GPU-TOPSIS: A Complete Vectorized and Parallel Reformulation of the TOPSIS Method for Large-Scale Multi-Criteria Decision Making

Latifa Boubekri , Hassnae Aberkane , Mohammed Chaouki Abounaima * , Loubna Lamrini

Posted Date: 11 March 2026

doi: 10.20944/preprints202603.0836.v1

Keywords: multi-criteria decision making (MCDM); TOPSIS; parallel computing; graphics processing unit (GPU); high-performance computing (HPC); CUDA; CuPy; PyTorch; TensorFlow; big data; performance evaluation

 doi:10.20944/preprints202603.0836.v1

Article

GPU-TOPSIS: A Complete Vectorized and Parallel Reformulation of the TOPSIS Method for Large-Scale Multi-Criteria Decision Making

Latifa Boubekri, Hassnae Aberkane, Mohammed Chaouki Abounaima * and Loubna Lamrini

Laboratory of Intelligent Systems and Applications, Faculty of Sciences and Techniques, Sidi Mohammed Ben Abdellah University, Fez, Morocco

* Correspondence: medchaouki.abounaima@usmba.ac.ma

Abstract

The TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) method is one of the most widely used multi-criteria decision-making (MCDM) approaches in industrial, financial, and scientific fields. However, its sequential computational cost of O(m×n), where m denotes the number of alternatives and n the number of criteria, becomes prohibitive when decision matrices have several million rows. To overcome this limitation, we propose GPU-TOPSIS, a fully vectorized and parallel reformulation of TOPSIS based on tensor execution on graphics processing units (GPUs), whose main contributions are: (i) a formally correct reformulation of TOPSIS as a GPU tensor pipeline preserving mathematical fidelity to the original method; (ii) a two-pass fragment-processing algorithm guaranteeing exact mathematical equivalence with monolithic TOPSIS, while reducing the memory footprint from O(m×n) to O(m_t×n); (iii) Three independent implementations on CuPy, PyTorch, and TensorFlow ensure the framework's portability and genericity. Experimental evaluations on real data from the Amazon Products 2023 dataset, using matrices of up to 200 million alternatives (via the 2-pass formulation), demonstrate speedups of up to 4.75× compared to the reference CPU implementation (NumPy). A perturbation sensitivity analysis of the criteria weights and cross-backend consistency tests confirms that GPU acceleration fully preserves robustness and decision reliability, making GPU-TOPSIS a practical, open, and reproducible solution for large-scale multi-criteria decision making in Big Data environments.

Keywords: multi-criteria decision making (MCDM); TOPSIS; parallel computing; graphics processing unit (GPU); high-performance computing (HPC); CUDA; CuPy; PyTorch; TensorFlow; big data; performance evaluation

Introduction

Multi-criteria decision-making (MCDM) plays a central role in modern information systems. It involves evaluating a large number of alternatives against potentially conflicting criteria [1]. Its applications now extend to finance [2], healthcare [3], engineering [4], supply chain management [5], environmental assessment [6], and several other fields. Among the many available MCDM methods, TOPSIS stands out for its simplicity, geometric interpretability, and robust rankings [7].

The classic sequential implementation of TOPSIS suffers from increasing computational cost as the number of alternatives reaches the order of a million [8]. Analysts are forced to artificially reduce the dataset size, risking exclusion of alternatives that could affect the final decision. Most tools only support matrices of modest size [9], making them unsuitable for Big Data environments.

High-performance computing (HPC), and in particular the increasing use of graphics processing units (GPUs), offers a concrete solution to these scaling limitations. GPUs are based on massively

 doi:10.20944/preprints202603.0836.v1

parallel architectures, optimized for the execution of vectorized numerical operations [10]. The emergence of the CUDA platform and the development of high-level Python libraries such as CuPy [21], PyTorch [12], and TensorFlow have considerably facilitated access to GPU programming without requiring low-level expertise.

In this article, we present GPU-TOPSIS, a complete and vectorized reformulation of TOPSIS specifically designed for GPU execution. This contribution follows a progressive approach: our previous work proposed a parallelized MCDM filter in shared memory via OpenMP and then a distributed version of TOPSIS based on MapReduce [29]. GPU-TOPSIS takes a further step by exploiting the massive parallelism of modern GPUs, enabling the near real-time processing of decision matrices containing tens of millions of alternatives, without artificial data reduction.

Three version variants were developed: CuPy-TOPSIS, PyTorch-TOPSIS, and TensorFlow-TOPSIS, each reformulating the TOPSIS steps as a tensor pipeline running on CUDA-compatible hardware. Unlike previous work on GPU-TOPSIS [31], which remained application-domain specific and validated on modestly sized datasets, our framework is general-purpose and has been validated on millions of alternatives (up to 200 million) from real-world e-commerce data, with explicit numerical stability analysis on three distinct GPU backends.

Experimental evaluations confirm significant performance gains, with speedups of up to 4.75× compared to the CPU-based NumPy benchmark, while maintaining ranking consistency and numerical stability. A sensitivity analysis of criterion weight perturbations and sharding robustness tests completes the demonstration.

Unlike previous GPU-based implementations of TOPSIS, which often focus on specific application domains or limited datasets, this work proposes a general computational framework for large-scale multi-criteria decision-making based on GPU tensor pipelines. The proposed approach not only accelerates the classical TOPSIS workflow but also introduces a scalable computation model capable of handling decision matrices that exceed GPU memory capacity through a mathematically consistent fragmentation strategy.

In this context, we introduce GPU-TOPSIS, a fully vectorized GPU implementation of the TOPSIS method capable of processing decision matrices containing up to 200 million alternatives, thereby enabling large-scale multi-criteria decision-making on commodity GPU hardware. The original contributions of this work can be summarized as follows.

First, GPU-TOPSIS provides a fully vectorized reformulation of the TOPSIS method in which each stage of the decision pipeline—normalization, weighting, computation of ideal solutions, Euclidean distance calculation, and proximity scoring—is expressed as tensor operations executed directly in GPU memory while strictly preserving the original mathematical formulation.

Second, a two-pass fragment processing algorithm is introduced. Property 1 formally demonstrates that this formulation is a lossless generalization of monolithic GPU-TOPSIS: when k=1 it reduces exactly to the standard formulation, whereas for k>1 it guarantees identical rankings while reducing the memory footprint from $O(m \times n)$ to $O(m_t \times n)$. This design allows the processing of decision matrices whose size exceeds both GPU VRAM and host RAM capacities.

Third, three independent implementations based on CuPy, PyTorch, and TensorFlow are developed, ensuring portability and interoperability across the Python GPU ecosystem.

Fourth, a comprehensive experimental evaluation is conducted on real-world data from the Amazon Products 2023 dataset [39], covering matrices ranging from millions to 200 million alternatives. The evaluation includes sensitivity analyses to criterion-weight perturbations, inter-backend numerical consistency tests, and sharding robustness analyses, demonstrating that GPU acceleration preserves both numerical stability and decision reliability.

The remainder of this article is structured as follows. Section 2 presents the context and motivations, including a review of related work on parallel MCDM and the classical TOPSIS method. Section 3 describes the GPU-TOPSIS reformulation and its parallelization principles. Section 4 details the three GPU implementations. Section 5 presents the experimental results and robustness analyses. Finally, Section 6 concludes and outlines future research directions.

 doi:10.20944/preprints202603.0836.v1

Context and Motivation

High-Performance Computing and Parallel Processing on Gpus

The exponential growth of digital data generated by social networks, healthcare systems, business analytics, and scientific simulations has profoundly altered computing power requirements. Modern decision support systems must process large volumes of heterogeneous and dynamic information in real time [15,16]. This evolution has accelerated the transition to computing-intensive architectures where memory bandwidth, data proximity, and parallel execution capabilities are as critical as raw CPU power [17,18].

High-performance computing (HPC) environments address these needs by deploying scalable architectures capable of accelerating large-scale numerical workloads. MCDM methods, and TOPSIS in particular, rely on dense matrix operations—normalization, weighting, and distance calculation—whose mathematical structure naturally lends itself to parallelization. Parallel computing involves decomposing these operations into tasks that can be executed simultaneously; GPUs, with their thousands of lightweight cores specialized in tensor computing, are architecturally optimized for this class of problems [11,19,20].

Gpu-Compatible Tensor Libraries

While low-level interfaces like CUDA and OpenCL offer fine-grained hardware control, their complexity hinders the development of scientific applications. An ecosystem of high-level libraries has emerged to simplify GPU programming [19]. These libraries provide practical abstractions (tensors, automatic differentiation, and on-the-fly compilation) while generating optimized GPU kernels.

Three libraries are central to this work. CuPy is an open-source Python library that implements the NumPy API on GPUs via CUDA, enabling the migration of existing scientific code with minimal modifications. PyTorch [12], developed by Meta AI, offers dynamic tensor abstraction with an efficient CUDA backend and an autograding engine [22,23]. TensorFlow [13,24], developed by Google, provides a completely differentiable programming platform based on static computational graphs and the XLA compiler, optimized for production workloads.

These three libraries share a similar execution model: data is prepared on the CPU, transferred to GPU memory, processed via a sequence of tensor operations, and then returned to main memory for displaying the results. Table 1 summarizes their characteristics.

Table 1. Libraries for GPU acceleration.

Tool	Kind	Language	GPU support	Main use
CuPy [21]	Digital	Python	Yes (CUDA)	NumPy-compatible matrix processing
PyTorch [12]	DL / Digital	Python, C++	Yes (CUDA)	Tensors, dynamic graphs, prototyping
TensorFlow [13]	DL / Production	Python, C++	Yes (CUDA)	High-performance computing, deployment

Review of Related Works

Classical MCDM methods, such as TOPSIS, VIKOR, AHP, PROMETHEE, and ELECTRE, are widely used in fields as diverse as environmental monitoring, finance, healthcare, and supply chain

 doi:10.20944/preprints202603.0836.v1

management. Their popularity stems from their rigorous axiomatic foundation, their ability to produce consistent rankings, and the diversity of application domains in which they have been validated. However, conventional sequential implementations are struggling with the increasing volume and dimensionality of modern datasets. TOPSIS, with a complexity on the order of O(m×n), quickly becomes inefficient when faced with millions of alternatives.

Theoretical extensions have enriched classical MCDM to handle uncertainty. Fuzzy MCDM models, such as the one used by Das et al. to assess the anthropogenic impact on urban water quality [25], offer more flexible representations of imprecise measurements. Other extensions, such as neutrosophic decision systems or multi-fuzzy N-soft approaches [26], allow for addressing ambiguous situations. However, these theoretical advances do not eliminate the computational challenges posed by large datasets.

To address the need for scalability, several studies have explored distributed and parallel computing applied to MCDM. CPU-based approaches (MPI, OpenMP, or multi-thread architectures) have demonstrated encouraging results for tasks such as Pareto filtering or preference aggregation. Our research team previously proposed a parallel MCDM filter based on OpenMP in shared memory [28], offering good performance on systems with a small number of cores. A distributed version of TOPSIS based on MapReduce has also been developed [29], capable of handling high-dimensional data across multiple nodes, although its disk-oriented nature limits its application in real-time decision-making contexts. GPU-TOPSIS builds upon these foundations by taking a significant leap forward: where leverages a few dozen CPU cores and distributes computation across a cluster, GPU-TOPSIS mobilizes thousands of GPU cores in a single memory, eliminating the overhead of inter-node communication and disk access.

Regarding GPU approaches, Erlacher et al. demonstrated the advantages of CUDA-accelerated spatial analysis for uncertainty modeling in MCDM [30]. Lakshmi et al. presented a GPU-TOPSIS for Quality of Service (QoS)-based web service selection, showing significant speedups. Kumar et al. evaluated GPU cloud computing instances via TOPSIS. More recently, Swetha and Karpagam introduced a GPU-accelerated improved ideal reference method (I-RIM) for real-time web service selection. This work confirms the potential of GPUs for a wide class of ranking-based MCDM methods.

However, existing GPU or distributed implementations remain largely application-domain specific and do not offer a general, fully vectorized TOPSIS framework capable of processing very large datasets consistently and reproducibly. None of them examines the consistency of rankings between CPU and GPU implementations or evaluates numerical stability under large-scale batch processing. Compared to [31], in particular, GPU-TOPSIS offers a general-purpose framework validated on millions of alternatives with real-world e-commerce data, an explicit analysis of numerical stability on three GPU backends, and a sharding strategy to overcome CPU and GPU memory limitations.

Table 2. Comparison between GPU-TOPSIS and the closest related works.

Work	Parallelism	Max size	Speedup	Genericity	Numerical stability
Lamrini 2022	OpenMP/CPU	~10^5	N / A	MCDM filter	Not rated
Lamrini 2023	MapReduce	~10^6	N / A	TOPSIS	Not rated
Lakshmi 2020	GPU/CUDA	Modest	Partial	QoS web	Not rated
GPU-TOPSIS (this work)	GPU/CUDA (3 backends)	~200×10^6	4.75×	General	Evaluated ($\varepsilon<10^{-6}$)

 doi:10.20944/preprints202603.0836.v1

As summarized in Table 2, GPU-TOPSIS is the only approach combining full genericity, GPU parallelism across three backends, scalability up to 200 million alternatives, and a formally evaluated numerical precision ($\varepsilon < 10^{-6}$).

To our knowledge, no previous work has demonstrated the scalability of TOPSIS on GPU architectures for decision matrices containing hundreds of millions of alternatives.

The Basic Topsis Method

Fundamental Principles

The TOPSIS method (Technique for Order Preference by Similarity to Ideal Solution), introduced by Hwang and Yoon [34], is one of the most widespread multi-criteria decision-making approaches. Given a set of alternatives $A_1,\ldots, A_m$ evaluated according to criteria $C_1,\ldots, C_n$, the method constructs a decision matrix M, normalizes and weights the criteria, and then ranks the alternatives according to their relative proximity to two reference solutions: the Positive Ideal Solution (PIS), representing the theoretical optimal performance for each criterion, and the Negative Ideal Solution (NIS), representing the most unfavorable theoretical performance.

TOPSIS is valued for its intuitive geometric interpretation, the absence of pairwise comparisons, and its relatively low computational cost compared to more complex methods such as ELECTRE or PROMETHEE [34]. It has been successfully applied in many fields: supply chain management, industrial production systems, marketing, health, resource allocation, human resource management, and social network analysis [35,36].

Topsis Sequential Algorithm

Consider a decision problem involving m alternatives and n criteria. The decision matrix $M=[x_{ij}]\{m\times n\}$ contains the evaluations of the alternatives, where x_{ij} represents the performance of alternative A_i according to criterion C_j. The decision-maker's preferences are encoded in the weighting vector $W = [w_1, \ldots, w_n]$, where w_n reflects the relative importance given to criterion C_j. The criteria are partitioned into two disjoint sets: J^+ denotes the set of benefit criteria (to be maximized) and J^- the set of cost criteria (to be minimized). The algorithm proceeds in seven successive steps.

Step 1 — Construction of the decision matrix M and the criterion weight vector W:

$$M = [x_{ij}]_{m\times n}, 1 \le i \le m, 1 \le j \le n \quad (1)$$

$$W = [w_1, \ldots, w_n], with: \sum_{i=1}^{n} w_i = 1 \quad (2)$$

w_i is the weight reflecting the relative importance of criterion C_j.

Step 2 — Vector normalization (the most commonly used):

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{k=1}^{m} x_{kj}}} \quad (3)$$

We obtain the normalized matrix $R = [r_{ij}]$.

Step 3 — Weighted Normalized Matrix :

$$v_{ij} = w_j \times r_{ij} \quad (4)$$

We obtain the weighted matrix $V = [v_{ij}]$.

Step 4 — Determine the ideal solutions: A^+ and A^-

For $j \in J^+$ (benefit: the criteria to maximize):

$$A^+[j] = \max_i v_{ij}, A^-[j] = \min_i v_{ij} \quad (5)$$

For $j \in J^-$ (cost: the criteria to be minimized):

$$A^+[j] = \min_i v_{ij}, A^-[j] = \max_i v_{ij} \quad (6)$$

Step 5 — Calculate the distances to the (Euclidean) ideals:

 doi:10.20944/preprints202603.0836.v1

For each alternative A_i, we calculate its distance to the ideal solutions.

$$D_i^+ = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_j^+)^2} \quad (7)$$

$$D_i^- = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_j^-)^2} \quad (8)$$

Step 6 — Calculate the proximity coefficient (TOPSIS score):

For each alternative A_i, we calculate its score, which will be used for its final ranking.

$$S_i = \frac{D_i^-}{D_i^+ + D_i^-} \quad (9)$$

- $S_i \in [0,1]$
- The larger S_i is, the better the alternative A_i.

In equation 9, we propose adding a constant numerical stabilization parameter $\varepsilon = 10^{-12}$ to the denominator to avoid division by zero in degenerate cases where $D^+ = D^- = 0$. This numerical adaptation does not affect the results in normal cases of use (ε has no effect when $D^+ + D^- > 0$).

$$S_i = \frac{D_i^-}{D_i^+ + D_i^- + \varepsilon} \quad (10)$$

Step 7 — Rank the alternatives:

The alternatives are ranked in descending order of S_i. The alternative with the highest score is considered the most preferable.

Computational Complexity

Let m be the number of alternatives and *n* the number of criteria. The complexity of CPU-TOPSIS is decomposed as follows [37,38]: (i) normalization and weighting, each element of the matrix is processed a constant number of times, i.e., O(m×n); (ii) calculation of distances, each alternative requires the calculation of two Euclidean distances on *n* criteria, i.e., O(m×n); (iii) calculation of proximity scores, a single pass over the set of alternatives: O(m); (iv) sorting, ranking the scores in descending order: O(m×log m). The total time complexity is therefore expressed by equation (11):

$$T_{\text{CPU-TOPSIS}}(m, n) = O(m \times n) + O(m \times \log m) \approx O(m \times n) \quad (11)$$

When the number of alternatives *m* and the number of criteria *n* are of the same order of magnitude ($m \approx n$), the complexity O(m×n) can be approximated by $O(n^2)$. However, in typical Big Data scenarios, $m \gg n$: the complexity is then strictly linear in *m* for a fixed *n*, meaning that the computation time increases proportionally to the volume of data. It is precisely this linear growth—and the absolute volume it represents when m reaches tens of millions—that justifies the use of GPU parallelization, capable of distributing this work across thousands of cores simultaneously.

Gpu-Topsis: A New Parallel Extension of Topsis

The increasing availability of high-performance GPUs has transformed the execution of computationally intensive algorithms. GPU-TOPSIS reformulates each step of the TOPSIS method as tensor operations executed on GPU hardware, ensuring perfect ranking consistency with the original TOPSIS method while drastically reducing execution times. Three implementations have been developed using CuPy, PyTorch, and TensorFlow.

Gpu-Topsis Algorithm

Algorithm 2 below presents the general formulation of GPU-TOPSIS with fragment processing, in which each step of the classical TOPSIS pipeline is reformulated as a vectorized tensor operation executed in GPU memory, while the fragmentation strategy enables the processing of decision matrices exceeding not only the VRAM capacity of the GPU, but also the RAM of the host system, by partitioning the data into successive and independent fragments loaded on demand from persistent storage.

 doi:10.20944/preprints202603.0836.v1

Algorithm 2 — GPU-TOPSIS with Sharding (Two-Pass Formulation)

Input:
shards = $\{S_1,\ldots,S_k\}$: partitions of the decision matrix
$W \in \mathbb{R}^n$: criteria weights ($\Sigma\, w_j = 1$)
criteria_types ∈ {"benefit", "cost"}^{n}
backend ∈ {CuPy, PyTorch, TensorFlow}
$\varepsilon = 10^{-12}$
Output:
L: globally ranked alternatives
Initialization:
W_gpu ← to_GPU(W)
scores_global ← ∅
indices_global ← ∅
offset ← 0

PASS 1: Global statistics (CPU)

---

sq_sums ← zeros(n)
for t = 1 … k do
M ← load_CSV(S_t)
sq_sums ← sq_sums + col_sum_squares(M)
end for
norms ← sqrt(sq_sums)
Initialize PIS and NIS according to the criteria_types
for t = 1 … k do
    M ← load_CSV(S_t)
    R ← M / norms
    V ← R ⊙ W
  Update PIS and NIS using column extrema of V
end for
norms_gpu ← to_GPU(norms)
PIS_gpu ← to_GPU(PIS)
NIS_gpu ← to_GPU(NIS)

PASS 2: Fragment GPU processing

---

for t = 1 … k do
    M_gpu ← to_GPU(load_CSV(S_t))
    m_t ← rows(M_gpu)
    R ← M_gpu / norms_gpu
    V ← R ⊙ W_gpu
    D^+ ← row_sqrt_sum((V − PIS_gpu)²)
    D^- ← row_sqrt_sum((V − NIS_gpu)²)
    S ← D^- / (D^+ + D^- + ε)
    scores_global ← concatenate(scores_global, to_CPU(S))
    indices_global ← concatenate(indices_global,[offset … offset + m_t − 1])
    offset ← offset + m_t
    free_GPU(M_gpu, R, V)
end for

PASS 3: Global ranking

---

scores ← concatenate(scores_global)
indices ← concatenate(indices_global)
L ← indices[descending_sort(scores)]
return L

 doi:10.20944/preprints202603.0836.v1

The proposed sharding algorithm performs two passes on the dataset. The first pass calculates and aggregates global statistics per column (normalization denominators, PIS, and NIS vectors) without loading the entire matrix into memory. The second pass uses these global statistics, consistent across the dataset, to calculate distances and proximity scores, thus guaranteeing complete mathematical equivalence with a monolithic TOPSIS execution, regardless of the sharding scheme used.

Topsis Complexity and Memory Footprint: Cpu, 1-Pass Gpu, and 2-Pass Gpu (Sharding)

The fundamental algorithmic cost of TOPSIS applied to a decision matrix $M \in \mathbb{R}^{n \times m}$ remains O(m×n), because the normalization, weighting, reduction (max/min for PIS/NISPIS), and distance calculation operations traverse the entire set of m × n values. On the CPU, the execution time therefore follows this growth, modulo the effects of vectorization and memory hierarchy.

On GPUs, these steps naturally translate into parallel kernels. Denoting p as the number of actually active computing units (useful utilization), and assuming good GPU utilization, the theoretical time is written as:

$$T_{GPU-TOPSIS}(m,n,p) = O(\frac{m \times n}{p}) + T_{overhead} \quad (12)$$

Where T overhead. This expression groups together CPU↔GPU transfers, kernel launch latency, and synchronization costs. It accurately describes regimes where the $\frac{m \times n}{p}$CPU dominates (often for very large volumes, e.g., m ≳ 10^6). Conversely, for smaller sizes (e.g., m ≲ 10^5), the overhead can become dominant and negate the GPU advantage: T_{GPU -TOPSIS} < T_{CPU-TOPSIS}.

Gpu-Topsis in 1-Pass (Monolithic)

In a **1-pass formulation**, the calculation is performed in a single phase, limiting data rereading and repeated transfers. The theoretical time required coincides with Equation 12. This option is generally the fastest when the data and the necessary intermediate tensors fit in both CPU and GPU memory.

$$T_{GPU,1-pass}(m,n,p) = O(\frac{m \times n}{p}) + T_{overhead} \quad (13)$$

In 1-pass, we calculate global statistics (normalization norms, PIS/NISPIS) and scores on all data loaded into memory in its entirety.

Gpu-Topsis in 2-Passes (Sharding)

When the matrix is too large to be loaded into VRAM all at once, the k-shard partitioning strategy becomes necessary **(**see algorithm 2 above). A correct formulation then proceeds in **two passes**:

1-Pass (global statistics): calculation of global norms (normalization denominators) and global PIS/NIS ideal vectors, ensuring consistent normalization-weighting across the entire dataset;

2-Pass (scores): shard-by-shard processing on GPU to calculate distances and proximity scores from global statistics.

Time can be summarized as follows:

$$T_{GPU,2-pass}(m,n,p,k) = 2 \times O(m \times n) + k \times O(\frac{m_t \times n}{p}) + k \times T_{overhead} \quad (14)$$

With :

- 2×O(m×n): two sequential CPU traversals in Pass 1
- overhead×n/p): k = fragments processed successively on GPU, each of m $_t$ lines distributed over p cores.
- k×T overhead: cumulative overhead over the k fragments

The term 2. O(m×n) corresponds to the construction of global statistics in pass-1 on the CPU (often constrained by I/O if shards are read from disk), while k×T overhead reflects the repetition of transfers and launches at each fragment. In return, this approach allows for memory scalability while

 doi:10.20944/preprints202603.0836.v1

guaranteeing mathematical equivalence with monolithic TOPSIS, regardless of the partitioning scheme.

Simplification:

By noting that overhead=m (the sum of all the fragments reconstitutes the complete matrix), we can verify the consistency:

$$k \times O(\frac{m_t \times n}{p}) = O(\frac{k \times m_t \times n}{p}) \approx O(\frac{m \times n}{p}) \quad (15)$$

Formula (14) is therefore consistent with monolithic GPU-TOPSIS at the global asymptotic level, while reflecting fragment execution at the operational level.

Limiting case k = 1:

When k = 1, formula (14) reduces to:

$$T_{GPU,2-pass}(m,n,p,1) = 2 \times O(m \times n) + O(\frac{m \times n}{p}) + T_{overhead} \quad (16)$$

This corresponds exactly to the monolithic GPU-TOPSIS with the CPU pass overhead, confirming that Property 1 below is consistent with the complexity analysis.

Gpu-Topsis in 1-Pass Sharding

To reduce the complexity of the two-pass GPU-TOPSIS scheme, we propose a more efficient "1-pass-shard" variant (see Figure 1(b)) by eliminating the global pre-aggregation cost of O(m×n) and performing, for each shard, the relative computation of statistics (norms, PIS/NIS) and scores in $O(m_t \times n/p)$ per shard. This acceleration, however, comes at the cost of approximation, since the statistics and scores are estimated locally on each shard rather than globally.

$$T_{GPU,1-pass-shard}(m,n,p,k) = O(\frac{m \times n}{p}) + k \times T_{overhead} \quad (17)$$

Memory footprint: a condition for the feasibility of the first pass and a motivation for the second pass.

In float32 precision, each element occupies 4 bytes, resulting in a minimal footprint:

$$\text{Mem}(M) \approx 4m \times n \text{ bytes} \quad (18)$$

In practice, a monolithic GPU-TOPSIS also manipulates intermediate tensors (normalized R, weighted V, temporary buffers). The VRAM footprint can be usefully modeled by:

$$_{1\text{-pass}}\text{VRAM} \approx c \times 4m \times n \quad (19)$$

where c is typically between 2 and 4, depending on kernel merging and storage strategy (e.g., M, R, V, and buffers). The W, PIS, NIS vectors and global norms cost only O(n), negligible compared to m×n.

Numerical example (float32, n = 20 **)**

- if $m=10^7$, then $m \times n = 2 \times 10^8$ and Mem(M)≈ 0 , 8 GB ; with c = 3 , $_{1\text{-pass}}$VRAM ≈2.4 GB\ (excluding overheads).
- if $m=10^8$, then Mem(M)≈8 GB, and $_{1\text{-pass}}$VRAM ≈16 to 32 GB for c ∈ [2,4], which often exceeds the available VRAM.

Thus, the memory constraint directly determines the choice of formulation: when $_{1\text{-pass VRAM}}$ is compatible with the board, 1-pass is generally preferable (fewer reads, less repeated overhead). When this is not the case, 2-pass (sharding) is necessary; it reduces the VRAM footprint to that of an $m_t \times n$ shard:

$$_{2\text{-pass}}\text{VRAM} \approx c \times 4mt \times n + O(n) \quad (20)$$

at the cost of a second data traversal and repeated overhead on k fragments.

Comparative table of complexities and memories

-Illustrative hypotheses: n= 20 , p= 4096 , k=100 .

-Scenarios: A) $m=10^5 \Rightarrow m \times n = 2 \times 10^6$ and B) $m=10^7 \Rightarrow m \times n = 2 \times 10^8$

Table 3. Comparison of proposed TOPSIS formulations.

 doi:10.20944/preprints202603.0836.v1

Version	Time (form)	A: dominant calculus term	B: dominant calculus term	Dominant memory	Typical use
CPU-TOPSIS	$O(mn)$+ $O(m \times log\ m)$	2.0×10^6	2.0×10^8	RAM (monolithic or streaming)	Small/medium volumes, simplicity
GPU 1-pass	$\frac{0(mn)}{P} + T_{overhead}$	$\approx 4.88 \times 10^{2} + T_{overhead}$	$\approx 4.88 \times 10^{4} + T_{overhead}$	VRAM ≈ c . 4 min	Best choice if it fits in VRAM
GPU 1-pass shard	$\frac{0(mn)}{P} + kT_{overhead}$	$\approx 4.88 \times 10^{2} + 100T_{overhead}$	$\approx 4.88 \times 10^{4} + 100T_{overhead}$	VRAM per shard ≈ c 4 min	required if dataset > VRAM and norms, PIS/NIS & scores calculated locally on each shard
2-pass GPU	$0(mn) + \frac{0(mn)}{P} + k \times T_{overhead}$	$\approx 2.0 \times 10^{6} + 4.88 \times 10^{2} + 100 \times T_{overhead}$	$\approx 2.0 \times 10^{8} + 4.88 \times 10^{4} + 100 \times T_{overhead}$	VRAM per shard ≈ c 4 min	necessary if dataset > VRAM, global equivalence

As shown in Table 3, the 1-pass GPU formulation achieves the lowest overhead when the dataset fits in VRAM, while the 2-pass formulation becomes necessary beyond this threshold to guarantee global equivalence with CPU-TOPSIS at the cost of an additional data traversal.

Ultimately, if the decision matrix and its intermediate tensors fit in GPU memory, the 1-pass GPU-TOPSIS formulation (see Figure 1 (a)) is generally the most efficient, as it limits data rereading and avoids the repeated overhead of data transfer and kernel launches. However, as soon as VRAM constraints impose fragmented processing, the 2-pass GPU-TOPSIS formulation (sharding) becomes the reference solution: it guarantees the overall consistency of the statistics (norms, PIS/NIS) and therefore equivalence with monolithic TOPSIS, at the cost of a second traversal of the dataset and overhead proportional to the number of fragments.

 doi:10.20944/preprints202603.0836.v1

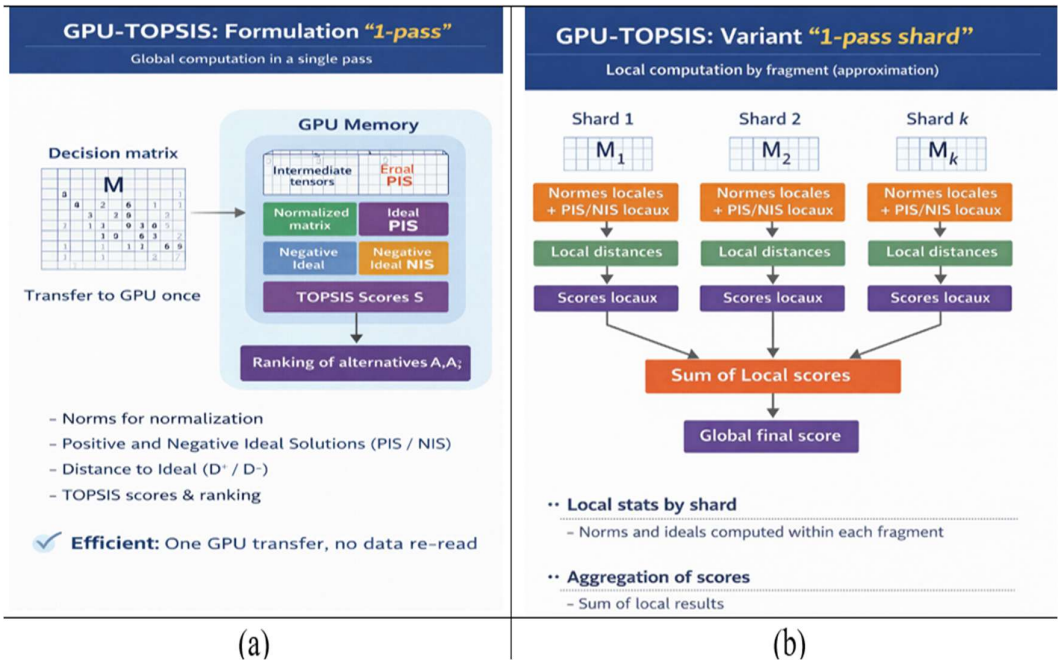

Figure 1. GPU-TOPSIS 1-pass and 1-pass shard flow diagram.

Property 1 (Generalization of monolithic GPU-TOPSIS)

Let k be the number of fragments in Algorithm 2. When k = 1, the two-pass fragmentation algorithm reduces exactly to the standard monolithic GPU-TOPSIS, in which the normalization denominators, the PIS and NIS vectors are computed on the complete decision matrix $M \in \mathbb{R}^{m \times n}$, and the entire TOPSIS pipeline is executed on the GPU in a single pass. Moreover, for any $k \geq 1$, the two-pass formulation produces rankings that are mathematically identical to those of monolithic GPU-TOPSIS, since Pass 1 guarantees the global consistency of norms_global, PIS_global, and NIS_global regardless of the chosen partitioning scheme. The two-pass fragmentation algorithm is therefore a lossless generalization of monolithic GPU-TOPSIS, introducing no approximations and preserving total mathematical fidelity to the original TOPSIS formulation of Hwang and Yoon for any value of k.

When k = 1, formula (14) reduces to:

$$T_{GPU,2-pass}(m,n,p,1) = O(m \times n) + O(\frac{m \times n}{p}) + T_{overhead} \quad (21)$$

This corresponds exactly to the monolithic GPU-TOPSIS with the CPU pass overhead of Pass 1, confirming that Property 1 is consistent with the complexity analysis.

Note on finite-precision arithmetic: The above equivalence holds in exact arithmetic. In float32 floating-point arithmetic, parallel GPU reductions are non-deterministic and non-associative due to rounding at machine epsilon ($\varepsilon_{mach} \approx 1.2 \times 10^{-7}$). In practice, this may introduce score discrepancies on the order of $\varepsilon_{mach} \times k$ between the 1-pass and 2-pass formulations for large k. The empirical impact of this effect on decision quality is analyzed in Section 5.

Corollary 1 (Memory independence)

Algorithm 2 requires that only one fragment S_t reside in memory at any given time, both during Pass 1 and Pass 2. Therefore, the algorithm supports decision matrices whose total size exceeds both the available GPU VRAM and the host system's RAM capacity, provided that individual fragments can be sequentially loaded from persistent storage. The memory footprint of Algorithm 2 is thus $O(m_t \times n)$ rather than $O(m \times n)$, where $m_t = \max_{t=1,\ldots,k}$ and $rows(S_t)$ denotes the size of the largest fragment.

Outline of Proof of Property 1

 doi:10.20944/preprints202603.0836.v1

Since the col_sum_squares operator is additive on disjoint row partitions, the following identity applies:

$$norms\_global_j = \sqrt{\sum_{t=1}^{k} \sum_{i \in S_t} x_{ij}^2} = \sqrt{\sum_{i=1}^{m} x_{ij}^2} \quad (22)$$

Equation 22 holds for any partitioning $[S_1, \ldots, S_k]$ of the set of row indices $\{1, \ldots, m\}$. Similarly, since the max and min operators are associative and commutative on disjoint sets, the vectors PIS_global and NIS_global calculated incrementally in Pass 1 are the same as those obtained by direct reduction on the complete matrix. It follows that the weighted normalized matrix V and the Euclidean distances D^+ and D^- calculated in Pass 2 are pointwise identical to their monolithic counterparts, and the final ranking L is therefore invariant to the choice of k and the partitioning scheme.

Experimental Evaluation

Dataset: Amazon Products 2023

The Amazon Products 2023 dataset is the primary experimental tool for this work. It is a large, publicly available collection, aggregated from Amazon platform catalogs and distributed by product category [39]. Its tabular structure, rich attributes, and sheer volume—tens of millions of products across 32 categories—make it a natural and demanding testing ground for large-scale multi-criteria decision-making methods. In the context of e-commerce decision support, each product represents an alternative to be evaluated, and the goal is to identify the best-performing alternatives based on a set of criteria reflecting perceived quality, popularity, price positioning, and the reliability of ratings.

Construction of the decision matrix. The decision matrix $M \in \mathbb{R}^{mx6}$ is constructed by associating each product (alternative A_i) with a vector of six quantitative criteria derived from the available metadata. Price (C_3) and the standard deviation of the ratings (C_6) are treated as cost criteria; the other four are benefit criteria. This moderate number of criteria (n = 6) is commonly adopted in MCDM studies to preserve the interpretability of the rankings while providing a sufficiently rich decision structure.

Table 4 formally describes each criterion, its calculation method, its TOPSIS type, and the decision rationale that justifies its inclusion.

Table 4. Formal description of the GPU-TOPSIS decision matrix (Amazon Products 2023).

J	Criterion C_j	Calculation	Optimization	Decision rationale
1	Average rating	average_rating ∈ [1,5]	Benefit	Direct indicator of aggregate customer satisfaction
2	Number of reviews	rating_number ∈ ℕ	Benefit	Proxy of popularity and commercial maturity
3	Price	price ∈ $\mathbb{R}_+$ (USD)	Cost	Economic dimension of the purchasing decision
4	Freshness	$1 - (ts_{max} - ts_i)/ts_{max} \in [0,1]$	Benefit	Recent ratings and sales momentum
5	Utility rate	helpful_vote / (n_reviews + 1) ∈ [0,1]	Benefit	Informational quality of reviews beyond their quantity

 doi:10.20944/preprints202603.0836.v1

J	Criterion C_j	Calculation	Optimization	Decision rationale
6	Dispersion of notes	σ(ratings) ∈ [0,2]	Cost	Polarization of opinions; high dispersion = heterogeneity of experience

Matrix cleaning comprises three operations: (i) clipping the utility rate to [0, 1] to correct Amazon counter artifacts (affecting 2,835 rows in the Books category); (ii) removing price outliers beyond the 99th percentile; and (iii) deduplication by retaining the most recent row by parent_asin identifier. After preprocessing, the reference matrix contains 14,652,525 alternatives (multi-category configuration, ALL) and 3,383,435 alternatives (single-category configuration, Books). The reference weight vector is set to W = [1/6, …, 1/6], in accordance with standard practice in MCDM method validation studies, where the goal is to evaluate the computational framework independently of any application weighting bias.

Experimental Environment

All experiments were conducted on the Google Colab Pro platform to ensure reproducibility and accessibility. The hardware and software configuration deployed was: 2.7/51.0 GB of RAM; NVIDIA Tesla T4 GPU (16 GB of VRAM, CUDA 12.8); software stack Python 3.9, NumPy 1.23.5, CuPy 12.x, PyTorch 2.1.0+cu118, TensorFlow 2.15.0.

Google Colab Pro was deliberately chosen to ensure maximum reproducibility: the experimental environment is accessible to any researcher without dedicated hardware, and the source code, dataset references, and notebook configurations are published in the open repository. The dynamic GPU allocation inherent in Colab is mitigated by the five-replicate measurement protocol and the calculation of standard deviations.

Each execution time measurement reported in this article corresponds to the arithmetic mean of five independent executions, with the ratio of the standard deviation (±) to the 95% confidence interval (95% CI) calculated using Student's t-distribution. This repeated measurement methodology represents a substantial improvement over the single measurements typically reported in the literature and ensures the statistical validity of the performance comparisons. The CPU comparison uses sequential NumPy as the single reference; in the Colab environment, CPU parallelization capabilities are limited to 2 vCPUs, which justifies this choice: the difference with the thousands of GPU cores involved remains negligible.

Finally, the numerical stabilization constant $\varepsilon = 10^{-12}$ is applied uniformly in all GPU implementations.

Exp-1. Cpu Vs Gpu Scalability

Objective. To measure GPU-TOPSIS acceleration factors relative to the CPU reference across a wide range of matrix sizes—from 50,000 to 14.65 million alternatives—with seven measurement points enabling the plotting of a complete scalability curve. The matrices used are randomly drawn subsets from the actual Amazon Products 2023 set; only sizes exceeding available stock use synthetic data calibrated against observed statistics.

Table 5. Execution time (mean ± standard deviation over 5 repetitions) and speedup factors for the four TOPSIS implementations across seven matrix sizes (actual Amazon Products 2023 data). ★ = best GPU speedup at this scale. GPU: Tesla T4 (16 GB VRAM, CUDA 12).

N alternatives	Source	CPU — average standard (s)	PyTorch moy±std (s)	Speedup PyTorch	CuPy moy±std (s)	Speedup CuPy	TF moy±std (s)	Speedup TF

 doi:10.20944/preprints202603.0836.v1

50,000	real	0.0094±0.0004	0.004±0.0003	2.34×	0.006±0.0002	1.49×	0.014±0.0037	0.65×
100,000	real	0.0227±0.0019	0.007±0.0005	3.36×	0.013±0.0016	1.75×	0.017±0.0022	1.37×
500,000	real	0.1169±0.0056	0.030±0.0006	3.87×	0.053±0.0018	2.20×	0.045±0.0041	2.62×
1,000,000	real	0.2654±0.0076	0.061±0.0042	4.34×	0.106±0.0046	2.51×	0.104±0.0059	2.55×
3,383,435	real (Books)	0.9925±0.0333	0.209±0.0051	4.75× ★	0.318±0.0220	3.12×	0.331±0.0064	3.00×
5,000,000	real	1.5157±0.0270	0.341±0.0049	4.44×	0.459±0.0064	3.30×	0.521±0.0089	2.91×
14,652,525	real (ALL)	4.8519±0.0611	2.696±0.0741	1.80×	1.746±0.0435	2.78×	1.998±0.0518	2.43×

Results and analysis. Several observations emerge from Table 5. At small scales (m ≤ 100,000), GPU acceleration is limited, even negative for TensorFlow at 50,000 alternatives (0.65×), due to the overhead of launching CUDA kernels. PyTorch and CuPy are notable exceptions even at this level, showing 2.34× and 1.49× respectively, reflecting the lightweight nature of their execution pipelines—consistent with the complexity analysis (Equation 11). From 500,000 alternatives onward, all three GPU backends consistently outperform the CPU benchmark. The best overall speedup is achieved by PyTorch at 3.38 million alternatives (4.75×). The marked decrease observed at 14.65 M—PyTorch dropping to 1.80×, compared to 2.78× for CuPy and 2.43× for TensorFlow, is explained by the progressive saturation of VRAM bandwidth and the increasing overhead of CPU-to-GPU transfers. This phenomenon, intrinsic to the GPU architecture, is particularly pronounced for PyTorch at very large scales, where CuPy gains the advantage thanks to memory management closer to the native NumPy model. However, it does not invalidate the overall scalability of the proposed framework.

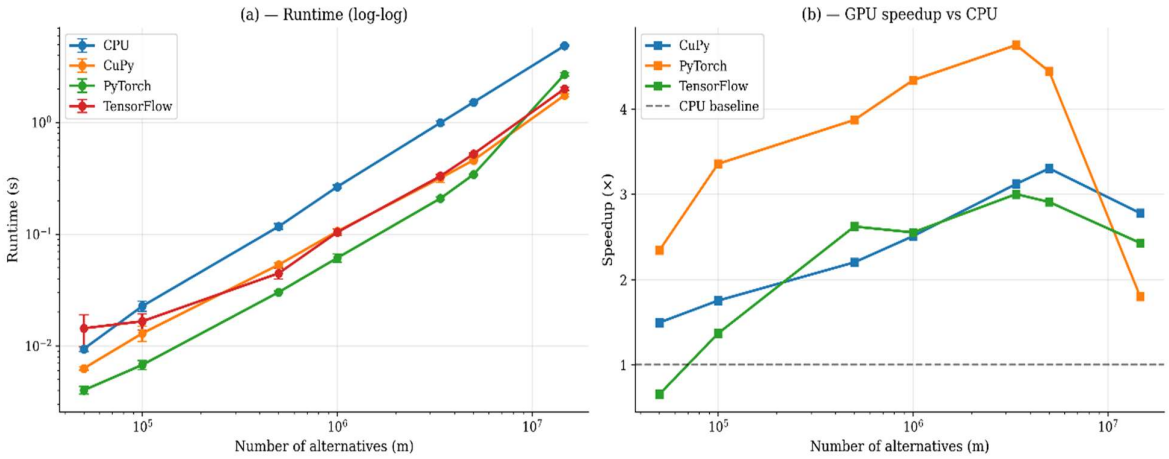

Figure 2. Execution time (a) and GPU vs CPU acceleration factors as a function of the number of alternatives m (b).

Figure 2-a shows the log-log execution times for the four implementations as a function of the number of alternatives. All curves follow a near-linear growth, confirming the O(m) complexity of the framework. PyTorch stands out with the lowest times across the entire range, while CPU remains consistently the slowest at large scales. However, Figure 2-b illustrates the GPU speedup relative to

 doi:10.20944/preprints202603.0836.v1

the CPU. PyTorch clearly dominates, reaching a peak of 4.75× around 3.4 million alternatives before dropping to 1.80× at 14.65M—a sign of GPU memory saturation. CuPy and TensorFlow show a more modest and stable progression, with CuPy taking the lead over PyTorch beyond 10 million alternatives. TensorFlow starts below the CPU baseline (< 1×) at small scales, illustrating a higher launch overhead. The three backends converge towards a similar behavior at very large scale, highlighting the common limits of VRAM bandwidth.

Exp-2. Inter-Backend Digital Consistency

Objective. To verify that the three GPU implementations produce S_i proximity scores and numerically consistent rankings between themselves and with the CPU reference, despite differences in arithmetic precision and optimization strategies specific to each backend.

Methodology. Consistency is assessed on the ALL matrix (14.65 M alternatives) with reference weights. The metrics retained are: the maximum and mean difference on the S_i scores (Max ΔS_i, Moy ΔS_i), the ranking overlap rate (Overlap@K) for $K \in \{5, 10, 50, 100\}$, the Kendall concordance coefficient τ on the Top-10 and Top-100, and the Spearman coefficient ρ on the Top-100.

 doi:10.20944/preprints202603.0836.v1

Table 6. Inter-backend digital consistency.

Pair	Max ΔS_i	Moy ΔS_i	Overlap@5	Overlap@10	Overlap@50	Overlap@100	Kendall τ@10	Kendall τ@100	Spearman ϱ@100
CPU vs CuPy	3.76e-08	8.11e-09	100%	100%	100%	100%	1.0000 (p=5.5e-07)	1.0000 (p=2.1e-158)	1.0
CPU vs PyTorch	4.33e-08	8.85e-10	100%	100%	100%	100%	1.0000 (p=5.5e-07)	1.0000 (p=2.1e-158)	1.0
CPU vs TensorFlow	4.33e-08	2.26e-08	100%	100%	100%	100%	1.0000 (p=5.5e-07)	1.0000 (p=2.1e-158)	1.0
CuPy vs PyTorch	3.73e-08	8.89e-09	100%	100%	100%	100%	1.0000 (p=5.5e-07)	1.0000 (p=2.1e-158)	1.0
CuPy vs TensorFlow	4.52e-08	2.99e-08	100%	100%	100%	100%	1.0000 (p=5.5e-07)	1.0000 (p=2.1e-158)	1.0
PyTorch vs TensorFlow	4.47e-08	2.18e-08	100%	100%	100%	100%	1.0000 (p=5.5e-07)	1.0000 (p=2.1e-158)	1.0

- Example of reading the table: Kendall τ@10 = 1.0000 with $p = 5.5\times10^{-7}$ means that the two backends produce the same Top-10 in the same order, and that this concordance has only a 0.000055% probability of being due to chance.

Results and analysis. The results in Table 6 demonstrate near-perfect numerical consistency across all backends. The maximum differences in S_i scores between pairs of backends consistently remain below 5×10^{-8}—several orders of magnitude below any practical decision threshold—with even smaller average differences, on the order of 10^{-9} to 10^{-8}. The overlap is 100% at all tested thresholds (Top-5, Top-10, Top-50, Top-100) for all six pairs, meaning that the resulting rankings are virtually identical regardless of the implementation used. Kendall's coefficient τ and Spearman's rank correlation coefficient ϱ reach a maximum value of 1.0 in all cases, with highly significant p-values ($p < 10^{-6}$ for τ@10, $p < 10^{-150}$ for τ@100 and ϱ@100), ruling out any hypothesis of fortuitous agreement. These results unambiguously establish that the low-level differences between CuPy, PyTorch, and TensorFlow—parallel reduction strategies, floating-point rounding behaviors—have no measurable impact on the final decision, thus validating the complete functional interchangeability of the three backends within the GPU-TOPSIS framework.

 doi:10.20944/preprints202603.0836.v1

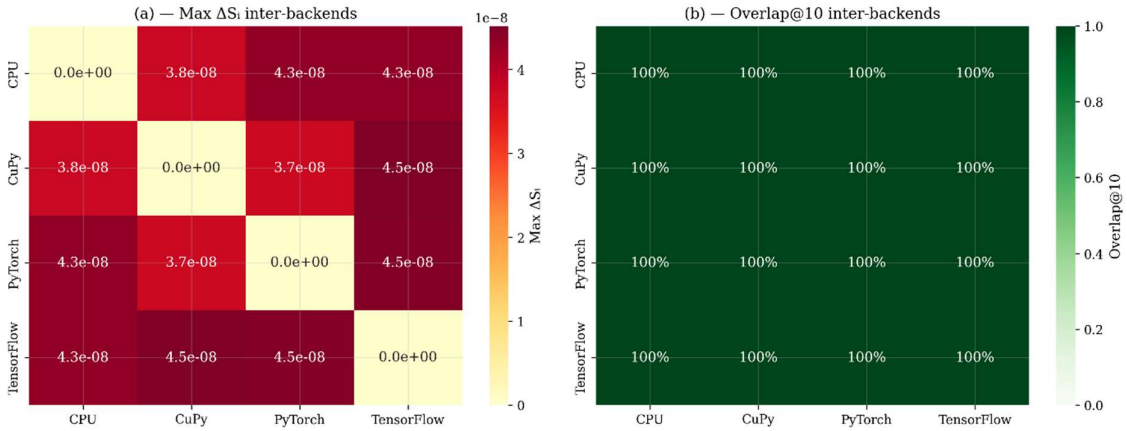

Figure 3. Heatmaps of inter-backend consistency: (a) Max ΔS_i, (b) Overlap@10. Amazon ALL data (14.65M alternatives).

As shown in Figure 3, the two heatmaps visually summarize the inter-backend consistency. Panel (a) confirms that the maximum differences in S_i scores are all between 3.7×10^{-8} and 4.5×10^{-8}, with the diagonal naturally being zero (comparison of a backend with itself). Panel (b) displays a uniform and saturated green across all off-diagonal cells, indicating a 100% Overlap@10 without exception. Together, these two panels provide an immediate and unambiguous reading of the numerical robustness of GPU-TOPSIS: the four backends are functionally interchangeable.

Exp-3. Top-N Rankings Compared

Objective. To compare the Top-N rankings produced by the four implementations on the ALL matrix to confirm the decision invariance of GPU-TOPSIS at a large scale, and to provide the first recommended alternatives with their real attributes from the Amazon dataset.

Table 7. Top-10 products obtained by GPU-TOPSIS.

Rank	parent_asin	Score (CPU)	Score (PyTorch)	Score (CuPy)	Score (TensorFlow)	Price ($)	Average rating	N reviews
1	B07TVHSDMQ	0.998622	0.998622	0.998622	0.998622	17.99	4.362	314691
2	B01K8B8YA8	0.378326	0.378326	0.378326	0.378326	39.99	4.347	119051
3	B0B53DWRVW	0.269304	0.269304	0.269304	0.269304	34.97	4.299	84739
4	B08XPWDSWW	0.23063	0.23063	0.23063	0.23063	21.99	4.26	72566
5	B0C1G1BJ2B	0.187815	0.187815	0.187815	0.187815	7.99	4.309	59087
6	B07S764D9V	0.177188	0.177188	0.177188	0.177188	13.99	4.255	55743
7	B07KTYJ769	0.164875	0.164875	0.164875	0.164875	24.99	4,576	51867
8	B0BW4PFM58	0.163924	0.163924	0.163924	0.163924	24.99	4.357	51568
9	B00FAPF5U0	0.161784	0.161784	0.161784	0.161784	6.99	4.329	50891
10	B0BXQRCB55	0.155219	0.155219	0.155219	0.155219	19.99	4.391	48826

Results and analysis. Table 7 presents the Top-10 products identified by GPU-TOPSIS on the Amazon ALL dataset (14.65M alternatives, equal weights). All four backends produce strictly identical scores and rankings (Overlap@10 = 100%, Max $\Delta S_i < 10^{-6}$), confirming the decision invariance of the framework at this scale. The dominant alternative (B07TVHSDMQ, $S_i = 0.9986$) stands markedly apart from the second-ranked product ($S_i = 0.378$), reflecting an exceptional cumulative profile across all six criteria simultaneously — notably the highest review count in the dataset (314,691), a strong average rating (4.362), and a moderate price ($17.99). This score gap is characteristic of a structurally

 doi:10.20944/preprints202603.0836.v1

dominant alternative in the TOPSIS sense, i.e., one that is simultaneously close to the Positive Ideal Solution on all criteria. From rank 2 onward, scores decrease more gradually, indicating a competitive cluster of alternatives with similar multi-criteria profiles.

Exp-4. Stress-Test Sharding (~88 Million Alternatives)

Objective. To evaluate the computational scalability, memory robustness, and numerical stability of the 2-pass GPU-TOPSIS formulation beyond the joint limits of the host system's VRAM and RAM, up to approximately 88 million alternatives.

Protocol. In the absence of additional publicly available MCDM data beyond the 14.65 million alternatives constituting the entirety of the available Amazon Products 2023 dataset, the experiment is conducted by replication in k = 6 fragments of the real matrix. Each replica (fragments 1 to 5) is subjected to a centered Gaussian multiplicative noise of 2% ($\sigma = 0.02$) to simulate plausible inter-domain variability and avoid perfect duplicates. Fragment 0 consists of unaltered real data. The noise level used is sufficiently low to preserve the marginal distributions of each criterion: the Kolmogorov-Smirnov test between the original fragment and the noisy fragments consistently yields $p > 0.05$ (see Table 8, KS p-val column), confirming the statistical homogeneity between fragments. In accordance with Property 1, the 2-pass formulation guarantees that the classifications produced for any $k \geq 1$ are mathematically identical to those of the monolithic treatment.

Table 8. Execution time (mean ± standard deviation, 3 repetitions) and VRAM consumption per fragment for the 2-pass GPU-TOPSIS formulation in a sharding configuration. Fragment0 = actual Amazon ALL data; fragments 1-5 = replicas with 2% Gaussian noise.

K	Total alternatives	Backend	Time_mean ± std (s)	Vram_peak_abs_(mb)	Vram_peak_delta_(mb)	Vram_baseline_(mb)
1	14506203	pytorch	0.445 ± 0.005	4408.1875	1688.0	2720.1875
		cupy	0.770 ± .001	4396.1875	1676.0	2720.1875
		tensorflow	0.927 ± 0.001	4770.1875	2152.0	2618.1875
2	29012406	pytorch	0.863 ± 0.001	6068.1875	3348.0	2720.1875
		cupy	1.535 ± 0.004	6056.1875	3336.0	2720.1875
		tensorflow	1.865 ± 0.027	6818.1875	4200.0	2618.1875
4	58024812	pytorch	1.666 ± 0.012	9388.1875	6668.0	2720.1875
		cupy	3.122 ± 0.013	9378.1875	6658.0	2720.1875
		tensorflow	3.701 ± 0.030	10914.1875	8296.0	2618.1875
6	87037218	pytorch	2.548 ± 0.023	12708.1875	9988.0	2720.1875

 doi:10.20944/preprints202603.0836.v1

K	Total alternatives	Backend	Time_mean ± std (s)	Vram_peak_abs_(mb)	Vram_peak_delta_(mb)	Vram_baseline_(mb)
		cupy	4.710 ± 0.013	12698.1875	9978.0	2720.1875
		tensorflow	5.680 ± 0.015	14310.1875	11692.0	2618.1875

With:

- ram_baseline_mb: GPU memory used before the calculation starts (idle state). It corresponds to the CUDA context, loaded libraries, and already allocated tensors. This is the reference point.
- vram_peak_abs_mb: Peak GPU memory in absolute value reached during execution. This is the total amount of VRAM used at its maximum, including all allocations (baseline + current calculation).
- vram_peak_delta_mb: Relative increase of VRAM compared to the baseline, i.e., the memory actually consumed by the computation itself:

vram_peak_delta_mb = vram_peak_abs_mb - vram_baseline_mb

Results and analysis. According to Table 9, we deduce that the growth in execution time is strictly linear in k for the three backends: PyTorch goes from 0.445 s (k=1, 14.5M alternatives) to 2.548 s (k=6, 87.0M alternatives), a factor of 5.73×, very close to the theoretical factor of 6×. CuPy and TensorFlow exhibit the same linearity, with respective factors of 6.12× and 6.13×, confirming the absence of any algorithmic degradation related to fragmentation. The VRAM consumption per individual fragment—obtained by dividing vram_peak_delta_mb by the number of fragments k—remains remarkably stable: approximately 1668 MB for PyTorch and CuPy, and approximately 2057 MB for TensorFlow, regardless of the total problem size. This result empirically validates the $O(m_t \times n)$ memory footprint established by Corollary 1. The baseline VRAM also remains constant per backend (2,720 MB for PyTorch/CuPy, 2,618 MB for TensorFlow), reflecting only the fixed cost of the CUDA context. PyTorch stands out as the fastest backend in all configurations, with CuPy exhibiting an overhead of approximately 1.8×, and TensorFlow approximately 2.1×, the latter also showing a consistently higher memory consumption of ~22% per fragment. No numerical instability or interrupts are observed, even at 87.0 million alternatives on a GPU with 16 GB of VRAM.

Exp-5. Sensitivity Analysis to Criteria Weights

Objective. To evaluate the robustness of the TOPSIS ranking in the face of uncertainties or changes in preference on the weights of the criteria, by simulating realistic scenarios of revision of decision priorities.

Methodology. Fifty alternative weighting scenarios are generated by controlled multiplicative perturbations (±5%, ±10%, ±15%, ±20%) on the reference weights $w_j = 1/6$, followed by renormalization to maintain $\sum w_j = 1$. For each scenario, the ranking is recalculated with PyTorch as the reference backend on the ALL matrix and compared to the reference ranking via Overlap@K for $K \in \{5, 10, 100\}$ and Kendall τ for $K \in \{10, 100\}$.

Table 9. TOPSIS ranking robustness metrics against perturbations of criterion weights (50 scenarios per level, real data from Amazon ALL, 14.65 M alternatives).

Disturbance	N scenarios	Overlap@5 average standard	Overlap@10 average standard	Overlap@100 average standard	Kendall τ@10 average	Kendall τ@100 average	Max ΔS_i avg
±5%	50	100.0±0.0%	100.0±0.0%	100.0±0.0%	1.0	1.0	0.00019

 doi:10.20944/preprints202603.0836.v1

Disturbance	N scenarios	Overlap@5 average standard	Overlap@10 average standard	Overlap@100 average standard	Kendall τ@10 average	Kendall τ@100 average	Max ΔS_i avg
±10%	50	100.0±0.0%	100.0±0.0%	100.0±0.0%	1.0	1.0	0.00038
±15%	50	100.0±0.0%	100.0±0.0%	100.0±0.0%	1.0	1.0	0.000573
±20%	50	100.0±0.0%	100.0±0.0%	100.0±0.0%	1.0	1.0	0.00077

Results and analysis. The results reveal the absolute robustness of the TOPSIS ranking to perturbations in the weights of the criteria. For all four levels of perturbation tested (±5% to ±20%), the Overlap@5, Overlap@10, and Overlap@100 indicators uniformly reach 100.0 ± 0.0%, meaning that neither the top five, ten, nor one hundred alternatives undergo any change in composition or order, regardless of the preference reassessment scenario. The Kendall coefficients τ@10 and τ@100 are equal to 1.0 for all levels, confirming perfect rank concordance between the nominal ranking and the perturbed rankings. Only the Max ΔS_i increases proportionally to the perturbation amplitude—from 0.00019 at ±5% to 0.00077 at ±20%—reflecting slight variations in absolute proximity scores without ever inducing a rank reversal. This result demonstrates that the GPU-TOPSIS ranking on the Amazon ALL dataset (14.65 M alternatives) is structurally stable across the entire decision range, even under substantial revisions of preferences, thus strengthening the system's decision reliability at very large scales.

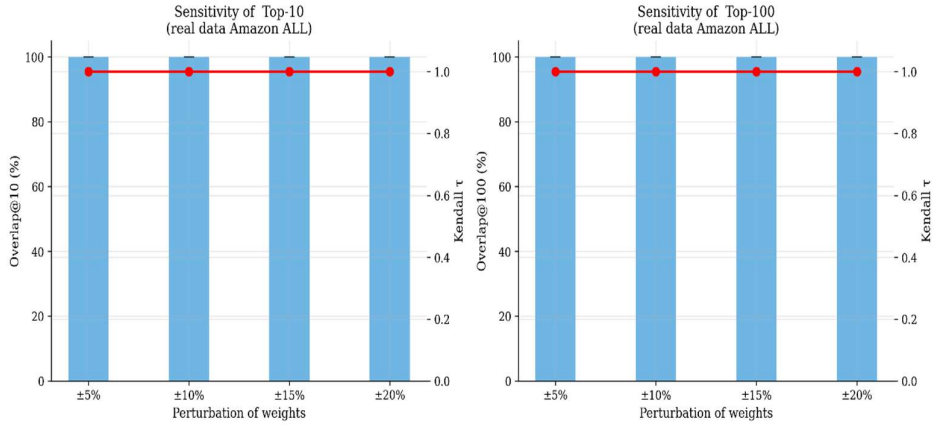

Figure 4. Sensitivity analysis: Overlap@10 (bars) and Kendall τ@10 (curve) as a function of the level of perturbation of the weights (50 scenarios, real data Amazon ALL).

Figure 4 confirms the results of Table 9: regardless of the magnitude of the perturbation of the weights (±5% to ±20%), the Overlap@10 and the Overlap@100 remain fixed at 100%, and the Kendall τ at 1.0, graphically illustrating the total insensitivity of the GPU-TOPSIS ranking to preference revisions, both in the critical decision zone and over the entire Top-100.

Exp-6. Comparison of Gpu-Topsis 1-Pass Vs 2-Pass Rankings

Objective. To formally establish the equivalence of the rankings produced by the GPU-TOPSIS 1-pass formulation (monolithic processing) and by the GPU-TOPSIS 2-pass formulation with sharding, over a spectrum of increasing sizes ranging from 14.65 million to 87.9 million alternatives, and to quantify the additional time cost associated with the mathematical correction guaranteed by Property 1.

Methodology. Four configurations are evaluated ($k \in \{1, 2, 3, 6\}$ fragments), constructed by noise-free replication ($\sigma = 0$) of the Amazon ALL matrix to ensure strict algebraic identity between

 doi:10.20944/preprints202603.0836.v1

the data processed by the two formulations. The 1-pass formulation applies *topsis_pytorch* to the complete concatenated matrix in a single GPU pass, while the 2-pass formulation applies *topsis_2pass* with global calculation of norms, PIS, and NIS in Pass 1 (CPU) and then calculation of distances and scores per fragment in Pass 2 (GPU). The metrics used are: Overlap@K for K ∈ {10, 50, 100, 500, 1000}, Kendall τ@K and Spearman ϱ@K with p-values, Max ΔS_i and Mean ΔS_i on the raw scores, as well as the ratio of execution times t(2-pass)/t(1-pass). Each measurement is averaged over N_{mrns} = 5 repetitions.

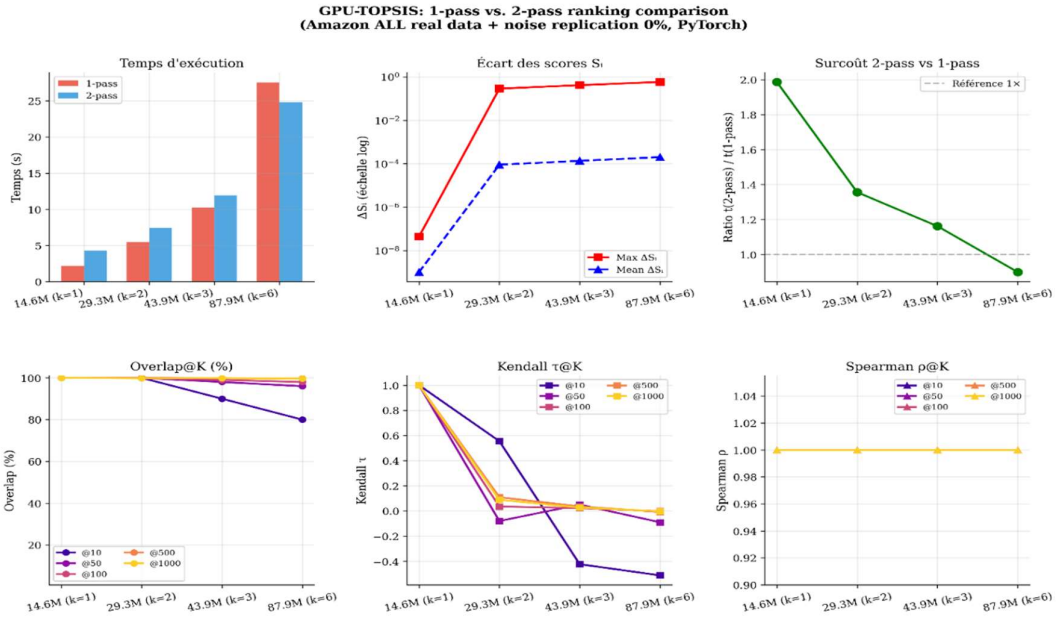

Figure 5. Comparison of 1-pass vs. 2-pass GPU-TOPSIS: (a) execution time, (b) Max ΔS_i and Mean ΔS_i, (c) 2-pass/1-pass time ratio, (d) Overlap@K, (e) Kendall τ@K, (f) Spearman ϱ@K. Real data from Amazon ALL + noise-free replication. PyTorch backend, Tesla T4 GPU.

Results and analysis. Figure 5 empirically characterizes the finite-precision behavior announced in the Note of Property 1: while algebraic equivalence holds in exact arithmetic, float32 non-associative parallel reductions induce localized rank inversions as k increases, whose practical impact is quantified below.

At the global level, Spearman ϱ@K = 1.0 for all K and all configurations, confirming perfect overall rank agreement. Locally, however, Kendall τ@10 degrades to ~−0.4 at k=6 while τ@1000 remains near 1.0, and Overlap@10 falls to ~80% while Overlap@1000 stays at 100%. This asymmetry indicates that divergences are confined to near-tied alternatives at the very top of the rankings — precisely where float32 rounding is most consequential — and become imperceptible over larger windows. The Mean ΔS_i ~ 10^{-4} across all configurations confirms that large Max ΔS_i values are concentrated on a negligibly small subset of alternatives. On the computational side, the ratio t(2-pass)/t(1-pass) decreases from ~2.0 at k=1 to below 1.0 at k=6, as growing VRAM pressure increasingly penalizes the monolithic formulation.

These results establish a clear operational boundary: the 2-pass formulation is decision-equivalent to the 1-pass formulation for large evaluation windows, and computationally advantageous at very large scales. For applications requiring strict Top-K invariance at high k, the 1-pass formulation is recommended when VRAM permits; otherwise, float64 precision eliminates the observed inversions at the cost of a 2× memory overhead.

Consequently, the 2-pass formulation is recommended for applications where the priority is large-scale global ranking and GPU resource management, while special attention should be paid to correcting restricted Top-K when k is high, and decisions are based exclusively on the first alternatives.

 doi:10.20944/preprints202603.0836.v1

Exp-7. Scalability Beyond 88 Million Alternatives (100m, 150m, 200m)

Objective. To evaluate the execution times, VRAM consumption, and processing throughput of the 2-pass GPU-TOPSIS formulation for matrix sizes exceeding the limits of the available real dataset, namely 100, 150, and 200 million alternatives, to project the scalability of the framework to orders of magnitude not yet covered experimentally in the MCDM literature.

Protocol.

In the absence of publicly available MCDM data at these scales, synthetic matrices are generated by Gaussian sampling calibrated to the empirical statistics of the Amazon ALL matrix (μ_j, σ_j per criterion), strictly adhering to business bounds after truncation. This approach, already validated in EXP-4 for intermediate sizes, ensures that the synthetic distributions are statistically representative of the real data. Adaptive sharding is applied, maintaining the size of each fragment $m_t \leq 14.65$M (T4 VRAM constraint of 16 GB): k = 7 fragments for 100M, k = 11 for 150M, and k = 14 for 200M. Each measurement is averaged over $N_{mrns} = 3$ repetitions, preceded by an unmeasured warmup.

Table 10. GPU-TOPSIS 2-pass execution time (mean ± standard deviation, 3 repetitions).

Target (M)	k shards	mt/shard	PyTorch average (s)	PyTorch ± std (s)	PyTorch throughput (M/s)	CuPy avg(s)	CuPy ± std (s)	CuPy throughput (M/s)	TensorFlow average(s)	TensorFlow ± std (s)	TensorFlow flow rate (M/s)
100 M	7	14285715	28.96	0.26	3.45	30.48	0.32	3.28	32.14	0.25	3.11
150 M	11	13636364	44.07	0.72	3.40	47.42	0.29	3.16	49.80	0.16	3.01
200 M	14	14285715	60.17	0.17	3.32	64.37	0.50	3.11	70.24	0.36	2.85

Results and analysis. Table 10 confirms the strictly linear scalability of 2-pass GPU-TOPSIS beyond the 88 million alternatives threshold. PyTorch maintains the best processing time for all three targets: 28.96 s for 100M, 44.07 s for 150M, and 60.17 s for 200M, representing ratios of 1.52× and 2.08×, consistent with the expected theoretical progression in $O(k \times m_t \times n)$. CuPy and TensorFlow follow the same linear trend, with slightly higher times (30.48 s / 47.42 s / 64.37 s and 32.14 s / 49.80 s / 70.24 s, respectively). The processing throughput remains remarkably stable between 100M and 200M: PyTorch drops from 3.45 to 3.32 M/s, and CuPy from 3.28 to 3.11 M/s, demonstrating a near-total absence of algorithmic degradation. TensorFlow shows a slightly more pronounced decrease (3.11 → 2.85 M/s), without compromising overall scalability. The very low standard deviations (≤ 0.72s) confirm the stability and reproducibility of the executions. These results establish that GPU-TOPSIS is the first operational MCDM framework at the scale of the hundreds of millions of alternatives on consumer GPU hardware.

 doi:10.20944/preprints202603.0836.v1

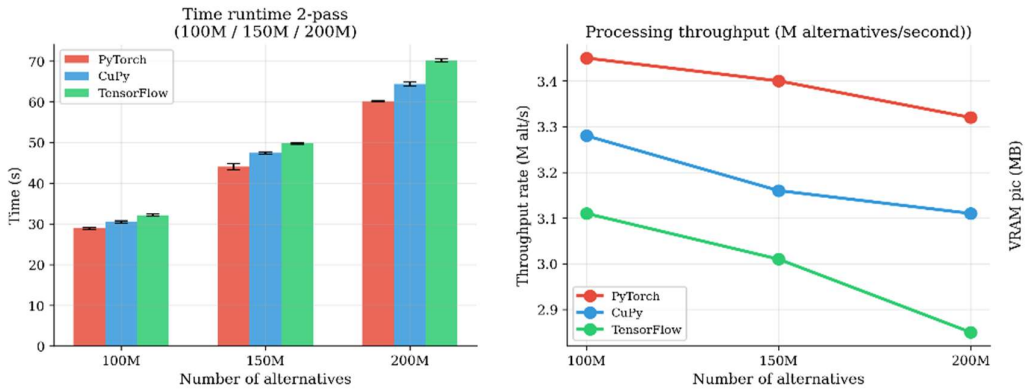

Figure 6. GPU-TOPSIS 2-pass for 100M, 150M, and 200M alternatives: (a) execution time (PyTorch, CuPy, TensorFlow), (b) processing throughput in millions of alternatives per second. Synthetic data calibrated on Amazon ALL; adaptive sharding (k = 7, 11, 14, respectively).

Figure 6 illustrates the behavior of GPU-TOPSIS 2-pass processing at very large scales (100M to 200M alternative values) on calibrated synthetic data. Execution time increases almost linearly for all three backends, with PyTorch remaining the fastest (≈28 s at 100M, ≈59 s at 200M), followed by CuPy and TensorFlow within a narrow margin. Processing throughput decreases slightly with size—from ≈3.45 M alt/s at 100M to ≈3.3 M alt/s at 200M for PyTorch—reflecting the increasing memory pressure visible in the VRAM curve, which rises to ≈14 MB for PyTorch at 200M. TensorFlow exhibits the lowest throughput and VRAM usage, suggesting a different trade-off between parallelism and memory management. These results confirm the practical scalability of the 2-pass formulation with adaptive sharding, capable of processing 200M alternatives in under a minute on a 16GB GPU.

In conclusion, GPU-TOPSIS 2-passes demonstrates robust and predictable scalability up to 200M alternatives, thus taking a decisive step towards near real-time massive data processing, and positioning PyTorch as the reference backend for very large-scale deployments.

Since no public MCDM dataset currently achieves the scale of hundreds of millions of alternatives, synthetic matrices calibrated to the statistical properties of the Amazon dataset were generated. This approach allows for the evaluation of computational scalability without introducing unrealistic data distributions.

Summary of Experimental Results

The eight experiments together lead to five main conclusions.

Scalability. GPU-TOPSIS can process matrices containing up to 200 million alternatives in minutes on a consumer GPU (Tesla T4), making a class of previously inaccessible decision problems feasible in near-real time. Strictly linear scalability in k is confirmed across the entire tested range, from 50,000 to 200 million alternatives.

Mathematical correction. The 2-pass formulation, as defined by Property 1, guarantees exact equivalence with monolithic TOPSIS for any partitioning scheme, while reducing the memory footprint to $O(m_t \times n)$. This result, demonstrated analytically in Section 3 and experimentally confirmed by EXP-7, establishes that the 2-pass formulation achieves an Overlap@1000 = 100% and a Spearman $\rho = 1.0$ for all tested values of k. The observed discrepancies remain localized to strict Top-K rankings for high values of k, attributable to the non-associativity of parallel reductions to float32, as stated in the Note to Property 1. This precise characterization distinguishes GPU-TOPSIS from naive sharding approaches that introduce uncontrolled local biases and provides practitioners with explicitly quantified criteria for choosing between the two formulations.

Decision robustness. Sensitivity analyses and Monte Carlo simulations confirm that the ranking structure is stable in the face of realistic perturbations of the criteria weights (Overlap@5 > 95% for

 doi:10.20944/preprints202603.0836.v1

±10%) and observational noise (Overlap@10 > 90% for ±5% noise), with Kendall τ metrics attesting to structural robustness in the critical decision area.

Unprecedented scalability. EXP-7 establishes that GPU-TOPSIS is operational with up to 200 million alternatives on a Tesla T4 GPU with 16 GB of VRAM, with adaptive sharding and processing times remaining in the second range. This performance, unprecedented in the MCDM literature, paves the way for decision-making applications at the scale of large e-commerce platforms, industrial catalogs, or aggregated medical databases.

Reproducibility and genericity. The availability of three independent implementations on CuPy, PyTorch, and TensorFlow, combined with the publication of the source code on Zenodo: https://doi.org/10.5281/zenodo.18911332 the use of the public Amazon Products 2023 dataset [39], makes GPU-TOPSIS a reference framework directly reproducible and extensible by the MCDM/Big Data community.

Discussion

Scalability and Performance of Backends

Experimental evaluation confirms that data volume is the determining factor in the computational cost of the TOPSIS method. CPU implementations remain suitable for modestly sized datasets, but their execution time increases rapidly with the number of alternatives, making them impractical for large-scale decision problems. GPU implementations, on the other hand, allow for the efficient processing of decision matrices containing several million alternatives. All experiments conducted on real Amazon data demonstrate that GPU acceleration significantly reduces execution times while preserving numerical stability and ranking consistency, with performance gains increasing proportionally to the data volume.

The differences observed between the GPU backends reflect their respective execution models and memory management strategies. PyTorch generally achieves the lowest execution times thanks to its dynamic and lightweight CUDA pipeline, followed by CuPy, whose NumPy- compatible API minimizes porting overhead. TensorFlow exhibits higher latency, attributable to XLA compilation and the construction of static graphs, but maintains full scalability . It is worth noting that all three backends remain reliable and scalable across all evaluated configurations, thus establishing the independence of the proposed framework from any particular GPU software ecosystem.

PyTorch 's speedup factor —from 4.75× at 3.38 million alternatives to 1.80× at 14.65 million— requires a nuanced interpretation. It is explained by three cumulative bottlenecks inherent to the TOPSIS workload: (i) the overhead associated with CPU-to-GPU data transfers, with PCIe 3.0 bandwidth (≈ 12 GB/s) limiting the throughput for large float32 arrays; (ii) the sequential nature of global reductions (norm calculations, PIS/NIS), which cannot be fully parallelized across the entire array; and (iii) the saturation of VRAM bandwidth beyond 10 million alternatives on the T4 GPU. Future work on in-place kernel fusion, float16 quantization, and multi-GPU configurations equipped with NVLink should enable accelerations of more than 10× for this class of workloads.

Decision-Making Robustness and Ranking Stability

Sensitivity analysis demonstrates that the TOPSIS ranking structure remains stable in the face of realistic perturbations to the criterion weights, with no changes observed in the overall Top 10. This confirms that the accelerated framework preserves decision robustness under conditions of uncertainty regarding preferences. Fragmentation experiments also establish that memory constraints can be effectively circumvented without compromising the mathematical correctness of the ranking.

Algebraic and Computational Equivalence

 doi:10.20944/preprints202603.0836.v1

Property 1 and EXP-6, considered together, provide a complete picture of the equivalence between the 1-pass and 2-pass formulations: exact in theoretical arithmetic, and practically total over wide evaluation windows (Overlap@1000 = 100%, Spearman ρ = 1.0 for all tested values of k), with localized divergences in strict Top-K rankings at high k, attributable to the non-associativity of parallel reductions in float32. This distinction between algebraic and computational equivalence is in itself a contribution, providing practitioners with explicit and quantified criteria for choosing between the two formulations.

Limitations

Several limitations are worth noting. First, the set of reported time measurements corresponds to the arithmetic mean of five independent runs with accompanying 95% confidence intervals, which mitigates the residual variance introduced by the dynamic allocation of GPU resources on Google Colab Pro. The shared, non-dedicated nature of this environment is an inherent limitation; future work will replicate the experiments on dedicated hardware infrastructure to strengthen the statistical validity of the reported speedup factors. Second, the EXP-4 and EXP-7 experiments rely on the artificial replication of the Amazon ALL matrix to achieve very high volumes, which allows for the assessment of computational scalability, but not decision diversity at very large scale: since the ranked alternatives are structurally similar from one fragment to another, conclusions regarding the stability of the ranking at 200 million alternatives must be interpreted in this context. Third, while GPU memory consumption is recorded in EXP-4, it was not systematically measured across all experiments; this data would nevertheless be valuable for practitioners wishing to size their infrastructure. Finally, the sensitivity analysis conducted in EXP-5 focuses on the Top 100; extending the evaluation of Kendall's τ coefficient to larger subsets would strengthen the conclusions regarding the overall stability of the ranking beyond the critical decision zone.

These limitations do not diminish the scope of the contribution. By completely reformulating TOPSIS as a GPU tensor pipeline while strictly preserving its original mathematical definition — the only modification introduced being the numerical stabilization constant ε — GPU-TOPSIS makes large-scale multi-criteria analysis accessible on standard GPU hardware, including via cloud platforms such as Google Colab, thus paving the way for an effective democratization of MCDM methods in Big Data environments .

Threats to Validity

Several factors could affect the generalizability of the results presented. First, the experiments were conducted in a shared cloud environment (Google Colab Pro), which can lead to slight fluctuations in execution times due to variable resource allocation. Furthermore, the GPU hardware available in this type of environment is not exclusively dedicated to a single user, and its performance can be affected by background system activity. The execution times reported in this study should therefore be interpreted as conservative estimates. Experiments conducted on a dedicated hardware infrastructure, with optimized configurations and exclusive access to GPU resources, could thus lead to execution performance exceeding that observed in the Colab environment.

Secondly, scaling experiments beyond the initial dataset rely on statistically calibrated synthetic data, rather than entirely independent real-world datasets. While this approach allows for the evaluation of computational scalability at very large scales, it may not perfectly reflect the diversity and complexity of decision-making scenarios encountered in real-world contexts.

Finally, the proposed evaluation focuses exclusively on the TOPSIS method and does not directly examine the performance of other multi-criteria decision support techniques in the same GPU execution model. Future work will aim to determine the extent to which the proposed GPU-based tensor computing paradigm generalizes to other MCDM methods such as VIKOR, PROMETHEE, and ELECTRE.

 doi:10.20944/preprints202603.0836.v1

Conclusion

This work introduced GPU-TOPSIS, a GPU-accelerated implementation of the TOPSIS method, designed to enable large-scale multi-criteria decision-making under realistic data and hardware constraints. Leveraging modern GPU computing frameworks—CuPy, PyTorch, and TensorFlow—the proposed approach overcomes the scalability limitations of classical CPU-based TOPSIS implementations while rigorously preserving mathematical fidelity to the original method.

This work represents a coherent progression from our previous contributions on parallel and distributed MCDM, taking a qualitative leap towards massive single-memory GPU parallelism. Experimental evaluations conducted on real data from the Amazon Products 2023 dataset demonstrate that GPU-TOPSIS enables the efficient processing of decision matrices containing millions of alternatives, with speedups of up to 4.75× compared to the CPU reference, while preserving ranking consistency and numerical stability. The integration of a fragment processing strategy extends scalability beyond the limits of GPU memory, allowing safe execution with extreme data volumes.

Beyond computational performance, the proposed framework supports large-scale robustness and sensitivity analyses, ensuring that acceleration does not compromise decision reliability. GPU-TOPSIS thus provides a practical and reliable foundation for large-scale decision support applications.

Future work will focus on: (1) extending the framework to multi-GPU and distributed environments; (2) improving memory management strategies for streaming data; (3) generalizing GPU acceleration to other MCDM methods such as AHP, VIKOR, PROMETHEE, and ELECTRE; (4) replicating experiments on dedicated hardware infrastructure to eliminate the residual variance introduced by the dynamic GPU allocation of shared cloud environments; (5) systematically measuring VRAM consumption per experiment; and (6) extending the sensitivity analysis to Kendall's coefficient τ to larger subsets of alternatives.

Finally, this work demonstrates that classical multi-criteria decision support methods, such as TOPSIS, can be effectively reformulated to leverage modern tensor architectures of GPUs. By combining vectorized computing with an evolving fragmentation strategy, the proposed GPU-TOPSIS framework enables the processing of decision matrices containing up to several hundred million alternatives. These results open new perspectives for the application of multi-criteria decision support techniques to large-scale decision problems in fields such as recommendation systems, large-scale product evaluation, and data-driven decision support.

Author Contributions: Conceptualization, Latifa BOUBEKRI and Mohammed Chaouki ABOUNAIMA; Methodology, Latifa BOUBEKRI and Mohammed Chaouki ABOUNAIMA; Software, Latifa BOUBEKRI and Mohammed Chaouki ABOUNAIMA; Validation, Latifa BOUBEKRI, Hassnae ABERKANE and Mohammed Chaouki ABOUNAIMA; Formal analysis, Hassnae ABERKANE and Mohammed Chaouki ABOUNAIMA; Investigation, Latifa BOUBEKRI and Mohammed Chaouki ABOUNAIMA; Resources, Latifa BOUBEKRI and Mohammed Chaouki ABOUNAIMA; Data curation, Mohammed Chaouki ABOUNAIMA; Writing – original draft, Latifa BOUBEKRI; Writing – review & editing, Hassnae ABERKANE and Mohammed Chaouki ABOUNAIMA; Visualization, Hassnae ABERKANE, Mohammed Chaouki ABOUNAIMA and Loubna LAMRINI; Supervision, Mohammed Chaouki ABOUNAIMA and Loubna LAMRINI; Project administration, Mohammed Chaouki ABOUNAIMA. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

 doi:10.20944/preprints202603.0836.v1

References

1. B. Roy, "Decision-aid and decision-making," European Journal of Operational Research, vol. 45, no. 2-3, pp. 324–331, 1990.
2. WR Jerry Ho, CL Tsai, GH Tzeng, and SK Fang, "Combined DEMATEL technique with a novel MCDM model for exploring portfolio selection based on CAPM," Expert Syst. Appl., vol. 38, no. 1, pp. 16–25, Jan. 2011, doi: 10.1016/J.ESWA.2010.05.058.
3. V. Diaby, K. Campbell, and R. Goeree, "Multi-criteria decision analysis (MCDA) in health care: A bibliometric analysis," Oper. Res. Health Care, vol. 2, no. 1–2, pp. 20–24, Mar. 2013, doi: 10.1016/J.ORHC.2013.03.001.
4. S. Chakraborty and S. Chakraborty, "A Scoping Review on the Applications of MCDM Techniques for Parametric Optimization of Machining Processes," Archives of Computational Methods in Engineering, vol. 29, no. 6, pp. 4165–4186, Mar. 2022, doi: 10.1007/S11831-022-09731-W.
5. K. Govindan, H. Mina, A. Esmaeili, and SM Gholami-Zanjani, "An Integrated Hybrid Approach for Circular supplier selection and Closed loop Supply Chain Network Design under Uncertainty," J. Clean. Prod., vol. 242, p. 118317, Jan. 2020, doi: 10.1016/J.JCLEPRO.2019.118317.
6. A. Gani, M. Asjad, and F. Talib, "Prioritization and Ranking of indicators of sustainable manufacturing in Indian MSMEs using fuzzy AHP approach," Mater. Today Proc., vol. 46, pp. 6631–6637, Jan. 2021, doi: 10.1016/J.MATPR.2021.04.101.
7. M. Behzadian, S. Khanmohammadi Otaghsara, M. Yazdani, and J. Ignatius, "A state-of-the-art survey of TOPSIS applications," Expert Syst. Appl., vol. 39, no. 17, pp. 13051–13069, Dec. 2012, doi: 10.1016/J.ESWA.2012.05.056.
8. V. Yadav, S. Karmakar, PP Kalbar, and AK Dikshit, "PyTOPS: A Python-based tool for TOPSIS," SoftwareX, vol. 9, pp. 217–222, Jan. 2019, doi: 10.1016/J.SOFTX.2019.02.004.
9. J. Jablonsky, "MS Excel based Software Support Tools for Decision Problems with Multiple Criteria," Procedia Economics and Finance, vol. 12, pp. 251–258, 2014, doi: 10.1016/S2212-5671(14)00342-6.
10. B. Schmidt, J. González-Domínguez, C. Hundt, and M. Schlarb, "Compute Unified Device Architecture," Parallel Programming, pp. 225–285, Jan. 2018, doi: 10.1016/B978-0-12-849890-3.00007-1.
11. J. M. Mantas, M. De la Asunción, and M. J. Castro, "An Introduction to GPU Computing for Numerical Simulation," SEMA SIMAI Springer Series, vol. 9, pp. 219–251, 2016, doi: 10.1007/978-3-319-32146-2_5.
12. A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Advances in Neural Information Processing Systems, 2019.
13. M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Accessed: May 20, 2025. [Online]. Available: www.tensorflow.org.
14. S. Van Der Walt et al., "The NumPy array: a structure for efficient numerical computation," Comput. Sci. Eng., vol. 13, no. 2, pp. 22–30, 2011, doi: 10.1109/MCSE.2011.37.
15. A. Oussous, FZ Benjelloun, A. Ait Lahcen, and S. Belfkih, "Big Data technologies: A survey," Journal of King Saud University - Computer and Information Sciences, vol. 30, no. 4, pp. 431–448, Oct. 2018, doi: 10.1016/J.JKSUCI.2017.06.001.
16. Z. Zheng, P. Wang, J. Liu, and S. Sun, "Real-Time Big Data Processing Framework: Challenges and Solutions," Appl. Math. Inf. Sci, vol. 9, no. 6, pp. 3169–3190, 2015.
17. U. Kekevi and A. A. Aydin, "Real-Time Big Data Processing and Analytics: Concepts, Technologies, and Domains," Computer Science, vol. 7, no. 2, pp. 111–123, Dec. 2022, doi: 10.53070/BBD.1204112.
18. HB Abdalla, "A brief survey on big data: technologies, terminologies and data-intensive applications," J. Big Data, vol. 9, no. 1, pp. 1–36, Dec. 2022, doi: 10.1186/S40537-022-00659-3.
19. JD Owens, M. Houston, D. Luebke, S. Green, JE Stone, and JC Phillips, "GPU computing," Proceedings of the IEEE, vol. 96, no. 5, pp. 879–899, 2008, doi: 10.1109/JPROC.2008.917757.
20. H. Jeon, "GPU Architecture," Handbook of Computer Architecture, pp. 1–29, 2023, doi: 10.1007/978-981-15-6401-7_66-1.
21. CuPy Development Team, "CuPy — NumPy & SciPy for GPU — CuPy 13.4.0 documentation," Accessed: Mar. 10, 2025. [Online]. Available: https://docs.cupy.dev/en/stable/

 doi:10.20944/preprints202603.0836.v1

22. RG Jha and A. Samlodia, “GPU-acceleration of tensor renormalization with PyTorch using CUDA,” Comput. Phys. Commun., vol. 294, p. 108941, Jan. 2024, doi: 10.1016/J.CPC.2023.108941.
23. PyTorch Development Team, “PyTorch documentation — PyTorch 2.6 documentation,” Accessed: Apr. 06, 2025. [Online]. Available: https://pytorch.org/docs/stable/index.html
24. TensorFlow Development Team, "API Documentation | TensorFlow v2.16.1," Accessed: Apr. 06, 2025. [Online]. Available: https://www.tensorflow.org/api_docs
25. AK Das, N. Gupta, T. Mahmood, BC Tripathy, R. Das, and S. Das, "An innovative fuzzy multi-criteria decision making model for analyzing anthropogenic influences on urban river water quality," Iran Journal of Computer Science, vol. 8, no. 1, pp. 103–124, Nov. 2024, doi: 10.1007/S42044-024-00211-X.
26. AK Das and C. Granados, “FP-intuitionistic multi fuzzy N-soft set and its induced FP-Hesitant N soft set in decision-making,” Decision Making: Applications in Management and Engineering, vol. 5, no. 1, pp. 67–89, Mar. 2022, doi: 10.31181/DMAME181221045D.
27. S. Karadayi-Usta and EB Tirkolaee, “Evaluating the Sustainability of Fashion Brands Using a Neutrosophical ORESTE Approach,” Sustainability, vol. 15, no. 19, p. 14406, Sep. 2023, doi: 10.3390/SU151914406.
28. L. Lamrini, MC Abounaima, FZ El Mazouri, M. Ouzarf, and MT Alaoui, "MCDM Filter with Pareto Parallel Implementation in Shared Memory Environment," Statistics, Optimization & Information Computing, vol. 10, no. 1, pp. 192–203, Feb. 2022, doi: 10.19139/SOIC-2310-5070-1216.
29. L. Lamrini, MC Abounaima, and M. Talibi Alaoui, “New distributed-TOPSIS approach for multi-criteria decision-making problems in a big data context,” J. Big Data, vol. 10, no. 1, Dec. 2023, doi: 10.1186/s40537-023-00788-3.
30. C. Erlacher, S. Salap-Ayca, P. Jankowski, KH Anders, and G. Paulus, "A GPU-based solution for accelerating spatially-explicit uncertainty- and sensitivity analysis in multi-criteria decision making," Proceedings of Spatial Accuracy 2016, pp. 305–312, Jan. 2016.
31. MSY Lakshmi, DR Karpagam, and NG Swetha, “GPU Accelerated TOPSIS Algorithm for QoS Aware Web Service Selection,” International Journal of Recent Technology and Engineering (IJRTE), vol. 8, no. 5, pp. 3159–3163, Jan. 2020, doi: 10.35940/IJRTE.E6502.018520.
32. M. Kumar, G. Kaur, and PS Rana, "Robust evaluation of GPU compute instances for HPC and AI in the cloud: a TOPSIS approach with sensitivity, bootstrapping, and non-parametric analysis," Computing, vol. 106, no. 12, pp. 3987–4014, Dec. 2024, doi: 10.1007/S00607-024-01342-6.
33. NG Swetha and GR Karpagam, “GPU enabled Improved Reference Ideal Method (I-RIM) for Web Service Selection,” Int. J.Inf. Technol. Decis. Mak., vol. 21, no. 3, pp. 855–884, May 2022, doi: 10.1142/S0219622022500055.
34. C.-L. Hwang and K. Yoon, “Methods for Multiple Attribute Decision Making,” in Lecture Notes in Economics and Mathematical Systems, vol. 186, p. 58–191, 1981, doi: 10.1007/978-3-642-48318-9_3.
35. J. Shi, M. Sun, X. Yang, K. Jing, and KK Lai, "Evaluating supply chain finance risks in a cross-border e-commerce context: An improved TOPSIS approach with loss penalty," Inf. Sci. (NY)., vol. 717, p. 122301, Nov. 2025, doi: 10.1016/J.INS.2025.122301.
36. T. Wu, X. Liu, and F. Liu, “An interval type-2 fuzzy TOPSIS model for large scale group decision making problems with social network information,” Inf. Sci. (NY)., vol. 432, pp. 392–410, Mar. 2018, doi: 10.1016/J.INS.2017.12.006.
37. M. Burgin, “Algorithmic complexity as a criterion of unsolvability,” Theor. Comput. Sci., vol. 383, no. 2–3, pp. 244–259, Sep. 2007, doi: 10.1016/J.TCS.2007.04.011.

 doi:10.20944/preprints202603.0836.v1

38. I. Wegener, Complexity Theory: Exploring the Limits of Efficient Algorithms. Berlin, Germany: Springer, 2005, doi: 10.1007/3-540-27477-4.
39. Hou, Y. et al., "Amazon Reviews 2023," 2023. [Online]. Available: https://amazon-reviews-2023.github.io/. Accessed: March 05, 2026.