The Efficiency vs. Accuracy Trade-off: Optimizing RAG-Enhanced LLM Recommender Systems Using Multi-Head Early Exit

Huixue Zhou [1,2], **Hengrui Gu** [3], **Zaifu Zhan**[2], **Xi Liu** [1], **Kaixiong Zhou** [3], **Mingfu Liang** [1], **Yongkang Xiao** [2], **Srinivas Govindan** [1], **Piyush Chawla** [1], **Jiyan Yang** [1], **Xiangfei Meng** [1], **Huayu Li** [1], **Buyun Zhang** [1], **Liang Luo** [1], **Wen-Yen Chen** [1], **Yiping Han** [1], **Bo Long** [1], **Rui Zhang** [2], **Tianlong Chen** [4]

Meta Platforms[1], University of Minnesota[2], NCSU[3], UNC at Chapel Hill[4]

zhou1742@umn.edu, xliu1@meta.com, tianlong@cs.unc.edu

Abstract

The deployment of Large Language Models (LLMs) in recommender systems for Click-Through Rate (CTR) prediction requires a careful balance between computational efficiency and predictive accuracy. This paper introduces **OptiRAG-Rec**, a comprehensive framework that integrates Retrieval-Augmented Generation (RAG) with a novel multi-head early exit architecture to address both challenges. By leveraging Graph Convolutional Networks (GCNs) as efficient retrieval mechanisms, the framework significantly reduces data retrieval times while maintaining high model performance. Additionally, the multi-head early exit strategy dynamically terminates inference based on real-time predictive confidence assessments, enhancing responsiveness without sacrificing accuracy. Experimental results demonstrate that **OptiRAG-Rec** reduces computation time while preserving the precision required for reliable recommendations, establishing a new benchmark for efficient and accurate LLM deployment in recommendation. [1]

1 Introduction

Due to their remarkable capabilities in semantic understanding and knowledge retention, Large Language Models (LLMs) have demonstrated impressive performance across various domains, becoming essential components of various text-based recommendation systems, such as sequential recommendation (Wang et al., 2023a; Harte et al., 2023; Li et al., 2023; Zheng et al., 2024) and ranking (Zhao et al., 2024; Acharya et al., 2023). In light of this, numerous researchers have sought to adapt LLMs for Click-Through Rate (CTR) prediction problem (Wang and Lim, 2023; Bao et al., 2023a; Lin et al., 2024a), leveraging their text-mining capabilities to analyze textual user behaviors for more accurate preference modeling.

[1]Disclaimer: No internal or proprietary Meta data was used in this study.

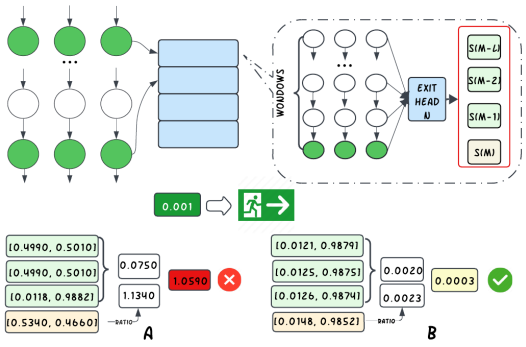

Figure 1: An example of multi-head early exit based on ratio scores of positive and negative predictions: **A**: No early exit due to the significant difference between the average ratio of previous layers and the exit layer exceeding the threshold (0.001). **B**: Early exit achieved with minimal changes in ratio scores.

The performance of LLM-based CTR predictors is enhanced by Retrieval-Augmented Generation (RAG) modules (Lin et al., 2024b; Hajiaghayi et al., 2024), which integrate diverse and user-relevant texts into the input contexts. This enrichment allows LLMs to extract more accurate user features for preference prediction.

However, the integration of RAG modules into LLMs introduces significant efficiency challenges, particularly concerning the Inference efficiency. Two critical bottlenecks arise from the introduction of RAG modules: ❶ **Retrieval Efficiency**: The RAG framework adds a multi-stage process involving encoding, retrieval, and prefilling steps for retrieved contexts. This sequential execution causes substantial delays due to the time required for retrieval, thus hindering the prompt commencement of the inference process. ❷ **Inference Overhead**: With extended input lengths, the computational demand surges, exacerbating the model's inference time due to the quadratic time complexity of LLMs relative to input length. This significant increase

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26443–26458

in computational overhead severely impacts the real-time responsiveness of the model, limiting its practical applicability in real-world scenarios.

In the realm of enhancing LLM inference efficiency, prevalent techniques such as model compression through quantization and Fast Decoding Algorithms like Speculative Decoding (e.g., Medusa(Cai et al., 2024) and Kangaroo(Liu et al., 2024b)) are widely employed. These methods aim to reduce model size and accelerate the decoding process, thereby shortening overall inference times. Despite their benefits, these strategies often come with trade-offs, particularly in terms of model accuracy(Xia et al., 2024; Cai et al., 2024; Jin et al., 2024). Quantization, for instance, achieves faster processing by reducing the precision of computations, which can lead to the loss of critical data details. Similarly, Fast Decoding Algorithms may prioritize speed over optimality, potentially selecting suboptimal predictive paths in their rush to generate quick responses. While these approaches address certain aspects of inference efficiency, they do not fully resolve the bottlenecks introduced by RAG modules, such as retrieval delays and increased computational overhead from extended input lengths. Consequently, there remains a need for more holistic solutions that balance efficiency, accuracy in RAG-enhanced LLMs.

To address these challenges, we propose **OptiRAG-Rec**, a novel framework designed to enhance the efficiency of LLM-based CTR while maintaining high recommendation quality. This framework integrates two innovative acceleration techniques tailored to the identified bottlenecks: retrieval efficiency and inference overhead.

First, to address retrieval efficiency (Bottleneck ❶), we introduce GCN-Retriever, a lightweight yet highly effective retrieval scheme. Leveraging the capabilities of Graph Convolutional Networks (GCNs) (Kipf and Welling, 2016; Chen et al., 2020), GCN-Retriever models structural data in user-item graphs, capturing multi-order interaction information to generate precise and informative representations. By replacing computationally expensive LLM embedders, GCN-Retriever significantly reduces retrieval times, minimizing inference delays while maintaining high recommendation performance. Second, to tackle inference overhead (Bottleneck ❷), we propose a multi-head early exit strategy integrated with an exit scoring mechanism tailored for CTR prediction tasks. This strategy dynamically terminates the inference process when predictions meet a confidence threshold, reducing unnecessary computational overhead. By doing so, the system accelerates response times without sacrificing recommendation accuracy. The components of **OptiRAG-Rec**—GCN-Retriever, the LLM-based recommender, and the multi-head early exit mechanism—are tightly integrated to address specific challenges: GCN-Retriever enhances retrieval efficiency using graph-based representations, the LLM-based recommender ensures precise predictions, and the multi-head early exit mechanism reduces computational overhead during inference. Together, they synergistically improve the recommender system's performance by balancing efficient data retrieval, accurate predictions, and optimized throughput. Below, we summarize the key contributions of our framework:

- **Enhanced LLM for CTR Prediction via RAG**: Improves CTR accuracy by integrating interaction data into LLM models.
- **Efficient GCN-Retriever**: Introduces a lightweight GCN-based retrieval mechanism, reducing retrieval latency without compromising recommendation quality.
- **Inference Time Optimization through Early Exit**: Implements a dynamic early exit strategy, reducing computational costs by terminating inference at intermediate layers when predictions meet confidence thresholds.
- **Novel Multi-Head Early Exit Adjustment**: Proposes a multi-head architecture for early exiting, maintaining accuracy while significantly improving inference efficiency.

2 Related Work

2.1 Language Models for Recommendation

Building on prior research(Wu et al., 2023), the integration of language models into recommender systems often focuses on their distinct functions within the recommendation process. These roles include serving as feature extractors(Bian et al., 2022; Zheng et al., 2023; Zhang et al., 2024a), where language models analyze item and user data to produce embeddings or tokens. These embeddings can be utilized by traditional recommender system models to enhance task-specific recommendations through knowledge-aware embeddings.

Furthermore, language models can function within scoring or ranking mechanisms(Wang et al.,

2023a; Zhu et al., 2024; Kim et al., 2024). This approach leverages pre-trained language models to transform recommendation systems significantly. Typically, the input sequence includes task instructions, behavioral prompts, with the output generating pertinent recommendation results.

Our methodology diverges from previous practices by employing the language model primarily within the scoring functions, while using a simple traditional model, specifically a GNN, as a retriever. This model extracts similar user profiles to construct the prompts for the language model, capitalizing on its ability to comprehend and synthesize user data and interactions, thereby generating personalized recommendations.

2.2 Efficient Inference

The inference performance of LLMs is often constrained by the sequential nature of auto-regressive decoding, where the generation of each token necessitates a full network forward pass. To address the high inference latency inherent in LLMs, several strategies have been proposed: Techniques such as quantization (Fan et al., 2020; Bai et al., 2022; Tao et al., 2022), pruning (Ma et al., 2023; Sun et al., 2023; Xia et al., 2023; Frantar and Alistarh, 2023a), and knowledge distillation (Liang et al., 2023; Sahu et al., 2023; Gu et al., 2024), aim to reduce the memory footprint of LLMs, thus lowering the computational demands. Early Exit Strategies allow a model to terminate the computation at intermediate layers if certain conditions are met, thereby accelerating inference and reducing computational overhead. Early exit has been explored across various machine learning domains, focusing on designing efficient early exit networks (Bae et al., 2023; Chen et al., 2023), and refining exit rules to balance accuracy and computational efficiency(Zhou et al., 2020; Li et al., 2021).

3 Preliminaries

LLM Architecture and Decoding Process. The typical architecture of LLMs sequentially consists of N transformer layers and a language head, denoted as $\text{Head}(\cdot)$, for decoding the next token. Given an input sequence of tokens $\{x_1, \ldots, x_{t-1}\}$, the standard decoding process can be formally described as follows:

$$p(x_t|x_{<t}) = \text{softmax}(\text{Head}(h_{t-1}^{(N)}))_{x_t} \quad (1)$$

where $h_{t-1}^{(N)}$ denotes the final hidden state output by the N-th (i.e., last) layer, while $p(x_t|x_{<t})$ represents the conditional distribution for sampling the next token.

LLM-Based CTR Prediction. We instruct LLMs to solve CTR prediction as a binary classification problem. Specifically, each task sample x_i is re-expressed into its natural language form x_i^{text}. Likewise, its corresponding binary label $y_i \in \{0, 1\}$ is mapped into a pair of binary response words $y_i^{\text{text}} \in \{\text{“yes”}, \text{“no”}\}$. To obtain the LLM's tendencies between these two options, and consistent with prior work (Wang and Lim, 2023; Bao et al., 2023a; Lin et al., 2024a), we consider only the binary response tokens (i.e., “yes” and “no”) as candidates, excluding all other tokens in the vocabulary. This approach enables the extraction of the LLM's predictive tendencies between these two responses, as illustrated below:

$$p(\text{yes}|x_{<t}) = \frac{\exp(s^{\text{yes}})}{\exp(s^{\text{yes}}) + \exp(s^{\text{no}})}.$$

where s denotes the logit score for the given token, while $p(\text{yes}|x_{<t})$ quantifies the LLM's preference for outputting the token “yes”. Naturally, $p(\text{yes}|x_{<t}) > 0.5$ is an intuitive and appropriate condition to finalize the outputted token.

4 Retrieval Efficiency: GCN-Retriever

To enable accurate learning and prediction, we construct textual sentences, denoted as X_{text} by integrating instructions, representative examples, and user input data. This process ensures the model aligns with the specific requirements and contexts of the task. For instance, in a product recommendation system, X_{text} is constructed as follows: 1) **Instruction:** Provide clear directives, such as "Predict whether the user will click on the given item." 2) **Examples:** include contextual examples, e.g., "User A rated Book X with 5 stars." 3) **Input:** Incorporate real-time user interactions and queries. Complete prompt templates for each dataset are detailed in Appendix A.1.

To address the challenge of time efficiency in recommendation systems, we propose GCN-Retriever, a streamlined approach leveraging GCNs. This model constructs a bipartite graph where nodes represent users and items, and edges denote interactions between them.Through multi-layer message passing, GCN-Retriever refines user embeddings by incorporating neighborhood information. Specifically, the embedding for a user u at the next layer k +1 is updated by aggregating features from connected neighbors:

$$e_u^{(k+1)} = \text{AGG}\left(e_u^{(k)}, \{e_i^{(k)} : i \in N_u\}\right)$$

where e_u is the embedding of user u, N_u denotes the neighbors of u, and AGG represents the aggregation function which combines features of a node with those of its neighbors.

To effectively capture the multidimensional signal of users, our GCN-Retriever model employs a strategy of averaging the embeddings obtained from different layers of GCNs. This approach provides a comprehensive representation that integrates diverse aspects of user behavior and attributes captured at the various levels of graph structure. The average of user embeddings is described by the following equation:

$$\overline{e_u} = \frac{1}{K}\sum_{k=1}^{K} e_u^{(k)}.$$

where $\overline{e_u}$ represents the final averaged embedding for user u, K is the number of layers from which embeddings are extracted and averaged, and $e_u^{(k)}$ is the embedding of user u at layer k. Finally, cosine similarity is used to measure the similarity between users based on their averaged embeddings. The complete algorithmic workflow, including embedding aggregation strategies and similar user retrieval process, is detailed in Algorithm 1 (see Appendix A.2).

5 Inference Acceleration: Dynamic Predictive Exiting

In this section, we introduce **Dynamic Predictive Exiting** as a solution to Inference Slowdown. Motivated by (Xin et al., 2021), this mechanism leverages additional language heads to enable flexible inference termination while maintaining prediction quality. Specifically, during the forward pass through model layers, these language heads, attached to designated exit layers, decode the intermediate hidden states into next-token distributions. We design straightforward yet effective strategies to dynamically monitor the prediction confidence at different layers, using it as a real-time criterion to determine when to terminate inference and accept these intermediate distributions as final outputs.

This early exit methodology has proven effective in capturing the evolving dynamics of prediction preferences across different model layers in LLMs, functioning efficiently even without specialized training (Chuang et al., 2024; Kao et al., 2020; Schuster et al., 2022). By applying the language head to the immature hidden states of the intermediate layers, we can calculate the probability of the next token solely conditioned on $h_{t-1}^{(j)}, j \in \{0, \ldots, N-1\}$, without finishing the entire inference process:

$$p^{(j)}(x_t|x_{<t}) = \text{softmax}(\text{Head}(h_{t-1}^{(j)}))_{x_t} \quad (2)$$

Despite the advantages of this layer-wise predictive analysis in LLMs, it is widely recognized that intermediate hidden states typically present a significant information gap (i.e., distribution shift) compared to the final hidden states, leading to an unacceptable trade-off between efficency and response quality (Elhoushi et al., 2024; Liu et al., 2024a). To address this issue, in the following section, we introduce additional language heads and propose a customized fine-tuning scheme for them. These fine-tuned language head can better "understand" the hidden states of earlier layers, thereby mitigating the information gap.

5.1 Workflow of Dynamic Predictive Exiting

After obtaining the set of fine-tuned language head $\{\text{Head}_\ell | \ell \in \mathcal{L}\}$ at specified exit layers ℓ, LLMs can decode intermediate hidden states in real-time to generate predictive distributions in advance. At this point, our proposed **Dynamic Predictive Exiting** mechanism can be applied to the target model for inference acceleration. The detailed workflow of this mechanism involves two steps as follows:

1. Dynamic real-time decoding. When the forward computation of LLMs reaches the ℓ-th decoding layer, we decode the hidden states h_ℓ in real-time to obtain the immature predictive distribution P_ℓ, which reflects the prediction tendencies of the LLMs at the current layer.

2. Predictive exiting strategies. Each time we obtain the immature predictive distribution P_ℓ at an exiting layer ℓ, a criterion is needed to decide whether to accept it as the final prediction and terminate early. To this end, we propose the strategy for LLM early exit for CTR. This strategy are inspired by an interesting finding (Chuang et al., 2024) that LLMs progressively refine their hidden states across decoding layers. For some simple prediction steps, the hidden states at intermediate layers have already encoded sufficient information to predict the next token and remain relatively stable throughout the rest of the inference process.

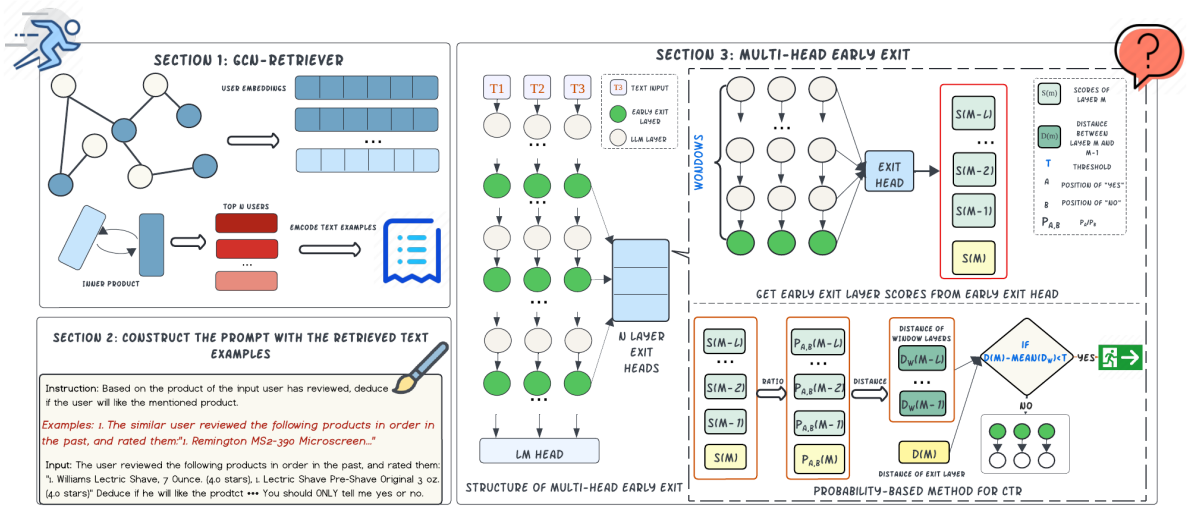

Figure 2: The process of OptiRAG-Rec: Section 1 GCN-Retriever: Constructs a GCN to generate user embeddings for identifying similar users; Section 2 Construct the Prompt with the Retrieved information: Forms LLM input by combining interaction data of the query user with similar users' data; Section 3 Multi-head Early Exit: Implements an early exit strategy in the LLM by designating potential exit layers and setting probability-based termination criteria.

The goal of these strategies is to avoid unnecessary computations on such steps, thereby reducing the time spent on "over-thinking". Below, we detail the Probability-Based Method for CTR.

Probability-Based Method for CTR. We track the stability of the model's binary predictions ("yes" vs. "no") across layers to determine early-exit readiness. Let $k \in \{\text{yes}, \text{no}\}$ be the binary decision token and $P_\ell(k)$ the predicted probability of token k at layer ℓ. The discrepancy in predictions between layer ℓ and a previous layer ℓ' is quantified by:

$$D(P_\ell, P_{\ell'}) = \left| P_\ell(k) - P_{\ell'}(k) \right|,$$

where $D_{\ell\ell'} \in \mathbb{R}$ measures the absolute difference in predicted probabilities. In particular, the distance between consecutive layers ℓ and $\ell - 1$ is represented as D_ℓ. A smaller D_ℓ indicates greater consistency in predictions across layers. To assess stability across multiple layers, we define the mean discrepancy over a window of m layers as:

$$\overline{D}_\ell^m = \frac{1}{m} \sum_{i=\ell}^{\ell+m-1} D_i,$$

If the absolute difference between the current discrepancy D_ℓ and the average discrepancy over the window is below a predefined threshold τ, i.e.,

$$\left| D_\ell - \overline{D}_{\ell-1}^m \right| < \tau,$$

we consider the output at layer ℓ to be stable and allow early termination of the forward pass. This logic is implemented in the multi-head early exit mechanism, as detailed in Algorithm 2 (Appendix A.2).

5.2 Optimization of Multi Heads

In prior to fine-tuning language models, we predefine exit layers $\mathcal{L} \subset \{1, \ldots, N-1\}$ and mount an additional language head Head_ℓ at each exit layer $\ell \in \mathcal{L}$, initialized using the LLM's existing head. These heads serve two key purposes: decoding intermediate hidden states and approximating subsequent layer inferences, enabling high-quality predictive distributions from exited states.

Our fine-tuning process begins with comprehensive instruction-tuning of the entire vanilla model to align it with the target application. After fine-tuning, we freeze the model's parameters and integrate multi-early-exit heads (i.e., additional language heads) at the designated layers. Our empirical results show that fine-tuning only these heads improves training stability and leads to a better convergence rate. Specifically, for a given textual input x and its ground truth token y, the layer-specific training loss for layer ℓ is defined as:

$$L_{\text{exit}}^{(\ell)} = -\log p^{(\ell)}(y|x),$$

where $p^{(\ell)}(y|x)$ is the probability that Head_ℓ assigns to the correct ground truth token y.

In training multi-head architectures, a depth-varying learning rate strategy is beneficial. Shal-

lower layers, capturing generic features, are assigned higher learning rates for aggressive updates, while deeper layers, adapting to specific features, benefit from finer updates. The learning rate for each head n, located at depth d_n in the architecture, is defined as follows:

$$\lambda_n = \lambda_0 \cdot e^{-\beta \cdot d_\ell},$$

where λ_0 is the base learning rate, β controls the decay rate with depth, and d_ℓ represents the head's depth, with shallower heads having a smaller d_ℓ.

Table 1: Dataset Statistics

Dataset	Users	Items	Samples
Beauty	324,037	32,892	6,525
BookCrossing	278,858	271,375	17,714
Video Games	37,890	17,381	221,465
Movies and TV	297,529	203,766	3,410,019
Yelp	228,195	146,927	2,956,589

6 Experiments

Dataset. We conduct experiments on three real-world datasets: BookCrossing(boo, 2005), Amazon Beauty, video games, Movies and TV(Ni et al., 2019) and Yelp. We present the processed dataset statistics in Figure 1. **BookCrossing:** The BookCrossing dataset comprises user ratings and detailed textual descriptions of books. **Amazon dataset:** The Amazon dataset comprises user purchase actions and rating information sourced from the Amazon platform. For our experiments, we selected two domains with a substantial number of overlapping users: **Beauty** 2018, **Video Games** 2018 and **Movies and TV** 2018. The Yelp dataset includes user-generated ratings, reviews, and check-ins for businesses such as restaurants, which are collected from Yelp.com. To prepare the dataset for the recommender system experiments, we initially processed the original data by organizing past interactions chronologically for each user. We then filtered out samples that had fewer than three past interactions to ensure sufficient data quality and reliability in the training set. For the construction of recommendation tuning samples, we retained up to 15 interactions that occurred prior to the target item. We further binarize the ratings according to a threshold of 3. The refined dataset was subsequently divided into training, validation, and testing sets, maintaining a ratio of 8:1:1.

Baseline Methods. Traditional CTR models are generally categorized into two types: feature interaction models and user behavior models. For our study, we selected DeepFM (Guo et al., 2017), AutoInt(Song et al., 2019), DCNv2(Wang et al., 2020), WuKong(Zhang et al., 2024b), GDCN(Wang et al., 2023b) and EulerNet(Tian et al., 2023) as representative feature interaction models. For user behavior models, we chose GRU4Rec (Hidasi et al., 2016) and DIN (Zhou et al., 2017). Additionally, we evaluated TALLRec (Bao et al., 2023b) as a representative LLM-based CTR model. To further benchmark the efficiency-accuracy trade-off in LLM-based recommendation, we included two lightweight and compressed LLMs as additional baselines: **SparseGPT** (Frantar and Alistarh, 2023b), a sparsity-aware pruned version of GPT for efficient inference, and **TinyLLaMA** (Kandala et al., 2024), a distilled and highly compact LLaMA variant designed for low-resource deployment. These models allow us to evaluate performance under extreme efficiency constraints.

We also implemented an **FM-based retriever**, a classical factorization machine model (Rendle, 2010) that scores user-item similarity based on learned embeddings. This provides a point of comparison to our GCN-retriever and highlights the benefits of graph-structured retrieval over traditional embedding similarity methods.

Implementation Details. For our experiments, we used Vicuna-7B, a model released by FastChat, as the base large language model. We employed few-shot training methods (randomly select less than 10% of the training data) to fine-tune the model for CTR task, while using the entire training dataset for training the traditional models. In OptiRAG-Rec, the number of examples was set to four. To reduce computational overhead in calculating exit scores, we selected layers (5,10,15, 20, 25, 30) as the early exit layers. For the multi-head early exit mechanism, we used a default window size of 3 and a threshold of 0.01. These values were selected for consistency across datasets in the main comparison. Additional ablation experiments with varying window sizes (3, 4, 5) and thresholds (0.005, 0.01, 0.05) are provided in Appendix A.4. For the GCN-retriever, we configured the model with 3 layers, an embedding dimension of 64, and trained it until the evaluation loss converged, ensuring optimal performance for recommendation. All methods used identical input features and preprocessing. Hyperparameter tuning followed consistent procedures across all baselines to ensure fair comparison. The superior LLM performance despite using only 10%

Table 2: Performance Comparison of Sequential Recommendation Models: Conventional Baselines, LLM Baseline, and Our Enhancements with GCN-Retriever and GCN-Early Exit.

	Model	BookCrossing		Beauty		Video Games		Movies and TV		Yelp	
		AUC	Log Loss	AUC	Log loss	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss
Full-shot	DeepFM	71.15	0.6897	66.67	0.539	57.68	0.5916	80.33	0.3884	83.15	0.4279
	DIN	71.17	0.6525	69.93	0.5727	62.96	0.7477	74.88	0.5210	71.59	0.7481
	GRU4Rec	60.75	2.1302	64.82	0.6195	58.26	0.9163	70.64	0.5830	63.15	0.9439
	AutoInt	58.00	0.6859	70.53	0.3899	63.59	0.8760	79.98	0.3902	83.02	0.4285
	DCNv2	58.52	0.6758	69.37	0.5254	67.89	0.5502	79.53	0.3968	83.21	0.4264
	WuKong	58.06	0.6798	65.64	0.3327	66.22	0.5926	77.36	0.4274	83.23	0.4261
	GDCN	58.03	0.6750	68.51	0.3587	66.27	0.7024	80.19	0.3922	83.08	0.4278
	EulerNet	57.86	0.7103	67.78	0.3212	64.14	0.8886	79.19	0.3931	82.97	0.4273
Few-shot	TALLRec	70.74	0.6306	90.37	0.2459	75.41	0.4754	71.14	0.4514	77.38	0.4792
	LLM-retriever	70.86	0.6907	89.65	0.2394	75.76	0.4805	73.89	0.7951	80.32	0.4577
	GCN-retriever	72.83	0.6158	94.72	0.2216	78.03	0.4850	90.34	0.4081	81.50	0.4692
	OptiRAG-Rec	**82.11**	**0.5269**	**96.37**	**0.2053**	**97.86**	**0.1911**	**98.46**	**0.3010**	**95.28**	**0.2460**

Table 3: AUC Scores and Log Loss by Dataset and Retrieval Layer.

Retriever	BookCrossing		Beauty		Video Games		Movies and TV		Yelp	
	AUC	Log Loss	AUC	Log Loss	AUC	LogLoss	AUC	Log Loss	AUC	Log Loss
Average Layer	72.83	**0.6158**	**94.72**	**0.2216**	**78.03**	**0.4850**	90.34	**0.3850**	**81.50**	0.4692
Last Layer	69.45	0.7151	93.55	0.3774	70.88	0.5461	**91.78**	0.4062	80.95	**0.4654**
Weighted Layer	**73.05**	0.6293	93.64	0.3012	72.48	0.6573	89.86	0.4442	81.38	0.4935

training data validates the effectiveness of our approach under practical computational constraints.

Measurement. Each configuration's performance was assessed using the area under the curve (AUC) and log loss for accuracy. The retrieval times, indicating computational demand, were normalized to the baseline (1x) set by the LLM retrievor. Inference speed was measured in terms of requests per second (RPS) per NVIDIA A100 GPU.

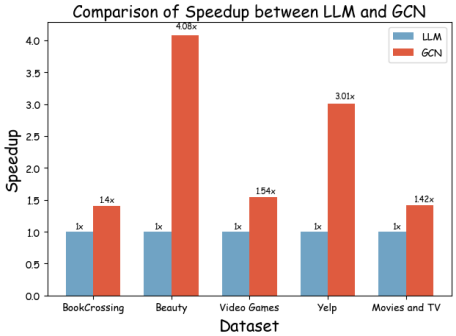

Figure 3: Speedup Comparison between LLM and GCN by Dataset.

Q1: Does OptiRAG-Rec Outperform traditional recommendation Models? Table 2 highlights key findings: OptiRAG-Rec significantly outperforms traditional methods, with AUC improvements of 16.14 for BookCrossing, 11.80 for Yelp, and 19.59 for Movies and TV(averaged across traditional baselines). This underscores the efficacy of our integrated framework combining GCN-based retrieval with LLM processing. Compared to traditional models, LLM-based methods excel in CTR prediction, especially for text-rich datasets like BookCrossing and Amazon reviews, which include item details such as titles, years, and prices. Unlike existing LLM-based approaches like TallRec, our method incorporates similar users' interactions, offering a more comprehensive view of user preferences and outperforming user-only LLM frameworks in CTR tasks.

Q2: What Are the Advantages of Multi-Head Early Exit Over Baselines and Efficiency Alternatives? Building on the overall framework success, we analyze the specific contribution of our novel multi-head early exit mechanism, a key component of OptiRAG-Rec's efficiency optimization. We evaluate the effectiveness of this approach by comparing it against (1) a baseline model without early exit, (2) a single-head early exit variant, and (3) other efficiency methods such as SparseGPT and TinyLLaMA. As shown in Table 4, the multi-head early exit model achieves consistently strong results across datasets. For example, on the Beauty dataset, it achieves an AUC of **96.37** with an RPS of 4.960, outperforming both the single-head variant (AUC 94.50, RPS 4.680) and the no-exit baseline.

Comparison with other efficiency methods (de-

Table 4: Ablation Study: Component Analysis.

Model Variant	BookCrossing		Beauty		Video Games		Movies and TV		Yelp	
	AUC	RPS	AUC	RPS	AUC	RPS	AUC	RPS	AUC	RPS
Baseline										
No Retriever, No Early Exit	70.74	**15.765**	90.37	<u>15.098</u>	75.41	**8.879**	71.14	**7.585**	77.38	**11.486**
Single Component										
Retriever Only	78.03	3.825	<u>94.72</u>	4.781	78.03	3.825	<u>90.34</u>	3.674	80.34	3.821
Early Exit Only	75.98	<u>11.313</u>	92.56	**20.318**	<u>96.37</u>	<u>7.917</u>	97.40	<u>7.034</u>	79.00	<u>8.216</u>
Combined Components										
Single Head Early Exit	<u>81.14</u>	7.191	94.50	4.680	96.13	4.814	**98.46**	4.977	**95.76**	4.065
Full Model (Both)	**82.11**	5.505	**96.37**	4.960	**97.86**	4.932	**98.46**	5.080	<u>95.28</u>	3.838

Note: Four samples were retrieved for each dataset in the retriever. Best values are in **bold**, second-best are <u>underlined</u>.

Table 5: Performance comparison between GCN-retriever and FM-retriever across datasets.

Model	BookCrossing		Beauty		Video Games		Movies and TV		Yelp	
	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss
GCN-retriever	**72.83**	**0.6158**	**94.72**	**0.2216**	**78.03**	**0.4850**	**90.34**	**0.4081**	**81.50**	**0.4692**
FM-retriever	69.33	0.6409	88.03	0.6263	67.82	0.7443	73.17	0.6260	74.01	0.5484

tailed results in Appendix A.3) reveals that our architectural approach outperforms traditional compression techniques. For instance, our method achieves 96.37 AUC vs 92.78 for SparseGPT on Beauty, while maintaining competitive throughput. Unlike quantization or pruning methods that reduce model capacity, our multi-head early exit preserves full precision while achieving efficiency through intelligent termination decisions.

We attribute the effectiveness of our design to two core factors:

- *Layer-wise specialization:* Each exit head is independently fine-tuned at different layers, enabling early predictions for simpler inputs while allowing deeper computation for more complex cases.
- *Robust confidence estimation:* A smoothed token-level scoring mechanism assesses prediction stability over time, supporting reliable and adaptive exit decisions.

The multi-head architecture provides superior reliability by aggregating confidence signals from multiple specialized perspectives, reducing the variance in early exit decisions that can occur with single-head approaches. These results validate that a multi-head architecture, when paired with a tailored exit criterion, offers a principled and effective approach to balancing computational efficiency and predictive performance in LLM-based recommendation systems.

Q3: Why is GCN-Based Retrieval Superior to LLM and Direct Similarity Methods? Our analysis (Figure 3 and Table 2) shows that GCN-retrievers achieve significantly faster retrieval speeds and higher accuracy than LLM-based retrievers. For instance, on the BookCrossing dataset, the GCN-retriever achieves an AUC of 82.11, outperforming the LLM retriever's 72.83. This performance gap reflects fundamental architectural differences between the two approaches:

- *Information Coverage:* LLM-based retrievers are constrained by limited context windows, incorporating only the 15 most recent interactions due to computational and memory limitations in our analysis. This truncation hinders the model's ability to model long-term user preferences. In contrast, GCN retrievers operate over the entire user-item interaction graph, leveraging multi-hop connectivity and graph structure to identify latent community-level patterns and indirect user-item relationships.
- *Computational Efficiency:* GCN retrievers use lightweight graph operations with linear complexity, while LLM retrievers require expensive transformer computations that scale quadratically with input length, adding overhead to similarity calculations.

To further isolate the contribution of graph topology, we compared our GCN-retriever with an FM-based retriever, which relies on direct user-item similarity scores without structural modeling. As

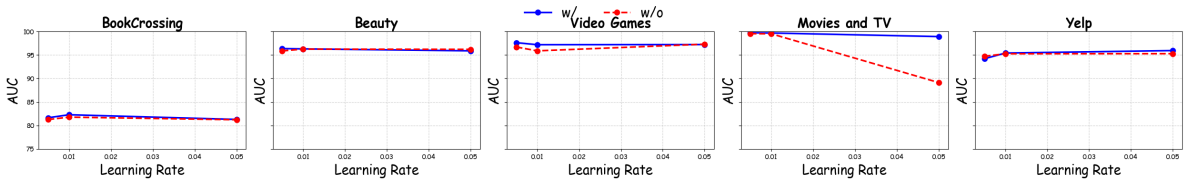

Figure 4: Comparison of model performance with (w/) and without (w/o) adaptive learning rate.

shown in Table 5, GCN-retriever consistently outperforms FM across all datasets, with AUC gains ranging from 2.68 to 17.17 points. These results confirm that the observed improvements are not merely due to representation learning, but stem from the ability of graph-based methods to capture complex, indirect user-item relationships via the interaction network.

Q4: How Do Different GCN Layer Aggregation Strategies Compare? Our analysis (Table 3) reveals that averaging embeddings from multiple GCN layers produces more robust representations compared to using either the final layer or weighted embeddings. This suggests that embeddings averaged across multiple layers provide a richer, more generalized representation that captures a broader spectrum of user-item interaction patterns.

Q5: What is the Impact of Combining Retrieval with Early Exit? We analyze the interaction effects between retrieval mechanisms and early exit strategies by examining four system configurations in Table 4: baseline (neither), retrieval only, early exit only, and the combined approach.

The results demonstrate clear synergistic benefits when combining both components. The combination substantially improves computational throughput across all datasets, with Video Games showing RPS increases from 3.825 (retrieval only) to 4.932 (combined), representing a 28.9% improvement. Similarly, Beauty demonstrates RPS improvement from 4.781 to 4.960, indicating that early exit effectively mitigates the computational overhead introduced by retrieval processing.

Notably, these efficiency gains do not come at the expense of accuracy. In fact, combining retrieval and early exit often enhances both metrics simultaneously. On Beauty, AUC increases from 94.72 (retrieval only) to 96.37 (combined), and on Movies and TV from 90.34 to 98.46. These improvements suggest that retrieval enhances input quality, enabling more confident and earlier predictions by the exit mechanism.

Importantly, the benefits of the combined system exceed the additive effects of the individual components. Retrieval improves representation quality, while early exit reduces inference cost. Their integration addresses complementary bottlenecks, resulting in a more efficient and accurate system than either component achieves in isolation.

Q6: How Do Key Hyperparameters Affect Performance? The Figure 4 show that the adaptive learning rate (blue line) does not significantly outperform the non-adaptive learning rate (red line), as the AUC scores remain closely aligned across all datasets. This suggests that the adaptive learning rate may not substantially enhance performance for the tested data and model configurations. However, the consistent performance indicates the model's robustness to learning rate variations, which could be advantageous for maintaining stability across different operational conditions.

7 Conclusion

As LLMs continue to advance, their potential to transform recommendation systems is becoming increasingly evident (Bao et al., 2023b). In this work, we introduced a comprehensive framework that integrates advanced retrieval mechanisms and early exit strategies to enhance both the efficiency and accuracy of LLM-based recommendations. By incorporating Graph GCNs as efficient retrieval mechanisms and implementing a multi-head early exit architecture, our framework significantly reduces computation time while maintaining or even improving system accuracy. This holistic approach not only accelerates the responsiveness of LLMs but also preserves their decision-making quality, making it highly suitable for real-time applications in commercial systems. Our results demonstrate the effectiveness of this framework in balancing performance and efficiency, setting a new standard for deploying LLMs in recommendation tasks.

8 Limitation

Our framework has several limitations requiring future consideration. The approach requires sub-

stantial textual metadata, limiting applicability to sparse-text domains. The GCN-retriever struggles with cold-start scenarios lacking sufficient interaction history. Despite efficiency improvements, initial training requires significant computational resources. Domain transferability beyond e-commerce and reviews needs additional validation. The framework lacks real-time graph update mechanisms for continuously evolving interactions.

References

2005. *WWW '05: Proceedings of the 14th international conference on World Wide Web*. Association for Computing Machinery, New York, NY, USA.

Arkadeep Acharya, Brijraj Singh, and Naoyuki Onoe. 2023. Llm based generation of item-description for recommendation system. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1204–1207.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. 2023. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5910–5924, Singapore. Association for Computational Linguistics.

H. Bai, L. Hou, L. Shang, X. Jiang, I. King, and M. R. Lyu. 2022. Towards efficient post-training quantization of pre-trained language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 1405–1418.

Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023a. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. *arXiv preprint arXiv:2305.00447*.

Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023b. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, volume 33 of *RecSys '23*, page 1007–1014. ACM.

Shuqing Bian, Wayne Xin Zhao, Jinpeng Wang, and Ji-Rong Wen. 2022. A relevant and diverse retrieval-enhanced data augmentation framework for sequential recommendation.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *Preprint*, arXiv:2401.10774.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. *Preprint*, arXiv:2312.04916.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James Glass, and Pengcheng He. 2024. Dola: Decoding by contrasting layers improves factuality in large language models. *Preprint*, arXiv:2309.03883.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin. 2020. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*.

Elias Frantar and Dan Alistarh. 2023a. Sparsegpt: Massive language models can be accurately pruned in one-shot. *Preprint*, arXiv:2301.00774.

Elias Frantar and Dan Alistarh. 2023b. Sparsegpt: Massive language models can be accurately pruned in one-shot. *Preprint*, arXiv:2301.00774.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: A factorization-machine based neural network for ctr prediction. *Preprint*, arXiv:1703.04247.

MohammadTaghi Hajiaghayi, Sébastien Lahaie, Keivan Rezaei, and Suho Shin. 2024. Ad auctions for llms via retrieval augmented generation. *Preprint*, arXiv:2406.09459.

Jesse Harte, Wouter Zorgdrager, Panos Louridas, Asterios Katsifodimos, Dietmar Jannach, and Marios Fragkoulis. 2023. Leveraging large language models for sequential recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1096–1102.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. *Preprint*, arXiv:1511.06939.

Renren Jin, Jiangcun Du, Wuwei Huang, Wei Liu, Jian Luan, Bin Wang, and Deyi Xiong. 2024. A comprehensive evaluation of quantization strategies for large language models. *Preprint*, arXiv:2402.16775.

Savitha Viswanadh Kandala, Pramuka Medaranga, and Ambuj Varshney. 2024. Tinyllm: A framework for training and deploying language models at the edge computers. *Preprint*, arXiv:2412.15304.

Wei-Tsung Kao, Tsung-Han Wu, Po-Han Chi, Chun-Cheng Hsieh, and Hung-Yi Lee. 2020. Bert's output layer recognizes all hidden layers? some intriguing phenomena and a simple way to boost bert. *arXiv preprint arXiv:2001.09309*.

Sein Kim, Hongseok Kang, Seungyoon Choi, Donghyun Kim, Minchul Yang, and Chanyoung Park. 2024. Large language models meet collaborative filtering: An efficient all-round llm-based recommender system. KDD '24, page 1395–1406, New York, NY, USA. Association for Computing Machinery.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Lei Li, Yongfeng Zhang, and Li Chen. 2023. Prompt distillation for efficient llm-based recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 1348–1357.

X. Li, Y. Shao, T. Sun, H. Yan, X. Qiu, and X. Huang. 2021. Accelerating bert inference for sequence labeling via early-exit. *arXiv preprint arXiv:2105.13878*.

Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Less is more: Task-aware layer-wise distillation for language model compression. In *International Conference on Machine Learning*, pages 20852–20867. PMLR.

J. Lin, R. Shan, C. Zhu, K. Du, B. Chen, S. Quan, R. Tang, Y. Yu, and W. Zhang. 2024a. Rella: Retrieval-enhanced large language models for lifelong sequential behavior comprehension in recommendation. In *Proceedings of the ACM on Web Conference 2024*, pages 3497–3508.

Jianghao Lin, Rong Shan, Chenxu Zhu, Kounianhua Du, Bo Chen, Shigang Quan, Ruiming Tang, Yong Yu, and Weinan Zhang. 2024b. Rella: Retrieval-enhanced large language models for lifelong sequential behavior comprehension in recommendation. In *Proceedings of the ACM Web Conference 2024*, volume 11 of *WWW '24*, page 3497–3508. ACM.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024a. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*.

Jiajun Liu, Yibing Wang, Hanghang Ma, Xiaoping Wu, Xiaoqi Ma, Xiaoming Wei, Jianbin Jiao, Enhua Wu, and Jie Hu. 2024b. Kangaroo: A powerful video-language model supporting long-context video input. *Preprint*, arXiv:2408.15542.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. 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)*, pages 188–197, Hong Kong, China. Association for Computational Linguistics.

Steffen Rendle. 2010. Factorization machines. *2010 IEEE International Conference on Data Mining*, pages 995–1000.

Gaurav Sahu, Olga Vechtomova, Dzmitry Bahdanau, and Issam H. Laradji. 2023. Promptmix: A class boundary augmentation method for large language model distillation. *Preprint*, arXiv:2310.14192.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19. ACM.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

C. Tao, L. Hou, W. Zhang, L. Shang, X. Jiang, Q. Liu, and N. Wong. 2022. Compression of generative pre-trained language models via quantization. *arXiv preprint arXiv:2203.10705*.

Zhen Tian, Ting Bai, Wayne Xin Zhao, Ji-Rong Wen, and Zhao Cao. 2023. Eulernet: Adaptive feature interaction learning via euler's formula for ctr prediction. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 1376–1385, New York, NY, USA. Association for Computing Machinery.

Dong Wang, Kavé Salamatian, Yunqing Xia, Weiwei Deng, and Qi Zhang. 2023a. Bert4ctr: An efficient framework to combine pre-trained language model with non-textual features for ctr prediction. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 5039–5050, New York, NY, USA. Association for Computing Machinery.

Fangye Wang, Hansu Gu, Dongsheng Li, Tun Lu, Peng Zhang, and Ning Gu. 2023b. Towards deeper, lighter and interpretable cross network for ctr prediction. *Preprint*, arXiv:2311.04635.

Lei Wang and Ee-Peng Lim. 2023. Zero-shot next-item recommendation using large pretrained language models. *arXiv preprint arXiv:2304.03153*.

Ruoxi Wang, Rakesh Shivanna, Derek Z. Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. 2020. Dcn v2: Improved deep cross network and practical lessons for web-scale learning to rank systems. *Preprint*, arXiv:2008.13535.

Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, Hui Xiong, and Enhong Chen. 2023. A survey on large language models for recommendation. *Preprint*, arXiv:2305.19860.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *Preprint*, arXiv:2401.07851.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

J. Xin, R. Tang, Y. Yu, and J. Lin. 2021. Berxit: Early exiting for bert with better fine-tuning and extension to regression. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume*, pages 91–104.

An Zhang, Yuxin Chen, Leheng Sheng, Xiang Wang, and Tat-Seng Chua. 2024a. On generative agents in recommendation. *Preprint*, arXiv:2310.10108.

Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Daifeng Guo, Yanli Zhao, Shen Li, Yuchen Hao, Yantao Yao, Guna Lakshminarayanan, Ellie Dingqiao Wen, Jongsoo Park, Maxim Naumov, and Wenlin Chen. 2024b. Wukong: Towards a scaling law for large-scale recommendation. *Preprint*, arXiv:2403.02545.

Yuyue Zhao, Jiancan Wu, Xiang Wang, Wei Tang, Dingxian Wang, and Maarten de Rijke. 2024. Let me do it for you: Towards llm empowered recommendation via tool learning. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1796–1806.

Zhi Zheng, Wenshuo Chao, Zhaopeng Qiu, Hengshu Zhu, and Hui Xiong. 2024. Harnessing large language models for text-rich sequential recommendation. In *Proceedings of the ACM on Web Conference 2024*, pages 3207–3216.

Zhi Zheng, Zhaopeng Qiu, Xiao Hu, Likang Wu, Hengshu Zhu, and Hui Xiong. 2023. Generative job recommendations with large language model. *arXiv preprint arXiv:2307.02157*.

Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2017. Deep interest network for click-through rate prediction. *Preprint*, arXiv:1706.06978.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341.

Yaochen Zhu, Liang Wu, Qi Guo, Liangjie Hong, and Jundong Li. 2024. Collaborative large language model for recommender systems. In *Proceedings of the ACM Web Conference 2024*, volume 20 of *WWW '24*, page 3162–3172. ACM.

A Appendix

A.1 Prompt Templates and Examples

Our framework uses dataset-specific prompt templates that integrate similar users' interactions with target user histories. Tables 6, 7, and 8 show representative examples from each dataset type. Note that Amazon datasets (Beauty, Video Games, Movies and TV) share the same prompt structure, differing only in product-specific details, so we present only the Beauty example as representative of all Amazon domains.

A.1.1 Prompt Engineering Considerations

Retrieval Integration: We retrieve 4 similar users to provide relevant context while maintaining manageable prompt length.

History Limitation: Target user history is limited to 15 most recent interactions to stay within LLM context windows while preserving recent preference signals.

Output Constraint: We restrict LLM output to binary tokens ("yes"/"no") and extract prediction probabilities from logit scores, enabling efficient CTR prediction without full text generation.

Instruction: Based on the product of the input user has reviewed, deduce if the user will like the mentioned product. Note that more stars the user rated the product, the user liked the product more. You should ONLY tell me yes or no.

Similar Users' Examples:
1. The similar user reviewed the following products and rated them: ['Mederma Scar Gel, 20 Grams. (5.0 stars)', 'Avalon Organics Wrinkle Therapy CoQ10 Cleansing Milk, 8.50 oz. It's price is $8.27 (5.0 stars)', ...]
2. The similar user reviewed the following products and rated them: ['Caswell-Massey - Newport Soap on a Rope. (5.0 stars)', 'Deep Steep 3 Piece Gift Set, Grapefruit Bergamot. (5.0 stars)', ...]
3. The similar user reviewed the following products and rated them: ['Dinur Cosmetics Bio Clean Drying Lotion 0.67 oz. (5.0 stars)', 'Avalon Organics Vitamin C Renewal Creme, 2 oz. (5.0 stars)', ...]
4. The similar user reviewed the following products and rated them: ['Plant Therapy Tea Tree Organic Essential Oil (5.0 stars)', 'Avalon Organics Vitamin C Renewal Moisture Plus Lotion SPF 15 (1.0 star)', ...]

Target User Input: The user reviewed the following products in order in the past, and rated them: ['Avalon Organics Wrinkle Therapy CoQ10 Cleansing Milk, 8.50 oz. (5.0 stars)', 'Organic Fiji Raw Organic Coconut Oil, 13-Ounce Jars (5.0 stars)', ...]

Question: Deduce if he will like the product ***(3 Pack) KLEANCOLOR Retractable Waterproof Lip & Eye Liner - Fuschia***. You should ONLY tell me yes or no.

Table 6: Amazon Beauty dataset prompt example.

Instruction: Deduce if the user will like the candidate book based on the user's past history. You can refer to the books rating history of other users. Note that more stars the user rated the book, the user liked the book more. You should ONLY tell me yes or no.

Similar Users' Reading History:
1. The user's location is USA. The user's age is unknown. The user read the following books and rated them: ['Face the Fire (Three Sisters Island Trilogy) (6 stars)', 'The Sigma Protocol (9 stars)', ...]
2. The user's location is Netherlands. The user's age is 35-39. The user read the following books and rated them: ['The Home DIY Expert (7 stars)', 'Firefly Summer (7 stars)', 'The Girls' Guide to Hunting and Fishing (9 stars)', ...]
3. The user's location is USA. The user's age is unknown. The user read the following books and rated them: ['Daughter of Fortune (8 stars)', 'Ship of Gold in the Deep Blue Sea (5 stars)', 'Thin Air (5 stars)', ...]
4. The user's location is USA. The user's age is 25-29. The user read the following books and rated them:['Chasing the Dime (9 stars)', 'Border Bride (10 stars)', 'Murder at the Library of Congress (8 stars)', ...]

Target User Input: The user's location is USA. The user's age is 45-49. The user read the following books and rated them: ['Dress Codes: Of Three Girlhoods (8 stars)', 'What Would Buddha Do? (5 stars)', 'Are You Somebody? (8 stars)', ...]

Question: Based on the books the user has read, deduce if the user will like the book ***Big Fish***. You should ONLY tell me yes or no.

Table 7: BookCrossing dataset prompt example.

Algorithm 1 GCN-Based User Retriever

Require: $\mathcal{H}$ (user histories), METHOD $\in$ {'MEAN', 'FINAL', 'WEIGHT'}, TOP_K
$\mathcal{G} \leftarrow \text{GCN}(\mathcal{H})$
$\mathbf{H}_u^{(l)} \leftarrow \mathcal{G}$.get_embeddings() for all users $u \in U$ and layers $l = 1, \ldots, L$
// Aggregate layer-wise embeddings
if METHOD == 'MEAN' **then**
  $\mathbf{H}_u \leftarrow \frac{1}{L} \sum_{l=1}^{L} \mathbf{h}_u^{(l)}$
else if METHOD == 'FINAL' **then**
  $\mathbf{H}_u \leftarrow \mathbf{h}_u^{(L)}$
else
  $\mathbf{H}_u \leftarrow \sum_{l=1}^{L} \alpha_l \mathbf{h}_u^{(l)}$ {α_l are layer weights}
end if
// Normalize user embeddings
$\mathbf{H}_u \leftarrow \frac{\mathbf{H}_u}{\|\mathbf{H}_u\|_2}$ for all $u \in U$
// Generate prompts using top-k similar users
prompts $\leftarrow$ [ ]
for each user $u \in U$ **do**
  $\mathbf{s}_u \leftarrow \mathbf{H}_u \cdot \mathbf{H}_U^\top$
  $\mathbf{s}_u[u] \leftarrow -\infty$ {Exclude self-matching}
  similar_users $\leftarrow$ ArgSort($\mathbf{s}_u$)[: TOP_K]
  prompt $\leftarrow$ Format(u, similar_users, $\mathcal{H}$)
  prompts.append(prompt)
end for
return prompts

Algorithm 2 Multihead Early-Exit

Require: $\mathcal{L}$: Exit layers; $h^{(\ell)}$: hidden states; Head$_\ell$: classifier head at layer ℓ;
  m: window size (≥ 3); τ: threshold for exit
Ensure: `True` if early exit condition is met, else `False`
Initialize multiheads: {Head$_\ell \leftarrow$ Copy(LM Head) | $\ell \in \mathcal{L}$}
for each $\ell \in \mathcal{L}$ **do**
  Select $m{+}1$ most recent hidden states: $\mathcal{H}_\ell \leftarrow \{h^{(\ell-m)}, ..., h^{(\ell)}\}$
  Initialize empty ratio list $\mathcal{R}$
  for each $h \in \mathcal{H}_\ell$ **do**
    $\mathbf{p} \leftarrow \text{Softmax}(\text{Head}_\ell(h))$
    Compute ratio: $r \leftarrow \frac{p_{\text{pos}}}{p_{\text{neg}}}$
    Append: $\mathcal{R}$.append(r)
  end for
  Compute deltas: $\delta_i = |\mathcal{R}[i] - \mathcal{R}[i+1]|$ for $i = 0$ to $m{-}1$
  $\overline{D}_\ell^m \leftarrow \frac{1}{m-1} \sum_{i=0}^{m-2} \delta_i$
  $D_\ell \leftarrow |\mathcal{R}[-1] - \mathcal{R}[-2]|$
  if $|D_\ell - \overline{D}_\ell^m| < \tau$ **then**
    return `True`
  end if
end for
return `False`

Instruction: Based on the Yelp store the user has visited, determine if the user will like the mentioned store. Note: The more stars the user rated the store, the more they liked it.You should ONLY tell me yes or no.

Similar Users' Store Visits:
1. The similar user joined Yelp in 2016-07-20. They have 0 fans and typically rate stores with 3.82 stars average. The user recently visited the following stores and rated them: ['Matt & Marie's, Philadelphia PA (1.0 star)', 'The Caketeria, Newtown PA (5.0 stars)', 'Kensington Quarters, Philadelphia PA (5.0 stars)', ...]
2. The similar user joined in 2011-02-21. They have 27 fans and rate at 3.86 stars average. The user recently visited the following stores and rated them: ['Brick House Tavern + Tap, Tampa FL (3.0 stars)', 'Brio Italian Grille, Tampa FL (5.0 stars)', 'Kona Grill - Tampa, Tampa FL (5.0 stars)', ...]
3. The similar user joined in 2014-06-04. They have 15 fans and rate at 3.97 stars average. The user recently visited the following stores and rated them: ['Mini Doughnut Factory, Tampa FL (5.0 stars)', 'Green Lemon, Tampa FL (5.0 stars)', 'Catch Twenty Three, Tampa FL (5.0 stars)', ...]
4. The similar user joined in 2014-09-27. They have 21 fans and rate at 4.42 stars average. The user recently visited the following stores and rated them: ['The Red Lion Pub, Indian Rocks Beach FL (4.0 stars)', 'Hai Street Kitchen & Co, Philadelphia PA (5.0 stars)', 'The Mütter Museum, Philadelphia PA (4.0 stars)', ...]

Target User Input: The user has been on Yelp since 2016-07-20. They have 0 fans and rate stores at 3.82 stars average. The user recently visited the following stores and rated them:['Cross Culture, Doylestown PA (3.0 stars)', 'Dan Dan, Philadelphia PA (5.0 stars)', 'Giwa Korean Kitchen, Philadelphia PA (5.0 stars)', ...]

Task: Deduce if the user will like the store ***Jamba located in Willow Grove PA***. Remember: You should ONLY respond with yes or no.

Table 8: Yelp dataset prompt example.

A.2 Algorithmic Details

Algorithm 2 and Algorithm 1 provide detailed pseudocode for our core components.

A.3 Comprehensive Efficiency Analysis

To address concerns about efficiency trade-offs, we conducted comprehensive experiments comparing OptiRAG-Rec with prominent alternative efficiency techniques. Table 9 presents detailed results across all datasets.

A.3.1 Experimental Setup

We evaluated two efficiency approaches:

SparseGPT + GCN-Retriever: We applied structured pruning via SparseGPT (Frantar and Alistarh, 2023b) to our backbone LLM while maintaining the GCN-Retriever architecture. Specifically, we introduced a sparsity level of 0.5 for the projection layers ('q_proj', 'k_proj', 'v_proj', 'o_proj') in each transformer layer, targeting the most computationally intensive components.

TinyLlama + Fine-tuned GCN-Retriever: We replaced the backbone LLM with TinyLlama (Kandala et al., 2024) while retaining our GCN-Retriever component, representing the model compression approach to efficiency optimization.

A.3.2 Key Findings and Analysis

Pruning Trade-offs: As shown in Tables 9 and 5, SparseGPT achieves higher throughput (up to 7.99 RPS vs 4.96 on Beauty) but suffers significant accuracy degradation. BookCrossing shows a 15.27-point AUC decrease (82.11 vs 66.84), demonstrating that static pruning cannot preserve the quality-efficiency balance of dynamic computation.

Model Compression Limitations: TinyLlama achieves the highest throughput (up to 17.60 RPS) but experiences catastrophic accuracy loss of 30-45% compared to OptiRAG-Rec. This reveals fundamental limitations of capacity reduction for recommendation requiring sophisticated reasoning.

Dynamic Efficiency Advantage: OptiRAG-Rec simultaneously enhances accuracy (9.28-point average improvement over baseline) while maintaining competitive throughput (3.84-5.51 RPS). Unlike static methods with fixed trade-offs, our dynamic approach adapts computational allocation based on sample complexity.

These results validate our design philosophy: architectural innovations in dynamic computation allocation provide superior efficiency gains compared to traditional model compression approaches. OptiRAG-Rec offers the optimal balance for production recommendation systems where both accuracy and latency are critical constraints, while alternative methods force suboptimal trade-offs between these competing objectives.

A.4 Hyperparameter Sensitivity Analysis

Figure 6 presents comprehensive analysis of key hyperparameters affecting system performance.

Retrieval Examples Impact: Figure 7 shows the trade-off between retrieval quantity and performance. AUC generally improves with more examples up to 4-5, after which diminishing returns occur. This indicates an optimal balance between context richness and computational overhead, with

4 examples providing the best accuracy-efficiency trade-off across datasets.

Early Exit Configuration: Figures 8 and 9 show the effects of varying early exit thresholds and window sizes. Results show that using a threshold of 0.01 with a window size of 3 consistently achieves the best trade-off between accuracy and efficiency (e.g., 96.37 AUC on BookCrossing, 97.86 on Video Games). Higher thresholds (0.05, 0.1) lead to premature exits with poor accuracy, while larger windows reduce throughput without clear gains. We adopt threshold=0.01 and window=3 as our default setting.

Table 9: Efficiency Method Comparison: Accuracy and Quality Metrics

Method	BookCrossing		Beauty		Video Games		Movies and TV		Yelp	
	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss
GCN-retriever	72.83	0.6158	94.72	0.2216	78.03	0.4850	90.34	0.4081	81.50	0.4692
OptiRAG-Rec	**82.11**	**0.5269**	**96.37**	**0.2053**	**97.86**	**0.1911**	**98.46**	**0.1911**	**95.28**	**0.2460**
SparseGPT	66.84	0.6632	92.78	0.2631	69.59	0.6348	90.61	0.3854	77.48	0.3854
TinyLlama	52.81	0.8754	66.61	0.3699	54.71	0.5534	78.86	0.4458	50.70	0.4456

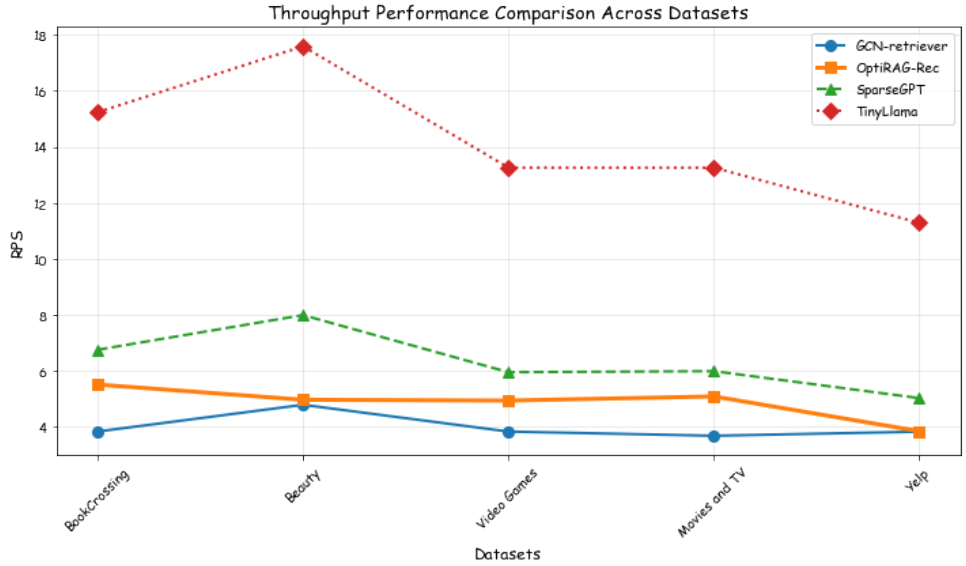

Figure 5: Throughput Performance Comparison (RPS) Across Datasets

Figure 6: Hyperparameter Sensitivity Analysis across Different Configurations

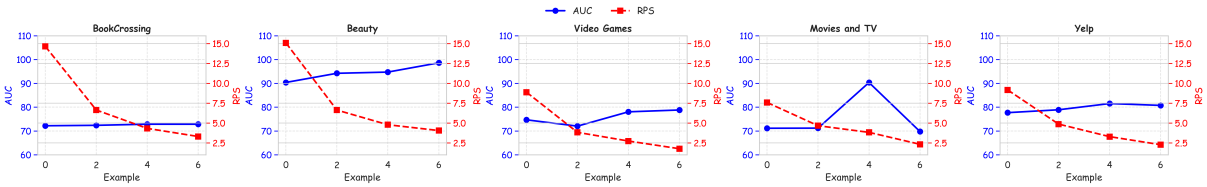

Figure 7: AUC and RPS by retrieval examples.

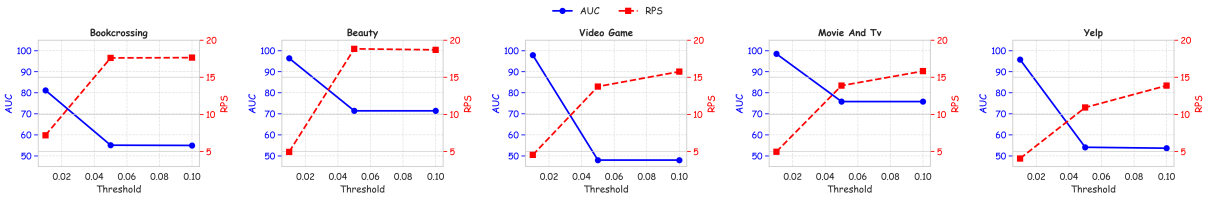

Figure 8: AUC and RPS by threshold.

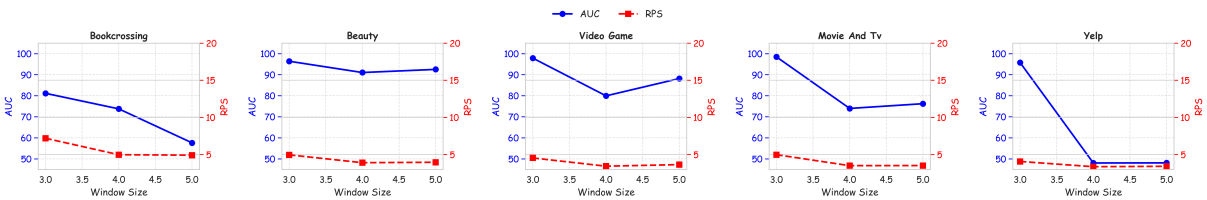

Figure 9: AUC and RPS by windows.