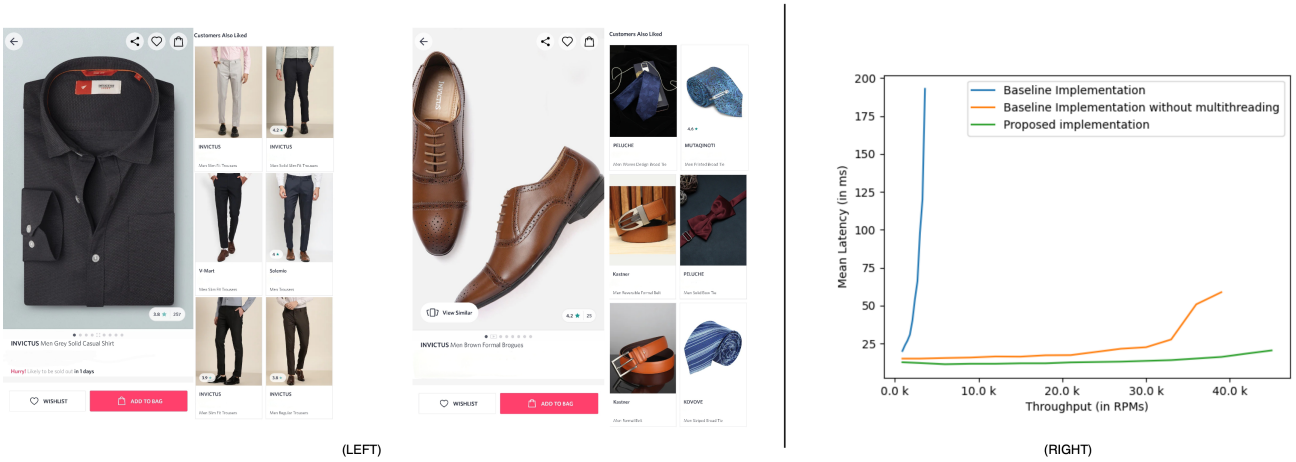
Rethinking 'Complement' Recommendations at Scale with SIMD

Shrey Pandey
shreypandey1509@gmail.com
Myntra Designs Pvt. Ltd.
Data Science
Bangalore, India

Saikat Kumar Das
saikat.kumar@myntra.com
Myntra Designs Pvt. Ltd.
Data Science
Bangalore, India

Hrishikesh V. Ganu*
hrishikeshvganu@gmail.com
Independent
Bangalore, India

Satyajeet Singh
satyajeet.singh@myntra.com
Myntra Designs Pvt. Ltd.
Data Science
Bangalore, India



(LEFT) (RIGHT)

Figure 1: 'Complement' Recommendations service in Myntra. Left: Primary product Added-to-Cart by user and corresponding 'complement' recommendations to increase discovery and item count in cart. Right: Comparative graph depicting the improvement in service scalability due to proposed vectorized reformulation of recommendation components.

ABSTRACT

Maximizing cart value by increasing the number of items in electronic carts is one of the key strategies adopted by e-commerce platforms for optimal conversion of positive user intent during an online shopping session. Recommender systems play a key-role in suggesting personalized candidate items that can be added to cart by the user. However, it is important to serve a diverse set of personalized recommendations that 'complement' user's cart content to practically increase item count in cart and also contribute towards product discovery. Borrowed from Quantum Physics, Determinantal Point Processes (DPP) are used widely in recommender systems to diversify personalized product recommendations for improved user engagement. However, vertically scaling DPP for recommendation sets, personalized with vector similarity metric like cosine similarity, to serve large scale real-time concurrent user requests is non-trivial. We propose a vectorized reformulation of cosine similarity and conditional DPP implementation to best utilize the highly improved vector computation capabilities (SIMD) of modern processors. Experimental evidence on real-world traffic shows that the proposed method can handle upto 15x more concurrent traffic while improving latency. The proposed method also uses portable SIMD constructs from Python libraries which can be easily adopted in most available SIMD supported CPUs with minimal code changes.

CCS CONCEPTS

• **Computer systems organization → Single instruction, multiple data**; • **Applied computing → Online shopping**; • **Computing methodologies → Linear algebra algorithms**.

*work was done while affiliated to Myntra Designs Pvt. Ltd.

ICPE '24, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0444-4/24/05…$15.00
https://doi.org/10.1145/3629526.3645041

KEYWORDS

Vectorization, SIMD, Diversification, Recommender Systems, Recommendations, Determinantal Point Processes

ACM Reference Format:

Shrey Pandey, Saikat Kumar Das, Hrishikesh V. Ganu, and Satyajeet Singh. 2024. Rethinking 'Complement' Recommendations at Scale with SIMD. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3629526.3645041

1 INTRODUCTION

'Add-To-Cart' (ATC) is a key event in a user session on e-commerce platforms. ATC indicates positive intent from user to complete a purchase from current session. However, this also presents a unique opportunity for the platform to improve cart value by increasing the number of items in cart and optimize conversion, generally referred to as *cart completion* or *cart filling*. Cart completion has been a subject of extensive research for decades, evolving continually to cater to changing customer behaviour and research community's understanding of the same. Early works on cart completion works on the principle of finding likely items to be added to cart based on previous co-click events [4] and browsing history from a cohort of similar users [48]. Though effective, these approaches do not differentiate browsing patterns and user motivation based on occurrence of key-events like ATC or 'Wish-listing'. Close *et al.* [15] conducted a detailed study for understanding user behaviour patterns related to the usage of electronic carts. Building on the broader patterns of user motivation of adding items in cart beyond immediate purchase as outlined by the researchers, we can identify scope of optimizing conversion by increasing the number of items in cart. Mcauley *et al.* [38] explored the concept of 'complement' and 'substitute' in their work, where they claim complements are items that can be bought alongside items already in cart and substitutes are used for replacing current cart items. For users looking to take advantage of price promotion or browsing for entertainment [15], recommending similar items or 'substitutes' to those already in cart can be detrimental to the eventual cart value. In the first case, user looking for price promotions may chose to replace a cart item with a new item of lower or same value, which may not lead to optimizing conversion. In the second case, users looking for entertainment may find it monotonous to keep browsing similar items which they have already explored. Apart from this, for users with immediate purchase intent, substitute recommendations from their cart can induce doubts about their selection, which may lead to session abandonment. Similar item recommendations can be useful to users looking for research or information search, but studies [15] have shown that purchase probability for such sessions are much lower than the cases mentioned above.

Thus, to optimize conversion through increasing items in cart, diversification of personalized recommendations to suggest 'complements' based on cart-items appears to be the better strategy as it complements user selections rather than competing. However naive diversification of personalized recommendations can lead to degrading quality of suggestion. Diversification should be closely aligned with user preferences and user journey on platform along with cart-items. Determinantal Point Process [6, 21, 30] is a popular method for diversifying personalized recommendations, personalized with vector similarity metric like cosine similarity. However, one practical consideration in using automated recommendation system on e-commerce platform [1, 11, 16, 29, 43, 63] is the scale and concurrency it has to handle in real-time shopping scenarios. Online recommendation systems can receive millions of requests per minute (RPM) on high traffic event days. Moreover, most e-commerce platform will host millions of items in their catalogue to serve millions of customers. Thus, it is imperative to build a system that can handle large scale traffic of few million RPM while also being flexible enough to be deployable without costly hardware requirements. Cosine similarity and DPP [59], although being an excellent algorithms to consider diversification of personalized recommendations, but is not scalable vertically, especially when deployed on CPU-only cloud servers. Tensorized versions of DPP has been proposed [58] to harness GPU powers in-order to provide higher order scalability but as per our knowledge, re-thinking DPP formulation and Cosine similarity implementation to best utilize the untapped compute capacity of SIMD (Single Instruction Multiple Data) enabled modern CPUs has not been explored previously in literature. To this end, we propose a reformulation of the DPP algorithm and vectorized implementation of Cosine Similarity, tailored for SIMD acceleration on CPU that is highly scalable yet portable across various available server CPUs (Sec. 3). Proposed, reformulated DPP reduces number of computation involving N candidate recommendations and k items from $O(x) + O(N^2)$ to $O(x) + O(N \times k))$, where $N >> k$ and $N^2 > x > N^3$; x being the number of computations for matrix multiplication.

Our proposed diversifying recommendation system (Fig. 1 (Left)) enabled us to reduce hardware requirement by nearly 12 times in production environment (Fig. 1 (Right)) while also improving the top 99th-percentile (P99) latency under the maximum throughput by 13 times (realtime traffic observations discussed in Sec. 5). Our key contributions in this work are

- Evaluation of parallel processing strategies for CPU for real-time concurrent services on e-commerce platform.
- Vectorized implementation of cosine similarity used for user personalization of recommendations that can leverage the superior SIMD compute capability of modern CPUs.
- Reformulation and vectorization of DPP, commonly used for diversification of recommendations, to best utilize the gain in efficiency with SIMD.
- A portable implementation of vectorized SIMD acceleration using Python libraries that can be migrated amongst various CPUs and architectures with minimal to no code changes.
- Provide long-term scalability solution to recommendation pipelines through SIMD vector optimization instead of parallel processing.

2 RELATED WORK

Recommendation system has been an active area of research for decades with ever evolving requirements to filter out information overload and generate optimal set of recommendations for consumers. Early works in recommendation systems started with finding correlations of amongst items in large scale databases from browsing sequences of users [4]. The widely studied *'GroupLens'*

project [27, 28, 46] introduced collaborative information filtering in recommendation systems inspired from the real world news groups. Sarwar *et al.* [48] proposed to use item based collaborative filtering in-place of the conventional user centric filtering, while Shahabi *et al.* [49] introduced large scale recommendation system by augmenting collaborative filtering with content-based querying. Amazon also published their report [34] on personalizing shopping with item-to-item collaborative filtering using co-browsing data in 2003. Hijikata *et al.* [23] proposed to use discovery driven collaborative filtering that for each user recommends undiscovered items with higher probability while personalizing the ranking based on user's discovery till that point of time.

On the other hand, the importance of diversity and maximizing representation of the entire collection in the recommended subset gained traction almost at the same time as well. Ko *et al.* [26] proposed to select the most informative subset of recommendations by maximizing entropy of the set. Adomavicius *et al.* [1] use multicriteria ranking to provide better and diverse recommendations. Carbinell *et al.* [7] proposed re-ranking of document search results to maximize information retrieved. Diversification of search results and its impact on information filtering was studied widely in the early 2010s [3, 9, 14, 17, 18, 22, 45, 47]. The impact of diversification on auto recommendations was also studied widely at the same time [55, 61, 62, 64]. Lathia *et al.* [33] studied relevance of temporal diversity in recommendations. Chapelle *et al.* [10] explored combining diversification objective with intent based metric ERR-IA to evaluate relevance and diversity together. Genre and category based diversification was also explored for generation better recommendations [43, 54]. Diversification also played a important part in improving relevance of video recommendations. Covington *et al.* [16] use a deep-learning based recommendation model for generating diverse recommendations. While, Mark *et al.* [59] used DPP for personalized diversification of video search results in Youtube. Another widely adopted diversification strategy are the submodular functions [42] which operates on the principle of maximizing information from a subset. Researcher's in [41, 51] use submodular functions to diversify recommendations in e-commerce.

Lot of research has been also done to better understand and interpret user-behaviour and inferred signals in e-commerce shopping. McNee *et al.* [39] attempted to establish a user-centric metric with informal arguments to better evaluate recommendation quality. Close and Kukar-Kinney [15] attempted to breakdown and analyse various user motivation behind the use of electronic carts. In their work they identify motivations such as entertainment, research also to play a major role apart from usual factors like price promotions and immediate purchase. The data evidence backed hypothesis from this work helped to identify opportunities to increase cart items and in effect the gross revenue. Hohnhold *et al.* [24] proposed to focus on long term user retention rather than short-term gains. Mcauley *et al.* [38] explored item relations based on substitutes and complements, here substitutes refer to products which can replace the item already in cart and complements are products which can be bought along with the item already planned for purchase. Zheng *et al.* [63] also explore substitute and complement relations between items to enrich recommendations list.

Determinantal Point Processes originated in Quantum Physics [6, 25, 35] and are a natural choice for modelling informative subset selection problem like document summarization, diversifying search and recommendations. Kulesza *et al.* [31] parameterized conditional DPP and attempted to learn DPPs from the resulting convex and tractable learning formulation. Kulesza *et al.* also proposed to improve the efficiency of DPP computation with the fixed size k-DPP formulation [30] and also advocated for using DPPs over MRF in ML tasks mentioned above for better tractability of the former. Considerable effort has been invested collectively to improve the efficiency of the DPP computation [5, 19, 20, 30, 36, 37]. Chen *et al.* [11] used DPP to improve recommendations with diversity. Mark *et al.* [59] also used DPP to generate diverse video recommendations for Youtube. However, reformulating DPP to reduce complexity of the algorithm without compromising on precision or exact solution has not been explored in-depth in literature. In our work, we introduce a reformulated personalization and DPP based diversification process, highly scalable for online recommendations by using vectorized computations efficiently. Proposed recommendation pipeline is able to serve $15x$ more traffic than the existing solutions without comprising on the quality of the results.

3 METHOD

For a given query product p_q and user u_q, Yan *et al.* [60] and Agarwal *et al.* [2] described the process of calculating relevant cross-category or 'complement' recommendations. A product-embedding vector e represents a product p present in the product catalogue $\mathbb{C}$ of millions of products. Similarly, an user-embedding vector v represents a user u of the e-commerce platform in vector space. Every embedding vector is part of a k-dimensional vector space $\mathcal{V}$ stored in the form of a one-dimensional array in computer memory (Eq. 1).

$$e, v \in \mathcal{V} \subset \mathbb{R}^k \tag{1}$$

For a query product p_q with product embedding e_q , a set of N products containing the relevant cross-category recommended products is generated using the product-embedding vectors e in $\mathcal{V}$. This set which acts as a candidate set of recommendations, is called the recall set C for the query product p_q. The sequence of products in the recall set is not personalized according to the user preference as it does not consider user-embedding vectors. It is essential to suggest products in a personalized sequence tailored to user preference to increase the relevance of the recommendations. Each product in the recall set, represented by $p_i \; \forall i \in [1..N]$, is assigned a user relevance score s_i, using user-embedding vector v_q of user u_q , and the product embedding e_i to personalize the product recommendations. The *personalized* recall set is then re-ranked by Determinantal Point Process (DPP) [35], which uses the relevance scores and product-embeddings for diversifying the recommendations. The re-ranked set of products are the recommended 'complements' of the product p_q, personalized to user u_q.

Fig. 2 shows the workflow for the cross-category-related product recommendations. The recommendation workflow is broadly divided into three steps:- (1) Recall set generation, (2) Personalised scoring, and (3) Diversified re-ranking.

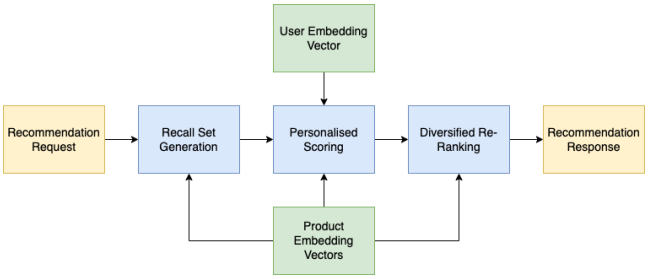


Figure 2: 'Complement' recommendation service workflow

3.1 Recall Set Generation

This step acts as a filter on a catalogue of millions of products. It filters out the less relevant products to generate the the recall set C, a set of the most relevant N products to the query product p_q. This step calculates the approximate nearest neighbours (ANN) of the query product p_q in the vector space $\mathcal{V}$ using the product-embedding vectors e_q of the query product and remaining products $e_l \forall\, l \in [1..\mathcal{N}_{\mathbb{C}}]$ in the catalogue $\mathbb{C}$.

$$sim(g, h) = \sum_{i=1}^{N} g_i \times h_i \tag{2}$$

The cosine similarity distance Eq. 2 between the embedding vectors is the metric for the relevance between two products. Distance between the embedding vector e_q and embedding vectors $e_i \forall i \in [1...N]$ of the products p_i in the recall set C gives the style relevance score r_i for the recommendation as shown in Eq. 3.

$$\forall\{i \in \mathbb{N}, i \leq N\},\ r_i = \sum_{j=1}^{k} e_{qj} \times e_{ij} \tag{3}$$

User embedding vector v_q depends on the real time in-session data and user browsing behaviour. Thus, it is expected to evolve with each action in a browsing session. Product attributes remain mostly unchanged over their lifetime, barring few exceptions. This makes user embeddings more volatile than product embeddings. This property allows us to calculate the recall set of products, offline, once a day without risking the consistency of the pipeline. Recall set is stored in appropriate data-stores like Aerospike [50] and Redis [8]. During online recommendations, recall set C for the query product p_q is fetched from the corresponding data-store and passed into next step of recommendation generation pipeline.

3.2 Personalised Scoring

The Recall set C generated till now is non-personalized. For personalised recommendations, the candidate recommendations in C should be re-scored, taking user affinity into account. The user-personalized score of a product in the recall set is the distance of the user embedding vector and product embedding vector in the k-dimensional space. A combination of the user-personalised score with the style relevance score of each product acts as input for re-ranking the recommendations.

Consider, v is the user embedding vector, V is the row-major matrix of size $N \times k$ with each row representing product embedding e_i for each product in recall set C *i.e.* candidate recommendation p_i . S is the vector containing user personalised score for every candidate recommendation. User-personalized score s_i of product p_i in the recall set C can be computed by Eq. 4.

$$\forall\{i \in \mathbb{N}, i \leq N\},\ s_i = \sum_{j=1}^{k} v_j \times V_{ij} \tag{4}$$

User and Product embedding are normalized, i.e $\|v\| = 1$ and $\|e_i\| = 1, \forall\, i \in [1..N]$. The baseline algorithmic implementation is described in Alg.1

Algorithm 1 Baseline implementation of cosine similarity

Require: (u, V)
Ensure: *user personalisation scores* $= S$
  $S \leftarrow \{0.0, 0.0, 0.0....0.0\}_N$
  $i \leftarrow 1$
  while $i \leq N$ **do**
    $score \leftarrow 0.0$
    $j \leftarrow 1$
    while $j \leq k$ **do**
      $X \leftarrow u[j] \times V[i][j]$
      $score \leftarrow score + X$
      $j \leftarrow j + 1$
    end while
    $S[i] \leftarrow score$
    $i \leftarrow i + 1$
  end while

Since each iteration of outer loop in the baseline implementation as mentioned in Alg. 1 calculates score s_i for each product p_i, every iteration of outer loop is independent of each other and can be computed in parallel. This limits the optimal efficiency of the baseline implementation as it works on one vector element at a time, increasing the latency of the cosine similarity calculation. For optimising the latency of the user-personalised score calculation, parallelism should be incorporated with the goal of the minimum overhead in data transfer, data copy, context switching and cache miss. Generally, parallelization is incorporated in the following fashion:-

3.2.1 Multithreading. This method of parallelization is helpful in case CPU cores are under-utilised and idle most of the time to share the compute-heavy work and reduce the latency. In a user-facing online system, thousands of concurrent users request recommendations at a given time, so CPU cores are occupied most of the time to compute recommendations for the users, and there are no free cores to incorporate more threads. Spawning more threads will increase the load average of the system due to increase in number of waiting threads. This will lead to frequent context switching due to OS scheduler which adds the extra overhead latency of context switching, thread creation and cache miss. This leads to the scenario that the amount of time spent on context switching can exceed the amount of time spent on computation. This behaviour suggests that multi-threading will not reduce the latency and cannot scale well for online system involving compute intensive machine learning workloads. Efficient use of CPU cores is the key to increasing the throughput, latency per request and performance.

3.2.2 Register level parallelism/SIMD. Modern processors usually have a wider register length which can consume more data in a single instruction cycle. The proposed method is efficient when the same operation is performed on multiple continuous and independent data, and the result is stored in multiple independent yet continuous memory locations. Thus, enabling parallel computations with added advantage of more cache hits results in efficiently generating user-personalised scores.

3.2.3 Proposed Cosine Similarity. Since cosine similarity calculation is a reduction operation, in which dependent data is continuous in computer memory, the implementation mentioned in Alg.1 cannot leverage register level parallelism. We propose a batch cosine similarity operation to calculate similarity between user u with N products in recall set C that is hardware-agnostic, optimized and can be easily implemented in any high level programming language like Python [53]. It is implemented according to instruction set architecture, processors vector length, cache length and scope, temporal and spatial locality of data. It is powered by the SIMD (or vector parallelism) which performs 216× better than baseline implementation.

The column-major matrix $\hat{V}$ ($\hat{V} = V^T$, V is the original product embedding matrix V) represents the product embeddings, where i^{th} column contains the embedding e_i for the product p_i in the candidate set C and j^{th} row represents the j^{th} feature element in the embedding of every product in C, *i.e.* $\hat{V}_{ij}$ represents j^{th} feature element of the i^{th} product in C. Let us consider Intel® Xeon® Platinum 8171M CPU @ 2.60GHz processor as the processing unit. It has a maximum register length of 512 bits, implying it can consume 16 consecutive 32-bit floating point numbers in a single instruction cycle, essentially processing 16 elements in parallel. In addition to wide register length, it also has the functionality of Fused Multiply Add (FMA) in the instruction set, which allows multiplication and addition in a single clock cycle.

$$fma(a, b, c) = a + b \times c$$

The proposed optimised cosine similarity calculations can be represented by Eq. 5,

$$S = \sum_{j=0}^{k} (\forall \{ i \in \mathbb{N}, i \leq N \} \; u_j \times V_{ji}) \tag{5}$$

Algorithm 2 Optimised implementation of cosine similarity

Require: $(u, \hat{V})$
Ensure: *user personalisation scores* $= S$
  $S \leftarrow \{0.0, 0.0, 0.0....0.0\}_N$
  $i \leftarrow 1$
  while $i \leq k$ **do**
    $j \leftarrow 1$
    while $j \leq N$ **do** ▷
      $S[j : j+16] \leftarrow fma(S[j : j+16],\ u[i],\ \hat{V}[i][j : j+16])$
      $j \leftarrow j + 16$
    end while
    $i \leftarrow i + 1$
  end while

The proposed algorithm Alg. 2 ensures the maximum utilisation of accumulator and CPU registers with the least data transfer, data copy and cache miss overhead because of FMA instruction, as illustrated in Fig. 3a. It processes 16 elements in a single clock cycle, which is efficient and has lower latency than other implementations. It also ensures maximum spatial and temporal locality for cache hits, reducing the latency. Additionally, the proposed algorithm reduces number of total computations bringing down the processing footprint of each request, enough to localize each request to a single core of processor. Thus, providing significant scaling opportunity by enabling multiple instance of computation process to run in a single node. The number of instances is equal to number of cores available in the node which is empirically calculated.

3.3 Diversified Re-Ranking

After generating the user-personalized score s_i from the scoring layer, the score is merged with the product relevance score r_i to obtain the recommendation quality for each product. Let q_i be the recommendation quality score for the product p_i in the recall set C.

$$q_i = \beta s_i + (1 - \beta) r_i \tag{6}$$

where β is personalization hyperparameter which controls the personalization in recommendations, whose value is derived empirically. The candidate recommendations are then re-ranked ensuring diversity and relevance in the recommendations, and the top M products are shown to the user. The ranking of each product should maximize the recommendation quality score q_i and minimize the repetition of similar products to maximize user engagement and product discovery. In other words, the cumulative quality score and the distance between the top M recommended products should be maximized jointly. Cosine similarity between two product embedding can be used to calculate the distance between two products.

Let W be a subset of products sampled from set of candidate recommendations C such that $|W| = M$; For each $W \subseteq C$, let $\mathcal{P}(W)$ be the probability that the user will browse and add products to cart from the recommended product set W. $\mathcal{P}(W)$ should be maximized to get the most optimal set of top M recommendations. This behaviour can be modelled as a Determinantal Point Process (DPP) as illustrated by Mark *et al.* [59] and Warlop *et al.* [58].

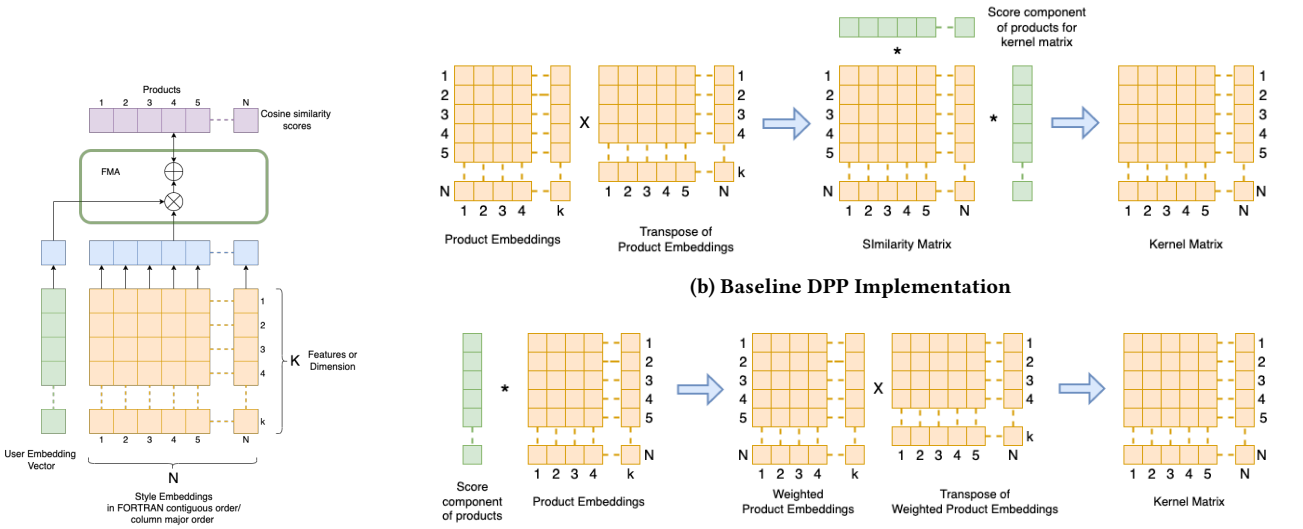
The process of diversification in the recommendations comprise of the following steps:-

(1) Learning the positive semi-definite kernel matrix L that can represent the point process.
(2) Sampling the top M products from the candidate from the DPP kernel matrix.

3.3.1 Learning the DPP kernel. Diversification in the recommended products considers the quality scores q and product-embedding vectors in recall set C.

Let d_{ij} is the cosine similarity distance between the embedding vector of the i^{th} and j^{th} product in the candidate set. The kernel matrix can be parameterized as follows:-

$$\begin{aligned} L_{ij} &= \exp(\alpha^2 q_i q_j) d_{ij} \\ \alpha &= \frac{\theta}{2 \times (1 - \theta)} \end{aligned} \tag{7}$$



(a) Optimised cosine similarity workflow

(b) Baseline DPP Implementation

(c) Proposed DPP Implementation

Figure 3: Optimization of cosine similarity and reformulation of DPP Kernel matrix generation

For diagonal elements since $i = j$, cosine distance between the same products will be 1,i.e. $d_{ij} = 1$. Eq. (7) can be simplified to

$$\begin{aligned} L_{ii} &= \exp(\alpha^2 q_i^2) \\ L_{ij} &= \exp(\alpha^2 q_i q_j) d_{ij} \quad for\ i \neq j \\ \alpha &= \frac{\theta}{2 \times (1-\theta)} \end{aligned} \tag{8}$$

$\theta \in [0, 1]$ is the tunable hyperparameter that controls the recommendations' relevance and diversity. In our experiments, we use $\theta = 0.7$, empirically derived, for best results in recommendations in terms of relevance and diversity. A high value of θ ensures a higher priority to the quality score of recommendations as the kernel matrix will be parameterized heavily on the quality of recommendations. Similarly, a small value of θ will prioritize the diversity of recommended products as the kernel matrix will be parameterized by cosine similarity distance between the embedding vectors.

Constructing the kernel matrix involves getting the cosine similarity distance d_{ij} for every (i, j) pair in candidate recommendations. Since the embedding vectors are normalized in the candidate generation phase, d_{ij} is the dot product of vector pair V_i and V_j. Dot product for every vector pair can be calculated by matrix multiplication of $\hat{V}$ by its transpose $\hat{V}^\intercal$. Let D is an $N \times N$ matrix in which each element D_{ij} contains the cosine similarity distance between i^{th} and j^{th} product in recall set C. In other words, $D_{ij} = dis(p_i, p_j) = d_{ij}$.

$$D = \hat{V}^\intercal \times \hat{V} \tag{9}$$

OpenBLAS [57] and Intel® MKL [56], which use the BLAS interface for OS's kernel-level optimized routines for linear algebra operations, can be utilized for computation of D. Each component of quality scores for product p_i, $\exp(\alpha \times q_i)$, is multiplied by the corresponding element of the similarity matrix D indexed by i to get the parameterized kernel matrix [12, 59]. The implementation is explained in Alg. 3 and illustrated in Fig. 3b. Let x be number of computations involved in matrix multiplication to generate similarity matrix D. Also, number of computations required to parameterise similarity matrix D to generate kernel matrix L is proportional to N^2 as shown in Alg. 3. Therefore, approx number of computations of generating kernel matrix from baseline implementation Alg.3 can be estimated to

$$Number\ of\ Computations = O(x) + O(N^2) \tag{10}$$

Algorithm 3 Baseline implementation of kernel matrix generation

Require: θ ▷ DPP hyperparamter for controlling relevance
Require: $q, \hat{V}$ ▷ Quality score and Product embeddings
Require: N ▷ Number of candidate recommendations
Ensure: L ▷ Parametrised Kernel Matrix

```
α ← θ ÷ (2 × (1 − θ))
L ← V̂ᵀ × V̂   ▷ Using BLAS matrix multiplication subroutine
i ← 1
while i ≤ N do
    while j ≤ N do
        L[i][j] ← exp(αq_i) × exp(αq_j) × L[i][j]
        j ← j + 1
    end while
    i ← i + 1
end while
```

3.3.2 Proposed DPP. We propose an efficient way to compute kernel matrix which involves less number of computation to generate kernel matrix than Alg. 3. Let $(*)$ represents element wise multiplication of a vector and matrix. In case of element wise multiplication of row vector and matrix, each element of vector is multiplied to every element of the corresponding columns. Similarly in case of element wise multiplication of column vector and matrix, each element

of vector is multiplied to every element of the corresponding rows. Let S be the row vector with component of scores, $S_i = \exp(\alpha \times q_i)$. The algorithm Alg. 3 can be represented using $(*)$ as follows:-

$$L = S^\intercal * (\hat{V}^\intercal \times \hat{V}) * S \tag{11}$$

Eq. 11 can be further simplified as,

$$\begin{aligned} & L = S^\intercal * (\hat{V}^\intercal \times \hat{V}) * S \\ \Leftrightarrow\ & L = S^\intercal * \hat{V}^\intercal \times \hat{V} * S \\ \Leftrightarrow\ & L = (S^\intercal * \hat{V}^\intercal) \times (\hat{V} * S) \\ \Leftrightarrow\ & L = (\hat{V} * S)^\intercal \times (\hat{V} * S) \end{aligned} \tag{12}$$

The term $\hat{V} * S$ represents a matrix with each column is the embedding vector e_i of each product p_i is elongated by the factor of the component of quality score $\exp(\alpha \times q_i)$. Using the simplification mentioned in Eq. 12, the implementation of kernel matrix creation can be further optimised by prior computation of $\hat{V} * S$ and the result of the matrix multiplication of the transpose of $\hat{V} * S$ with itself will give the parameterised kernel matrix L as illustrated in Fig. 3c.

Algorithm 4 Proposed Optimised implementation of kernel matrix generation

Require: θ ▷ DPP hyperparamter for controlling relevance
Require: $q, \hat{V}$ ▷ Quality score and Product embeddings
Require: N ▷ Number of candidate recommendations
Ensure: L ▷ Parametrised Kernel Matrix

```
α ← θ ÷ (2 × (1 − θ))
i ← 1
while i ≤ k do
    while j ≤ N do
        V[i][j] ← exp(αq_i) × exp(αq_j) × V[i][j]
        j ← j + 1
    end while
    i ← i + 1
end while
L ← V̂^⊺ × V̂   ▷ Using BLAS matrix multiplication subroutine
```

Since, the number of computations required to parameterise similarity matrix D in Alg. 4 to generate kernel matrix L is proportional to $N \times k$. Therefore, number of computations of generating kernel matrix from proposed implementation Alg. 4 can be estimated to

$$Number\ of\ Computations = O(x) + O(Nk) \tag{13}$$

For production use cases, since k can be treated as constant as it is independent on number of products in the recall set C and for various practical cases $k << N$, the algorithm Alg. 4 works better than Alg. 3 as it iterates over less number of elements to generate the parameterised kernel matrix.

3.3.3 Sampling the top relevant recommendations. Sampling the top M relevant products involves fetching the optimal set of products satisfying Eq. 7. Since, finding the optimal subset of products W_M is NP-Hard, a greedy algorithm [12] for submodular maximisation [42] is used.

4 IMPLEMENTATION DETAILS

We have implemented the proposed workflow of the cross-category recommendations system using Python3.8 with Numba [32] and NumPy [52].

- NumPy is a Python library used to store numerical data in the form of arrays. Internally, NumPy uses low-level functions and kernel libraries for fast mathematical operations. Also, it stores the arrays in a continuous memory buffer like any other low-level programming language like C or Fortran, making it easier to utilize SIMD due to the increase in memory colocation. User and Product embedding vectors are stored in the form of NumPy arrays.
- Numba is a JIT(just-in-time) python compiler that compiles high-level python functions to low-level machine code using the LLVM compiler library. SIMD capabilities can be utilized in by translating into low-level machine code according to embedding vectors memory layout, CPU specification and available registers on the cloud machines without manually specifying the compilation flags.

We avoided assembly-level SIMD to make the implementation portable across systems within x86 architecture. This also makes the implementation easy to develop, debug and maintain as programmer does not have to understand the underlying hardware.

5 EXPERIMENTS AND RESULTS

In this section we discuss the benchmarking of the proposed reformulated components and its impact on the whole recommendation pipeline against the non-optimized baseline method. We benchmarked the baseline workflow and the proposed workflow and established the latency and throughput gains on a private recommendation dataset. We also benchmarked the latency of various compute intensive components and the throughput and latency gain on baseline and proposed workflows.

5.1 Experimental setup

Recommendation dataset consists of 5000 query products and user pairs. These query products are randomly selected from a product catalogue of 1.5 million products. Each embedding are in 81-dimensional vector space $\mathcal{V}$ ($k = 81$) and each product have a recall set of 500 products($N = 500$) and 60 products are sampled as the top relevant products to recommend to user. The experiments are performed on Microsoft Azure cloud based virtual machines which uses Microsoft Azure Cloud Hypervisor based on Microsoft Hyper-V. The underlying hardware on the virtual machine has Intel® Xeon® Platinum 8171M CPU with frequency 2.60GHz with 16 cores and 64GB memory.

5.2 Experiments

We evaluate the impact of our proposed method on scalability of user-facing online services through a series of experiments on the existing and proposed pipelines. We first evaluate the baseline process pipeline (*P-Base*), which is a implementation of the personalization and diversification algorithm as described in [59]. *P-Base* is agnostic of the concurrency in online services and does not contain any explicit thread or multiprocessing control. We introduce

Table 1: Latency(in milliseconds) benchmarking of the baseline implementation(*P-Base*), single threaded baseline implementation(*P-Base-OnlineServ*) and proposed implementation(*P-Proposed*) of recommendation service. KO indicates request timeout.

Traffic	*P-Base*					*P-Base-OnlineServ*					*P-Proposed*				
	Latency (in ms)			Components (in ms)		Latency (in ms)			Components (in ms)		Latency (in ms)			Components (in ms)	
	Mean	P95	P99	Cos Sim	DPP	Mean	P95	P99	Cos Sim	DPP	Mean	P95	P99	Cos Sim	DPP
0.9 k	20.1	43.7	54	3.70	2.78	14.6	24.7	25.2	2.03	1.32	11.1	16.5	19.7	0.052	0.677
3.0 k	98.1	348	555	11.5	50.7	15.1	27.0	29.9	2.14	1.40	11.2	17.2	19.9	0.053	0.678
15.0 k	KO	KO	KO	KO	KO	16.7	32.7	41.5	2.74	1.63	12.0	19.2	25.3	0.053	0.776
42.0 k	KO	KO	KO	KO	KO	196	742	1050	4.18	2.28	18.4	41.5	61.3	0.059	0.909
45.0 k	KO	KO	KO	KO	KO	KO	KO	KO	KO	KO	20.5	43.5	**63.5**	0.061	0.936

Table 2: Ablation study (in milliseconds) of individual proposed components of the recommendation pipeline.

Throughput	Latency (in ms)											
	Baseline [P-Base-OnlineServ]			Baseline + Vec. Similarity [P-VecSim]			Baseline + Vec. DPP [P-VecDPP]			Baseline + Vec. Sim. & DPP [P-Proposed]		
	Mean	P95	P99	Mean	P95	P99	Mean	P95	P99	Mean	P95	P99
3.0 k	15.1	27.0	29.9	11.9	17.7	19.7	14.6	25.3	28.2	11.2	17.2	19.9
12.0 k	16.5	31.7	40.2	12.6	20.4	25.5	15.1	29.1	36.6	11.7	17.8	22.2
18.0 k	17.3	34.7	47.4	13.6	22.7	30.5	15.9	31.7	41.5	12.0	19.2	26.5
24.0 k	19.5	41.5	62.0	14.6	28.7	31.5	17.4	35.9	49.1	14.1	27.2	29.0
30.0 k	22.6	51.3	75.2	15.1	34.5	34.9	22.1	45.7	64.3	14.5	32.5	33.1
36.0 k	50.9	150.0	217.0	19.4	44.7	49.3	39.1	62.5	78.2	18.2	37.6	41.9

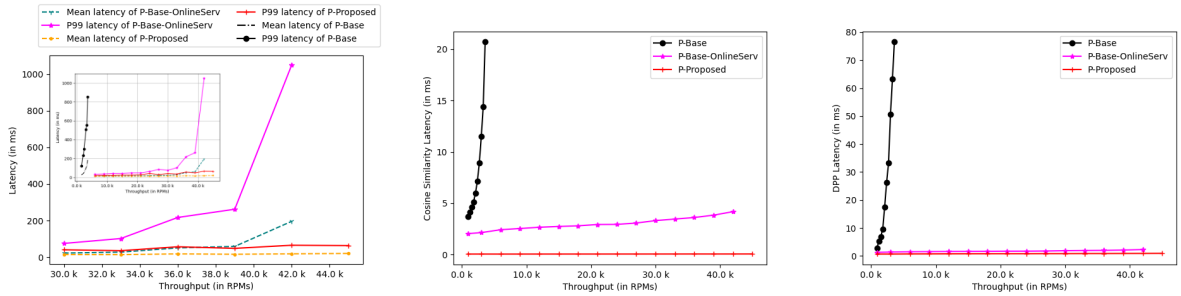
Table 3: Benchmarking the impact of using multithreading in the proposed implementation of the recommendation service.

Throughput	Latency (in ms)											
	1 Threads			4 Threads			8 Threads			16 Threads		
	Mean	P95	P99	Mean	P95	P99	Mean	P95	P99	Mean	P95	P99
0.9 k	11.1	16.5	19.7	15.0	24.0	26.7	16.6	25.3	33.4	20.1	43.7	54
1.5 k	11.2	16.9	19.8	16.2	25.8	29.7	17.9	34.4	59.3	26.9	75.2	120
3.0 k	11.2	17.2	19.9	18.2	30.6	45.7	27.9	79.1	131	98.1	348	508

optimal multiprocessing conditions, suited to online services handling concurrent requests, to *P-Base* and establish new pipeline *P-Base-OnlineServ* by controlling number of intra and inter process threads spawned for each request. Finally, we experiment with our proposed pipeline *P-Proposed* which contains the thread control mentioned before and also the vectorized implementation of personalization (Alg. 2) and diversification (Alg. 4). To analyze the scalability of the pipelines under consideration we subject each to varying load of concurrent requests ranging from 900 to 45000 Requests Per Minute (0.9K-45K RPM). We monitor the change of latency under the increasing throughput of requests for each of the pipeline and ascertain the breaking-point as the load under which the pipelines go out-of-service due to congestion and non-serviced timeout of incoming requests. The metrics used for comparative analysis of the experiments are average and peak latency of end-to-end service of a request. Average latency is nothing but the mean of all observed latency in a observation set and peak latency refers to the 99th percentile of all observations, also written as P-99. The permissible limit of P-99 latency is 100ms. The observations are recorded in Table. 1, where latency numbers in red represents unacceptable because P-99 latency exceeds permissible limit and 'KO' represents the experiment could not complete at higher level of concurrency and requests being terminated by system.

5.3 Comparative analysis

Table. 1 shows the observed latency numbers of the *P-Base*, *P-Base-OnlineServ* and *P-Proposed* under various loads on the system simulated with the real-world traffic simulator. As mentioned above, *P-Base* does not contain any explicit thread control built into it and runs with out-of-box settings of high level production APIs in Python. At 0.9K RPM traffic the pipeline runs without any issue, serving requests in 20.1ms on average and the peak latency lies around 54ms. However, with increase in the load, the pipeline starts



(a) Comparison of P-99 latencies **(b) Cosine Similarity** **(c) DPP**

Figure 4: a) Comparison of mean and peak P-99 latency of single threaded baseline implementation (*P-Base-OnlineServ*) and proposed implementation (*P-Proposed*). Complete benchmarking including *P-Base* in inset. Benchmarking of b) Optimization of cosine similarity and c) reformulation of DPP Kernel matrix generation

to choke up and finally gives away under 3.0K RPM load with average latency of 98.1ms and the peak latency climbing upto 555ms. Thus, it can be concluded that *P-Base* is not vertically scalable. From our analysis we ascertained the over usage of multi-threaded parallelization to be the source of bottleneck in the pipeline. Python APIs, in general, are designed with goal of achieving optimal performance of single request on dedicated infra. Thus, lift-and-shift implementation of such APIs in user facing online services that are expected to handle large concurrency is not suitable. Hence, we decided to re-establish the baseline after optimizing the multi-threaded behaviour to make the implementation more suitable to online services for fair comparison. Next we benchmark the thread controlled implementation (*P-Base-OnlineServ*) of Alg. 1 and Alg. 3. The pipeline becomes much more resilient under scale as it can easily serve requests in 15.1ms on average under a load of 3.0K RPM. It is easily able to handle even 15.0K RPM with 16.7ms average and 41.5ms Peak latency. However, breaking-point for *P-Base-OnlineServ* occurs around 42.0K RPM traffic. Next we benchmark our proposed pipeline *P-Proposed* with vectorized implementation of Alg. 2 and Alg. 4 and optimal thread control, derived empirically. *P-Proposed* performs much better under the basic load of 0.9K throughput with average serving latency of 11.1ms and peak latency at P99 of 19.7ms. We also observe that *P-Proposed* scales efficiently under increasing load of 3K and 15K RPM load with less than 10% increase in average serving latency. Under 15.0K RPM, *P-Proposed* has peak latency of only 25.3ms which is significantly improved over *P-Base-OnlineServ*. We validated the proposed pipeline *P-Proposed* to be operational under traffic load of 45.0K RPM. Under this extreme load, the pipeline is still able to serve requests under 20.5ms on average and with peak latency (P99) of 63.5ms. Fig. 4a shows the latency vs. throughput plot of *P-Base*, *P-Base-OnlineServ* and *P-Proposed* which visually depicts the improvement in vertical scalability between the pipelines. Considering horizontal scaling to serve production traffic beyond 45k RPM, we can conclude that *P-Proposed* requires 50-times less hardware instances than *P-Base* and 3-times less hardware instances than *P-Base-OnlineServ* to support a certain amount of traffic. Next, we benchmark each of the individual components impacted by our proposed optimization in *P-Proposed*. Table. 1 also shows the observed average latency for cosine similarity and DPP calculations in *P-Base*, *P-Base-OnlineServ* and *P-Proposed* under increasing traffic. Both cosine similarity and DPP module latency grows aggressively in *P-Base* and reaches 11.5ms and 50.7ms on average respectively. This validates the earlier observations that 3.0K RPM is the breaking-point traffic for *P-Base*. At 0.9K RPM traffic, in *P-Base-OnlineServ*, latency of cosine similarity computation is 2.03ms while that of DPP is 1.32ms on average which is acceptable but on the higher side for a real-time recommendation service. However, with increase in the traffic, latency of both components increase rapidly. This causes 'thrashing' in the system *i.e.* more time is wasted in context switching than computing. This causes congestion and non-served request queue build-up in the system, leading to rapid increase in latency. At 42.0K RPM traffic, average latency of cosine similarity goes upto 4.18ms and DPP latency reaches upto 2.28ms. Thus, it can be concluded that the baseline implementation of cosine similarity and DPP are not entirely vertically scalable. However, in contrast, the cosine similarity and DPP component serving latency in the proposed vectorized implementation *P-Proposed* is significantly lower compared to the baseline implementation *P-Base-OnlineServ*.

In case of *P-Proposed*, vectorized cosine similarity incurs a average serving latency of 52μs under nominal load of 0.9K RPM, which only increases upto 61μs under 45.0K RPM traffic. This indicates the proposed vectorized implementation of Alg. 2 is highly scalable and can perform with almost constant latency under aggressively increasing requests traffic. Fig. 4b shows the comparative plots of mean latency of cosine similarity pipeline. Similarly, the reformulated vectorized DPP operates with average execution latency of 677μs under 0.9K RPM traffic and is able to constrain the average latency to just under 1ms (936μs) when subjected to a traffic of 45.0K RPM. Fig. 4c shows the comparative plots of mean latency of DPP pipeline. Although the plots of DPP component in *P-Base-OnlineServ* and *P-Proposed* seem to be close to each other but that is due to the unconstrained growth of the DPP latency in *P-Base*.

Thus, the comparative analysis of critical components of our proposed method comprehensively support our claims of significantly improving the scalability of the diversified recommendation pipeline. We take a further closer look at the impact of these critical components later in Sec. 6 by evaluating their influence in isolation.

6 ABLATION STUDY

In this section, we analyse the results of our ablation study of the individual components of the proposed method to assess their individual and collective influence on improving the scalability of the pipeline. We also evaluate the impact of using traditional optimization methods like threaded parallelization on our proposed method.

6.1 Impact of individual components

We analyse the influence of individual proposed components in improving the scalability of the recommendation pipeline through a set of ablation experiments. Table. 2 shows the result of the ablation experiments. The behaviour of *P-Base-OnlineServ* is already discussed above and noted to be not scalable to a high degree. However, upon introducing the proposed vectorized Cosine similarity (Alg. 2) to this pipeline, we observed that the resultant pipeline *P-VecSim* becomes significantly more scalable. *P-VecSim* can serve with an average and P-99 latency of 19.4ms and 49.3ms respectively under 36.0K RPM traffic. This shows significant improvement over *P-Base-OnlineServ* which have average and P-99 latency of 50.9ms and 217.0ms respectively under similar conditions. Next, we evaluate the influence of the proposed vectorized DPP (Alg. 4) module. Upon introducing same to *P-Base-OnlineServ*, the resultant pipeline *P-VecDPP* also becomes readily more scalable. The average and peak latency of *P-VecDPP*, 39.1ms and 78.2ms respectively, when subjected to a traffic of 36.0K RPM, remains decently under control to keep the pipeline operational. Introducing both vectorized similarity and vectorized DPP to *P-Base-OnlineServ* creates our proposed pipeline *P-Proposed*, which by virtue of including both the scalable components demonstrates highest scalability. Subjected to a traffic of 36.0K RPM, *P-Proposed* incurs average and peak latency of 18.2ms and 41.9ms respectively, growing only 60% over the average latency of 11.2ms under 3.0K RPM. The peak latency growth under maximum traffic is also constrained under only to 3 times the original peak latency for a 12x growth in traffic. These observations prove our claims of proposing individual scalable components, whose sum is even greater than the parts in terms of improving scalability of the recommendation pipeline.

6.2 Impact of threaded parallelization

We also evaluated impact of using conventional optimization of multithreaded parallelism on scalability of proposed recommendation system pipeline. Table. 3 shows that the single threaded pipeline scales most efficiently under increasing traffic with less than 10% increase in the average serving latency over the range of loads. However, even with 4 threads per request, the average and peak latency doubles at 3.0K RPM than the 1-thread pipeline. 8 and 16 threaded implementations cross real-time SLA limits under only 3K RPM load with nearly 7x and 25x growth in peak latency over 1-thread pipeline. Thus, it can be concluded that naive usage of multithreading without vectorization does not work well in practice for online recommendation systems.

7 OPEN PROBLEMS AND FUTURE WORK

We have conducted in-depth experiments to evaluate the performance of our proposed modification to the similarity and DPP algorithms to leverage benefits of SIMD multiprocessing in server and virtual machine environments. Evaluating the performance of the same algorithms with SIMD multiprocessing in a containerized environment e.g. Kubernetes (k8s) is one of the goals of our future work. We have also been exposed solely to x86 and x86_64 Instruction Set Architecture (ISA) as these are the most commonly found instruction set in server and cloud VM environments. Evaluating the impact of SIMD multiprocessing on different ISAs such as ARM also remains an open problem to be addressed in future publications from this body of work. On the other hand, we have focused on constituting portable SIMD accelerated method and thus had to look beyond Assembly level SIMD which becomes bound to specific CPUs with specific register counts. However, this problem can be solved alternatively by bypassing the dependency of register lengths in Assembly level SIMD and we believe considerable amount of research scope is present in that area.

8 CONCLUSION

In this paper, we explored the opportunities of moving past conventional optimization strategies like threaded parallelization for CPU only online recommendation systems and adopting the SIMD optimizations by redesigning critical components, with motivations of vectorization, of a 'complement' recommendation system. We experimentally show that vectorized implementation of the well known blocks like personalization and diversification can be made significantly more scalable by utilizing SIMD compute powers of modern day CPUs. We use portable SIMD constructs in Python to make the implementation easily portable across different CPUs and architectures. The approach advocated-for in this work can be extended to many available recommendation systems that are deployed in CPU only servers and can be pivotal shift towards making efficient large scale e-commerce services. In addition to endorsing the use in recommender systems, we propose and encourage further exploration of proposed algorithm in vector similarity operations, particularly within applications such as vector databases, which serve as efficient storage and retrieval systems for vector embeddings (e.g., Milvus [40], Pinecone [44], ChromaDB [13]). As these applications are heavily reliant on similarity algorithms of the kind proposed in this work, we believe further research into same will help make these applications more optimized and real-time. Furthermore, considering the influence of these applications in the increasingly popular Generative-AI paradigm, optimizing them holds the promise of rendering Generative-AI applications more responsive and suitable for large-scale deployments in real-world scenarios.

REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. 2007. New Recommendation Techniques for Multicriteria Rating Systems. *IEEE Intelligent Systems* 22, 3 (2007), 48–55. https://doi.org/10.1109/mis.2007.58

[2] Pankaj Agarwal, Sreekanth Vempati, and Sumit Borar. 2018. Personalizing similar product recommendations in fashion e-commerce. *AI for fashion, The third international workshop on fashion and KDD* (2018).

[3] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. 2009. Diversifying Search Results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining* (Barcelona, Spain) *(WSDM '09)*. Association for Computing Machinery, New York, NY, USA, 5–14. https://doi.org/10.1145/1498759.1498766

[4] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. *SIGMOD Rec.* 22, 2 (jun 1993),

207–216. https://doi.org/10.1145/170036.170072
[5] Rémi Bardenet and Michalis Titsias RC AUEB. 2015. Inference for determinantal point processes without spectral knowledge. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/2f25f6e326adb93c5787175dda209ab6-Paper.pdf
[6] Alexei Borodin. 2015. 231Determinantal point processes. In *The Oxford Handbook of Random Matrix Theory*. Oxford University Press. https://doi.org/10.1093/oxfordhb/9780198744191.013.11 arXiv:https://academic.oup.com/book/0/chapter/365880931/chapter-ag-pdf/45289415/book_43656_section_365880931.ag.pdf
[7] Jaime Carbonell and Jade Goldstein. 1998. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Melbourne, Australia) *(SIGIR '98)*. Association for Computing Machinery, New York, NY, USA, 335–336. https://doi.org/10.1145/290941.291025
[8] Josiah Carlson. 2013. *Redis in action*. Simon and Schuster.
[9] Olivier Chapelle, Shihao Ji, Ciya Liao, Emre Velipasaoglu, Larry Lai, and Su-Lin Wu. 2011. Intent-based diversification of web search results: metrics and algorithms. *Information Retrieval* 14 (2011), 572–592.
[10] Olivier Chapelle, Shihao Ji, Ciya Liao, Emre Velipasaoglu, Larry Lai, and Su-Lin Wu. 2011. Intent-based diversification of web search results: metrics and algorithms. *Information Retrieval* 14, 6 (2011), 572–592. https://doi.org/10.1007/s10791-011-9167-7
[11] Laming Chen, Guoxin Zhang, and Hanning Zhou. 2017. Improving the diversity of top-N recommendation via determinantal point process. (2017).
[12] Laming Chen, Guoxin Zhang, and Hanning Zhou. 2018. Fast Greedy MAP Inference for Determinantal Point Process to Improve Recommendation Diversity. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 5627–5638.
[13] ChromaDB. 2023. *Chroma : the AI-native open-source embedding database*. https://www.trychroma.com/
[14] Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. 2008. SIGIR - Novelty and diversity in information retrieval evaluation. *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '08* NA, NA (2008), 659–666. https://doi.org/10.1145/1390334.1390446
[15] Angeline G. Close and Monika Kukar-Kinney. 2010. Beyond buying: Motivations behind consumers' online shopping cart use. *Journal of Business Research* 63, 9 (2010), 986–992. https://doi.org/10.1016/j.jbusres.2009.01.022 Advances in Internet Consumer Behavior & Marketing Strategy.
[16] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190
[17] Van Dang and W. Bruce Croft. 2012. Diversity by Proportionality: An Election-Based Approach to Search Result Diversification. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Portland, Oregon, USA) *(SIGIR '12)*. Association for Computing Machinery, New York, NY, USA, 65–74. https://doi.org/10.1145/2348283.2348296
[18] Marina Drosou and Evaggelia Pitoura. 2010. Search result diversification. *ACM SIGMOD Record* 39, 1 (2010), 41–47. https://doi.org/10.1145/1860702.1860709
[19] Mike Gartrell, Ulrich Paquet, and Noam Koenigstein. 2016. RecSys - Bayesian Low-Rank Determinantal Point Processes. *Proceedings of the 10th ACM Conference on Recommender Systems* NA, NA (2016), 349–356. https://doi.org/10.1145/2959100.2959178
[20] Jennifer Gillenwater. 2014. Approximate inference for determinantal point processes. *NA* NA, NA (2014), NA–NA.
[21] Jennifer A Gillenwater, Alex Kulesza, Emily Fox, and Ben Taskar. 2014. Expectation-Maximization for Learning Determinantal Point Processes. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/4462bf0ddbe0d0da40e1e828ebebeb11-Paper.pdf
[22] Sreenivas Gollapudi and Aneesh Sharma. 2009. WWW - An axiomatic approach for result diversification. *Proceedings of the 18th international conference on World wide web - WWW '09* NA, NA (2009), 381–390. https://doi.org/10.1145/1526709.1526761
[23] Yoshinori Hijikata, Takuya Shimizu, and Shogo Nishida. 2009. Discovery-Oriented Collaborative Filtering for Improving User Satisfaction. In *Proceedings of the 14th International Conference on Intelligent User Interfaces* (Sanibel Island, Florida, USA) *(IUI '09)*. Association for Computing Machinery, New York, NY, USA, 67–76. https://doi.org/10.1145/1502650.1502663
[24] Henning Hohnhold, Deirdre O'Brien, and Diane Tang. 2015. KDD - Focusing on the Long-term: It's Good for Users and Business. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* NA, NA (2015), 1849–1858. https://doi.org/10.1145/2783258.2788583
[25] J Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint Virág. 2006. Determinantal processes and independence. *Probability surveys* 3 (2006), 206–229.
[26] Chun-Wa Ko, Jon Lee, and Maurice Queyranne. 1995. An exact algorithm for maximum entropy sampling. *Operations Research* 43, 4 (1995), 684–691. https://doi.org/10.1287/opre.43.4.684
[27] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. 1997. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM* 40, 3 (1997), 77–87.
[28] Joseph A Konstan, John Riedl, Al Borchers, and Jonathan L Herlocker. 1998. Recommender systems: A grouplens perspective. In *Recommender Systems: Papers from the 1998 Workshop (AAAI Technical Report WS-98-08)*. AAAI Press Menlo Park, 60–64.
[29] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37. https://doi.org/10.1109/MC.2009.263
[30] Alex Kulesza and Ben Taskar. 2011. K-DPPs: Fixed-Size Determinantal Point Processes. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (Bellevue, Washington, USA) *(ICML'11)*. Omnipress, Madison, WI, USA, 1193–1200.
[31] Alex Kulesza and Ben Taskar. 2011. Learning Determinantal Point Processes. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence* (Barcelona, Spain) *(UAI'11)*. AUAI Press, Arlington, Virginia, USA, 419–427.
[32] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A LLVM-Based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (Austin, Texas) *(LLVM '15)*. Association for Computing Machinery, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/2833157.2833162
[33] Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. 2010. SIGIR - Temporal diversity in recommender systems. *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10* NA, NA (2010), 210–217. https://doi.org/10.1145/1835449.1835486
[34] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344
[35] Odile Macchi. 1975. The coincidence approach to stochastic point processes. *Advances in Applied Probability* 7, 1 (1975), 83–122.
[36] Zelda Mariet and Suvrit Sra. 2015. Fixed-point algorithms for learning determinantal point processes. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 2389–2397. https://proceedings.mlr.press/v37/mariet15.html
[37] Zelda E. Mariet and Suvrit Sra. 2016. Kronecker Determinantal Point Processes. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/bad5f33780c42f2588878a9d07405083-Paper.pdf
[38] Julian McAuley, Rahul Pandey, and Jure Leskovec. 2015. Inferring Networks of Substitutable and Complementary Products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD '15)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2783258.2783381
[39] Sean M McNee, John Riedl, and Joseph A Konstan. 2006. CHI Extended Abstracts - Being accurate is not enough: how accuracy metrics have hurt recommender systems. *CHI '06 Extended Abstracts on Human Factors in Computing Systems* NA, NA (2006), 1097–1101. https://doi.org/10.1145/1125451.1125659
[40] Milvus. 2023. *Milvus: Vector database*. https://milvus.io/
[41] Houssam Nassif, Kemal Oral Cansizlar, Mitchell Goodman, and SVN Vishwanathan. 2016. Diversifying Music Recommendations. *arXiv: Multimedia* NA, NA (2016), NA–NA.
[42] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming* 14, 1 (1978), 265–294. https://doi.org/10.1007/bf01588971
[43] Yonathan Perez, Michael Schueppert, Matthew Lawlor, and Shaunak Kishore. 2015. Category-Driven Approach for Local Related Business Recommendations. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (Melbourne, Australia) *(CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 73–82. https://doi.org/10.1145/2806416.2806495
[44] Inc. Pinecone Systems. 2023. *Pinecone: Vector Database for Vector Search*. https://www.pinecone.io/
[45] Davood Rafiei, Krishna Bharat, and Anand Shukla. 2010. WWW - Diversifying web search results. *Proceedings of the 19th international conference on World wide web - WWW '10* NA, NA (2010), 781–790. https://doi.org/10.1145/1772690.1772770
[46] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: An Open Architecture for Collaborative Filtering of

Netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work* (Chapel Hill, North Carolina, USA) *(CSCW '94)*. Association for Computing Machinery, New York, NY, USA, 175–186. https://doi.org/10.1145/192844.192905

[47] Rodrygo LT Santos, Craig Macdonald, and Iadh Ounis. 2010. WWW - Exploiting query reformulations for web search result diversification. *Proceedings of the 19th international conference on World wide web - WWW '10* NA, NA (2010), 881–890. https://doi.org/10.1145/1772690.1772780

[48] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (Hong Kong, Hong Kong) *(WWW '01)*. Association for Computing Machinery, New York, NY, USA, 285–295. https://doi.org/10.1145/371920.372071

[49] Cyrus Shahabi, Farnoush Banaei-Kashani, Yi-Shin Chen, and Dennis McLeod. 2001. Yoda: An Accurate and Scalable Web-Based Recommendation System. In *Cooperative Information Systems*, Carlo Batini, Fausto Giunchiglia, Paolo Giorgini, and Massimo Mecella (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 418–432.

[50] V. Srinivasan, Brian Bulkowski, Wei-Ling Chu, Sunil Sayyaparaju, Andrew Gooding, Rajkumar Iyer, Ashish Shinde, and Thomas Lopatic. 2016. Aerospike: Architecture of a Real-Time Operational DBMS. *Proc. VLDB Endow.* 9, 13 (sep 2016), 1389–1400. https://doi.org/10.14778/3007263.3007276

[51] Choon Hui Teo, Houssam Nassif, Daniel Hill, Sriram Srinivasan, Mitchell Goodman, Vijai Mohan, and SVN Vishwanathan. 2016. RecSys - Adaptive, Personalized Diversity for Visual Discovery. *Proceedings of the 10th ACM Conference on Recommender Systems* NA, NA (2016), 35–38. https://doi.org/10.1145/2959100.2959171

[52] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. 2011. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering* 13, 2 (2011), 22–30. https://doi.org/10.1109/MCSE.2011.37

[53] Guido Van Rossum and Fred L Drake. 2009. *Python 3 reference manual*. CreateSpace.

[54] Saúl Vargas, Linas Baltrunas, Alexandros Karatzoglou, and Pablo Castells. 2014. RecSys - Coverage, redundancy and size-awareness in genre diversity for recommender systems. *Proceedings of the 8th ACM Conference on Recommender systems - RecSys '14* NA, NA (2014), 209–216. https://doi.org/10.1145/2645710.2645743

[55] Erik Vee, Utkarsh Srivastava, Jayavel Shanmugasundaram, Prashant Bhat, and Sihem Amer Yahia. 2008. ICDE - Efficient Computation of Diverse Query Results. *2008 IEEE 24th International Conference on Data Engineering* NA, NA (2008), 228–236. https://doi.org/10.1109/icde.2008.4497431

[56] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. *Intel Math Kernel Library*. Springer International Publishing, Cham, 167–188. https://doi.org/10.1007/978-3-319-06486-4_7

[57] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. 2013. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on X86 CPUs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '13)*. Association for Computing Machinery, New York, NY, USA, Article 25, 12 pages. https://doi.org/10.1145/2503210.2503219

[58] Romain Warlop, Jérémie Mary, and Mike Gartrell. 2019. Tensorized Determinantal Point Processes for Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 1605–1615. https://doi.org/10.1145/3292500.3330952

[59] Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H. Chi, and Jennifer Gillenwater. 2018. Practical Diversified Recommendations on YouTube with Determinantal Point Processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18)*. Association for Computing Machinery, New York, NY, USA, 2165–2173. https://doi.org/10.1145/3269206.3272018

[60] An Yan, Chaosheng Dong, Yan Gao, Jinmiao Fu, Tong Zhao, Yi Sun, and Julian Mcauley. 2022. Personalized Complementary Product Recommendation. In *Companion Proceedings of the Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 146–151. https://doi.org/10.1145/3487553.3524222

[61] Cong Yu, Laks Lakshmanan, and Sihem Amer-Yahia. 2009. EDBT - It takes variety to make a world: diversification in recommender systems. *Proceedings of the 12th International Conference on Extending Database Technology Advances in Database Technology - EDBT '09* NA, NA (2009), 368–378. https://doi.org/10.1145/1516360.1516404

[62] Mi Zhang and Neil Hurley. 2008. RecSys - Avoiding monotony: improving the diversity of recommendation lists. *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08* NA, NA (2008), 123–130. https://doi.org/10.1145/1454008.1454030

[63] Jiaqian Zheng, Xiaoyuan Wu, Junyu Niu, and Alvaro Bolivar. 2009. Substitutes or Complements: Another Step Forward in Recommendations. In *Proceedings of the 10th ACM Conference on Electronic Commerce* (Stanford, California, USA) *(EC '09)*. Association for Computing Machinery, New York, NY, USA, 139–146. https://doi.org/10.1145/1566374.1566394

[64] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. WWW - Improving recommendation lists through topic diversification. *Proceedings of the 14th international conference on World Wide Web - WWW '05* NA, NA (2005), 22–32. https://doi.org/10.1145/1060745.1060754