SCOUT: Coupling-Free Bounds for Trillion-Scale Top-k Retrieval in Sparse Tensor Factorization

JUN-GI JANG, Daegu Gyeongbuk Institute of Science and Technology, Republic of Korea
HYUNSIK YOO, University of Illinois Urbana-Champaign, USA
RUIZHONG QIU, University of Illinois Urbana-Champaign, USA
YINGLONG XIA, Meta, USA
HANGHANG TONG, University of Illinois Urbana-Champaign, USA

Ranked retrieval over massive implicit result spaces is a core query-processing problem in data management. A natural instance arises when using tensor factorization (TF) models for retrieval over sparse tensors, where candidate tuples correspond to unobserved higher-order interactions over a multiway Cartesian product and are scored by a learned TF model. Real-world tensors in domains such as biomedical analysis, recommendation, and temporal interaction modeling are sparse and incomplete, and the candidate space can reach the trillion scale, making exhaustive scoring and sorting infeasible. While most prior work focuses on improving TF training or model accuracy, scalable inference-time top-k ranked retrieval over learned multiway scores has been relatively underexplored. Moreover, existing pairwise top-k search techniques, such as Maximum Inner Product Search (MIPS), do not directly extend to TF due to coupled multiway scores and the combinatorial candidate space.

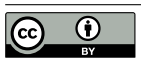
We propose Scout, a scalable and compatible framework for top-k ranked retrieval over massive implicit tensor spaces. Scout reparameterizes TF scores into nonnegative per-mode potentials and a normalized interaction term, yielding a coupling-free product-form bound that enables ranked access, admissible pruning, and reliable candidate ordering, without mode-coupled materialization as in MIPS-style reductions. Using this bound, Scout performs an N-dimensional best-first traversal with deferred scoring, supporting budgeted anytime retrieval and certified stopping when exact top-k is required. Scout further verifies candidates in GPU-friendly block-wise batches, enabling high-throughput execution under memory and latency budgets. Experiments show that Scout achieves over $10^3\times$ speedup over full enumeration for various TF methods and up to 11× over MIPS-based baselines, while maintaining comparable retrieval quality.

CCS Concepts: • **Information systems → Top-k retrieval in databases**; *Database query processing*; • **Computing methodologies → Factorization methods**.

Additional Key Words and Phrases: Sparse tensor factorization, top-k retrieval, coupling-free bounds, ranked query processing

ACM Reference Format:
Jun-Gi Jang, Hyunsik Yoo, Ruizhong Qiu, Yinglong Xia, and Hanghang Tong. 2026. SCOUT: Coupling-Free Bounds for Trillion-Scale Top-k Retrieval in Sparse Tensor Factorization. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 285 (September 2026), 27 pages. https://doi.org/10.1145/3837123

Authors' Contact Information: Jun-Gi Jang, jungi@dgist.ac.kr, Daegu Gyeongbuk Institute of Science and Technology, Daegu, Republic of Korea; Hyunsik Yoo, hy40@illinois.edu, University of Illinois Urbana-Champaign, Champaign, IL, USA; Ruizhong Qiu, rq5@illinois.edu, University of Illinois Urbana-Champaign, Champaign, IL, USA; Yinglong Xia, yxia@meta.com, Meta, Menlo Park, CA, USA; Hanghang Tong, htong@illinois.edu, University of Illinois Urbana-Champaign, Champaign, IL, USA.



ACM 2836-6573/2026/9-ART285
https://doi.org/10.1145/3837123

1 Introduction

Ranked retrieval over implicit result spaces is a well-established query-processing problem in data management. Classical settings include top-k join processing [24], where the goal is to identify the highest-ranked tuples from a join result without full materialization, and ranked retrieval over middleware [15, 16], where scores from multiple sources are aggregated efficiently. The core challenge in these settings is to provide ranked access and admissible pruning without enumerating the full result space.

Sparse tensor retrieval with tensor factorization (TF) models presents a particularly challenging instance of this problem, where candidate tuples are defined by a multiway Cartesian product and scored by a learned TF model. Such settings arise in applications including knowledge graph completion [6, 35, 45], forecasting in time-evolving networks [11], and recommendations [10, 21, 48]. Real-world tensors in these domains are typically sparse and incomplete: only a small fraction of higher-order interactions are observed while the vast majority remain unobserved. For example, in the Yahoo dataset used in this paper, the candidate space contains about 1.13×10^{12} unobserved user–music–time interactions, although only 785,749 are observed. Thus, the retrieval space is a massive virtual relation whose size grows multiplicatively with the mode cardinalities, making exhaustive evaluation infeasible in both time and memory. Figure 1(a) illustrates this challenge. Existing TF approaches require exhaustively searching the massive implicit interaction space before returning the top-k results, leading to prohibitive retrieval costs.

The central problem is therefore not training the scorer itself, but supporting efficient ranked access over the implicit candidate relation induced by the tensor modes. Unlike classical ranked retrieval and top-k join processing, which often assume explicitly ranked inputs or decomposable score components, sparse tensor retrieval involves learned high-order scores over an implicit Cartesian-product space, for which no ranked-access structure is directly available. As a result, straightforward retrieval requires scoring and sorting the full unobserved space, which quickly reaches billions or trillions of candidates. Despite this query-processing bottleneck, most prior TF work [2, 19, 40, 41, 51, 53] focuses on accelerating training rather than inference-time ranked retrieval.

One might consider leveraging top-k search techniques for pairwise scoring functions, such as maximum inner product search (MIPS). MIPS typically assumes a flat query–database setting, where each candidate is a single vector scored by an inner product. In contrast, an N-way TF candidate is a tuple of entities, and reducing it to MIPS requires materializing coupled $(N-1)$-mode vectors, e.g., Khatri–Rao (KR) products [29], whose number is $\prod_{m \neq q} I_m$. Thus, the difficulty is not merely the lack of a faster scoring routine, but the absence of a practical ranked-access structure over the original implicit tuple space. Even when a dot-product rewriting is algebraically possible, it requires flattening multiple modes into composite database items, which does not necessarily provide a scalable ranked-access path over the original multiway tuple space. Consequently, MIPS-style reductions can incur prohibitive preprocessing and memory costs, especially when coupled representations are materialized over large multiway candidate spaces.

We propose Scout, a scalable and compatible framework for top-k ranked retrieval over massive implicit tensor spaces. As illustrated in Figure 1(b), instead of evaluating the full implicit space, Scout prunes unpromising regions using potential-based bounds and selectively verifies only promising interactions. Scout reparameterizes a general TF score into nonnegative per-mode entity potentials and a normalized interaction term, yielding a coupling-free product-form upper bound that serves as a ranked-access path for pruning and prioritization. Unlike MIPS-style reductions that require mode-coupled materialization, Scout relies only on per-mode potentials, avoiding prohibitive memory overhead and sensitivity to the query mode. Table 1 provides a high-level

Table 1. Comparison of SCOUT and competitors for top-k retrieval in sparse tensors. Full enumeration is general but infeasible at scale. MIPS-based approaches can be fast, but typically require memory-heavy mode-coupled materialization and are most direct for CP-style reductions, limiting scalability and compatibility with broader TF scorers. SCOUT achieves fast retrieval with low memory, strong scalability, and broad TF compatibility.

Method	Speed	Memory	Scalability	Compatibility
Full enumeration	✗	✗	✗	✓
MIPS-based approaches	✓	✗	▲	✗
SCOUT (proposed)	✓	✓	✓	✓

comparison of SCOUT with competitors. Building on this access path, SCOUT performs an N-dimensional best-first traversal with deferred exact scoring, supports budgeted anytime retrieval, and provides certified stopping when exact top-k results are required. To exploit GPU parallelism, SCOUT verifies candidates in GPU-friendly block-wise batches, enabling high-throughput retrieval under tight memory and latency budgets. SCOUT applies to both multilinear and nonlinear TF models without modifying the scorer architecture.

Our main contributions are summarized as follows:

- **Ranked Retrieval over Implicit Tensor Spaces.** We formulate sparse-TF inference as ranked query processing over a massive implicit Cartesian-product space, and propose SCOUT, a scorer-preserving framework for ranked access and admissible pruning.
- **Scalable Ranked Retrieval Framework.** SCOUT supports bound-guided N-dimensional traversal with deferred multiway scoring and GPU-efficient block-wise verification, enabling budgeted anytime retrieval as well as certified stopping when exact top-k results are required.
- **Theoretical Analysis.** We establish sublinear interaction-evaluation complexity $O(I^{\rho N}(\log I)^{N-1})$ $(0 < \rho < 1)$, implying an approximate $O(I^{(1-\rho)N}/(\log I)^{N-1})$-factor reduction in interaction-score evaluations relative to exhaustive enumeration.
- **Experiments.** Experiments demonstrate that SCOUT achieves orders-of-magnitude speedups (over $10^3\times$) against full enumeration and outperforms MIPS-based baselines by up to 11×, effectively bridging the gap between diverse TF architectures and real-time retrieval requirements.

Our code and datasets are available at **https://github.com/jungijang/SCOUT**.

2 Preliminaries

Tensor Notation and Operations. An N-th order tensor is an N-dimensional array, denoted by $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, where I_n is the dimensionality of mode n. A vector $\mathbf{x}$ and a matrix $\mathbf{X}$ are first- and second-order tensors, respectively. Let $i_n \in [I_n]$ denote the index of an entity in mode n, and let $\mathbf{i} = (i_1, \ldots, i_N)$ denote an interaction (index tuple). We use the *n-mode matricization* $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{m \neq n} I_m}$, which reshapes $\mathcal{X}$ into a matrix whose rows correspond to mode-n entities and whose columns correspond to joint indices over the remaining modes. We also use the *Khatri-Rao product* $\odot$, i.e., the column-wise Kronecker product: for $\mathbf{A} \in \mathbb{R}^{I \times R}$ and $\mathbf{B} \in \mathbb{R}^{J \times R}$, $(\mathbf{A} \odot \mathbf{B})[:, r] = \mathbf{A}[:, r] \otimes \mathbf{B}[:, r]$. We refer to the coupled representation of selected factor rows across modes as a Khatri–Rao (KR)-style coupled vector. For CANDECOMP/PARAFAC (CP) scoring [8, 22], this coupling is the element-wise product of selected latent vectors and remains in $\mathbb{R}^R$. Here, $\mathbf{a}_{i_n}^{(n)} \in \mathbb{R}^R$ denotes the latent vector of entity i_n in mode n. For further background on tensor notation and operations, see Kolda and Bader [31]. Table 2 summarizes frequently used symbols.

Tensor Factorization. Given an N-order tensor and target rank R, tensor factorization approximates a given tensor into factor matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ and associated parameters θ. The column

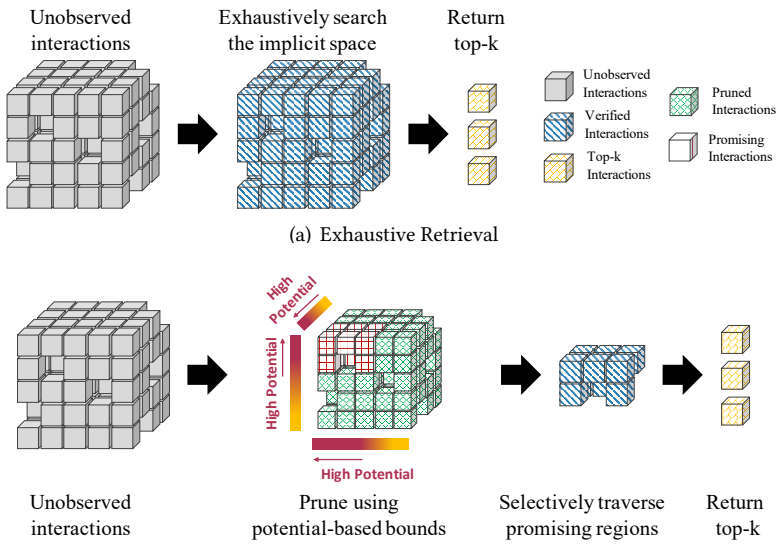


Fig. 1. Scalable top-k retrieval over implicit tensor spaces. Exhaustive retrieval requires searching the entire implicit interaction space, whereas Scout prunes unpromising regions and selectively verifies promising interactions.

size of the factor matrices is R. Then, we can obtain the prediction for a given interaction $(i_1, ..., i_N)$ as follows:

$$s_{(i_1,...,i_N)} \leftarrow f(\{\mathbf{a}_{i_n}^{(n)}\}_{n=1}^{N}; \theta) \tag{1}$$

where f is a scoring function of tensor factorization. $\mathbf{a}_{i_n}^{(n)}$ is the latent vector of the i_nth entity in the nth dimension. The factor matrices $\mathbf{a}_{i_1}^{(1)}, ..., \mathbf{a}_{i_N}^{(N)}$ and associated parameters θ are learned by minimizing an objective function such as a mean squared error (MSE) and a pairwise ranking loss.

Problem Definition. We view inference-time prediction in sparse TF as a ranked retrieval problem over an implicit virtual relation. The primary objective is to identify the top-k higher-order interactions from a vast, unobserved candidate space in a sparse tensor.

Definition 1 (Top-k Prediction in Sparse Tensors). *Given an N-th order sparse tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and a trained tensor factorization (TF) model f with parameters θ, the goal is to find a set of k unobserved interactions $\mathcal{L}_k \subset S_{can}$ such that:*

$$\mathcal{L}_k = \arg \max_{\mathcal{L} \subset S_{can}, |\mathcal{L}|=k} \sum_{(i_1,...,i_N) \in \mathcal{L}} f(\{\mathbf{a}_{i_n}^{(n)}\}_{n=1}^{N}; \theta) \tag{2}$$

where $S_{\text{all}} = [I_1] \times \cdots \times [I_N]$ is the implicit Cartesian-product space of all possible tuples, and $S_{\text{can}} = S_{\text{all}} \setminus S_o$ is the candidate space obtained by excluding the observed interactions S_o. Thus, S_{can} can be viewed as an implicit virtual relation over which the learned TF scorer defines a ranking.

This formulation corresponds to ranked retrieval over the full implicit relation and can also express subset-constrained retrieval by restricting one or more modes, analogous to selection predicates in relational queries.

Table 2. Symbol description.

Symbol	Description
$\mathcal{X}$	sparse tensor
$\mathbf{A}^{(n)}$	n-th factor matrix
$\mathbf{a}_{i_n}^{(n)}$	latent vector representing i_n-th entity of n-th dimension
S_o	set of observed entries of a tensor $\mathcal{X}$
S_{can}	candidate set consisting of unobserved interactions in $\mathcal{X}$
i_n	i_n-th entity index in mode n
$(i_1, ..., i_N)$	interaction in a tensor
I_n	the size of the n-th dimension of a tensor
R	column size of factor matrices
$g_n(\cdot)$	function for predicting entity scores in the n-th dimension
$f(\cdot)$	function for predicting interaction scores
$\tilde{f}(\cdot)$	function for normalized interaction scores $\frac{f(\cdot)}{\prod g_n(\cdot)}$
$s_{(i_1,...,i_N)}$	score of an interaction $(i_1, ..., i_N)$
$\mathcal{L}_k$	list of top-k interactions
$\langle \cdot, \cdot \rangle$	inner product
$\bigodot$	Khatri-Rao product
$\hat{s}_{(i_1,...,i_N)}$	upper bound of the interaction score
π_n	rank-ordering of entities in mode n sorted by $g_n(\cdot)$
$\mathbf{r} = (r_1, \ldots, r_N)$	rank vector in the N-D lattice
τ	current top-k threshold, i.e., the k-th largest verified score
b	block size for GPU-friendly block-wise verification
α	decay factor for aggressive anytime stopping
λ	regularization strength
β	bound-tightness scaling factor

Example 1. Consider a third-order sparse tensor $\mathcal{X}$ representing (Drug A, Drug B, Side Effect) interactions where only a few interactions are observed. This tensor has size $629 \times 645 \times 1{,}317$, yielding more than 534 million possible interactions. Given a learned TF scorer f, the goal of top-k retrieval is to identify the k highest-scoring unobserved interactions, without exhaustively evaluating all candidates in the implicit Cartesian-product space.

For the top-k prediction in a sparse tensor, we can train a TF method using the set S_o of a sparse tensor $\mathcal{X}$. We adopt the Bayesian Personalized Ranking (BPR) loss [39], a standard pairwise ranking objective for implicit-feedback and factorization-based top-k retrieval.

$$loss = \sum\nolimits_{(\mathbf{i}^+, \mathbf{i}^-)} -\log \sigma(s_{\mathbf{i}^+} - s_{\mathbf{i}^-}) \tag{3}$$

where $\mathbf{i}^+$ is an observed interaction in the set S_o of observed interactions, $\mathbf{i}^-$ is a negative sampled interaction paired with $\mathbf{i}^+$, and σ is the sigmoid function.

Limitations of Traditional TF Inference. A variety of tensor factorization (TF) models have been developed, ranging from multilinear decompositions [8, 22, 50] to nonlinear variants [3, 17, 33]. We train a chosen TF scorer using the objective in Eq. (3), and then use the trained scorer to retrieve the top-k unobserved interactions as defined in Definition 1. However, a naïve ranked-access strategy is unavailable: traditional inference requires enumerating and scoring every tuple in S_{can}, whose size grows as $O(\prod_n I_n)$. This is because the TF scorer $f(\cdot)$ assigns scores only to fully specified tuples and does not provide an access path for reaching high-scoring tuples without evaluating a large fraction of S_{can}. While prior work [2, 19, 40, 41, 51, 53] has extensively studied accelerating the *training* phase of TF, such work largely overlooks the computational burden of

inference-time retrieval. As a result, top-k retrieval reduces to a prohibitively expensive full scan over S_{can}, making naïve TF inference infeasible at scale.

Limitations of MIPS-based Approaches. To avoid the prohibitive cost of full enumeration, one potential way is to leverage Maximum Inner Product Search (MIPS). It is designed to efficiently retrieve high-scoring candidates by computing pairwise matches between a query and a set of candidates under an inner-product scoring function. However, MIPS does not directly apply to higher-order TF scoring, since the score depends on *multiple* entity vectors for N dimensions rather than a single query-candidate pair.

To use MIPS in the higher-order setting, one must reduce the N-way scoring to a matrix-form inner product via n-mode matricization. Concretely, under CANDECOMP/PARAFAC (CP) decomposition [8, 22] (i.e., $f_{\text{CP}}(\{\mathbf{a}_{i_n}^{(n)}\}_{n=1}^{N}) = \sum_{r=1}^{R}\prod_{n=1}^{N}\mathbf{a}_{i_n}^{(n)}[r]$), we can rewrite the score as an inner product by treating the mode-1 factor vector as the query and coupling the remaining factors from modes 2 to N into a single candidate representation: $s_{(i_1,\ldots,i_N)} = \left\langle \mathbf{a}_{i_1}^{(1)}, \bigodot_{n=2}^{N}\mathbf{a}_{i_n}^{(n)} \right\rangle$, which allows treating $\mathbf{a}_{i_1}^{(1)}$ as a query and $\bigodot_{n=2}^{N}\mathbf{a}_{i_n}^{(n)}$ as a database vector. However, this reduction requires materializing and/or indexing the coupled $(N-1)$-mode representations for all combinations of $(i_2,\ldots,i_N)$. Concretely, it involves constructing the Khatri-Rao product of the remaining factor matrices, $\mathbf{A}^{(2)}\odot\cdots\odot\mathbf{A}^{(N)} \in \mathbb{R}^{(\prod_{n=2}^{N} I_n)\times R}$, which can incur substantial memory overhead and may even lead to out-of-memory (O.O.M) in practice. In addition, the overhead is highly mode-dependent: choosing a query mode q yields a coupled database of size $\prod_{n\neq q} I_n$, making memory/latency highly sensitive to dataset characteristics.

A MIPS-style reduction gives an algebraic rewriting but not a scalable access path over the original multiway search space. Even for multilinear scorers, it requires choosing a query mode and flattening the remaining modes into $\prod_{n\neq q} I_n$ composite database items, which can dominate memory and preprocessing cost. For example, in the Yahoo tensor, flattening the user–item side would create about 6.8×10^9 composite vectors, making direct materialization impractical. Such reductions are most direct for CP-style scores, where the coupled database vector can be formed via Khatri–Rao products. For Tucker-style scorers, the core tensor induces richer cross-rank interactions, and a comparable reduction generally requires Kronecker-style coupled representations (e.g., $\bigotimes_{n\neq q}\mathbf{a}_{i_n}^{(n)}$), whose dimension grows multiplicatively with the ranks of the coupled modes. For nonlinear TF scorers, exact inner-product reductions are generally unavailable.

Design requirements. Efficient top-k retrieval in higher-order TF must satisfy: (i) **scalability** without full enumeration or memory-heavy coupling, (ii) **compatibility** across diverse TF scorers beyond CP, and (iii) **practicality** under latency/memory budgets.

3 Proposed Method: Scout

In this section, we propose Scout, a scalable and compatible top-k retrieval framework for identifying the highest-scoring unobserved interactions at a trillion scale. Our goal is to make top-k retrieval feasible for existing TF scorers over massive implicit Cartesian-product spaces, without exhaustive enumeration or mode-coupled materialization. We address three challenges:

- **C1. Entangled high-order interactions.** In N-way TF, interaction scores are entangled across modes, making full enumeration infeasible. Due to the structural mismatch, extending 2D retrieval methods requires memory-prohibitive mode coupling (e.g., Khatri–Rao materialization).
- **C2. Scorer-dependent pruning assumptions.** TF scorers range from multilinear models to nonlinear neural scorers, yet existing pruning techniques often rely on model-specific structures (e.g., inner products) that do not generalize to multiway TF scoring.

- **C3. Budgeted retrieval under constraints.** Practical deployments demand fast retrieval under latency budgets and limited GPU memory, requiring memory-aware execution.

We tackle these challenges via three ideas:

- **I1. Scorer-preserving reparameterization.** We reparameterize a TF score into nonnegative per-mode entity potentials and a normalized interaction term without modifying the scorer. This yields admissible bounds for multilinear TF and stable bound-guided prioritization for nonlinear scorers via magnitude control.
- **I2. Coupling-free access path.** From the reparameterized form, we derive a product-form bound over per-mode potentials that directly serves as a coupling-free access path, enabling certified pruning when admissible and reliable prioritization otherwise.
- **I3. Block-wise anytime-to-exact GPU execution.** We verify candidates in bound-guided N-D blocks, using block sizing to control runtime and memory. A two-stage pipeline enables high-throughput anytime retrieval and certified exact stopping when admissible.

Together, these ideas make existing TF scorers retrieval-ready through ranked access, pruning, and GPU-efficient verification.

3.1 Scorer-Preserving Reparameterization with Admissible Bounds

Our primary goal is to efficiently identify the top-k higher-order interactions from a combinatorially large candidate space without resorting to exhaustive enumeration. A key bottleneck in standard TF models is that their scores are inherently coupled across modes, leaving no decomposable criterion to safely prioritize candidates before full evaluation. This entanglement effectively forces a blind search over the entire space, rendering large-scale retrieval intractable. To bypass this limitation, we propose a structural reparameterization that decouples the scoring function into independent per-mode entity potentials and a normalized interaction term. This formulation is designed to be scorer-preserving, ensuring mathematical equivalence to the original model while providing admissible upper bounds for the effective prioritization of unobserved interactions.

Formally, let $s_{(i_1,\ldots,i_N)}$ denote the prediction score for a given interaction tuple $(i_1,\ldots,i_N)$. We reparameterize the scoring function $f(\cdot)$ of any TF model into the following decoupled form:

$$s_{(i_1,\ldots,i_N)} = f(\{\mathbf{a}_{i_n}^{(n)}\}_{n=1}^{N};\theta) = \underbrace{\left(\prod_{n=1}^{N} g_n(\mathbf{a}_{i_n}^{(n)};\theta_n)\right)}_{\text{Entity Potentials}} \cdot \underbrace{\tilde{f}(\{\mathbf{a}_{i_n}^{(n)}\}_{n=1}^{N};\theta)}_{\text{Normalized Interaction}} \tag{4}$$

where $g_n(\cdot)$ is the potential function for the n-th mode entity, and $\tilde{f}(\cdot) \triangleq f(\cdot)/\prod g_n(\cdot)$ is the normalized interaction term. This reparameterization separates computationally cheap, per-entity potentials from the more expensive multiway interaction term and is scorer-preserving by construction.

Crucially, the decoupled form yields an admissible upper bound for efficient search. Define the potential-based bound for a tuple $\mathbf{i} = (i_1,\ldots,i_N)$ as $\hat{s}_\mathbf{i} \triangleq \prod_{n=1}^{N} g_n(\mathbf{a}_{i_n}^{(n)};\theta_n)$. Since $s_\mathbf{i} = \hat{s}_\mathbf{i} \cdot \tilde{f}(\cdot)$, we establish the following guarantee:

Theorem 2 (Admissibility of $\hat{s}$). *If (1) $g_n(\cdot) \geq 0$ for all n and (2) $\tilde{f}(\cdot) \leq 1$, then $\hat{s}_\mathbf{i}$ never underestimates the true score, i.e., $\hat{s}_\mathbf{i} \geq s_\mathbf{i}$ for all $\mathbf{i}$.*

Proof. By Eq. (4), $s_\mathbf{i} = \left(\prod_{n=1}^{N} g_n(\mathbf{a}_{i_n}^{(n)};\theta_n)\right)\tilde{f}(\cdot) = \hat{s}_\mathbf{i}\tilde{f}(\cdot)$. If $g_n(\cdot) \geq 0$ for all n and $\tilde{f}(\cdot) \leq 1$, then $s_\mathbf{i} \leq \hat{s}_\mathbf{i}$, proving admissibility. □

This admissibility allows us to safely prioritize and discard candidates during retrieval without sacrificing exactness:

Corollary 1 (Safe Pruning). *Let τ be the current k-th largest* verified *score during search. Any tuple* $\mathbf{j}$ *with* $\hat{s}_{\mathbf{j}} < \tau$ *can be safely pruned without full evaluation, as it is guaranteed not to belong to the exact top-k.*

Proof. For any tuple $\mathbf{j}$, admissibility gives $s_{\mathbf{j}} \le \hat{s}_{\mathbf{j}}$. If $\hat{s}_{\mathbf{j}} < \tau$, then $s_{\mathbf{j}} < \tau$, so $\mathbf{j}$ cannot appear among tuples whose true scores are at least the current k-th largest verified score τ. Thus, pruning $\mathbf{j}$ cannot remove any true top-k tuple. □

The two conditions in Theorem 2 are addressed by the design of our potentials. First, we construct $g_n(\cdot)$ to be nonnegative by design with norm- or magnitude-based functions. Second, the scaling and normalization ensure $\tilde{f}(\cdot) \le 1$ for multilinear models; for nonlinear TF models, we additionally apply a lightweight score-range constraint to encourage this condition. This reformulation applies to both multilinear and nonlinear TF models; for multilinear models, it converts norm-based entity magnitudes into a tuple-level product-form upper bound. Although simple norm-based ordering can prioritize individual entities, it does not by itself define a tuple-level admissible bound, a monotone multiway access path, or a certified stopping condition. By separating nonnegative mode-wise potentials from the normalized interaction term, our framework yields the product-form upper bound in Theorem 2, which enables the bound-guided lattice traversal in Section 3.2. In the following, we instantiate this design for both multilinear and nonlinear TF models.

3.1.1 Instantiation for Multilinear TF Models. We focus on the CP decomposition [8, 22] and Tucker decomposition [50] which have been widely used in various tensor-based applications; we also discuss M^{2}DMTF [17], a variant of Tucker decomposition. For these models, admissibility is inherent. By defining the entity potential $g_n(\cdot)$ as the vector norm (e.g., L_2 norm), the non-negativity constraint (Constraint 1) is automatically met. Furthermore, the Cauchy-Schwarz inequality (or its generalized Hölder variant) guarantees that the interaction score is upper-bounded by the product of component norms, thereby satisfying the bound constraint (Constraint 2) without any architectural changes.

Instantiation of CP. We demonstrate our instantiation on CP decomposition, a representative multilinear TF model. Consider the CP decomposition with factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$. For a tuple $(i_1, \ldots, i_N)$, the CP score is

$$s_{(i_1,\ldots,i_N)} = \sum_{r=1}^{R} \prod_{n=1}^{N} \mathbf{A}^{(n)}[i_n, r] = \left\langle \mathbf{a}^{(1)}_{i_1}, \bigodot_{n=2}^{N} \mathbf{a}^{(n)}_{i_n} \right\rangle, \tag{5}$$

where $\bigodot$ denotes the Hadamard product over the selected latent vectors. We set $g_n(\mathbf{a}^{(n)}_{i_n}) \triangleq \|\mathbf{a}^{(n)}_{i_n}\|_2$ and $\bar{\mathbf{a}}^{(n)}_{i_n} \triangleq \mathbf{a}^{(n)}_{i_n} / \|\mathbf{a}^{(n)}_{i_n}\|_2$. Then

$$s_{(i_1,\ldots,i_N)} = \underbrace{\prod_{n=1}^{N} \|\mathbf{a}^{(n)}_{i_n}\|_2}_{\hat{s}_{(i_1,\ldots,i_N)}} \cdot \underbrace{\left\langle \bar{\mathbf{a}}^{(1)}_{i_1}, \bigodot_{n=2}^{N} \bar{\mathbf{a}}^{(n)}_{i_n} \right\rangle}_{\tilde{f}(\cdot)}. \tag{6}$$

Nonnegativity holds by definition, and by Cauchy–Schwarz with $\|\bar{\mathbf{a}}^{(n)}_{i_n}\|_2 = 1$ we have $\tilde{f}(\cdot) \le 1$. Hence, $\hat{s}_{(i_1,\ldots,i_N)} = \prod_{n=1}^{N} \|\mathbf{a}^{(n)}_{i_n}\|_2$ is an admissible bound for CP scores.

3.1.2 Instantiation for Nonlinear TF Models. We can use nonlinear tensor factorization methods including CostCo [33] and NeAT [3]; we can also use MLP (Multi-Layer Perceptron) where its input is the concatenation of the latent vectors of entities. For neural architectures where geometric inequalities do not directly apply, we enforce nonnegativity by design and encourage the second admissibility condition through regularization. For the first constraint, our idea is to design the function $g_n(\cdot)$ as follows:

$$g_n(x) = 1 + \text{softplus}(\text{MLP}_n(x)). \tag{7}$$

This function satisfies the first constraint (i.e., $g_n(x) > 0$) since the softplus function [14] (i.e., $\log(1+e^x)$) generates nonnegative value. We add +1 because if $0 < g_n(x) \leq 1$, the product $\prod_n g_n(\cdot)$ can become too small, which can make the normalized interaction term $\tilde{f}(\cdot) = f(\cdot)/\prod_n g_n(\cdot)$ excessively large and destabilize training.

We then need to satisfy the second constraint $\tilde{f}(\cdot) \leq 1$. A simple approach would involve using an activation function that produces values less than or equal to 1, such as the sigmoid or tanh function. However, this degrades the performance of a nonlinear TF method by limiting its representational power. To encourage the second constraint without compromising the model's expressiveness, we add a regularization term that serves as a soft constraint on the interaction magnitude, encouraging the normalized interaction term to remain bounded without distorting the vector angles required for capturing complex patterns. In neural network-based models, $f(\cdot)$ is typically computed as $\mathbf{w}^T\mathbf{o}_{(i_1,\ldots,i_N)} + b$, where $\mathbf{w}$ and b are the weight and bias of the final layer and $\mathbf{o}_{(i_1,\ldots,i_N)}$ is the single representation vector of a given interaction. We add the following regularization term in the loss function:

$$\lambda * \text{softplus}\left(\beta(\|\mathbf{w}\|_2\|\mathbf{o}_{(i_1,\ldots,i_N)}\|_2 + |b|) - \left(\prod_{n=1}^{N} g_n(\mathbf{a}_{i_n}^{(n)}; \theta_n)\right)\right) \tag{8}$$

where λ and $\beta(\geq 1)$ are the regularization hyperparameter and scaling factor, respectively. Here, λ controls how strongly the regularization is enforced, while β determines the target bound. The term $\|\mathbf{w}\|_2\|\mathbf{o}_{(i_1,\ldots,i_N)}\|_2 + |b|$ serves as an upper bound for $\mathbf{w}^\top\mathbf{o}_{(i_1,\ldots,i_N)} + b$ (i.e., $f(\cdot)$), according to the Cauchy–Schwarz inequality. Thus, the regularization encourages $\tilde{f}(\cdot) \leq \frac{1}{\beta}$. A larger λ enforces this constraint more strongly, and a larger β makes the target bound tighter. Using $\|\mathbf{w}\|_2\|\mathbf{o}_{(i_1,\ldots,i_N)}\|_2$ instead of $\mathbf{w}^T\mathbf{o}_{(i_1,\ldots,i_N)} + b$ avoids regularization on the angle $\frac{\mathbf{w}^T\mathbf{o}_{(i_1,\ldots,i_N)}}{\|\mathbf{w}\|_2\|\mathbf{o}_{(i_1,\ldots,i_N)}\|_2}$ between vectors, helping reduce the risk of adversely affecting expressiveness.

Remark. For nonlinear TF models, admissibility is conditional on satisfying $\tilde{f}(\cdot) \leq 1$. The scaling factor β in Eq. (8) serves as a practical control knob: increasing β tightens the magnitude constraint and helps enforce admissibility. When it holds, $\hat{s}_\mathbf{i}$ is a certified upper bound, enabling safe pruning and exact top-k retrieval.

Instantiation of CostCo. We demonstrate our instantiation on CostCo [33], a representative nonlinear TF model, and defer additional instantiations (e.g., NeAT and MLP-based variants). Given a tuple $(i_1, \ldots, i_N)$, CostCo first stacks the N latent vectors into a matrix $\mathbf{X} \in \mathbb{R}^{N\times R}$, where the n-th row is $\mathbf{a}_{i_n}^{(n)}$. It then applies convolutional layers to $\mathbf{X}$ to obtain an interaction representation $\mathbf{o}_{(i_1,\ldots,i_N)}$, which is finally scored by an MLP head as $f(\cdot) = \mathbf{w}^\top\mathbf{o}_{(i_1,\ldots,i_N)} + b$. To encourage the bound condition $\tilde{f}(\cdot) \leq 1$ in Theorem 2 without hard-saturating the nonlinear scorer (e.g., via sigmoid/tanh), we add the regularizer in Eq. (8) to the training objective to control the magnitude of the normalized interaction term. This helps obtain an admissible bound $\hat{s}$ during search when the condition holds.

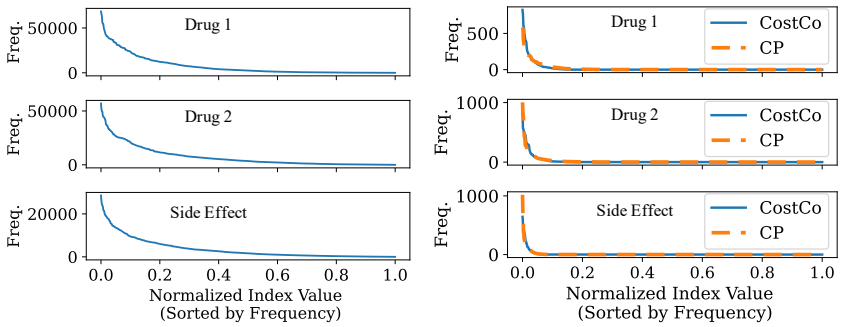
Algorithm 1: Bound-Guided Lattice Traversal for Top-k

Require: Entity potentials $g_n(\cdot)$, interaction function $f(\cdot)$, observed tuple set S_o, factor matrices $\mathbf{A}^{(n)}$, and k.
Ensure: Top-k interaction list $\mathcal{L}_k$.

For each mode $n \in \{1, \ldots, N\}$, sort entities in descending order of $g_n(\mathbf{a}_{i_n}^{(n)})$ to obtain the rank-ordering π_n.
Initialize a min-heap $\mathcal{L}_k$ to store top-k verified scores (threshold $\tau \leftarrow \mathcal{L}_k$.peek() if $|\mathcal{L}_k| = k$, else $-\infty$).
Initialize a max-priority queue Q with the origin rank vector $\mathbf{r}_{start} = (1, \ldots, 1)$.
while Q is not empty **do**
  Pop $\mathbf{r} = (r_1, \ldots, r_N)$ with the highest upper bound $\hat{s}(\mathbf{r}) = \prod_{n=1}^{N} g_n(\pi_n(r_n))$ from Q.
  if $|\mathcal{L}_k| = k$ **and** $\hat{s}(\mathbf{r}) < \tau$ **then**
    break
  end if
  Compute exact score $s_{\mathbf{i}} = f(\{\mathbf{a}_{\pi_n(r_n)}^{(n)}\}_{n=1}^{N})$ based on Eq. (4).
  if tuple($\mathbf{r}$) $\notin S_o$ **and** ($|\mathcal{L}_k| < k$ or $s_{\mathbf{i}} > \tau$) **then**
    $\mathcal{L}_k$.push($s_{\mathbf{i}}$, tuple($\mathbf{r}$))
    if $|\mathcal{L}_k| > k$ **then**
      $\mathcal{L}_k$.pop()
    end if
    $\tau \leftarrow \mathcal{L}_k$.peek()
  end if
  for each successor $\mathbf{r}'$ of $\mathbf{r}$ in the N-D lattice **do**
    if $\mathbf{r}'$ has not been visited **then**
      Q.push($\mathbf{r}', \hat{s}(\mathbf{r}')$)
    end if
  end for
end while
return $\mathcal{L}_k$

3.2 Bound-Guided N-D Lattice Traversal for Top-k Retrieval

Next, we leverage the admissible bound $\hat{s}$ in Section 3.1 to develop an exact top-k retrieval algorithm over the unobserved candidate space $S_{\text{can}} = \left(\mathcal{I}^{(1)} \times \cdots \times \mathcal{I}^{(N)}\right) \setminus S_o$. Note that observed tuples in S_o are excluded before insertion into the top-k list. Our key idea is to impose a search-friendly structure on S_{can} by exploiting the factorized form $\hat{s}_{\mathbf{i}} = \prod_{n=1}^{N} g_n(\mathbf{a}_{i_n}^{(n)}; \theta_n)$. Specifically, for each mode n, we sort the entities in descending order of their potential values g_n and denote the resulting rank-ordering by π_n. This induces an N-dimensional lattice over rank vectors $\mathbf{r} = (r_1, \ldots, r_N)$, where each node $\mathbf{r}$ maps to the interaction tuple $(\pi_1(r_1), \ldots, \pi_N(r_N))$ and inherits the bound $\hat{s}_{\mathbf{r}} = \prod_{n=1}^{N} g_n(\mathbf{a}_{\pi_n(r_n)}^{(n)}; \theta_n)$. Since each list π_n is sorted by g_n in descending order, $\hat{s}(r)$ is monotonically non-increasing in each coordinate, i.e., $\hat{s}(r') \leq \hat{s}(r)$ for any $r' \succeq r$. Unlike Fagin-style rank aggregation [15], which performs ranked access on sorted lists of *true* scores, we use $\hat{s}_{\mathbf{i}}$ only as an admissible upper bound to prioritize exploration and defer evaluating the original TF score $s_{\mathbf{i}} = f(\cdot)$. Moreover, in multiway TF, ranked access to true scores is nontrivial without explicit mode coupling, whereas our lattice is constructed directly from per-mode potentials in a coupling-free manner. Here, coupling-free means that we never materialize mode-coupled vectors (e.g., Khatri–Rao products); instead, $\hat{s}_{\mathbf{i}} = \prod_{n=1}^{N} g_n(\cdot)$ is computed on-the-fly via scalar composition across modes.



(a) Distributions of Entity Frequencies (Input data)

(b) Distributions of Entity Frequencies (Top-10^4 predictions)

Fig. 2. Observation on DDS dataset. Details are described in Section 3.3.1.

Crucially, $\hat{s}_{\mathbf{r}}$ is monotone on this lattice: moving away from the origin $(1, \ldots, 1)$ along any dimension cannot increase the bound. Therefore, we perform a bound-guided best-first traversal that systematically expands the frontier node with the largest $\hat{s}$, evaluates its true score s, and updates the current threshold τ (i.e., the k-th largest verified score). Leveraging admissibility, we can safely discard any node whose bound falls below τ; furthermore, by monotonicity, this pruning extends to the entire dominated sub-lattice originating from that node. This targeted exploration concentrates on computing high-potential regions and provides a certified stopping condition for exact top-k retrieval.

The detailed procedure for constructing the exact top-k list $\mathcal{L}_k$ using the trained model is as follows (see also Algorithm 1):

(1) **Preprocessing:** For each mode $n \in \{1, \ldots, N\}$, we sort the entities in descending order of their potentials $g_n(\cdot)$ to establish the rank-ordering π_n.
(2) **Lazy Best-First Search:** We explore the N-dimensional lattice using a best-first search strategy [11] guided by the upper bound $\hat{s}(\mathbf{r})$. To avoid the prohibitive cost of precomputing all possible interaction bounds, we incrementally expand the frontier of the lattice. A priority queue is maintained to store candidate rank vectors $\mathbf{r}$, prioritized by $\hat{s}(\mathbf{r})$.
(3) **Deferred Evaluation:** For each rank vector $\mathbf{r}$ extracted from the priority queue, we compute the exact score $s = f(\cdot)$ only at the time of its evaluation. If s exceeds the current k-th largest verified score τ in $\mathcal{L}_k$, we update $\mathcal{L}_k$.
(4) **Pruning and Termination:** We stop the traversal as soon as the maximum bound in the priority queue, $\hat{s}_{max}$, falls below the threshold τ. By the admissibility and monotonicity of $\hat{s}$, any unexplored nodes in the lattice (and their dominated sub-lattices) are guaranteed to have scores smaller than τ, enabling safe pruning without evaluation.

Note that $\mathcal{L}_k$ is efficiently managed as a min-priority queue of size k to facilitate rapid updates of the threshold τ.

Example. As illustrated in Figure 1(b) for a third-order case, the search concentrates only on high-potential regions across all modes. In this scenario, only the top-2 entities from the first and third modes and the top-3 from the second mode are ever evaluated. Consequently, the top-1 is identified by examining only $2 \times 3 \times 2$ candidates, safely ignoring the remaining combinatorial space.

Algorithm 2: GPU-Friendly Block-wise Anytime Retrieval

Require: Potentials $g_n(\cdot)$, normalized interaction $\tilde{f}(\cdot)$, observed tuple set S_o, block size b, top-k, decay α, anytime flag *stop*.
Ensure: Top-k interaction list $\mathcal{L}_k$.

Sort entities in each mode n by potentials g_n; partition them into M_n blocks $\{B_{n,1}, \ldots, B_{n,M_n}\}$.
Compute block-max potentials: $\mathcal{G}_{n,j} \leftarrow \max_{e \in B_{n,j}} g_n(e)$.
Generate block combinations $\mathbf{J} = (j_1, \ldots, j_N)$ and sort them by $\hat{S}_{\mathbf{J}} = \prod_n \mathcal{G}_{n,j_n}$ in descending order.
Initialize $\mathcal{L}_k$ as a size-k buffer filled with $-\infty$ on GPU; $\tau \leftarrow -\infty$, $m_{\max} \leftarrow 0$.
for each block combination $\mathbf{J}$ in sorted order, indexed by $c = 0, 1, \ldots$ **do**
  if *stop* is **true and** $c > 0$ **and** $\hat{S}_{\mathbf{J}} \cdot (\alpha^c m_{\max}) < \tau$ **then**
    break ▷ *Anytime stopping via block bound*
  else if *stop* is **false and** $\hat{S}_{\mathbf{J}} < \tau$ **then**
    break ▷ *Certified exact stopping*
  end if
  Generate candidate set $C = B_{1,j_1} \times \cdots \times B_{N,j_N}$ via Cartesian product.
  Pack candidate indices in C into batched GPU tensors.
  $\mathbf{p} \leftarrow$ compute potential products $\prod_n g_n(i_n)$ for all $\mathbf{i} \in C$ in parallel on GPU.
  $\mathcal{P} \leftarrow$ **Pre-filter** C where $p_{\mathbf{i}} > \tau$.
  if $\mathcal{P}$ is empty **then**
    continue
  end if
  $\tilde{\mathbf{f}} \leftarrow$ compute normalized interactions for all $\mathbf{i} \in \mathcal{P}$ in parallel on GPU.
  $m_{\max} \leftarrow \max(m_{\max}, \max(\tilde{\mathbf{f}}))$.
  Compute final scores $\mathbf{s} \leftarrow \mathbf{p}_{\mathcal{P}} \odot \tilde{\mathbf{f}}$. ▷ *Scoring on GPU*
  $\mathcal{V} \leftarrow$ filter $\{(s_{\mathbf{i}}, \mathbf{i}) : \mathbf{i} \in \mathcal{P}\}$ where $s_{\mathbf{i}} > \tau$ and $\mathbf{i} \notin S_o$.
  if $\mathcal{V}$ is not empty **then**
    $\mathcal{L}_k \leftarrow$ top-$k(\mathcal{L}_k \cup \mathcal{V})$.
    $\tau \leftarrow \min(\mathcal{L}_k)$.
  end if
end for
return $\mathcal{L}_k$

3.3 Anytime Retrieval and GPU-Friendly Block-wise Verification

3.3.1 Anytime Top-k Retrieval via Aggressive Pruning. In scenarios requiring near-instantaneous responses, the exact stopping condition in Theorem 2 may still examine a significant portion of the space if the potentials $\hat{s}$ remain marginally above the threshold τ. While this stopping rule is certified when admissibility holds, we primarily use the anytime mode in large-scale settings to achieve reliable high-precision retrieval under strict latency budgets. To further accelerate retrieval, we propose an anytime retrieval mode that employs an aggressive stopping criterion. Specifically, we modify the termination threshold by incorporating the observed distribution of the interaction terms. Let m_c denote the maximum normalized interaction value observed before processing the block indexed by c. We terminate the search if:

$$\tau > (m_c \cdot \alpha^c) \cdot \hat{s}, \tag{9}$$

where $\alpha \in (0, 1]$ is a decay hyperparameter. This aggressive bound is based on two empirical observations: (1) the normalized interaction $\tilde{f}(\cdot)$ is often significantly lower than its theoretical

upper bound of 1, and (2) the likelihood of identifying a new top-k interaction tends to decrease as the traversal moves farther from the origin, which we model using an exponential decay factor. By tuning α, SCOUT can navigate the trade-off between efficiency and accuracy. Smaller α triggers earlier termination for faster anytime retrieval, while $\alpha = 1$ (with $m_c = 1$) recovers exact stopping. This anytime capability ensures that our framework remains robust and scalable even under strict latency constraints.

We further discuss the aforementioned empirical observations using the DDS dataset. As representative multilinear and nonlinear TF models, we train CP and CostCo [33], respectively. First, we verify that the normalized interaction term $\tilde{f}(\cdot)$ is often far below its theoretical upper bound of 1 over the entire interaction space. Specifically, we observe $\mathbb{E}[\tilde{f}] \approx 0.02$ (std. ≈ 0.14) with $\max \tilde{f} \approx 0.53$ for CP, and $\mathbb{E}[\tilde{f}] \approx 3.2 \times 10^{-3}$ (std. $\approx 4.5 \times 10^{-3}$) with $\max \tilde{f} \approx 0.22$ for CostCo. This suggests that the theoretical bound $\tilde{f}(\cdot) \leq 1$ is far from tight in practice, motivating the use of an aggressive stopping rule. Second, we analyze the frequency distributions of entities in the observed data and in the top-10^4 predicted interactions. Figure 2(a) shows that entity frequencies in each dimension are highly skewed: only a small number of entities appear frequently, while the majority are rarely observed. This skewness is also evident in the top-10^4 predictions in Figure 2(b), where a limited subset of entities appears with high frequency. Collectively, these observations suggest that top-k interactions concentrate on a small subset of entities and that the normalized interaction term is typically far below its theoretical bound, supporting the design of our method.

3.3.2 GPU-friendly Block-wise Verification. While the best-first traversal in Algorithm 1 significantly reduces the number of evaluations, its one-at-a-time verification limits GPU utilization. To address this, we propose a block-wise enhancement that evaluates interaction scores in parallel. The bound-guided traversal in Algorithm 1 is inherently sequential because candidates are expanded one at a time according to the current priority queue and threshold. In contrast, GPUs require regular batched computation to achieve high throughput. Thus, the key challenge is to preserve the pruning benefit of bound-guided search while exposing enough parallel work to the GPU. Specifically, we partition the sorted entities in each mode n into M_n contiguous blocks $\{B_{n,1}, B_{n,2}, \ldots, B_{n,M_n}\}$, and let $M_B = \prod_{n=1}^{N} M_n$ denote the number of block combinations. By taking one block from each mode, we define a block-interaction $\mathcal{B} = B_{1,j_1} \times \cdots \times B_{N,j_N}$, representing a Cartesian product of candidates. Each block-interaction $\mathcal{B}$ inherits an upper bound $\hat{s}(\mathcal{B}) = \prod_n \max_{e \in B_{n,j_n}} g_n(e)$, which, by our sorting, is simply the product of the potentials of the first entities in each constituent block. We materialize and sort these M_B block combinations at the beginning of each retrieval by their bounds $\hat{s}(\mathcal{B})$, requiring $O(M_B \log M_B)$ upfront time and $O(NM_B)$ space; they are then processed in descending order. When a block-interaction $\mathcal{B}$ is processed, we do not blindly evaluate all b^N tuples. Instead, a lightweight pre-filter within each block screens candidates based on their individual potential products, ensuring that only promising survivors invoke the expensive interaction term in batched GPU kernels. In practice, after forming the candidate set C from each block-interaction $\mathcal{B}$, we pack candidate indices into batched GPU tensors. The potential products $\prod_n g_n(i_n)$ are computed for all candidates in the block via batched GPU operations. Only the candidates passing this bound-based pre-filter invoke the normalized interaction computation $\tilde{f}(\cdot)$, which is also evaluated in parallel on GPU. The final scores are then obtained by element-wise multiplication of the potential products and normalized interaction values. This two-tier approach combines lattice-level pruning and block-level parallelization, allowing the theoretical pruning benefits to translate into substantial wall-clock speedups on modern hardware.

The procedure, integrating the aggressive pruning strategy and block-wise verification, is detailed in Algorithm 2.

3.4 Complexity Analysis

We analyze the time and space complexities of Scout.

Notation and Assumptions. For each mode n, define $g_n(i_n) := g_n(\mathbf{a}_{i_n}^{(n)}; \theta_n)$. Without loss of generality, assume that g_n is sorted in non-increasing order with respect to i_n, i.e., $g_n(i_n)$ is the i_n-th largest entry of g_n. Motivated by the observation described in Section 3.3.1 (See Figure 2), we assume a power-law decay:

$$g_n(i_n) \le \mathrm{O}(i_n^{-\gamma_n}) \quad \text{for some } \gamma_n > 0. \tag{10}$$

Let $\gamma_{\min} := \min_{n\in[N]} \gamma_n$. Let s_* denote the k-th largest score among all candidates $s(i_1, \dots, i_N)$. We further assume that s_* is not too small, namely,

$$s_* \ge \Omega\left(I^{-\rho\gamma_{\min}N}\right) \quad \text{for some constant } 0 < \rho < 1, \tag{11}$$

where I denotes the (common) mode size for simplicity.[1] Throughout, we treat the number of modes N as a constant. Finally, we assume that the constraints in Theorem 2 hold.

Theoretical Analysis. The time complexity of Scout is as follows:

Theorem 3 (Sublinear interaction-evaluation complexity). *Under the assumptions stated above, Scout evaluates interaction scores for* $\mathrm{O}(I^{\rho N}(\log I)^{N-1})$ *candidates, where* $0 < \rho < 1$. *Consequently, the corresponding interaction-scoring cost is* $\mathrm{O}(T_{\mathrm{f}} \cdot I^{\rho N}(\log I)^{N-1})$. *This scoring complexity is sublinear with respect to the tensor size* I^N.

Interpretation. The larger s_* and $\gamma_{\min}$ are, the faster our algorithm is. On the other hand, if s_* is too small (e.g., the signal is too low) or $\gamma_{\min}$ is too small (e.g., all i's are equally important), then ρ would be large, and our algorithm would reduce to brute force.

Proof sketch. Since $g_n(i_n) \le \mathrm{O}(1/i_n^{\gamma_n})$, there exists a constant C_n such that $g_n(i_n) \le C_n/i_n^{\gamma_n}$. Since $s_* \ge \Omega(1/I^{\rho\gamma_{\min}N})$, there exists a constant C_* such that $s_* \ge C_*/I^{\rho\gamma_{\min}N}$. Then, let $C := \left(\left(\prod_{n=1}^N C_n\right)/C_*\right)^{1/\gamma_{\min}}$. Note that the number of checked interactions is upper bounded by

$$\begin{aligned}
&\#\left\{(i_1,\dots,i_N)\in[I]^N : \prod_{n=1}^N g_n(i_n) \ge s_*\right\} \le \#\left\{(i_1,\dots,i_N)\in[I]^N : \prod_{n=1}^N \frac{C_n}{i_n^{\gamma_n}} \ge \frac{C_*}{I^{\rho\gamma_{\min}N}}\right\} \\
&\le \#\left\{(i_1,\dots,i_N)\in[I]^N : \prod_{n=1}^N i_n \le CI^{\rho N}\right\} \le \#\left\{(i_1,\dots,i_N)\in\mathbb{N}_+^N : \prod_{n=1}^N i_n \le CI^{\rho N}\right\} \\
&\le \mathrm{O}\left(\frac{CI^{\rho N}(\ln(CI^{\rho N}))^{N-1}}{(N-1)!}\right) = \mathrm{O}(I^{\rho N}(\log I)^{N-1}),
\end{aligned} \tag{12}$$

where the last inequality follows from a standard divisor-counting bound. If evaluating one interaction score requires T_{f} time, the corresponding interaction-scoring cost is $\mathrm{O}(T_{\mathrm{f}} \cdot I^{\rho N}(\log I)^{N-1})$. □

Note that the anytime criterion in Eq. (9) is weaker than the exact stopping rule $\tau > \hat{s}$, and therefore it terminates no later than the exact mode since $\tilde{f}(\cdot) \le 1$ (hence $m_c \le 1$) and $\alpha^c \le 1$. This reduces the explored multiway lattice region to a standard divisor-counting problem over integer tuples with bounded product.

The space complexity of Scout is as follows:

[1] The same analysis extends to heterogeneous mode sizes.

Table 3. Comparison of time and space complexities. Here, I, N, and R denote the number of items per mode, number of modes, and embedding dimension; T_{f} and S_{f} denote the time and space for computing one interaction score; M_B denotes the number of block combinations. ρ denotes the method-specific exponent used by each approach. Space complexities report peak working memory, including batch-wise scoring buffers with batch size B; for Scout, $B = b^N$. For MIPS-based approaches, the per-score cost is $\Theta(R)$.

Method	Time Complexity	Space Complexity
Full enumeration	$\mathrm{O}\big(T_{\mathrm{f}} \cdot I^N\big)$	$\mathrm{O}(NIR + B \cdot S_{\mathrm{f}})$
MIPS-based approach	$\mathrm{O}\Big(R \cdot I^{1+\rho(N-1)}\Big)$	$\mathrm{O}\big(I^{N-1}R\big)$
Scout (Proposed)	$\mathrm{O}\big(T_{\mathrm{f}} \cdot I^{\rho N}(\log I)^{N-1}\big)$	$\mathrm{O}(NIR + NM_B + B \cdot S_{\mathrm{f}})$

Theorem 4 (Space complexity). *Assume that each candidate set generated via a Cartesian product contains at most b^N tuples, where $b \ll I$, and let $M_B = \prod_{n=1}^{N} M_n$ denote the number of block combinations. Let S_{f} denote the peak working memory required to evaluate the score $s_{(i_1,\ldots,i_N)}$ for a single candidate. Then, $\mathrm{O}\big(NIR + NI + NM_B + b^N \cdot S_{\mathrm{f}} + k\big)$ is the space complexity of Scout.*

Proof sketch. The bound follows by accounting for the memory required for (i) factor matrices, (ii) per-entity potentials, (iii) the M_B materialized and sorted block combinations, (iv) temporary block buffers for batched verification, and (v) the top-k heap. □

For brevity, since $R \geq 1$, NI is dominated by NIR. We keep NM_B explicit, since it is not necessarily dominated by NIR, and write the space complexity as $\mathrm{O}\big(NIR + NM_B + b^N \cdot S_{\mathrm{f}}\big)$.

Complexity Comparison. Table 3 summarizes the time and space complexities of full enumeration, MIPS-based reductions, and Scout. For Scout, the reported time complexity corresponds to the interaction-scoring cost after block ordering, while the upfront block-ordering cost is $O(M_B \log M_B)$. Full enumeration scores all I^N candidates, incurring $\mathrm{O}(T_{\mathrm{f}} I^N)$ time, which is infeasible at the trillion scale. After block ordering, Scout evaluates interaction scores for only a sublinear number of candidates, incurring $\mathrm{O}\big(T_{\mathrm{f}} \cdot I^{\rho N}(\log I)^{N-1}\big)$ scoring cost with $0 < \rho < 1$. This yields a substantial reduction in expensive interaction evaluations relative to exhaustive scoring.

MIPS-based approaches necessitate a query-specific Khatri–Rao materialization of size I^{N-1}. Their memory footprint is thus dominated by $\mathrm{O}(I^{N-1}R)$, which quickly becomes impractical as I (or N) grows. In addition, this coupled materialization also makes memory highly sensitive to the choice of the query mode. By contrast, Scout eliminates coupled-vector materialization and performs bound-guided search over per-mode ranked lists with deferred multiway score verification. As a result, the peak memory is $\mathrm{O}(NIR + NM_B + B \cdot S_{\mathrm{f}})$ under batch-wise scoring, where $B = b^N$ is controlled by the block size. Overall, Scout improves scalability while avoiding the coupled-materialization bottleneck inherent to MIPS-style reductions.

4 Experiments

We evaluate Scout by addressing the following questions:

(1) **Performance Comparison (Section 4.2).** How does Scout compare with MIPS-based reductions in terms of running time, accuracy, and memory on large sparse tensors?
(2) **Compatibility (Section 4.3).** Can Scout be seamlessly integrated with diverse TF scorers (both multilinear and nonlinear) without modifying their architectures?
(3) **Scalability (Section 4.4).** Does Scout exhibit improved scalability as the candidate space grows?
(4) **Ablation Study (Section 4.5).** Is $\tilde{f}(\cdot)$ necessary, and how effective is Scout under CPU-only execution?

(5) **Hyperparameter Sensitivity (Section 4.6).** How sensitive is SCOUT to the key hyperparameters and retrieval size k?
(6) **Extension (Section 4.7).** Can SCOUT generalize to other higher-order retrieval tasks beyond sparse tensors?
(7) **Case study (Section 4.8).** Can SCOUT surface meaningful top-ranked interactions in real applications?

4.1 Experiment Setting

In this section, we provide experimental settings for this paper.

Environment. We use a workstation equipped with an NVIDIA RTX Pro 6000 Max-Q.

Dataset. Table 4 summarizes the datasets used in this paper. Ohmnet[2] [56] dataset represents a tensor of the form (protein, protein, human tissue). DDS[3] [55] dataset has (drug, drug, side effect) interactions. Gowalla[4] dataset is POI recommendation data whose form is (user, location, weekly time slot). The weekly time slot is a recurring 30 minute interval within a week (e.g., Monday 13:00-13:30), and the same slot in different weeks shares a common index. Reddit[5] [37] dataset is a sparse tensor, where the dimensions correspond to users, subreddits, and timestamps, with the posted status serving as the observed interactions. FB15k dataset[6] [7] consists of knowledge triples (subject, object, predicate). Yahoo[7] dataset is a large-scale tensor where its form is (user, music, timestamp), and the number of unobserved interactions is trillion-scale.

TF Methods. We use the following multilinear and nonlinear tensor decomposition methods: **(1) CP decomposition** [8, 22], **(2) Tucker decomposition** [50], **(3) M^{2}DMTF** [17], **(4) CostCo** [33], **(5) NeAT** [3], and **(6) MLP**. The first three methods are multilinear tensor methods, while the remaining three are nonlinear. We improve the scalability of the above methods using SCOUT. For the nonlinear TF models, SCOUT uses a lightweight potential network for all $g_n(\cdot)$.

MIPS-based Approaches. We consider representative algorithmic families for maximum inner product search (MIPS). **(1) Faiss [13, 28] (IndexFlatIP)** serves as a GPU-optimized brute-force approach. **(2) LEMP [47] and FEXIPRO [32]** represent pruning-based methods that aim to shrink the candidate set without full enumeration. **(3) ScaNN [20, 43]** is included as a quantization-based approach, which is widely used as a de facto standard for large-scale approximate search. To apply these methods to an N-way tensor, we follow the standard CP-to-MIPS reduction by treating the first mode as the query and materializing the coupled vectors over the remaining modes as the search database. Regarding hardware settings, all methods are evaluated on the same machine. ScaNN is evaluated on CPU as only a CPU-optimized implementation is available. LEMP and FEXIPRO were reimplemented on GPU using PyTorch to enable a fair hardware comparison. SCOUT and Faiss are evaluated on GPU. This setup allows comparisons among GPU-available methods under the same hardware environment.

Evaluation Protocol. We adopt a standard train/validation/test hold-out strategy. For each dataset, we randomly partition the set of observed interactions into three disjoint splits, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$, and $\mathcal{D}_{\text{test}}$ with the ratio 60%/20%/20%. We train each TF method using $\mathcal{D}_{\text{train}}$ only, and select the checkpoint (epoch) that achieves the best retrieval performance on $\mathcal{D}_{\text{valid}}$. We then report the final results on $\mathcal{D}_{\text{test}}$ using the selected checkpoint. At evaluation time, we retrieve the top-k

[2]https://snap.stanford.edu/biodata/datasets/10013/10013-PPT-Ohmnet.html
[3]https://snap.stanford.edu/biodata/datasets/10017/10017-ChChSe-Decagon.html
[4]https://snap.stanford.edu/data/loc-gowalla.html
[5]https://github.com/claws-lab/DAIN?tab=readme-ov-file
[6]https://www.scidb.cn/en/detail?dataSetId=e6327206214d442790926d709f1960dd
[7]https://webscope.sandbox.yahoo.com/catalog.php?datatype=r

Table 4. Description of real-world tensor datasets. For each dataset, we split all observed interactions into train, valid, and test sets with the ratio 6:2:2.

Scale	Dataset	Tensor Size	# of Observed	# of Unobserved
Medium	Ohmnet[2]	$3,955 \times 4,134 \times 96$	2,364,452	1.57×10^9
	DDS[3]	$629 \times 645 \times 1,317$	4,649,441	5.29×10^8
	Gowalla[4]	$3,761 \times 4,834 \times 168$	333,739	3.05×10^9
	Reddit[5]	$8,894 \times 971 \times 199$	168,111	1.72×10^9
Large	FB15k[6]	$14,951 \times 14,951 \times 1,345$	592,213	3.01×10^{11}
	Yahoo[7]	$82,309 \times 82,308 \times 168$	785,749	1.13×10^{12}

highest-scoring interactions from the candidate space

$$S_{\text{can}} = \left\{ (i_1, \ldots, i_N) \;\middle|\; (i_1, \ldots, i_N) \notin \mathcal{D}_{\text{train}} \right\}, \tag{13}$$

i.e., we *mask* all interactions observed in training during retrieval, and treat only $\mathcal{D}_{\text{test}}$ as positives.

Retrieval Procedure. We evaluate three retrieval procedures: (1) our anytime retrieval with Eq. (9), (2) full enumeration, and (3) MIPS-based baselines via the standard CP-to-MIPS reduction. Except for hyperparameter sensitivity, SCOUT uses the anytime mode with the stopping criterion in Eq. (9). We report end-to-end retrieval latency, averaged over 5 runs. Full enumeration computes scores for all candidates in S_{can} and returns the top-k by sorting. For MIPS-based approaches, we select the largest mode (i.e., first mode) as the query and materialize coupled database vectors over the remaining modes, yielding the smallest mode-coupled materialization and thus favoring MIPS in memory usage.

Metrics. Let $\mathcal{L}_k$ denote the retrieved top-k interactions and $\mathcal{D}_{\text{test}}$ be the held-out test positives. We report Precision@k ($P@k$):

$$\text{P@}k = \left| \mathcal{L}_k \cap \mathcal{D}_{\text{test}} \right| / k. \tag{14}$$

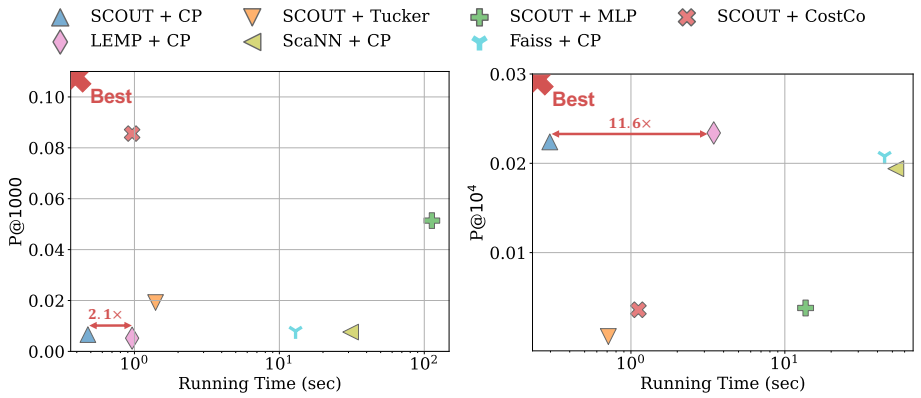
We focus on $P@k$ since the practical objective is to return a small set of high-quality candidates from a massive unobserved candidate space, which can contain up to 10^{12} interactions. Recall becomes less informative in this setting unless k is set very large. We choose k according to the scale of the unobserved candidate space. For medium-size tensors, we use $k = 100$ as a top-ranked retrieval setting. For large-size tensors, we use larger cutoffs, $k = 1{,}000$ or 10^4, because the candidate space is substantially larger, making extremely small cutoffs less stable and less representative for large-scale retrieval evaluation. In such large spaces, even a single additional hit can significantly change $P@k$ when k is very small.

Hyperparameters. For model training, we set the learning rate to 10^{-2} and the number of epochs to 20. We use a batch size of 2048 for CP decomposition and 256 for the other models, with the rank R set to 10. We use Adam optimizer [30] for training models. For the hyperparameters of SCOUT, we set $b = 200$, $\alpha = 0.99$, $\beta = 2$, and $\lambda = 0.01$. Unless otherwise noted, we report results using the anytime retrieval mode with the stopping criterion in Eq. (9).

4.2 Comparison with MIPS-based Approaches

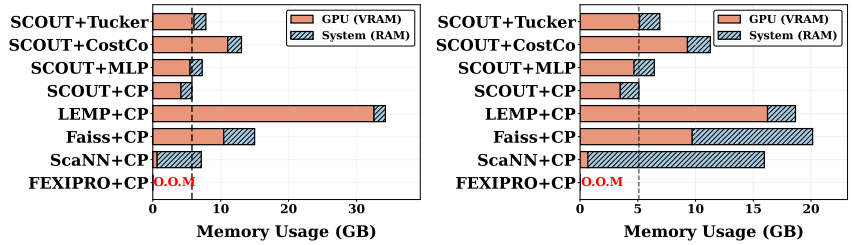
We compare SCOUT integrated with various TF methods against MIPS + CP decomposition for the two large datasets, FB15k and Yahoo. We consider two representative multilinear TF models (CP and Tucker) and two nonlinear TF models (MLP and CostCo).

Running Time-Accuracy Trade-off. Figure 3 shows that SCOUT coupled with a TF scorer (e.g., CP or MLP) consistently lies closest to the desirable accuracy–running time region. First, under the same CP scoring function, SCOUT + CP consistently runs faster than MIPS + CP, and the advantage becomes more pronounced on the larger Yahoo dataset. This suggests that SCOUT offers better practical scalability as the tensor size grows. Second, SCOUT is not restricted to CP-style reductions.



(a) FB15k data (b) Yahoo data

Fig. 3. Comparison of Scout with various TF scorers against MIPS+CP. Scout +CP is faster than MIPS+CP, especially on Yahoo, while non-CP scorers with Scout achieve higher retrieval quality.



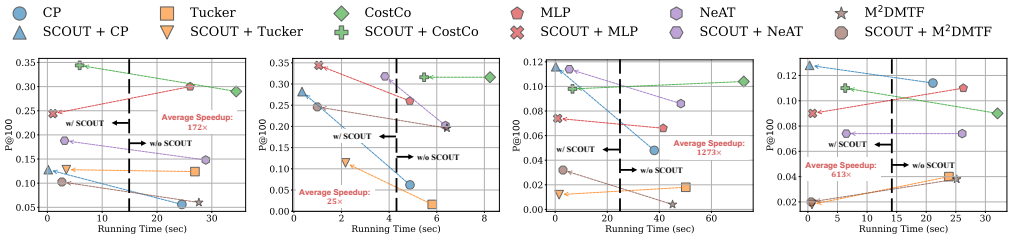
(a) FB15k Data (b) Yahoo Data

Fig. 4. Scout remains memory-efficient, whereas MIPS methods incur high memory cost or O.O.M. failures due to coupled materialization.

On FB15k, Scout + MLP achieves the highest P@1000, and Scout + CostCo and Scout + Tucker also outperform CP-style reductions in accuracy. These results indicate that supporting non-CP TF architectures can lead to higher retrieval quality. Notably, Scout + MLP remains competitive with MIPS + CP in latency, demonstrating that Scout can provide a favorable accuracy–latency trade-off even with more expressive (and typically costlier) scoring functions.

Memory footprint. Figure 4 compares the system/GPU memory footprints of MIPS + CP and Scout coupled with four representative TF methods on FB15k and Yahoo. First, we compare Scout + CP with MIPS + CP. MIPS-style reductions incur substantial memory overhead because they require materializing coupled $(N-1)$-mode product vectors, e.g., $\odot_{n\neq m}^{N}\mathbf{a}_{i_n}^{(n)}$. This coupling quickly becomes impractical at scale and can even trigger out-of-memory (O.O.M.) failures (e.g., FEXIPRO). In contrast, Scout + CP remains lightweight by avoiding explicit cross-mode coupling, operating directly on the learned factor matrices via anytime retrieval with bounded best-first traversal and batched GPU block verification.

We further observe that more expressive TF architectures integrated with Scout naturally incur additional memory due to larger model parameters, but remain substantially more memory-efficient than MIPS+CP on both datasets. Overall, as datasets grow, mode coupling becomes a dominant memory bottleneck for MIPS-based reductions, whereas Scout mitigates this issue by eliminating explicit coupled materialization.

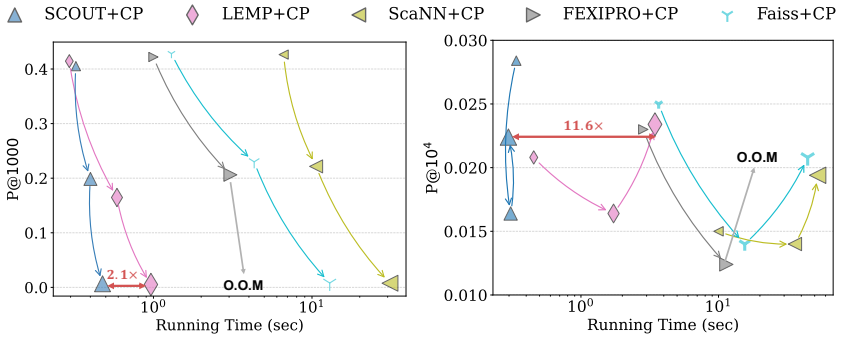


(a) Ohmnet dataset (b) DDS dataset (c) Gowalla dataset (d) Reddit dataset

Fig. 5. Performance comparison between TF methods with and without SCOUT. SCOUT substantially speeds up existing TF methods, with speedups exceeding 1000× on Gowalla, while preserving retrieval quality in most settings.

Table 5. Sample tensor descriptions from the FB15k and Yahoo datasets.

Dataset	Tensor Size	# of observed	# of unobserved
FB15k$_{small}$	$3,716 \times 3,732 \times 1,344$	233,770	1.86×10^{10}
FB15k$_{medium}$	$14,951 \times 3,732 \times 1,344$	397,275	7.50×10^{10}
FB15k	$14,951 \times 14,951 \times 1,345$	592,213	3.01×10^{11}
Yahoo$_{small}$	$19,478 \times 18,772 \times 168$	154,995	6.14×10^{10}
Yahoo$_{medium}$	$82,309 \times 18,772 \times 168$	286,009	2.59×10^{11}
Yahoo	$82,309 \times 82,308 \times 168$	785,749	1.13×10^{12}



(a) FB15k Data (b) Yahoo Data

Fig. 6. Scalability comparison between SCOUT and MIPS-based approaches. Marker sizes indicate candidate-space scale (small→medium→large). SCOUT shows a more favorable time-accuracy trade-off as the candidate space grows.

4.3 Compatibility with TF Methods

We investigate the compatibility of SCOUT with standard TF methods, focusing on the trade-off between accuracy and speed. In Figure 5, SCOUT achieves average speedups of 25× to over 10^3× compared to full enumeration, while preserving retrieval quality in most settings. SCOUT +CP remains the fastest method across all datasets, excelling on sparse datasets like Gowalla and Reddit due to CP's robustness in data-scarce environments. For denser datasets such as Ohmnet and DDS, nonlinear variants like SCOUT +CostCo and SCOUT +MLP achieve the highest accuracy. Run-to-run variability is more pronounced for models that additionally learn $g_n(\cdot)$ (e.g., CostCo and NeAT), whereas SCOUT +CP consistently improves retrieval quality across datasets.

Table 6. Average number of candidate tuples for which SCOUT evaluates the normalized interaction term $\tilde{f}(\cdot)$ after potential-based pre-filtering, compared with the total unobserved interaction space. M, B, and T denote million, billion, and trillion, respectively.

Dataset	Total Unobserved Interactions	CP + SCOUT (# Evaluated)	CostCo + SCOUT (# Evaluated)
FB15k	301B	233.13M	701.01M
Yahoo	1.13T	98.44M	866.56M

Table 7. CPU-only running time (sec.) of SCOUT and MIPS-based baselines.

Dataset	SCOUT (s)	LEMP (s)	ScaNN (s)	Faiss (s)
FB15k	14.90	72.87	31.30	34.19
Yahoo	7.91	310.66	50.09	131.96

4.4 Scalability and Interaction-Evaluation Reduction

We first investigate scalability under CP decomposition by measuring both running time and retrieval quality. Overall, SCOUT achieves high scalability with respect to the size of sparse tensors. We measure *P@k* against the retrieval time for SCOUT +CP and MIPS+CP for FB15k and Yahoo datasets. To investigate scaling behavior with respect to tensor size, we vary the tensor size by *subsampling entities* along each mode (i.e., restricting the tensor to a randomly selected subset of entities per mode), which naturally scales down both observed and unobserved interactions as shown in Table 5. In Figure 6, SCOUT +CP achieves a more favorable time-accuracy trade-off than MIPS+CP, and the running time gap becomes more pronounced in the original datasets than in the sampled datasets, indicating improved scalability as the candidate space grows.

Search-space reduction. Table 6 further explains why SCOUT scales to large candidate spaces. On FB15k and Yahoo, the total unobserved interaction spaces contain approximately 301 billion and 1.13 trillion tuples, respectively. After potential-based pre-filtering, SCOUT evaluates the normalized interaction term for only about 98–867 million candidate tuples when integrated with CP or CostCo. Based on the averaged counts, this represents a reduction by factors of approximately 429× to 11,480× in expensive normalized-interaction evaluations relative to full enumeration. These results demonstrate that SCOUT substantially reduces the number of expensive interaction evaluations at scale, helping explain the favorable time–accuracy trade-off observed in Figure 6.

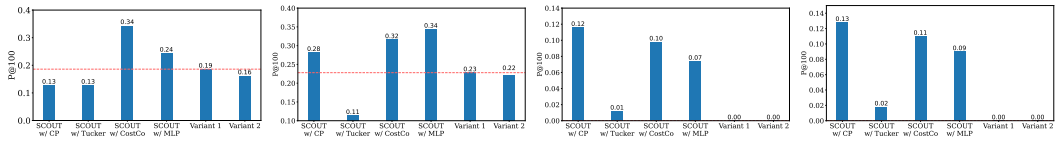
4.5 Ablation Study

Effect of normalized interaction term. We study the role of the term $\tilde{f}(\cdot) = \frac{f(\cdot)}{\prod_n g_n(\cdot)}$ in Eq. (4). Our full scorer computes $s(\mathbf{i}) = (\prod_n g_n(\cdot))\,\tilde{f}(\cdot)$, whereas the ablated variants remove $\tilde{f}(\cdot)$ and score an interaction solely by $\prod_n g_n(\cdot)$. This ablation corresponds to a monotone aggregation over per-mode ranked lists (akin to a generalization of the Fagin framework [15]), while our full method goes beyond rank aggregation by explicitly modeling the residual multiway dependency via $\tilde{f}(\cdot)$.

We compare SCOUT with the ablated variant in terms of P@100, using two choices of $g_n(\cdot)$:

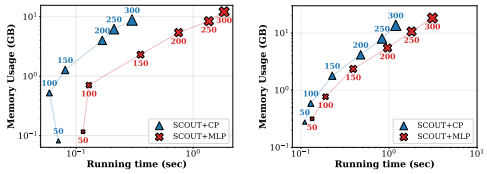
- **Variant 1 (multilinear)**: $g_n(\cdot) = \|\cdot\|_2$ (e.g., for CP/Tucker).
- **Variant 2 (nonlinear)**: $g_n(\cdot) = 1 + \mathrm{softplus}(\mathrm{MLP}(x))$ shared across modes.

As shown in Figure 7, the effectiveness of removing $\tilde{f}(\cdot)$ strongly depends on the observation density. In relatively dense datasets, entity-wise aggregation $\prod_n g_n(\cdot)$ can capture coarse interaction patterns and thus remains competitive for some multilinear scorers (e.g., CP/Tucker). However, even in this regime, SCOUT consistently achieves higher P@100, especially when combined with nonlinear scorers (e.g., MLP/CostCo), showing that modeling the residual interaction $\tilde{f}(\cdot)$ provides



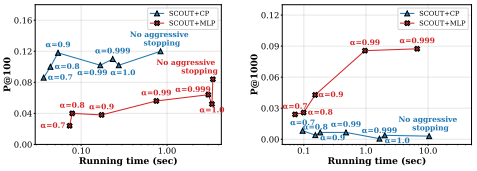
(a) Ohmnet data (b) DDS data (c) Gowalla data (d) Reddit data

Fig. 7. Ablation study on the interaction term $\tilde{f}(\cdot)$. Potential-only variants can be competitive on dense datasets but collapse on sparse datasets, highlighting the importance of $\tilde{f}(\cdot)$.



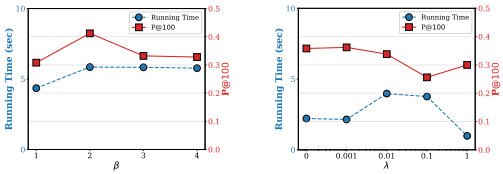
(a) Reddit Data (b) FB15k Data

Fig. 8. Runtime–memory trade-off of SCOUT as the block size b varies (SCOUT +CP/MLP).



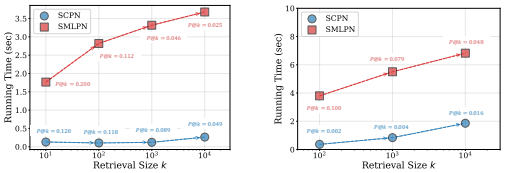
(a) Reddit Data (b) FB15k Data

Fig. 9. Trade-off of SCOUT between running time and $P@k$ under different stopping factors α (SCOUT +CP/MLP).



(a) Effect of β (b) Effect of λ

Fig. 10. Sensitivity analysis of β and λ.



(a) Reddit Data (b) FB15k Data

Fig. 11. Running time and $P@k$ of SCOUT with respect to the retrieval size k.

additional signal beyond monotone rank aggregation. This gap becomes more pronounced in sparse datasets, where the two variants collapse (P@100=0), whereas SCOUT continues to retrieve nontrivial ground-truth interactions by preserving higher-order dependencies under combinatorial noise.

Effect of GPU Acceleration. We further evaluate SCOUT and MIPS-based baselines under CPU execution. For this experiment, we use a fixed block size $b = 100$ for the FB15k and Yahoo datasets, since larger blocks can increase block-wise verification overhead, especially under CPU execution with limited parallelism, consistent with Figures 8(a) and 8(b). As shown in Table 7, SCOUT outperforms all MIPS-based baselines on both datasets. Together with the interaction-evaluation reduction analysis in Table 6, these results indicate that SCOUT's efficiency gain is not solely due to GPU acceleration, but also stems from its bound-guided pruning and traversal framework, which substantially reduces the number of candidates requiring normalized-interaction evaluation.

4.6 Hyperparameter Sensitivity

We analyze the sensitivity of SCOUT to two key hyperparameters: the block size b used for GPU block-wise verification and the decay factor α in the aggressive stopping rule (Eq. (9)). We report results using a representative multilinear scorer (CP) and a representative nonlinear scorer (MLP).

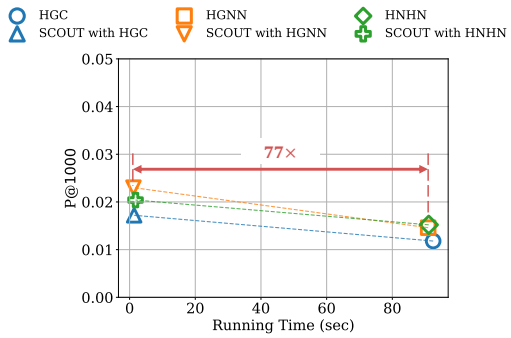


Fig. 12. Performance comparison of HNN baselines with and without Scout on the Dawn dataset. Scout accelerates top-k retrieval by up to 77× while preserving high P@1000.

Block size b. We vary the block size b from 50 to 300 and report the running time and peak memory footprint in Figures 8(a) and 8(b). As b increases, each GPU verification batch covers a larger Cartesian-product block, increasing batched scoring cost and peak memory usage. Overall, b serves as a practical knob for scaling verification throughput and trading off exploration budget against running time and memory constraints; we set $b = 200$ as a balanced default.

Aggressive stopping factor α. We vary $\alpha \in \{0.7, 0.8, 0.9, 0.99, 0.999, 1.0\}$ and report the trade-off between running time and $P@k$. Figures 9(a) and 9(b) show that the impact of α is more pronounced for nonlinear scorers; Scout +MLP at $\alpha = 1.0$ on FB15k exceeded our runtime budget and is therefore omitted. For Scout +MLP, $P@k$ generally improves as α approaches 1.0, since conservative stopping explores deeper into the lattice. In contrast, Scout +CP exhibits a weaker dependence on α, as the top-k list is often completed early from high-$\hat{s}$ candidates. Overall, $\alpha = 0.99$ strikes a practical balance between speed and accuracy and is used as the default.

Bound-tightness scaling factor β and regularization strength λ. We vary $\beta \in \{1, 2, 3, 4\}$ and $\lambda \in \{0, 0.001, 0.01, 0.1, 1\}$, and report running time and retrieval quality. As shown in Figure 10, $\beta = 2$ achieves the highest retrieval quality among the tested values, while running time remains within a comparable range. For λ, larger values tend to reduce retrieval quality, whereas moderate values maintain competitive $P@100$. Accordingly, we use $\beta = 2$ and $\lambda = 0.01$ as default settings in the other experiments.

Retrieval size k. We report running time and $P@k$ by varying the retrieval size k. In Figure 11, running time generally increases with k, since larger output sizes require maintaining a larger top-k set and verifying more candidates. In contrast, $P@k$ generally decreases as k increases, as larger cutoffs include lower-ranked candidates that often have lower positive density. The different trend observed for Scout + CP on FB15k reflects the cutoff-dependent nature of $P@k$, as the distribution of positives can vary across rank ranges. Overall, the results show predictable runtime scaling with respect to k, while the corresponding $P@k$ values follow the expected behavior of cutoff-based retrieval evaluation.

4.7 Extension to Hyperedge Prediction

Thanks to the generalizability of our scoring framework, Scout can be applied to higher-order interaction retrieval beyond TF. We demonstrate this on hyperedge prediction.

Data. We use the Dawn dataset[8] [4], a drug-combination hypergraph with 2,109 nodes and 87,104 observed hyperedges. We split observed interactions into train/valid/test with a ratio of 6:2:2.

[8]Available at https://www.cs.cornell.edu/~arb/data/cat-edge-DAWN/

Table 8. Top-5 predicted DDS interactions by MLP with SCOUT. Entity ranks based on $g_n(\cdot)$ are shown in parentheses; blue and red cells denote test-set positives and high-scoring unobserved predictions, respectively.

Rank	Drug 1	Drug 2	Side Effect
1	Bupropion (302)	Alendronate (422)	Fatigue (122)
2	Bupropion (302)	Alendronate (422)	Chest pain (54)
3	Estradiol (257)	Alendronate (422)	Chest pain (54)
4	Estradiol (257)	Alendronate (422)	Edema extremities (327)
5	Estradiol (257)	Alendronate (422)	Arterial pressure NOS decreased (428)

Task. We retrieve top-k *unobserved triples* among $2{,}109^3 = 9.39 \times 10^9$ candidates. Hypergraph models are trained on hyperedges of diverse sizes using Eq. (3), and evaluated on test triples by P@1000.

Baselines. We consider three HNN encoders: HyperGraphConv (HGC) [5], HGNN [18], and HNHN [12]. For all methods, we use max pooling as the aggregator and a 2-layer MLP as the predictor.

Applying SCOUT. We apply SCOUT to each hyperedge scoring pipeline (HNN encoder + aggregator + predictor) by substituting its original scoring function $f_{\text{hg}}(\cdot)$ into Eq. (4) where the subscript hg denotes the original hyperedge scorer built on an HNN backbone. We compute per-node potentials via a shared MLP $g_n(\cdot)$ on node embeddings, and capture the remaining higher-order dependency through $\tilde{f}_{\text{hg}}(\cdot)$. For nonlinear components, we add the regularization in Eq. (8) to encourage the bound condition required for admissibility.

Result. Figure 12 shows that SCOUT accelerates top-k triple retrieval by up to 77× while maintaining comparable P@1000. HGNN+SCOUT provides the best speed–accuracy trade-off, highlighting the versatility of SCOUT for higher-order interaction tasks.

4.8 Case Study

Table 8 presents a qualitative case study of the top-5 predicted interactions for MLP with SCOUT on the DDS dataset. SCOUT identifies meaningful interactions even when the constituent entities do not have the highest individual potentials. For instance, Bupropion (302nd) and Alendronate (422nd) receive high interaction scores, with Fatigue appearing as a test-set positive and Chest pain as a high-scoring unobserved prediction.

5 Related Work

We review existing literature in three key areas: ranked retrieval and top-k query processing, maximum inner product search, and tensor factorization.

Ranked Retrieval and Top-k Query Processing. Ranked retrieval and top-k query processing have long been studied in the database community. Classical work on rank aggregation and middleware systems [15, 16] studies how to combine multiple ranked inputs and identify top-ranked answers without fully accessing all objects. Top-k join processing [24] further addresses ranked query processing over join results, where the output space may be large and full materialization is undesirable. These works provide the principles of ranked access, threshold-based stopping, and pruning. However, they typically assume explicitly available ranked lists, decomposable score components, or relational join inputs. In contrast, our setting ranks tuples over an implicit Cartesian-product tensor space using learned high-order TF scores, where no ranked-access structure is directly available. SCOUT addresses this gap by deriving a scorer-preserving, coupling-free bound that induces ranked access and admissible pruning over the implicit candidate space.

Maximum Inner Product Search (MIPS). MIPS can be viewed as a widely used retrieval primitive for pairwise learned scores. Prior work has proposed a wide range of MIPS solutions [1,

13, 20, 28, 32, 43, 47, 52] using pruning, indexing, and quantization techniques. Importantly, the main obstacle in applying MIPS to TF is structural rather than method-specific. Thus, the limitation is not merely whether a score can be rewritten as an inner product, but whether the rewriting yields a scalable ranked-access path without materializing composite items. MIPS is tailored to pairwise inner-product retrieval, whereas higher-order TF scoring involves multi-entity interactions and typically requires explicit mode coupling (e.g., Khatri–Rao materialization) to form an access path, which becomes memory-prohibitive as in Section 4. A potential alternative is to apply MIPS slice-by-slice and aggregate the per-slice results; however, this can still incur worst-case query work scaling as $O(I^{N-1})$ (e.g., $O(I)$ queries over $O(I^{N-2})$ slices for balanced modes), making it computationally impractical at scale. In contrast, Scout enables efficient top-k retrieval over *higher-order* interactions under *TF-based* scoring functions, where each score depends on three or more entities.

Tensor Factorization and Tensor-based Retrieval. To analyze large-scale tensors, extensive research has focused on efficient and scalable TF for both dense tensors [26, 27, 36, 38, 42, 44, 49, 54] and sparse tensors [2, 19, 40, 41, 51, 53]. These advances primarily improve the efficiency of *learning* TF representations, whereas our focus is on *retrieving* top-k unobserved interactions efficiently at inference time. In sparse regimes, predictive accuracy is equally critical, prompting the development of more accurate TF methods [3, 17, 25, 33]. However, despite their strong predictive power, these models typically lack dedicated retrieval mechanisms and often require exhaustive scoring over the combinatorial candidate space. In contrast, Scout enables scalable top-k retrieval while seamlessly integrating with accurate TF scorers and maintaining comparable predictive performance. These tensor-based retrieval tasks arise across domains including context-aware recommendation [10, 21, 48], knowledge graph completion [35, 45], computational biology [34, 46], and social network analysis [9, 23], where the common bottleneck is identifying top-k interactions from a combinatorial candidate space without exhaustive enumeration.

6 Conclusion

We propose Scout, a scalable and compatible framework for top-k ranked retrieval over massive implicit tensor spaces. Scout leverages a scorer-preserving reparameterization to derive a coupling-free upper bound, enabling admissible pruning and GPU-efficient block-wise verification, while supporting budgeted anytime retrieval and certified exact stopping. Empirically, Scout achieves orders-of-magnitude speedups over full enumeration and outperforms MIPS-based baselines in most settings while maintaining comparable retrieval quality. By bridging accurate tensor modeling and scalable inference-time retrieval, Scout makes higher-order interaction retrieval practical. Future work includes extending Scout to streaming settings for real-time retrieval over dynamic tensor streams.

Acknowledgments

This work is supported by NSF (2316233). The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2026-25490900). It was also supported by the DGIST Start-up Fund Program of the Ministry of Science and ICT (2026010244).

References

[1] Firas Abuzaid, Geet Sethi, Peter Bailis, and Matei Zaharia. 2019. To index or not to index: Optimizing exact maximum inner product search. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1250–1261.

[2] Dawon Ahn, Seyun Kim, and U Kang. 2021. Accurate online tensor factorization for temporal tensor streams with missing values. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2822–2826.

[3] Dawon Ahn, Uday Singh Saini, Evangelos E Papalexakis, and Ali Payani. 2024. Neural additive tensor decomposition for sparse tensors. In *CIKM*. 14–23.

[4] Ilya Amburg, Nate Veldt, and Austin Benson. 2020. Clustering in graphs and hypergraphs with categorical edge labels. In *Proceedings of the web conference 2020*. 706–717.

[5] Song Bai, Feihu Zhang, and Philip HS Torr. 2021. Hypergraph convolution and hypergraph attention. *Pattern Recognition* 110 (2021), 107637.

[6] Ivana Balažević, Carl Allen, and Timothy M Hospedales. 2019. Tucker: Tensor factorization for knowledge graph completion. *arXiv preprint arXiv:1901.09590* (2019).

[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013).

[8] J Douglas Carroll and Jih-Jie Chang. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika* 35, 3 (1970), 283–319.

[9] Martina Contisciani, Federico Battiston, and Caterina De Bacco. 2022. Inference of hyperedges and overlapping communities in hypergraphs. *Nature communications* 13, 1 (2022), 7229.

[10] Yizhou Dang, Enneng Yang, Guibing Guo, Linying Jiang, Xingwei Wang, Xiaoxiao Xu, Qinghui Sun, and Hong Liu. 2023. Uniform sequence better: Time interval aware data augmentation for sequential recommendation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 4225–4232.

[11] Miguel Ramos de Araujo, Pedro Manuel Pinto Ribeiro, and Christos Faloutsos. 2017. Tensorcast: Forecasting with context using coupled tensors. In *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 71–80.

[12] Yihe Dong, Will Sawin, and Yoshua Bengio. 2020. Hnhn: Hypergraph networks with hyperedge neurons. *arXiv preprint arXiv:2006.12278* (2020).

[13] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[14] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. 2000. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems* 13 (2000).

[15] Ronald Fagin. 1996. Combining fuzzy information from multiple systems. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 216–226.

[16] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *PODS*. 102–113.

[17] Jicong Fan. 2021. Multi-mode deep matrix and tensor factorization. In *international conference on learning representations*.

[18] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. 2019. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 3558–3565.

[19] Ekta Gujral, Ravdeep Pasricha, and Evangelos E Papalexakis. 2018. Sambaten: Sampling-based batch incremental tensor decomposition. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 387–395.

[20] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *International Conference on Machine Learning*. https://arxiv.org/abs/1908.10396

[21] Casper Hansen, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi, and Mounia Lalmas. 2020. Contextual and sequential user embeddings for large-scale music recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 53–62.

[22] Richard A Harshman et al. 1970. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA working papers in phonetics* 16, 1 (1970), 84.

[23] Iacopo Iacopini, Márton Karsai, and Alain Barrat. 2024. The temporal dynamics of group interactions in higher-order social networks. *Nature Communications* 15, 1 (2024), 7391.

[24] Ihab F Ilyas, Walid G Aref, and Ahmed K Elmagarmid. 2004. Supporting top-k join queries in relational databases. *The VLDB journal* 13, 3 (2004), 207–221.

[25] Jun-Gi Jang, Jingrui He, Andrew J. Margenot, and Hanghang Tong. 2026. DuET: Dual-View Tensor-to-Topology Spectral Adapter for Enhancing Sparse Tensor Factorization. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*.

[26] Jun-Gi Jang and U Kang. 2021. Fast and memory-efficient tucker decomposition for answering diverse time range queries. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 725–735.

[27] Jun-Gi Jang and U Kang. 2023. Static and streaming tucker decomposition for dense tensors. *ACM Transactions on Knowledge Discovery from Data* 17, 5 (2023), 1–34.

[28] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[29] Chinubhai G Khatri and C Radhakrishna Rao. 1968. Solutions to some functional equations and their applications to characterization of probability distributions. *Sankhyā: the Indian journal of statistics, series A* (1968), 167–180.

[30] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[31] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[32] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 835–850.

[33] Hanpeng Liu, Yaguang Li, Michael Tsang, and Yan Liu. 2019. Costco: A neural tensor completion model for sparse tensors. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 324–334.

[34] Xuan Liu, Congzhi Song, Shichao Liu, Menglu Li, Xionghui Zhou, and Wen Zhang. 2022. Multi-way relation-enhanced hypergraph representation learning for anti-cancer drug synergy prediction. *Bioinformatics* 38, 20 (2022), 4782–4789.

[35] Yu Liu, Quanming Yao, and Yong Li. 2021. Role-aware modeling for n-ary relational knowledge bases. In *Proceedings of the Web Conference 2021*. 2660–2671.

[36] Osman Asif Malik and Stephen Becker. 2018. Low-rank tucker decomposition of large tensors using tensorsketch. *Advances in neural information processing systems* 31 (2018).

[37] Sejoon Oh, Sungchul Kim, Ryan A Rossi, and Srijan Kumar. 2021. Influence-guided data augmentation for neural tensor completion. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1386–1395.

[38] Ruizhong Qiu, Jun-Gi Jang, Xiao Lin, Lihui Liu, and Hanghang Tong. 2024. TUCKET: A Tensor Time Series Data Structure for Efficient and Accurate Factor Analysis over Time Ranges. *Proc. VLDB Endow.* 17, 13 (Sept. 2024), 4746–4759. doi:10.14778/3704965.3704980

[39] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[40] Shaden Smith, Kejun Huang, Nicholas D Sidiropoulos, and George Karypis. 2018. Streaming tensor factorization for infinite data sources. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 81–89.

[41] Qingquan Song, Xiao Huang, Hancheng Ge, James Caverlee, and Xia Hu. 2017. Multi-aspect streaming tensor completion. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 435–443.

[42] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 374–383.

[43] Philip Sun, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar. 2023. SOAR: Improved Indexing for Approximate Nearest Neighbor Search. In *Neural Information Processing Systems*. https://arxiv.org/abs/2404.00774

[44] Yiming Sun, Yang Guo, Charlene Luo, Joel Tropp, and Madeleine Udell. 2020. Low-rank tucker approximation of a tensor from streaming data. *SIAM Journal on Mathematics of Data Science* 2, 4 (2020), 1123–1150.

[45] Ava Swevels, Remco Dijkman, and Dirk Fahland. 2023. Inferring missing entity identifiers from context using event knowledge graphs. In *International Conference on Business Process Management*. Springer, 180–197.

[46] Chunyan Tang, Cheng Zhong, Mian Wang, and Fengfeng Zhou. 2022. FMGNN: A method to predict compound-protein interaction with pharmacophore features and physicochemical properties of amino acids. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 20, 2 (2022), 1030–1040.

[47] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. 2015. Lemp: Fast retrieval of large entries in a matrix product. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 107–122.

[48] Viet Anh Tran, Guillaume Salha-Galvan, Bruno Sguerra, and Romain Hennequin. 2023. Attention mixtures for time-aware sequential recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1821–1826.

[49] Charalampos E Tsourakakis. 2010. Mach: Fast randomized tensor decompositions. In *SDM*. SIAM, 689–700.

[50] Ledyard R Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31, 3 (1966), 279–311.

[51] Fan Yang, Fanhua Shang, Yuzhen Huang, James Cheng, Jinfeng Li, Yunjian Zhao, and Ruihao Zhao. 2017. LFTF: A framework for efficient tensor analytics at scale. *Proceedings of the VLDB Endowment* 10, 7 (2017), 745–756.

[52] Hsiang-Fu Yu, Cho-Jui Hsieh, Qi Lei, and Inderjit S Dhillon. 2017. A greedy approach for budgeted maximum inner product search. *Advances in neural information processing systems* 30 (2017).

[53] Shuo Zhou, Sarah Erfani, and James Bailey. 2018. Online cp decomposition for sparse tensors. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1458–1463.

[54] Shuo Zhou, Nguyen Xuan Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. 2016. Accelerating online cp decompositions for higher order tensors. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1375–1384.
[55] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34, 13 (2018), i457–i466.
[56] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (2017), i190–i198.

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026