Large Scale Retrieval for the LinkedIn Feed Using Causal Language Models

Sudarshan Srinivasa Ramanujam*, Antonio Alonso*, Saurabh Kataria*, Siddharth Dangi*, Akhilesh Gupta*, Birjodh Singh Tiwana*, Manas Haribhai Somaiya*, Luke Simon*, David Byrne*, Sojeong Ha, Sen Zhou, Andrei Akterskii, Zhanglong Liu, Samira Sriram, Zihan Xiong, Zhoutao Pei, Angela Shao, Alex Li, Annie Xiao, Caitlin Kolb, Thomas Kistler, Zach Moore, Hamed Firooz*†

LinkedIn Corporation, Sunnyvale, CA, USA

Abstract

In large-scale recommendation systems like LinkedIn's, the retrieval stage is critical for narrowing billions of potential candidates to a manageable subset for ranking. LinkedIn's feed now serves suggested content based on the topical interests of members, where 2000 candidates are retrieved from several million candidates with a latency budget of a few milliseconds and inbound QPS of several thousand per second. This paper presents a novel retrieval approach that fine tunes a large causal language model (Meta's LLaMA 3) as a dual encoder to generate high quality embeddings for both users (members) and content (items), using only textual input. We describe the end to end pipeline, including prompt design for embedding generation, techniques for fine tuning at LinkedIn scale, and infrastructure for low latency, cost effective online serving. We share our findings on how quantizing numerical features in the prompt enables the information getting encoded in the embedding facilitating greater alignment between the retrieval and ranking layer. The system was evaluated using offline metrics and an online A/B test, which showed substantial improvements in member engagement. We observed significant gains among newer members, who often lack strong network connections, indicating that high-quality suggested content aids retention. This work demonstrates how generative language models can be effectively adapted for real time, high throughput retrieval in industrial applications.

1 Introduction

Modern content recommendation systems rely heavily on fast, scalable retrieval techniques to surface relevant items from large candidate pools. At LinkedIn, the Feed retrieval stack has evolved into a highly complex ecosystem comprising multiple index types, including inverted indices of chronologically ordered member activities (Swapnil Ghike 2016), trending sources, collaborative filtering and two-tower embedding-based retrieval (EBR) systems (Borisyuk et al. 2024) for surfacing content from unconnected members. While this multi-index architecture has enabled targeted and personalized Feed experiences, it has also introduced significant engineering complexity and operational overhead, particularly when integrating heterogeneous retrieval signals at scale. Recent advancements in large language models (LLMs) present a compelling opportunity to re-imagine retrieval architectures (Zhai 2024). LLMs pretrained on massive corpora have demonstrated strong capabilities in representation learning and semantic understanding, making them increasingly attractive for retrieval tasks. However, these models are typically not optimized for platform-specific engagement objectives or constrained retrieval latency requirements. In this work, we propose a novel LLM-based approach to retrieval for the LinkedIn Feed. Our method builds upon an off-the-shelf pretrained LLM, which we further fine-tune using large-scale engagement data specific to LinkedIn. The goal is to directly optimize the model to generate semantically rich query and item representations that capture the nuanced preferences of our member base. We then use these representations in a dense retrieval setup to replace our existing retrieval pipelines (see Figure 1 for the current list of sources in the suggested content model). A key benefit of this approach is the consolidation of disparate retrieval pathways into a unified, embedding-based system. By leveraging a single EBR framework fine-tuned with engagement-supervised LLMs, we can simplify system architecture, reduce maintenance overhead, and enable more coherent ranking stages downstream. Empirical results demonstrate that our approach not only improves relevance compared to multiple retrieval sources with different index structures, but also improves upon latency and throughput.

1.1 Key Contributions

- We present a practical method for fine-tuning a large language model using real-world engagement signals to enhance retrieval performance in a production-scale feed retrieval setting.
- We provide extensive offline and online evaluation to

*These authors contributed equally.

†Work done while at LinkedIn.

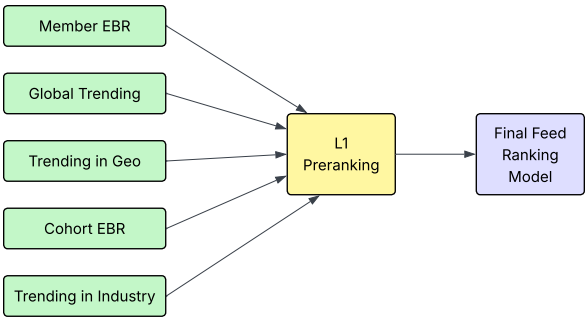

Figure 1: Architecture of the current Suggested Content Ranking and Retrieval Set up. Five sources are shown as examples to illustrate retrieval from multiple sources. In practice, the system has many more sources

show the effectiveness of our approach in improving relevance and retrieval efficiency for the LinkedIn Feed.

- We share our learnings on practical techniques to ensure the important features we added to the prompt are effectively encoded into embeddings.

2 Related Work

Modern content recommendation systems are typically built upon a multi-stage architecture, commonly involving a retrieval stage followed by a ranking stage. Our work contributes to the evolving landscape of retrieval in recommendation systems, particularly at the intersection of traditional multi-index approaches and the emerging capabilities of Large Language Models (LLMs).

Traditional Retrieval Architectures in Recommendation Systems Early recommendation systems often relied on simpler retrieval mechanisms, such as collaborative filtering (CF) based on user-item interaction histories (Sarwar et al. 2001; Koren, Bell, and Volinsky 2009) or content-based filtering using item metadata. As content platforms scaled, the need for faster and more efficient retrieval led to the adoption of inverted indices for keyword-based search and chronological retrieval for feeds, as seen in social media platforms (Wang, Li, and Li 2018). LinkedIn's Feed retrieval stack exemplifies this evolution, employing a combination of inverted indices for chronologically ordered activities (Swapnil Ghike 2016), trending sources, and collaborative filtering. This heterogeneous approach allows for diverse signals to contribute to candidate generation, addressing various aspects of user interest (e.g., recency, popularity, similarity to past interactions). More recently, Embedding-Based Retrieval (EBR) systems, particularly two-tower models, have become a cornerstone of large-scale retrieval (Covington, Adams, and Weinberger 2016; Ying et al. 2018). These models learn dense vector representations (embeddings) for both queries (users or contexts) and items, enabling efficient nearest-neighbor search in a high-dimensional space. The "two-tower" architecture separates the embedding computation for queries and items, allowing for pre-computation of item embeddings and real-time query embedding, facilitating fast approximate nearest neighbor (ANN) search using techniques like FAISS (Johnson, Douze, and Jégou 2019) or ScaNN (Andoni et al. 2020). LinkedIn's adoption of two-tower EBR systems (Borisyuk et al. 2024) for surfacing content from unconnected members aligns with this industry trend.

Large Language Models in Recommendation and Retrieval The advent of Large Language Models (LLMs) has marked a significant paradigm shift across various NLP tasks, and their application to recommendation systems is a rapidly growing area of research (Covington, Adams, and Weinberger 2016; Huang et al. 2023). LLMs, pre-trained on vast text corpora, possess remarkable capabilities in semantic understanding, contextual reasoning, and representation learning. This has led to their exploration in recommendation for tasks such as generative recommendation (Gao et al. 2023; Liu et al. 2024), LLM based feature extraction from text associated with user profiles, item descriptions, and interaction history for retrieval and ranking models (Yan 2024), and integrating LLM for multi-turn dialogues based recommendations (Feng, Zhang, and Ji 2023). For retrieval, LLMs offer the potential to create semantically richer embeddings that capture nuanced relationships between users and items, moving beyond simple keyword matching or explicit collaborative signals. Models like BERT (Devlin et al. 2019) and more recent LLMs have been used as powerful encoders for text, enabling highly effective dense retrieval, where the relevance is determined by the similarity of dense vectors (Karpukhin et al. 2020).

Fine-tuning LLMs for Recommendation To address the limitations of off-the-shelf LLMs, a growing body of work focuses on fine-tuning these models for specific recommendation tasks and leveraging domain-specific data. This often involves:

- **Supervised Fine-tuning (SFT)** Training the LLM on labeled datasets of user-item interactions, where the goal is to optimize for explicit engagement signals (e.g., positive vs. negative interactions) (Casalegno 2025). This aligns the LLM's representations with the actual behaviors and preferences observed on the platform.
- **Continued Pre-training** Further pre-training LLMs on large-scale unlabeled domain-specific data (e.g., a platform's entire content corpus or user-generated text) to enhance their understanding of the particular domain's vocabulary and concepts (Prasad 2025).
- **Reinforcement Learning from Human Feedback (RLHF)** While more commonly applied to text generation, RLHF principles can be adapted to align LLM-based recommenders with human preferences for relevance, diversity, and quality (Ouyang et al. 2022).

3 Datasets And Prompt Construction for Finetuning

Our overall optimization target is to increase the number of daily unique "Professional Interactors", which is defined as

the number of unique members who take one or more "professional interaction" (PI) actions (e.g., long dwell, react, comment, repost, etc.) that contribute to the LinkedIn Feed's Knowledge Marketplace. Our training data comes from historical Feed interaction logs, and each row in our training dataset is a tuple of (target post features, member features, label), where the member and target post features are feature dictionaries and label is a binary label indicating whether the member took a PI on that post or not.

For the target post, we have the following features:

- type of the post (original post, group post, like/comment on a previous post)
- what is being shared (text, image, video, job change, etc.)
- author information (author name, profile headline, company, industry, title)
- post popularity features (# of times the post has been liked, viewed for more than T secs, etc.)
- article title/source (if the post contains an article link)
- text of the post

The member features include the following: name, profile headline & summary, industry, skill(s), location, job and education history, certifications, and languages spoken. For each member, we also have their "activity history sequence" – a time-ordered list of Feed posts on which the member has previously taken a PI. Note that we did perform several ablation studies for what posts to include in this history sequence:

- All posts (i.e., both positive and negative engagements)
- Only positive PI engagements
- No history (i.e., only include member profile information)

These ablation study results will be presented in the Results section. In the end, including only positive PI engagement in the activity history sequence performed the best, and is the method that we used.

A "prompt library" is used to convert these features into a target post prompt and member prompt, respectively.

The text of the target post prompt looks like:

```
<ST_P1>Post feature 1
<ST_P2>Post feature 2
...
<ST_PN>Post feature N
```

The text of the member prompt looks like:

```
<ST_M0>System prompt
<ST_M1>Member feature 1
<ST_M2>Member feature 2
...
<ST_MN>Member feature N
<ST_history>
  <ST_history_post><post 1 text>
  <ST_history_post><post 2 text>
  ...
  <ST_history_post><post H text>
```

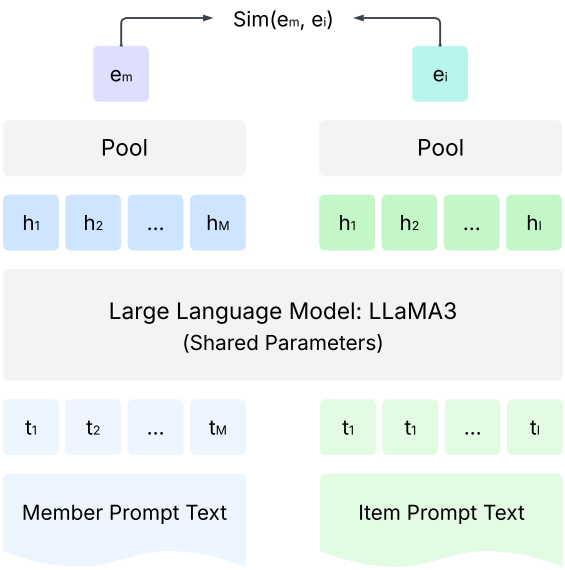

Figure 2: Dual encoder architecture utilizing a shared LLM for text-based retrieval. Member and item texts are processed separately through the LLM, generating token-level hidden representations. A pooling layer aggregates these representations into embeddings, which are then compared using a similarity function.

The `<post 1 text>, ..., <post H text>` inside of the member prompt are generated similarly to the target post's prompt. The number of history posts that fit into the prompt is dynamic based on the total maximum context length (20,480 in our experiments) and the tokenized length of each post.

The system prompt is the following: "You are provided with a member's profile information, along with a set of historical feed posts that the member engaged with. Your task is to analyze the historical engagement data along with the member profile."

Note that the `<ST_...>` strings above are special token strings that are added to the tokenizer's vocabulary so that they get tokenized as a single token, in order to reduce the size of the prompts. The strings shown above are just examples for explanatory purposes – the actual strings are not shared here for security purposes to discourage any prompt injection attacks.

As shown in Figure 2, the target post prompt and the member prompt are fed as input to an LLM in order to generate the target post and member embedding, respectively.

4 Modeling Architecture

We leverage a pre-trained, decoder-only transformer-based causal language model as our initial base model (Meta LLaMA-3 (Dubey et al. 2024)). We optimize the base LLM for embedding-based retrieval (EBR) through fine-tuning

using a dual-encoder architecture (with a single shared LLM) that is used to encode both members and items into a shared embedding space. Figure 2 depicts a high level illustration of the architecture.

Embedding Generation Both member and item texts undergo tokenization before being processed by the LLM. Given a tokenized sequence t of length L, the LLM produces a sequence of hidden states, $H \in \mathbb{R}^{L \times d}$ where d denotes the dimensionality of the hidden states. A pooling function is subsequently applied to generate a fixed-dimensional, dense representation. Specifically, the embedding for a member token sequence t_m is computed as: $e_m = pool(H_m)$. An analogous process is used to generate an item embedding e_i. We discuss pooling functions further in subsection 4.1.

Measuring Member-Item Similarity The similarity between member and item embeddings is quantified using a similarity function, $s(e_m, e_i)$. In this study, cosine similarity is employed: $s(e_m, e_i) = \frac{e_m \cdot e_i}{||e_m|| \cdot ||e_i||}$. This similarity score serves as the primary retrieval ranking metric, enabling efficient identification of the most relevant items for a given member.

4.1 Pooling

The pooling function aggregates the token-level hidden states into the fixed-dimensional, dense embeddings.

Mean Pooling: Given an input sequence consisting of L tokens with hidden states $H \in \mathbb{R}^{L \times d}$ (with H_i as the i-th token), the pooled embedding is

$$e = \frac{1}{L} \sum_{i=1}^{L} H_i.$$

This method yields a holistic representation by averaging over tokens. In this work, we considered mean pooling various sequence lengths to land on the best performing model and results are discussed in subsection 8.5

4.2 Training Objectives

When fine-tuning for the retrieval task, we wish to optimize an objective where embeddings of positive member-item pairs are drawn closer together, while negative pairs are pushed apart in the embedding space. For training, we leverage binary data that captures whether a positive or negative action was taken between a member and an item. In this setting, each member–item pair is assigned a binary label $y \in \{0, 1\}$. We explore the following loss functions that leverage the labelled data to effectively learn embedding similarities.

Binary Cross-Entropy (BCE): In this formulation, the similarity score $s(e_m, e_i)$ is scaled by the temperature τ and interpreted as a logit. The corresponding probability is computed using the sigmoid function $\sigma(\cdot)$:

$$P(y = 1 \mid e_m, e_i) = \sigma\left(\frac{s(e_m, e_i)}{\tau}\right).$$

The BCE loss is then expressed as:

$$\begin{aligned} L_{\text{BCE}} = -\Big[& y \log \sigma\left(\frac{s(e_m, e_i)}{\tau}\right) \\ & + (1 - y) \log\left(1 - \sigma\left(\frac{s(e_m, e_i)}{\tau}\right)\right)\Big] \end{aligned} \tag{1}$$

InfoNCE: For a given member embedding e_m and a corresponding positive item embedding e_i^+, along with a set of negative item embeddings $\{e_i^-\}$, the InfoNCE loss (Oord, Li, and Vinyals 2018) is defined as:

$$L_{\text{InfoNCE}} = -\log \frac{\exp\left(\frac{\text{s}(e_m, e_i^+)}{\tau}\right)}{\exp\left(\frac{\text{s}(e_m, e_i^+)}{\tau}\right) + \sum_j \exp\left(\frac{\text{s}(e_m, e_{i_j}^-)}{\tau}\right)} \tag{2}$$

where $\text{s}(\cdot, \cdot)$ denotes the similarity function used, and τ is a temperature parameter. This loss encourages the similarity of positive pairs to be higher than that of negative pairs by emphasizing relative ranking. InfoNCE is more commonly used for retrieval and we expected this to perform better. Binary Cross Entropy Loss served as a good baseline for us to compare the performance of the model with the infoNCE loss.

4.3 Easy and Hard Negative Mining

The negatives used for the InfoNCE loss described in Equation 2 are a combination of easy and hard negatives. Following Google's two-tower retrieval literature (Yang et al. 2020), we mix easy and hard negatives. Mixed Negative Sampling (MNS) combines batch (in-batch) negatives with uniformly sampled corpus negatives to reduce selection bias in implicit feedback data and has shown offline and online gains in large-scale production (e.g., Google Play). Our setup extends this idea by adding per-member hard-negative mining on top of in-batch sampling.

In each training step, in-batch negatives are sampled from the global mini-batch (as opposed to the local mini-batch on each individual GPU). These examples provide weak negative pairs, artificially creating impressions with no action, improving training stability and increasing the number of training examples seen by a factor of batch size[2].

We also built tunable parameters to dynamically sample hard negatives per batch for each individual member, in addition to the aforementioned easy negatives. Operationally, our sampler mirrors triplet training: for each anchor–positive, we mine K hard negatives (near-miss impressions for that member) and sample J easy negatives (global in-batch). Triplet loss maximizes a margin between the anchor–positive and anchor–negative; InfoNCE replaces the margin with a softmax over the pooled negatives, which often yields smoother optimization at scale (Schroff, Kalenichenko, and Philbin 2015).

- **Easy Negatives** Negatives sampled from the global batch (across all GPUs)
- **Hard Negatives** Items which were impressed by a member without an engagement action. We create this map

offline and store it in memory for sampling required number of hard negatives at training time.

4.4 Matryoshka Embeddings

As part of our training procedure, we also employ Matryoshka Representation Learning (MRL) (Kusupati et al. 2022), a framework that learns nested, size-adaptive representations by optimizing multiple sub-representations simultaneously. At each level, progressively larger subsets of the embedding are encouraged to capture increasingly rich and informative features. This property is advantageous in production environments, where models must often operate under varying computational or memory constraints. By ensuring that smaller sub-representations remain effective for the downstream task, MRL enables flexible, efficient deployment without the need for retraining or architecture modifications. The learning process is guided by the Matryoshka Loss, defined as:

$$\mathcal{L}_{\text{MRL}} = \sum_{k=1}^{K} \lambda_k \cdot \mathcal{L}_k \tag{3}$$

where K denotes the total number of representation sizes, λ_k are weighting coefficients, and $\mathcal{L}_k$ is the task-specific loss computed using only the first k dimensions of the representation. In our use case, we average the infoNce loss that is computed for each of first 'k' dimensions.

5 Offline Evaluation

We evaluate our retrieval model using **Recall@**k, treating the final ranking model as the oracle. The evaluation proceeds as follows:

1. Randomly sample a set of unique members who have had engagement in the LinkedIn Feed.
2. For each member, rank all N items from their aggregated sessions using the final ranking model, and save the top n suggested content items in Set_1.
3. Rank the top n items per member using similarity scores from the retrieval model (e.g., cosine similarity between query and item embeddings), and save the top k items in Set_2.
4. Compute Recall@k as:

$$\text{Recall@}k = \frac{|Set_1 \cap Set_2|}{|Set_1|}$$

   where Set_1 contains the top-n items according to the ranking model, and Set_2 contains the top-k items according to the retrieval model.
5. Average the Recall@k values across all members to obtain the final metric.

We computed recall at the member level rather than the session level because the number of suggested content items viewed within a single session is typically very small. This limited interaction scope may not provide a comprehensive assessment of the performance of the retrieval model. By evaluating recall over all suggested content items that a member has engaged with within a defined time period (e.g., one week), we obtain a more holistic measure of the model's ability to retrieve relevant content across multiple interactions. This approach aligns with our objective of developing a retrieval model that effectively serves the interests of our members.

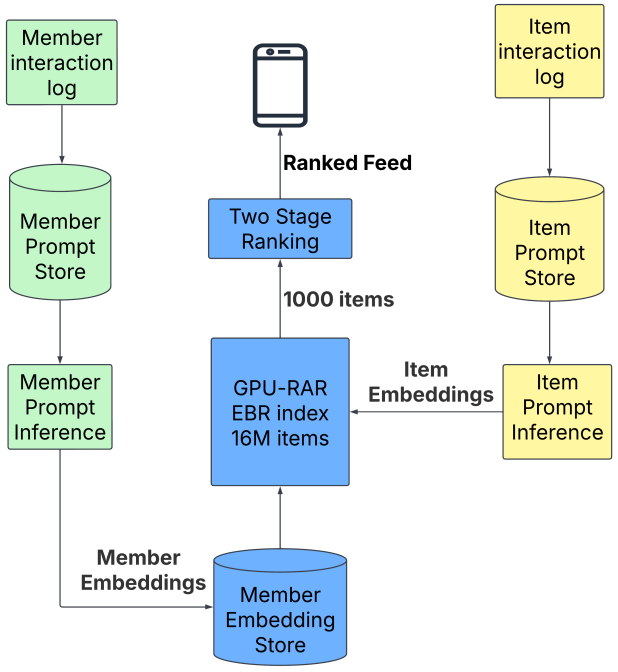

Figure 3: Online System for the Retrieval and Ranking Model

6 Online System

For running online retrieval for serving member queries, we built multiple workflows, as we detail next.

Nearline Item & Member Activity Log Generation When items or member profiles are created or updated on LinkedIn, we capture those triggers into an item and member activity log. When members interact with items (e.g. like, comment, share, etc.), we capture those interactions too by processing tracking data from clients into the item and member activity logs. We minimize the latency in capturing these interactions by using direct RPC calls for service-to-service communication, where possible, over nearline stream processing.

Nearline Item & Member Prompt Generation Next, we process the item and member activity logs into corresponding item and member prompts using pre-defined prompt templates where data such as item text, member profile information, and item popularity counts are fetched and populated into the templates to construct prompts that include interaction history. This ensures freshness of prompt data. Finally, we push these fully decorated prompts to a key-value store for online access during retrieval and to a near-

line stream processor for generating embeddings.

Nearline Item & Member Embedding Generation Using Online LLM inference We feed updated item prompts for each item creation and item update into a LLM inference server hosting our fine-tuned LLM to generate embeddings as described in Section 4. We ingest the generated item embeddings into a GPU index for online kNN (k nearest neighbor) retrieval (Borisyuk et al. 2024). The GPU index currently lets us define a custom pytorch model in order to do the kNN operation. For this specific use case, we use a simple cosine similarity model and the top 'k' documents with the highest cosine similarity scores with the member embedding are retrieved from the index. We refer to this as a "GPU Retrieval as Ranking" (GPU-RAR) index. Similarly, we feed updated member prompts for each member profile creation and member interaction activity to the LLM inference server to generate embeddings and ingest them into an online key-value store for access during online retrieval. This embedding generation process is done using nearline stream processing to control the LLM inference rate by batching updates in configurable window sizes because our scale results in thousands of input prompt updates per second. We trade-off GPU compute used against embedding freshness by using shorter window sizes for increased freshness which helps capture evolving member interests and item popularity. We ensure that newly created items are indexed in the GPU-RAR index within a minute of creation and interactions on existing items result in an update of their embeddings within 30 minutes. Similarly, newly added member profiles are captured in their member query embeddings within a minute and activities by existing members result in an update of their embeddings within 30 minutes.

Online GPU-RAR kNN Retrieval with Attribute-Based Matching To serve an online LinkedIn feed query for a member, we fetch member query embeddings and run online kNN against the GPU-RAR item embeddings index to retrieve top K items, while applying business logic filtering and privacy rules, that are then sent to the ranker layer. We apply filters as part of the kNN query to ensure that the selected items are approved by our trust classifiers, match the languages understood by the viewer, not authored by members blocked by the viewer, and not already seen by the viewer. The pre-computation of embeddings allows us to achieve sub-50ms retrieval latency for serving tens of thousands of queries per second on a corpus of hundreds of millions of items while maintaining embedding freshness of a few minutes.

7 Implementation Details

We used 5M member-item pairs from public engagement in the LinkedIn Feed as our training samples. We used 8 H100 GPUs for each training run with a per-GPU batch size of 4. We experimented with both Meta-LLaMA 3B and 1B parameter models with various combinations of Matryoshka Embeddings.

We used a cluster of 24 GPUs for indexing item embeddings and for performing online GPU-RAR kNN retrieval with attribute-based matching. We used a retrieval model that employed cosine similarity between member and item candidates to get the top 1000 candidates to feed to subsequent layers of the ranking stack.

8 ResultsCorporation

The Meta LLaMA-3 model with 3B parameters was leveraged as the base model for further fine-tuning to get all the results discussed in this section. This model by default has an output dimension of 3072 unless specified otherwise.

8.1 Dual Encoder Results

Loss	Recall@10
Random	0.0700
Llama-3 Without Finetuning	0.2434
BCE	0.3944
InfoNCE	0.4238
InfoNCE with Matryoshka Loss (full 3072 dim)	0.4242

Table 1: Results for the Dual Encoder architecture under different training objectives and using all dimensions

From Table 1, we observe that the InfoNCE loss is able to outperform BCE loss. We also observe that using Matryoshka Representation Learning to learn multiple dimensionalities together does not hurt overall performance.

8.2 Dual Encoder Results with Matryoshka Learning

One of the key motivations to do Matryoshka learning was to reduce the dimensionality of the final embedding. This enables us to reduce the storage cost in the GPU index from which documents are retrieved in the online system. We trained a model with the method described in subsection 4.4 and evaluated multiple embedding dimensions in parallel.

Embedding Dimensions	Recall@10
3072 (all dims)	0.4242
2048	0.4248
1024	0.4237
512	0.4225
50	0.3716

Table 2: Evaluating performance on reducing embedding size after training with Matryoshka Loss.

The plot in Figure 4 demonstrates that lowering the dimension to 512 does not significantly impact the recall numbers and offers a lot of potential to reduce the storage size of the final embeddings.

Comparison of Matryoshka Learning Against Directly Lowering Dimension Using MLPs As illustrated in Table 3, it is possible to fine-tune a model with lower dimensional embeddings by adding MLPs to the final layer. We experimented with 2 methods: 1) pooling and then using an MLP to reduce the dimension, and 2) reducing the dimension before pooling. From the results(recall@10), we observed that

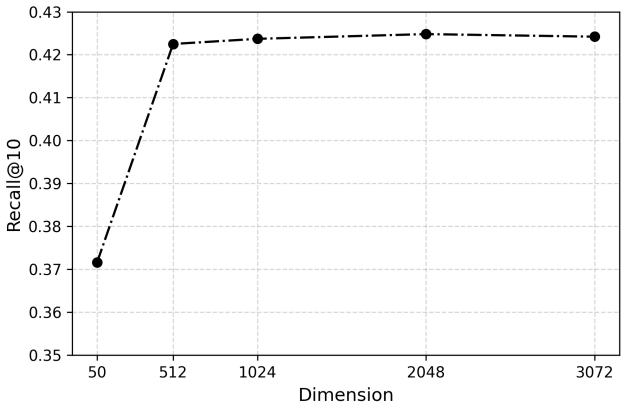

Figure 4: Results for Recall@10 versus Embedding Dimension when training using MRL.

MRL outperformed pooling to a specific dimension using MLPs.

Dimension	Pre-pool	Post-pool	MRL
2048	-3.2%	-2.7%	**+0.1%**
1024	-2.7%	-2.3%	**-0.2%**
512	-1.1%	-1.0%	**-0.8%**

Table 3: Percentage difference in Recall@10 for different embedding dimension sizes compared to using the full 3072 dimensions. We consider a pre-pooling projection (project to desired dimension then pool embeddings), post-pooling projection (pool embeddings and then project to desired dimension) and applying MRL.

8.3 Effect of Hard Negatives on Recall Metrics

Our experiment results clearly demonstrate that adding hard negatives helps to improve recall metrics (Table 4). We are exploring ways to introduce more hard negatives without hurting recall metrics too much. Some ideas currently being worked on are progressively increasing number of hard negatives during training, curriculum learning by increasing hard negatives in an iterative way

Hard Negatives	Recall@10
Easy negatives only	Baseline
Easy negatives + 1 hard negative/member	+2.0%
Easy negatives + 2 hard negatives/member	+3.6%

Table 4: Effect of adding per-member hard negatives on Recall@10.

8.4 Effect of Filtering for Positives in Interaction History

When we removed all negative engagements from the prompt history and instead extended more positive items that were engaged on, we saw a significant improvement in recall metrics (Table 5). In all subsequent iterations, negative interactions were removed from the member prompt.

Loss	Full History Recall@10	Pos-Only History Recall@10
BCE	0.307	0.3944
InfoNCE	0.398	0.4238

Table 5: Recall@10 using different member history strategies.

8.5 Effect of Last-N Pooling

We share our findings on pooling different set of last 'n' tokens and compare it against pooling all tokens in the last layer in this section:

Last_n tokens used	Percent
All	Baseline
500	-7.0603%
250	-5.0040%
50	-11.2637%
1	-12.3594%

Table 6: Percent change across different Last_n values.

From Table 6, we noticed that the baseline (pooling the embeddings of all the tokens) performed the best. Pooling any other subset resulted in recall@10 drops.

8.6 Can LLMs Capture Count Features that are Important for Ranking?

In the course of our work, we came up with a framework with which we could enable the final embeddings to encode important count-based features that are used in the final ranking model. This was critical for the overall success of the retriever and we will share some of our learnings in this section. One of the most important metrics for the retriever is how well it aligns with the ranking model. Hence, if some of the important features for the ranking model cannot be captured in the embedding space, this would result in subpar candidates sent to the ranking model resulting in engagement drops.

Results of Adding Engagement Rate to Input Text and Truncating Feed Post Length One of the key lessons we learned was that quantizing the count features and passing them to the input prompt ensured better correlation between the final cosine similarity score and the feature value. We demonstrate this with a simple study we did using the item popularity counts, which is an important feature for the final ranking model using a smaller dataset.

- For the **Baseline** in Table 7, inputs had the entire post text and the raw popularity counts of the posts in the engagement history of members (query side) and in the post text (candidate side). Popularity counts could go to pretty large values (tens of thousands).

- For the **Candidate Model** in Table 7 all post texts in the query and item side were truncated to the first 60 tokens instead of considering the whole post and popularity rates (range of 1-100 percent) were added as features on top of raw counts.

Model	Correlation between popularity counts and cosine similarity scores	Recall@10
Baseline	-0.0037	0.158
Candidate Model	0.1156	0.1839

Table 7: Results for measuring correlation between retrieval scores and popularity percentages

The column titled Correlation in Table 7 refers to correlation between item popularity counts and cosine similarity score between the embeddings. From the results in Table 7, there is evidence (improved correlation and recall improvements by 15%)that adding count-based features important for the ranking model in a quantized manner to the input prompt can be captured by the embeddings we generate for retrieval. This aligns with the fact that the Llama-3 tokenizer groups 3 digits together as a single token (Arnett 2024). Since the counts were now captured by a single token, this could be an added factor in the feature correlation getting better.

8.7 Online Ramp Results

We used all the learnings from the offline experimentation and used the following set of hyperparameters to prepare a model for the online ramp:

- Llama 3B base model with 3072 dimensions in last layer
- Use only positive interactions in member prompt history
- Mean pool embeddings of all tokens
- Employed 2 hard negatives for a per GPU batch size of 4
- Quantized popularity features to percentages

We A/B tested the new LLM-based retriever against the existing suggested content retrieval model (Figure 1) and the final ranking stages were identical for both setups.

We observed significantly improved suggested content recommendations, which were served to our members, and as expected we impacted members with few connections and new members to LinkedIn substantially more in addition to overall metric improvements online.

- **Revenue** Increased by **+0.8%** (pval of 0.03) as a result of increased scroll depth and member interaction in the LinkedIn Feed
- **Daily Unique Professional Interactors** Increased by **+0.2%** (pval of 0.005) in the LinkedIn feed

Inspecting the metrics among member cohorts with lower liquidity of candidate posts and fewer connections to other members, we observed the following metric impacts:

- **Daily Active Unique Users** Increased by **+0.23%** (pval of 0.05)
- **Daily Unique Professional Interactions** Increased by **+1.17%** (pval less than 0.0001)
- **Revenue** Increased by **+3.29%** (pval of 0.03) for this user group, indicating that the majority of platform-wide impact came from infrequent members and members with fewer connections, for whom suggested content plays a much more vital role.

9 Conclusion

In this paper, we presented the redesign and implementation of a modern retrieval system for the LinkedIn feed. We explored modeling choices, loss functions, and pooling strategies, and demonstrated through offline experiments and online A/B tests that fine-tuning large language models can substantially improve the quality of recommended content for our members.

Looking ahead, we plan to improve handling of unimpressed content at the retrieval stage to encourage exploration, and to investigate more effective distillation strategies for deriving dual encoders from cross encoders. Building on the efficacy of matryoshka learning, we are working on reducing embedding dimensionality to lower storage costs. We are also exploring LLM-powered embeddings for content generated by members' connections, which represents the bulk of impressed items in the LinkedIn feed.

On the efficiency side, we are investigating methods to shorten input sequences to improve GPU throughput in our nearline system. Finally, we have begun prototyping user-prompt–driven feed recommendation as a re-ranking layer, which is already showing promising early results.

Acknowledgements

This work represents the joint efforts across multiple teams in LinkedIn without whom this would not have been possible. We would like to thank (in alphabetical order) Adrian Englhardt, Aman Gupta, Ata Fatahi Baarzi, Bhargav Patel, Bo Wang, Chinmay Nayak, Christine Lin, Dhritiman Das, Dre Olgiati, Frank Shyu, Ganesh Parameshwaran, Ghulam Ahmed Ansari, Hemeng Tao, Hristo Danchev, Jashua Gupta, Jason Ko, Kirill Talanine, Lars Hertel, Mingzhou Zhou, Mohit Kothari, Pratik Dixit, Qing Lan, Qingquan Song, Samira Sriram, Samaneh Moghaddam, Satyam Kumar, Steven Shimizu, Sundararaman Ramachandran, Tejas Dharamsi, Tim Chao, Tim Jurka, Tugrul Bingol, Vignesh Kothapalli, Vishal Shah, Ying Xuan, Youngchae Kim and Yun Dai for supporting this work.

References

Andoni, A.; Indyk, P.; Laarhoven, T.; and Neumayer, R. 2020. ScaNN: Efficient Vector Similarity Search. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '20)*, 1217–1227.

Arnett, C. 2024. wHy DoNt YoU jUsT uSe ThE lLaMa ToKeNiZeR?? *Hugging Face Community Blog*. Community article.

Borisyuk, F.; Song, Q.; Zhou, M.; Parameswaran, G.; Arun, M.; Popuri, S.; Bingol, T.; Pei, Z.; Lee, K.-H.; Zheng, L.; et al. 2024. LiNR: Model Based Neural Retrieval on GPUs at LinkedIn. *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '24)*.

Casalegno, F. 2025. Fine-Tuning LLMs for Report Summarization: Analysis on Supervised and Unsupervised Data. ResearchGate.

Covington, P.; Adams, J.; and Weinberger, E. 2016. Deep Neural Networks for YouTube Recommendations. *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*, 191–198.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv e-prints*, arXiv–2407.

Feng, Y.; Zhang, Y.; and Ji, S. 2023. Leveraging Large Language Models in Conversational Recommender Systems. *arXiv preprint arXiv:2306.07548*.

Gao, Y.; Huang, T.; Zhang, Y.; and Ji, S. 2023. Generative Recommendation with Large Language Models. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23)*, 3862–3871.

Huang, C.; Fu, Z.; Zhang, Z.; Yuan, Q.; and Ren, X. 2023. A Survey of Large Language Models for Recommendation. *arXiv preprint arXiv:2308.01912*.

Johnson, J.; Douze, M.; and Jégou, H. 2019. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 30(5): 1083–1097.

Karpukhin, V.; Oğuz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Grave, E.; and Yih, W.-t. 2020. Dense Passage Retrieval for Open-Domain Question Answering. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6718–6731.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8): 30–37.

Kusupati, A.; Bhatt, G.; Rege, A.; Wallingford, M.; Sinha, A.; Ramanujan, V.; Howard-Snyder, W.; Chen, K.; Kakade, S. M.; Jain, P.; and Farhadi, A. 2022. Matryoshka Representation Learning. In *Neural Information Processing Systems*.

Liu, Y.; Fan, Y.; Deng, Z.; Fu, R.; and Luo, J. 2024. Taxonomy-Guided Zero-Shot Recommendations with LLMs. *Proceedings of the 2025 International Conference on Computational Linguistics (COLING 2025)*.

Oord, A. v. d.; Li, Y.; and Vinyals, O. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Ouyang, L.; Wu, J.; Jiang, X.; Kelly, D.; Datta, K.; Xiao, N.; T. Du, K.; K. W. Lo, S.; Ren, J.; H. Huang, M.; Paek, B.; Lowe, A.; S. Li, C.; and Amodei, D. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744.

Prasad, A. 2025. Netflix's Transformer Revolution: How LLMs Are Reshaping What You Watch Next. Accessed: 2025-08-12.

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. *Proceedings of the 10th International Conference on World Wide Web (WWW '01)*, 285–295.

Schroff, F.; Kalenichenko, D.; and Philbin, J. 2015. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 815–823. IEEE Computer Society.

Swapnil Ghike, S. G. 2016. FollowFeed: LinkedIn's Feed Made Faster and Smarter. *LinkedIn Engineering Blog. Available online: https://www.linkedin.com/blog/engineering/feed/followfeed-linkedin-s-feed-made-faster-and-smarter*.

Wang, Y.; Li, Y.; and Li, S. 2018. Understanding and Predicting User Engagement with Social Media Feeds. *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '18)*, 209–218.

Yan, E. 2024. Improving Recommendation Systems & Search in the Age of LLMs. Keynote at AI Engineer World's Fair, San Francisco.

Yang, J.; Yi, X.; Cheng, D. Z.; Hong, L.; Li, Y.; Wang, S. X.; Xu, T.; and Chi, E. H. 2020. Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations. In Seghrouchni, A. E. F.; Sukthankar, G.; Liu, T.; and van Steen, M., eds., *Companion of The 2020 Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 441–447. ACM / IW3C2.

Ying, R.; Cai, T.; Chen, T.; Hao, K.; Wang, Y.; and Lu, Z. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18)*, 976–985.

Zhai, C. 2024. Large Language Models and Future of Information Retrieval: Opportunities and Challenges. *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024)*.