Karlsruhe Institute of Technology

AI4LT

News Recommendation via a Session-Based Transformer

Bachelor's Thesis of

Martin Scheuermann

Artificial Intelligence for Language Technologies (AI4LT) Lab
Institut for Anthropomatics and Robotics (IAR)
KIT Department of Informatics

Reviewer:	Prof. Dr. Jan Niehues
Second reviewer:	Prof. Dr.-Ing. Rainer Stiefelhagen
Advisor:	M.Sc. Lukas Hilgert

10. Mai 2025 – 10. September 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 10.9.2025

. . . . . .

(Martin Scheuermann)

Abstract

As news consumption moves online, news recommender systems must balance personalization and scalability while ensuring low latency. The news domain poses additional challenges: articles have short lifecycles, user interests change rapidly, and article popularity distributions are highly skewed, making personalization difficult.

This thesis investigates whether Two-Tower models (TTMs) can adress these challenges. We design and evaluate SNeRT (Session-based News Recommender via Transformer), a transformer-based TTM that encodes user sessions and candidate articles into a shared embedding space for efficient large-scale retrieval and ranking. Using the EB-NeRD dataset, we address three research questions: focusing on (1) global recommendation across the catalog, (2) in-view recommendation within candidate sets, and (3) beyond-accuracy measures such as diversity, novelty, and coverage. SNeRT outperforms a strong popularity baseline in retrieval and improves coverage and diversity, though its ranking performance remains below that of cross-encoders. Using a FAISS index on CPU, the system achieves a retrieval latency of 4.47 ms, demonstrating its practical suitability for latency-sensitive applications.

A key contribution of this work is the analysis of embedding collapse, a central weakness of two-tower models in which embeddings degenerate into a low-rank space and lose expressive capacity. Our results show that existing strategies fail under the long-tail distribution of article popularity, whereas spectral regularization with a log-determinant loss offers a robust and efficient solution. To our knowledge, this is the first application of log-determinant spectral regularization in a TTM for recommendation, improving embedding stability and retrieval metrics.

Zusammenfassung

Mit dem zunehmenden Online-Konsum von Nachrichten stehen Empfehlungssysteme vor der Herausforderung, Personalisierung und Skalierbarkeit zu vereinen und gleichzeitig eine geringe Latenz sicherzustellen. Der Nachrichtenbereich stellt dabei zusätzliche Herausforderungen: Artikel haben nur kurze Lebenszyklen, Nutzerinteressen ändern sich schnell, und die Beliebtheitsverteilung von Artikeln ist stark verzerrt, was eine effektive Personalisierung erschwert.

In dieser Arbeit wird mit SNeRT ein Transformer-basiertes Two-Tower-Modell (TTM) vorgestellt, das Nutzersitzungen und Kandidatenartikel in einen gemeinsamen Embedding Space abbildet und dadurch effizientes Large-Scale Retrieval ermöglicht. Auf Basis des EB-NeRD-Datensatzes adressieren wir drei Forschungsfragen: globale Empfehlungen über den gesamten Katalog, In-View-Empfehlungen innerhalb vorgegebener Kandidatenmengen sowie qualitative Aspekte jenseits klassischer Genauigkeitsmetriken, darunter Diversity, Novelty und Coverage. Die Ergebnisse zeigen, dass SNeRT eine starke Popularitätsbaseline im Retrieval übertrifft und insbesondere Coverage und Diversity verbessert, im Reranking jedoch hinter Cross-Encodern zurückbleibt. Mit einer Retrieval-Latenz von 4,47 ms mit einem FAISS-Index erfüllt das Modell zudem die Anforderungen für latenzkritische Anwendungen im praktischen Einsatz.

Ein wesentlicher Beitrag dieser Arbeit ist die Analyse von Embedding Collapse, einer zentralen Schwäche von TTMs, bei der Embeddings in einen niederdimensionalen Raum degenerieren und dadurch ihre Ausdruckskraft verlieren. Während etablierte Ansätze unter der stark unausgeglichenen Verteilung der Artikelpopularität wirkungslos bleiben, erweist sich spektrale Regularisierung mit dem Log-Determinanten-Loss als robuste und effiziente Lösung. Nach unserem Wissen handelt es sich hierbei um die erste Anwendung von Log-Determinant-Regularisierung in einem TTM für Empfehlungssysteme, wodurch die Stabilität der Embeddings erhöht und die Retrieval-Performance verbessert wird.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ANN	Approximate Nearest Neighbor
BERT	Bidirectional Encoder Representations from Transformers
BNN	Badische Neueste Nachrichten
CE	Cross Encoder
CL	Contrastive Learning
CTR	Click-Through-Rate
EB-NeRD	Ekstra Bladet News Recommendation Dataset
erank	effective rank
MoCo	Momentum Contrast
MLP	Multi-Layer-Perceptron
NER	Named Entity Recognition
NewsRec	News Recommendation
SNeRT	Session-based News Recommender via Transformer
NLP	Natural Language Processing
PLM	Pretrainend Language Model
RQ	Research Question
RecSys	Recommendation System
RecSys24	RecSys Challenge 2024 [31]
QA	Question Answering
TTM	Two-Tower Model

1. Introduction

This thesis focuses on the development and evaluation of a deep learning-based recommendation system for news articles, with practical relevance to real-world deployment in collaboration with a regional newspaper.

1.1. Motivation

Over the past few years, news consumption has shifted from print to mobile devices. In the online space, newspapers face increasing competition for reader attention while having limited front-page space to highlight articles. As a result, front-page content must be carefully selected. Similar industries, such as social media and streaming platforms, have successfully integrated recommendation engines as a core feature for user retention. Likewise, news recommendation systems can personalize article suggestions based on a user's reading history and interests, creating a more personalized news experience. However, news recommendation introduces unique challenges compared to other recommendation systems. The short lifespan of most news articles leads to a cold-start problem, making it difficult to generate recommendations for newly published content. Additionally, due to the societal and political impact of news, metrics beyond accuracy —such as coverage, novelty, and diversity— are crucial for ensuring balanced and effective recommendations. These challenges highlight the need for practical and adaptable solutions in real world news settings. In this context, the project was carried out in collaboration with the local newspaper *Badische Neueste Nachrichten* (BNN)[1].

1.2. Research Questions

The goal of BNN was to explore the feasibility of a deep learning-based recommendation system capable of recommending relevant articles from a large catalog with low latency and without continuous retraining. Initially, we aimed to develop and evaluate our algorithms using user-interaction data from BNN. However, due to privacy concerns and organizational delays, we were unable to gain access to this data during the course of this work. As an alternative, we based our experiments on the EB-NeRD dataset [30] provided by Ekstra Bladet[2], a Danish newspaper. This dataset was the basis of the RecSys '24 Challenge[3][31]. This enables us to compare our results to teams that participated in the challenge, although the challenge's focus differs substantially from our main task. The

[1] https://bnn.de/

[2] https://ekstrabladet.dk/

[3] https://recsys.acm.org/recsys24/

goal of this thesis is to build a recommendation system for news articles and evaluate it on different metrics to assess its capabilities. We break these requirements down into three research questions (RQs), each addressing a distinct aspect of the recommendation task:

- **RQ1: Global Recommendation Task** – Can a model be trained to recommend articles to users based on their past interaction history, without continuous retraining, low latency, and with good recommendation metrics across the whole article catalog?
- **RQ2: In-View Recommendation Task** – How does the model designed for RQ1 perform on the in-view recommendation task proposed at RecSys '24, which focuses on ranking the articles displayed to the user in a candidate set by their likelihood of being clicked?
- **RQ3: Beyond-Accuracy Evaluation** – How does our model's predictions score on beyond-accuracy metrics such as coverage, serendipity, novelty, and diversity?

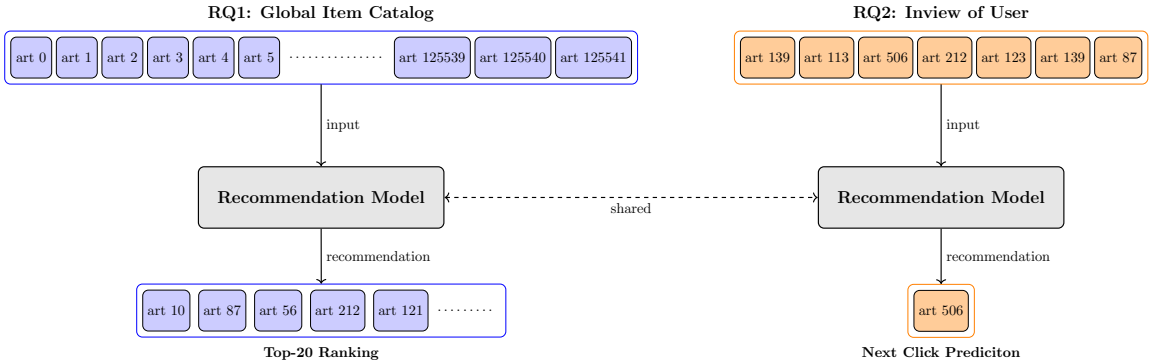


Figure 1.1.: Comparison between Global Recommendation Task (RQ1) and In-View Recommendation Task (RQ2).

While these research questions aim to cover multiple aspects of the news recommendation task, certain limitations remain.

1.3. Scope and Limitations

As we were unable to train our recommendation model on the BNN dataset, we could not carry out the planned cross-dataset comparison between BNN and EB-NeRD as initially proposed in the thesis proposition.

Beyond dataset access, there are broader challenges in evaluating recommendation systems (RecSys) effectively. RecSys operate within a feedback loop, where recommended items gain more exposure and clicks, potentially reinforcing existing biases. Production systems must account for this to avoid feedback-driven skew. Since we work with a static dataset, we are limited to offline evaluation and cannot assess the model's live impact. In the conclusion, we briefly suggest strategies for addressing these issues in a live system.

Finally, the scope of this thesis is restricted to a specific modeling approach (contrastive two-tower models). While this allows for a focused study, alternative architectures (e.g., cross-encoders, graph-based recommenders, or reinforcement learning methods) remain unexplored, although we briefly tested cross-encoders as a second-stage reranker.

1.4. Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2: Background and Related Work** Reviews the foundational techniques underlying our model and surveys existing approaches to news recommendation.
- **Chapter 3: Dataset and Feature Engineering** Introduces the EB-NeRD dataset, presents exploratory analysis, and outlines the feature engineering process used for model training and evaluation.
- **Chapter 4: Approach** Describes the architecture of our recommendation model, explains the design choices and training setup, and evaluates strategies for mitigating embedding collapse of user representations.
- **Chapter 5: Experiments and Evaluation** Reports the results of our experiments, including both accuracy-based and beyond-accuracy evaluations, and analyzes different negative sampling compositions as well as hyperparameter optimizations.
- **Chapter 6: Conclusion and Future Work** Summarizes the main findings of the evaluation, discusses their implications for adapting the model to BNN, and outlines directions for future research.
- **Appendix** Provides the formal definitions of metrics we used, dataset examples, full feature lists, additional figures and further experimental results referenced throughout the thesis.

2. Background Information and Related Work

In this chapter, we first review foundational methods and technologies for our model and existing work on recommendation systems. For the foundational methods we take a look at transformer architectures, text encoders and contrastive learning. Afterwards we introduce the recommendation task in general and different approaches taken to solve it. Here we introduce the two-tower model and its differences to a cross-encoder model. Before taking a closer look at the challenges of news recommendation and current approaches of solving them.

2.1. Transformers

Transformer models have revolutionized multiple fields of AI, such as Natural Language Processing (NLP), Computer Vision, and Speech Recognition [23]. In NLP, these models treat text as a sequence of tokens. The underlying attention mechanism captures short- and long-range dependencies in sequences [65], while being fully parallelizable.

2.1.1. Attention Mechanism

Multiple versions of attention exist for different purposes [3]. We will focus on scaled dot-product attention, which is used in the original transformer paper [65]. To understand scaled dot-product attention, it is useful to examine the sequence of vector operations involved.

Dot-product attention Given a sequence of input embeddings $X = [x_1, \ldots, x_n] \in \mathbb{R}^{n \times d_{\text{model}}}$, each embedding x_i is transformed via a learned linear mapping into a query vector q_i, a key vector k_i, and a value vector v_i [35, 65]. For a query vector q_i and the key-value pairs $\{(k_j, v_j)\}_{j=1}^{n}$, we calculate [18, 35]:

$$\begin{aligned} s_{ij} &= q_i \cdot k_j && \text{(dot product of query } q_i \text{ with key } k_j\text{)}, \\ \alpha_{ij} &= \frac{\exp(s_{ij})}{\sum_{\ell=1}^{n} \exp(s_{i\ell})} && \text{(softmax normalized attention weight)}, \\ z_i &= \sum_{j=1}^{n} \alpha_{ij} v_j && \text{(weighted sum of values)}. \end{aligned} \tag{2.1}$$

This query-key system is conceptually similar to the dot-product relevance calculation in TTMs (introduced later in Section 2.4.1), in both cases a dot product determines the

relevance of two embeddings to each other. Here the score determines how relevant the value vector of one position is to another position, enabling the introduction of long range dependencies, without the vanishing gradient problem of RNN training [39, 65].

Self-Attention While dot-product attention measures similarity for a single query, self-attention applies this mechanism across all positions in a sequence. For efficient computation, the input embeddings are stacked into a matrix $X \in \mathbb{R}^{n \times d_{\text{model}}}$, and three learnable projection matrices are introduced:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V, \qquad \text{where} \qquad W^Q, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k},\ W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}. \tag{2.2}$$

Then attention gets applied as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{2.3}$$

The scaling factor $\frac{1}{\sqrt{d_k}}$ prevents the raw dot-product values from growing too large as d_k increases; helping to mitigate vanishing gradient problems of the softmax distribution [65]. These self-attention concepts can be expanded upon with multi-head attention. In the case of multi-head attention each head learns distinct projection matrices for their individual query, key, and value matrices. After applying self-attention the results get concatenated and the linear layers applied (see Figure 2.1) [65]. This design enables each head to attend to different aspects of the sequence. For example, analyses of attention weights suggest that different heads capture distinct linguistic functions, such as focusing on nouns versus verbs in NLP tasks [7]. Based on self-attention a transformer-layer can be defined in the next chapter.

2.1.2. Architecture

A Transformer layer consists of a multi-head self-attention mechanism and a position-wise feed-forward network, each wrapped with residual connections and layer normalization [65]. Stacking such layers yields the full encoder-decoder architecture shown in Figure 2.1, where residual connections and normalization stabilize training of deep models [65].

Encoder vs. Decoder The Transformer architecture consists of encoder layers, which attend bidirectionally to the input sequence, and decoder layers, which combine masked self-attention with cross-attention over the encoder (see Figure 2.1). Encoder-only models such as BERT [9] use the encoder stack, decoder-only models such as GPT [48] rely on the decoder stack, while sequence-to-sequence models employ both. In all variants, positional encodings provide tokens with information about their order in the sequence.

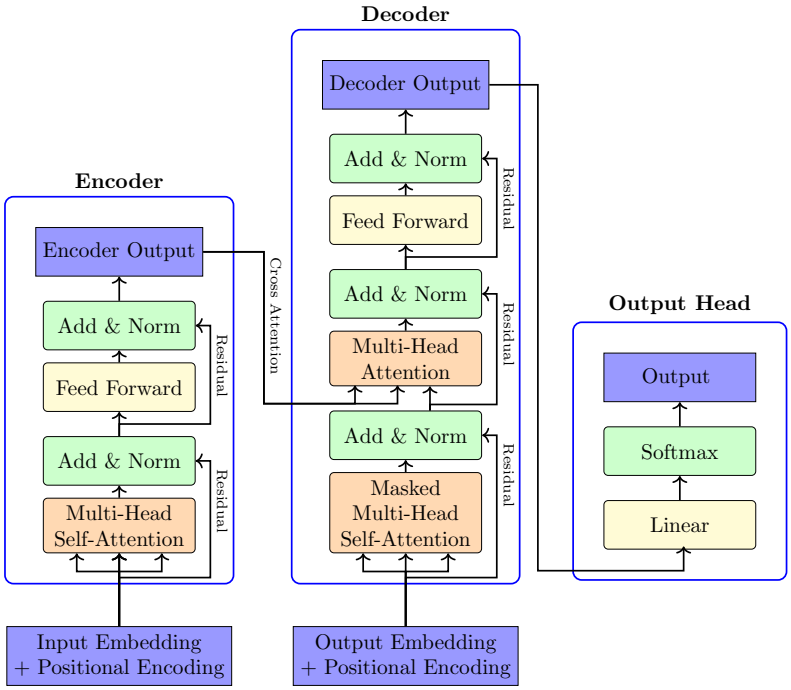


Figure 2.1.: Transformer architecture with separate encoder and decoder stacks, showing the flow from input embeddings with positional encoding through attention, feed-forward, and normalization layers to the output probabilities. Adapted from [65].

Positional Encoding The self-attention mechanism has no concept of absolute and relative positions of inputs. Instead, this information has to be encoded into the input. Various positional encodings have been proposed for this purpose [4]. The original implementation uses sinusoidal functions with different frequencies to encode relative and absolute token positions [65] .

$$\mathbf{PE}(\text{pos})_{2i} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \quad \mathbf{PE}(\text{pos})_{2i+1} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right). \tag{2.4}$$

Learned positional embeddings are an alternative for fixed-length input sequences. Here positions get encoded like a category, a trainable matrix $P \in \mathbb{R}^{N_{\max} \times d}$ maps each position $i \in \{1, \ldots, N_{\max}\}$ to its own vector $\mathbf{p}_i = P_{[i,:]}$. Encoder-only transformers like the BERT-models, typically use learned-positional embeddings [9, 59]. No matter the positional encoding approach, the positional encoding is added in the same way. To calculate the final input $\mathbf{x}_i$,the position embedding $E_{\text{pos}}[i]$ gets added to the input $E_{\text{tok}}[i]$) before the first Transformer layer.

$$\mathbf{x}_i = E_{\text{tok}}[i] + E_{\text{pos}}[i] \tag{2.5}$$

2.2. Text Encoders

Content-based recommendation (see Subsection 2.4) requires compact embeddings of each article's text. Early approaches like TF-IDF [55] represent documents with sparse word counts to measure similarity, but fail to capture semantics. Neural word embeddings such as Word2Vec [36] improved this by training a shallow model to predict words from context, yielding dense vectors in $\mathbb{R}^d$. Document embeddings can then be obtained by averaging word vectors, but these static representations ignore context and long-range dependencies. Transformer-based models (see Subsection 2.1) overcome this by conditioning token embeddings on their context, with BERT setting new benchmarks in text understanding [9].

2.2.1. BERT Models

BERT-based models are widely applied in retrieval, classification, sentiment analysis, and question answering (QA) [9, 59, 62, 82]. Their strength lies in contextual embeddings: each token is represented in relation to the full sequence, overcoming the limitations of static word embeddings.

Architecture and Pretraining A BERT model consists of three main components: an input embedding module, a stack of Transformer encoders, and task-specific heads. The input layer combines token embeddings with positional encodings and optional segment embeddings to represent sentence pairs [9]. These are processed by multiple layers of bidirectional self-attention and feed-forward networks, which iteratively refine contextual representations for each token. During pretraining, BERT is optimized on two objectives: Masked Language Modeling, where 15% of tokens are randomly masked and predicted from context, and Next Sentence Prediction, which trains the model to identify whether two sentences follow each other in the source text [9]. Together, these objectives encourage BERT to capture both fine-grained word semantics and higher-level discourse relations.

Fine-tuning Fine-tuning adapts pretrained language models to downstream tasks by adding lightweight task-specific heads (e.g., linear classifiers) and retraining on small task-specific datasets [9, 37]. For sequence-level tasks such as classification, the final hidden state of the `[CLS]` token is typically passed to the task head [59], whereas token-level tasks such as Named Entity Recognition (NER) directly use the contextualized embeddings of individual tokens [64, 82]. A key challenge in this setting is fine-tuning stability, especially on small datasets. Prior work shows that stability and generalization can be improved by using small learning rates with bias correction and extended training schedules [37].

BERT for Article Embeddings BERT can be directly used for article representations by pooling token embeddings (e.g. `[CLS]`, average, or attention pooling) to yield fixed-length vectors [73]. These embeddings capture synonyms and topical similarity, making them effective features in RecSys. Fine-tuning improves results by adapting representations to article-level signals. Multilingual models such as XLM-RoBERTa [8] and LaBSE [14] broaden coverage beyond English. ModernBERT [72] further improves performance with

an extended context window (8192 tokens), though no multilingual version is yet available. Beyond embeddings, BERT variants can also provide sentiment scores [59] or extract entities such as named persons via NER [64].

2.3. Contrastive Learning

Contrastive learning (CL) is a powerful paradigm for self-supervised representation learning [5]. The core principle of contrastive learning is the comparison of positive and negative examples. The model learns to map semantically similar items closer in the embedding space, while pushing dissimilar items away [24]. The resulting embedding space encodes patterns and relationships in the data, which then can be used in classification [5], retrieval [49], and recommendation tasks [78]. To train these models, we define a similarity function $\text{sim}(x, y)$, which measures distance in the embedding space. The most common choices is cosine similarity [24].

$$\text{sim}(x, y) = \frac{x^\top y}{\|x\| \, \|y\|} \tag{2.6}$$

To learn such representations, we require a loss function that minimizes the distance between positive pairs while maximizing the distance to negative examples. A widely used approach in contrastive learning is the InfoNCE loss.

2.3.1. InfoNCE Loss

Given a similarity function $\text{sim}(x, y)$, we can calculate the loss based on positive and negative pairs. The most common choice is the InfoNCE loss [38], which for a user embedding u, a positive item i^+, and the set of all sampled items $\mathcal{N}$ is defined as:

$$\mathcal{L}_{\text{InfoNCE}} = -\log\left(\frac{\exp\left(\frac{\text{sim}(u,i^+)}{\tau}\right)}{\sum_{j\in\mathcal{N}} \exp\left(\frac{\text{sim}(u,j)}{\tau}\right)}\right). \tag{2.7}$$

The temperature parameter τ in the InfoNCE loss controls the sharpness of the underlying softmax distribution [5]. Choosing an appropriate τ is critical: too low can lead to unstable training, while too high may underemphasize difficult contrasts and slow convergence [5, 71]. Usually not all non-positive items can be used as negatives in the loss-function. To be more efficient, a sampling strategy for negatives is required [79].

2.3.2. Negative Sampling Strategies

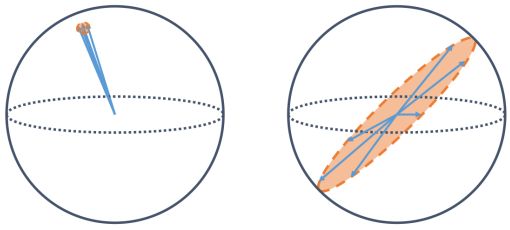
Effective contrastive learning hinges on selecting a diverse and informative set of negatives [24, 79]. A key challenge lies in choosing negatives of the right difficulty: overly easy examples provide little training signal, while excessively hard ones can destabilize learning and hinder convergence. Several strategies have been proposed to address this. In-batch

negatives treat all other samples in the batch as negatives, making the approach simple and efficient [5]. Memory banks extend this idea by maintaining a large repository of past embeddings for negative sampling [20]. Hard negative mining instead selects those items most similar to the positive (highest $\text{sim}(u, j)$), forcing the model to learn finer-grained distinctions [56]. Debiased sampling reweights negatives to correct for false negatives, i.e., semantically similar items mistakenly treated as negatives [6].

Each method involves trade-offs in computational cost, memory requirements, and the quality of sampled examples. In practice, domain-specific constraints such as time-dependency (see Section 2.5) must also be considered. Regardless of strategy, embedding collapse remains a common challenge in contrastive learning.

2.3.3. Embedding Collapse

Embedding collapse occurs when output embeddings converge to a single point or a low-dimensional subspace during training (see Figure 2.2). In this situation, items lose their discriminative capacity, as the model maps them into nearly identical representations. This severely limits the expressiveness of the embedding space and reduces the overall effectiveness of the model [61, 71].



(b) complete collapse (c) dimensional collapse

Figure 2.2.: Illustration of embedding collapse from [26], showing both complete collapse and dimensional collapse.

The literature identifies several factors that can hinder contrastive learning. Sampling negatives that are too few or too easy fails to provide a strong repulsive signal [5]. Excessively strong data augmentations can cause a collapse of the augmented features, as the resulting high data variance prevents the model from extracting a useful downstream signal [26]. Finally, reliance on simple discriminators such as item popularity can induce a simplicity bias: instead of capturing richer semantic dependencies, the model exploits these shortcuts while neglecting more informative features [81], leading to a collapsed representation.

Metrics for Embedding Collapse To evaluate embedding collapse, two metrics are commonly used. The first is the average cosine similarity (avg_sim), which directly quantifies redundancy among embeddings. Given embeddings $X = \{x_i\}_{i=1}^{n}$, the average pairwise

cosine similarity is defined as [16]:

$$\text{avg\_sim}(X) = \frac{1}{n(n-1)} \sum_{i \neq j} \frac{x_i^\top x_j}{\|x_i\| \, \|x_j\|}. \tag{2.8}$$

High values of avg_sim(X) indicate that embeddings are highly similar to each other, meaning that most users are mapped into nearly the same point in the space. This reflects a complete embedding collapse (see Figure 2.2). The second is the effective rank (erank), which measures the spectral diversity of the embedding space, i.e., the amount of independent information content carried by the embeddings. Given the eigenvalues $\{\sigma_i\}_{i=1}^{d}$ of the embedding matrix, we define $p_i = \dfrac{\sigma_i}{\sum_{j=1}^{d} \sigma_j}$ and compute the erank [42] with the following formula.

$$\text{erank} = \exp\Big(-\sum_{i=1}^{d} p_i \ln p_i\Big). \tag{2.9}$$

A higher erank indicates that the embedding space spans many independent directions and thus encodes richer information, while a low erank reflects dimensional collapse (see Figure 2.2). Several strategies have been proposed in the literature to mitigate embedding collapse, which we explore and adapt in our own work (see Subsection 4.5).

2.4. Recommender Systems

Recommender Systems (RecSys) filter and rank items from large catalogs based on user and item information, with the goal of providing personalized suggestions [10]. Formally, recommendation can be defined as a scoring function

$$f : U \times I \to \mathbb{R},$$

where U is the set of users, I the set of items, and $f(u, i)$ the score for a given pair. The top-ranked item is

$$i'_u = \arg\max_{i \in I} f(u, i),$$

and a top-k list is obtained by selecting the k highest-scoring items. A fundamental challenge in RecSys is predicting user preferences despite the sparsity of both implicit interactions (e.g., clicks, views) and explicit signals (e.g., ratings) [57]. Recommender systems are commonly categorized into collaborative filtering, content-based filtering, and hybrid approaches [57].

Collaborative Filtering Collaborative filtering infers preferences from user-item interaction patterns: if two users behaved similarly in the past, items liked by one are likely relevant to the other [57, 84]. Memory-based methods directly compute similarities, while model-based approaches (e.g. matrix factorization) learn predictive patterns [84]. Its main drawbacks are poor performance with sparse data and inability to handle the cold-start problem, where new users or items lack interaction history [84].

Content-based Filtering Content-based methods recommend items similar to those previously interacted with, using metadata, categories, or learned embeddings [40, 57]. They mitigate cold-start issues but tend to produce narrow recommendations with low catalog coverage [15, 57].

Hybrid Approaches Hybrid systems combine collaborative and content-based methods to combine their strengths [57]. Strategies include weighted aggregation, switching (depending on context), and feature combination. Recent deep learning-based hybrids include two-tower models, which jointly learn user and item embeddings from both interaction and content data, offering scalable and flexible recommendation [87].

2.4.1. Two-Tower Models

Two-Tower Models (TTM) are a widely adopted architecture in deep learning–based RecSys [87]. One tower encodes user information, while the other encodes item features, and both produce dense embeddings (see Figure 2.3).

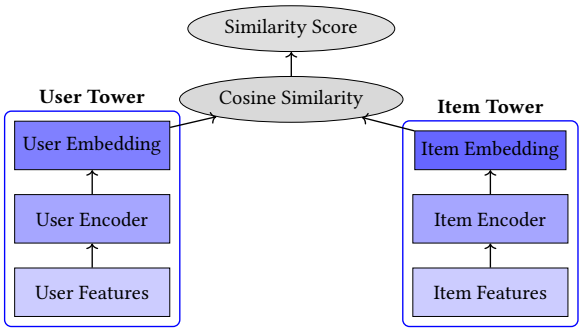


Figure 2.3.: Baseline Two-Tower architecture.

The recommendation ranking is then computed with a similarity function, typically cosine similarity. This modular design enables efficient candidate retrieval via approximate nearest neighbor (ANN) search with vector databases such as FAISS [11], while allowing flexibility across user and item types [25]. Each tower can use different neural networks —for instance, Transformers (see Subsection 2.1) to model user sequences [68] or pre-trained language models like BERT [9] for richer text features. Training TTMs is typically formulated as a contrastive learning task, where user and item embeddings are learned jointly by aligning positive pairs and separating negatives.

Two-Tower Model vs. Cross-Encoder An alternative is the Cross-Encoder (CE), which processes user and item features together in a single model, allowing richer interactions and higher accuracy [54] (see Figure 2.4).

The main drawback is scalability: since every user–item pair must be encoded jointly, precomputing becomes infeasible and latency increases significantly. Moreover, CEs tend to degrade in performance when too many candidates are reranked at once, as the decoder must distribute attention across all items, diluting its expressiveness [33, 43]. In

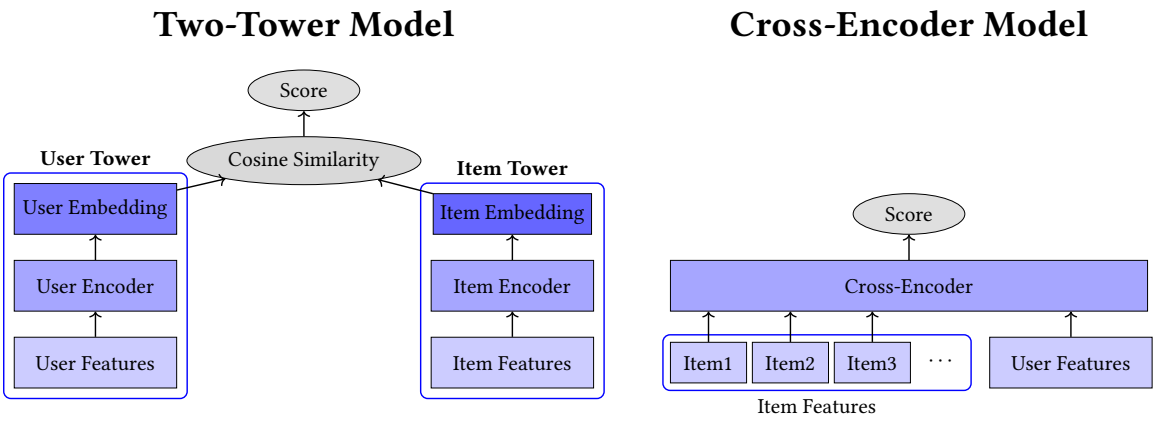


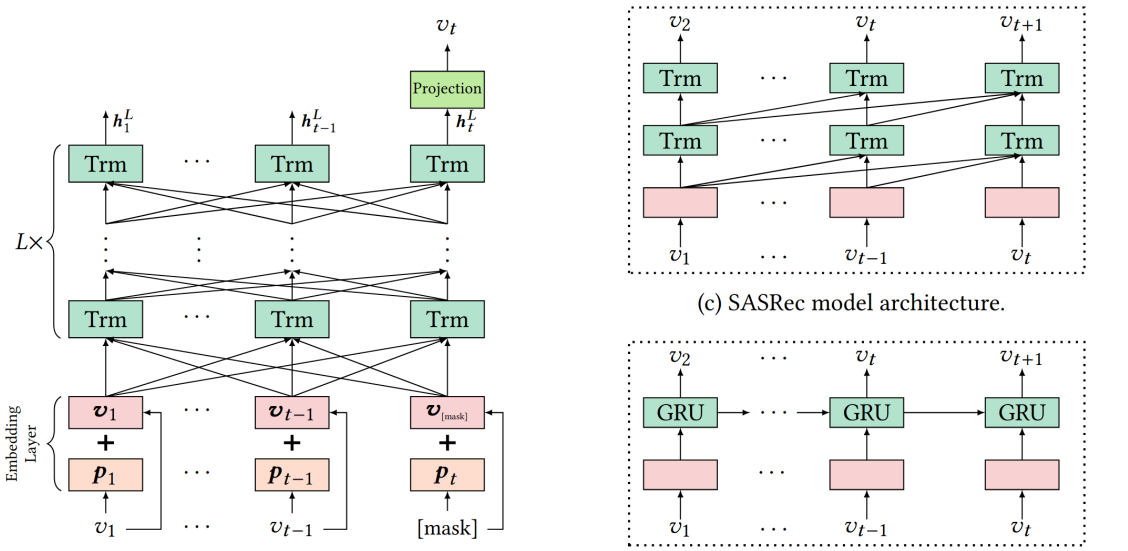
Figure 2.4.: Side-by-side comparison of Two-Tower and Cross-Encoder architectures.

practice, many RecSys adopt a two-stage pipeline: a TTM efficiently retrieves a shortlist of candidates, which a CE then reranks for improved accuracy [51, 61].

2.4.2. Transformer-based Sequential Recommendation

Sequential recommendation predicts a user's next interaction by modeling the order and context of past behaviors. Formally, given a sequence of past interactions s, the task is to estimate the conditional probability distribution $P(y \mid s)$, where y denotes the next item [27]. Early models such as GRU4Rec [21] used recurrent networks, but more recent approaches adopt self-attention and the Transformer architecture (see Subsection 2.1). Two representative models are SASRec [27] and BERT4Rec [63]. SASRec applies transformer-encoder layers to user interaction sequences and is trained autoregressively for next-item prediction. BERT4Rec follows the BERT paradigm (see Subsection 2.2.1), masking random interactions and predicting them with cross-entropy over the item pool [9]. This bidirectional objective improves robustness on sparse or noisy sequences. A visual comparison of these architectures with earlier RNN-based approaches is shown in Figure 2.5, highlighting their shift from recurrent to self-attention-based sequence modeling.

A common limitation of sequential recommendation models is the use of item IDs as prediction targets. This restricts their applicability in news recommendation, where new articles are continuously introduced and unseen IDs have to get recommended.



(b) BERT4Rec model architecture.

(c) SASRec model architecture.

(d) RNN based sequential recommendation methods.

Figure 2.5.: Shift from RNN-based architectures to self-attention in sequential recommendation: SASRec applies Transformer encoders autoregressively, while BERT4Rec adopts bidirectional masked prediction. Figure reproduced from [63].

2.5. News Recommendation

The goal of news recommendation (NewsRec) is to suggest news articles that align with user preferences. Compared to other recommendation domains, NewsRec presents several unique challenges [32]. First, news platforms face high item churn, as new articles appear continuously while older ones quickly become irrelevant, leading to a persistent cold-start problem. Second, feedback signals are typically implicit, such as clicks or reading time, which are generally weaker indicators of user satisfaction than explicit ratings [32]. Third, user interests can shift rapidly in response to current events, such as elections or major sports tournaments, requiring models to adapt quickly. Fourth, article relevance is often context-dependent, varying with factors like the time of day, device type, or user location [32]. Finally, news recommendation carries societal and political sensitivity, since systems take on an editorial role and must ensure diversity and coverage in their recommendations [32].

Addressing these challenges is essential for building effective NewsRec systems. While traditional approaches such as collaborative filtering, content-based filtering, and hybrid methods have been widely applied, recent years have seen a shift toward deep learning-based models, which have achieved superior performance across many benchmarks [32, 69]. NewsRec models can be split into TTMs and CEs (see Subsection 2.4.1). We will focus the following analysis on TTMs, in line with the scope of this thesis. The two key tasks in a NewsRec system are news modeling and user modeling [32, 69].

2.5.1. News Modeling

The goal of News modeling is to learn a dense vector representation of an article, used as the item vector in TTMs or to cluster similar articles based on their embedding [69]. This is a form of content-based filtering (see Subsection 2.4). In a TTM-NewsRec model we refer to the item encoder as an article encoder. Important input features for article encoders are usually a mix of textual and contextual features (see Table 2.1). Earlier

Feature	Description
Text	Raw text or precomputed text embeddings
Meta data	Category, article type, article length etc.
Premium status	Free or premium required
Image embeddings	Embedding of the title image
Location	Geographical location of the article content
Sentiment	Sentiment polarity derived from the text
Temporal features	Article age, current popularity

Table 2.1.: Overview of commonly used article features [50, 66, 70, 75].

models used CNNs, which are fast and effective for classification but struggle with deeper semantic understanding [66]. To address this limitation, attention-based architectures such as NRMS [75] were introduced, which use multi-head self-attention to model contextual relationships between words in news articles. Later pretrained language models (PLM) like BERT were used as a Text Encoder (see Subsection 2.2.1). Examples include PLM4NewsRec [73], which integrates a domain-adapted PLM, and NewsBert [76], which distills and finetunes BERT [9] for efficient news encoding.

Since TTMs are trained jointly, we need an aligned user encoder that models the user interactions for our final predicition.

2.5.2. User Modeling

In a TTM, user modeling is handled by the user encoder, which learns personalized user representations from interaction history and contextual signals [69]. Key input features for the user encoder are described in Table 2.2.

Feature	Description
Behavioral history	Viewed articles, scroll depth, and reading time
Article embeddings	Embeddings of previously read articles
Contextual information	Time of day, device type, and user location
User metadata	Age group, subscription status, etc.
User statistics	Category preferences, session length, content diversity
Sequential patterns	Order and temporal gaps of interactions

Table 2.2.: Overview of commonly used user features [50, 66, 70, 75].

Early methods pooled previously clicked article embeddings (either by averaging or weighting) to obtain a user embedding [74]. While simple and interpretable, this approach ignores sequence, recency, and interaction information. LSTUR [1] improves on this by using recurrent neural networks (RNNs) to model both long- and short-term user interests [58]. More recent models incorporate attention mechanisms to focus on the most relevant interactions: simple attention-based pooling [74] and multi-level attention as in Hi-Fi Ark [34], which applies attention first to article content and then across the user's history.

A further advance is the use of Transformer-based encoders (see Subsection 2.1), which capture global dependencies across all interactions. For example, NRMS [75] applies multi-head self-attention to learn which past articles are most relevant to each other and to the current candidate article.

Most existing models compress user interests into a single embedding, potentially oversimplifying diverse reading preferences. Multi-interest methods like MINS [70] and PENR [67] overcome this by learning multiple latent interest vectors per user, with each interest vector representing a topic the user is interested in. They compute similarity between each interest vector and the candidate article embedding, then aggregate scores via max- or attention-based pooling. Alongside user interests, the dynamic popularity of news articles represents another important modeling target.

2.5.3. Popularity Modeling

In news recommendation, article popularity plays a central role due to the short lifecycle of news items and the concentration of clicks on a few trending articles. Most deep NewsRec models capture this only implicitly via click data rather than using live popularity statistics as explicit input features. Only a few models incorporate popularity directly [50]: PENR introduces popularity-enhanced user representations [67], and PP-Rec explicitly combines personalized matching with a time-aware popularity score based on real-time click-through rate (CTR), recency, and content [46]. However, explicit integration of dynamic popularity into embedding spaces remains underexplored, despite its potential for improving accuracy. At the same time, over-reliance on popularity can amplify bias and reduce diversity, motivating hybrid approaches that balance popularity with content and user features.

These results establish the baseline context and motivate our subsequent focus on dataset preparation and feature engineering.

3. Dataset and Feature Engineering

In this chapter, we introduce the structure and key properties of the EB-NeRD dataset used in this thesis. We begin by describing the dataset format, including the logged user-article interactions, article metadata, and precomputed popularity information. In the following section, we present descriptive statistics to analyze patterns in user behavior and article engagement. Motivated by the observed challenges in the data we then describe our feature engineering approach. This includes content-based, temporal, popularity, and interaction features, each designed to help the model generalize across different scenarios. Finally, we explain how we construct individual user sessions by splitting click sequences into fixed-sized blocks.

3.1. EB-NeRD dataset

The EB-NeRD dataset [12] contains six weeks of user interaction data and articles from the Danish newspaper *Ekstra Bladet*. It was released as part of the RecSys24 Challenge [31] and is provided in train, validation, and test splits. Since clicked labels in the challenge test set are withheld, we resplit the validation set into our own validation and test splits.

The dataset comprises three main tables. The *history* table records, for each user_id, the sequence of clicked article_ids with timestamps and read times. The *articles* table stores metadata such as title, body text, category_id, publication time, and sentiment score (computed via a pretrained model). The *behaviors* table contains impression logs, listing the articles shown to the user (*inview*) and those actually clicked. Example rows are shown in Appendix A.2. In total, EB-NeRD comprises about 125k unique articles, 1.1M users, and 251M user–article interactions [12]. To contextualize these interactions, we next present a descriptive analysis of user click dynamics over time and across articles.

3.2. Dataset Analysis

Before building a recommendation model, it is important to understand how users interact with articles in the dataset. In this section, we look at how long articles stay relevant and how their popularity changes over time. This helps us understand key challenges like the short article lifecycle, the cold-start problem, and the long-tail distribution of relative popularity.

3.2.1. Article Lifecycle

The news domain is characterized by the constant release of new articles, while older content quickly loses relevance. Figure 3.1 shows this dynamic: articles peak in popularity

about 10 minutes after publication, then follow an exponential decline in engagement. This illustrates the short-lived nature of news consumption, where user attention rapidly shifts to newer content.

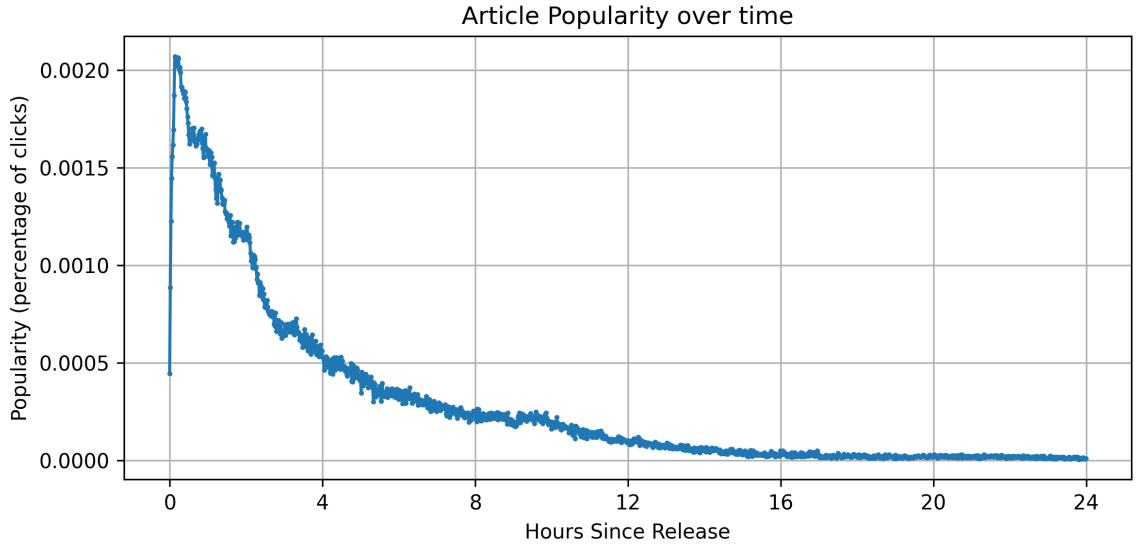


Figure 3.1.: The lifecycle of an article in the EB-NeRD dataset, showing a rapid rise followed by an exponential decline in relative popularity over time.

Such temporal sensitivity means that an article relevant at one point in time quickly loses relevance. It also exacerbates the cold start problem, as articles are most relevant early in their life when little interaction data exists. We define an article's lifecycle as the time from release until it accumulates 90% of its total clicks (see Appendix A.2.1). The median lifecycle is 5.3 hours, though this varies across types—for example, breaking news decays faster than opinion pieces (see Figure A.4.1). This highlights the importance of temporal features such as article age and content signals to estimate an article's current stage.

3.2.2. Long-tail Distribution of Relative Popularity

Relative popularity captures how article clicks are distributed within a 15-minute sliding window at any given time (see Def. A.2.1). Figure 3.2 presents the corresponding Lorenz curve, illustrating the characteristic long-tail behavior in which a small fraction of articles accounts for the majority of clicks.

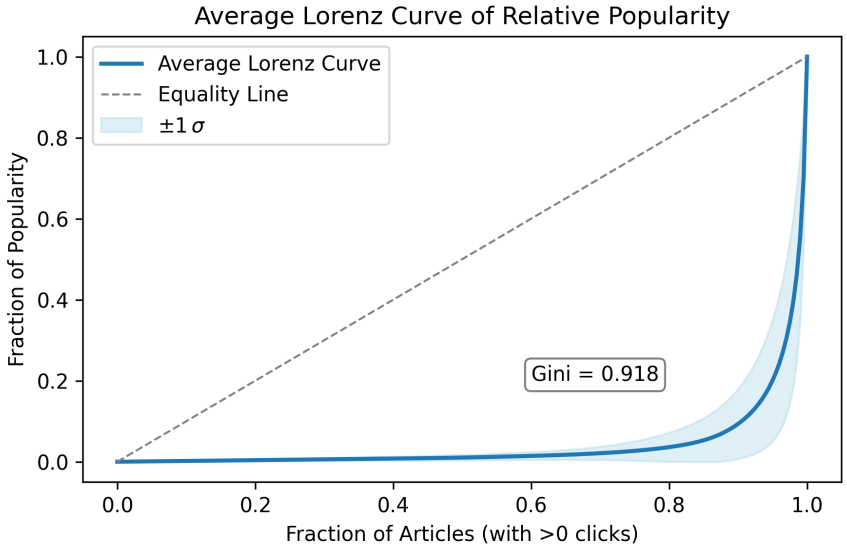


Figure 3.2.: Lorenz curve of relative article popularity. The x-axis shows the fraction of articles, while the y-axis shows their cumulative share of clicks within 15 minutes. Articles with zero clicks in a given time window are excluded. The curve is averaged over all timesteps, and the shaded area shows $\pm 1\sigma$ across timesteps, reflecting temporal variability in popularity inequality. .

This inequality (Gini index of 0.918) is largely driven by front-page exposure, where 71% of all clicks originate [13]. On average, the five most popular articles at the recommendation time account for 42.5% of the total clicks. This long-tail distribution is a central challenge for news recommendation: while a few articles dominate user interactions, the majority receive little attention, making it harder for models to learn meaningful representations and recommend beyond the most popular items. These insights directly informed our feature engineering strategy.

3.3. Feature Engineering

Although our datasets is static, we restrict our feature engineering to information that would be reasonably available in a live production system. The features used in our model can be grouped into four categories: content-based, temporal, popularity, and interaction features. Each category addresses a specific challenge in news recommendation. Content-based features help mitigate the cold-start problem. Temporal features like article age, approximate the current stage in the article's lifecycle. Popularity features reflect an

article's current success. Lastly, interaction features capture the context in which clicks occurred, providing insight into user behavior and intent.

3.3.1. Content Features

Content features include text embeddings, image embeddings, and metadata such as category or text sentiment. These features let the model learn which topics a user is interested in.

Text Embeddings We embed the title and body of each article using the pretrainend model `bert_base_multilingual_cased`[1]. Since the Ekstra Bladet[2] and BNN[3] websites display only the title and the title image on their front pages, we include body embeddings to capture deeper contextual information and title embeddings to represent the headline text shown to users.

Image Embedding We use the precomputed image embeddings from the EB-NeRD datasets artifacts [12], as the raw images were not available due to copyright concerns. The dataset documentation does not specify which model produced these embeddings.

Article Metadata The article's premium status and topic category are the most important metadata features. In addition to the deep embeddings described above, we incorporate additional text features, such as body length, sentence counts, punctuation frequencies, etc., to capture complementary syntactic and stylistic features. We use logarithmic transformations and robust scaling for heavily skewed distributions.

3.3.2. Temporal Features

Temporal features describe when an interaction occurred or how much time has passed since an article was published. These features help the model understand changes in article relevance and user behavior over time.

Age Our previous analysis shows that an article's relevance changes substantially over its lifecycle (see Figure 3.1). Therefore, the age of an article is a key feature in our model. We compute the age t in minutes since its release timestamp. To normalize this feature, we first apply a logarithmic transformation and then standardize it to have zero mean and unit variance:

$$\text{age} = \frac{\log(1+t) - \mu}{\sigma}, \tag{3.1}$$

where μ and σ are the mean and standard deviation of $\log(1+t)$ computed over all clicked articles in the training set. This transformation captures both the rapid early dynamics and the slower long-term decay in article relevance.

[1] https://huggingface.co/google-bert/bert-base-multilingual-cased
[2] `https://ekstrabladet.dk/`
[3] `https://bnn.de/`

Time The time of interaction (hour of day, weekday) can change user preferences: for example, users might prefer short articles on a weekday morning to catch up, while on a weekend evening they have time for longer, more in-depth articles [44]. To account for this, we include the current timestamp for each interaction to encode it as an input later.

3.3.3. Popularity Features

At any given point in time most clicks concentrate on a few very popular items (see Figure 3.2). Therefore the current popularity of each article is a very important input signal. We first build a per-minute article-by-time matrix $P_{1\,\text{min}}$ by binning each click to the floor of its timestamp and counting impressions per (article_id, minute) pair into a SciPy sparse matrix. To efficiently obtain smoothed popularity over longer windows (5 min, 15 min, 1 h, 2 h, 4 h, 24 h), we apply a sliding-window sum implemented as sparse matrix multiplications over the previous 5, 15, 60, ... columns of $P_{1\,\text{min}}$ (see Appendix A.3). To counteract the long-tail distribution, we apply logarithmic normalization as in equation 3.1 to each time window.

3.3.4. Interaction Features

Interaction features capture the context in which a user engages with an article. These features include the timegap between subsequent interactions. This captures which articles were viewed in a single sitting and after which the user left the platform. Additionally, we use the `readtime` column from user history to record how long the user spent reading the article.

A full list of our features and the normalization can be found in the Appendix A.5. The content-based features are defined at the article level, while interaction-level features are associated with each click in a user's history. For training, user histories are segmented into sessions.

3.4. Session Splitting

Following the sequential recommendation paradigm introduced in Subsection 2.4.2, we cast news recommendation as a next-item prediction task. To prepare training data for this setting, user histories are segmented into sessions of past interactions, from which session-target pairs are constructed. For each user $u \in \mathcal{U}$, the interaction history is defined as a sequence $H_u = (i_{u,1}, i_{u,2}, \ldots, i_{u,n_u})$, where n_u is the number of interactions for user u. Each interaction is represented as a tuple $i_{u,j} = (a_{u,j}, f_{u,j}, \tau_{u,j})$, with a denoting the clicked article, f the associated interaction features (e.g., dwell time), and τ the timestamp. Applying a sliding window of length w over H_u yields training examples of the form (s, y) with $s = (i_{u,k}, \ldots, i_{u,k+w-1})$ and $y = i_{u,k+w}$ (see Figure 3.3). Aggregating all such examples across all users defines the training corpus

$$\mathcal{D} = \{(s, y) \mid s \subset H_u,\ y \in H_u,\ u \in \mathcal{U}\}$$

.

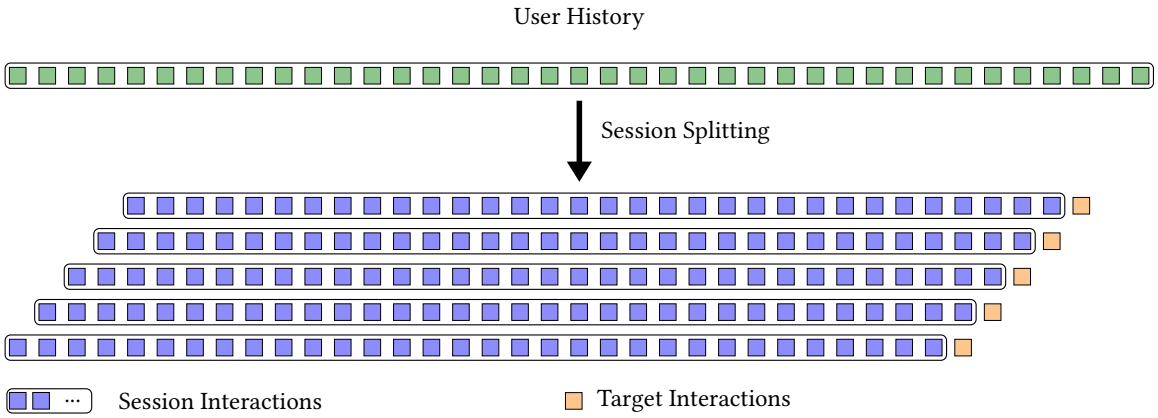


Figure 3.3.: Illustration of session splitting. User histories are segmented into sessions of past interactions s, with the subsequent click i_{k+w} serving as the target.

For each training example with target article $a_{u,i}$, we construct an exclusion mask of nearby clicked articles in the same user history. This mask contains the five preceding articles, the target itself, and the five subsequent articles around the target position.

$$\mathcal{F}_{u,i} = \{\, a_{u,j} \mid \max(1, i-5) \leq j \leq \min(n_u, i+5) \,\}. \tag{3.2}$$

During negative sampling, we apply this mask as a post-processing step by removing $\mathcal{F}_{u,i}$ from the negative pool (see Subsection 4.4.2). With the preprocessing complete, we are now ready to conduct experiments and present our approach. With these preprocessing procedures complete, the dataset is now suitably structured to serve as input for the proposed model.

4. Approach

In this chapter, we first introduce the overall design of our model and motivate the key architectural choices. We then describe the individual components in detail, followed by an explanation of the training process. Finally, we discuss modifications and alternative training setups that were explored to improve robustness and mitigate embedding collapse.

4.1. Model Overview

Based on our data analysis, literature review, and discussions with BNN, we have identified the following key challenges that our model must address.

ID	Requirement	Description
R1	Low Latency	The system must deliver predictions in real time, even when operating over an large and frequently updated article catalog.
R2	Short Article Lifecycle	New articles are constantly introduced while older ones become obsolete. Articles encountered during evaluation are often unseen during training, requiring strategies that generalize to unseen content.
R3	Personalization	Recommendations should be tailored to user preferences, behavior history, reading patterns, and contextual factors.
R4	Feature Integration	The model must effectively integrate multiple feature types, including textual embeddings, contextual metadata, popularity signals, and article age.

Table 4.1.: Key requirements identified for our news recommendation system.

These requirements were the starting point for choosing a suitable model architecture.

Model Architecture To satisfy the latency requirement (R1), we designed SNeRT (Session-based News Recommender via Transformer), a TTM (see Subsection 2.4.1) that computes separate embeddings for users and articles. Their relevance is scored via dot-product similarity, enabling precomputation and thus low-latency ranking even with large catalogs. Cross-Encoders (CEs) promise higher accuracy by jointly encoding users and candidates, but their need for a forward pass per pair makes them impractical for our main task (RQ1). A detailed comparison of TTMs and CEs is given in Section 2.4.1. Our TTM consists of

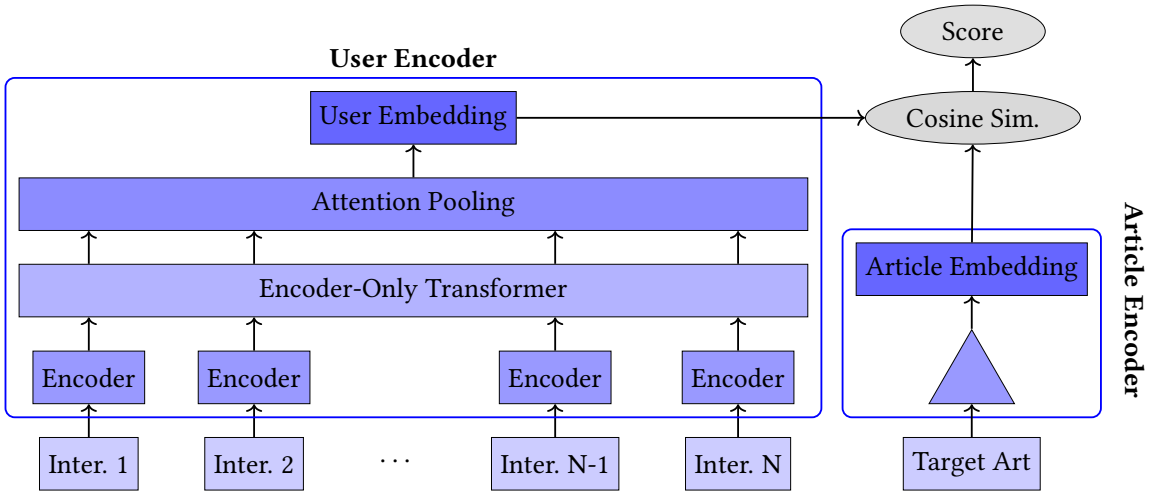


Figure 4.1.: Our Two-Tower framework, featuring distinct user and article encoding towers.

two independent encoders: a user encoder and an article encoder. To capture the article lifecycle (R2), the article encoder embeds articles with minute-level temporal precision. To personalize recommendations (R3), the user encoder takes the session as input and uses a transformer-based architecture to produce a user embedding. During training, we apply contrastive learning (CL), aligning the user embedding with the positive (target) article embedding while separating it from sampled negative article embeddings to learn a shared representation space. In the following sections, we break down the article encoder, user encoder and training process in greater detail.

4.2. Article Encoder

The article encoder consists of a simple multi-layer perceptron (MLP) that maps a concatenation of content, metadata, temporal, age, and popularity features into an article embedding vector. We use a three-layer MLP that projects the input to the target embedding size and applies L^2-normalization in the final step to produce a unit-length vector.

A key challenge in this setup is integrating a diverse set of features (R4) with varying relevance to the recommendation task. To control the influence of each feature type, we apply noise and dropout individually to each of them. This allows, for example, the application of a higher dropout rate to strong predictors such as article popularity, thereby discouraging the model from relying too heavily on these features.

The input features can be grouped into three distinct categories, each requiring a different processing approach.

Deep Embeddings and Metadata Text embeddings and metadata features are precomputed during pre-processing using a pretrained BERT encoder. We did not fine-tune BERT (see Subsection 2.2.1), keeping the model compact and enabling extensive hyperparameter

search due to lower training time. Newspaper-assigned category and premium status are represented as learned embeddings.

Time embeddings Although we initially experimented with the Time2Vec encoding [28], we removed it to avoid allowing the model to overfit absolute timestamps and clicking patterns tied to specific publication times. Instead, we encode temporal context using simple cyclical features (sine and cosine of hour, day and weekday), to capture periodic user habits without exposing the exact dates. This is especially important because or testing data is from a different time-period than the training data.

Age and Popularity The age of the article and popularity features are loaded from our preprocessed trainings data, log transformed and then normalized according to our description in the feature engineering part. Applying a logarithmic transformation reduces the skewness of the long-tail distribution in article popularity (see Figure 3.1), yielding a distribution that more closely resembles a normal distribution. Including age and popularity features lets the article encoder capture the current point in the article lifecycle.

Having defined how articles are encoded, we now describe how these representations are used in the user encoder.

4.3. User Encoder

The user encoder is responsible for transforming the past user interactions (session) into a single fixed-size embedding. Each interaction consists of article, context, and positional features. We first encode these components into a unified interaction embedding. To extract long- and short- term preferences from this interaction embedding sequence we use an encoder-only transformer stack. The output of this encoder is then aggregated with attention pooling (see Figure 4.1).

4.3.1. Interaction Encoder

Each past interaction in a session is embedded individually. To calculate the final embedding for each click, we sum the article embedding, context embedding, and positional embedding (see Figure A.4.2 in Appendix). The resulting vectors form a fixed-length sequence, that is then fed into the transformer encoder.

Article Embedding The same article encoder is used to encode both the articles from past interactions and the target articles, with shared weights across both cases. This weight tying, a technique commonly used in NLP [22, 45], serves as a form of regularization and improves the representation of long-tail items in RecSys, as they receive training signals both when appearing in the session history and as target articles [60].

Context Embeddings The context embedding captures the circumstances surrounding each article click, providing the model with additional information about the conditions under which a user clicked the article. For this dataset we encode the timegap between

interactions and the readtime by first bucketizing these features, the resulting index are used to lookup embeddings in a trainable embedding table. Additional context features like the device and user location could be added here as well, but were not present in our dataset. All these individual embeddings are added to get the context embedding.

Positional Embedding The fixed sequence length allows us to use learned positional embeddings (see Subsection 2.1.2). This choice was made to mimic other encoder-only models with fixed sequence length like BERT [9].

4.3.2. Encoder Stack

At the core of our user encoder lies a stack of Encoder-only Transformer layers. This architecture was chosen because transformers are known to outperform other sequence modeling techniques on many tasks [9, 65], including recommendation (see Subsection 2.4.2). We use multihead attention (see Subsection 2.1.1) so each head can focus on different aspects of the sequence. The output of the transformer is a sequence of embeddings, to get one user embedding we need to apply a pooling method.

4.3.3. Attention Pooling

To get a single user embedding we use attention pooling, which we adapted from a different news recommendation system to our needs [77]. For attention pooling, we introduce a single trainable query vector $q \in R^d$ that scores each position in the transformer-encoded sequence. Given hidden states $H = [h_1, \ldots, h_L] \in \mathbb{R}^{d \times L}$, we compute attention weights by applying a softmax over the dot-product scores $H^\top q$, and form the session embedding as

$$u \;=\; H\,\mathrm{softmax}\big(H^\top q\big). \tag{4.1}$$

This is a variation of dot-product attention (see Subsection 2.1.1) with a single query vector. Our initial test used mean-pooling, which led to an already collapsed user embedding (see Subsection 2.3.3) at the start of the training run. Attention Pooling lets the model learn how to weigh the interactions of the user encoder. We finally L^2-normalize u before similarity matching. With the user and article representations defined, we next describe the training procedure that aligns them in a shared embedding space.

4.4. Training

Our TTM model is trained to maximize the dot product between matching user embeddings and article embeddings, while minimizing the similarity to the sampled negative articles. This encourages the model to position relevant articles closer to the user in the shared embedding space. To achieve this, we adopt the InfoNCE-loss (see Subsection 2.3.1). An important aspect of effectively applying the InfoNCE loss is how training batches are constructed, since the choice of negatives directly shapes the embedding space. We therefore next describe our batch construction strategy.

4.4.1. Batch Construction

We chose a batch size of 512 based on early experiments balancing model stability and convergence speed. Initially, we explored using in-batch negatives by sampling sessions from a narrow time window. However, due to the concentration of clicks on a few highly popular articles, batches became overly correlated, resulting in poor convergence. Sampling sessions randomly across the entire training period resolved the issue of highly correlated batches caused by narrow time windows. However, due to the temporal dependence of article embeddings, standard contrastive learning strategies like in-batch negatives or memory banks proved ineffective, as they use negative samples from mismatched time periods. To address this, we instead sample negatives individually for each data point using a combination of strategies.

4.4.2. Negative Sampling

Negative sampling is a crucial component of contrastive learning (see Subsection 2.3.2) and must be tailored to the characteristics of news recommendation. In our setting, an effective strategy should consider only articles published at recommendation time, provide negatives that challenge but do not overwhelm the model, ensure coverage of the entire catalog while emphasizing hard examples, and remain fully vectorizable to avoid pipeline bottlenecks.

For each training example with target time t_{rec}, a negative sample a^- is an article assumed not to be relevant to the user. Negatives are drawn from distributions defined over the set of articles available at that time $\mathcal{A}_{t_{\text{rec}}} = \{\, a \mid \text{pub}(a) \leq t_{\text{rec}} \,\}$. We employ four complementary strategies to construct these negatives:

1. **Uniform random.** Negatives are sampled uniformly from the entire availability set, $a^- \sim \text{Unif}(\mathcal{A}_{t_{\text{rec}}})$. This ensures catalog-wide coverage and prevents the model from ignoring rarely clicked items.

2. **Popularity-based.** Negatives are sampled proportionally to their relative popularity at time t_{rec}, $a^- \sim \text{Cat}(p_{t_{\text{rec}}}(a))$. Here $p_{t_{\text{rec}}}(a)$ is the relative popularity computed over a sliding window $(t_{\text{rec}} - \Delta,\, t_{\text{rec}}]$ (see Appendix A.2.1). This strategy emphasizes harder negatives from articles that are frequently clicked.

3. **Recency-based.** To oversample fresh content, we restrict candidates to the N most recently published articles, $\mathcal{A}^{(N)}_{t_{\text{rec}}} \subseteq \mathcal{A}_{t_{\text{rec}}}$. Negatives are then drawn uniformly from this subset, $a^- \sim \text{Unif}(\mathcal{A}^{(N)}_{t_{\text{rec}}})$, which balances coverage of new but less-clicked items (e.g., $N = 4000$).

4. **In-view.** Negatives are taken from the set of articles displayed to user u at time t_{rec} but not clicked. Formally, $a^- \in \mathcal{V}_{u,t\text{rec}}$ where $\mathcal{V}_{u,t_{\text{rec}}}$ denotes the set of all articles shown to user u at time t_{rec}. This provides true negatives grounded in the actual recommendation context.

The overall negative sampler can be expressed as a mixture distribution that combines the four strategies introduced above:

$$a^- \sim \lambda_{\text{unif}} \,\text{Unif}\big(\mathcal{A}_{t_{\text{rec}}}\big) + \lambda_{\text{pop}} \,\text{Cat}\big(p_{t_{\text{rec}}}\big) + \lambda_{\text{rec}} \,\text{Unif}\big(\mathcal{A}^{(N)}_{t_{\text{rec}}}\big) + \lambda_{\text{inv}} \,\text{Unif}\big(\mathcal{V}_{u,t_{\text{rec}}} \setminus \{a^+\}\big),$$
$$\text{with } \lambda_{\text{unif}} + \lambda_{\text{pop}} + \lambda_{\text{rec}} + \lambda_{\text{inv}} = 1, \quad \lambda_* \geq 0. \tag{4.2}$$

As a post-processing step, we filter out potential false negatives from the sampled pool by applying the exclusion mask of false negatives defined in Equation 3.2. Given an initial sample set $\tilde{\mathcal{N}}_{t_{\text{rec}}}$ drawn from the strategies above, the final negative pool is obtained as

$$\mathcal{N}_{t_{\text{rec}}} = \tilde{\mathcal{N}}_{t_{\text{rec}}} \setminus \mathcal{F}_{u,i}. \tag{4.3}$$

While careful batch construction and this post-processing step mitigate issues arising from the temporal dependence of article embeddings, they do not address a more fundamental challenge we encountered: embedding collapse.

4.5. Mitigating Embedding Collapse

Our initial test runs showed very poor performance on our core metrics, and the embedding similarity scores and the effective rank of the user embeddings pointed to an embedding collapse issue (see Subsection 2.3.3) as the likely cause. In this section we describe the different techniques we implemented and evaluated to solve the embedding collapse problem.

4.5.1. Problem Overview and Initial Fixes

Embedding collapse is a well-known issue in Two-Tower Models. It limits both the scalability and performance of the architecture and is cited as the key issue holding back scalability and performance of recommendation systems [16] (see Subsection 2.3.3). If the user embeddings collapse to a single point in the latent space (i.e., total collapse), the model effectively degenerates to a small MLP in which only the article embeddings encode useful information. Even a partial collapse, where the embeddings lie in a low-rank subspace, severely limits the expressiveness of the user representation and results in poor performance across all evaluation metrics. While embedding collapse is common in RecSys, in our case it was so severe that the model underperformed even simple baselines.

Based on experimentation and a review of related literature, we identified three key factors contributing to degraded representation learning:

1. **Insufficient normalization of long-tail item features**, which amplifies the embedding collapse problem [52].

2. **Simplicity bias from popularity features**, causing the model to rely on popularity while neglecting more complex user–item interaction patterns [81].

3. **High variance in user histories**, as front-page placement can be a stronger predictor than individual user history for the next target article. This variance acts like overly strong data augmentation in contrastive learning and, as noted by [26], can induce embedding collapse when the model learns to ignore noise instead of meaningful user signals.

To address the first issue, we reworked our feature preprocessing and embedding strategies to better capture the long-tail distribution of article age and popularity by applying a logarithmic transformation (see Section 3.3). Additionally, we replaced mean pooling in the transformer output with attention pooling (see Section 4.3.3), as mean pooling resulted in an already partially collapsed state at the start of training. To combat the simplicity bias, we experimented with applying heavy dropout and/or noise to the popularity features during training to force the model not to over-rely on these features.

We also tried different hyperparameter settings for the InfoNCE temperature and varied the composition and number of negative samples. While these changes showed slight improvements, the resulting models underperformed basic baselines. We then focused on adapting known techniques from related contrastive learning setups to our use case and evaluated their impact on embedding collapse and recommendation metrics.

4.5.2. Exploratory Attempts

Before arriving at spectral regularization as a more direct and effective solution, we explored several alternative strategies that attempted to mitigate embedding collapse indirectly.

Momentum Contrast (MoCo) We hypothesized that embedding collapse was caused by an imbalance in convergence speed: the article encoder quickly overfits to recommending the most popular items, whereas the user encoder must learn more complex relationships. To mitigate this, we adapted the momentum update mechanism from [20], slowing the article encoder's convergence and providing the user encoder with a more stable training target. Although MoCo prevented total embedding collapse, the representations remained low-rank and performance gains were marginal. We therefore explored strategies aimed at directly learning user embeddings without the long-tail distribution of article popularity interfering.

Session Augmentation Loss To encourage the user encoder to learn representations that are independent of simple signals such as popularity, we explored adding a secondary contrastive learning objective between sessions directly (see Figure 4.2).

Prior work has proposed two main approaches: the first generates two augmented views of the same session through techniques such as sequence cropping, item masking, semantic replacement, and subsequence shuffling [78, 85]. The second approach treats different sessions that lead to the same target article as positive pairs [47], with all other sessions in the batch serving as negatives. We implemented both variants (a description of the session augmentation pipeline is provided in Appendix A.1). However, our experiments showed that this secondary loss did not resolve the embedding collapse problem in our use

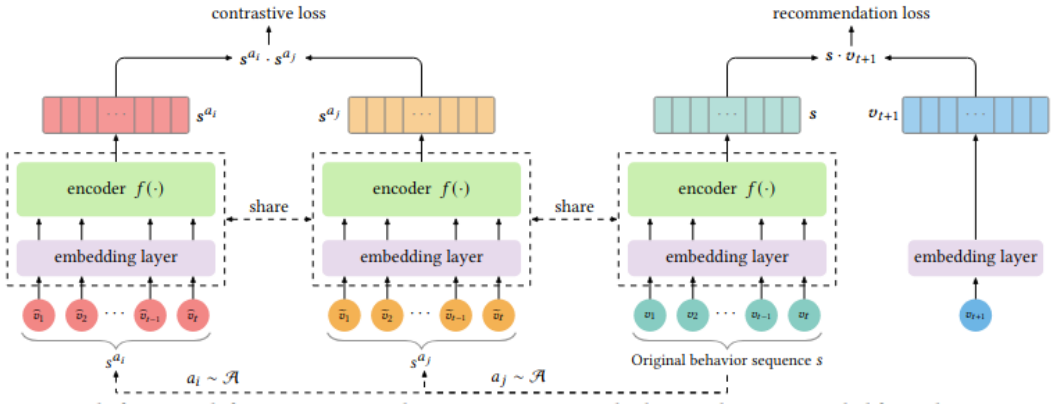


Figure 4.2.: Session-level contrastive loss setup (adapted from CL4SREC [78]). A secondary contrastive objective is added alongside the main loss to stabilize the latent space.

case. On the contrary, it made training less stable, as the two loss functions interacted and caused oscillations during training. We therefore turned to alternative solutions targeting the user encoder directly.

Modifying Attention Another explanation for embedding collapse is the interaction-collapse theory [80]. It suggests that when long-tail features—which are undertrained and exhibit high variance—interact with other features, the model treats this variability as noise. This effect resembles overly strong data augmentation, causing the model to ignore these features entirely and fall back into a low-rank representation. As a result, the embedding encodes less meaningful information. This problem is particularly severe in large-scale recommendation tasks, where long-tail features occur frequently and their limited representational capacity constrains overall model performance [80]. To mitigate this, we implemented the collapse-avoiding attention mechanism proposed in [80]. The approach computes the average modulus length of interaction embeddings $\{e_i\}_{i=1}^{n}$ within each batch as threshold thr,

$$\text{thr} = \lambda_{thresh} \frac{1}{n^2} \sum_{i=1}^{n^2} \|e_i \cdot e_j^\top\|_2, \tag{4.4}$$

and applies a binary mask to filter out interactions whose scores fall below this dynamic threshold. The goal was to discard low-information long-tail interactions during attention computation and thereby preserve richer feature representations. However, after implementing this mechanism within the transformer layers and conducting a hyperparameter search over the threshold strength λ_{tresh} we found that it neither resolved embedding collapse nor improved the ranking metrics, while significantly slowing training times. Instead, we turned to spectral regularization losses, which directly penalize low-rank representations.

4.5.3. Spectral Regularization Losses

We implemented and tested two different spectral regularization losses. The first is our own implementation, inspired by the DirectSpec paper [41] and the definition of effective rank (see Subsection 2.3.3). The second is the Log-Determinant Loss adapted from graph collaborative filtering [86].

Singular Value Loss We defined a custom spectral loss that penalizes deviations from a uniform singular value spectrum, based of the singular values $\{\sigma_i\}_{i=1}^{d}$ of the embedding matrices:

$$\mathcal{L}_{\text{spec}} = \lambda_{\text{spec}} \cdot \left(\sum_{i=1}^{D} (\sigma_i - \bar{\sigma})^2 \right), \tag{4.5}$$

where $\bar{\sigma} = \frac{1}{D} \sum_{i=1}^{D} \sigma_i$ is the mean singular value of the embedding matrix, and λ_{spec} controls the strength of the regularization. This loss pulls the singular values together, which helps increase the effective rank of the embedding space. While this approach improved our core metrics, it proved highly sensitive to small changes in the training setup. Minor modifications such as adjustments in negative sampling or dropout values often required retuning the regularization lambda. To address this instability, we adopted a more robust alternative described in the literature.

Log-Determinant Loss A recent work in Graph Collaborative Filtering [86] identified a similar issue: user embeddings collapsing due to the long-tail distribution of recommended items. To address this, the authors proposed an alternative spectral regularization based on the Log-Determinant Loss. We translated this loss to our setting, where it is definend as follows:

$$\mathcal{L}_{\text{logdet}} = \lambda_{\text{logdet}} \left(\text{tr}(\Sigma_U + \Sigma_A) - \log \det(\Sigma_U \Sigma_A) \right), \tag{4.6}$$

where Σ_U and Σ_A denote the covariance matrices of the user and article embeddings, respectively. This formulation penalizes low-rank covariance embedding matrices, since the loss diverges to ∞ when the embeddings collapse into a lower-dimensional subspace. To quantify embedding space utilization, we employ the relative effective rank, which measures how much of the available embedding space carries information (see Appendix A.2.3).

Our experiments in Figure 4.3 show that stronger spectral regularization consistently increases the relative effective rank of the embeddings and therefore mitigates embedding collapse. However, enforcing higher spectral diversity introduces a trade-off with target alignment.

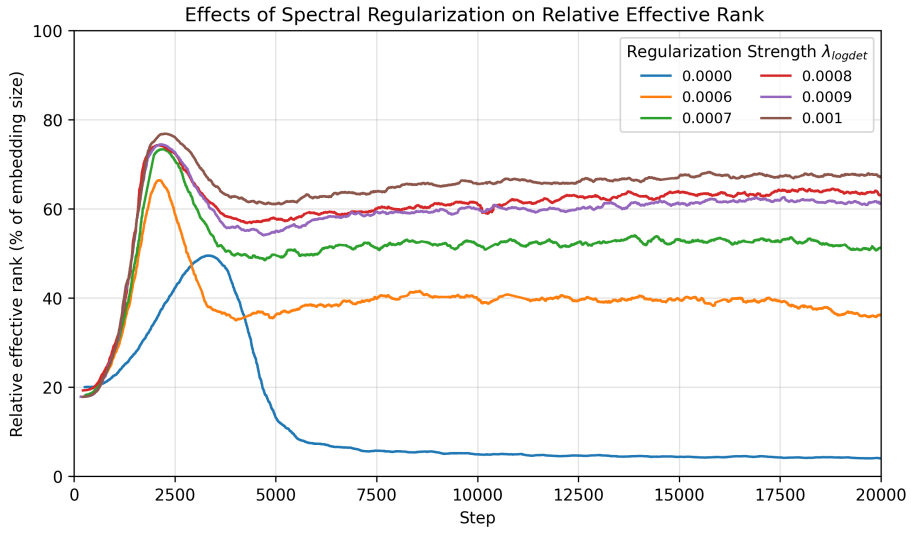


Figure 4.3.: Effects of regularization strength on the relative effective rank of user embeddings over the first 20k batches. Stronger spectral regularization mitigates embedding collapse by maintaining higher effective rank, whereas unregularized runs quickly collapse in erank.

Trade-off between Spectrum Regularization and Alignment As noted in the DirectSpec paper [41], spectrum balancing can conflict with the goal of aligning positive users with their target articles. Given user embeddings u_1 and u_2 which clicked the same article a. The alignment to our primary prediction loss maximizes the dot-product between them:

$$\langle u_1, a \rangle \to 1, \quad \langle u_2, a \rangle \to 1, \tag{4.7}$$

While the spectral regularization loss pushes user embeddings to an orthogonalized state, thereby minimizing the dot-product between them.

$$\langle u_1, u_2 \rangle \to 0. \tag{4.8}$$

These two objectives are contradictory: a single article vector cannot be simultaneously aligned with two orthogonal user vectors in Euclidean space [41]. As a result, increasing spectral regularization can weaken alignment quality. Thus, spectrum balancing is effective in preventing total embedding collapse, but its strength must be carefully tuned to avoid overshadowing the primary training objective. This theoretical trade-off is consistent with our experiments, in which extensive hyperparameter tuning was necessary to find a good balance.

Our experiments confirmed the usefulness of spectral regularization, as our test runs consistently outperformed the Most Popular baseline. Having established the methodological foundations and key design choices of our approach, we now turn to the experimental evaluation.

5. Experiments and Evaluation

In this chapter, we present the experimental setup, evaluation methodology, and results of our proposed model. We first describe the hardware, software, and data preparation steps, followed by the training and evaluation protocols. The remainder of the chapter is structured around answering our research questions (RQ1–RQ3). At the end, we discuss the insights gained from hyperparameter optimization, the exploration of different negative sampling strategies, and a set of ablation studies.

5.1. Experimental Setup

Hardware and Software We implemented the preprocessing and training pipeline in Python. For preprocessing, we used NumPy[1] and pandas[2], while the model was implemented using PyTorch. The BERT version used in our experiments is `bert-base-multilingual-cased`[3], which we obtained from HuggingFace. For live monitoring of our training, we used Weights & Biases (W&B)[4]. GPT-4o[5] was used as a programming and debugging assistant, as well as for improving grammar and spelling. To train our model we used NVIDIA A100 GPUs[6] from BwUniCluster3.0[7]. As CPU-bound data loading and negative sampling quickly became a bottleneck for our training setup, we executed multiple single-GPU training runs in parallel during hyperparameter optimization to make more efficient use of the hardware.

Training and Evaluation Protocol As described in Section 3.1, we construct user sessions from five consecutive weeks. Since impression data is only available for the final two weeks, we use only sessions with targets in the fourth week for training and split the subsequent clicks from the fifth week into validation and test sets. For our main evaluation, we use 100k impressions to compute the primary metrics, while a reduced subset of 10k impressions is used for hyperparameter optimization and the negative sampling study. Unless specified otherwise, we train the models with the setup defined in Table 5.1.

During training, we monitor Recall@20, NDCG@20 (global task), and AUC (in-view task) on the validation set for early stopping and model selection. All other metrics, including beyond-accuracy measures (diversity, novelty, serendipity, coverage, and category

[1] `https://numpy.org/`
[2] `https://pandas.pydata.org/`
[3] `https://huggingface.co/google-bert/bert-base-multilingual-cased`
[4] `https://wandb.ai/`
[5] `https://openai.com/index/hello-gpt-4o/`
[6] `https://www.nvidia.com/en-us/data-center/a100/`
[7] `https://wiki.bwhpc.de/e/BwUniCluster3.0`

Model Parameters		Training Parameters	
Embedding size	128	Optimizer	AdamW
Article encoder layers	3	Learning rate	1×10^{-4}
Transformer layers	4	Learning rate decay	Cosine decay
Attention heads	2	Warmup	5% of training steps
Spectral reg. λ	0.0008	Batch size	512
Session Length	16	Epochs	3
InfoNCE temperature	0.7	Training steps	60k

Table 5.1.: Default training configuration, grouped by model parameters and training parameters.

distribution), are computed only once during the final test phase. We use this training and evaluation protocol to answer our research questions.

5.2. Evaluating Research Questions

We address three main research questions (RQs) through targeted experiments with SNeRT, outlining the setup, evaluation metrics, and results for each. This allows us to assess performance across different recommendation scenarios and examine both ranking effectiveness and qualitative aspects.

5.2.1. RQ1: Global Recommendation Task

To evaluate the Global Recommendation Task, we first embed all articles available at the recommendation timestamp t_{rec}, ensuring that their age and popularity statistics reflect the state at t_{rec}. Using these embeddings, we retrieve the top-20 candidate articles via cosine similarity and measure ranking quality using Recall@20 and NDCG@20. The short article lifecycle of articles limits the effectiveness and relevance of standard baseline algorithms such as content similarity or matrix factorization [31, 69]. Instead, we use a *Most Popular baseline*, which chooses the articles that have received the most clicks in the last 15 minutes. This baseline is strong, since on average 42.22% of all clicks are concentrated on the five currently most popular articles (see Subsection 3.2.2).

Metric	Most Popular	SNeRT	Δ
Recall@20	0.707	**0.770**	+0.063
NDCG@20	0.299	**0.323**	+0.024

Table 5.2.: Performance comparison between the popularity baseline and our best model.

Performance Against Baseline Our best model, SNeRT, outperforms the popularity baseline on both ranking metrics: Recall@20 improves by 6.3 percentage points, ensuring

users are more likely to encounter relevant articles in the top-20 list, while NDCG@20 rises by 2.4 percentage points, reflecting more effective prioritization of relevant items. In contrast, exploratory experiments with the cross-encoder reranker (see Appendix A.1) showed strong overfitting to the training period and failed to deliver consistent gains over SNeRT, indicating that more advanced temporal modeling and robust reranking strategies are needed before a second-stage reranker becomes beneficial.

Latency Low-latency recommendations are essential for the Global Recommendation Task (RQ1). Since article and user embeddings are computed independently, article embeddings can be precomputed globally once per minute. This enables the construction of a global FAISS[8] index, which takes on average 8.52 ms to build. User embeddings change only after every interaction and can therefore be computed immediately after each click. With both user and article embeddings precomputed, the top-20 articles for each user can be retrieved as a simple FAISS lookup. This takes on average 4.47 ms on the CPU and is even faster on a GPU. This latency is short enough for real-time recommendation systems and leaves a time budget for an additional re-ranking stage.

5.2.2. RQ2: In-view Recommendation Task

We measured the AUC score for the in-view recommendation task to enable comparison with the RecSys24 results [31]. Since the models in that challenge encoded future information that is not available in realistic settings, we instead compare our results to those reported in the ablation study addressing this data leakage.

Metric	Most Popular	SNeRT	Our CE	Transformer CE	Ensemble Method
AUC	71.66	**72.22**	75.70	77.13	**82.20**

Table 5.3.: AUC scores for the Most Popular baseline, SNeRT, our transformer-based cross-encoder (CE), a reference transformer CE, and the best-performing ensemble method in the in-view recommendation task (RQ2) from RecSys'24 [31].

SNeRT improves only slightly over the Most Popular baseline (+0.56% AUC), confirming its strength lies in large-scale retrieval rather than fine-grained reranking. Introducing our second-stage cross-encoder reranker (see Appendix A.1) yields a more substantial improvement (+4.04% AUC), but still lags behind the structurally similar Transformer CE from RecSys'24, which demonstrates significant headroom for further optimization. Both cross-encoder variants, however, fall short of the ensemble method, which achieved the best overall performance in the challenge. This progression illustrates a well-established limitation of two-tower architectures: while highly efficient for retrieval, they consistently underperform in reranking tasks where direct item-item comparisons are critical [54, 61].

[8] https://github.com/facebookresearch/faiss

5.2.3. RQ3: Quality Evaluation

To assess recommendation quality, we evaluate category distribution alongside beyond-accuracy metrics.

Category Distribution RecSys24 models differ greatly in category exposure: the 5th-ranked hrec model recommends 50.1% auto-generated content, while the winner never recommends auto-generated content but heavily favors the "news" category [31]. Such biases are problematic, as newspapers aim to cover the full range of content categories. Motivated by this, we examined whether our model displays strong category biases. We computed the category distribution of the top-20 recommendations from the Global Recommendation Task (RQ1) and compared it with the Most Popular baseline (see Figure 5.1 for a visual comparison and Table A.6 in the Appendix for exact values). From Figure 5.1,

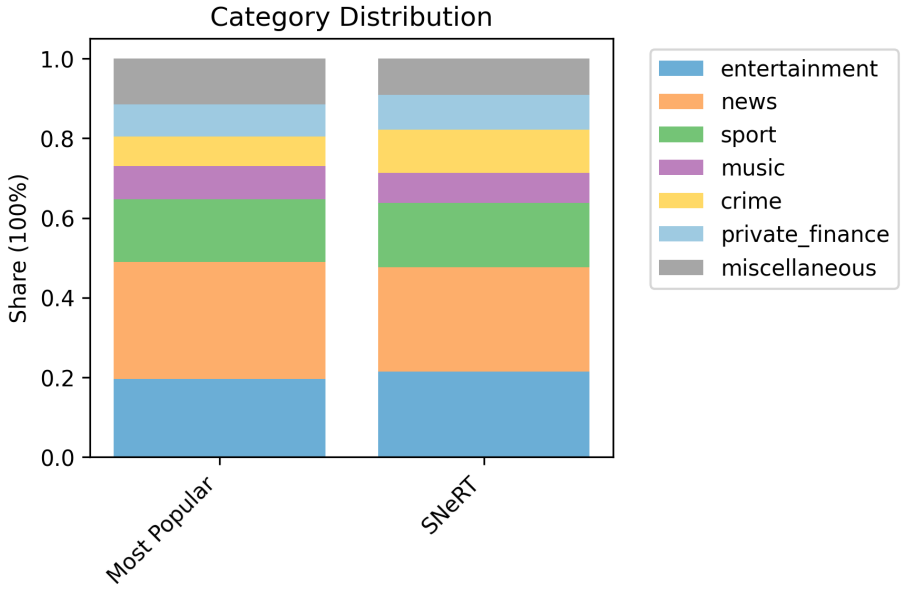


Figure 5.1.: Comparison of category distribution between our model and the Most Popular baseline.

it is evident that our model recommends a broad mix of categories. The only notable deviation from the baseline is a 46% relative increase in crime-related articles, while the remaining categories show only minor changes. This pattern remains consistent across different hyperparameter settings and negative sampling mixes. Overall, this suggests that SNeRT does not exhibit strong category biases that would pose a problem in a production setting.

Beyond-Accuracy Metrics To further assess the recommendation quality of our model, we adapt the beyond-accuracy metrics from the RecSys'24 Challenge to our global recommendation task [31]. In the challenge, these metrics were computed on a small, non-public subset of articles; in our case, we calculate them over the entire catalog using BERT-based

embeddings. Consequently, our values are not directly comparable to the official challenge results and are instead evaluated relative to the Most Popular baseline. The beyond-accuracy evaluation includes four metrics. Diversity (ILD) measures the dissimilarity among the top-K recommended articles for a user, encouraging varied content within the recommendation list. Serendipity measures how different the recommendations are compared to the user's reading history, promoting relevant but unexpected articles. Novelty rewards recommending less popular items to avoid over-representing highly clicked content and is calculated with respect to global popularity over the entire evaluation period. Coverage measures the proportion of the catalog exposed across all users, promoting balanced content exposure. The exact formulas are described in the Appendix A.2.2.

Metric	Most Popular	SNeRT	Relative Change (%)
Diversity	0.037	0.040	+8.11
Serendipity	0.043	0.045	+4.65
Novelty	13.133	13.203	+0.53
Coverage	11.31%	13.81%	+22.09

Table 5.4.: Beyond-accuracy metrics comparing the Most Popular baseline and SNeRT.

Table 5.4 shows that SNeRT outperforms the Most Popular baseline across all beyond-accuracy metrics. Gains in Diversity (ILD) and Serendipity are modest but indicate slightly more varied and unexpected recommendations compared to the heavily concentrated lists of the baseline. Novelty is also marginally higher for SNeRT, suggesting reduced bias toward globally popular items. The largest relative improvement is in Coverage, where SNeRT exposes a broader portion of the catalog to users. Interestingly, SNeRT achieves these improvements while also increasing accuracy, even though recommender systems often face a trade-off between precision and beyond-accuracy goals such as diversity or coverage [17]. This indicates that our approach not only balances these competing objectives but can, to some extent, overcome this usual trade-off. Overall, these results indicate that SNeRT not only improves accuracy but also produces more balanced, diverse, and less popularity-skewed recommendations compared to the Most Popular baseline.

5.3. Model Tuning

Our approach involves several hard-to-tune hyperparameters, such as the negative sampling mix, spectral regularization strength, and embedding size, which make a full grid search computationally infeasible. In preliminary exploratory runs, we first identified suitable values for batch size, learning rate and weight decay in order to establish a stable training setup. Based on these results, we defined a baseline model configuration (see Table 5.1), which we then used to systematically evaluate different negative sampling compositions. In a second step, we applied a Manhattan search over the other hyperparameters.

5.3.1. Negative Sampling

Model performance is highly sensitive to the choice of negative sampling strategy. We combined four approaches (see Subsection 4.4.2) in varying proportions and evaluated different mixtures to identify the most effective composition.

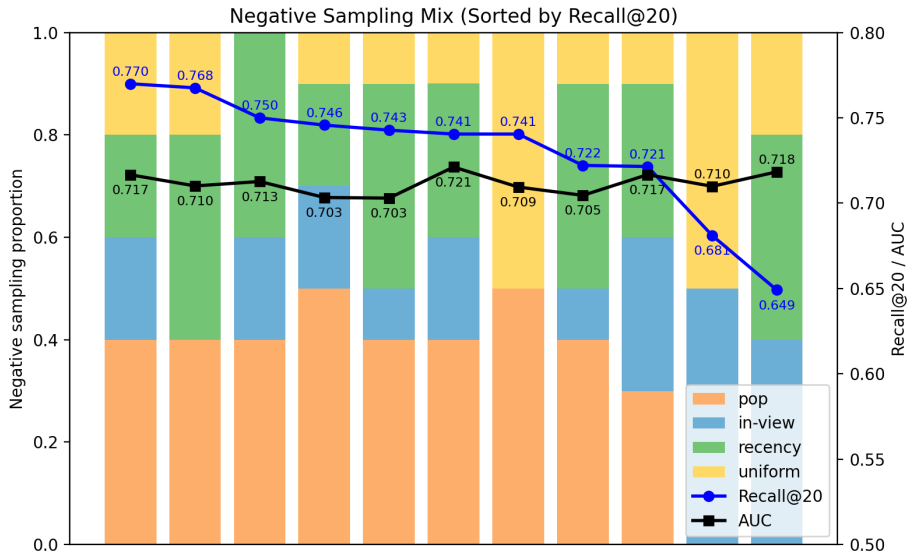


Figure 5.2.: Recall@20 and AUC for different negative sampling mixtures. The results highlight the strong influence of negative composition on model performance.

Since our optimization focused primarily on RQ1, we adopted a negative sampling mix of 40% popularity-based, 20% recency-based, 20% in-view, and 20% uniform random samples, which we then used for subsequent hyperparameter optimization. We further analyzed and interpreted how the different sampling strategies contributed to the final results.

Influence of Sampling Strategies From Figure 5.2 we conclude: popularity-based sampling is essential for achieving high recall, since it ensures that frequently clicked articles are oversampled as negatives. This effect would normally be provided automatically by in-batch negatives, but our time-dependent article encoding required explicit negative sampling. Recency-based sampling further improves recall by exposing the model to fresh articles that would otherwise be underrepresented due to lower click counts. In-view negatives only marginally increase recall for RQ1, but contribute more strongly to AUC in RQ2, reflecting their role as hard negatives that improve ranking quality rather than retrieval breadth. Finally, a small proportion of randomly sampled negatives helps the model learn to ignore the long tail of irrelevant articles.

5.3.2. Manhattan Search

Having fixed the negative sampling composition as part of our baseline setup, we next turned to the remaining hyperparameters that govern model architecture and regulariza-

tion. Since jointly optimizing all parameters would be prohibitively expensive, we adopted a Manhattan search strategy to systematically explore their individual effects.

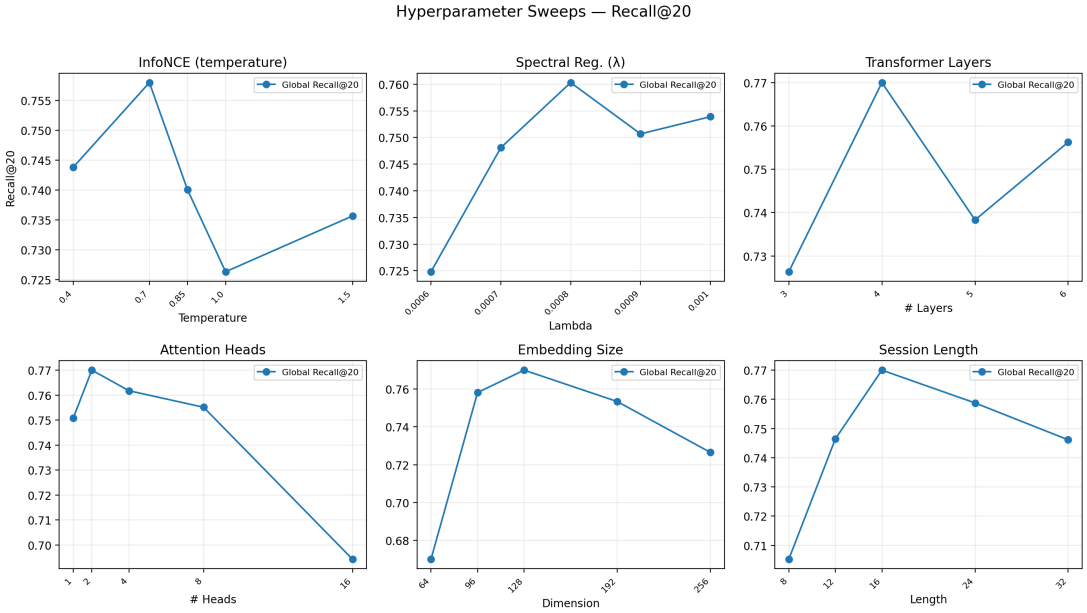


Figure 5.3.: Hyperparameter ablation study using a Manhattan search strategy: each subplot varies one parameter at a time while holding others fixed, showing its effect on Recall@20.

From Figure 5.3, we observe that our initial baseline model could not be substantially improved through a simple Manhattan search. This confirms our broader experience that training the TTM is highly sensitive: small changes can destabilize training and invalidate previously tuned settings. As a result, meaningful improvements cannot be achieved by varying one parameter in isolation, but instead require coordinated adjustment across multiple hyperparameters, essentially amounting to a full sweep.

5.3.3. Feature Ablation Study

To assess the impact of different input features on model performance, we conducted a feature ablation study where features were systematically varied during training. For efficiency, we used the fixed training setup from Table 5.1 without re-running a full hyperparameter search for each configuration. Since our TTM is sensitive to changes in the input distribution, these ablation results should be interpreted with caution. Table 5.5 summarizes the tested feature combinations, grouped by feature families.

Ablation Results From Table 5.5 we conclude: among the deep content embeddings, the body embedding proves most valuable: configurations including body text (e.g., MV, BI) consistently outperform those without. In contrast, titles alone (T) or images (I) provide weaker signals: titles are often too short and thus highly variable, while image embeddings introduce additional noise due to inconsistent visual representations of topics.

Table 5.5.: Feature ablation study grouped by feature families. Pop. = Popularity features, Inter. = Interaction features. Time combines both timestamp (cyclical encoding of hour/weekday) and article age. Meta features include categorical attributes such as article category, sentiment polarity, and premium status, among others. (The full feature list is part of the appendix A.5.)

Run	Embeddings			Meta	Time	Pop.	Inter.	Metrics	
	Body	Title	Image					Recall@20	NDCG@20
Baseline features									
B	•			•	•	•	•	**0.770**	**0.323**
Deep Embedding variants									
T		•		•	•	•	•	0.747	**0.323**
I			•	•	•	•	•	0.734	0.313
BT	•	•		•	•	•	•	0.599	0.243
BI	•		•	•	•	•	•	0.689	0.273
Feature Removals									
NE				•	•	•	•	0.593	0.217
MV	•					•	•	0.752	0.321
NP	•			•	•		•	0.294	0.120
NI	•			•	•	•		0.731	0.303

Metadata (NE) offers only limited benefit, showing that shallow categorical information alone is insufficient. The ablation study confirms that popularity and temporal features are essential. When popularity is removed (NP), performance collapses (Recall@20 = 0.294), underlining that news recommendation in this dataset is strongly popularity- and time-driven. Removing interaction features (NI) reduces Recall@20 to 0.731, demonstrating that modeling session context across clicks provides useful additional signal beyond content and popularity.

Interestingly, a minimal viable setup with body, popularity, and interaction features (MV) already achieves competitive performance (Recall@20 = 0.752), while adding metadata or explicit time features yields only small, but still relevant, improvements over this baseline. Together, these results validate our feature engineering strategy: retain the dominant popularity and temporal features, and enhance personalization with BERT-based body embeddings and interaction features.

These findings round off our analysis of feature contributions and lead us to the overall conclusions of this study.

6. Conclusion

This chapter summarizes the key findings of our study, relating them to the three research questions (RQ1–RQ3). We then discuss how SNeRT can be adapted to the BNN use case and conclude with an outlook on promising directions for future research.

6.1. Discussion of Results

RQ1: Global Recommendation Task SNeRT outperforms the baseline in our prioritized RQ1, showing measurable gains in global retrieval effectiveness. The model's very low latency makes it suitable for deployment in real-world news recommendation pipelines, where responsiveness is essential. Together, the improvements in retrieval suggest that SNeRT is a practical and scalable solution for large-scale retrieval. While gains over a strong popularity baseline remain modest, they show that representation learning captures user-content signals beyond short-term popularity, providing a solid foundation for reranking stages.

RQ2: In-view Recommendation Task For RQ2, which examined the in-view recommendation task, results highlight the structural limitations of two-tower architectures. Although SNeRT surpasses the Most Popular baseline, its performance remains well behind cross-encoders and ensemble methods. This confirms that two-tower models work for large-scale retrieval but struggle in reranking scenarios requiring fine-grained item-item comparisons. In practice, this necessitates a second-stage reranker, which partly undermines the simplicity and efficiency of two-tower designs. Thus, while SNeRT scales well for retrieval, it alone cannot match state-of-the-art performance in more context-sensitive tasks.

RQ3: Beyond-Accuracy Evaluation RQ3 examined qualitative aspects such as diversity, novelty, and category balance. SNeRT produces recommendations that are consistently more diverse, novel, and balanced than those of the baseline. While improvements in serendipity and diversity are moderate, they demonstrate that the model can partially counteract the strong influence of popularity signals. The most notable gain is in coverage, which increases substantially without a loss in ranking performance. This is particularly important in the news domain, where an overemphasis on trending articles risks reinforcing filter bubbles and limiting user exposure to the full spectrum of content.

Mitigating Embedding Collapse In addition to addressing the three research questions, this thesis makes a contribution by systematically investigating embedding collapse in two-tower recommendation models. Our experiments showed that indirect strategies (MoCo, session augmentation, modified attention) failed to resolve the issue under the

long-tail distribution of article popularity. In contrast, direct spectral regularization via the LogDet-loss consistently improved both embedding stability and retrieval performance. To our knowledge, this represents the first application of log-determinant regularization in a TTM. While the method requires careful tuning to balance spectrum preservation with user–item alignment, it provides a practical way to counteract collapse. Having established the effectiveness of spectral regularization in the EB-NeRD setting, we now turn to its adaptation in a different domain, namely the BNN dataset.

6.2. Adaptation to BNN

As a local newspaper, BNN differs substantially from the national tabloid setting of Ekstra Bladet. User clicks are expected to be less concentrated on a handful of front-page articles and more strongly influenced by the content itself, reflecting regional relevance and longer-term interests rather than short-lived popularity spikes that dominate in a tabloid newspaper. This aligns well with our design, since the heavy popularity skew was the key challenge in the EB-NeRD experiments and is likely to be less dominant at BNN. We therefore expect the performance gap between the Most Popular baseline and SNeRT to be significantly larger in this setting.

Additional features Adapting SNeRT to BNN also requires extending the feature set to better capture local reading preferences. An important addition is article location, enabling the model to learn that users engage more with local content (e.g., Bruchsal) than with articles from other regions (e.g., Pforzheim). Another useful feature is the device type, which can be encoded as a learned embedding within the interaction encoder. This provides further context about user behavior, for example distinguishing between mobile sessions that may favor shorter articles and desktop sessions that support longer reading times. These features were not available in the EB-NeRD dataset.

Adaptations for production deployment Finally, several adjustments are required to make the model suitable for a production environment. First, our current setup optimizes solely for click-through rate (CTR) and does not account for reading time as a measure of engagement. A straightforward way to incorporate this signal is to filter out articles with very short reading times during training, or even sample these articles as negatives. This ensures that the model prioritizes content that users actually consume rather than quickly abandon. Second, a multi-armed bandit strategy can be applied to introduce a controlled proportion of newly published articles, guaranteeing initial exposure and preventing fresh content from being systematically overlooked [83]. An additional consideration for deployment is the need for continuous online evaluation. Offline metrics such as Recall and NDCG are useful for benchmarking, but they do not always capture real user satisfaction or long-term engagement. In production, SNeRT should therefore be evaluated through A/B testing and online monitoring, tracking signals such as click-through rate, dwell time, and subscription conversions. This enables the system to validate offline improvements under realistic conditions, detect distributional shifts in user behavior, and evaluate and adapt the model accordingly.

6.3. Outlook and Future Work

Future work could explore fine-tuning BERT to improve the quality of article embeddings. Such fine-tuning would allow the model to better capture nuances of local reporting, complementing the structural features proposed for BNN.

Another promising direction is to evaluate the approach on a more balanced dataset, such as content from a regional newspaper. This would provide insights into how the model performs in settings with less extreme popularity skew and could validate our expectation that the gap between popularity-based baselines and our model becomes more pronounced.

Beyond dataset transfer, architectural extensions such as late interaction models, originally proposed in information retrieval [29], represent an interesting research avenue. These approaches combine the efficiency of two-tower retrieval with richer cross-interactions in the final layers, improving ranking quality while remaining scalable.

Finally, hybrid pipelines could be investigated, where efficient first-stage retrieval is combined with more expressive rerankers. In sum, future work should emphasize stronger feature integration, more advanced reranking, and evaluation across diverse settings.

Bibliography

[1] Mingxiao An et al. "Neural News Recommendation with Long- and Short-term User Representations". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 336–345. DOI: 10.18653/v1/P19-1033. URL: https://aclanthology.org/P19-1033/.

[2] Arda Antikacioglu and R. Ravi. "Post Processing Recommender Systems for Diversity". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 707–716. ISBN: 9781450348874. DOI: 10.1145/3097983.3098173. URL: https://doi.org/10.1145/3097983.3098173.

[3] Gianni Brauwers and Flavius Frasincar. "A General Survey on Attention Mechanisms in Deep Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 35.4 (Apr. 2023), pp. 3279–3298. ISSN: 2326-3865. DOI: 10.1109/tkde.2021.3126456. URL: http://dx.doi.org/10.1109/TKDE.2021.3126456.

[4] Pu-Chin Chen et al. *A Simple and Effective Positional Encoding for Transformers*. 2021. arXiv: 2104.08698 [cs.CL]. URL: https://arxiv.org/abs/2104.08698.

[5] Ting Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. arXiv: 2002.05709 [cs.LG]. URL: https://arxiv.org/abs/2002.05709.

[6] Ching-Yao Chuang et al. *Debiased Contrastive Learning*. 2020. arXiv: 2007.00224 [cs.LG]. URL: https://arxiv.org/abs/2007.00224.

[7] Kevin Clark et al. *What Does BERT Look At? An Analysis of BERT's Attention*. 2019. arXiv: 1906.04341 [cs.CL]. URL: https://arxiv.org/abs/1906.04341.

[8] Alexis Conneau et al. *Unsupervised Cross-lingual Representation Learning at Scale*. 2020. arXiv: 1911.02116 [cs.CL]. URL: https://arxiv.org/abs/1911.02116.

[9] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[10] Zhenhua Dong et al. *A Brief History of Recommender Systems*. 2022. arXiv: 2209.01860 [cs.IR]. URL: https://arxiv.org/abs/2209.01860.

[11] Matthijs Douze et al. *The Faiss library*. 2025. arXiv: 2401.08281 [cs.LG]. URL: https://arxiv.org/abs/2401.08281.

[12] *EB-NeRD: Ekstra Bladet News Recommendation Dataset A Large-Scale Dataset for News Recommendation*. https://recsys.eb.dk/dataset/#about. Last Accessed: 2025-07-20.

[13] Ekstra Bladet. *Ekstra Bladet Recommender System repository, created for the RecSys'24 Challenge*. 2024. URL: https://github.com/ebanalyse/ebnerd-benchmark (visited on 07/20/2025).

[14] Fangxiaoyu Feng et al. *Language-agnostic BERT Sentence Embedding*. 2022. arXiv: 2007.01852 [cs.CL]. URL: https://arxiv.org/abs/2007.01852.

[15] Rafael Glauber and Angelo Loula. *Collaborative Filtering vs. Content-Based Filtering: differences and similarities*. 2019. arXiv: 1912.08932 [cs.IR]. URL: https://arxiv.org/abs/1912.08932.

[16] Xingzhuo Guo et al. *On the Embedding Collapse when Scaling up Recommendation Models*. 2024. arXiv: 2310.04400 [cs.LG]. URL: https://arxiv.org/abs/2310.04400.

[17] Jungkyu HAN and Hayato YAMANA. "A Survey on Recommendation Methods Beyond Accuracy". In: *IEICE Transactions on Information and Systems* E100.D.12 (2017), pp. 2931–2944. DOI: 10.1587/transinf.2017EDR0003.

[18] Harvard NLP. *The Annotated Transformer*. https://nlp.seas.harvard.edu/2018/04/03/attention.html. Online; accessed 30-June-2025. Apr. 2018.

[19] Joe Hasell. "Measuring inequality: what is the Gini coefficient?" In: *Our World in Data* (2023). https://ourworldindata.org/what-is-the-gini-coefficient.

[20] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. 2020. arXiv: 1911.05722 [cs.CV]. URL: https://arxiv.org/abs/1911.05722.

[21] Balázs Hidasi et al. *Session-based Recommendations with Recurrent Neural Networks*. 2016. arXiv: 1511.06939 [cs.LG]. URL: https://arxiv.org/abs/1511.06939.

[22] Hakan Inan, Khashayar Khosravi, and Richard Socher. *Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling*. 2017. arXiv: 1611.01462 [cs.LG]. URL: https://arxiv.org/abs/1611.01462.

[23] Saidul Islam et al. *A Comprehensive Survey on Applications of Transformers for Deep Learning Tasks*. 2023. arXiv: 2306.07303 [cs.LG]. URL: https://arxiv.org/abs/2306.07303.

[24] Ashish Jaiswal et al. *A Survey on Contrastive Self-supervised Learning*. 2021. arXiv: 2011.00362 [cs.CV]. URL: https://arxiv.org/abs/2011.00362.

[25] Xuyang Jin. "Recommender Systems with Two-Stream Pyramid Encoder Network". In: *2022 2nd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*. 2022, pp. 548–552. DOI: 10.1109/CEI57409.2022.9950172.

[26] Li Jing et al. *Understanding Dimensional Collapse in Contrastive Self-supervised Learning*. 2022. arXiv: 2110.09348 [cs.CV]. URL: https://arxiv.org/abs/2110.09348.

[27] Wang-Cheng Kang and Julian McAuley. *Self-Attentive Sequential Recommendation*. 2018. arXiv: 1808.09781 [cs.IR]. URL: https://arxiv.org/abs/1808.09781.

[28] Seyed Mehran Kazemi et al. *Time2Vec: Learning a Vector Representation of Time*. 2019. arXiv: 1907.05321 [cs.LG]. URL: https://arxiv.org/abs/1907.05321.

[29] Omar Khattab and Matei Zaharia. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. 2020. arXiv: 2004.12832 [cs.IR]. URL: https://arxiv.org/abs/2004.12832.

[30] Johannes Kruse et al. "EB-NeRD a large-scale dataset for news recommendation". In: *Proceedings of the Recommender Systems Challenge 2024*. RecSys Challenge '24. ACM, Oct. 2024, pp. 1–11. DOI: 10.1145/3687151.3687152. URL: http://dx.doi.org/10.1145/3687151.3687152.

[31] Johannes Kruse et al. "RecSys Challenge 2024: Balancing Accuracy and Editorial Values in News Recommendations". In: *18th ACM Conference on Recommender Systems*. RecSys '24. ACM, Oct. 2024, pp. 1195–1199. DOI: 10.1145/3640457.3687164. URL: http://dx.doi.org/10.1145/3640457.3687164.

[32] Miaomiao Li and Licheng Wang. "A Survey on Personalized News Recommendation Technology". In: *IEEE Access* 7 (2019), pp. 145861–145879. DOI: 10.1109/ACCESS.2019.2944927.

[33] Nelson F. Liu et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[34] Zheng Liu et al. "Hi-Fi Ark: Deep User Representation via High-Fidelity Archive Network". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 3059–3065. DOI: 10.24963/ijcai.2019/424. URL: https://doi.org/10.24963/ijcai.2019/424.

[35] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Ed. by Lluís Màrquez, Chris Callison-Burch, and Jian Su. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421. DOI: 10.18653/v1/D15-1166. URL: https://aclanthology.org/D15-1166/.

[36] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

[37] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. *On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines*. 2021. arXiv: 2006.04884 [cs.LG]. URL: https://arxiv.org/abs/2006.04884.

[38] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG]. URL: https://arxiv.org/abs/1807.03748.

[39] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, III–1310–III–1318.

[40] Michael J. Pazzani and Daniel Billsus. "Content-Based Recommendation Systems". In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_10. URL: https://doi.org/10.1007/978-3-540-72079-9_10.

[41] Shaowen Peng et al. "Balancing Embedding Spectrum for Recommendation". In: *ACM Transactions on Recommender Systems* (2024). URL: https://api.semanticscholar.org/CorpusID:270562071.

[42] Shaowen Peng et al. *Balancing Embedding Spectrum for Recommendation*. 2025. arXiv: 2406.12032 [cs.IR]. URL: https://arxiv.org/abs/2406.12032.

[43] Aleksandr V. Petrov, Sean MacAvaney, and Craig Macdonald. "Shallow Cross-Encoders for Low-Latency Retrieval". In: *Advances in Information Retrieval*. Springer Nature Switzerland, 2024, pp. 151–166. ISBN: 9783031560637. DOI: 10.1007/978-3-031-56063-7_10. URL: http://dx.doi.org/10.1007/978-3-031-56063-7_10.

[44] Pew Research Center. *State of the News Media 2016*. Accessed: 2025-07-22. Pew Research Center, 2016. URL: https://www.pewresearch.org/journalism/2016/05/05/3-users-spend-more-time-with-content-in-the-morning-or-late-at-night/.

[45] Ofir Press and Lior Wolf. *Using the Output Embedding to Improve Language Models*. 2017. arXiv: 1608.05859 [cs.CL]. URL: https://arxiv.org/abs/1608.05859.

[46] Tao Qi et al. *PP-Rec: News Recommendation with Personalized User Interest and Time-aware News Popularity*. 2021. arXiv: 2106.01300 [cs.IR]. URL: https://arxiv.org/abs/2106.01300.

[47] Ruihong Qiu et al. "Contrastive Learning for Representation Degeneration Problem in Sequential Recommendation". In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. WSDM '22. ACM, Feb. 2022, pp. 813–823. DOI: 10.1145/3488560.3498433. URL: http://dx.doi.org/10.1145/3488560.3498433.

[48] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018. URL: https://api.semanticscholar.org/CorpusID:49313245.

[49] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2103.00020.

[50] Shaina Raza and Chen Ding. *News Recommender System: A review of recent progress, challenges, and opportunities*. 2021. arXiv: 2009.04964 [cs.IR]. URL: https://arxiv.org/abs/2009.04964.

[51] Ruiyang Ren et al. *TOME: A Two-stage Approach for Model-based Retrieval*. 2023. arXiv: 2305.11161 [cs.IR].

[52] Weijieying Ren et al. *Mitigating Popularity Bias in Recommendation with Unbalanced Interactions: A Gradient Perspective*. 2022. arXiv: 2211.01154 [cs.IR]. URL: https://arxiv.org/abs/2211.01154.

[53] Juan Manuel Rodriguez and Antonela Tommasel. "Leveraging User History with Transformers for News Clicking: The DArgk Approach". In: *RecSysChallenge '24: Proceedings of the Recommender Systems Challenge 2024*. Association for Computing Machinery (ACM), 2024, pp. 48–52. DOI: 10.1145/3687151.3687161. URL: https://dl.acm.org/doi/10.1145/3687151.3687161.

[54] Guilherme Rosa et al. *In Defense of Cross-Encoders for Zero-Shot Retrieval*. 2022. arXiv: 2212.06121 [cs.IR]. URL: https://arxiv.org/abs/2212.06121.

[55] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information Processing Management* 24.5 (1988), pp. 513–523. ISSN: 0306-4573. DOI: https://doi.org/10.1016/0306-4573(88)90021-0. URL: https://www.sciencedirect.com/science/article/pii/0306457388900210.

[56] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015, pp. 815–823. DOI: 10.1109/cvpr.2015.7298682. URL: http://dx.doi.org/10.1109/CVPR.2015.7298682.

[57] Kunal Shah et al. "Recommender systems: An overview of different approaches to recommendations". In: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. 2017, pp. 1–4. DOI: 10.1109/ICIIECS.2017.8276172.

[58] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL: http://dx.doi.org/10.1016/j.physd.2019.132306.

[59] Frederico Souza and João Filho. *BERT for Sentiment Analysis: Pre-trained and Fine-Tuned Alternatives*. 2022. arXiv: 2201.03382 [cs.CL]. URL: https://arxiv.org/abs/2201.03382.

[60] Gabriel de Souza Pereira Moreira et al. "Transformers4Rec: Bridging the Gap between NLP and Sequential / Session-Based Recommendation". In: *Proceedings of the 15th ACM Conference on Recommender Systems*. RecSys '21. Amsterdam, Netherlands: Association for Computing Machinery, 2021, pp. 143–153. ISBN: 9781450384582. DOI: 10.1145/3460231.3474255. URL: https://doi.org/10.1145/3460231.3474255.

[61] Liangcai Su et al. "Beyond Two-Tower Matching: Learning Sparse Retrievable Cross-Interactions for Recommendation". In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '23. Taipei, Taiwan: Association for Computing Machinery, 2023, pp. 548–557. ISBN: 9781450394086. DOI: 10.1145/3539618.3591643. URL: https://doi.org/10.1145/3539618.3591643.

[62] Chi Sun et al. *How to Fine-Tune BERT for Text Classification?* 2020. arXiv: 1905.05583 [cs.CL]. URL: https://arxiv.org/abs/1905.05583.

[63] Fei Sun et al. *BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer*. 2019. arXiv: 1904.06690 [cs.IR]. URL: https://arxiv.org/abs/1904.06690.

[64] Mo Sun et al. *Positional Attention for Efficient BERT-Based Named Entity Recognition*. 2025. arXiv: 2505.01868 [cs.CL]. URL: https://arxiv.org/abs/2505.01868.

[65] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[66] Hongwei Wang et al. *DKN: Deep Knowledge-Aware Network for News Recommendation*. 2018. arXiv: 1801.08284 [stat.ML]. URL: https://arxiv.org/abs/1801.08284.

[67] Jingkun Wang et al. "Popularity-Enhanced News Recommendation with Multi-View Interest Representation". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, pp. 1949–1958. ISBN: 9781450384469. DOI: 10.1145/3459637.3482462. URL: https://doi.org/10.1145/3459637.3482462.

[68] Mingze Wang and Weinan E. *Understanding the Expressive Power and Mechanisms of Transformer for Sequence Modeling*. 2024. arXiv: 2402.00522 [cs.LG]. URL: https://arxiv.org/abs/2402.00522.

[69] Rongyao Wang, Veronica Liesaputra, and Zhiyi Huang. *A Survey on LLM-based News Recommender Systems*. 2025. arXiv: 2502.09797 [cs.IR]. URL: https://arxiv.org/abs/2502.09797.

[70] Rongyao Wang and Wenpeng Lu. *Modeling Multi-interest News Sequence for News Recommendation*. 2022. arXiv: 2207.07331 [cs.IR]. URL: https://arxiv.org/abs/2207.07331.

[71] Tongzhou Wang and Phillip Isola. *Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere*. 2022. arXiv: 2005.10242 [cs.LG]. URL: https://arxiv.org/abs/2005.10242.

[72] Benjamin Warner et al. *Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference*. 2024. arXiv: 2412.13663 [cs.CL]. URL: https://arxiv.org/abs/2412.13663.

[73] Chuhan Wu et al. "Empowering News Recommendation with Pre-trained Language Models". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 1652–1656. ISBN: 9781450380379. DOI: 10.1145/3404835.3463069. URL: https://doi.org/10.1145/3404835.3463069.

[74] Chuhan Wu et al. *Neural News Recommendation with Attentive Multi-View Learning*. 2019. arXiv: 1907.05576 [cs.CL]. URL: https://arxiv.org/abs/1907.05576.

[75] Chuhan Wu et al. “Neural News Recommendation with Multi-Head Self-Attention”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 6389–6394. DOI: 10.18653/v1/D19-1671. URL: https://aclanthology.org/D19-1671/.

[76] Chuhan Wu et al. *NewsBERT: Distilling Pre-trained Language Model for Intelligent News Application*. 2021. arXiv: 2102.04887 [cs.CL]. URL: https://arxiv.org/abs/2102.04887.

[77] Chuhan Wu et al. “Two Birds with One Stone: Unified Model Learning for Both Recall and Ranking in News Recommendation”. In: *Findings of the Association for Computational Linguistics: ACL 2022*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3474–3480. DOI: 10.18653/v1/2022.findings-acl.274. URL: https://aclanthology.org/2022.findings-acl.274/.

[78] Xu Xie et al. *Contrastive Learning for Sequential Recommendation*. 2021. arXiv: 2010.14395 [cs.IR]. URL: https://arxiv.org/abs/2010.14395.

[79] Lanling Xu et al. *Negative Sampling for Contrastive Representation Learning: A Review*. 2022. arXiv: 2206.00212 [cs.IR]. URL: https://arxiv.org/abs/2206.00212.

[80] Yi Xu et al. *Addressing Information Loss and Interaction Collapse: A Dual Enhanced Attention Framework for Feature Interaction*. 2025. arXiv: 2503.11233 [cs.IR]. URL: https://arxiv.org/abs/2503.11233.

[81] Yihao Xue et al. *Which Features are Learnt by Contrastive Learning? On the Role of Simplicity Bias in Class Collapse and Feature Suppression*. 2023. arXiv: 2305.16536 [cs.LG]. URL: https://arxiv.org/abs/2305.16536.

[82] Wei Yang et al. “End-to-End Open-Domain Question Answering with BERTserini”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Ed. by Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 72–77. DOI: 10.18653/v1/N19-4013. URL: https://aclanthology.org/N19-4013/.

[83] Mengyan Zhang et al. *Two-Stage Neural Contextual Bandits for Personalised News Recommendation*. 2022. arXiv: 2206.14648 [cs.IR]. URL: https://arxiv.org/abs/2206.14648.

[84] Ruisheng Zhang et al. “Collaborative Filtering for Recommender Systems”. In: *2014 Second International Conference on Advanced Cloud and Big Data*. 2014, pp. 301–308. DOI: 10.1109/CBD.2014.47.

[85] Xiaokun Zhang et al. *Rethinking Contrastive Learning in Session-based Recommendation*. 2025. arXiv: 2506.05044 [cs.IR]. URL: https://arxiv.org/abs/2506.05044.

[86] Yifei Zhang et al. “Mitigating the Popularity Bias of Graph Collaborative Filtering: A Dimensional Collapse Perspective”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 67533–67550. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/d5753be6f71fbfefaf47aa27ec41279c-Paper-Conference.pdf.

[87] Yu Zhao and Fang Liu. *A Survey of Retrieval Algorithms in Ad and Content Recommendation Systems*. 2024. arXiv: 2407.01712 [cs.IR]. URL: https://arxiv.org/abs/2407.01712.

A. Appendix

A.1. Cross-Encoder Reranker

While the primary focus of this thesis is the Two-Tower Model (TTM), our results reveal its limitations in ranking scenarios. In industrial applications, this weakness is typically addressed by adding a second-stage Cross-Encoder (CE) for re-ranking (see Subsection 2.4.1).

Architecture To evaluate the viability of this approach, we additionally implemented a CE reranker that combines the user encoder from the TTM (see Subsection 4.3) with a candidate-decoder transformer inspired by a RecSys'24 model [53]. The CE uses the same article encoder described in Subsection 4.2. While this design provides stronger ranking expressiveness, its cross-attention over all candidates substantially increases latency, making it more suitable as a second-stage reranker rather than a standalone retrieval model.

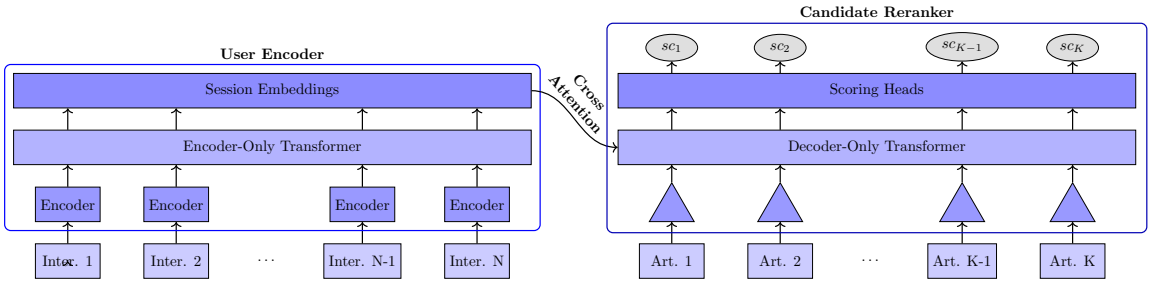


Figure A.1.: Cross-Encoder reranker architecture: user sessions are encoded with the TTM user encoder, candidate articles with the shared article encoder. A decoder-only transformer with cross-attention produces contextualized representations, which are scored by an MLP head to yield click logits.

The scoring head maps decoder outputs to click logits through a lightweight MLP. These scalar scores are then used to rank candidate articles by predicted click likelihood.

Training The CE is trained using the same features and negative sampling pipeline as the TTM. The loss function is a listwise cross-entropy objective, ensuring that the clicked article receives the highest score among sampled negatives,

$$\mathcal{L}_{\text{rank}} = -\log \frac{\exp(sc_0/\tau)}{\sum_{j=0}^{K} \exp(sc_j/\tau)}, \tag{A.1}$$

where sc_j are the predicted scores and τ is the temperature of the softmax. At inference, logits are used directly as ranking scores without normalization, since softmax is a monotonic transformation and does not affect ranking order.

We also experimented with a joint user–article encoder, which in principle could reduce latency. However, due to the high sensitivity of TTM training to small changes (see Subsection 5.3.2), separate training of the two encoders proved more stable and practical.

A.2. Formal Definitions

This section provides the formal definitions of non-standard metrics used in our work. For each metric, we specify the mathematical formulation along with the variables involved, ensuring clarity and reproducibility of our experiments.

A.2.1. Article Popularity and Lifecycle

To analyze the distribution of clicks on articles, we define different metrics to better quantify trends in the article-user interactions.

Relative Popularity We define the relative popularity of an article a at time t as the fraction of clicks it receives within a sliding 15-minute window:

$$p_t(a) = \frac{c_t(a)}{\sum_{a' \in \mathcal{A}_t} c_t(a')}, \tag{A.2}$$

where $c_t(a)$ is the number of clicks on article a in the interval $[t - 15\text{min}, t]$, and $\mathcal{A}_t$ is the set of articles with at least one click in this window.

Inequality of Relative Popularity To measure how skewed relative popularity is, we use the Lorenz curve and the Gini coefficient. The Lorenz curve visualizes the cumulative distribution of relative popularity across articles, while the Gini coefficient summarizes this inequality in a single scalar. These metrics are used in RecSys research as measures of distributional fairness and exposure imbalance [2].

Formally, articles are sorted by $p_t(a)$ in ascending order, where $p_t(a)$ denotes the relative popularity of article a at time t. The Lorenz curve is then defined by the cumulative share of articles x and their cumulative share of clicks $L_t(x)$ at time t [2, 19]. To obtain a stable estimate, we average $L_t(x)$ across all timesteps, resulting in an aggregated Lorenz curve $\bar{L}(x)$.

The Gini coefficient is then defined as

$$G = 1 - 2 \int_0^1 \bar{L}(x)\, dx, \tag{A.3}$$

with $G = 0$ indicating perfect equality (all articles receive the same share of clicks) and $G = 1$ representing maximum inequality (all clicks concentrated on a single article).

Article Lifecycle We define the lifecycle of an article a as the time from its publication until it has accumulated 90% of its total observed clicks. Formally,

$$L(a) = \min\Big\{\tau \mid \sum_{t=t_0}^{t_0+\tau} c_t(a) \;\geq\; 0.9 \cdot C(a)\Big\}, \tag{A.4}$$

where t_0 is the publication time of a, $c_t(a)$ is the number of clicks on a at time t, and $C(a) = \sum_{t=t_0}^{\infty} c_t(a)$ is the total number of clicks.

To avoid bias from the dataset cutoff, we excluded articles published during the final day of the collection period from our analysis. Thus, $L(a)$ captures the effective duration during which article a remains relevant for user engagement.

Most Popular Baseline The Most Popular baseline recommends the globally most clicked articles within a short, recent time window. Formally, at recommendation time t_{rec}, we define the popularity score of an article a as

$$\pi_{t_{\text{rec}}}(a) = \sum_{\tau=t_{\text{rec}}-\Delta}^{t_{\text{rec}}} c_\tau(a)\,, \tag{A.5}$$

where $c_\tau(a)$ is the number of clicks on article a at time τ, and Δ is the window size.

In our case, $\Delta = 15$ minutes, i.e., the baseline always recommends the K articles with the highest click counts in the last 15 minutes. This choice reflects the strong recency and popularity bias in news consumption, making the Most Popular baseline a competitive reference in this domain.

A.2.2. Beyond-Accuracy Metrics

To complement ranking metrics such as Recall and NDCG, we assess recommendation quality using four beyond-accuracy measures—diversity, serendipity, novelty, and coverage—selected based on their role in the evaluation protocol of the RecSys'24 Challenge [31].

Diversity (ILD). Intra-List Diversity (ILD) measures how different the recommended items are from one another. It is defined as

$$\text{ILD} = \frac{2}{K(K-1)} \sum_{i<j} \big(1 - \cos(\mathbf{v}_i, \mathbf{v}_j)\big)\,, \tag{A.6}$$

where K is the number of recommended items for a user, and $\mathbf{v}_i$ denotes the L2-normalized BERT embedding of item i. Higher ILD values indicate more varied recommendation lists.

Serendipity. Serendipity quantifies how unexpected recommendations are relative to a user's history, while still being relevant. For user b, it is defined as

$$\text{Serendipity}_b = \frac{1}{KL_b} \sum_{i \in \text{rec}_b} \sum_{j \in \text{hist}_b} \big(1 - \cos(\mathbf{v}_i, \mathbf{v}_j)\big)\,, \tag{A.7}$$

where rec_b is the set of recommended items, hist_b the set of historical items, and L_b the length of the history.

Novelty. Novelty encourages the recommendation of less popular items. It is defined as

$$\text{Novelty} = \frac{1}{K} \sum_{i \in L} -\log_2 p(i), \tag{A.8}$$

where L is the recommendation list and $p(i)$ is the normalized popularity (click probability) of item i.

Coverage. Coverage captures how much of the item catalog is exposed across all users. It is defined as

$$\text{Coverage} = \frac{|\bigcup_{s \in S} \text{rec}_s|}{|\text{Pub}|}, \tag{A.9}$$

where S is the set of all sessions and Pub is the set of all published items during evaluation. High coverage indicates that recommendations draw from a broader portion of the catalog.

A.2.3. Relative Effective Rank

To allow comparison across different embedding dimensions, we normalize the effective rank (see Equation 2.9) with the embedding size d:

$$\text{rerank}(X) = \frac{\text{erank}(X)}{d}. \tag{A.10}$$

The resulting value lies in $[0, 1]$ and quantifies the proportion of the embedding space that is effectively utilized, with lower values indicating stronger collapse.

A.3. Implementation Notes

A central requirement of our approach is to provide the model with time-dependent relative popularity over different timeframes for every article. To this end, we precompute a global popularity tensor $\mathcal{P} \in \mathbb{R}^{51841\times125542\times7}$, covering 51k time indices, 125k articles, and seven feature variants (different window sizes). This corresponds to roughly 45 billion values in total. The scale of such a tensor makes dense in-memory storage infeasible, so we store it as a memory-mapped array, enabling efficient random access to slices without loading the full structure.

This motivates two implementation details described below: first, a sliding-window band-matrix construction to compute relative popularity from click logs (see Section A.2.1); second, batched lookups into $\mathcal{P}$ during training to map negatives to the correct popularity values (see Section 4.4.2) efficently.

A.3.1. Sliding-Window Relative Popularity

We represent the click log as a sparse article–time matrix $X \in \mathbb{R}^{n_{\text{articles}}\times n_{\text{timestamps}}}$, where each entry $X_{i,t}$ is the number of clicks on article i at time step t. For a sliding window of size w, we construct a band matrix $W^{(w)} \in \{0,1\}^{n_{\text{timestamps}}\times n_{\text{timestamps}}}$ that marks the preceding w time steps. Formally,

$$W^{(w)}_{t',t} = \begin{cases} 1 & \text{if } 0 \le t - t' < w, \\ 0 & \text{otherwise.} \end{cases} \tag{A.11}$$

Multiplying gives the windowed counts

$$S^{(w)} = XW^{(w)}, \tag{A.12}$$

so that each column contains click counts aggregated over the last w steps.

To obtain a probability distribution, we normalize each column by the total number of clicks in that window:

$$P^{(w)} = XW^{(w)}\,\mathrm{diag}\big(c^{(w)}\big)^{-1}, \qquad c^{(w)}_t = \sum_i S^{(w)}_{i,t}. \tag{A.13}$$

The result $P^{(w)}$ is a sparse column-stochastic matrix, where each column t gives the relative popularity of articles at time t. Using sparse matrix multiplication makes this computation fast and memory-efficent, reducing the computation time from multiple hours to under 5 minutes on a single CPU.

A.3.2. Memory-Mapped Popularity Lookup

To efficiently integrate time-dependent popularity features into the training pipeline, we implement a memory-mapped lookup mechanism. For a batch with target timestamps $(t^{(b)})_{b=1}^{B}$ and candidate article IDs $\mathbf{A} \in \mathbb{N}^{B\times K}$, we define two index mappings

$$\phi : \mathbb{T} \to \{0,\dots,T-1\}, \qquad \iota : \mathbb{N} \to \{0,\dots,A-1\}, \tag{A.14}$$

where ϕ maps timestamps to discrete time indices and ι maps article IDs to their row indices in the tensor.

Popularity features are retrieved by batched tensor indexing:

$$\text{NegPop}_{b,j,:} \;=\; \mathcal{P}\big[\phi(t^{(b)}),\, \iota(\mathbf{A}_{b,j}),\, :\,\big], \qquad b = 1,\dots,B,\;\; j = 1,\dots,K. \tag{A.15}$$

Since $\mathcal{P}$ is stored as a memory-mapped array (`numpy.memmap`), only the required slices are loaded into RAM. This allows efficient retrieval of $\text{NegPop} \in \mathbb{R}^{B\times K\times R}$ for batch size B, candidate set size K, and feature variants R, even though the full tensor is far too large to fit in memory. In practice, vectorized indexing yields retrieval times on the order of milliseconds, effectively removing a major bottleneck from the training pipeline.

A.3.3. Session Augmentation Pipeline

For our experiments with a session augmentation loss (see Subsection 4.5.2) we adapt a processing pipeline to our work. To get two augmented views of the same session we adapt ideas from different prior implementations to our work [78, 85].

We further list the augmentation strategies used to construct perturbed session views for this loss.

Table A.1.: Data augmentation techniques and hyperparameters used for the session-level contrastive loss.

Technique	Description	Parameter	Value
Item Swap	Replace articles with similar ones based on BERT embedding cosine similarity.	α_{swap}	0.2
Time Jittering	Perturb timestamps within ±60 minutes to model temporal noise.	α_{jitter}	0.3
Gap & Readtime Swap	Swap gaps/readtimes across positions to vary engagement patterns.	α_{gap}	0.15
Shuffling	Shuffle random subsequences within a session.	α_{shuffle}	0.1
Item Masking	Zero-mask interactions and their features.	α_{mask}	0.25

A.4. Additional Figures

In this section, we provide supplementary figures and plots that were omitted from the main text for brevity, but still add to our work.

A.4.1. Article Lifecycles by Category

This section support our claim that article lifecycles vary across categories. We plot the distribution of article lifespans.

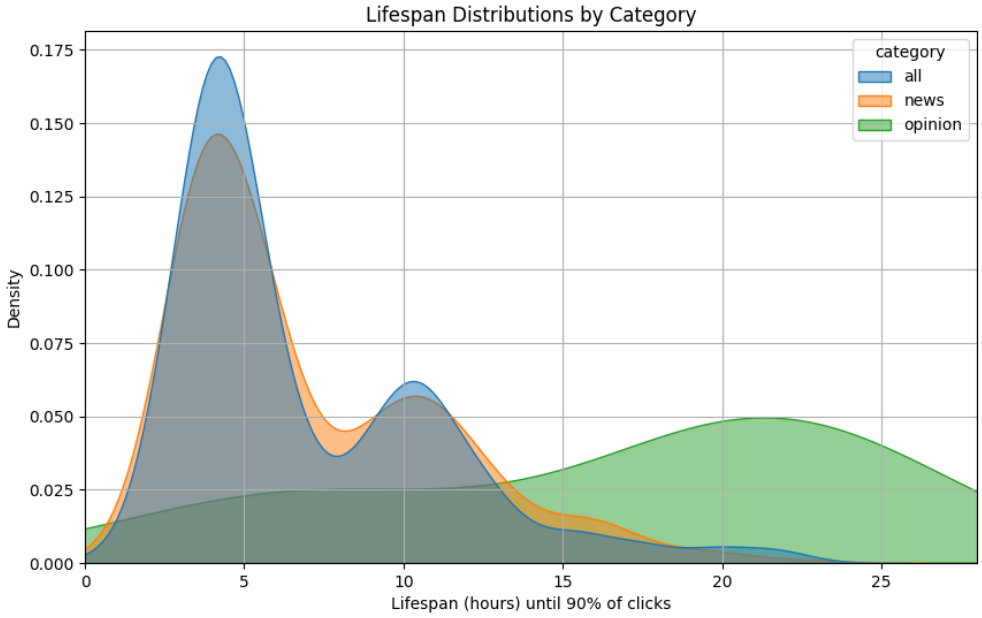


Figure A.2.: Kernel Density Estimate (KDE) of article lifespans by category, defined as the time in hours until 90% of total clicks are accumulated. Only articles with at least 1,000 clicks and 24 hours of available data were included.

A.4.2. Interaction Encoder

This section illustrates the computation of interaction embeddings used in our model.

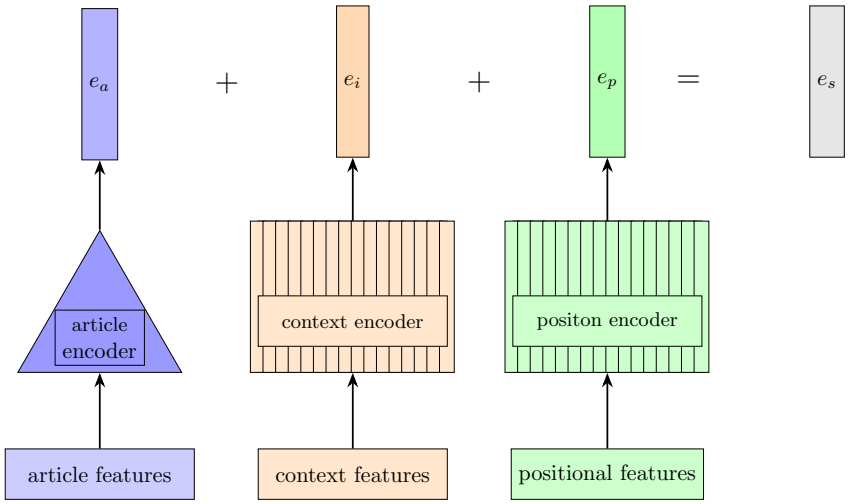


Figure A.3.: Computation of interaction embeddings. The article encoder is shared between history and target articles (weight tying), while context features (read time and session gap) and positional information are represented using learned embeddings.

A.5. Example Data from EB-Nerd

This section provides excerpts from the EB-NeRD dataset to give a general overview. We include three complementary tables: an example of user interaction history with article identifiers, impression times, and read times; a sample of the article metadata table containing publication time, content, sentiment, and additional attributes; and an excerpt of the impression-level behavior log, which illustrates the inview field alongside clicked articles and timestamps. Together, these tables highlight the core structures of EB-NeRD.

Table A.2.: Example user history table

user_id	**article_id**	**impression_time**	**read_time** (s)
10029	[3000022, 3000063, 3000613]	[2023-04-28 06:16, 06:17, 06:20]	[28, 24, 11]
10033	[3000063, 3000022, 3000613]	[2023-04-27 11:11, 11:12, 11:13]	[2, 2, 718]
10034	[3000613, 3000063, 3000022]	[2023-04-30 09:46, 09:47, 09:49]	[21, 103, 28]

Table A.3.: Example article table

article_id	**title**	**body**	**sent_score**
3000022	Hanks beskyldt for mishandling…	Tom Hanks skulle angiveligt have…	0.9911
3000063	Bostrups aske spredt i Furesøen…	Strålende sensommersol. Jazzede toner…	0.5155
3000613	Jesper Olsen ramt af hjerneblødning…	Jesper Olsen, der er noteret for 43…	0.9433

Table A.4.: Example behavior (impression-level) log from EB-NeRD.

impression_id	**user_id**	**inview_article_ids**	**clicked_article_ids**	**impression_time**
5001123	10029	[3000022, 3000063, 3000613,· · ·]	[3000022]	2023-04-28 06:16:40
5001124	10029	[3000063, 3000613]	[]	2023-04-28 06:17:05
5001150	10033	[3000063, 3000022, 3000613,· · ·]	[3000063]	2023-04-27 11:11:22
5001178	10034	[3000613, 3000063,· · ·]	[3000613,· · ·]	2023-04-30 09:46:50

A.6. Feature Engineering

A.6.1. Full Feature List

Table A.5 provides the complete set of engineered features used in our models, grouped by feature family. Metadata covers both learned embeddings (e.g., category, premium status) and engineered linguistic statistics extracted from article text (e.g., word counts, sentence length).

Table A.5.: Complete list of engineered features grouped by family, with corresponding transformations.

Feature	**Processing / Normalization**
Content Features	
body_embedding	Pretrained BERT, L2-normalized
title_embedding	Pretrained BERT, L2-normalized
image_embedding	Precomputed Embeddings from RecSys'24 challenge[1]
Metadata Features	
category_id	Trainable embedding
sentiment_score	StandardScaler
premium_status	Trainable embedding
body_word_count	$\log(1+x)$, then RobustScaler (1st–99th pct)
body_sentence_count	$\log(1+x)$, then RobustScaler (1st–99th pct)
body_unique_ratio	$\log(1+x)$, then RobustScaler (1st–99th pct)
body_exclamation_count	$\log(1+x)$, then RobustScaler (1st–99th pct)
body_comma_count	$\log(1+x)$, then RobustScaler (1st–99th pct)
body_digit_ratio	QuantileTransformer(N=1000) $\rightarrow$ Normal output
title_word_count	RobustScaler (1st–99th pct)
subtitle_word_count	RobustScaler (1st–99th pct)
body_avg_token_len	RobustScaler (1st–99th pct)
body_avg_sentence_len	RobustScaler (1st–99th pct)
Temporal Features	
min_sin, min_cos	Cyclical encoding
hour_sin, hour_cos	Cyclical encoding
weekday_sin, weekday_cos	Cyclical encoding
age_feature	$\log(1+x)$, then RobustScaler
Popularity Features	
popularity(1min, 5min, 15min, 1h,2h,4h, 24h)	$\log(1+x)$, then RobustScaler
Interaction Features	
interactiongap_bucket	Trainable embedding
readtime_bucket	Trainable embedding

[1]Due to copyright restrictions, images were excluded from the dataset, and we therefore relied on precomputed embeddings from [12].

A.7. Results

For completeness, we include below the numerical values corresponding to the plots shown in the main text. These tables provide the exact values for category distributions, negative sampling mixtures, and hyperparameter search outcomes, which were primarily discussed in Section 5 using figures.

A.7.1. Category Distribution

Table A.6 gives the full category proportions underlying Figure 5.1, allowing a direct comparison between the Most Popular baseline and SNeRT.

Table A.6.: Category distribution (fraction of recommendations) for Most Popular baseline and SNeRT, with differences (Δ) reported. Categories are assigned by Ekstra Bladet.

Run	Entertainment	News	Sport	Music	Crime	Private Finance	Misc.
Most Popular	0.1960	0.2933	0.1581	0.0828	0.0744	0.0801	0.1154
SNeRT	0.2143	0.2619	0.1613	0.0751	0.1086	0.0878	0.0910
Δ	+0.0183	-0.0314	+0.0032	-0.0077	+0.0342	+0.0077	-0.0244

A.7.2. Negative Sampling

Table A.7 complements Figure 5.2, reporting the detailed Recall@20 and in-view AUC scores for each mixture.

Table A.7.: Negative sampling strategies and mixtures with corresponding Recall@20 and in-view AUC. Pop = popularity-based, Inv = in-view negatives, Rec = recency-based, Uni = uniform random. Best results in bold.

Pop	Inv	Rec	Uni	Recall@20	In-View AUC
0.4	0.2	0.2	0.2	**0.770**	0.717
0.4	0.0	0.4	0.2	0.768	0.710
0.4	0.2	0.4	0.0	0.750	0.713
0.5	0.2	0.2	0.1	0.746	0.703
0.4	0.1	0.4	0.1	0.743	0.703
0.4	0.2	0.3	0.1	0.741	**0.721**
0.5	0.0	0.0	0.5	0.741	0.709
0.4	0.1	0.4	0.1	0.722	0.705
0.3	0.3	0.3	0.1	0.721	0.717
0.0	0.5	0.0	0.5	0.681	0.710
0.0	0.4	0.4	0.2	0.649	0.718

A.7.3. Hyperparameter Search

Table A.8 contains the values summarized in Figure 5.3, showing the impact of individual parameter choices on Recall@20.

InfoNCE Temp.		**Spectral Reg.**		**Transformer Layers**	
Param.	**Recall@20**	**Param.**	**Recall@20**	**Param.**	**Recall@20**
0.40	0.744	0.0006	0.725	3	0.726
0.70	**0.758**	0.0007	0.748	4	**0.770**
0.85	0.740	0.0008	**0.760**	5	0.738
1.00	0.726	0.0009	0.751	6	0.756
1.50	0.736	0.0010	0.754		
Attention Heads		**Embedding Size**		**Session Length**	
Param.	**Recall@20**	**Param.**	**Recall@20**	**Param.**	**Recall@20**
1	0.751	64	0.670	8	0.705
2	**0.770**	96	0.758	12	0.746
4	0.762	128	**0.770**	16	**0.770**
8	0.755	192	0.753	24	0.759
16	0.694	256	0.727	32	0.746

Table A.8.: Hyperparameter search results (Recall@20). Best values in bold. Each cell shows a mini-table where one parameter is varied while others are fixed.