Linear-Time Self Attention with Codeword Histogram for Efficient Recommendation

Yongji Wu[12*], Defu Lian[2†], Neil Zhenqiang Gong[1], Lu Yin[3],
Mingyang Yin[3], Jingren Zhou[3], Hongxia Yang[3†]
[1]Duke University, [2]University of Science and Technology of China, [3]Alibaba Group
[1]{yongji.wu769, neil.gong}@duke.edu, [2]liandefu@ustc.edu.cn,
[3]{theodore.yl, hengyang.ymy, jingren.zhou, yang.yhx}@alibaba-inc.com

ABSTRACT

Self-attention has become increasingly popular in a variety of sequence modeling tasks from natural language processing to recommendation, due to its effectiveness. However, self-attention suffers from quadratic computational and memory complexities, prohibiting its applications on long sequences. Existing approaches that address this issue mainly rely on a sparse attention context, either using a local window, or a permuted bucket obtained by locality-sensitive hashing (LSH) or sorting, while crucial information may be lost. Inspired by the idea of vector quantization that uses cluster centroids to approximate items, we propose LISA (**LI**near-time **S**elf **A**ttention), which enjoys both the effectiveness of vanilla self-attention and the efficiency of sparse attention. LISA scales linearly with the sequence length, while enabling full contextual attention via computing differentiable histograms of codeword distributions. Meanwhile, unlike some efficient attention methods, our method poses no restriction on casual masking or sequence length. We evaluate our method on four real-world datasets for sequential recommendation. The results show that LISA outperforms the state-of-the-art efficient attention methods in both performance and speed; and it is up to 57x faster and 78x more memory efficient than vanilla self-attention.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Users and interactive retrieval.*

KEYWORDS

self-attention, efficient-attention, sequential recommendation, quantization

ACM Reference Format:
Yongji Wu, Defu Lian, Neil Zhenqiang Gong, Lu Yin, Mingyang Yin, Jingren Zhou, Hongxia Yang. 2021. Linear-Time Self Attention with Codeword Histogram for Efficient Recommendation. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3442381.3449946

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449946

1 INTRODUCTION

Since the introduction of the self-attention mechanism in Transformers [33], it has seen incredible success in a variety of sequence modeling tasks in a variety of fields, such as machine translation [4], object detection [37], music generation [**?**] and bioinformatics [25]. Recently, self-attention has also demonstrated its formidable power in recommendation [17, 30, 45].

However, despite impressive performance attributable to its ability to identify complex dependencies between elements in input sequences, self-attention based models suffers from soaring computational and memory costs when facing sequences of greater length. As a consequence of computing attention scores over the entire sequence for each token, self-attention takes $O(L^2)$ operations to process an input sequence of length L. This hinders the scalability of models built on self-attention in many settings.

Recently, a number of solutions have been proposed to address this issue. The majority of these approaches [1, 2, 6, 18, 27, 32, 43] leverages sparse attention patterns, limiting the number of keys that each query can attend to. Although these sparse patterns can be established in a variety of content-depended ways like LSH [18], sorting [32] and k-means clustering [27], crucial information may be lost by clipping the receptive field for each query. While successfully reducing the cost of computing attention weights from $O(L^2D)$ to $O(LBD)$, where B is the fixed bucket size, extra cost incurs in assigning the keys/values into buckets. This cost typically is still quadratic with respect to L, and it may cause significant overheads dealing with shorter sequences. We observe that Reformer [18] could be 7.6x slower than the vanilla Transformer on sequences of length 128. Other techniques are also being employed to improve the efficiency of self-attention. For instance, low-rank approximations of the attention weights matrix is used in [36]. This method, however, only supports a bidirectional attention mode and assumes a fixed length of input sequences.

We observe that self-attention essentially computes a weighted average of the input sequences for each query, and the weights are computed based on the inner product between the query and the keys. For each query, keys with larger inner product will be paid more attention to. We relate this to the Maximum Inner Product Search (MIPS) problem. The MIPS problem is of great importance in many machine learning problems [11, 20, 28], and fast approximate MIPS algorithms are well studied by researchers. Among them, vector (product) quantization [8, 13, 14] has been a popular and successful method. Armed with vector quantization, we no

[*]This work was conducted while he was a research intern at Alibaba Group.
[†]Corresponding authors.

Figure 1: Illustration of codeword histograms. There are four codewords in each of the four codebooks.

longer have to exhaustively compute the inner product between a given query and all the# points in the database. We can only compute that for the centroids (i.e., codewords), whereis a budget hyperparameter. We therefore successfully avoid redundant computations since the points belong to the same centroid share the same inner product with the query.

The idea of vector quantization has also been applied to compress the item embedding matrix and improve the memory and search e ciency of recommendation systems [5, 23]. In the state-of-the-art lightweight recommendation model, LightRec [23], a set of di erentially learnable codebooks are used to encode items, each of which is composed of codewords. An item is represented by a composition of the most similar codeword within each codebook. Hence we only need to store the indices of its corresponding codewords, instead of its embedding vector. Since the codeword index in a codebook can be compactly encoded with log, bits, the overall memory requirements to store item representations can be reduced from4# bytes to $\frac{1}{8}$# log, ¸ 4 , bytes [23].

Inspired by the bene t that redundant inner product computations can be circumvented in MIPS algorithms based on vector quantization, and the ability of using codebooks to quantize any embedding matrix, we propose LISA (LInear-time Self-Attention), an e cient attention mechanism based on computing codeword histograms. Equipped with a series of codebooks to encode items (or any form of tokens), LISA can dramatically reduce the costs of inner product computation in a similar vein. Since each item (token) is represented as a composition of codewords, and the entire input sequence can be compressed to a histogram of codewords for each codebook (illustrated in Figure 1), we are essentially performing attention over codewords. The histograms are used to compute the attention weights matrix in$ ¹! º time. We then pool over the codewords with the attention weights to get the outputs. To enable self-attention in a unidirectional setting (i.e., with casual masking [18]), we can resort to the mechanism of pre x-sums and compute a histogram at each position of the sequence.

Compared to the e cient attention methods that rely on sparse patterns, our proposed method performs full contextual attention over the input sequence, with a computational and memory complexity linear in the sequence length. Our proposed method also enjoys the compression of item embeddings brought by LightRec. Particularly, in an online recommendation setting, our method can encapsulate a user's entire history with a xed size histogram, greatly reduce the storage costs.

Our contributions can be summarized as follows:

We propose LISA (LInear-time Self-Attention), a novel attention mechanism for e cient recommendation that reduces the complexity of computing attention scores from$ ¹! ² º to $ ¹! , º, while simultaneously enabling model compression. The total number of codewords, is a budget hyperparameter balancing between performance and speed.

We also propose two variants of LISA, one of them allows soft codeword assignments, and the other uses a separate codebook to encode sequences. These techniques allow us to use much smaller codebooks, resulting in further e ciency improvements.

We conduct extensive experiments on four real-world datasets. Our proposed method obtains similar performance to vanilla self-attention, while signi cantly outperforms the state-of-the-art e cient attention baselines in both performance and e ciency.

2 RELATED WORK

2.1 Applications of Self-Attention Mechanisms

The scaled dot product self-attention introduced in Transformers [33] has been extensively used in natural language understanding [10, 41]. As a powerful mechanism that connects all tokens in the inputs with a pooling operation based on relevance, self-attention has also made tremendous impacts in various other domains like computer vision [39, 44], graph learning [34].

Recently, self-attention networks are successfully applied to sequential recommendation. Kang and McAuley [17] adapted a Transformer architecture by optimizing the binary cross-entropy loss based on inner product preference scores, while Zhang et al. [45] propose to optimize a triplet margin loss based on Euclidean distance preference. Self-attention is also used for geographical modeling in location recommendation [22, 24]. They have demonstrated signi cant performance improvements over the RNN based models.

2.2 Improving E ciency of Attention

Considerable e orts have been made trying to scale Transformers to long sequences. Transformer-XL in [9] captures longer-term dependency by employing a segment-level recurrent mechanism, which splits the inputs into segments to perform attention. Sukhbaatar et al. [29] limited the self-attention context to the closest samples. However, these techniques do not improve the$ ¹! ² º asymptotic complexity of self-attention.

In another line of work, attempts in reducing the asymptotic complexity are made. Child et al. [6] proposed to factorize the attention computation into local and strided ones. Tay et al. [32], on the other hand, improved local attention by introducing a differentiable sorting network to re-sort the buckets. Reformer [18] hashes the query-keys into buckets via hashing functions based on random projection, and attention is computed within each bucket. In a similar manner, Roy et al. [27] assign tokens to buckets through clustering. Built on top of ETC [1], Big Bird [43] considers a mixture of various sparse patterns, including sliding window attention and random attention. Clustered Attention, introduced in [35], however, groups queries into clusters and perform attention on centroids. Linformer [36] resorts to a low-rank projection on the length dimension. However, it can only operate in a bidirectional mode without casual masking.

Most of the aforementioned approaches rely on sparse attention patterns, while our method performs full contextual attention over the whole sequence. Besides, Linformer and Sinkhorn Transformer assume a xed sequence length due to the use of sorting network and projection, while our method poses no such constraint. Our method is also notably faster than the existing approaches, enjoying an asymptotic complexity of $O(n)$, while inner product can be stored in a table.

3 METHODOLOGY

In this section, we rst quickly go through some of the underlying preliminaries. Then we introduce our proposed method step by step, starting from a simple case. We propose two more variants for further e ciency improvement. Finally, we analysis the complexity of our method.

3.1 Preliminaries

3.1.1 Regular Self-A ention Mechanism. The vanilla dot-product attention, introduced in [33], accepts matrices $Q \bullet K \bullet V$ representing queries, keys and values, and computes the following outputs:

$$O = \mathrm{so}\text{‰}\mathrm{max}\left(\frac{QK^{)}}{\sqrt{d}}\right)V \tag{1}$$

In the self-attention setting, we let the input sequence attend to itself. Concretely, given an input sequence $X \in \mathbb{R}^{n \times d}$, we linearly project X via three matrices to get $Q = XW_Q \bullet K = XW_K$ and $V = XW_V$. The results are then computed using Eq. (1). This operation can be interpreted as computing a weighted average of the all other positions for every position in the sequence.

Self-attention has already been widely used in recommendation [17, 30, 40, 42]. Kang and McAuley [17] used self-attention along with the feed-forward network from [33] to encode user's sequential behaviors, and recommend the next item by computing the inner product between the encoded representation and target items' embeddings.

However, the computation of Eq. (1) su ers from quadratic computational and memory complexities, as computing the attention scores (the softmax term) and performing the weighted average both require $O(n^2)$ operations.

3.1.2 Embedding antization with Codebooks. Our e cient attention method is motivated by the idea of using codebooks to compress the embedding matrix [12, 16, 23]. LightRec, proposed in [23], encodes items with a set of codebooks, each containing c -dimensional codewords that serve as a basis of the latent space. An item's embedding E_i can be approximately encoded as:

$$E_i \approx \tilde{E}_i = \sum_{j=1}^{m} C_j^{k_{ij}} \bullet \text{s.t} k_{ij} = \arg\max_k \mathrm{sim}(E_i \bullet C_j^k) \tag{2}$$

where $\mathrm{sim}(\bullet\,\bullet)$ is a similarity metric between two vectors. C_j^k denotes the k-th codeword in the j-th codebook. In LightRec, a bilinear similarity function is adopted: $\mathrm{sim}(E_i \bullet C_j^k) = E_i^{)} W_1 C_j^k W_2$, where C_j^k denotes the k-th codeword in the j-th codebook. $W_1 \bullet W_2$ are learnable weights.

At training time, the codebooks and the item embeddings can be jointly trained using a softmax relaxation and the straight-through estimator [3]. At the inference stage, the item embedding E_i can be discarded completely. For each item, we only store its corresponding codeword indices in each codebook, i.e., $k_i = (k_{i1} \bullet k_{i2} \bullet \text{"""} \bullet k_{im})$. Because each codeword index can be encoded with $\log_2 c$ bits, the memory cost of storing n items is reduced from $32nd$ bits to $32md + nm \log_2 c$ bits, where the rst term is for codeword indices, and the second term is for codebooks.

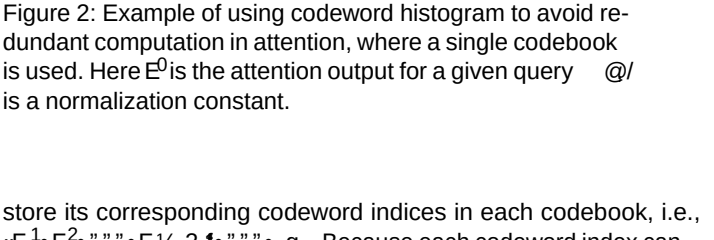

Figure 2: Example of using codeword histogram to avoid redundant computation in attention, where a single codebook is used. Here O is the attention output for a given query q, Z is a normalization constant.

3.2 Motivation: A Simple Case

To illustrate the motivation behind our proposed method, we rst look at a simple case where a single codebook is used to encode items.

In this case, an item is directly represented by the codeword with the maximum relevance score to it. That is, the i-th item in the sequence is therefore given by $E_i = C_{k_i}$, where $k_i = \arg\max_k \mathrm{sim}(E_i \bullet C_k)$ and $C_k \in \mathbb{R}^c$ denotes the k-th codeword in the codebook. Then, to perform the dot-product attention for a query $q \in \mathbb{R}^c$ (with keys and values being the sequence E), we compute the inner product between q and the corresponding codeword C_{k_i} for every item in the sequence. The output O of the attention is computed as follows:

$$O = \frac{\sum_{i=1}^{n} \exp(q C_{k_i}^{)}) C_{k_i}}{\sum_{i=1}^{n} \exp(q C_{k_i}^{)})} = \frac{\sum_{k=1}^{m} \exp(q C_k^{)}) C_k}{\sum_{k=1}^{m} \exp(q C_k^{)})} \tag{3}$$

where n is the sequence length. For the sake of simplicity, we omit the projection matrices $W_Q \bullet W_K \bullet W_V$ at this moment. From the above equation, we observe that we may have repeatedly compute the inner product of q with the same codeword C_k, since a number of items in the sequence may all share C_k as their representations. This redundant computation signi cantly hampers the e ciency, especially when $n \gg |\Omega|$, where Ω is the set of unique codeword indices that the n items in the sequence correspond to, i.e., $\Omega = \{k \mid \exists i : k = k_i\}$.

To address this issue, we note that O is just a weighted average of all the codewords in Ω, and the weight of each codeword depends only on its inner product with q and its number of occurrences. Therefore, we only need to count how many times each codeword C_k in Ω is used in the sequence, and compute the inner product of q with C_k once. The computation of Eq. (3) can be reformulated

as:

$$E^0 = \frac{\sum_{F \in \mathcal{F}} \#_F \exp(q \cdot e_F) e_F}{\sum_{F \in \mathcal{F}} \#_F \exp(q \cdot e_F)} \quad (4)$$

We illustrate this idea in Figure 2.

3.3 Linear-Time Self Attention

As we can see, the mechanism of codebook allows us to obtain the exact results of dot-product attention with less computation (both in computing the attention scores and computing the final weighted average), at least in the case of a single codebook. Now, we turn to the case that multiple codebooks are used. The items in the sequence are represented by an additive composition of codewords in all codebooks, as given by Eq. 2. The result of dot-product attention for a given query q is as follows:

$$E^0 = \frac{\sum_{i=1}^{n} \exp(q \cdot \sum_{b=1}^{B} e_{F_i^b}) \sum_{b=1}^{B} e_{F_i^b}}{\sum_{i=1}^{n} \exp(q \cdot \sum_{b=1}^{B} e_{F_i^b})} \quad (5)$$

Unlike the single codebook scenario, although in each codebook many items may correspond to the same codeword, their representations will diverge after the additive composition. Hence we still have to compute the inner product between q and every item G_i in the sequence.

To tackle this problem, we propose to relax the attention operation. We split the computation, perform the attention in each codebook separately, and then take the sum:

$$E^{00} = \sum_{b=1}^{B} \frac{\sum_{i=1}^{n} \exp(q \cdot e_{F_i^b}) e_{F_i^b}}{\sum_{i=1}^{n} \exp(q \cdot e_{F_i^b})} \quad (6)$$

This additive compositional formulation can be considered as a form of "multi-head" attention, where each attention head correlates with a codebook. Since different codebooks form different latent spaces, Eq. (6) in fact, aggregates information from different representational subspaces of the items, using independent attention weights.

Equipped with the above relaxation, we can once again reuse the inner product by computing the frequencies of each codeword that appeared in the sequence, separately for every codebook. We can reformulate the computation of Eq. (6) as follows:

$$E^{00} = \sum_{b=1}^{B} \frac{\sum_{F \in \mathcal{F}_b} \#_F^b \exp(q \cdot e_F) e_F}{\sum_{F \in \mathcal{F}_b} \#_F^b \exp(q \cdot e_F)} \quad (7)$$

where $\mathcal{F}_b = \{F \mid \exists i : F = F_i^b\}$ is the set of unique codeword indices of the b-th codebook, and $\#_F^b$ is the number of occurrences of e_F^b in the sequence.

However, the cardinality of $\mathcal{F}_b$ varies across different sequences and different codebooks. The computation of Eq. (7) therefore operates on different sizes of tensors, which is sub-optimal for efficient batching in GPU and TPU [19]. For batching purpose, we perform the attention over all codewords in each codebook, fixing the "context size" of the attention to n:

$$E^{00} = \sum_{b=1}^{B} \frac{\sum_{F=1}^{n} \#_F^b \exp(q \cdot e_F) e_F}{\sum_{F=1}^{n} \#_F^b \exp(q \cdot e_F)} \quad (8)$$

For a codeword e_F^b that is not used by any item in the sequence, the occurrence count $\#_F^b = 0$, e_F^b will not contribute to the weighted average. Hence Eq. (7) and Eq. (8) is equivalent.

Now we put it to the self-attention setting, where we use the input sequence as queries to attend to itself, the i-th query is just G_i, i.e., $q_i = G_i = \sum_{b=1}^{B} e_{F_i^b}^b$. Since we regard the attention in different codebook as independent heads that attend in different latent spaces, we further reduce the computation of the inner product from $q_i e_{F'}^b = \sum_{b=1}^{B} e_{F_i^b}^b e_{F'}^b$ to $e_{F_i^b}^b e_{F'}^b$, considering only the term in the same codebook. This gives us:

$$G_i^0 = \sum_{b=1}^{B} \frac{\sum_{F'=1}^{n} \#_{F'}^b \exp(e_{F_i^b}^b e_{F'}^b) e_{F'}^b}{\sum_{F'=1}^{n} \#_{F'}^b \exp(e_{F_i^b}^b e_{F'}^b)} \quad (9)$$

where G_i^0 is the i-th output of the attention operation.

Eq. (9) computes the bidirectional attention (each position can attend over all positions in the input sequence), since $\#_{F'}^b$ indicates the frequency of $e_{F'}^b$ in the entire sequence. However, in the recommendation setting, the model should consider only the first i items when making the $i+1$-th prediction [17], we therefore favor a unidirectional setting (each position can only attend to positions up to and including that position). This requires us to compute the codeword histogram of every codebook, up to i-th position, for each $i = 1, \ldots, n$. This can be implemented via the mechanism of prefix-sum. We first transform the codeword index F_i^b into a one-hot representation $H_i^b = \text{one-hot}(F_i^b)$, where $H_i^b \in \{0,1\}^{|\mathcal{F}_b|}$. The one-hot vectors H_i^b for each codebook b at each position i forms a tensor E of shape $n \times B \times |\mathcal{F}_b|$, we compute the prefix-sum along the first dimension to get the histograms up to each position in the sequence:

$$F_{i,b,:} = \sum_{j=1}^{i} E_{j,b,:} \quad (10)$$

There exists an efficient algorithm [7, 21] for prefix-sum with a computational complexity of $O(\log n)$ when compute in parallel.

As we mentioned earlier, in the vanilla self-attention [33], linear projections are applied on the input sequence to get queries, keys and values. Similarly, we can directly apply the projection matrices W_Q, W_K, W_V on the codebooks since every item in the input sequence is just a composition of codewords. Combining this with Eq. (10), we obtain the following unidirectional attention mechanism:

$$G_i^0 = \sum_{b=1}^{B} \frac{\sum_{F'=1}^{n} F_{i,b,F'} \exp(q_i W_Q W_K^\top e_{F_i^b}^b) e_{F'}^b W_V}{\sum_{F'=1}^{n} F_{i,b,F'} \exp(q_i W_Q W_K^\top e_{F_i^b}^b) e_{F'}^b W_V} \quad (11)$$

As we only need to compute the inner product between codewords in the same codebook, we can store them (after taking the exponent) in tables, and retrieve required terms via table lookup at inference time. We can achieve this by storing B tables with, m^2 items each, resulting in a memory cost of $O(Bm^2)$. However, this is not feasible without embedding quantization via codebooks, which leads to memory complexity of $O(B\#^2)$, where $\#$ is the number of items. We present the workflow of LISA in Figure 3, and we outline the main algorithm for LISA formally in Algorithm 1.

Figure 3: Work ow of LISA (unidirectional mode). Codeword indices of the items in the sequence are taken as input, they are used to index the inner product tables, and compute the codeword histograms for each codebook at every position. The histograms are element-wise multiplied with the inner product extracted from the table, and then normalized to obtain the attention scores, which are used to compute the weighted average of the projected codebooks. In this example, three codebooks are used, each contains four codewords.

Algorithm 1: LISA

Input : Codeword indices of the sequence $2 N^{!}$
Parameters: Inner product table (after taking the exponent) $M \; 2 \; R \; , \; , \;$, where $M_{1 \bullet 8 \bullet 9} = \exp \; {}^{1}\%^{\&} \; 2^{1o)}_{8} \; {}^{1}\% \; 2^{1o}_{9}$; projected codebooks for values $C^{+} = C\%^{+} \; 2 \; R \; ,$
Output : The results of self-attention $- \; {}^{0} \; 2 \; R^{!}$

Convert to one-hot representations $E = \text{one-hot}^{t} \; {}^{o} \; 2 \; f0 \bullet 1g^{!} \; , \; ;$
if unidirectional mode then
  Compute the pre x-sums $F^{!} \; , \;$ of E along the rst dimension according to Eq. (10);
else
  Compute the sums $F \; , \;$ of E along the rst dimension and broadcast F to shape $! \; , \; ;$
end
Gather inner products $S \; 2 \; R^{!} \; , \;$ from M along the rst two dimensions using indices ;
$A = F \; S$ (element-wise multiplication);
Normalize the attention scores along the last dimension;
$- \; {}^{0} = \sum_{1=1} A_{\bullet 1 \bullet} C^{+}_{1 \bullet \bullet} ;$
return X'

3.4 Variants

We notice that the computational cost of LISA is determined by the xed context size of , (i.e., the total number of codewords). To further increase the e ciency, especially on shorter sequences, we propose to use a separate set of codebooks to encode the sequence with a much smaller , . In our experiments, we nd that using a , of 128/256 is enough to obtain decent performance, compared to a , of 1024/2048 that we used in our base model. We investigate the following two variants:

- LISA-Soft: Instead of assigning a unique codeword F^{1}_{8} for each item 8, we allow a soft codeword assignment. In this case, E becomes the softmax scores where $E_{8 \bullet 1 \bullet} = \text{so‰max}^{1}{}_{1} G^{\bullet}_{8} \; {}^{1oo}$. With a soft assignment we can no longer compress the embedding matrix by storing discrete codeword indices at inference time. Hence we directly use the original embeddings for target items.
- LISA-Mini : To enable embedding compression, we still use a hard codeword assignment. We adopt two separate sets of codebooks: a smaller one (i.e., with a smaller ,) to encode the sequence, and a larger one to encode target items.

3.4.1 Extensions. Vanilla Transformer could stack multiple self-attention layers to improve performance. However, we nd that using multiple attention layers is not particularly helpful in recommendation, as with [17]. Therefore we only employ a single layer. Our method can easily extend to multiple layer cases. A straightforward solution is to use a di erent set of codebooks to remap the attention outputs to codewords in a di erent set of latent spaces. Our method can also be adapted beyond self-attention, as long as queries, keys and values can be encoded via codebooks. Besides recommendation, we can employ LISA in other domains since codebooks are able to quantize any embedding matrices. For example, the inputs in NLP tasks are just token embeddings, where our method can easily be applied.

3.5 Complexity Analysis

We see that computing the codeword histograms takes $\$ \; , \; {}^{o}$ steps, as we have to compute the pre x-sums along the sequence length dimension for every codeword in all codebooks. The time complexity for computing the nal outputs (weighted sum of values) is $\$ \; {}^{1}! \; , \; {}^{o}$, as this operation is essentially a batched matrix

Table 1: Dataset statistics.

Dataset	#users	#items	#ratings	avg. length
Alibaba	99,979	80,000	25M	252.93
ML-1M	6,040	3,416	1M	165.50
Video Games	59,766	33,487	0.5M	8.82
ML-25M	162,541	32,720	25M	153.47

multiplication between an attention score tensor of shape , and a tensor of shape , representing codebooks. Computing the inner product tables requires $ 1 , 2 o time, but at inference time, we can save this cost via table lookup. At training time, this is still a negligible term compared to $ 1 # ! , o, where # is the batch size. Hence, our method has an overall asymptotic time complexity of $ 1 ! , o.

4 EXPERIMENTS

In this section, we empirically analyze the recommendation performance of our proposed method, compared to the vanilla Transformer and existing e cient attention methods. Following that, we present the computational and memory costs of LISA with respect to di erent sequence lengths. We also investigate how the number of codewords a ects the performance of our method. Finally, we show the e ciency improvement brought by LISA in an online setting. We have also published our code[1].

4.1 Datasets

We use four real-world datasets for sequential recommendation that vary in platforms, domains and sparsity:

Alibaba : A dataset sampled from user click logs on Alibaba e-commerce platform, collected from September 2019 to September 2020. This is a dataset that contains relatively longer behavior sequences than the other datasets used in the experiments.
AmazonVideo Games [26]: A series of product reviews data crawled from Amazon spanning from 1996 to 2018. The data is split into separate datasets according to the top-level product categories. In this work, we consider the "Video Games" category. This dataset is notable for its sparsity.
MovieLens [15]: A widely used benchmark dataset of movie ratings for evaluating recommendation algorithms. We adopt two versions: MovieLens 1M (ML-1M) and MovieLens 25M (ML-25M), which include 1 million and 25 million ratings, respectively.

Following the common pre-processing practice in [17, 30, 31], we treat the presence of a rating as implicit feedback. Users and items with fewer than ve interactions are discarded. Table 1 shows the statistics of the processed dataset.

4.2 Compared Methods

We evaluate our proposed base model, denoted as LISA-Base, as well as its two variants: LISA-Soft and LISA-Mini. We compare these methods with the vanilla Transformer [33], as well as the following e cient attention methods:

Reformer [18]: It utilizes LSH to restrict queries to only attend to keys that fall in the same hash bucket, reducing the computational complexity to $ 1 ! log ! o. We do not use the reversible layers since this technique can be applied to all methods, including ours.
Sinkhorn Transformer [32]: It extends local attention by learning a di erentiable sorting of buckets. Queries can then attend to keys in the corresponding sorted bucket. This model has a computational complexity of $ 1 ! , 1 ! o 2 o, where is the bucket size.
Routing Transformer [27]: It is a clustering-based attention mechanism. K-means clustering is applied to input queries and keys. The attention context for a query is restricted to keys that got into the same cluster with the query. The computational complexity is $ 1 ! : , ! 2 o, where: is the number of clusters.
ImprovedClustered Attention [35]: Another clustering-based attention method. This approach, however, only groups queries into clusters, and attend cluster centroids over all keys. The top- : keys for each cluster centroid are extracted to compute the attention scores with queries in this cluster. This results in a computational complexity of $ 1 ! , ! : o, where is the number of clusters.
Linformer [36]: An e cient attention mechanism based on low-rank approximation. Linformer projects the keys and values of shape ! to : , e ectively reducing the context size to a tunable hyperparameter . This leads to a complexity of $ 1 ! : o. We note that it is the only baseline that does not support unidirectional attention.

For simplicity, we ignore the term regarding the latent dimension size in the above-mentioned asymptotic complexities.

4.3 Settings & Metrics

4.3.1 Parameter Se ings. We use the SASRec [17] architecture as the building block for our experiment setup, as SASRec purely relies on self-attention to perform sequential recommendation. Hence we can simply replace the regular Transformer self-attention with

Table 2: Settings for LISA and achieved compression ratio on each dataset (shown in the last four rows) . We present the settings of codebooks used to encode sequence. LISA-Mini also applies a separate set of codebooks to encode target items, with the same settings as LISA-Base.

	LISA-Base	LISA-Soft	LISA-Mini
#codebooks ()	8	8	8
#codewords, ( )	128 (ML-1M) 256 (Others)	16	32
Alibaba	24.26	-	18.45
ML-1M	3.19	-	2.51
Video Games	13.02	-	10.62
ML-25M	12.78	-	10.44

[1] Available at: https://github.com/libertyeagle/LISA

Table 3: Recommendation performance on Alibaba, ML-1M and Video Games. The number in the parentheses in baseline methods denotes the bucket size used for Sinkhorn Transformer and Routing Transformer, and #clusters for Clustered Attention. Bold font denotes the best-performing method among the e cient attention baselines, LISA-Soft and LISA-Mini.

	Alibaba				ML-1M				Video Games			
	HR@5	NDCG@5	HR@10	NDCG@10	HR@5	NDCG@5	HR@10	NDCG@10	HR@5	NDCG@5	HR@10	NDCG@10
Transformer	0.6597	0.5528	0.7569	0.5843	0.6841	0.5376	0.7914	0.5725	0.5525	0.4337	0.6583	0.4680
Linformer	0.3829	0.3007	0.4929	0.3360	0.4171	0.2899	0.5704	0.3394	0.4643	0.3605	0.5671	0.3937
Reformer (LSH-1)	0.6209	0.5189	0.7212	0.5513	0.6753	0.5248	0.7806	0.5590	0.5637	0.4429	0.6694	0.4771
Reformer (LSH-4)	0.6184	0.5156	0.7199	0.5484	0.6492	0.5040	0.7627	0.5408	0.5648	0.4446	0.6685	0.4781
Sinkhorn (32)	0.6298	0.5278	0.7260	0.5589	0.6743	0.5256	0.7796	0.5599	0.5479	0.4289	0.6557	0.4638
Sinkhorn (64)	0.6331	0.5319	0.7289	0.5629	0.6775	0.5310	0.7844	0.5656	0.5469	0.4258	0.6541	0.4605
Routing (32)	0.5742	0.4789	0.6724	0.5106	0.6623	0.5186	0.7704	0.5537	0.5615	0.4412	0.6657	0.4750
Routing (64)	0.6037	0.5037	0.7023	0.5356	0.6535	0.5100	0.7616	0.5452	0.5570	0.4369	0.6604	0.4704
Clustered (100)	0.5924	0.4937	0.6941	0.5266	0.6573	0.5127	0.7697	0.5492	0.5591	0.4394	0.6642	0.4734
Clustered (200)	0.5934	0.4936	0.6962	0.5268	0.6538	0.5095	0.7712	0.5478	0.5578	0.4384	0.6633	0.4725
LISA-Base	0.6660	0.5460	0.7702	0.5798	0.6940	0.5406	0.7962	0.5740	0.6203	0.4788	0.7338	0.5157
LISA-Soft	0.6575	0.5393	0.7622	0.5732	0.6795	0.5229	0.7887	0.5587	0.5951	0.4592	0.7035	0.4944
LISA-Mini	0.6430	0.5146	0.7559	0.5511	0.6853	0.5308	0.7886	0.5644	0.5917	0.4497	0.7102	0.4881

Table 4: Recommendation performance on ML-25M.

	HR@5	NDCG@5	HR@10	NDCG@10
Transformer	0.9338	0.8073	0.9752	0.8209
Linformer	0.8627	0.7086	0.9367	0.7329
Reformer (LSH-1)	0.9214	0.7847	0.9694	0.8005
Reformer (LSH-4)	0.9150	0.7765	0.9667	0.7935
Sinkhorn (32)	0.9195	0.7836	0.9682	0.7995
Sinkhorn (64)	0.9161	0.7820	0.9649	0.7980
Routing (32)	0.9167	0.7829	0.9658	0.7990
Routing (64)	0.9215	0.7890	0.9685	0.8044
Clustered (100)	0.9215	0.7830	0.9700	0.7989
Clustered (200)	0.9199	0.7818	0.9692	0.7980
LISA-Base	0.9254	0.7933	0.9713	0.8083
LISA-Soft	0.9269	0.7964	0.9710	0.8109
LISA-Mini	0.9243	0.7900	0.9701	0.8050

our method or the aforementioned baselines to compare the performance. We nd that the number of attention layers has negligible impacts on the recommendation performance, and the performance of using multiple attention heads is consistently worse than single head [17]. Multiple attention layers and attention heads only lead to greater computational cost. Hence we use a single layer and a single head for all compared methods.

All methods are implemented in PyTorch and trained with the Adam optimizer with a learning rate of 0.001 and a batch size of 128. We use an embedding dimension of 128, and the dropout rate is set to 0.1 on all datasets. We train all methods for a maximum of 200 epochs. Following the settings in the original papers, we consider two settings for Reformer: LSH-1 and LSH-4, which use one and four parallel hashes, respectively. For Sinkhorn Transformer and Routing Transformer, we consider a bucket (window) size of 32 and 64. We set the number of clusters to 100 and 200 for Clustered Attention. We use a low-rank projection size of 128 for Linformer. We apply casual masking for all methods except Linformer.

We report the settings of codebooks used for all three versions of our proposed method in Table 2. Since LISA-Base and LISA-Mini can simultaneously compress the embedding matrix, we also report the achieved compression ratios on all four datasets. We see that the item embeddings can be compressed up to 24x.

4.3.2 Metrics. Following [17, 24, 30], we apply two widely used metrics of ranking evaluation: Hit Rate (HR) and NDCG [38]. HR@ , counts the fraction of times that the target item is among the top- . NDCG@ , rewards methods that rank positive items in the rst few positions of the top- ranking list. We report the two metrics at : = 5 and : = 10. The last item of each user's behavior sequence is used for evaluation, while the remaining are used for training. For each user, We randomly generate 100 negative samples that the user has not interacted with, pairing them with the positive sample for the compared methods to rank.

4.4 Recommendation Performance

We report the results of the comparison of recommendation performance with baselines in Table 3 and Table 4. Since we also care about e ciency besides performance, we use bold font to denote the best-performing method among the e cient attention baselines and the two more e cient variants of our approach, excluding LISA-Base.

From the two tables, we have the following important ndings:

LISA-Base consistently outperforms all the state-of-the-art e cient attention baseline on all four datasets. It attains improvements of up to 8.78% and 7.29% over the best-performing baseline in terms of HR@10 and NDCG@10. This demonstrates the e ectiveness of our proposed attention method based on codeword histograms, as we compute the full contextual attention, compared to the sparse attention mechanism most baselines built upon. Since we use more codewords, LISA-Base also outperforms LISA-Soft and LISA-Mini on all datasets except ML-25M, where it has similar performance to LISA-Soft. On some metrics and datasets, LISA-Base even obtains higher performance than Transformer. This does make sense, considering LISA-Base could attend over a

Figure 4: Inference speed of di erent methods. Note that the y-axis is in a logarithmic scale. For Transformer the plot is shown up to 16K sequences, as longer sequences will produce an out of memory error on a 16GB V100 GPU.

broader context, encapsulating relevant information from a large number codewords in each codebook (providing diverse views). LISA-Soft and LISA-Mini achieve decent performance with much smaller codebooks. Even with 16 codewords used per codebook, LISA-Soft still outperforms the best-performing baseline by 2.46% and 1.16% in terms of HR@10 and NDCG@10, on average. Only on ML-1M, it is slightly worse than Sinkhorn Transformer (64) in terms of NDCG. We suppose that the issue might be we only use the soft codeword assignment scores when computing the codeword histograms F, we still use a unique codeword $\frac{1}{8}$ per codebook to approximate the query. Otherwise, it would pose challenges to handle the cross terms between di erent codebooks when computing the inner product. This could create a potential mismatch between queries and keys/values, leading to the performance gap on this dataset. However, in most cases, LISA-Soft achieves comparable performance with respect to LISA-Base, using 94% fewer codewords. Even when model compression is desired, LISA-Mini can still improve the best-performing baseline by 2.46% in terms of HR@10, on average.

Our proposed method, and the ones that allocate items to buckets based on similarity, even lead to increased recommendation performance on Video Games dataset. With an average length of only 8.8, the user sequences in Video Games tend to be noisy for making next-item recommendations. Full-context attention in this scenario would confuse the model with the noise. Reformer, Routing Transformer and Clustered Attention remedy this issue by only attending to the informative items selected through hashing or clustering (note that the number of buckets/clusters are predetermined according to the maximum sequence length in the dataset and the desired bucket/cluster size). Meanwhile, LISA addresses this issue by summarizing information from di erent codebooks, which can be reckoned as a way of denoising.

In general, sparse attention via sorting the buckets seems to be more e ective than learning the bucket assignments. We observe that Sinkhorn Transformer is a strong baseline, considerably outperforms Reformer, Routing Transformer and Clustered Attention on Alibaba and ML-1M, while has almost identical performance with them on ML-25M. Only on Video Games it performs slightly worse, due to the above-mentioned intrinsic noise in this dataset. In this instance, Sinkhorn Transformer will perform full contextual attention, as it divides the sequence into consecutive blocks of xed size.

LSH is better than clustering in bucket assignment. Reformer and Routing Transformer are both content-based sparse attention methods that di er mostly by the technique used to infer sparsity patterns. Reformer employs LSH while Routing Transformer resorts to online k-means clustering. We see that Reformer consistently outperforms Routing Transformer. The latter one sorts tokens by their distances to each cluster centroid and assigns membership via the top-k threshold. The centroids are updated by an exponential moving average of training examples. Unlike Reformer, this approach does not guarantee that each item belongs to a single cluster, which may partially contribute to Routing Transformer's worse performance.

Unidirectional attention is vital for satisfactory performance in recommendation. Observing the results of Linformer we can obtain this conclusion. Because the projection is applied to the length dimension, causing the mixing of sequence information, it is non-trivial to apply casual masking for Linformer. This bidirectional attention leads to signi cant performance degradation, as our attempts with other methods in bidirectional mode corroborate this nding. The designs of certain baseline methods also induce some issues in order to enforce casual masking. For example, in the unidirectional mode, Sinkhorn Transformer sorts the buckets only according to the rst token in each bucket. Bidirectional Clustered Attention could rst approximate the full contextual attention scores with that of the cluster centroid each query belongs to, while separately computing on the top- keys. However, this technique is not viable in a unidirectional setting.

Using a larger bucket size does not necessarily improve the performance. We observe this phenomenon from the results of Sinkhorn Transformer and Routing Transformer. While the bucket size is increased, hence the context size for each query, we use fewer buckets/clusters. This would make it harder for k-means clustering and Sinkhorn sorting to group relevant items together. Hence, one has to carefully tune the bucket size to achieve ideal

performance, as it balances between the size of attention context and the quality of sorting / bucket assignment. Surprisingly, we nd using multiple rounds of hashing in Reformer does not enhance the performance either.

4.5 Computational Cost

4.5.1 Se ings. To evaluate the computational e ciency of our proposed method, we compare the inference speed of our method with the vanilla Transformer and the aforementioned e cient attention baselines. Following [18, 36], we use synthetic inputs with varying lengths from 128 to 64K, and perform a full forward pass. The batch size is scaled inversely with the sequence length, to keep the total number of items (tokens) xed. We report the average running time on 100 batches. For each baseline model, we only consider the less time-consuming variant. For example, we only report the LSH-1 variant for Reformer, as the LSH-4 version is far more computationally intensive. Since the asymptotic complexity of our proposed method is $O(n)$, the inference speed of all the three versions of LISA only depends on the total number of codewords used to encode sequences (i.e.,). We evaluate two settings of LISA that use a total of 128 and 256 codewords (denoted as LISA-128 and LISA-256), corresponding to the settings we used for LISA-Soft and LISA-Mini in Section 4.4. We only measure the cost of self-attention, since other components are the same for all compared models. We consider latent dimension sizes of 128 and 1024. All the experiments are conducted on a single Tesla V100 GPU with 16GB memory. The results are shown in Figure 4.

4.5.2 Findings.

Our method consistently and dramatically outperforms Transformer and all e cient attention baselines in inference speed. Only on sequences of length 128 and using an embedding size of 128, LISA is slightly slower than Transformer. When = 128, LISA-128 is 3.1x faster than Reformer on 64K sequences. Bene ting from using inner product tables, our method is even way faster than others when = 1024, achieving a speed boost of 57x compared to Transformer on 16K sequences. All other methods take considerably longer time as the cost of computing the inner product dominates in this scenario. Linformer has an almost identical speed to LISA-128 when = 128. However, its recommendation performance is notably worse than ours. From Figure 4, we also verify the linear complexity of LISA, as the inference time remains constant when the total number of items in a batch is constant.

Sinkhorn Transformer and Routing Transformer still su er from enormous computational cost with growing sequence length. Especially when = 128, the inference time increases by 5x for Sinkhorn and 27x for Routing moving from sequences of 128 to 64K. Both the two methods require $O(n^2)$ time to compute query/key dot product within each bucket, where is the bucket size. Sinkhorn Transformer takes $O(n^2/)$ time to sort buckets, while Routing Transformer spends $O(n^2)$ time to perform cluster assignments. With the bucket size xed, the cost of sorting/clustering becomes dominant. Increasing the bucket size, on the other hand, we would have to pay an extra price in computing attention scores within each bucket.

Table 5: Memory e ciency of di erent methods. The numbers in the table are the ratios between the peak memory usage of the Transformer and that of the compared e cient attention method. Bold font denotes the most memory e cient one.

	sequence length					
	512	1024	2048	4096	8192	16384
Linformer	2.46x	4.48x	8.49x	16.51x	32.65x	65.53x
Reformer	0.66x	1.16x	2.15x	4.16x	8.31x	11.26x
Sinkhorn	0.97x	1.70x	3.15x	6.09x	12.14x	25.74x
Routing	1.27x	2.22x	4.08x	7.73x	14.87x	29.45x
Clustered	2.32x	4.17x	7.85x	15.26x	30.63x	64.91x
LISA-128	2.94x	5.14x	9.55x	18.45x	36.86x	78.26x
LISA-256	1.50x	2.62x	4.87x	9.40x	18.78x	39.93x

Though the extra overhead dominates when sequences are short, Reformer tends to be almost linear when facing longer sequences. We see that hashing items into buckets via LSH is exceptionally time-consuming. When = 128, Reformer is signi cantly slower than the vanilla Transformer on sequences shorter than 512, even slower than on 64K sequences due to larger batch size. Our method, on the contrary, does not su er from this issue, being up to 6.5x faster than Reformer on sequences of 128. On longer sequences, Reformer scales almost linearly, since the length is quite small in its asymptotic complexity $O(n \log n)$.

Clustered Attention fails to demonstrate its advantage of linear complexity even on sequences of 64K. From Figure 4, we observe that the Clustered Attention is indeed linear (although bears the same extra overhead problem handling short sequences as Reformer). It seems that there underlies a substantial computational cost by computing full-contextual attention using the cluster centroids, and then improving the approximation for each query on the top- keys. Clustered Attention is still slower than Reformer on 64K sequences.

4.6 Memory Consumption

4.6.1 Se ings. We also evaluate the memory e ciency of di erent methods by measuring the peak GPU memory usage. The settings of the compared methods are the same as the previous section's. The latent dimensionality is set to 128. For a given sequence length, we choose the batch size to be the maximum that all compared models can t in memory. We report results on sequences up to 16K long, as the vanilla Transformer could not t longer sequences even with a batch size of 1. The compression ratios with respect to Transformer are shown in Table 5.

4.6.2 Findings. All the e cient attention baselines greatly reduce the memory consumption on longer sequences. Among which LISA-128 is the most e cient one, requiring only 1.3% of the memory needed by Transformer in the best case. Although Reformer enjoys faster inference speed on long sequences, we see that it is more memory-hungry than other baselines. This again re ects the LSH bucketing overhead of Reformer.

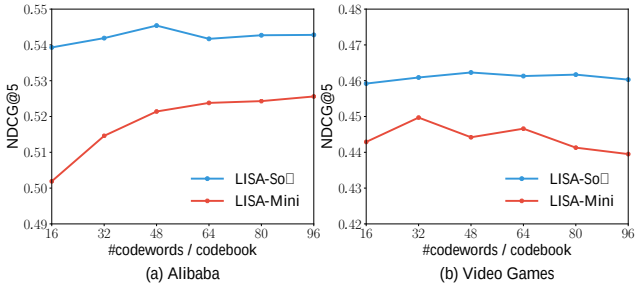

Figure 5: The impact of the number of codewords.

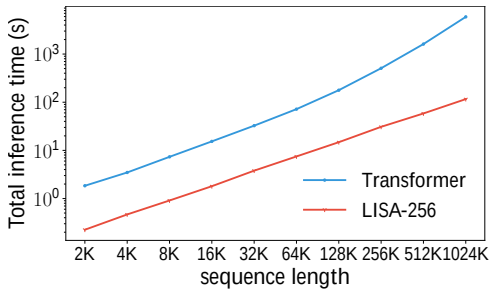

Figure 6: Inference speed of Transformer and LISA in an interactive setting.

4.7 Sensitivity w.r.t. Number of Codewords

4.7.1 Settings. We investigate the impact of the number of codewords per codebook (i.e., W) used in LISA-Soft and LISA-Mini on recommendation performance. We keep the number of codebooks to be 8 and vary W from 16 to 96. We leave the settings of the codebooks used to encode target items in LISA-Mini unchanged. We show the results on Alibaba and Video Games in Figure 5.

4.7.2 Findings. The performance of LISA-Mini on Alibaba consistently improves with the increasing number of codewords used. Due to the sparsity of Video Games, it is challenging to learn two large codebooks well simultaneously. Hence the performance drops a bit when using a large number of codewords on this dataset. On the other hand, the performance of LISA-soft is relatively stable w.r.t. W on both datasets, indicating that we can attain desirable performance with only a small number of codewords, greatly boost the inference efficiency.

4.8 Improving Efficiency for Online Recommendation

Here we consider a practical setting that users and the recommender interact in a dynamic manner. The recommender makes recommendations based on the user's historical behaviors. The user then interacts with the recommendations, and the response is appended to the user's history. This process is repeated as the recommender makes new recommendations using the updated user sequence.

A particular advantage of our method emerges in this setting. In our method, the computation of the attention scores only depends on the codeword histogram and the codebooks themselves. For each user, instead of having to store his entire history sequence at the cost of $O(L)$, we can just save the codeword histogram and the last item's codeword indices to represent the user's state. The codeword histogram and the indices can be dynamically updated, resulting in a constant storage cost of $O(BW)$. At each inference step, our method can utilize the stored histogram to compute a weighted average of codebooks in a constant time of $O(BWD)$, compared with the $O(LD)$ complexity for the vanilla self-attention.

We simulate this scenario with randomly generate data. The total time required to make stepwise inferences from scratch up to some length L is measured. Since most efficient attention baselines face challenges when dealing with variable sequence length (recall that Sinkhorn Transformer and Linformer assume a fixed sequence length as their model parameters depend on this length), we only compare LISA-256 with the Transformer.

We show the results in Figure 6. We see that our method is considerably faster than Transformer in this setting, especially at a larger number of steps. Concretely, our method takes about 0.11ms to progress a step, no matter how long the sequence it. However, it would take Transformer 0.98ms to compute attention for a single query when the sequence is at 2K length, 1.50ms at 64K, and 11.01ms at 1024K, ~100x slower than our method.

Table 6: Performance of LISA-Base using codebooks pretrained with the vanilla Transformer.

	HR@5	NDCG@5	HR@10	NDCG@10
Alibaba	0.6697	0.5492	0.7711	0.5821
ML-1M	0.7002	0.5456	0.7945	0.5763
Video Games	0.6188	0.4800	0.7333	0.5172
ML-25M	0.9287	0.7991	0.9725	0.8135

4.9 Migrating Codebooks from Vanilla Self-Attention

4.9.1 Settings. We note that the codebooks serve as a plug and play module, which can be used to replace any embedding matrix. We can also train the model based on vanilla self-attention with codebooks. The pretrained codebooks are directly applied to LISA-Base and are frozen. We evaluate the performance of this model and report the results in Table 6.

4.9.2 Findings. We find that directly use codebooks trained with regular dot-product attention does not cause performance degradation, but actually improves the performance of the LISA-Base model a little. This implies that our method indeed can approximate dot-product attention to some extent.

5 CONCLUSIONS AND FUTURE WORKS

In this paper, we propose LISA, an efficient attention mechanism for recommendation, built upon embedding quantization with codebooks. In LISA, codeword histograms for each codebook are computed over the input sequences. We then use the histograms and the inner product between codewords to compute the attention weights, in time linear in the sequence length. Our method performs on par with the vanilla Transformer in terms of recommendation