Positional encoding is not the same as context: A study on positional encoding for Sequential recommendation

Anonymous Submission
Under Review for The Web Conference 2025

ABSTRACT

The rapid growth of streaming media and e-commerce has driven advancements in recommendation systems, particularly Sequential Recommendation Systems (SRS). These systems employ users' interaction histories to predict future preferences. While recent research has focused on architectural innovations like transformer blocks and feature extraction, positional encodings, crucial for capturing temporal patterns, have received less attention. These encodings are often conflated with contextual, such as the temporal footprint, which previous works tend to treat as interchangeable with positional information. This paper highlights the critical distinction between temporal footprint and positional encodings, demonstrating that the latter offers unique relational cues between items, which the temporal footprint alone cannot provide. Through extensive experimentation on eight Amazon datasets and subsets, we assess the impact of various encodings on performance metrics and training stability. We introduce new positional encodings and investigate integration strategies that improve both metrics and stability, surpassing state-of-the-art results at the time of this work's initial preprint. Importantly, we demonstrate that selecting the appropriate encoding is not only key to better performance but also essential for building robust, reliable SRS models.

ACM Reference Format:
Anonymous Submission. 2025. Positional encoding is not the same as context: A study on positional encoding for Sequential recommendation. In *Proceedings of The Web Conference 2025 (WWW '25).* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/1122445.112245

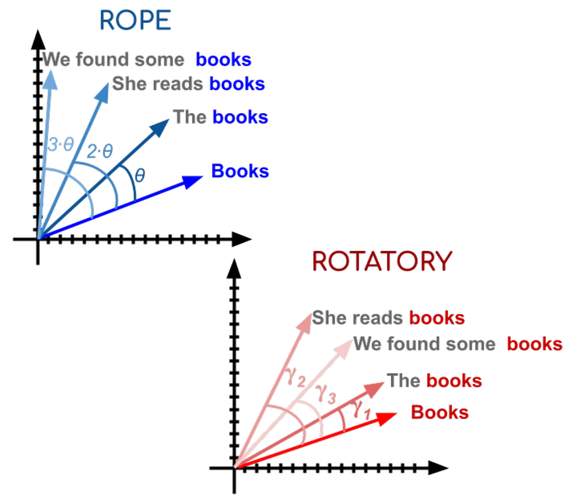

Figure 1: Illustration of our proposed ***Rotatory*** embeddings. Similar to ***RoPE*** , it encodes positional information through the angles of the embeddings. However, unlike *RoPE* , which assigns angles proportionally based on position, *Rotatory* learns these angles during training. It allows the model to determine the most relevant positional information and adjust their significance accordingly.

1 INTRODUCTION

Sequential Recommendation Systems (SRS) have become integral to personalizing user experiences by predicting the next item of interest based on historical interaction sequences. These systems are widely deployed across e-commerce, social media, and streaming platforms and significantly enhance user engagement and satisfaction. As defined in [6], SRS records user-item interactions sequentially, closely related to session-based and session-aware recommendations, which focus specifically on interactions within a single session.

Capturing the order of data is essential in sequential systems. Initially, Deep Learning techniques utilized Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to model sequential dependencies. However, **attentional-based models** have gained prominence due to their superior effectiveness, leading to numerous **self-attention** approaches [31]. Most state-of-the-art (SOTA) models in SRS now leverage attentional-based networks. A critical component of these systems is **positional encoding (PE)**, which imparts information about the order of items within a sequence. The choice and implementation of PEs are crucial for optimizing performance and stability.

Despite the widespread adoption of various PE techniques, there is a noticeable gap in research concerning the stability of model performance across different runs. However, many existing publications focus on enhancing performance metrics without adequately addressing the underlying stability issues introduced by different encoding strategies. Other studies underestimate the relevance of positional encoding, considering them part of the context.

To bridge this gap, we aim to investigate the stability problem in SRS models through positional encodings. We conduct extensive experiments to evaluate how different positional encodings influence these models' performance and stability. Building on this foundation, we introduce a novel positional encoding method termed ***Rotatory*** , along with its concatenated variant, ***Rotatory + Con*** , and ablations of the previous encoding method.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW '25, April 28 – May 02, 2025, Sydney, Australia
© 2025 ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/1122445.112245

To summarise, our contribution is fourfold:

(1) We analyze existing work and highlight that more than the standard usage of 5 runs is needed to ensure the reliability of results.
(2) We conduct a detailed analysis of various encoding types, their features (such as in-head and affected heads), and perform ablation studies to identify factors that improve stability and performance.
(3) We introduce a novel rotatory-based encoding method, alongside integration techniques, such as concatenation, which have not been explored previously. These contributions aim to improve both performance metrics and training stability.
(4) Our findings show that encoding effectiveness is primarily influenced by dataset deviation, rather than sparsity or data type. This insight, combined with our analysis, offers a clearer guide for using encodings in SRS with transformer architectures.

2 RELATED WORK

2.1 Architectures & Encoding in SRS

From an architectural perspective, SRS have significantly evolved over the past two decades. Early approaches employed K-Nearest Neighbors (KNN) for item-based recommendations [3, 20] and Markov Chains for modeling sequential interactions [9–11]. To address interaction sparsity, matrix factorization methods like DeepFM [7] and CFM [33] were introduced, enhancing the ability to handle sparse data effectively.

With the advent of Deep Learning, RNNs became popular for capturing sequence information and item positions, exemplified by models such as GRU4Rec [13] and DREAM [34]. Concurrently, CNNs were utilized to capture local features and temporal information, as seen in Caesar [19] and NextItNet [35], among others [27, 29]. **Attention mechanisms** were first introduced in NARM [18], and transformers with self-attention were pioneered in SRS by AttRec [36].

Different types of models are used to predict user behavior: **attribute-aware** [24], **context-aware** [38], and **time-aware** [17]. These models focus on item information, user interaction characteristics, and the timing of interactions, respectively. The integration of transformers has introduced **positional embeddings**. Some approaches, like CARCA [24], use time-aware information to replace traditional positional encodings, assuming that the order of items can be deduced from timestamps. However, recent NLP research [1] has shown that incorporating positional information directly within the attention mechanism, rather than relying on timestamps, yields better results.

One of the initial models to capture sequential information was GRU4Rec [13], which extracts data from item sequences. The context-aware DeepFM [7] combines factorization methods with deep learning to leverage contextual information. Similarly, SASRec [15] employs a self-attention layer to balance short-term intent and long-term preferences, using transformer encoders to compare past behavior with target items via dot-products. This encoder-decoder architecture, which extracts past information while synthesizing target embeddings, led to the development of CARCA [24].

We focus on positional encoding across various datasets used in these studies. Each work approaches encoding differently: AttRec [36] employs absolute embeddings to incorporate time information into the query and key, while SASRec utilizes absolute learnable position embeddings in the input sequence. CARCA experimented with absolute encodings but observed worse performance compared to using no positional encoding. Recent advancements, such as EulerFormer [28], introduce complex vector attention to integrate semantic and positional information. This variety in encoding strategies highlights the need for a systematic analysis, which we provide in this work.

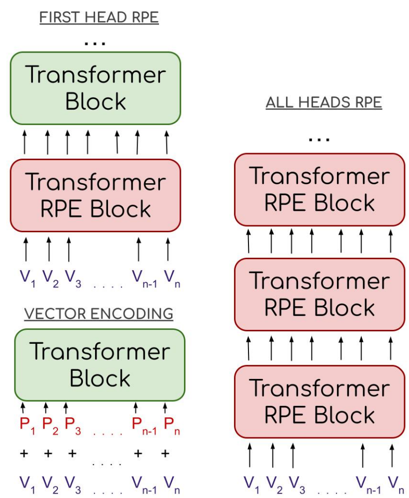

Figure 2: (1) Vector encoding added before Transformer blocks, (2) First head RPE: Relative encoding applied only to the first block (*RopeOne*), and (3) All head RPE: Relative encoding integrated into every block (*RMHA-4*).

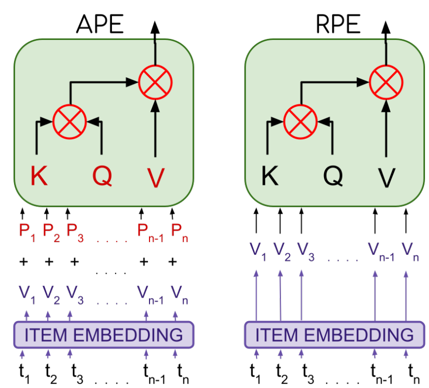

Figure 3: *APE* vs *RPE* : While *APE* encodings introduce the positional information as vectors before the V, Q and K. *RPE* add this information at the coefficient level to K and Q. Parts affected by the position information appear in red.

Encoding	Learnable	Type	Range	Layers	Concatenation	Integration
Abs	No	*APE*	Fix.	Vector Encoding	No	Add
Abs + Con	Yes	*APE*	Fix.	Vector Encoding	Yes	Joint
Rotatory	Yes	*APE*	Unl.	Vector Encoding	No	Add
Rotatory + Con	Yes	*APE*	Unl.	Vector Encoding	Yes	Joint
RMHA-4	No	*RPE*	4	All	No	Multi.
RoPE	No	*APE*	Unl.	All	No	Multi.
RopeOne	No	*APE*	Unl.	First	No	Multi.
Learnt	Yes	*APE*	Fix.	Vector Encoding	No	Add
Learnt + Con	Yes	*APE*	Fix.	Vector Encoding	Yes	Joint
None	No	N/A	N/A	0	N/A	N/A

Table 1: Encoding types: 'Type' specifies absolute or relative. 'Learnable' indicates if encoding parameters are trained, including concatenation for learnable encodings. 'Range' denotes position coverage: 'Fix.' for fixed length, 'Unl.' for unlimited, and 4 for relative range. 'Layers' refers to encoding application: 'Vector Encoding' adds vectors at the start (left in Fig 3), 'All' applies to all transformer blocks, and 'First' to the first block. 'Concatenation' shows if concatenation is used. 'Integration' describes the method: 'Add' for vector sum, 'Joint' for concatenation, and 'Multi.' for matrix multiplication in heads or element-wise otherwise. '*None* ' means no encoding.

2.2 Positional encodings

Positional encoding enables transformer-based models to incorporate item order within sequences. Encoding methods can be categorized into absolute and relative encodings [37] (Figure 3). In Table 1, we have a detailed classification of the main encodings in this paper.

***Absolute Positional Encoding** (APE).* They assign a unique position vector to each item based on its position in the sequence. These encodings are added before the first Transformer block, providing absolute positional information. Models like SASRec [15] utilize learnable absolute position embeddings, allowing the model to learn position representations during training. There are two common variations of absolute positional encodings: fixed (*Abs*) and learnable (*Learnt*).

***Absolute Positional Encoding** (Abs).* Absolute positional encoding adds fixed values to item embeddings based on their absolute position in the sequence. The most common approach uses sinusoidal functions, as seen in BERT [5]. This encoding is added at the input layer as part of the item encoding (Figure, left 3).

The encoding for an item at position *pos* in a sequence is calculated using sine and cosine functions:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{(2i/d)}}\right) \tag{1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{(2i/d)}}\right) \tag{2}$$

where i is the dimension, and d is the embedding dimension. This encoding is added to the original embedding:

$$x = x + PE_{(pos,*)} \tag{3}$$

***Learnable Encoding** (Learnt).* Learnable position embeddings treat positional information as trainable parameters. Unlike fixed encodings, the model learns optimal position representations based on the data context, denoted as **learnable encoding** (*Learnt*). Positional encodings are added to the input sequence embeddings.

***Relative Positional Encoding** (RPE).* Relative Positional Encoding (Figure 3, right) encodes the relative distances between items instead of their absolute positions, enabling models to better capture contextual relationships within user interactions. Unlike Absolute Positional Encoding (*APE*), which requires fixed input lengths and provides positional information from the start to the end of the sequence, *RPE* offers flexibility by focusing on the relationships between items regardless of their absolute positions. A downside of these positional encoding mechanisms is that they are significantly slower during training and inference since they add an extra step in the self-attention layer [23].

***Self-attention**.* This mechanism allows models to weigh different parts of the input sequence differently, focusing on relevant information during processing. Introduced in [30], the self-attention mechanism computes attention weights based on the similarity between queries (Q), keys (K), and values (V):

$$\alpha_{ij} = \frac{\exp\left((Q_i K_j^T)/\sqrt{d}\right)}{\sum_{t=1}^{n} \exp\left((Q_i K_t^T)/\sqrt{d}\right)} \tag{4}$$

$$\text{Attention}(Q,K,V) = \sum_{j=1}^{N} \alpha_{ij} V_j \tag{5}$$

***Relative Positional Encoding Implementation** (RMHA-4).* This encoding addresses the limitations of *APE* by encoding the relative positions within the attention mechanism. Models like NARM [18] and TransformerXL [2] adopt *RPE* to enhance the modeling of contextual dependencies. In *RPE* , positional information is incorporated into the key and value projections, allowing the model to focus on the relationships between items dynamically.

The attention weight with *RPE* is computed as:

$$\alpha_{ij} = \frac{\exp\left((Q_i (K_j + a_{ij}^K)^T)/\sqrt{d}\right)}{\sum_{t=1}^{n} \exp\left((Q_i (K_t + a_{it}^K)^T)/\sqrt{d}\right)} \tag{6}$$

Additionally, positional information can be added to the value projections:

$$\text{Attention}(Q,K,V) = \sum_{j=1}^{N} \alpha_{ij}(V_j + a_{ij}^V) \tag{7}$$

We will consider the case which bounds the relative indexes to a distance of 4 and called it ***RMHA-4*** , Relative Multi-Head Attention with distance 4.

***RoPE** (RoPE).* Rotary Positional Encoding [26] encodes positional information by applying rotational transformations to the Query and Key matrices. This preserves the dot product between vectors, maintaining effective attention mechanisms. *RoPE* is particularly useful for complex item orders, providing a more flexible representation than traditional encodings.

For a two-dimensional vector $\mathbf{x}$ and angle θ:

$$f_\theta(\mathbf{x}) = \mathbf{R}(\theta)\mathbf{W}\mathbf{x} \quad (8)$$

where $\mathbf{R}(\theta)$ is a rotation matrix defined by sines and cosines, and $\mathbf{W}$ represents either K or Q. Higher-dimensional vectors are split into two-dimensional chunks, with rotations applied to each. To reduce computational costs, RoFormer [26] implements these rotations using two vector multiplications and one vector addition.

3 METHODOLOGY

In this section, we present our contributions to positional encodings by introducing variants of existing methods and new encodings.

3.1 Encoding Variants

The most commonly used encodings are *Abs* encodings, which are vector encodings added to the initial item vector by addition. This approach preserves the original embedding dimension without requiring additional encoding dimensions (Table 1). In contrast, non-vector encodings, such as Relative Positional Encoding (RPE) and Rotary Positional Encoding (RoPE), are typically introduced as multiplications within the attention mechanism. These general rules apply depending on whether encodings are relative or absolute.

To further explore encoding strategies, we introduce an encoding variant that employs concatenation with a linear layer and activation function to revert to the original embedding dimension. To the best of our knowledge, this approach has not been previously utilized in SRS and has seen limited application in NLP. While concatenation benefits include separating semantic and positional information, it introduces a size increase in the architecture, which is particularly critical for Large Language Models. To mitigate this, we incorporate an additional layer, allowing for a gradual introduction of positional information without excessively increasing model complexity.

***Concatenated Absolute Encoding**.* To enhance positional information, we introduce **concatenated absolute encoding**, which concatenates the positional encoding to the item embedding:

$$x = \text{Concat}(x, PE_{(pos,*)}) \quad (9)$$

A Linear layer is then applied to restore the original embedding dimension without increasing model parameters. This method explicitly incorporates positional information, allowing the model to gradually rely on positional cues and achieving more stable performance (Section 5).

***Concatenated Learnable Encoding** (Learnt + Con).* Additionally, we apply a concatenation approach to learnable encodings, denoted as **concatenated learnable encoding** (*Learnt* + *Con*). However, *Learnt* + *Con* generally yields worse results compared to the classic residual connection (Section 5), likely because concatenation adds complexity in this case, making it harder for the model to effectively learn embeddings. Furthermore, the gradual introduction of positional information is already inherent in the learnable encoding mechanism.

We do not apply concatenated versions for *RoPE* and *RMHA-4* , as they are relative encodings.

***RoPE One Block** (RopeOne).* In our experiments (Section 5) we found that *RMHA-4* produce more stable results than other encodings. To investigate whether the stability improvements of *RMHA-4* were due to introducing positional information at each Transformer block or its role as an *RPE* -based head, we tested *RoPE* . The results showed that that *RoPE* was in a middle point between relative and absolute encodings regarding stability.

We hypothesize that this intermediate stability is due to the application of *RoPE* across all Transformer blocks ((3) in Figure 2). To investigate this further, we introduced ***RopeOne*** , which applies *RoPE* exclusively in the first Transformer block ((2) in Figure 2). As detailed in Section 5, this modification yields slightly improved results. This implies that the rotary aspect, rather than merely being relative, is key to the observed improvements and leads us to the new encoding introduced below.

3.2 Rotatory Encoding

We propose a new encoding method termed *Rotatory* (Figure 1), with its concatenated variant referred to as *Rotatory* + *Con* This approach applies a rotation to define the position of each item. Unlike traditional methods that add positional information during self-attention, our method incorporates these rotations at the initial item embeddings, similar to sinusoidal encoding (*Abs*). Vector embeddings offer the advantage of faster training and inference [23]. Therefore, we adopt a formulation similar to *RoPE* , applying the rotations at the outset. We enhance flexibility by making the rotation angles learnable, allowing the model to capture both absolute and relative positional relationships effectively.

The positional encoding for an item at position pos and embedding dimension i is calculated using a trigonometric approach, modified to include alternating sine signs for a rotational effect. The encoding is defined as follows:

$$PE_{(pos,2i)} = (-1)^i \sin\left(\theta_{(pos,i)}\right) \quad (10)$$

$$PE_{(pos,2i+1)} = \cos\left(\theta_{(pos,i)}\right) \quad (11)$$

Here, pos is the position of the item in the sequence, i represents the embedding dimension, and $\theta_{(pos,i)}$ is a learnable angle defined as:

$$\theta_{(pos,i)} = \frac{E_{pos}}{10000^{(2i/d)}} \cdot 2\pi, \quad \forall pos \in [0, L), \forall i \in [0, H) \quad (12)$$

where $E_{pos} \in \mathbb{R}^{L\times H}$ are learnable embeddings. The term $(-1)^i$ in the sine function alternates the sign, introducing the rotation effect in the encoding. These positional encodings are then added to the original item embeddings by addition:

$$x'_{(pos)} = x_{(pos)} + PE_{(pos)} \quad (13)$$

where $x_{(pos)}$ is the original semantic embedding of the item at position pos, and $PE_{(pos)}$ is the learnable ***Rotatory*** encoding. In the case of ***Rotatory + Con***, the positional encoding is concatenated with the original item embedding and passed through a learned linear transformation for further adaptation. This learnability introduces flexibility, enabling the model to capture both absolute and relative positional information in a dynamic manner, which leads to improved performance in handling complex sequence relationships.

4 EXPERIMENTAL SETUP

4.1 Datasets

We selected several Amazon datasets to evaluate performance of various encoding types. For this purpose, we utilised four distinct real-world datasets extracted from product reviews on Amazon.com. These datasets are widely used for SRS under the leave-one-out protocol [12, 14, 16, 25, 32, 38]. Since multiple versions of these datasets exist, we opted for the preprocessed versions provided by [24].

Dataset	Users	Items	Interactions	Attributes
Men	34,244	110,636	254,870	2,048
Fashion	45,184	166,270	358,003	2,048
Games	31,013	23,715	287,107	506
Beauty	52,204	57,289	394,908	6,507

Table 2: Dataset Statistics.

Dataset	*Avg Dev Hit*	*Avg Dev NDCG*	CI-length	Density
Men	3.9143	8.2742	5.6686	$6.727 \cdot 10^{-5}$
Fashion	4.87	9.7814	9.2257	$4.765 \cdot 10^{-5}$
Games	1.124	0.952	1.616	$3.904 \cdot 10^{-4}$
Beauty	0.935	0.87	1.22	$1.320 \cdot 10^{-4}$

Table 3: Deviations among the different datasets without Encoding. This pattern is generally preserved along the different encodings. The density refers to the inverse of Sparsity.

Covering various product categories, these datasets offer a broad source for training and evaluating recommendation models. They include essential features for each interaction, such as user IDs, item IDs, timestamps, and additional vector-based contextual information. As shown in Tables 2 and 3, they exhibit diversity in terms of sparsity, defined as the ratio of missing user-item interactions. This will be a crucial aspect of our analysis, along with the discreteness of the features.

1.Beauty. [16, 32, 38] The dataset includes discrete and categorical attributes for all beauty products, including fine-grained categories and brands. Mainly categorical and discrete features.

2.Video Games. [16, 32] The Video Games sub-dataset includes user interactions, reviews, and product details specific to the video game category, such as price, brand and categorical features. Most of the attributes here are discrete and categorical.

3.Men. [12] The men's dataset encompasses a comprehensive collection of items falling under men's clothing. The attributes are dense vectors from image-based features extracted from the last layer of a ResNet50 [4] on the ImageNet dataset [8].

4.Fashion. [12, 14] It contains six categories for men's and women's clothing. The features were extracted as dense using the same ResNet50 approach.

4.2 Models and Training

To implement CARCA, we converted the original TensorFlow model into PyTorch[1]. We used the loss implementation from [21]. The original TensorFlow version built upon prior models within the same framework. During our evaluation, we observed that the implementations checked metrics every 20 epochs, operating under the assumption that the optimal performance point would be reached later. This approach was likely adopted due to the extensive number of epochs, exceeding 1000, which increases the computational cost of evaluations. To achieve more reliable evaluations, we implemented a logarithmic measurement of the metrics, which resulted in slightly improved outcomes in certain cases.

As in previous works [15, 24], we use an ADAM optimiser to minimise the binary cross-entropy loss of the CARCA model while masking the padded items to prevent them from contributing to the loss function. For a given sequence of items interactions for a user u as $\{i_1^u, \ldots, i_N^u\}$, we create an input list as $I^{u+} = \{i_1^u, \ldots, i_{N-1}^u\}$ by deleting the last item of the list, a positive target list as $T^{u+} = \{i_2^u, \ldots, i_N^u\}$ by shifting the input list by one, and a negative target list as $T^{u-} = \{i^{rand_1}, \ldots, i^{rand_{N-1}}\}$ generated by random negative items. Then the loss is given by:

$$-\sum_{u \in U} \sum_{r \in T^{u+} \cup T^{u-}} \left[Y_r \log(\hat{Y}_r) + (1 - Y_r) \log(1 - \hat{Y}_r) \right]$$

Here, U represents the set of users, $T^{u+} \cup T^{u-}$ denotes the union of positive and negative items for user u, Y_r is the observed interaction label for item r, and $\hat{Y}_r$ is the predicted probability of interaction with item r.

Encoding	*Avg Dev Hit*	*Avg Dev NDCG*	runs	CI-length
Abs	5.43	8.25	9.38	6.71
Abs + Con	3.52	5.56	7.12	5.35
Learnt	4.79	7.64	7.88	7.05
Learnt + Con	3.17	6.36	6.62	4.42
None	2.35	5.26	8.00	3.29
RMHA-4	0.51	1.08	6.00	0.88
RoPE	2.38	5.45	5.00	4.04
Rotatory	2.46	5.01	8.38	3.48
Rotatory + Con	2.48	5.18	6.75	3.49
RopeOne	2.82	6.54	4.29	5.46
RoPE-Longer	3.33	8.15	5.25	5.99
Rotatory-Longer	1.05	1.04	4.67	2.08
Rotatory + Con-Longer	0.99	1.24	6.00	1.59
RMHA-4-Longer	0.71	4.00	6.00	1.14

Table 4: Deviations among the different encodings for the default upper bound values (0.0001 for all except NaN for Games). Relative encoding in all the heads, *RMHA-4*, presents more stability, followed by our *Rotatory* versions.

[1] https://github.com/researcher1741/Position_encoding_SRS

Dataset	Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*
Beauty	silu	*Rotatory-Longer*	0.0001	61.87	1.04	42.60	0.98
Beauty	silu	*Rotatory*	0.0001	61.72	1.14	42.54	1.52
Beauty	silu	*Rotatory + Con*	0.0001	61.68	1.04	42.31	0.97
Beauty	leaky	*Rotatory + Con*	0.0001	61.16	0.77	42.83	0.48
Men	silu	*RMHA-4-Longer*	0.0001	70.13	0.42	46.41	4.18
Men	silu	*RMHA-4*	0.0001	69.70	0.64	43.75	1.44
Men	leaky	*None*	0.1000	69.69	1.28	57.94	1.69
Men	leaky	*RMHA-4-Longer*	0.0001	68.72	1.47	43.46	0.79
Fashion	silu	*RMHA-4*	0.0001	77.26	0.24	49.75	2.33
Fashion	silu	*RMHA-4-Longer*	0.0001	77.00	0.48	49.40	0.54
Fashion	leaky	*RMHA-4*	0.0001	76.46	0.25	49.49	2.18
Fashion	leaky	*RMHA-4-Longer*	0.0001	76.20	0.51	49.85	3.92
Games	leaky	*Rotatory + Con*	NaN	80.62	0.55	56.07	0.79
Games	silu	*Rotatory + Con-Longer*	NaN	80.29	0.47	55.18	0.55
Games	leaky	*Rotatory + Con-Longer*	NaN	80.21	1.51	55.06	1.93
Games	silu	*Learnt*	NaN	79.82	1.88	54.30	2.95

Table 5: The table displays the top 4 results for each dataset on cases with a standard deviation for *Hit Mean*below 3.

Dataset	Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*
Beauty	leaky	*Abs + Con*	0.0001	67.93	4.17	48.71	3.59
Beauty	silu	*Learnt*	0.0001	67.62	6.95	46.33	7.08
Beauty	silu	*Abs*	0.0001	67.27	10.59	45.34	10.75
Beauty	silu	*Abs + Con*	0.0001	65.87	5.23	45.61	5.12
Men	leaky	*Learnt + Con*	0.1000	73.86	7.18	58.89	8.15
Men	leaky	*Learnt*	0.0001	72.08	4.44	56.55	9.98
Men	leaky	*Learnt + Con*	0.0001	71.34	7.13	57.92	13.07
Men	leaky	*Rotatory + Con*	0.1000	70.92	3.92	59.84	4.88
Fashion	silu	*RMHA-4*	0.0001	77.26	0.24	49.75	2.33
Fashion	silu	*RMHA-4-Longer*	0.0001	77.00	0.48	49.40	0.54
Fashion	leaky	*RMHA-4*	0.0001	76.46	0.25	49.49	2.18
Fashion	leaky	*RMHA-4-Longer*	0.0001	76.20	0.51	49.85	3.92
Games	leaky	*Rotatory + Con*	NaN	80.62	0.55	56.07	0.79
Games	silu	*Rotatory + Con-Longer*	NaN	80.29	0.47	55.18	0.55
Games	leaky	*Rotatory + Con-Longer*	NaN	80.21	1.51	55.06	1.93
Games	silu	*Learnt*	NaN	79.82	1.88	54.30	2.95

Table 6: The table displays the top 4 results for each dataset on cases with a standard deviation for *Hit Mean*below 12.

4.3 Metrics

We employ two widely used metrics for Next-Item Recommendation and SRS: Hit@10 and Normalised Discounted Cumulative Gain (NDCG).

Hit@10: This metric measures the fraction of times the ground truth next item appears within the top 10 recommended items. For simplicity, we refer to Hit@10 as *Hit*, its mean as *Hit Mean*, its devisation as *Hit Dev* and the average of the deviations as *Avg Dev Hit*.

NDCG: Normalised Discounted Cumulative Gain evaluates the ranking quality by considering both the relevance of items and their positions in the recommended list. It is defined as:

$$NDCG = \frac{DCG}{IDCG} \tag{14}$$

where *DCG* (Discounted Cumulative Gain) is calculated as the sum of the relevance scores of items at each position in the ranked list, discounted by their position:

$$DCG = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{15}$$

and *IDCG* (Ideal Discounted Cumulative Gain) represents the maximum possible *DCG* for a perfectly ranked list:

$$IDCG = \sum_{i=1}^{n} \frac{2^{max(rel_i,0)} - 1}{\log_2(i+1)} \tag{16}$$

Here, rel_i denotes the relevance score of the item at position i in the ranking. We will denote its mean as *NDCG Mean*, its deviation as *NDCG Dev* and the average of the deviations as *Avg Dev NDCG*.

5 RESULTS

We present comprehensive results for all datasets and encodings in Appendix A.1. Due to space limitations, only selected sub-tables are included in this section.

In addition to positional encoding, we analyze other factors such as maximum values and activation functions that may influence model stability. In CARCA [24], an upper bound of 0.0001 (*nmax* in our Tables) for the encoding vectors was identified as optimal across all datasets except for Games. To investigate whether the optimal upper bound varies with different encodings, we experimented with maximum values of 0.1 and 0.0001. Generally, an upper bound of 0.0001 yielded better results, with a few exceptions. Furthermore, we compared activation functions by replacing *leakyrelu* with *silu*. The *silu* activation function resulted in slight improvements in stability; however, the correlation was not strong.

5.1 Stability

A common practice in the literature is to present the average results of 5 runs with different random seeds. However, our preliminar experiments revealed significant variability between individual runs. To better understand this inconsistency, we analyzed the standard deviation across different trainings. Detailed deviation and mean values are provided in all the training tables. Additionally, we computed the 95% confidence intervals for these metrics and included them in the complete results tables in Appendix A.1.

Inter-Dataset Deviation: The stability of results varies across different datasets. Beauty is the most stable, showing an average Hit deviation of 0.9 and an average NDCG deviation of 0.87, while Fashion has the highest deviations. Datasets with lower sparsity, such as Beauty and Games, generally exhibit greater stability. In contrast, datasets with higher sparsity, like Men and Fashion, show reduced stability. Additionally, there is a correlation with data type: lower-sparsity datasets often use discrete embedding representations. As discussed in Section 5.3, this stability pattern persists even when we adjust sparsity levels. By reducing datasets to achieve comparable sparsity, we show that both encoding selection and stability challenges remain for each data type.

Inter-Encoding Deviation: Tables 4 and 19 (Appendix) reveal that some encodings yield more stable results than others. In particular, the concatenated versions of *Abs* , *Learnt* , and *Rotatory* demonstrate greater stability (Table 19). Under the most stable scenario with an upper bound of 0.0001, as shown in Table 4, *Abs + Con* and *Learnt + Con* remain significantly more stable. However, *Rotatory + Con* shows no significant difference here, presenting deviations similar to *None* , likely due to the need to learn the angles.

The most notable encoding for stability is *RMHA-4*, which, combined with the 0.0001 upper bound, produces the most stable results, with an *Avg Dev Hit* of 0.51 compared to 2.36 for the no-encoding case. This pattern persists even with higher upper bounds, suggesting that *RMHA-4* offers excellent stability, which can also be observed in the loss graph, as detailed in Appendix A.6. As we will discuss in Section 5.3, this encoding is particularly worth considering in unstable scenarios.

Encoding Ablation: To assess whether the stability is due to *RMHA-4* 's positional information at each transformer block or its *RPE* -based nature, we tested *RoPE* , an *APE*. The results rank *RoPE* between *RMHA-4* and *Learnt* /*Abs*, showing no significant change in stability compared to *None* (Table 4). We hypothesized that this lack of change in stability could be due to *RoPE* 's application across all transformer blocks. To test this, we introduced *RopeOne* , which applies *RoPE* only in the first block. Results in Table 19 show that *RopeOne* slightly outperforms *RoPE*, while Table 4 shows similar performance with a slight lower stability. This suggests that positional information at each transformer block does not significantly impact stability. Finally, we introduced *Rotatory* , which incorporates rotatory properties in a vector encoding. The results showed similar stability to *RoPE* and its ablation *RopeOne* . This led us to the conclusion that the rotatory nature of some encodings and the relative encoding nature of *RMHA-4* contribute to the stability of the experiments.

5.2 Best Results

Which is the best encoding? The optimal encoding depends on the maximum deviation considered. For instance, examining the Beauty dataset in Table 9, we observe lower deviations. However, in cases like *Learnt* with *nmax* = 0.1 and leakyrelu, despite achieving 74 in *Hit Mean*, the *Hit Dev* increases to 17.19. This highlights the trade-off between performance and stability, as some encodings may offer better performance metrics at the cost of higher instability. These findings support the conclusions of [22], which suggest that, in certain scenarios, selecting a robust set of random seeds can be more important than choosing the best encoding method.

Maximum deviation of 12*:* When setting the maximum *Hit Dev* to 12, we get Table 6. In the CARCA [24], an ablation analysis of the Men dataset revealed that using *Abs* results in poorer performance than *None*. Similar results were observed for the Men in our experiments, where both cases showed a deviation of around 6, suggesting that the observed effects were more due to initialization than encoding choice. With a maximum of 12 on *Hit Dev*, Men and Beauty achieve their best performance with *Learnt*-like and *Abs*-like encodings, respectively, although both are unstable. In contrast, Fashion and Games reach their optimal results with *RMHA-4* and *Rotatory*, respectively, both of which are stable.

Maximum deviation of 3*:* Reducing the maximum *Hit Dev* to 3 yields optimal results, as presented in Table 5. In this Table, the best encodings are *RMHA-4* and *Rotatory* (or its concatenated version), which correlate strongly with sparsity, datatype, and stability (low-deviation). As discussed in Section 5.3, datasets with high-deviation without encoding benefit from the stability provided by *RMHA-4* , while low-deviation datasets like Beauty and Video Games perform best with the purely rotatory approach of *Rotatory* . Even among stable datasets, these results surpass the previous SOTA benchmarks (CARCA) at the time. We also observe a significant improvement over the *None* case for high-deviation datasets, with gains exceeding 5%, while achieving greater stability.

Impact of Extended Training on Encoding Stability: Ablation studies were conducted on the encoding techniques to investigate the source of stability. Our initial experiments used hyperparameters similar to those of the original CARCA model. Upon reviewing the loss metrics, we found that several top-performing models did not

Dataset	Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*
Submen	leaky	*RMHA-4*	0.1000	77.28	2.66	68.45	3.65
Submen	leaky	*RMHA-4*	0.0001	75.96	2.71	66.03	4.00
Subfashion	leaky	*RMHA-4*	0.0001	79.72	2.85	71.57	3.66
Subfashion	leaky	*None*	0.1000	32.94	1.89	16.77	3.00
Subgames	leaky	*Rotatory + Con*	NaN	52.23	1.57	31.51	0.96
Subgames	silu	*Rotatory + Con*	NaN	50.73	0.89	30.48	0.72
Subgames	leaky	*RMHA-4*	NaN	50.43	0.95	29.96	0.87
Subgames	leaky	*None*	NaN	49.53	1.01	28.84	0.69
Submen3	leaky	*Rotatory + Con*	0.1000	73.12	2.38	62.86	2.99
Submen3	silu	*Rotatory + Con*	0.0001	72.77	2.96	62.56	4.31

Table 7: The table displays the top 4 results for each ablation dataset on cases with a *Hit Dev*below 3. In the case of the Fashion dataset ablation we did not get any training below 3 in the deviation.

converge, as both loss and performance metrics continued to improve. To address this, we extended the training by an additional 400 epochs beyond the default values specified in Table 18. Despite this extension, the improvements in model performance were minimal, indicating that the optimizer may have been oscillating around a local minimum.

5.3 Sparsity and Deviation Reduction

As seen in Section 5.2, two encodings, *RMHA-4* and *Rotatory + Con* , consistently provide the best results (with maximum *Hit Dev* of 3). There appears to be a correlation between the deviation of datasets and the choice of optimal encoding. For high-deviation datasets like Men and Fashion, *RMHA-4* yields better results, whereas for low-deviation datasets like Games and Beauty, *Rotatory + Con* performs best. Importantly, both Men and Fashion share two other key characteristics: dense vector representations and higher-sparsity. This makes it difficult to attribute the performance solely to the default dataset deviation.

Sparsity Reduction: To investigate this further, We conducted a sparsity reduction experiment by creating subdatasets with densities comparable to other datasets. Specifically, we reduced the sparsity of Fashion from $4.765 \cdot 10^{-5}$ to $3.564 \cdot 10^{-4}$ and Men from $6.727 \cdot 10^{-5}$ to $2.860 \cdot 10^{-4}$, resulting in Subfashion and Submen. We applied *RMHA-4* , *Rotatory + Con* , and *None* to these datasets.

Table 7 shows that *RMHA-4* consistently produced the best results across both subdatasets, confirming that its effectiveness is not solely due to sparsity. In Subfashion, *Hit Dev*increased to 15.105, and *NDCG Dev*to 17.935, while Submen experienced a small reduction in deviations (*Hit Dev*7.296, *NDCG Dev*9.096). Although Men saw a slight reduction on deviation (Table 8), it wasn't substantial.

Additionally, we created Subgames for comparison, reducing the Games dataset's sparsity to see if it influences performance. Interestingly, *Rotatory + Con* remained the best encoding for this more stable dataset (Table 7), suggesting that even after sparsity reduction, dataset characteristics like deviation may drive encoding performance.

Deviation Reduction: To rule out the correlation with the vector representation and isolate the effect of deviation, we further reduced the Men dataset to $10k$ users and $80k$ items (10000 users and 45129 items after filtering). This reduced sparsity to $1.648 \cdot 10^{-4}$ and helped stabilize the experiments, decreasing *Hit Dev*to 5.198 and *NDCG Dev*to 6.826. The reduced deviation was enough to make *Rotatory + Con* the best-performing encoding. Although the performance is close to that of the *None* case (Table 17), there was a shift in the optimal encoding.

These findings indicate that high-deviation datasets, even after sparsity reduction, benefit from the added stability of *RMHA-4* . In contrast, low-deviation datasets like Subgames, perform better with *Rotatory + Con* .

Dataset	Users	Items	Interac.	*Hit Dev*	*NDCG Dev*	CI-length	Density
Subfashion	3,840	5,000	16,080	15.105	17.935	24.175	$3.564 \cdot 10^{-4}$
Submen	8,332	10,000	28,603	7.2925	9.0925	10.515	$2.860 \cdot 10^{-4}$
Subgames	5,000	4,601	33,353	1.115	0.775	1.79	$1.3341 \cdot 10^{-3}$
Submen 2	10,000	45,129	74,391	5.1975	6.8225	8.31	$1.648 \cdot 10^{-4}$

Table 8: Statistics of the ablations: Deviations for Hit@10 and NDCG without Encoding. Together with the average length of the *CI* intervals, and the density.

6 CONCLUSION

In this study, we emphasize that encodings are not mere technical details but rather critical components that can significantly influence both the performance and stability of SRS. Our findings reveal that high-deviation datasets benefit from relative encodings, such as *RMHA-4* , which not only stabilize training but also yield optimal results. These encodings play a vital role in mitigating performance fluctuations, allowing the model to consistently perform well across different runs. On the other hand, for low-deviation datasets, our proposed *Rotatory* encoding offers a robust alternative that further improves results without introducing unnecessary complexity.

Importantly, at the time of publication, we achieved state-of-the-art performance using only these encoding methods, surpassing previous benchmarks and demonstrating greater reliability. Our results indicate that encodings are a pivotal factor in ensuring the stability of transformer-based models, especially in recommendation systems. We encourage the research community to place greater emphasis on positional encodings, as they can drastically enhance model effectiveness, particularly in tasks involving sequential data.

In the Appendix A.7, there is a decision tree for the encoding election.

REFERENCES

[1] Pu-Chin Chen, Henry Tsai, Srinadh Bhojanapalli, Hyung Won Chung, Yin-Wen Chang, and Chun-Sung Ferng. 2021. A simple and effective positional encoding for transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2974–2988, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

[2] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

[3] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 293–296, New York, NY, USA. Association for Computing Machinery.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[6] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. 2020. Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations. *ACM Trans. Inf. Syst.*, 39(1).

[7] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 1725–1731. AAAI Press.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[9] Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. 2016. Vista: A visually, socially, and temporally-aware model for artistic recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16. ACM.

[10] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 161–169, New York, NY, USA. Association for Computing Machinery.

[11] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 191–200.

[12] Ruining He and Julian McAuley. 2016. Vbpr: visual bayesian personalized ranking from implicit feedback. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 144–150. AAAI Press.

[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks.

[14] Min Hou, Le Wu, Enhong Chen, Zhi Li, Vincent W. Zheng, and Qi Liu. 2019. Explainable fashion recommendation: A semantic attribute region guided approach. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4681–4688. International Joint Conferences on Artificial Intelligence Organization.

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206.

[16] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation.

[17] Jiacheng Li, Yujie Wang, and Julian McAuley. 2020. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 322–330, New York, NY, USA. Association for Computing Machinery.

[18] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1419–1428, New York, NY, USA. Association for Computing Machinery.

[19] Lei Li, Li Chen, and Ruihai Dong. 2021. Caesar: context-aware explanation based on supervised attention for service recommendations. *Journal of Intelligent Information Systems*, 57(1):147–170. Publisher Copyright: © 2020, Springer Science+Business Media, LLC, part of Springer Nature. Copyright: Copyright 2020 Elsevier B.V., All rights reserved.

[20] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

[21] Rastislav Papso. 2023. Complementary product recommendation for long-tail products. In *Proceedings of the 17th ACM Conference on Recommender Systems*, RecSys '23, page 1305–1311, New York, NY, USA. Association for Computing Machinery.

[22] David Picard. 2021. Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision.

[23] Ofir Press, Noah A. Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *ICLR*.

[24] Ahmed Rashed, Shereen Elsayed, and Lars Schmidt-Thieme. 2022. Context and attribute-aware sequential recommendation via cross-attention. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 71–80, New York, NY, USA. Association for Computing Machinery.

[25] Harald Steck. 2019. Embarrassingly shallow autoencoders for sparse data. *CoRR*, abs/1905.03375.

[26] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

[27] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. *CoRR*, abs/1809.07426.

[28] Zhen Tian, Wayne Xin Zhao, Changwang Zhang, Xin Zhao, Zhongrui Ma, and Ji-Rong Wen. 2024. Eulerformer: Sequential user behavior modeling with complex vector attention. *ACM SIGIR Forum*.

[29] Trinh Xuan Tuan and Tu Minh Phuong. 2017. 3d convolutional networks for session-based recommendation with content features. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 138–146, New York, NY, USA. Association for Computing Machinery.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

[31] Shoujin Wang, Yan Wang, Quan Sheng, Mehmet Orgun, Longbing Cao, and Defu Lian. 2021. A survey on session-based recommender systems. *ACM Computing Surveys*, 2021:39.

[32] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. 2020. Sse-pt: Sequential recommendation via personalized transformer. In *Proceedings of the 14th ACM Conference on Recommender Systems*, RecSys '20, page 328–337, New York, NY, USA. Association for Computing Machinery.

[33] Xin Xin, Bo Chen, Xiangnan He, Dong Wang, Yue Ding, and Joemon Jose. 2019. Cfm: Convolutional factorization machines for context-aware recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3926–3932. International Joint Conferences on Artificial Intelligence Organization.

[34] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A dynamic recurrent model for next basket recommendation. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 729–732, New York, NY, USA. Association for Computing Machinery.

[35] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. 2018. A simple convolutional generative network for next item recommendation. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*.

[36] Shuai Zhang, Yi Tay, Lina Yao, Aixin Sun, and Jake An. 2018. Next item recommendation with self-attentive metric learning.

[37] Liang Zhao, Xiaocheng Feng, Xiachong Feng, Dongliang Xu, Qing Yang, Hongtao Liu, Bing Qin, and Ting Liu. 2024. Length extrapolation of transformers: A survey from the perspective of positional encoding.

[38] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 1893–1902, New York, NY, USA. Association for Computing Machinery.

A APPENDIX

A.1 Experiments

In this section, we present all our results for the four datasets, thereby completing the missing experiments from the sub-tables in the main corpus. In tables 9, 10, and 11, we can find *RoPE-Longer* , which is not presented in Table 1. These are just a few cases in which we extend the number of epochs since we found that for *RoPE* , it takes longer to reach its maximum.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	0.0001	56.00	0.96	37.93	0.65	9	(55.37, 56.63)	1.26
silu	*None*	0.0001	57.03	0.83	38.42	0.68	9	(56.49, 57.57)	1.08
leaky	*None*	0.1000	49.02	1.18	30.73	1.21	9	(48.25, 49.79)	1.54
silu	*None*	0.1000	49.69	0.77	31.35	0.94	9	(49.19, 50.19)	1.00
leaky	*Learnt + Con*	0.0001	56.36	0.65	37.94	1.54	3	(55.62, 57.1)	1.48
silu	*Learnt + Con*	0.0001	56.69	0.51	37.82	0.65	3	(56.11, 57.27)	1.16
leaky	*Learnt + Con*	0.1000	49.29	0.71	30.98	0.99	3	(48.49, 50.09)	1.60
silu	*Learnt + Con*	0.1000	49.85	1.18	31.42	1.48	3	(48.51, 51.19)	2.68
leaky	*Abs + Con*	0.0001	67.93	4.17	48.71	3.59	5	(64.27, 71.59)	7.32
silu	*Abs + Con*	0.0001	65.87	5.23	45.61	5.12	5	(61.29, 70.45)	9.16
leaky	*Abs + Con*	0.1000	51.42	1.41	32.10	0.46	3	(49.82, 53.02)	3.20
silu	*Abs + Con*	0.1000	54.54	3.33	35.43	3.34	5	(51.62, 57.46)	5.84
leaky	*Rotatory + Con*	0.0001	61.16	0.77	42.83	0.48	3	(60.29, 62.03)	1.74
silu	*Rotatory + Con*	0.0001	61.68	1.04	42.31	0.97	3	(60.5, 62.86)	2.36
leaky	*Rotatory + Con*	0.1000	50.51	1.59	32.25	1.76	3	(48.71, 52.31)	3.60
silu	*Rotatory + Con*	0.1000	50.88	1.00	32.70	0.87	3	(49.75, 52.01)	2.26
leaky	*Learnt*	0.0001	59.83	3.71	39.85	3.11	3	(55.63, 64.03)	8.40
silu	*Learnt*	0.0001	67.62	6.95	46.33	7.08	5	(61.53, 73.71)	12.18
leaky	*Learnt*	0.1000	74.49	17.19	51.36	22.65	6	(60.74, 88.24)	27.50
silu	*Learnt*	0.1000	61.75	7.84	40.02	7.34	10	(56.89, 66.61)	9.72
silu	*Rotatory-Longer*	0.0001	61.87	1.04	42.60	0.98	2	(60.43, 63.31)	2.88
silu	*Rotatory-Longer*	0.1000	49.33	2.08	30.89	2.28	2	(46.45, 52.21)	5.76
leaky	*Abs*	0.0001	63.65	8.36	43.68	8.36	13	(59.11, 68.19)	9.08
silu	*Abs*	0.0001	67.27	10.59	45.34	10.75	15	(61.91, 72.63)	10.72
leaky	*Abs*	0.1000	57.25	10.84	37.65	10.73	12	(51.12, 63.38)	12.26
silu	*Abs*	0.1000	61.23	15.24	39.14	11.78	16	(53.76, 68.7)	14.94
leaky	*RMHA-4*	0.0001	56.29	0.81	37.66	1.14	3	(55.37, 57.21)	1.84
silu	*RMHA-4*	0.0001	57.00	0.35	37.22	0.32	3	(56.6, 57.4)	0.80
leaky	*RMHA-4*	0.1000	51.30	0.32	33.38	0.15	3	(50.94, 51.66)	0.72
silu	*RMHA-4*	0.1000	51.35	0.42	33.04	0.12	3	(50.87, 51.83)	0.96
leaky	*RopeOne*	0.0001	55.33	1.22	37.89	1.24	3	(53.95, 56.71)	2.76
silu	*RopeOne*	0.0001	57.25	1.16	38.76	1.01	3	(55.94, 58.56)	2.62
leaky	*RopeOne*	0.0001	55.95	0.21	38.41	0.65	3	(55.71, 56.19)	0.48
silu	*RopeOne*	0.0001	56.23	0.63	37.69	0.55	3	(55.52, 56.94)	1.42
leaky	*RoPE*	0.0001	56.30	0.92	38.60	0.82	3	(55.26, 57.34)	2.08
silu	*RoPE*	0.0001	56.59	0.72	37.21	0.98	3	(55.78, 57.4)	1.62
leaky	*RoPE*	0.1000	50.04	0.80	31.91	0.89	3	(49.13, 50.95)	1.82
silu	*RoPE*	0.1000	49.15	0.71	31.31	1.01	3	(48.35, 49.95)	1.60
leaky	*Rotatory*	0.0001	58.69	1.31	40.83	0.71	5	(57.54, 59.84)	2.30
silu	*Rotatory*	0.0001	61.72	1.14	42.54	1.52	5	(60.72, 62.72)	2.00
leaky	*Rotatory*	0.1000	49.11	1.17	30.57	1.28	5	(48.08, 50.14)	2.06
silu	*Rotatory*	0.1000	49.40	0.96	31.14	1.04	5	(48.56, 50.24)	1.68

Table 9: Beauty results with different positional encodings. Original results from [24] at the end.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	0.0001	65.44	3.54	46.46	9.99	9	(63.13, 67.75)	4.62
silu	*None*	0.0001	70.38	1.82	53.34	7.00	6	(68.92, 71.84)	2.92
leaky	*None*	0.1000	54.04	7.99	38.82	9.53	3	(45.0, 63.08)	18.08
silu	*None*	0.1000	61.53	11.84	48.13	14.97	3	(48.13, 74.93)	26.80
leaky	*Learnt + Con*	0.0001	69.31	7.74	50.73	12.02	14	(65.26, 73.36)	8.10
silu	*Learnt + Con*	0.0001	69.81	2.03	49.49	7.59	7	(68.31, 71.31)	3.00
leaky	*Learnt + Con*	0.1000	65.29	7.61	52.91	9.70	10	(60.57, 70.01)	9.44
silu	*Learnt + Con*	0.1000	66.15	7.47	50.59	9.46	14	(62.24, 70.06)	7.82
leaky	*Abs + Con*	0.0001	67.30	4.48	52.26	9.94	10	(64.52, 70.08)	5.56
silu	*Abs + Con*	0.0001	70.83	1.64	59.93	1.95	6	(69.52, 72.14)	2.62
leaky	*Abs + Con*	0.1000	64.93	10.58	52.21	13.11	16	(59.75, 70.11)	10.36
silu	*Abs + Con*	0.1000	65.14	8.20	51.71	11.06	13	(60.68, 69.6)	8.92
leaky	*Rotatory + Con*	0.0001	69.13	2.80	55.33	7.69	9	(67.3, 70.96)	3.66
silu	*Rotatory + Con*	0.0001	71.16	1.38	48.61	8.62	7	(70.14, 72.18)	2.04
leaky	*Rotatory + Con*	0.1000	68.61	2.76	57.08	3.72	7	(66.57, 70.65)	4.08
silu	*Rotatory + Con*	0.1000	65.23	7.76	52.99	9.31	11	(60.64, 69.82)	9.18
leaky	*Learnt*	0.0001	68.37	5.75	51.41	12.43	13	(65.24, 71.5)	6.26
silu	*Learnt*	0.0001	67.63	5.75	46.36	11.95	13	(64.5, 70.76)	6.26
leaky	*Learnt*	0.1000	64.99	9.00	47.66	14.48	16	(60.58, 69.4)	8.82
silu	*Learnt*	0.1000	63.94	9.49	46.44	10.58	16	(59.29, 68.59)	9.30
leaky	*RMHA-4-Longer*	0.0001	76.20	0.51	49.85	3.92	6	(75.79, 76.61)	0.82
silu	*RMHA-4-Longer*	0.0001	77.00	0.48	49.40	0.54	6	(76.62, 77.38)	0.76
leaky	*RoPE-Longer*	0.0001	68.63	3.97	51.05	11.71	5	(65.15, 72.11)	6.96
silu	*RoPE-Longer*	0.0001	71.50	1.22	49.69	6.41	7	(70.6, 72.4)	1.80
leaky	*Abs*	0.0001	66.39	4.61	51.99	9.51	10	(63.53, 69.25)	5.72
silu	*Abs*	0.0001	69.52	4.76	49.72	10.39	10	(66.57, 72.47)	5.90
leaky	*Abs*	0.1000	59.75	9.74	44.73	12.86	15	(54.82, 64.68)	9.86
silu	*Abs*	0.1000	58.39	8.48	41.86	11.31	15	(54.1, 62.68)	8.58
leaky	*RMHA-4*	0.0001	76.46	0.25	49.49	2.18	6	(76.26, 76.66)	0.40
silu	*RMHA-4*	0.0001	77.26	0.24	49.75	2.33	6	(77.07, 77.45)	0.38
leaky	*RMHA-4*	0.1000	60.63	5.14	40.75	8.06	11	(57.59, 63.67)	6.08
silu	*RMHA-4*	0.1000	63.61	4.41	39.48	7.97	11	(61.0, 66.22)	5.22
leaky	*RopeOne*	0.0001	73.03	6.52	62.96	8.59	3	(65.65, 80.41)	14.76
silu	*RopeOne*	0.0001	71.70	1.82	51.07	11.17	3	(69.64, 73.76)	4.12
leaky	*RopeOne*	0.0001	69.27	5.02	54.24	11.80	6	(65.25, 73.29)	8.04
silu	*RopeOne*	0.0001	70.49	1.66	49.17	10.70	6	(69.16, 71.82)	2.66
leaky	*RoPE*	0.0001	66.73	4.81	47.49	12.62	5	(62.51, 70.95)	8.44
silu	*RoPE*	0.0001	71.40	1.75	52.83	8.88	5	(69.87, 72.93)	3.06
leaky	*RoPE*	0.1000	61.24	8.78	48.13	10.86	6	(54.21, 68.27)	14.06
silu	*RoPE*	0.1000	62.79	8.79	49.27	10.69	8	(56.7, 68.88)	12.18
leaky	*Rotatory*	0.0001	68.95	2.90	53.66	9.58	15	(67.48, 70.42)	2.94
silu	*Rotatory*	0.0001	70.87	1.22	48.23	7.43	15	(70.25, 71.49)	1.24
leaky	*Rotatory*	0.1000	62.02	10.97	48.79	13.69	18	(56.95, 67.09)	10.14
silu	*Rotatory*	0.1000	62.68	8.53	49.41	10.86	15	(58.36, 67.0)	8.64
leaky	*None*	0.0001	59.1	–	38.1	–	5	–	–

Table 10: Fashion Dataset results with different positional encodings. Results from [24] at the end.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	0.0001	65.19	6.37	48.99	11.96	9	(61.03, 69.35)	8.32
silu	*None*	0.0001	64.90	3.06	47.51	9.84	6	(62.45, 67.35)	4.90
leaky	*None*	0.1000	69.69	1.28	57.94	1.69	3	(68.24, 71.14)	2.90
silu	*None*	0.1000	65.66	0.89	53.12	0.67	3	(64.65, 66.67)	2.02
leaky	*Learnt + Con*	0.0001	71.34	7.13	57.92	13.07	7	(66.06, 76.62)	10.56
silu	*Learnt + Con*	0.0001	66.64	4.49	48.38	12.61	7	(63.31, 69.97)	6.66
leaky	*Learnt + Con*	0.1000	73.86	7.18	58.89	8.15	7	(68.54, 79.18)	10.64
silu	*Learnt + Con*	0.1000	62.18	7.83	47.82	10.83	7	(56.38, 67.98)	11.60
leaky	*Abs + Con*	0.0001	70.84	5.13	57.63	9.89	9	(67.49, 74.19)	6.70
silu	*Abs + Con*	0.0001	65.55	4.59	49.37	10.61	6	(61.88, 69.22)	7.34
leaky	*Abs + Con*	0.1000	67.31	6.18	54.49	9.17	9	(63.27, 71.35)	8.08
silu	*Abs + Con*	0.1000	63.33	7.50	48.62	10.54	9	(58.43, 68.23)	9.80
leaky	*Rotatory + Con*	0.0001	70.64	5.29	59.30	6.98	10	(67.36, 73.92)	6.56
silu	*Rotatory + Con*	0.0001	67.18	5.82	50.27	13.21	10	(63.57, 70.79)	7.22
leaky	*Rotatory + Con*	0.1000	70.92	3.92	59.84	4.88	6	(67.78, 74.06)	6.28
silu	*Rotatory + Con*	0.1000	63.88	7.42	50.69	9.75	10	(59.28, 68.48)	9.20
leaky	*Learnt*	0.0001	72.08	4.44	56.55	9.98	7	(68.79, 75.37)	6.58
silu	*Learnt*	0.0001	66.18	7.81	46.61	11.21	8	(60.77, 71.59)	10.82
leaky	*Learnt*	0.1000	67.71	9.94	49.97	9.10	8	(60.82, 74.6)	13.78
silu	*Learnt*	0.1000	67.15	12.19	51.53	14.44	8	(58.7, 75.6)	16.90
leaky	*RMHA-4-Longer*	0.0001	68.72	1.47	43.46	0.79	3	(67.06, 70.38)	3.32
silu	*RMHA-4-Longer*	0.0001	70.13	0.42	46.41	4.18	6	(69.79, 70.47)	0.68
leaky	*RoPE-Longer*	0.0001	68.74	4.59	56.80	5.87	4	(64.24, 73.24)	9.00
silu	*RoPE-Longer*	0.0001	67.31	3.54	51.40	8.62	5	(64.21, 70.41)	6.20
leaky	*Abs*	0.0001	65.46	6.64	50.75	11.48	9	(61.12, 69.8)	8.68
silu	*Abs*	0.0001	64.15	2.94	42.68	8.87	6	(61.8, 66.5)	4.70
leaky	*Abs*	0.1000	69.20	4.78	54.43	8.87	6	(65.38, 73.02)	7.64
silu	*Abs*	0.1000	65.30	8.34	52.33	10.38	9	(59.85, 70.75)	10.90
leaky	*RMHA-4*	0.0001	68.65	0.48	43.24	0.35	6	(68.27, 69.03)	0.76
silu	*RMHA-4*	0.0001	69.70	0.64	43.75	1.44	6	(69.19, 70.21)	1.02
leaky	*RMHA-4*	0.1000	58.72	3.28	41.22	6.16	6	(56.1, 61.34)	5.24
silu	*RMHA-4*	0.1000	60.77	1.59	40.89	5.90	6	(59.5, 62.04)	2.54
leaky	*RopeOne*	0.0001	69.77	7.86	54.24	17.65	3	(60.88, 78.66)	17.78
silu	*RopeOne*	0.0001	62.14	0.74	37.49	3.31	3	(61.3, 62.98)	1.68
leaky	*RopeOne*	0.0001	67.82	6.59	54.10	12.14	6	(62.55, 73.09)	10.54
silu	*RopeOne*	0.0001	66.24	4.18	49.30	11.44	6	(62.9, 69.58)	6.68
leaky	*RoPE*	0.0001	65.29	6.06	49.86	11.15	7	(60.8, 69.78)	8.98
silu	*RoPE*	0.0001	63.93	3.25	46.10	7.61	5	(61.08, 66.78)	5.70
leaky	*RoPE*	0.1000	62.76	9.47	49.77	12.13	6	(55.18, 70.34)	15.16
silu	*RoPE*	0.1000	61.24	8.88	46.52	12.55	6	(54.13, 68.35)	14.22
leaky	*Rotatory*	0.0001	67.71	5.57	52.99	11.95	9	(64.07, 71.35)	7.28
silu	*Rotatory*	0.0001	67.07	3.63	54.88	4.54	6	(64.17, 69.97)	5.80
leaky	*Rotatory*	0.1000	68.07	4.68	56.31	6.21	6	(64.33, 71.81)	7.48
silu	*Rotatory*	0.1000	63.07	8.92	49.38	12.30	10	(57.54, 68.6)	11.06
leaky	*None*	0.0001	55.0	–	34.9	–	5	–	–

Table 11: Men Dataset results with different positional encodings. Results from [24] at the end.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	NaN	78.95	1.25	53.37	1.21	6	(77.95, 79.95)	2.00
silu	*None*	NaN	78.49	0.98	52.83	0.74	10	(77.88, 79.1)	1.22
leaky	*Learnt + Con*	NaN	78.34	1.28	54.14	1.39	6	(77.32, 79.36)	2.04
silu	*Learnt + Con*	NaN	79.32	1.50	53.68	1.97	6	(78.12, 80.52)	2.40
leaky	*Rotatory + Con*	NaN	80.62	0.55	56.07	0.79	6	(80.18, 81.06)	0.88
silu	*Rotatory + Con*	NaN	79.61	2.17	54.80	2.71	6	(77.87, 81.35)	3.48
leaky	*Abs + Con*	NaN	79.71	1.28	54.93	1.50	6	(78.69, 80.73)	2.04
silu	*Abs + Con*	NaN	79.34	1.67	54.23	1.84	10	(78.3, 80.38)	2.08
leaky	*Learnt*	NaN	78.64	2.05	54.30	2.41	6	(77.0, 80.28)	3.28
silu	*Learnt*	NaN	79.82	1.88	54.30	2.95	8	(78.52, 81.12)	2.60
leaky	*Rotatory + Con-Longer*	NaN	80.21	1.51	55.06	1.93	6	(79.0, 81.42)	2.42
silu	*Rotatory + Con-Longer*	NaN	80.29	0.47	55.18	0.55	6	(79.91, 80.67)	0.76
leaky	*Rotatory-Longer*	NaN	78.00	1.10	52.30	1.05	6	(77.12, 78.88)	1.76
silu	*Rotatory-Longer*	NaN	76.41	1.00	51.48	1.10	6	(75.61, 77.21)	1.60
leaky	*Abs*	NaN	75.27	2.30	49.61	2.90	6	(73.43, 77.11)	3.68
silu	*Abs*	NaN	73.27	3.25	47.22	3.75	6	(70.67, 75.87)	5.20
leaky	*RMHA-4*	NaN	78.85	0.51	53.32	0.44	12	(78.56, 79.14)	0.58
silu	*RMHA-4*	NaN	76.59	0.78	51.54	0.43	6	(75.97, 77.21)	1.24
leaky	*RopeOne*	NaN	78.82	0.62	53.45	0.42	6	(78.32, 79.32)	1.00
silu	*RopeOne*	NaN	78.27	1.19	52.85	0.88	6	(77.32, 79.22)	1.90
leaky	*RoPE*	NaN	78.97	0.33	53.64	0.27	6	(78.71, 79.23)	0.52
silu	*RoPE*	NaN	77.56	1.17	52.04	1.25	6	(76.62, 78.5)	1.88
leaky	*Rotatory*	NaN	76.56	2.17	51.09	2.29	6	(74.82, 78.3)	3.48
silu	*Rotatory*	NaN	76.21	1.77	51.06	2.04	6	(74.79, 77.63)	2.84
leaky	*None*	NaN	78.2	–	57.3	–	5	–	–

Table 12: Games Dataset results with different positional encodings. Results from [24] at the end.

A.2 Sparsity and deviation experiments

In this section, we present tables of the datasets created by reduction to evaluate whether the encoding type depends on the vector representation of the items, deviation, and sparsity. For these cases, we considered only three encodings: *RMHA-4* , *Rotatory* , and *None* . Table 17 contains results for datasets with a deviation below 12, which we deemed invalid and have therefore moved to this appendix.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	0.0001	48.00	22.19	33.53	26.31	6	(30.24, 65.76)	35.52
silu	*None*	0.0001	66.30	18.61	55.17	21.99	6	(51.41, 81.19)	29.78
leaky	*None*	0.1000	32.94	1.89	16.77	3.00	6	(31.43, 34.45)	3.02
silu	*None*	0.1000	63.85	17.73	52.02	20.44	6	(49.66, 78.04)	28.38
leaky	*Rotatory* + *Con*	0.0001	50.04	19.62	35.82	22.95	6	(34.34, 65.74)	31.40
silu	*Rotatory* + *Con*	0.0001	63.87	19.99	52.59	22.77	6	(47.87, 79.87)	32.00
leaky	*Rotatory* + *Con*	0.1000	49.17	21.02	35.89	25.10	6	(32.35, 65.99)	33.64
silu	*Rotatory* + *Con*	0.1000	67.94	19.26	57.24	23.51	6	(52.53, 83.35)	30.82
leaky	*RMHA-4*	0.0001	79.72	2.85	71.57	3.66	6	(77.44, 82.0)	4.56
silu	*RMHA-4*	0.0001	46.55	14.12	31.68	16.59	6	(35.25, 57.85)	22.60
leaky	*RMHA-4*	0.1000	57.99	20.07	45.36	24.01	6	(41.93, 74.05)	32.12
silu	*RMHA-4*	0.1000	61.14	14.64	48.46	18.10	6	(49.43, 72.85)	23.42

Table 13: SubFashion Dataset results for the best positional encodings and the None encoding case.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	0.0001	77.10	3.72	67.89	5.25	12	(75.0, 79.2)	4.20
silu	*None*	0.0001	76.33	6.11	67.00	7.08	12	(72.87, 79.79)	6.92
leaky	*None*	0.1000	69.29	15.21	58.64	18.19	6	(57.12, 81.46)	24.34
silu	*None*	0.1000	74.66	4.13	64.38	5.85	6	(71.36, 77.96)	6.60
leaky	*Rotatory* + *Con*	0.0001	71.82	11.90	61.69	14.11	12	(65.09, 78.55)	13.46
silu	*Rotatory* + *Con*	0.0001	75.28	3.63	65.31	5.03	12	(73.23, 77.33)	4.10
leaky	*Rotatory* + *Con*	0.1000	73.93	5.87	63.77	8.00	6	(69.23, 78.63)	9.40
silu	*Rotatory* + *Con*	0.1000	73.11	5.43	62.64	6.80	6	(68.77, 77.45)	8.68
leaky	*RMHA-4*	0.0001	75.96	2.71	66.03	4.00	8	(74.08, 77.84)	3.76
silu	*RMHA-4*	0.0001	70.44	9.61	59.67	11.50	8	(63.78, 77.1)	13.32
leaky	*RMHA-4*	0.1000	77.28	2.66	68.45	3.65	6	(75.15, 79.41)	4.26
silu	*RMHA-4*	0.1000	72.54	4.57	61.75	5.96	6	(68.88, 76.2)	7.32

Table 14: SubMen Dataset results for the best positional encodings and the None encoding case.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	NaN	49.53	1.01	28.84	0.69	6	(48.72, 50.34)	1.62
silu	*None*	NaN	48.31	1.22	28.36	0.86	6	(47.33, 49.29)	1.96
leaky	*Rotatory* + *Con*	NaN	52.23	1.57	31.51	0.96	6	(50.97, 53.49)	2.52
silu	*Rotatory* + *Con*	NaN	50.73	0.89	30.48	0.72	6	(50.02, 51.44)	1.42
leaky	*RMHA-4*	NaN	50.43	0.95	29.96	0.87	6	(49.67, 51.19)	1.52
silu	*RMHA-4*	NaN	48.97	1.35	28.57	0.79	6	(47.89, 50.05)	2.16

Table 15: Subgames Dataset results for the best positional encodings (*Rotatory* + *Con* and *RMHA-4*) and the *None* encoding case.

A.3 Resources

Due to the inherent instability of these models, it was necessary to run multiple seeds to obtain reliable results. Initially, we executed each model configuration three times. However, as we analyzed the results, we determined that additional seeds were needed for cases with high-deviation, aiming to maintain a confidence interval within ten points.

All experiments were conducted on $V100$ GPUs with 32 GB of memory. Training times varied depending on the dataset and encoding type. For the Men and Fashion datasets, training durations ranged from 4 to 6 hours, while for Beauty, the training times extended from 22 to 26 hours. Given the available resources, these experiments were spread over a period of six months.

We used the following seed values to ensure diversity in our training runs: 42, 43, 44, 45, 46, 1809, 1810, 1811, 1812, 1741, 1742, 1743, 1744, 1745, 123, 234, 345, 456, 567, 678, 789, 890, 102, 203, 304, 405, 506, 607, 708, 808, 901, 1001, 987, 8765, 7654, and 6543.

Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*	runs	CI	CI-length
leaky	*None*	0.0001	72.08	6.04	61.47	7.96	6	(67.25, 76.91)	9.66
silu	*None*	0.0001	73.95	3.33	64.05	4.71	6	(71.29, 76.61)	5.32
leaky	*None*	0.1000	71.56	5.38	60.96	6.93	6	(67.26, 75.86)	8.60
silu	*None*	0.1000	70.51	6.04	59.29	7.69	6	(65.68, 75.34)	9.66
leaky	*Rotatory + Con*	0.0001	71.52	6.39	60.71	8.17	6	(66.41, 76.63)	10.22
silu	*Rotatory + Con*	0.0001	72.77	2.96	62.56	4.31	6	(70.4, 75.14)	4.74
leaky	*Rotatory + Con*	0.1000	73.12	2.38	62.86	2.99	6	(71.22, 75.02)	3.80
silu	*Rotatory + Con*	0.1000	72.03	3.90	61.33	4.76	6	(68.91, 75.15)	6.24
silu	*RMHA-4*	0.0001	53.67	8.81	36.15	12.73	12	(48.69, 58.65)	9.96
leaky	*RMHA-4*	0.1000	61.82	12.74	48.93	14.82	6	(51.63, 72.01)	20.38
silu	*RMHA-4*	0.1000	61.27	5.87	47.24	6.96	6	(56.57, 65.97)	9.40

Table 16: SubMen2 Dataset results for the best positional encodings and the None encoding case.

Dataset	Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*
Submen	leaky	*RMHA-4*	0.1000	77.28	2.66	68.45	3.65
Submen	leaky	*None*	0.0001	77.10	3.72	67.89	5.25
Submen	silu	*None*	0.0001	76.33	6.11	67.00	7.08
Submen	leaky	*RMHA-4*	0.0001	75.96	2.71	66.03	4.00
Subfashion	leaky	*RMHA-4*	0.0001	79.72	2.85	71.57	3.66
Subfashion	leaky	*None*	0.1000	32.94	1.89	16.77	3.00
Subgames	leaky	*Rotatory + Con*	NaN	52.23	1.57	31.51	0.96
Subgames	silu	*Rotatory + Con*	NaN	50.73	0.89	30.48	0.72
Subgames	leaky	*RMHA-4*	NaN	50.43	0.95	29.96	0.87
Subgames	leaky	*None*	NaN	49.53	1.01	28.84	0.69
Submen3	silu	*None*	0.0001	73.95	3.33	64.05	4.71
Submen3	leaky	*Rotatory + Con*	0.1000	73.12	2.38	62.86	2.99
Submen3	silu	*Rotatory + Con*	0.0001	72.77	2.96	62.56	4.31
Submen3	leaky	*None*	0.0001	72.08	6.04	61.47	7.96

Table 17: The table displays the top 4 results for each dataset on cases with a standard deviation for HIT below 12.

A.4 Hyperparameters

In this section, we outline the hyperparameters used during training. Consistent with the approach used in CARCA [24], we adopted the same baseline hyperparameters across most experiments to maintain comparability. In specific cases, we experimented with variations in the upper bound of the norm for the encodings, activation functions, as well as adjusting the training duration to explore its impact on stability and performance.

Hyperparameter	**Men**	**Fashion**	**Games**	**Beauty**
Learning Rate	0.000006	0.00001	0.0001	0.0001
Max Seq. Length	35	35	50	75
Number of attention blocks	3	3	3	3
Number of attention heads	3	3	3	1
Dropout Rate	0.3	0.3	0.5	0.5
L2 reg. weight	0.0001	0.0001	0	0.0001
Embedding Dimension d	390	390	90	90
Embedding Dimension g	1950	1950	450	450
Residual connection in the cross-attention block	No	No	Yes	Yes

Table 18: Hyperparameters with corresponding options were used in the original CARCA model. This parameter were written in tensor flow. In "$L2$ reg. weight", the value 0 corresponds to no bound, which in Pytorch is Nan, and we denote it as *None* .

A.5 Deviations over all the trainings

In this section, we present the complete table of deviation values from all training sessions. In the main text, we focused on the training runs where the default upper bound values (0.0001) were used, along with *None* for the Games dataset. The trends discussed in Section 5.1 remain consistent here. Encodings that incorporate concatenation generally demonstrate greater stability, with *RMHA-4* continuing to be the most stable overall. Additionally, we observe that with extended training durations, the *Rotatory* encodings surpass *RoPE* in stability.

encoding	*Avg Dev Hit*	*Avg Dev NDCG*	runs	CI-length
Abs	7.20	9.42	10.57	8.42
Abs + Con	4.67	6.58	8.00	6.36
Learnt	7.43	9.98	9.07	10.17
Learnt + Con	4.09	6.53	6.93	5.66
None	3.05	5.08	6.71	5.62
RMHA-4	1.37	2.64	6.29	1.98
RoPE	4.03	6.55	5.14	6.52
Rotatory	3.92	6.10	9.00	4.92
Rotatory + Con	3.16	5.12	6.71	4.47
RopeOne	2.82	6.54	4.29	5.46
RoPE-Longer	3.33	8.15	5.25	5.99
Rotatory-Longer	1.31	1.35	4.00	3.00
Rotatory + Con-Longer	0.99	1.24	6.00	1.59
RMHA-4-Longer	0.71	4.00	6.00	1.14

Table 19: Deviations among the different encodings. Those per head, like *RMHA-4* or *RoPE* , are more stable than others, together with *Rotatory* .

A.6 Losses

This section shows some loss cases from our trainings showing the stability of *RMHA-4* . The first image shows four test losses for 0.0001 as upper bound, with silu activation. The first row corresponds to the *None* encoding, while the second row is the *RMHA-4* encoding. Each row contains the losses of four different seeds. The seeds are not paired vertically but randomly selected.

The second image shows the validation Losses for the analogous case in the Men Dataset.

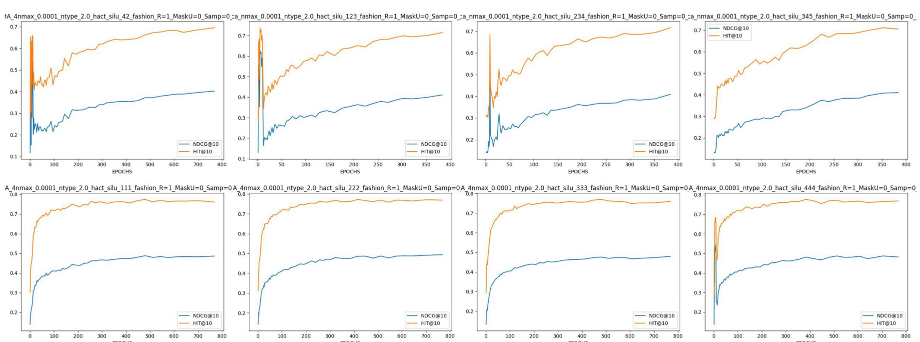

Figure 4: Losses randomly selected from *None* , first row, and *RMHA-4* , second row, for the test set with 0.0001 and silu. Dataset: Fashion

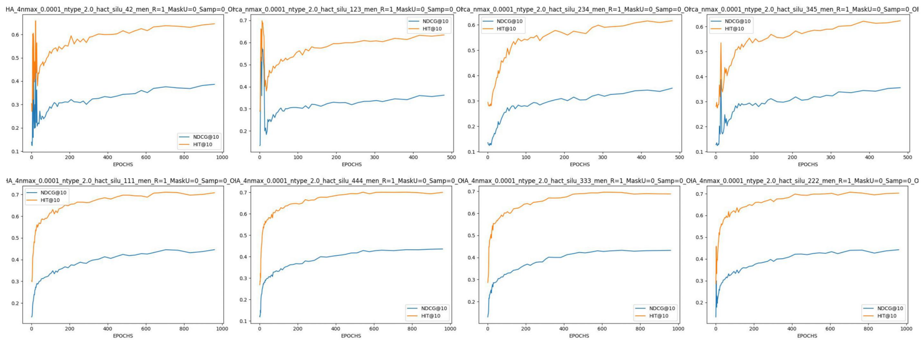

Figure 5: Losses randomly selected from *None* , first row, and *RMHA-4* , second row, for the test set with 0.0001 and silu. Dataset: Men

A.7 Decision diagram

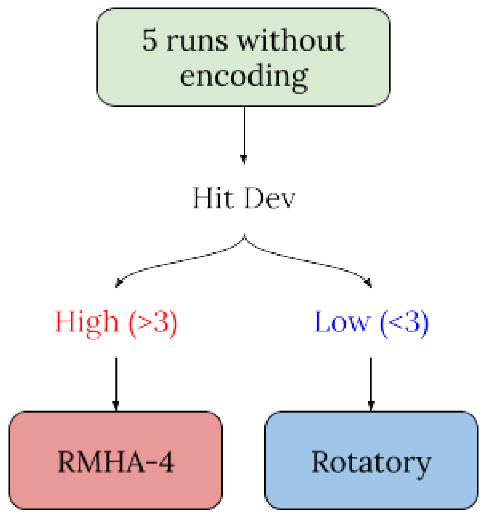

Figure 6: Decision tree for encoding selection based on dataset deviation. The diagram outlines the recommended encodings.

A.8 Extra experiments with SASRec++

All of our experiments were conducted using the CARCA model [24]. To assess whether the encoding behaviors observed in CARCA extend to other architectures, we also evaluated the SASRec++ model, an extension of the original SASRec model [15]. SASRec++ is conceptually similar to CARCA but differs in that it replaces the transformer decoder with a simple dot product operation.

We applied SASRec++ to the Submen dataset using the following encodings: *None*, *Learnt + Con*, *Rotatory + Con*, *Abs + Con*, *Learnt*, *Abs*, *RMHA-4*, *RopeOne*, *RoPE*, and *Rotatory*. As shown in Table 20, we find that *RMHA-4* continues to outperform other encodings, maintaining its position as the best-performing encoding across all experiments. Most of the results cluster around 40%, with minimal differences observed between encodings. The difference between *RopeOne* and other encodings is particularly small, with only slight improvements seen for *RMHA-4* with a low *Hit Dev* and *Learnt + Con* (as seen in Table 21) with a higher *Hit Dev*.

Dataset	Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*
Submen	silu	*RMHA-4*	0.1000	44.12	2.58	19.73	1.17
Submen	silu	*RMHA-4*	0.0001	42.45	2.91	18.91	1.11
Submen	leaky	*RopeOne*	0.0001	40.69	0.75	18.90	0.60
Submen	leaky	*RopeOne*	0.1000	40.66	0.74	18.67	0.39

Table 20: The table displays the top 4 results for Submen for SASRec++ on cases with a standard deviation for *Hit Mean*below 3.

Dataset	Act	encoding	nmax	*Hit Mean*	*Hit Dev*	*NDCG Mean*	*NDCG Dev*
Submen	silu	*Learnt + Con*	0.0001	58.08	11.49	26.41	6.37
Submen	silu	*RMHA-4*	0.1000	44.12	2.58	19.73	1.17
Submen	leaky	*RMHA-4*	0.0001	42.86	5.52	20.57	5.34
Submen	silu	*RMHA-4*	0.0001	42.45	2.91	18.91	1.11

Table 21: The table displays the top 4 results for Submen for SASRec++ on cases with a standard deviation for *Hit Mean*below 12.