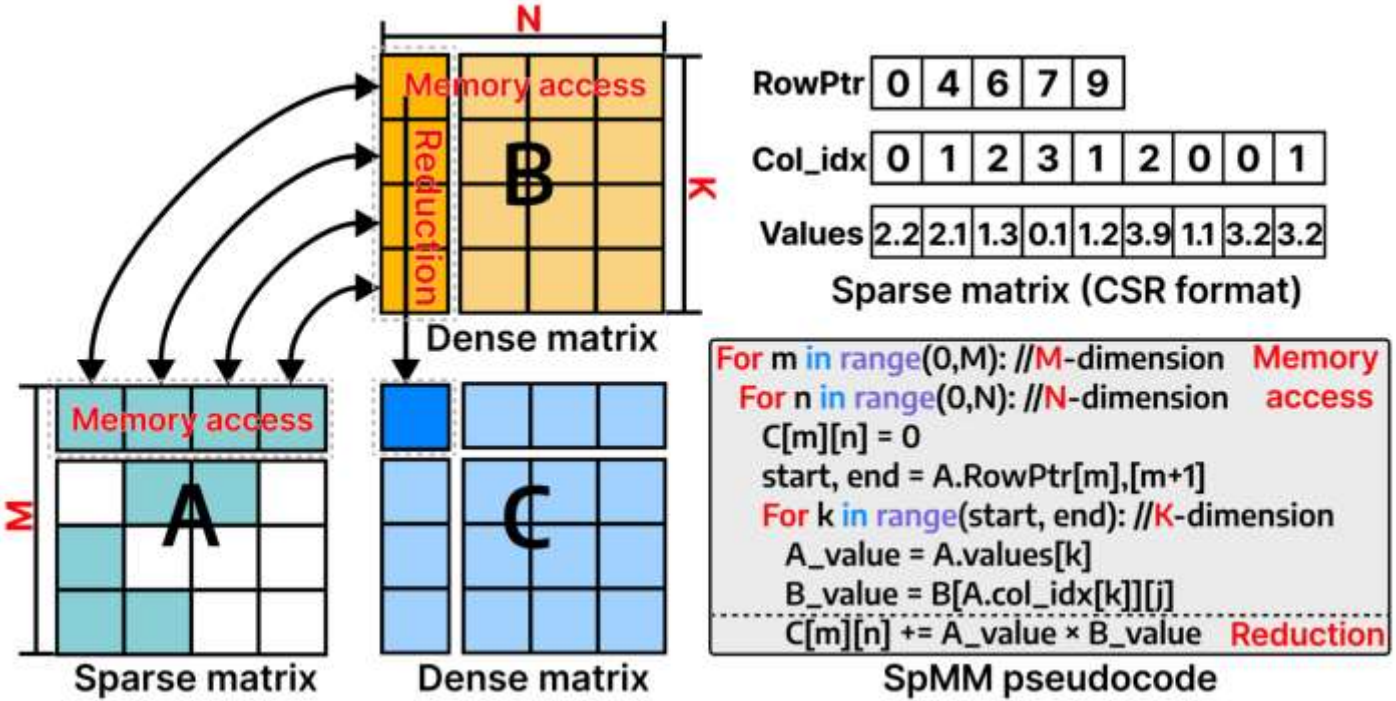
Existing SpMM solutions show a significant decline in computational and memory access efficiency when dealing with diverse sparse patterns.

### Demand Calls for Innovation

There is an urgent need for an SpMM algorithm that can adaptively adjust to overcome the limitations of existing technologies.

# SpMM (Sparse Matrix- Dense Matrix Multiplication)

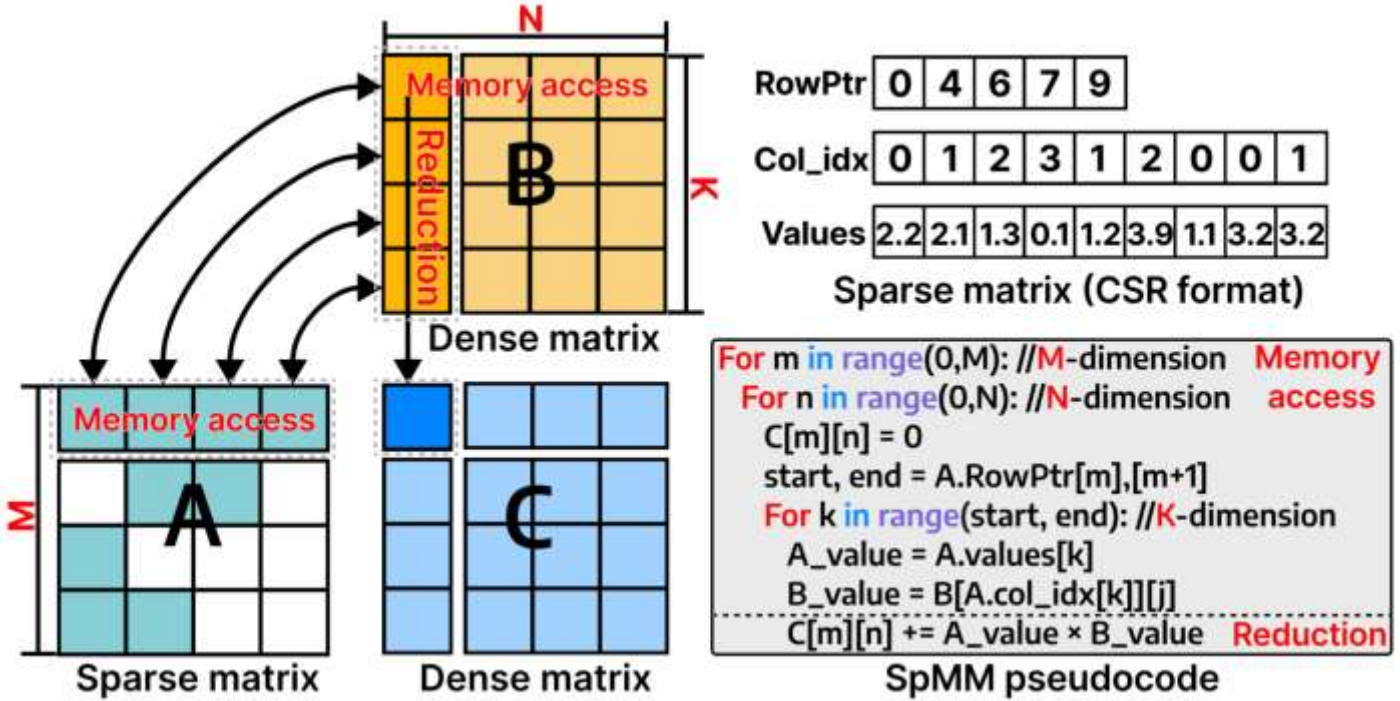
We analyze the computation process of SpMM and consider two key aspects that affect the performance.



# SpMM (Sparse Matrix- Dense Matrix Multiplication)

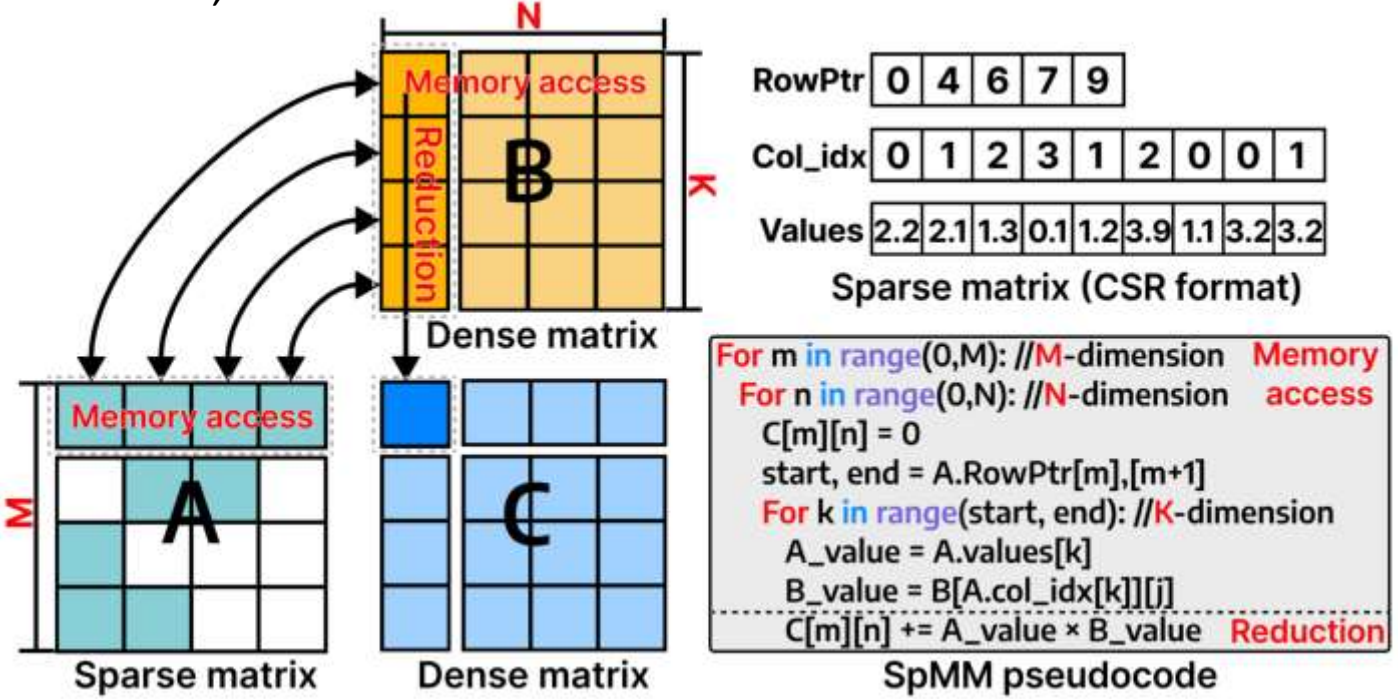
As shown in pseudo code, calculation process of SpMM has three loops.

1. One aspect is how to distribute three loops (M/N/K dimension) of sparse and dense matrix to GPU process units for parallel memory access.



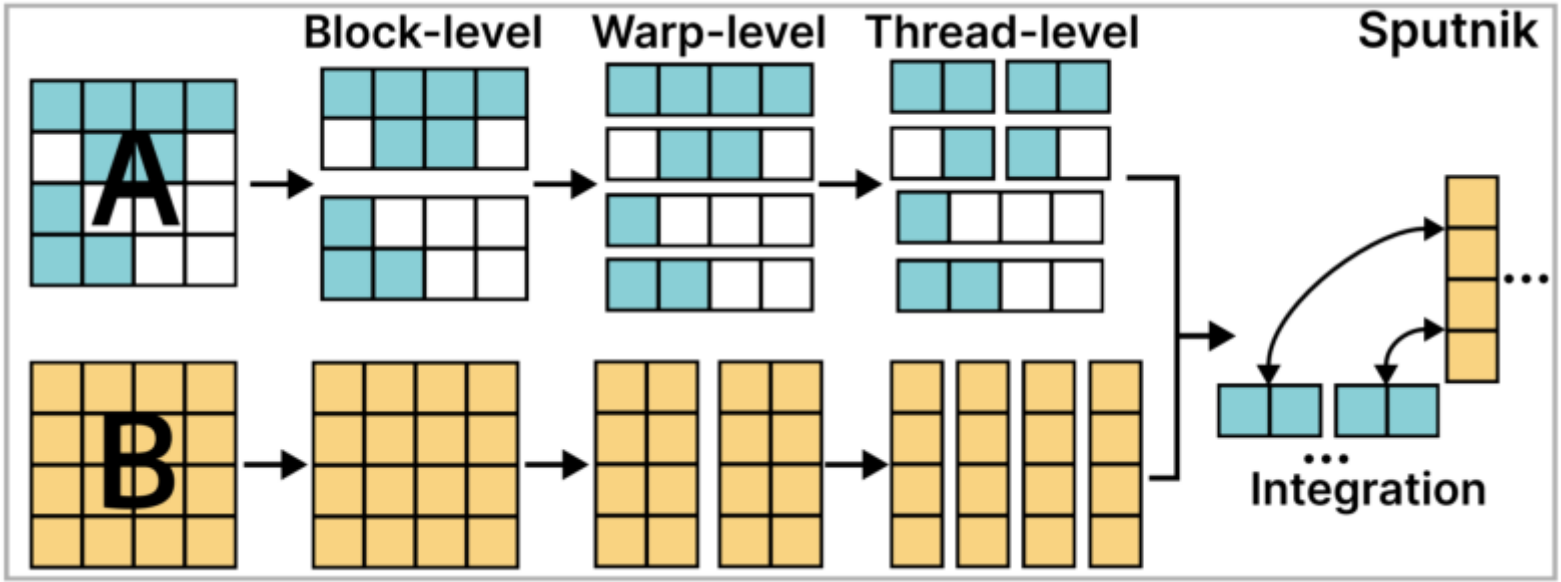
# SpMM (Sparse Matrix- Dense Matrix Multiplication)

2. The other aspect is how to accumulate (reduce) intermediate results with reduction dependencies of K-dimension.
Because the most important thing about parallel computing is the correctness of the results, there must be no write-back conflicts



# Motivation1 – Memory access space

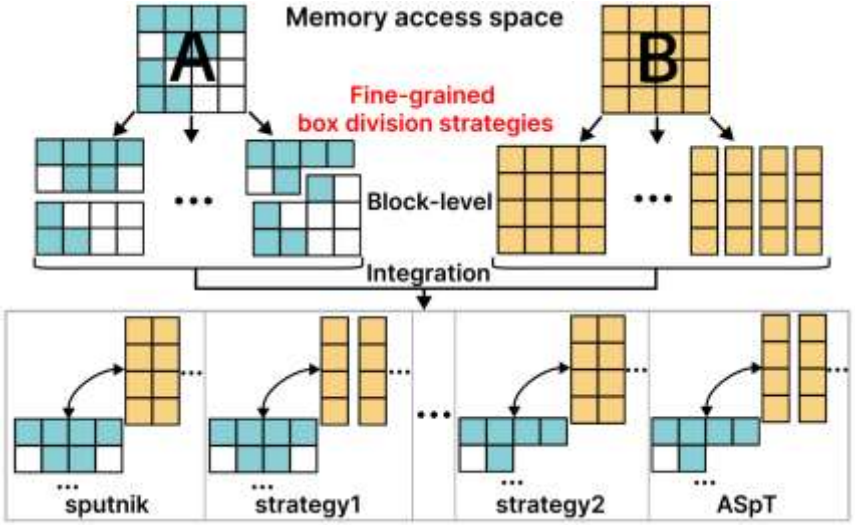
Parallel memory access strategies can be abstracted as the process of dividing boxes in three-levels of GPU (Block/Warp/Thread) (e.g., Sputnik[1]).

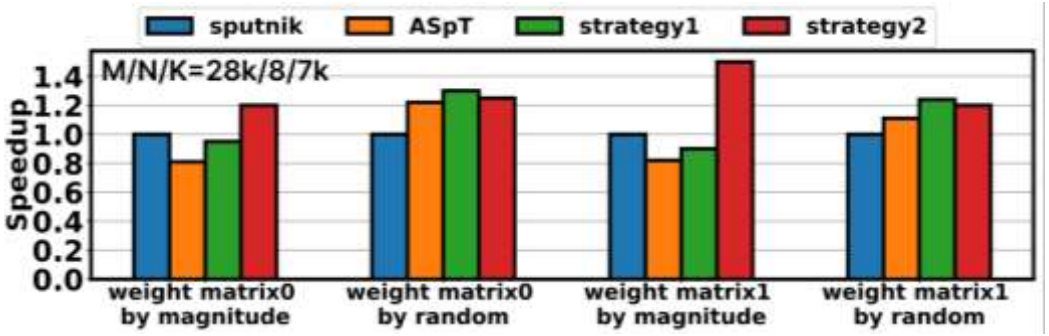



[1] Gale T, Zaharia M, Young C, et al. Sparse gpu kernels for deep learning[C]//SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020: 1-14.

# Motivation1 – Memory access space

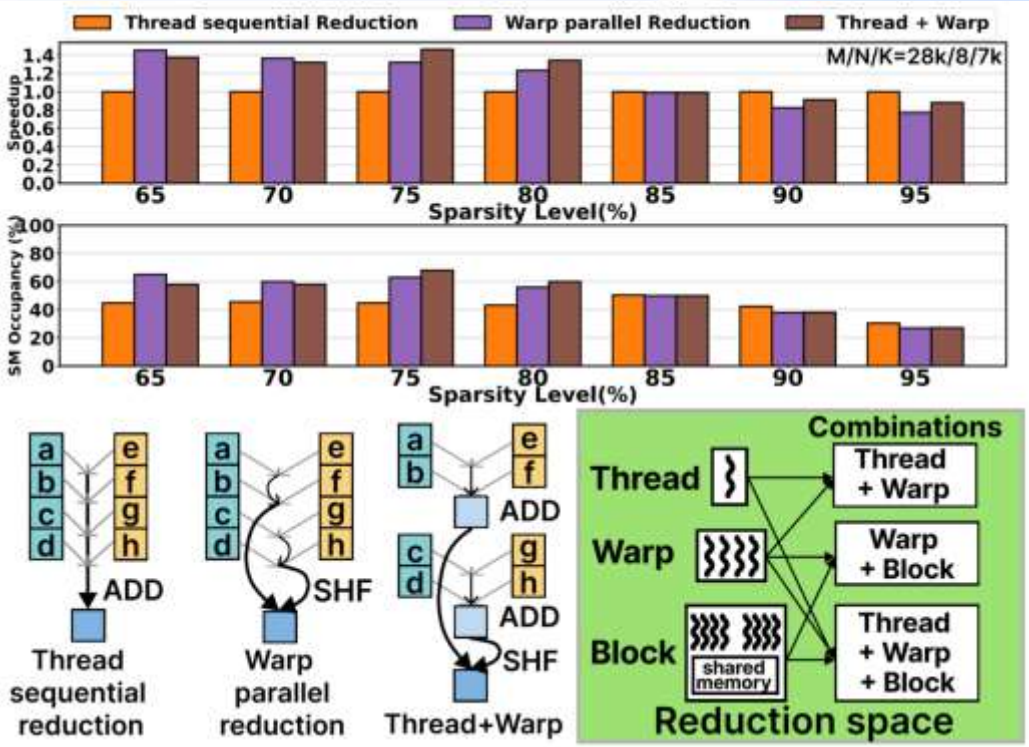
Memory access space is composed of many fine-grained box division strategies of the sparse and dense matrix. For simplicity, warp/thread-level is not shown here.



# Motivation2 – Reduction space

Multi-level reduction algorithms can be used in combination for varied sparsity levels in LLMs.

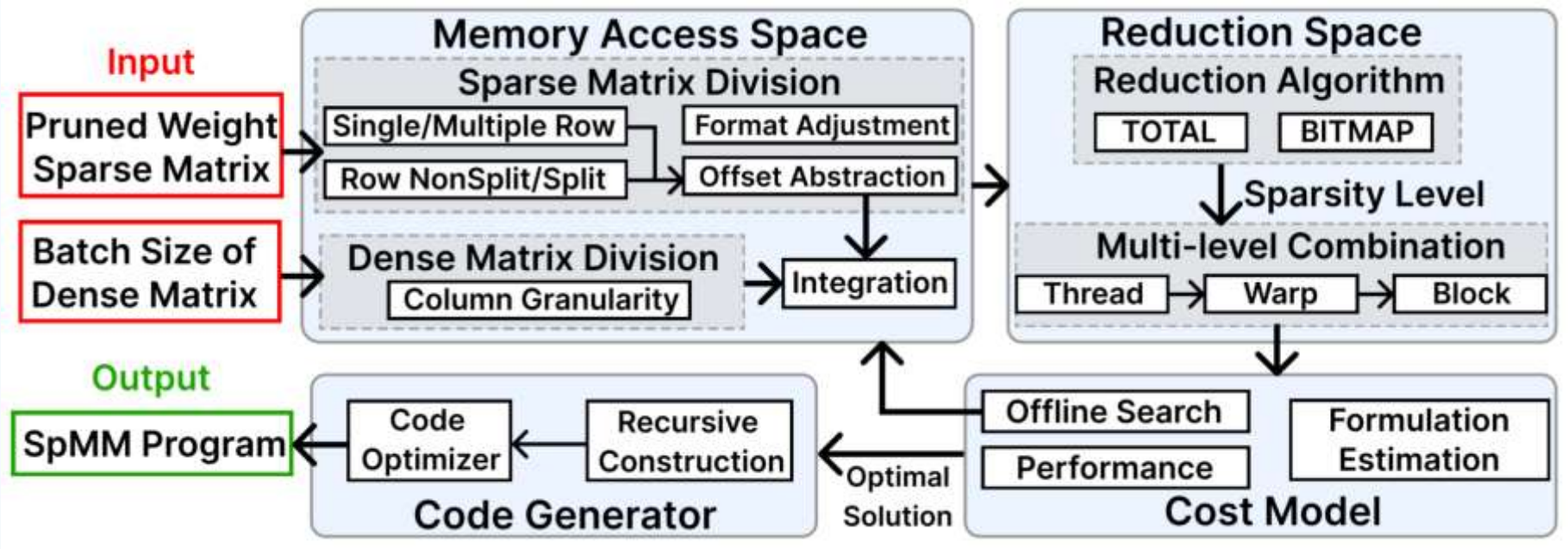


# GeneralSparse Design

03

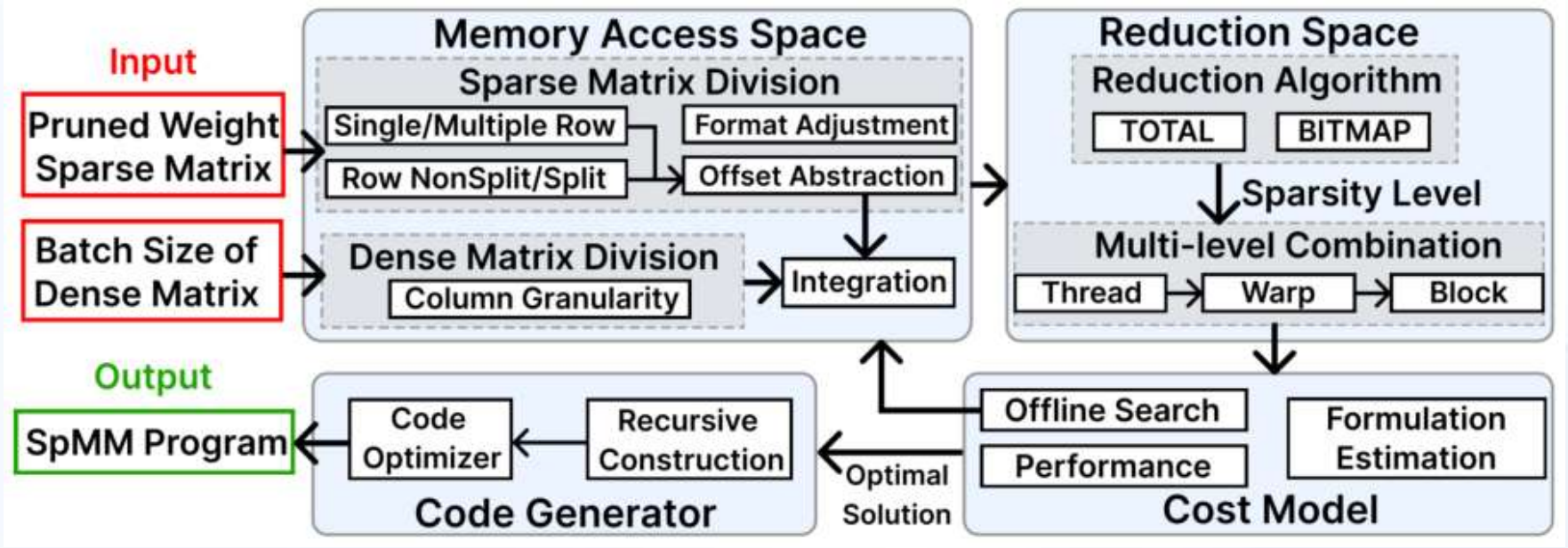
# Overview--GeneralSparse

We propose GeneralSparse, which automatically designs and implements high-performance SpMM programs on GPUs, as illustrated in Figure.



# Overview--GeneralSparse

- Consist of four parts: memory access space, reduction space, cost model, and code generator.
  - The pruned sparse matrix and dense matrix are initially processed within the memory access space.
  - Subsequently, reduction algorithms are selected and applied in the reduction space.
  - Through the cost model for the optimal solution, the code generator generates the SpMM program.

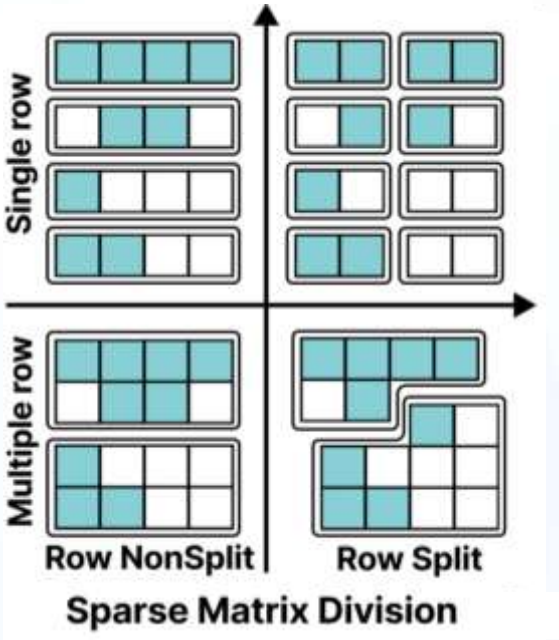


# Memory Access Space

## Orthogonal dimension in Sparse Matrix Division (allocation) for diverse sparsity

- ***Single/multiple-row*** dimension means non-zero elements of single/multiple rows in the sparse matrix are processed by a worker
  - ***Single-row*** is beneficial when the matrix rows have distinct characteristics and require individual processing.
  - ***Multiple-row*** takes advantage of the parallel processing capabilities of GPUs, parallelly process more elements.

- ***Row-nonsplit/split*** dimension means whether a row of the sparse matrix is split and processed by different workers
  - ***Row-nonsplit.*** each row of the sparse matrix is kept intact during processing, means that there is no need to consider the accumulation of additional different rows.
  - ***Row-split.*** means dividing each row of the sparse matrix into smaller segments, which is beneficial from load balancing.

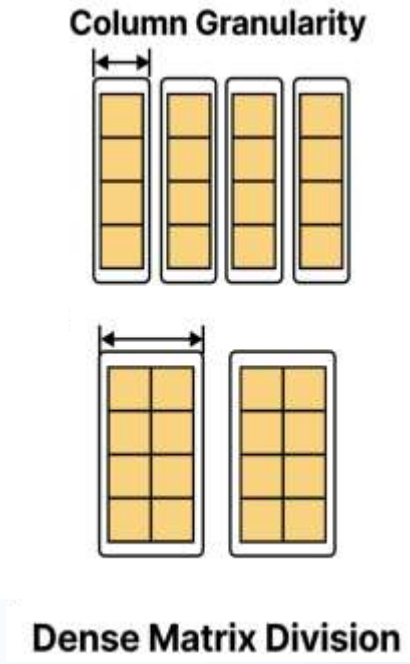


# Memory Access Space

## Dense matrix division (allocation)

The computations on different columns of a dense matrix are independent, and only the elements in the same column are accumulated.

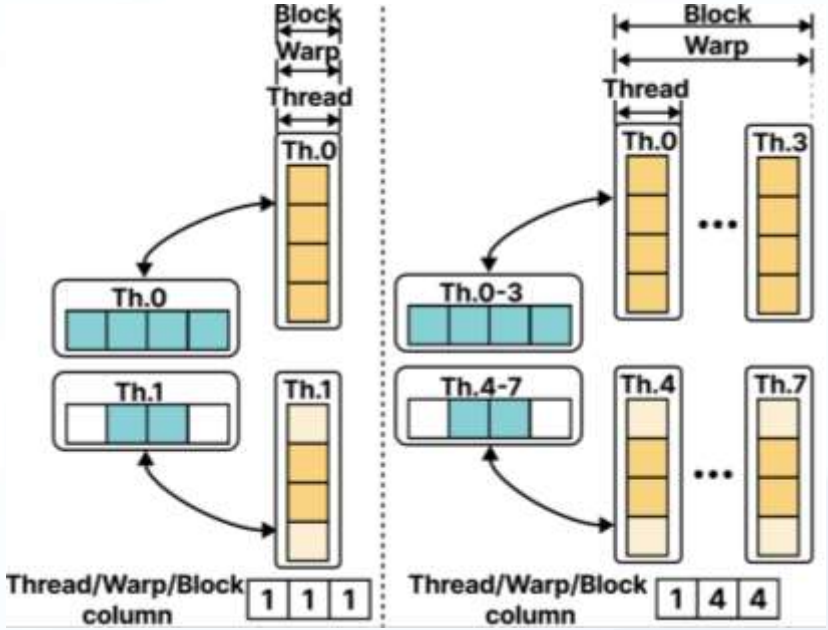
- The dense matrix is distributed to workers with a uniform pattern of data block in ***column granularity,*** because computations between columns are independently parallel.
- An important aspect is the ***adjustable column granularity*** of allocation. The number of columns grouped for a particular allocation unit can be changed
- We respectively group **1 and 2 columns** into a single allocation unit.



# Memory Access Space

## Integration between sparse and dense matrix

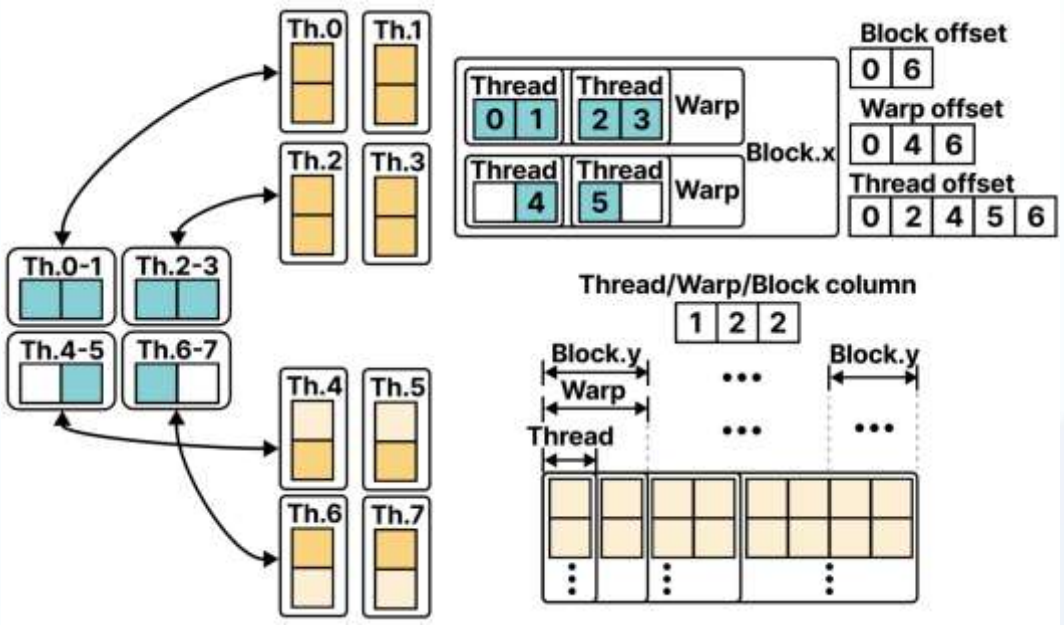
- When adjacent consecutive threads process different columns of dense matrix (In figure, warp/block column > 1),
- Threads load the same non-zero elements in sparse matrix but different columns of dense matrix, which forms the thread-groups (e.g., Th.0-3 and Th.4-7 in middle of Figure b).
- This creates an opportunity for data reuse, by pre-fetching data into shared memory or transferring data using the shuffle instruction.



# Memory Access Space

**In order to cover a variety of sparse matrix division, we propose offset abstraction.**

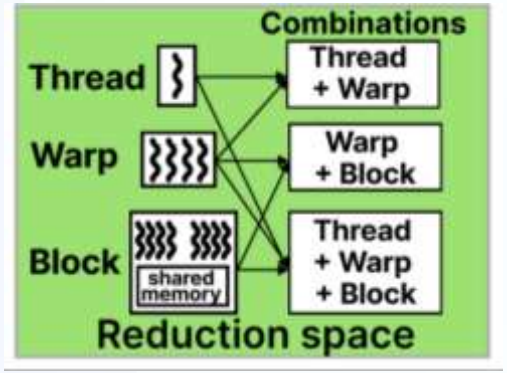
- ***Offset abstraction of adaptable division in sparse matrix.*** We use a hierarchical approach with thread/warp/block offset for sparse matrix division.
- ***Thread offset*** indicates the starting position of processing non-zeros range by thread-groups (e.g., 0 and 2 of Th.0-1 and Th.2-3).
- ***Offset abstraction*** can clearly locate the positions of different rows or row segments, effectively implementing data processing under single/multiple-row and row-split/nonsplit.

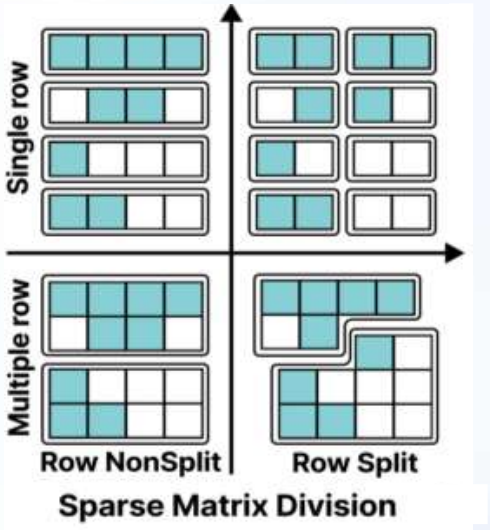


# Reduction Space

## Design Motivation

- Reduction algorithms can be used in combination, with consideration of the pruning sparsity levels and the GPU hierarchy
- Reduction algorithms need to be aligned with the memory access space, especially irregular sparse matrix allocation

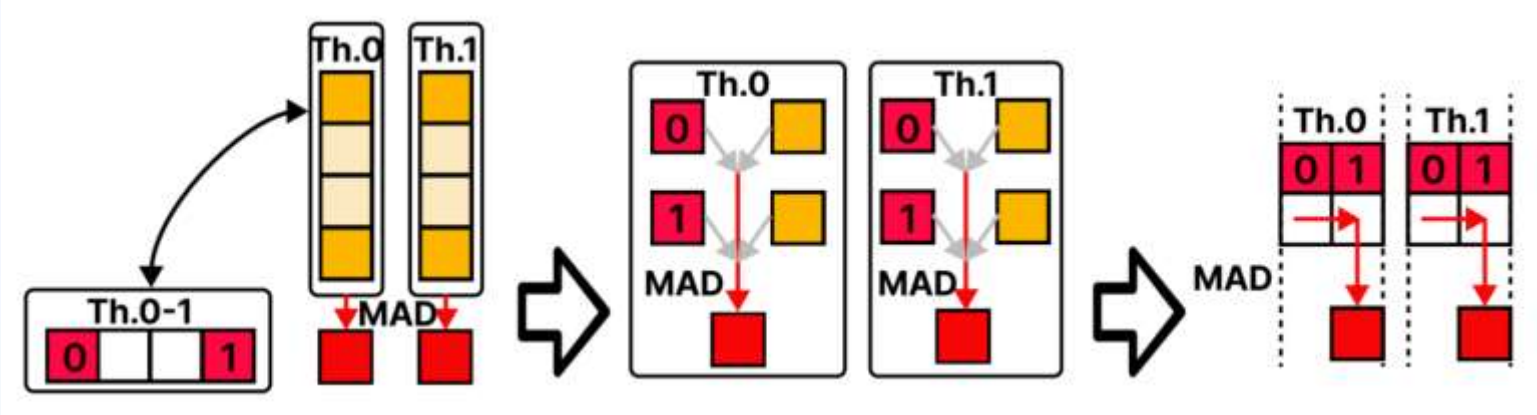


# Reduction Space

**We simplify figure expression of the reduction algorithm, Just for better understanding.**

- It depicts that two threads (Th.0 and Th.1) use Multiply-Add (MAD) operations to reduce the results of two adjacent columns corresponding with the same row.

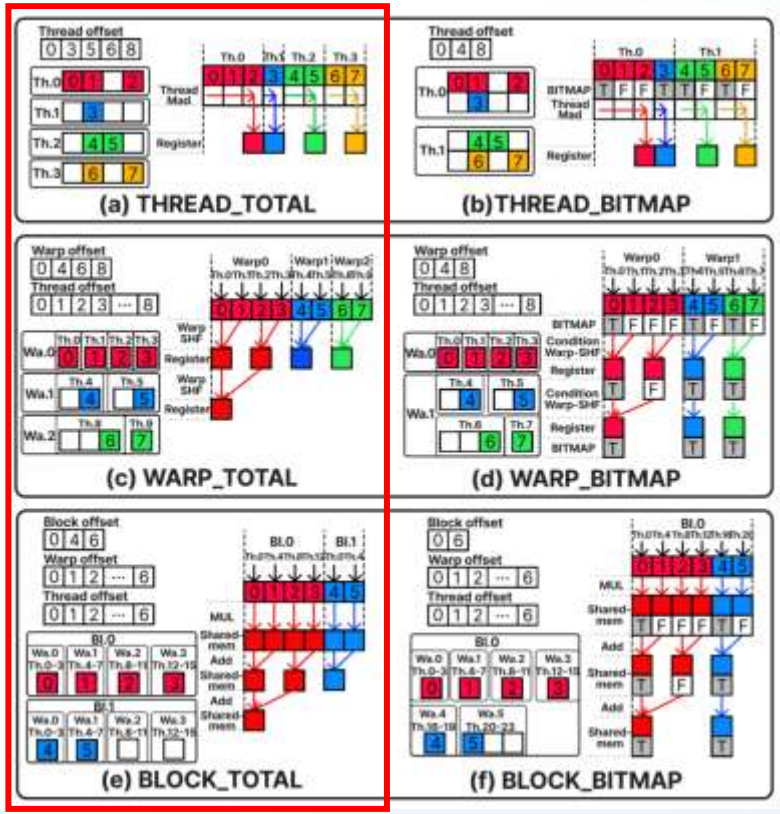


# Reduction Space

**Reduction algorithms are designed from GPU hierarchy, and aligned with single/multiple-row dimension. (TOTAL and BITMAP algorithm )**

In TOTAL, a GPU's parallel unit processes only elements in one row (single-row) and does not cross rows, so the results can be directly accumulated and written back

- Figure (a) shows that THREAD_TOTAL uses the MAD operation and registers to sequentially reduce the results of non-zero elements within the same row.
- Figure (c) shows that WARP_TOTAL uses the warp-level shuffle instruction and registers to reduce results processed by a warp.
- Figure (e) shows that BLOCK_TOTAL uses MUL and ADD instructions on block-level shared memory to reduce results.



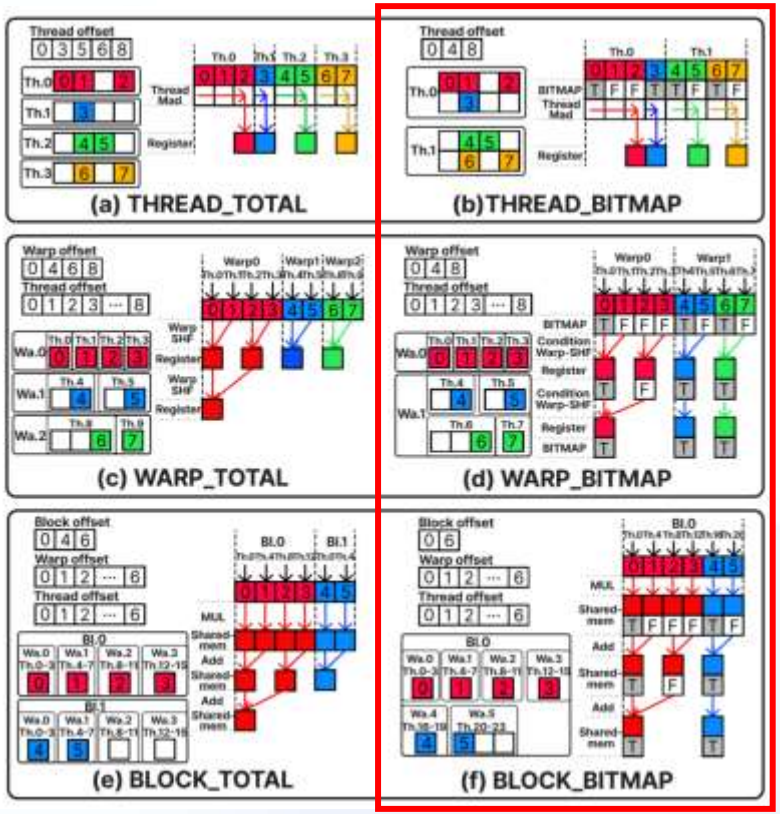
# Reduction Space

**Reduction algorithms are designed from GPU hierarchy, and aligned with single/multiple-row dimension. (**TOTAL and BITMAP algorithm **)**

In BITMAP, a GPU parallel unit needs to process elements across rows(multiple-row), so it is necessary to determine the row number of the element being processed to avoid incorrect accumulation of elements in different rows.

Each binary bit of BITMAP represents a non-zero element, T represents the starting position of the row, and F represents the positions within the row

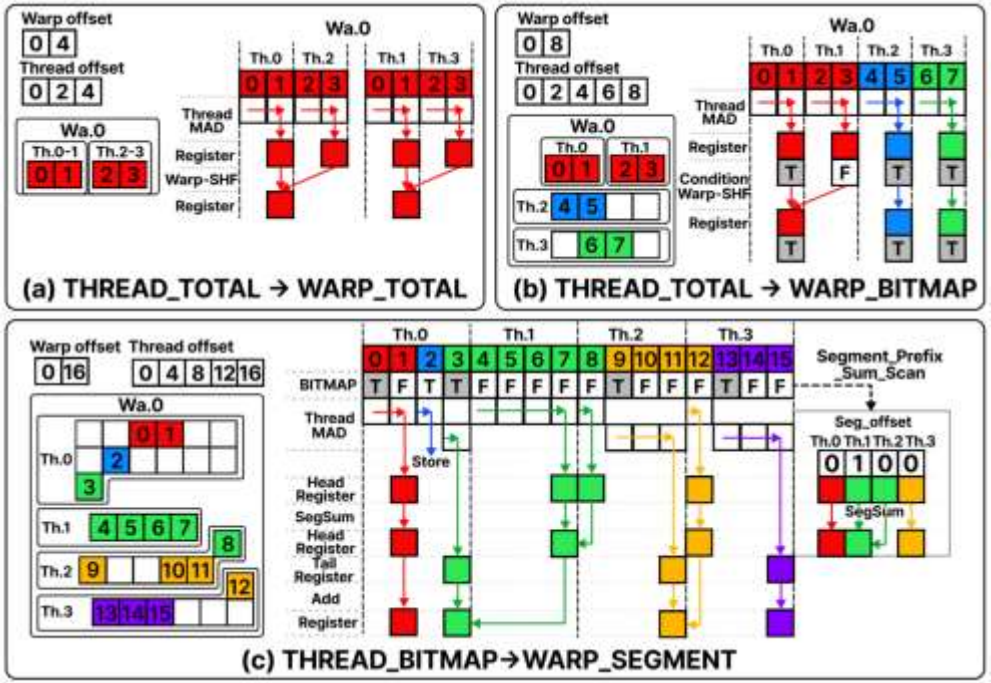
- Figure (b) shows that THREAD_BITMAP uses the BITMAP to determine whether a thread continues accumulating the current result. If the next step crosses rows, the result will be written back directly.
- In warp/block level (Figure (d,f)), multiple threads perform reduction in parallel



# Reduction Space

**Multi-level combination of reduction algorithms can be used on GPU for diverse sparsity levels, which are aligned with row-split/nonsplit dimension**

- When a row is assigned to different parallel units, multiple-level accumulation are required.
- ***Low-level TOTAL to High-level TOTAL***. Thread-level allocation is in single-row and row-split dimension, which corresponds with the THREAD_TOTAL to reduce results, and needs further warp-level reduction.
- ***Low-level TOTAL to High-level BITMAP,*** which corresponds with the WARP_BITMAP
- ***Low-level BITMAP to High-level SEGMENT.*** SEGMENT algorithm (Algorithm 2) is proposed to process the remaining intermediate results of low-level BITMAP.



# Automatic Implementation

- The cost model predicts the performance of strategies based on linear regression.

- For training, we use the characteristics of the sparse matrix (e.g., row numbers, column numbers, average non-zero elements per row and etc.) as the input for linear regression.
- We then use this model to fit the actual running performance of the sparse matrices

Table 2: Variable Definitions

| Variable | Type | Description |
|---|---|---|
| $x_{ij}$ | Binary | 1 if the division strategy $i$ of dense matrix and $j$ of sparse matrix is selected; 0 otherwise. |
| $c_{ij}$ | Real | Division cost associated with the $ij$-th division strategy. |
| $d_{ij}$ | Real | Post-processing (reduction) cost associated with the $ij$-th division strategy. |

$$\min_{i \in I, j \in J} \quad x_{ij}(c_{ij} + d_{ij}) \tag{1}$$

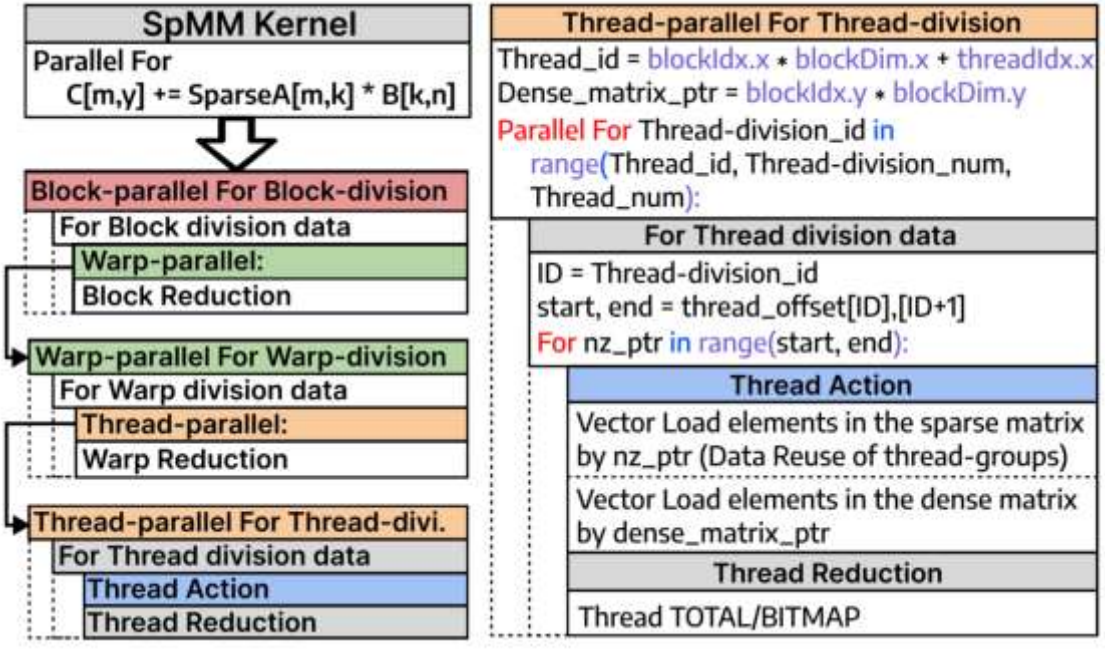
**subject to:**

$$x_{ij} = 0, \quad \text{if exceeding hardware limitations} \tag{2}$$

# Automatic Implementation

- Code generator recursively constructs parallel loops to optimize the memory access pattern.

- At the highest level, the SpMM kernel initiates the block-level parallelism and performs block reduction algorithms (if needed).
- Then move down to the warp and thread level.
- Each part generates the corresponding code fragments.



# Experiment

04

# Experimental Setup:

# The platform is NVIDIA A100 GPU, CUDA 12.1, include LLM pruned weights and SuiteSparse collection.

**Experimental Platform**

The NVIDIA A100 GPU is used, equipped with CUDA 12.1, ensuring an advanced and stable experimental environment.

**Software Environment**

The CUDA 12.1 version provides the latest GPU programming interface, optimizing parallel computing performance.

**Dataset Selection**

The datasets include LLM pruned weights and SuiteSparse matrices, covering a wide range of sparse patterns to verify the versatility of GeneralSparse.

**Comprehensiveness of the Experiment**

The datasets cover practical applications and benchmark tests, ensuring the practical value and academic reference significance of the experimental results.

# Baseline

1. Library:
   - (i) CuSPARSE (v12.1), which inputs sparse matrices in CSR format and offers three parameter choices, with the best result chosen.

2. Artificial designs and auto-tuners:
   - (i) ASpT, which uses column sorting and adaptive tiling to partition sparse matrices.
   - (ii) Sputnik, a well-designed sparse library for matrices in DL.
   - (iii) DgSPARSE, as the auto-tuner, offers eight algorithm choices, with the best result chosen.
   - (iv) FlashLLM, which uses tensor core to accelerate sparse weight matrices in DL.
   - (v) SparTA, which supports structured sparsity on tensor core.
   - (vi) TC-GNN, which fits the sparse GNN workload on dense TCs.
   - (vii) DTC-SpMM, which is tailored to harness TCs.

3. Sparse compiler technologies:
   - (i) TACO, which supports the code generation of SpMM with the default schedule.
   - (ii) SparseTIR, which provides a sparse tensor compilation abstraction.

# Performance: 20.82x speedup on LLM pruning weights, 6.39 to 7.46x speedup on SuiteSparse, 2.33x speedup on end-to-end LLM inference

### LLM pruning weight

GeneralSparse's SpMM operation on LLM pruning weights achieved an acceleration effect of up to 20.82 times, significantly improving the inference speed.

### SuiteSparse matrix

For the SuiteSparse matrix collection, our method achieves speedup ranging from 6.39 to 7.46 times, demonstrating strong adaptability to a variety of sparsity.

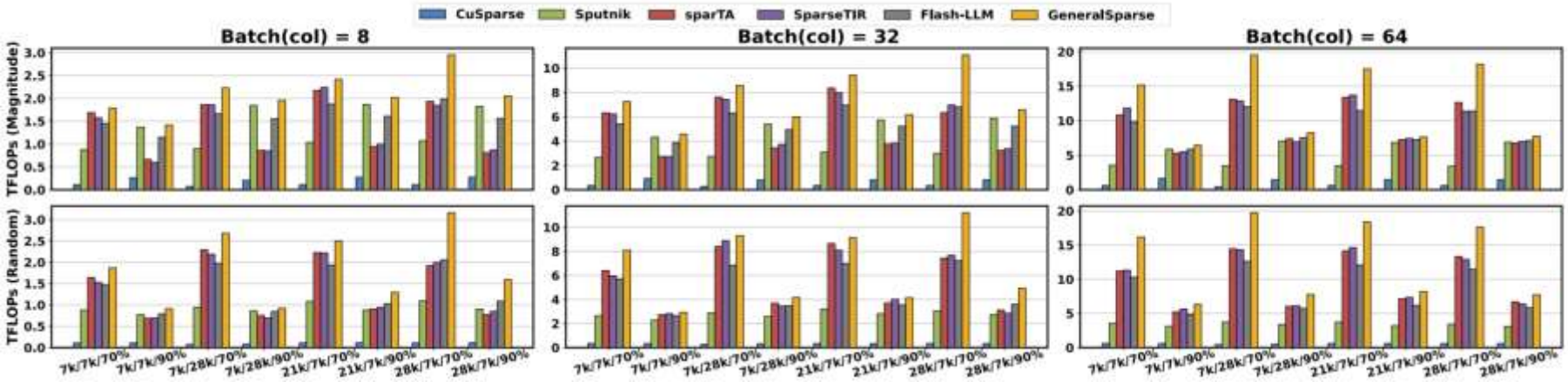
### End-to-end model

Overall, GeneralSparse accelerates the end-to-end LLM reasoning process by 2.33 times, proving its high efficiency in practical applications.

### Performance improvement

By optimizing memory access and reduction strategies, GeneralSparse not only improves the execution efficiency of SpMM, but also enhances the utilization of GPU resources.

# Performance of Pruned weight matrix




(1) Notably, GeneralSparse achieves the average speedup of 17.15/19.14/20.82× at 8/32/64 batch size over cuSPARSE.

(2) GeneralSparse achieves performance improvements of average 1.84/1.57/1.30× at 8 batch size on A100 over Sputnik, sparTA and Flash-LLM (2.24/1.37/1.31× at 32 and 3.37/1.27/1.38× at 64 batch size). Other methods do not support half-precision computation on GPU.

(3) Average speedup of 1.21/1.34/1.33× at 8/32/64 batch size over SparseTIR.

# Performance of Suitesparse matrix

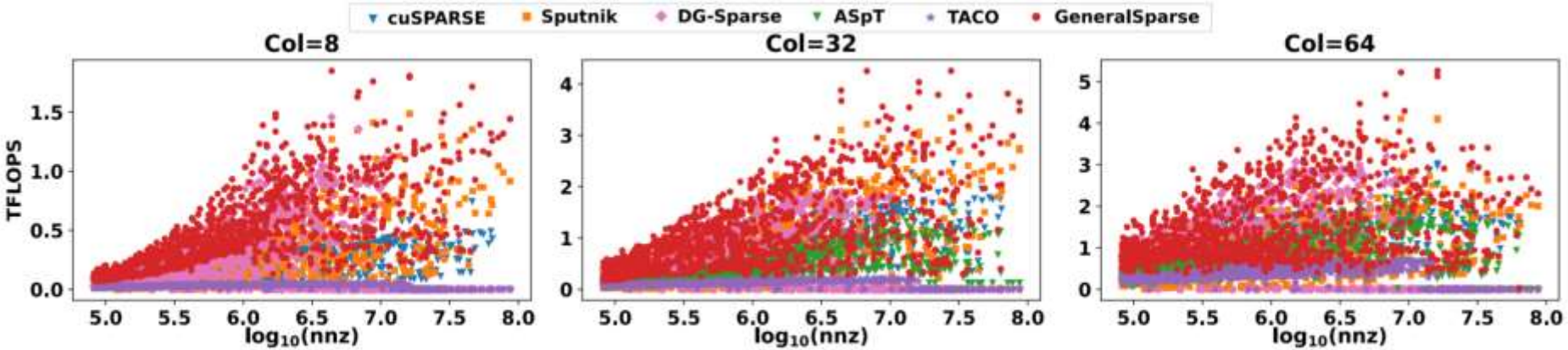



Figure 15: Kernel Performance on SuiteSparse matrix.

(1) Notably, GeneralSparse achieves the average speedup of 6.39/4.38/7.46× over cuSPARSE at 8/32/64 batch size.

(2) GeneralSparse achieves performance improvements of average 2.32/1.37/N.A.× on A100 over Sputnik, DgSPARSE, and ASpT at 8 (1.22/1.20/7.69× at 32 and 1.29/1.23/2.15× at 64) batch size.

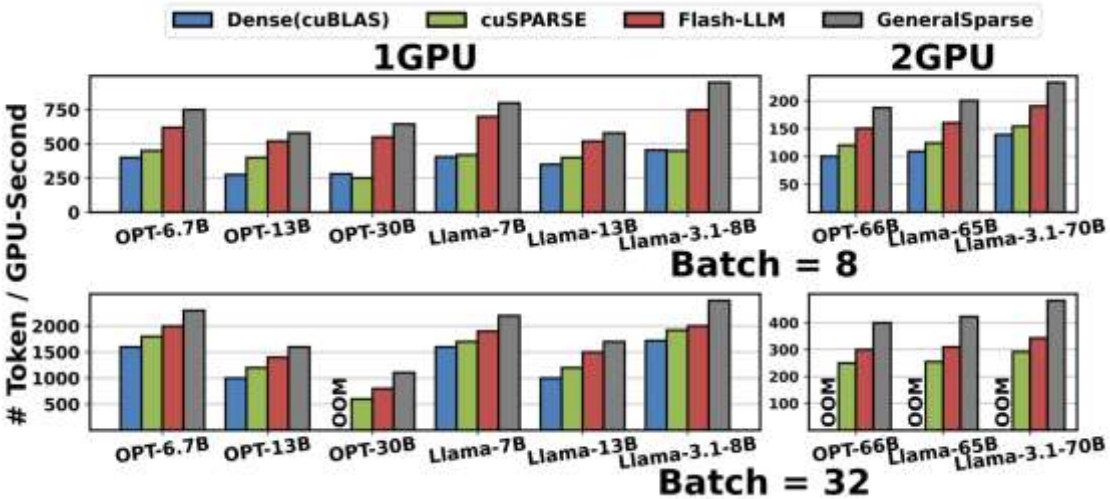
(3) GeneralSparse achieves the average speedup of 10.60/4.97/2.73× on A100 over TACO at 8/32/64 batch size. A primary factor is that TACO’ s design is not tailored for GPUs.

# End-to-end model throughput.

Speedups on models. (OPT-60B, Llama-65B and Llama-70B models).

(1) Notably, GeneralSparse achieves the average speedup of 2.33/1.58× over Dense(cuBLAS) at 8/32 batch size.

(2) GeneralSparse achieves performance improvements of average 1.50/1.20× at 8 batch size, 1.48/1.23× at 32 batch size over cuSPARSE/FlashLLM



# End-to-End Model Analysis

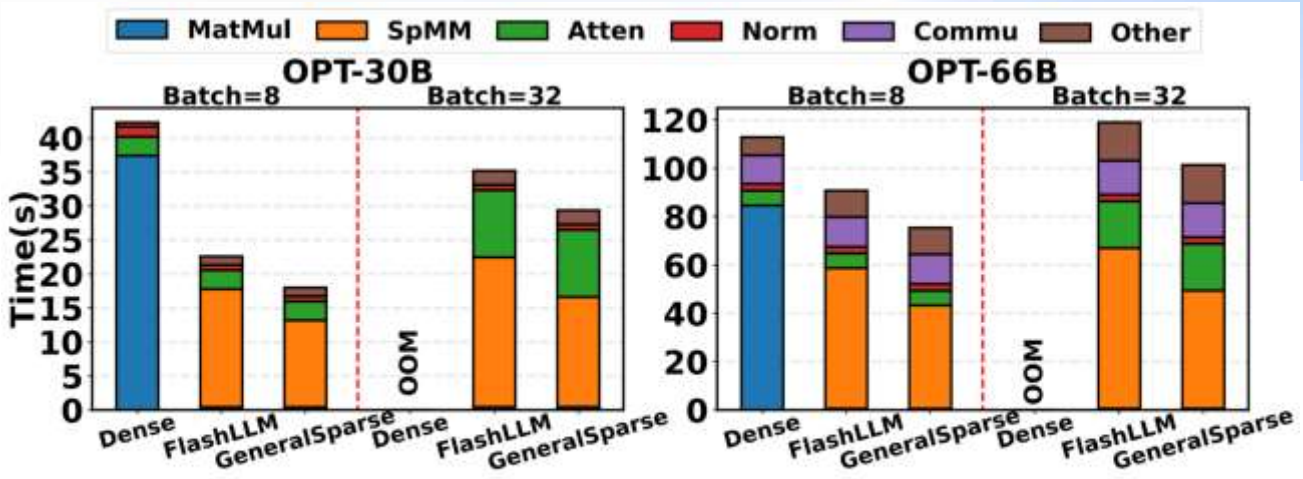


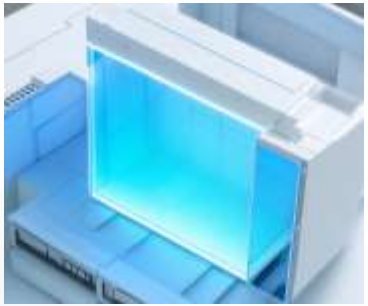

Figure 21: Time breakdown compared with other methods.

GeneralSparse can achieve lower inference latency mainly because we replace the dense MatMul with the more efficient SpMM.

# Related Work

05

# SpMM Optimization: cuSPARSE Library, Manual Design (Sputnik), Tensor Core(SparTA)



### cuSPARSE Library

It is a sparse matrix operation library officially provided by CUDA. It supports various sparse matrix formats and is applicable to SpMM calculations on GPUs



### Manual Design

Manually-optimized SpMM implementation for specific sparse patterns. It has weak versatility and self-adaptation capabilities.



### Tensor Core

Taking advantage of the hardware characteristics of Tensor Core, optimize the dense calculations in SpMM.



### Comparative Analysis

Although the above-mentioned methods have their respective advantages, when dealing with the diverse sparse patterns after pruning in LLMs, performance bottlenecks still exist. GeneralSparse aims to provide a more versatile solution

# Large language model acceleration

Accelerating LLMs has become a critical research focus due to their growing computational demands.

Existing efforts, such as FasterTransformer, DeepSpeed, and FlashAttention, have optimized various components of LLM inference, including attention mechanisms and scheduling strategies.

By integrating seamlessly with existing LLM acceleration frameworks, GeneralSparse enhances the overall efficiency of sparsified LLM inference.



# Conclusion & Future Work

06

**Conclusion**:

The performance bottleneck of SpMM under diverse sparsity has been addressed through abstract and automated design.



**The Power of Abstract Design**

GeneralSparse, through high - level abstraction, uniformly handles various sparse patterns, significantly enhancing SpMM performance.



**Automated Optimization Strategy**

The cost model and code generator work in tandem to automatically select the optimal execution strategy, maximizing GPU resource utilization.

Code available at https://github.com/Wangyaoyuu/GeneralSparse


Thanks All Co-authors!

Xiao Guo, Junmin Xiao, De Chen, Guangming Tan

# THANKS

Contact information:
wangyaoyu21s@ict.ac.cn