Improving Sequential Recommendations with LLMs

ARTUN BOZ*, Individual contributor, Delft, Netherlands
WOUTER ZORGDRAGER*, Individual contributor, Delft, Netherlands
ZOE KOTTI*, Athens University of Economics and Business, Athens, Greece
JESSE HARTE*, Delft University of Technology, Delft, Netherlands
PANOS LOURIDAS, Athens University of Economics and Business, Athens, Greece
VASSILIOS KARAKOIDAS, Delivery Hero SE, Berlin, Germany
DIETMAR JANNACH, University of Klagenfurt, Klagenfurt, Austria
MARIOS FRAGKOULIS*, Delft University of Technology, Delft, Netherlands

The sequential recommendation problem has attracted considerable research attention in the past few years, leading to the rise of numerous recommendation models. In this work, we explore how Large Language Models (LLMs), which are nowadays introducing disruptive effects in many AI-based applications, can be used to build or improve sequential recommendation approaches. Specifically, we design three orthogonal approaches and hybrids of those to leverage the power of LLMs in different ways. In addition, we investigate the potential of each approach by focusing on its comprising technical aspects and determining an array of alternative choices for each one. We conduct extensive experiments on three datasets and explore a large variety of configurations, including different language models and baseline recommendation models, to obtain a comprehensive picture of the performance of each approach.

Among other observations, we highlight that initializing state-of-the-art sequential recommendation models such as BERT4Rec or SASRec with embeddings obtained from an LLM can lead to substantial performance gains in terms of accuracy. Furthermore, we find that fine-tuning an LLM for recommendation tasks enables it to learn not only the tasks, but also concepts of a domain to some extent. We also show that fine-tuning OpenAI GPT leads to considerably better performance than fine-tuning Google PaLM 2. Overall, our extensive experiments indicate a huge potential value of leveraging LLMs in future recommendation approaches. We publicly share the code and data of our experiments to ensure reproducibility.[1]

CCS Concepts: • **Information systems → Recommender systems**.

Additional Key Words and Phrases: Recommender Systems, Large Language Models, Sequential Recommendation, Evaluation

1 Introduction

The sequential recommendation problem considers a sequence of past user interactions in order to predict the next user interest or action. The problem's wide application in many popular domains, such as next-purchase

*Part of the author's work was carried out while at Delivery Hero Research.
[1]https://github.com/dh-r/LLM-Sequential-Recommendation

Authors' Contact Information: Artun Boz, Individual contributor, Delft, Netherlands; e-mail: artun.boz@deliveryhero.com; Wouter Zorgdrager, Individual contributor, Delft, Netherlands; e-mail: wouter.zorgdrager@deliveryhero.com; Zoe Kotti, Athens University of Economics and Business, Athens, Attica, Greece; e-mail: zoe.kotti@deliveryhero.com; Jesse Harte, Delft University of Technology, Delft, Zuid-Holland, Netherlands; e-mail: jesse.harte@deliveryhero.com; Panos Louridas, Athens University of Economics and Business, Athens, Attica, Greece; e-mail: louridas@aueb.gr; Vassilios Karakoidas, Delivery Hero SE, Berlin, Germany; e-mail: bkarak@deliveryhero.com; Dietmar Jannach, University of Klagenfurt, Klagenfurt, Austria; e-mail: dietmar.jannach@aau.at; Marios Fragkoulis, Delft University of Technology, Delft, Zuid-Holland, Netherlands; e-mail: mfrank.23@gmail.com.

ACM 2770-6699/2025/1-ART
https://doi.org/10.1145/3711667

prediction [28], next-track music recommendation [9], and next Point-of-Interest suggestions for tourism [47], has sparked substantial research interest in the recent years. This had led to the inception of novel algorithmic approaches [23, 34, 53, 67], including approaches that utilize side information about the items, such as an item's category [39, 79].

Lately, the advent of Large Language Models (LLM), which in an *auto-regressive mode* repeatedly complete a sequence until they fulfil a prompt task [61], has opened a fruitful research direction for sequential recommendation. In fact, LLMs based on Generative Pretrained Transformers [62] are only the latest of a stream of innovations in Natural Language Processing (NLP) that inspired the birth of new sequential recommendation models. Notable examples are GRU4Rec [23], SASRec [34], and BERT4Rec [67], which were influenced by the Gated Recurrent Unit (GRU) [10], the transformer architecture [70], and BERT [13], respectively. The key contribution of LLMs to the sequential recommendation problem is a semantic model of real world concepts and their representation in natural language sequences. This consideration is neglected by typical ID-based sequential recommendation models [34, 67].

The rapidly growing interest for involving LLMs in recommender systems clusters around cross-domain [26, 85], domain-specific [4, 86], and multi-modal [72, 77] recommendations. Within the scope of recommendation approaches, LLMs are involved either by providing item embeddings [24] or by responding to a recommendation task conveyed in a prompt, which may contain zero [14] or a few examples [19] to enrich the context of the task available to an LLM. Alternatively, an LLM can be fine-tuned to learn how to perform better for a specific recommendation task [25]. Regarding the nature of recommendation tasks posed to LLMs, they are typically used to generate [42] or rerank [26] recommendations. Altogether, the current state of research is summarized in the following surveys [48, 78].

In this paper, we build on the ideas developed in these early works on involving LLMs in recommender systems, and we design and research the potential of three orthogonal approaches and hybrids of those that leverage LLMs for the particular problem setting of sequential recommendations. We decompose each approach into its constituent technical aspects, identify or devise alternatives for each aspect, and investigate the impact of choices to the approach's performance. Finally, we thoroughly tune and evaluate all approaches under the same testbed across three datasets of different domains and characteristics. These datasets are Amazon Beauty from the e-commerce beauty domain, a proprietary Delivery Hero dataset from the e-commerce darkstore domain, and Steam from the gaming domain.

In the first approach (LLMSeqSim), we retrieve a semantically-rich embedding from an existing LLM for each item in a session, optionally reduce the embedding to a target number of dimensions, and then compute an aggregate session embedding. We use the latter to recommend catalog products with a similar embedding. In this first approach we investigate the following four technical aspects: the choice of LLM to retrieve embeddings from, dimensionality reduction methods, target embedding dimensions, and session embedding computation strategies. In the second approach (LLMSeqPrompt), we fine-tune an LLM with dataset-specific information in the form of prompt-completion pairs and ask the model to produce next item recommendations for a set of test prompts. We explore four different aspects of LLMs, namely models, versions, parameters, and task specifications. Our third approach (LLM2Sequential) consists of enhancing existing sequential models with item embeddings obtained from an LLM. Here, we examine LLM embedding models, dimensionality reduction methods, and sequential models. Finally, we create hybrid recommendation approaches based on the aforementioned ones.

Our work results in the following contributions and insights.[2]

[2]The present paper substantially extends our previous work on leveraging LLMs for the sequential recommendation problem [20]. We detail our introduced extensions in Section 2.

- Building on the growing literature regarding the use of LLMs in recommender systems, we design and research three orthogonal methods and two hybrid approaches of leveraging LLMs for sequential recommendation. In particular, we investigate the technical aspects of each method, determine alternatives for each, thoroughly tune them, and evaluate their impact.
- Experiments on three datasets, including a proprietary real-world dataset from Delivery Hero, reveal that LLM embeddings boost the performance of different classes of sequential models across all datasets. Specifically, LLM2SASRec and LLM2BERT4Rec, which are implementations of the LLM2Sequential approach, top the leaderboard increasing NDCG@20 on average by 45% on the Amazon Beauty dataset and 9% on the Delivery Hero dataset. Beyond accuracy metrics, LLM2SASRec almost doubles catalog coverage and substantially increases serendipity by 21% over SASRec across all datasets.
- Both OpenAI GPT and Google PaLM fine-tuned models (LLMSeqPrompt) perform significantly better than their counterpart base models across tasks marking 16% average improvement in terms of NDCG@20. Most importantly, the fine-tuned models exhibit as low as half the number of hallucinations and noticeably higher semantic similarity to actual recommendation options compared to the base models indicating that fine-tuned models can learn not only tasks, but also concepts of a domain to some extent. Finally, `GPT 3.5 turbo` considerably outperforms `PaLM 2 bison` by more than 78% on average in terms of NDCG@20 across fine-tuning tasks and datasets.
- In one of the datasets, Amazon Beauty, the semantic item recommendation model via LLM embeddings (LLMSeqSim) takes the third place and achieves the best MRR score, while a fine-tuned model (LLMSeqPrompt) outperforms GRU4Rec and SASRec. Both results support the potential of LLM-based models for datasets with specific characteristics.

Overall, the main contribution of our work is an in-depth analysis of the potential of various previously-explored and novel ways of leveraging LLMs for sequential recommendation problems. Our comprehensive analyses, which involve different datasets, three LLMs, and a set of orthogonal technical approaches reveal that substantial performance improvements can indeed be obtained by involving LLMs for sequential recommendations.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Next, Sections 3, 4, 5, and 6 present the researched recommendation approaches. Then, Section 7 details the experiments and, finally, Section 8 conveys the conclusions of our work.

2 Background & Related Work

The recent developments in LLMs have taken the world by surprise. Models like OpenAI GPT [7], Google BERT [13], and Meta LLaMA [68], which employ deep transformer architectures, demonstrate how innovations in NLP can reshape and advance mainstream online activities, such as search, shopping, and customer care. Inevitably, research in recommender systems is significantly impacted by the developments in the area of LLMs as well.

According to recent surveys [48, 78], LLMs are mainly utilized for recommendation problems in two ways: by providing embeddings that can be used to initialize existing recommendation models [52, 76, 84], and by producing recommendations leveraging their inherent knowledge encoding [4, 22, 35]. LLMs as recommendation models can provide recommendations given (i) only a task specification (zero-shot), (ii) one or few examples given inline to the prompt of the task (few-shot), or (iii) after fine-tuning the model's weights to the task at hand given a set of training examples [7]. This incremental training process deviates from typical recommendation models, which have to be trained from zero on domain data. In fact, LLMs show early indications of adaptability to different recommendation domains with modest fine-tuning [24, 25]. Finally, LLMs have so far been used for a number of recommendation tasks, such as rating prediction [41], item generation [42], and reranking [26] across domains (e.g., news [76], information retrieval [52]).

In this work we explore the potential of using LLMs for sequential recommendation problems [31]. In these problems, we consider as input a sequence of user interactions (i.e.a session) $S^u = (S^u_1, S^u_2, \dots, S^u_n)$, where u is a user, n is the length of the sequence and S^u_i are individual items. The aim is to predict the next interaction of the given sequence. Besides the recent sequential recommendation models mentioned in the introduction [23, 34, 67], S3-Rec [87] pre-trains a transformer architecture enhanced with self-supervised signals aiming to learn better data representations using the mutual information maximization principle. In earlier works the sequential recommendation problem has been modelled as a Markov Chain [17] or a Markov Decision Process [65]. Neighborhood-based approaches, such as SKNN [30], have also been proposed.

Related pieces of work in the sequential recommendation problem focus on cross-domain (Section 2.1), domain-specific (Section 2.2), and multi-modal (Section 2.3) recommendations.

2.1 Cross-domain LLM-based Sequential Recommendation Models

Early research work regarding LLMs for sequential recommendation problems showed mixed results for zero-shot and fine-tuned recommendations [14, 19, 24, 26, 50, 66, 73, 85]. The first deep zero-shot generative recommender model named ZESRec [14] comprises four unique properties: *cold users* (i.e., no overlapping users between training and test data); *cold items* (i.e., no overlapping items between training and test data); *domain gap* (i.e., training and test data originate from different domains); and *no access to target data* (i.e., target data are only available at inference time). ZESRec outperforms the zero-shot embedding-KNN and random baselines substantially, and proves beneficial for data-scarce startups and early-stage products. In [73] and [85] both GPT [61] and BERT [13] perform better than a random baseline in zero-shot sequential recommendations, while GRU4Rec [23] outperforms the fine-tuned versions of the LLMs. GRU4Rec and other baselines also outperform ZESRec [14] for zero-shot recommendations across domains. Conversely, P5 [19], which can produce zero-shot or few-shot recommendations without fine-tuning, has been shown to outperform a number of sequential recommendation models and ChatGPT—the least performant of all models [50].

The very recent VQ-Rec model [24] employs a transformer architecture and applies a novel representation scheme to embeddings retrieved from BERT to adapt to new domains. VQ-Rec outperforms a number of sequential recommendation models across datasets of different domains. Chat-Rec [16] converts user profiles and past interactions into prompts to build a conversational recommender system based on ChatGPT. The system is effective in learning user preferences and transferring them to different products, potentially allowing cross-domain recommendations. Finally, in both [24] and [81] SASRec with LLM embeddings is shown to improve over SASRec.

To explore the performance limits of text-based collaborative filtering with LLMs, Li *et al.* [43] conduct a series of experiments contrasting the former with the dominant identifier-based approach and the trending prompt-based recommendation using ChatGPT. Interestingly, ChatGPT seems inferior than text-based collaborative filtering, while the simple identifier-based approach remains highly competitive in the warm item recommendation setting. However, in the book recommendation tasks (i.e., book rating, user rating, and book summary recommendation), a ChatGPT-like system (BookGPT) developed by Li *et al.* [46] achieves promising results, especially in zero-shot or one-shot learning, compared to the classic book recommendation algorithms. Another investigation on the capabilities of ChatGPT in recommender systems [12] highlights that the model performs better with list-wise ranking, compared to point-wise and pair-wise. Overall, LLM-based recommenders surpass popularity-based recommenders in the movie, book, and music domains, but not so in the news domain.

2.2 Domain-specific LLM-based Sequential Recommendation Models

In the *movie, book, and e-commerce* domains, the lightweight TALLRec framework proposed by Bao *et al.* [4] aims to adapt LLMs for recommendations by structuring the recommendations as instructions, and tuning the LLMs through an instruction-tuning process. Experiments using the LLaMA model [68] demonstrate that TALLRec significantly advances the LLM capabilities, and is highly efficient even on a single conventional GPU. Similarly, PALR [80] employs user/item interactions for candidate retrieval, and a fine-tuned LLM-based ranking model to produce recommendations. Using LLaMA for the LLM, and compared to representative baselines including GRU4Rec [23] and SASRec [34], PALR appears more effective in various sequential recommendation tasks.

Furthermore, GPTRec [60], which is based on the GPT-2 architecture [62], employs a novel SVD tokenization algorithm and a Next-K recommendation strategy to generate sequential recommendations, accounting for already recommended items. GPTRec matches the quality of SASRec [34] but is more GPU-memory efficient due to its sub-item tokenization. Inspired by search engines, GPT4Rec [42] generates a set of search queries based on the item titles from a user's history, and then recommends items by searching these queries using beam search. Using GPT-2 [62] as a language model to generate queries that are used in a search engine using BM25 [64] as its score function, GPT4Rec outperforms state-of-the-art models including YouTubeDNN [11] and BERT4Rec [67]. In addition, multi-query generation with beam search increases the diversity of recommended items and the coverage of a user's interests.

In the *news* domain, Zhang and Wang [86] have recently released a framework for prompt learning (Prompt4NR) to predict whether a user would click a candidate news item. Prompt4NR slightly outperforms BERT-based models, reflecting the usefulness of the prompt learning approach for embedding knowledge in the pre-trained BERT models employed for news recommendation. Similarly, the prompt-based news recommender system (PBNR) by Li *et al.* [44] leverages the text-to-text T5 model [63] for personalized recommendations. PBNR is very accurate even with varying lengths of past user interactions, adaptable to new data, and can also satisfy user requirements through human-computer interaction. Finally, Li *et al.* investigate ChatGPT's performance in news recommendation [45] with respect to personalized recommendations, fairness, and fake news detection.

To enhance content-based recommendation, Liu *et al.* [51] propose a framework named ONCE which combines the deep layers of open-source LLMs (i.e., LLaMA [68]) as content encoders to improve the representation embeddings, and the prompting techniques of closed-source LLMs (i.e., ChatGPT [50]) to enrich the training token data. ONCE shows a substantial improvement of 19.32% compared to the state-of-the-art (NAML [74], NRMS [2], Fastformer [75]), showcasing the synergistic relationship between fine-tuning on the open-source LLMs and prompting on the closed-source LLMs. UniTRec, a unified text-to-text transformer and joint contrastive learning framework for text-based recommendation is proposed by Mao *et al.* [56] to better model two-level contexts of user history. The framework reaches state-of-the-art performance on three text-based recommendation tasks, namely news, quotation, and social media post recommendation.

2.3 Multi-modal LLM-based Sequential Recommendation Models

A number of research works combine both textual and visual information to develop multi-modal sequential recommendation models. MM-Rec by Wu *et al.* [77] includes a cross-modal candidate-aware attention network that selects relevant historical clicked news and models user interest in candidate news. Experiments suggest that multi-modal news information accelerates news recommendation performance. TransRec [72] slightly modifies the popular identifier-based recommendation framework by directly learning from the raw features of the multi-modal items in an end-to-end style, enabling transfer learning without depending on overlapped users or items. TransRec appears to be a generic model that can be transferred to various recommendation tasks. However, it entails a high training cost due to its end-to-end learning method, and can only be applied to image-only, text-only, and image-text scenarios.

To reduce TransRec's high training cost, Fu *et al.* [43] research whether an adapter-based variant of TransRec with two task-specific neural modules inserted into each Transformer block could perform similarly to the original fine-tuned model [72]. Their findings suggest a comparable performance between the two for both text and image recommendations. To shed light on the modality-based versus identifier-based recommendation architecture dilemma, Yuan *et al.* [81] conduct an empirical investigation tackling various associated questions. They conclude that modality-based recommendation is comparable to identifier-based, and could potentially surpass it given the rapid NLP and computer vision advances. Still, the former is more expensive in training, aligning with TransRec's findings [72].

2.4 Comparison to Previous Paper and Related Work

Our present work builds on top of our previous paper [20], where we proposed three orthogonal approaches of leveraging LLMs for the sequential recommendation problem. The first one, LLMSeqSim, obtains from an LLM the embeddings of items in a session, combines them to construct a session embedding, and maps the session embedding to the embedding space of available catalog items in order to produce next-item recommendations. The second approach, LLMSeqPrompt, involves fine-tuning an LLM to generate next-item recommendations. We provided session data in the form of prompt-completion pairs, where the prompt contains all items of each session but the last and the completion contains the last item, that is the ground truth. At prediction time, we asked the fine-tuned model to recommend an item for each session of the test set. Finally, the third approach, LLM2BERT4Rec, entails initializing the embedding layer of BERT4Rec with LLM embeddings to enable BERT4Rec to leverage the embeddings in its training process. Experimental results on the Amazon Beauty and a Delivery Hero dataset showed that this approach improves accuracy in terms of NDCG@20 by 15–20%.

In the meantime, a new body of research work involving LLMs in the sequential recommendation problem has swiftly emerged as we discuss in Sections 2.1, 2.2, and 2.3. With respect to leveraging LLM embeddings for producing recommendations, we identify no similar work to our LLMSeqSim approach. The majority of research efforts target fine-tuning LLMs for various recommendation tasks, and result in mixed conclusions, aligning with our LLMSeqPrompt observations. Researchers have devised a variety of methods including combining closed and open-source LLMs [51], using algorithms from search engines [42], utilizing user profiles [16], and structuring recommendations as instructions [4]. Overall, these methods appear mainly suitable for cold-start and data-scarce recommendation problems, and highly depend on the applied domain and dataset [12].

Regarding the initialization of neural sequential models with LLM embeddings, the recent approaches presented in [24], [27], and [81] differ from our work in particular in terms of the goals they pursue. VQ-Rec [24] targets cross-domain recommendations with a novel item representation scheme, while [27] and [81] evaluate whether recommendation models leveraging different modalities perform better than existing recommendation models that rely on item identifiers. In general, there seems to be a common conclusion that enhancing neural sequential models with LLM embeddings advances model performance due to the enriched information contributed by the embeddings [24, 81, 86]. As we will notice, this conclusion is also supported by our own results.

In this work we propose multiple and substantial extensions to our previous paper. Overall, the extensions regard (i) a new breed of hybrid approaches born from the above-mentioned approaches and from existing recommendation models, (ii) additional LLM embedding models, (iii) more dimensionality reduction methods, (iv) additional LLMs available for fine-tuning and new versions of those, (v) new fine-tuning task specifications, and (vi) more sequential recommendation models enhanced with LLM embeddings.

These extensions are implemented in three orthogonal plus a new breed of hybrid approaches, shown in Figure 1. Each approach comprises a number of technical aspects for which we devise and investigate alternative choices. We evaluate a large variety of configurations for all approaches on the same experimentation testbed, against the state-of-the-art sequential recommendation models, using three datasets of different domains. In this way, we delve into each of the fundamental approaches regarding the involvement of LLMs in the sequential recommendation problem and assess their potential to advance the existing state of the art and point to promising new avenues for research. Finally, based on encouraging results, we explore hybrid approaches by combining

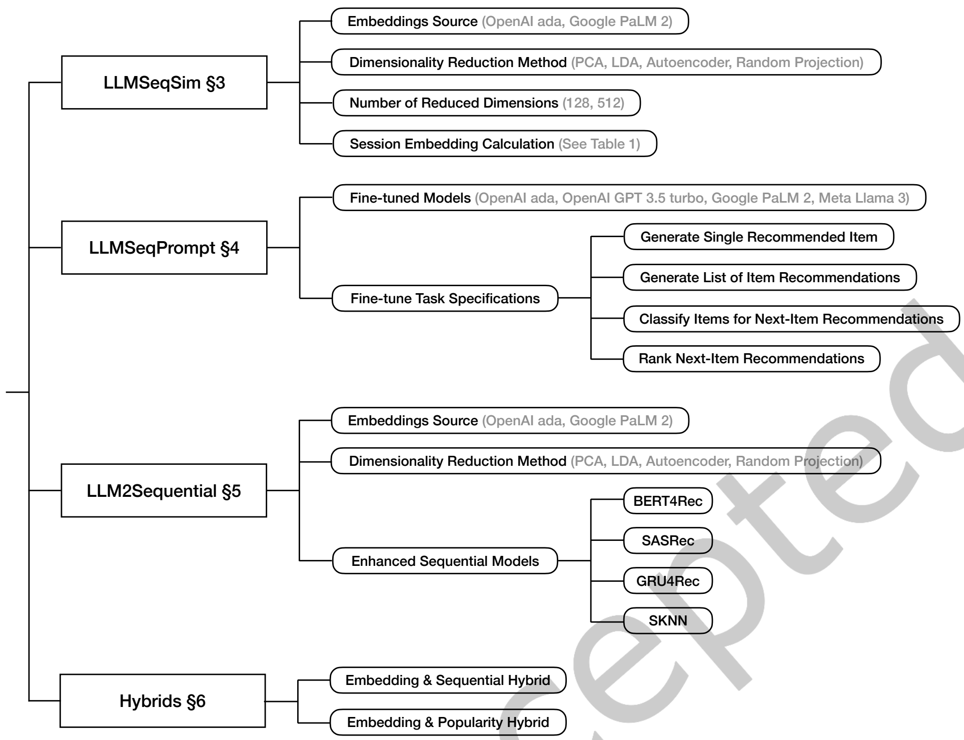

Fig. 1. Layout of work. Each top-level branch of the tree shows one approach presented in the corresponding section. In brackets we indicate the alternatives we used.

variations of the proposed methods, aiming to use each individual method's dominant characteristics in an ensemble to improve recommendations further.

3 LLMSeqSim: Semantic Item Recommendations via LLM Embeddings

Our goal with this first approach is to explore if recommendations can benefit from a notion of semantic similarity as provided by LLMs. Specifically, we assume that LLM embeddings encapsulate semantic relationships regarding items of many domains based on their massive training data. Therefore, when we encounter a recommendation use case for a particular domain, the LLM embeddings can position items of the domain relative to other items, considering the items' semantics and their relationships with relevant concepts of the domain. In this way, embeddings can enable recommendations that stem from a deep semantic intuition of the items' nature. In addition, putting together a sequence of such item embeddings opens a potential for exploring a space of recommendations that complete the sequence in novel ways not envisioned before, as we elaborate in Section 3.4.

In the proposed LLMSeqSim (Sequential Similarity) approach, we leverage LLM embeddings to produce recommendations in six steps. We depict these steps in Figure 2, applied on a food recommendation use case. At the first step we supply metadata of the catalog's items, such as the names of the products, to an LLM embedding API and retrieve the corresponding embeddings. Then, at the second step, we can optionally reduce the dimensions of the embeddings to improve, if possible, the performance of the recommendation task. Third, we store the embeddings of the item catalog.

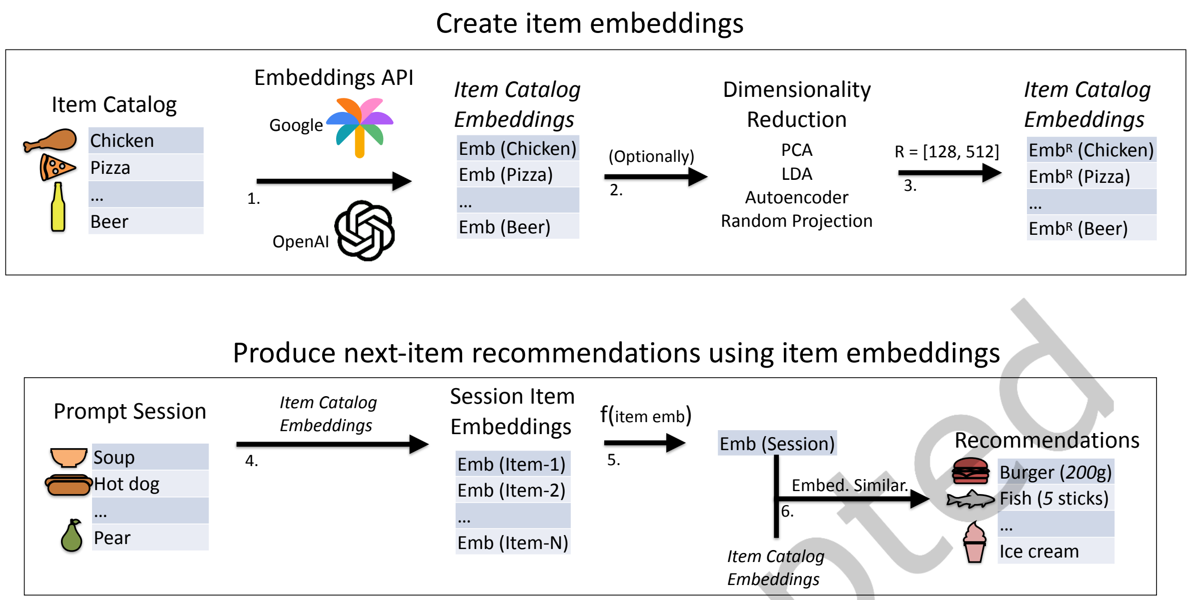

Fig. 2. Semantic item recommendations via LLM embeddings.

When a prompt session arrives, for instance a cart of products, we query the item catalog embeddings using the prompt's item metadata and retrieve their embeddings at the fourth step. Then, we compute a session embedding for each prompt session in our test set by combining the embeddings of the individual items in the session. As a sixth step, we compare the session embedding to the embeddings of the items in the item catalog using cosine, Euclidean, and dot product similarity.[3] Finally, we recommend the top-k products from the catalog with the highest embedding similarity to the session embedding.

Overall, LLMSeqSim shares similarities with the sequential model proposed in [41], which also relies on creating item embeddings from textual representations derived from metadata features, although not based on LLMs. We identified four technical aspects in this processing pipeline, where different implementation alternatives are possible. Below, in Sections 3.1–3.4, we elaborate our investigation of the different aspects involved in the LLMSeqSim approach.

3.1 Embedding Sources

With regard to LLM sources that provide embeddings in the first step, we explore both OpenAI's embedding model (`text-embedding-ada-002`)[4] as well as Google's PaLM2-based embedding model (`textembedding-gecko@002`).[5] Our motivation for choosing these embedding models is to investigate and uncover commonalities and differences between embeddings provided by diverse and highly performing foundational models.

[3]The choice of the similarity measure did not significantly impact the results. We used cosine similarity.
[4]https://platform.openai.com/docs/guides/embeddings/second-generation-models
[5]https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/get-text-embeddings

3.2 Dimensionality Reduction Methods

After retrieving the embeddings, we can directly use them. However, given that the embeddings are of multidimensional nature and especially LLMs utilize a large number of dimensions to model a vast number of concepts in the same space, we take the opportunity to lower the embedding dimensions using a set of dimensionality reduction methods that may improve the performance of the recommendation task.

We applied four different dimensionality reduction methods. The first dimensionality reduction method we used is Principal Component Analysis (PCA), a method that has proved very effective for the initialization of the embedding layer of neural models [20]. Then, as an alternative to PCA we also tried Linear Discriminant Analysis (LDA), which projects the data to $k - 1$ dimensions so that the projected data best fit k classes. We used the most characteristic item metadata to denote classes, such as the name or category of each item depending on the dataset. The third approach was to apply an autoencoder to reduce the dimensions of embeddings, while experimenting also with $\ell 2$ regularization that might further benefit the reduction by driving minor embedding scores to zero. Finally, the fourth approach was a random projection of embeddings [32] to a lower dimensional space with the premise that the lower space provides a suitable approximation of the original space in terms of distances between the points.

3.3 Number of Reduced Dimensions

In each of the four dimensionality reduction methods we tried different numbers of lower embedding dimensions. The motivation behind this exploration is to discover whether and to which extent dimensionality reduction produces embeddings of higher quality that lead to improvements in recommendation performance.

Our starting dimensions were 768, which is the number of dimensions of Google embeddings, and 1536, which is the number of dimensions of OpenAI embeddings. We tried reducing these to 128 and 512 dimensions to balance the overall hyperparameter search space and the performance impact of the dimensions based on observations from ad-hoc experiments. Prior to the hyperparameter search process, we also explored 256, 64, and less than 64 dimensions manually in ad-hoc experiments, but did not identify any potential for further investigation in low dimensions.

3.4 Session Embedding Computation

In the third step, we compute an aggregate session embedding for each session in our test set by combining the embeddings of the individual products in the session. The idea is similar to that of Global Average Pooling (GAP) [49] used in Convolutional Neural Networks (CNNs). Before the final, top layers of a network, the inputs have been transformed to two-dimensional structures. In CNNs, each of these structures, called a feature map, can be understood as the result of applying filters to the input (typically image) data. GAP takes each feature map and produces a single value, its average, which is then used in the following layers. In this way, a structure of (*height* × *width* × *feature maps*) dimensions is transformed to a one-dimensional vector with *feature maps* entries. Instead of images, we deal with a structure of (*session length* × *embeddings dimension*), which we want to reduce to a one-dimensional vector with *embeddings dimension* entries. This vector will pool the session embeddings so that the resulting vector will represent the whole session. Intuitively, as each embedding is a vector that represents the semantics of a session item, a global average over all item embeddings in a session will produce a vector whose semantics will represent the average semantics of the session.

It is possible to use different pooling methods, beyond global average pooling. We therefore try a number of different pooling strategies: (i) the average of the product embeddings, (ii) a weighted average using linear, harmonic, and exponential decay functions depending on the position of the item in the session, and (iii) only

Technique	Decay per Step
Constant linear	$1/10$
Scaling linear	$1/\lvert S\rvert$
Scaling quadratic	$1/\lvert S\rvert^2$
Scaling cubic	$1/\lvert S\rvert^3$
Log	$1/\log(i+1)$
Harmonic	$1/i$
Squared harmonic	$1/i^2$

Table 1. Computation techniques for weighted averages; $|S|$ stands for the length of session S.

the embedding of the last product.[6] We provide the exact computation techniques of the weighted average in Table 1, where for each technique we specify the decay computation at each step of a session.

4 LLMSeqPrompt: Prompt-based Recommendations by a Fine-Tuned LLM

In this approach, we inject domain knowledge to the collective information that a base LLM incorporates. Our goal is to thereby combine fundamental information about the items of the domain encoded in the LLM with domain-specific information conveyed in the fine-tuning process. Related approaches to fine-tune pre-trained LLMs for recommendation problems in different ways were successfully explored in [4, 72, 83].

To this end, we fine-tune an OpenAI `ada`[7] model, an OpenAI `GPT-3.5 turbo`[8] model, and the model `text-bison@001`[9], which is based on Google PaLM2. GPT succeeded `ada` as OpenAI's recommended base model for fine-tuning, while PaLM2 is a base model created by Google. While there is general consensus on the overall architecture and training process of a base LLM, differences in architectural choices, its configuration, and the training data mean that LLMs can differ significantly. Thus, the three selected models enable us to compare and contrast two fundamentally different LLMs, and to also assess the performance evolution of the two OpenAI models.

We fine-tune the models on training samples consisting of a prompt (the input) and a completion (the intended output). First, we take the training sessions and format them as prompt-completion pairs according to the fine-tuning task specification.[10] We also separate a small subset of the training sessions and provide them to the fine-tuning process as validation set. We supply 500 sessions for validation to OpenAI and 250 sessions to Google PaLM, which is the maximum number of validation cases that the PaLM API allows. Then, we supply the prompt-completion pairs to a model's fine-tuning API. In all configurations, we fine-tune the model until the validation loss converges in order to achieve the best performance.

After fine-tuning, we provide the prompts of the sessions in the test set to the fine-tuned model and retrieve recommendations. Finally, for generation tasks that may lead to hallucinated recommendations, we map non-existing items back to the item catalog through dot product embedding similarity[11] and obtain the final recommendations.

Besides employing a set of diverse LLMs, we propose four alternative specifications for the fine-tuning task. Our rationale is to examine the behavior and performance of a model on different problem formulations, namely *generation of a single recommended item* (Section 4.1), *generation of a list of item recommendations*

[6]We also tried to create an aggregated session embedding by concatenating the plain product names and then querying the OpenAI embeddings API. This however led to worse results.

[7]https://platform.openai.com/docs/guides/fine-tuning

[8]https://platform.openai.com/docs/guides/fine-tuning/what-models-can-be-fine-tuned

[9]https://cloud.google.com/vertex-ai/docs/generative-ai/models/tune-models

[10]We present the precise task specifications in Sections 4.1 to Section 4.4.

[11]We use OpenAI embeddings to resolve hallucinations, which are normalized. Therefore, the dot product is the same as cosine similarity.

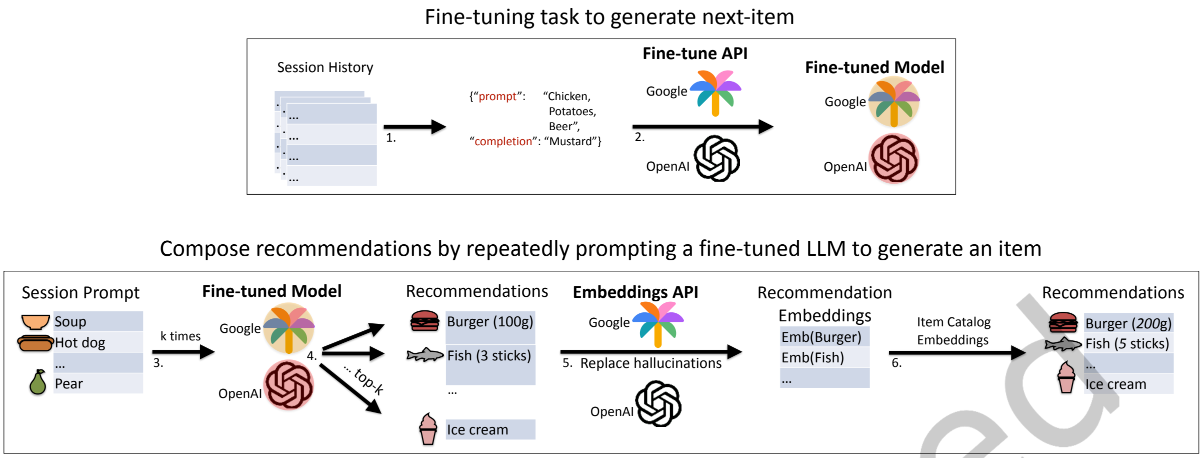

Fig. 3. Next-item generation by fine-tuned LLM.

(Section 4.2), *classification* (Section 4.3), and *ranking* (Section 4.4).[12] Generation, classification, and ranking enable an investigation of fine-tuning from less to more restricted settings. Fine-tuning a model on a generation task allows the model to recommend virtually anything. In a classification task, in contrast, a fine-tuned model is presented with a candidate pool of item recommendations presented as categories and selects a top-k subset of those similarly to multi-class classification. Finally, the ranking task further narrows the freedom of the model in that the set of available recommendations is provided in the prompt. This allows the model to focus on the best ranking of those recommendations.

4.1 Fine-tuning Task: Generate a Single Recommended Item

In this variant, the prompt is a session, which contains a list of item metadata, e.g., product names, except for the last item, and the completion contains the metadata of the last item in the same session according to the first step in Figure 3. At prediction time, following the fine-tuning process, we repeat a prompt k times ($k = 20$) and collect a single recommendation produced by the model per prompt invocation according to the fine-tune task formulation. We use the tendency of the model to make the same recommendation in different invocations for the same prompt as a proxy of its confidence for the recommendation. We then collect and deduplicate the recommendations over the k invocations and rank the recommendations by frequency of appearance. The fine-tuned LLM, being a generative model, may also return hallucinated products. To remedy that, in the fifth step, we retrieve the embedding of each deduplicated recommendation and finally take the catalog's product that is closest in terms of embedding similarity using the dot product.

Repeating the prompt allows us to get a number of different recommendations from the single recommendation returned each time. Alternatively, we can overcome the limitation of a single recommendation per prompt, as we do next.

4.2 Fine-tuning Task: Generate a List of Item Recommendations

In this approach, we teach an LLM to embrace the role of a typical recommendation model and generate a list of next-item recommendations ranked by order of relevance.

[12]Budget and time limitations allowed us to apply the fine-tuning tasks of Sections 4.2, 4.3, and 4.4 to the Beauty dataset only.

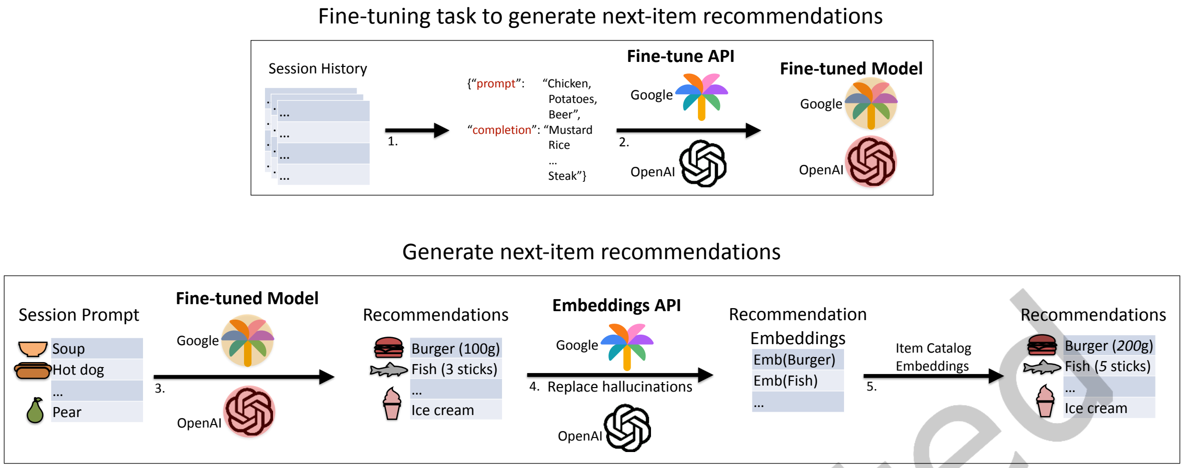

Fig. 4. Recommendation list generation by fine-tuned LLM.

To convey this task in the fine-tuning process as we depict in Figure 4, we provide in the prompt a session containing a list of items except the last one. In the prompt completion we provide a list of item recommendations that is compiled by a state of the art recommendation model. The rationale is to ask a proven recommendation model to provide recommendations for the training sessions and use those as prompt completions for the fine-tuning process. In our case, we used LLMSeqSim in the Beauty dataset due to its high performance.[13] Finally, we insert the ground truth item at the top of the list, if not already there, and fine-tune an LLM with the same prompts and the corresponding compiled list of recommendations as completion.

At prediction time, we ask the model to produce recommendations by passing the items of each test session except the last item. The fine-tuned model may return hallucinated recommendations, which we map back to the item catalog following the same method as in Section 4.1.

4.3 Fine-tuning Task: Classify Items for Next Item Recommendations

In this approach, we explicitly provide the LLM with the eligible options for recommendation, instead of allowing it to 'freely' return recommendations. The learning goal is thus to select and rank a set of pertinent item recommendations from an overall pool of items that we pose to the model as classification categories. The intuition behind this formulation is that there exists a small subset of popular and diverse items in each domain that will please most customers. Restricting the set of options this way also helps us to avoid niche items that receive scant attention.

We state the next item recommendation problem as a multi-class classification task as follows, see also Figure 5. Prior to the fine-tuning process, we apply a clustering algorithm, namely K-means, to group items to clusters based on the embedding distances of their metadata.[14] The resulting clusters can be seen as a set of diverse item groups that are representative of the whole item space. We pick a relatively large number of clusters, i.e., 200, to maximize the probability of a good representation of semantically different items. Then we elevate the most popular item of a cluster to represent the group of items in the cluster. In this fashion, we pick an array of

[13] In theory, we could also query more than one recommendation model and use the recommendations of the model that achieves the highest NDCG, or another metric.

[14] We use product names in the Amazon Beauty dataset.

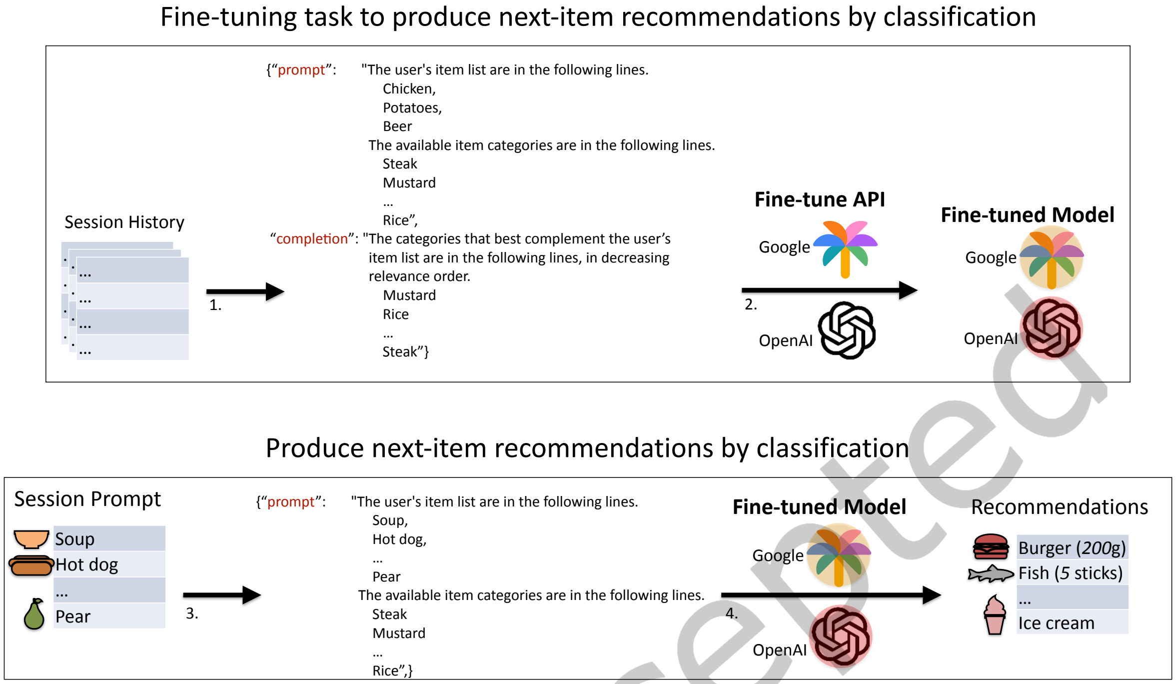

Fig. 5. Multi-class classification of items by fine-tuned LLM.

popular and diverse candidate items that could represent good-quality recommendations across the total item space. Finally, for each training session, we take the embeddings of the ground truth item and the items that represent the classification categories and select the top-k semantically closest classification categories for each ground truth item.

The prompts for fine tuning include the input session of items (except) the last item and the classification categories. In the corresponding completions, we include the top-k classification categories that we computed in the previous step, see Figure 5 for an example. At prediction time, we ask the model to classify a prompt session to top-k categories, which correspond to next item recommendations. Note that since the candidate recommendations are included in the prompt, there is no need for an extra step to map the recommendations back to the item space.

4.4 Fine-tuning Task: Rank Next Item Recommendations

In this final variant, visualized in Figure 6, we fine-tune the models to learn a ranking task. Specifically, we create a prompt that consists of *(a)* a session of items except the last one, and *(b)* a set of recommendations including the ground truth, that is the last item of the session. We take the set of recommendations from an existing recommendation model depending on the dataset[15] similarly to Section 4.2, and shuffle them. Then, in the prompt completion we provide the recommendations in the order produced by the model and ensure that the ground truth item is at the top of the list, putting or moving it there as necessary. At prediction time, we ask the

[15]We chose LLMSeqSim for Amazon Beauty.

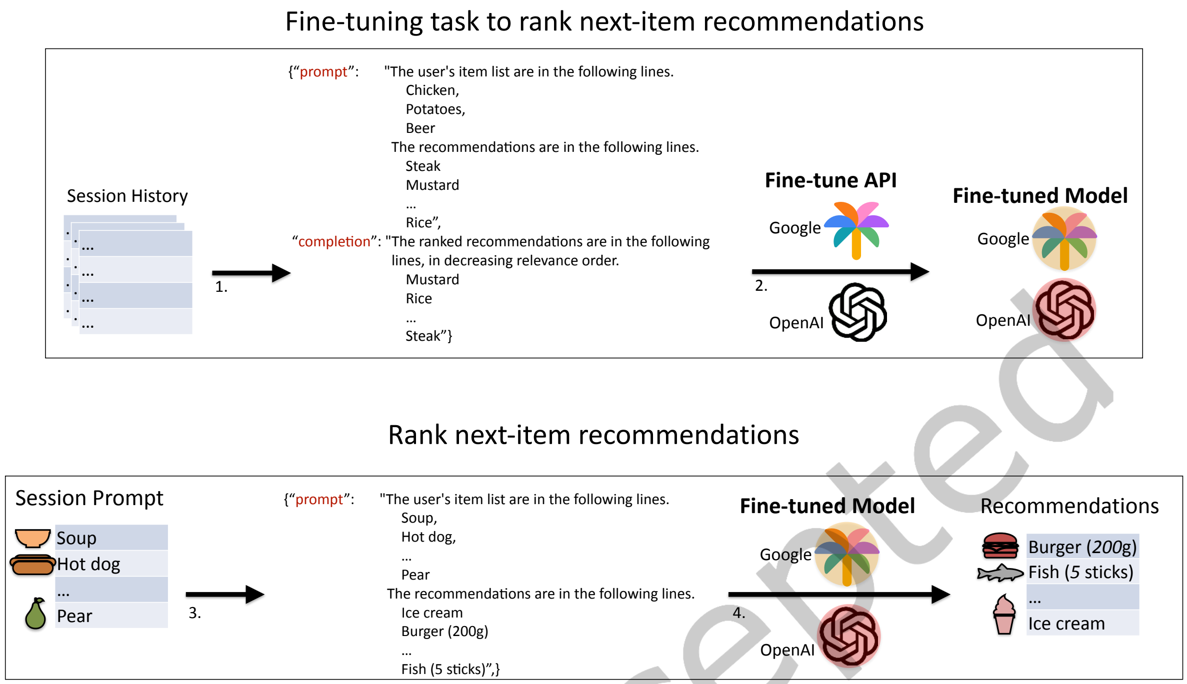

Fig. 6. Rank recommendation slate by fine-tuned LLM.

fine-tuned model to re-rank the recommendations in the prompt based on what would be the best completion for a prompt session.

5 LLM2Sequential: LLM-enhanced Sequential Models

In the third approach, our goal is to leverage the semantically-rich item representations provided by an LLM to enhance an existing sequential recommendation model. Existing recommendation models typically operate on unique identifiers of the items of a domain by learning motifs between the items from the interaction data they are trained with. By equipping the models with meaningful item representations we aim to enable them to learn deeper relationships that can lead to more advanced recommendations.

We reinforce a number of sequential models with LLM capabilities, namely BERT4Rec [67], SASRec [34], GRU4Rec [23], and SKNN [53]. Of those, BERT4Rec, SASRec, and GRU4Rec are neural recommendation models that operate on item embeddings. Specifically, these models feature an embedding layer that accommodates embedding representations of items. We supply item embeddings retrieved from an LLM as input to the embedding layer of these models. On the other hand, SKNN is a neighborhood-based sequential model that computes session similarity based on the presence of common items included in the sessions. In this work we adapt SKNN to consider another notion of session similarity grounded on the embedding similarity of items comprising a session.

Besides the consideration of different sequential models, we explore different configurations as done for LLMSeqSim described in Section 3:

- we use alternative LLM embedding models, i.e., those from OpenAI and Google;
- we explore different dimensionality reduction methods.

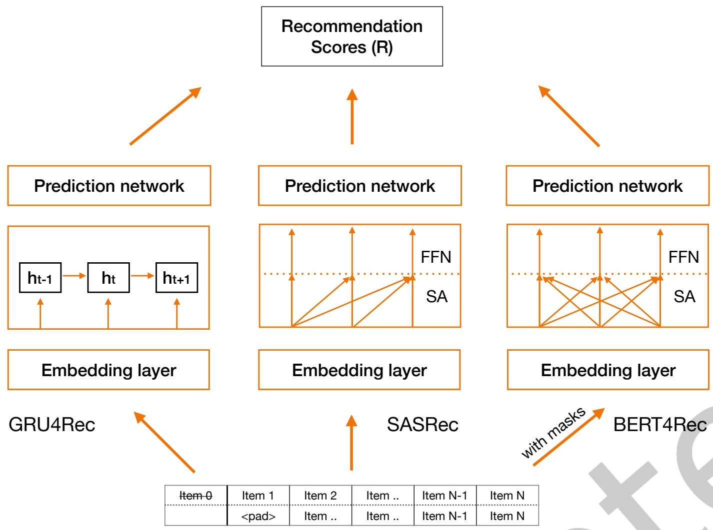

Fig. 7. Architectural components of GRU4Rec, SASRec, and BERT4Rec.

Following the above-mentioned overview of our work on the enhancement of sequential models with LLM capabilities, we organize the description of this approach in two parts. First, we summarize each of the sequential models we consider (Section 5.1). Then, we elaborate how we incorporate LLM embeddings to the workings of the models (Section 5.2).

5.1 Baseline sequential models

We provide a concise description of the baseline sequential models we target, namely SKNN, GRU4Rec, SASRec, and BERT4Rec. Figure 7 depicts the main architectural components of the neural models. We note all the details of our neural model implementations, including any deviations from the original model implementations, in the online material.[16]

SKNN (Session-based kNN) finds items to recommend by considering sessions similar to the input session[53]. Given a similarity function $\mathrm{sim}(S_1, S_2)$ between two sessions S_1 and S_2, the model considers the top-k most similar neighboring sessions N_S of a session S and computes the recommendation score of an item i according to the following equation, where $\mathbf{1}(\cdot)$ represents the indicator function.

$$\mathrm{score}(i, S) = \sum_{S' \in N_S} \mathrm{sim}(S', S) \cdot \mathbf{1}(i \in S') \tag{1}$$

The scores from Equation 1 can be normalized to produce the probability distribution $P(s_{n+1}|S)$. Though the similarity function is intentionally left open, cosine similarity between the sessions' binary interaction vectors has proven to work well and lead to competitive results in the literature [36, 54].

GRU4Rec employs a recurrent neural network based on GRU [10]. Its overall architecture consists of an item embedding layer, a GRU layer, and a prediction network. GRU4Rec is trained by sequentially processing sessions, where at each timestep it uses the current item in the session and a hidden state computed from previous sessions to predict the item at the next timestep. To elaborate, the item embedding layer consumes the item ID s_t at

[16]https://github.com/dh-r/LLM-Sequential-Recommendation/blob/main/online_material.md

the current timestep t and returns its associated embedding e_{s_t} from the item embedding matrix E. This item embedding matrix has dimensions $|I| \times e$, where $|I|$ is the number of items and e is the embedding dimension. Then the embedding e_{s_t} is fed to a GRU layer. A GRU layer is stateful in that it maintains a hidden state h_t while processing a session. This allows the model to use information from the previously seen items $s_0, s_1, \ldots, s_{t-1}$ next to item s_t to predict s_{t+1}. Hence, the GRU layer computes h_t based on its previous hidden state h_{t-1} and s_t. Finally, the model's output is a score R_t for each item, which represents the model's confidence for each item that it will be the item in the session on timestep s_{t+1}. Using R_t we can compute next-item recommendations by taking the items with the top-k scores on the last timestep.

In the original paper [23] the authors experimented with an embedding layer but dropped it and favored a one-hot item encoding because of slightly better performance. We kept the embedding layer to be able to draw a meaningful comparison with other models using LLM embeddings. We elaborate the model's implementation details in the online material.

SASRec is the first transformer architecture [70] designed for the sequential recommendation task. It consists of an embedding layer, one or multiple transformer layers, and a prediction network. The transformer layers are based on the encoder stack introduced in [70]. In short, each of the L transformer layers consists of a self-attention (SA) and a feed-forward network (FFN) module. The model is trained by predicting the identity of s_{t+1} for each timestep t like GRU4Rec. However, SASRec computes R for all timesteps $t \in [1, N]$ at once. R is a $N \times I$ matrix, where $R_{t,i}$ denotes the model's confidence that item i will be the item at position $t + 1$.

Given a sequence S, either from the embedding layer or a preceding transformer layer, the self-attention module takes a weighted average of all the embeddings in the sequence, for each timestep in the sequence. Given the output sequence of embeddings S^L of the last transformer layer, the prediction network of SASRec projects this sequence into scores over all the items. For next-item prediction, we therefore use R_N, which denotes the vector of the last item N in a session and contains the scores for all items to be the item succeeding the last item in the session.

BERT4Rec is a state-of-the-art neural recommendation model, which employs the transformer architecture [70] of BERT [13]. BERT's transformer architecture consists of an embedding layer, a stack of transformer layers, and a prediction network. Furthermore, BERT features a masked language model training protocol, which involves masking items at random positions and letting the model predict their true identity.

Initially, the embedding layer embeds an input sequence of (potentially masked) item IDs into a sequence of embeddings using both the item ID and the item position. Then the transformer encoder layers process the embedding sequence using a multi-head attention module and a feed-forward network shared across all positions. Finally, the projection head projects the embeddings at each masked position to a probability distribution in order to obtain the true identity of the masked item. The projection head reuses the item embeddings of the embedding layer to reduce the model's size and to avoid overfitting.

5.2 LLM-enhanced Sequential Models

In this approach, we incorporate LLM embeddings to two different classes of widely applied sequential models: neural sequential models represented by BERT4Rec, SASRec, and GRU4Rec, and neighborhood-based ones represented by SKNN.

Neural Models. As we depict in Figure 8, to allow the neural sequential models to leverage the rich information encoded in LLMs, we employ LLM embeddings for the initialization of the neural models' item embeddings located in the embedding layer. In order to align the embedding dimension of the LLM embeddings (e.g., 1536) with the configured dimension of the neural models' embedding layer (e.g., 64), we employ and assess different dimensionality reduction methods. Finally, we train the enhanced model in the same manner as our baseline

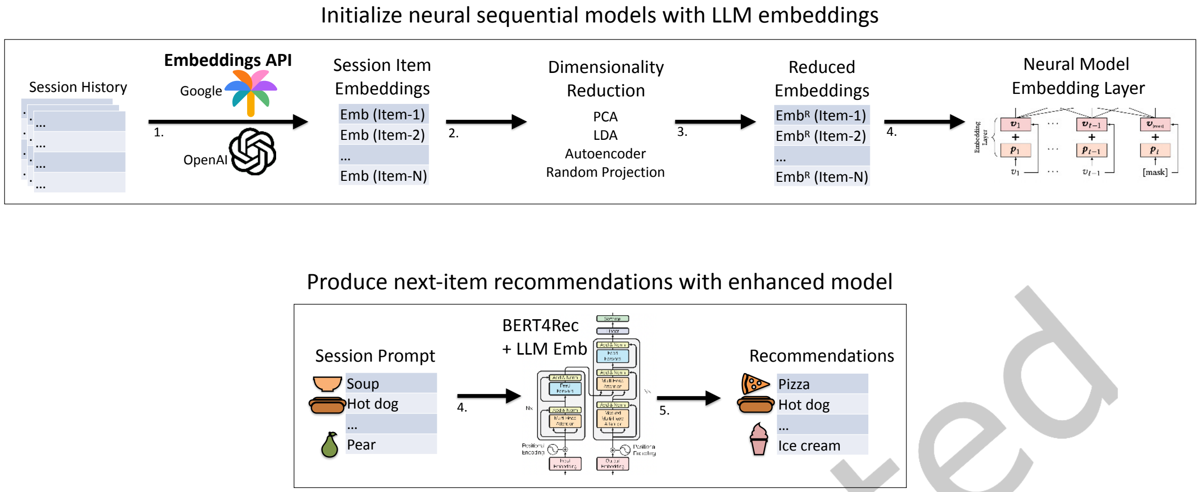

Fig. 8. Neural sequential models enhanced with LLM embeddings.

neural sequential model. The LLM embeddings and dimensionality reduction methods used are described in Section 3.

In terms of technical aspects, we note that the item embeddings in our approach are trainable and they are computed based on item metadata of the domain. For the Beauty and the Delivery Hero datasets, the metadata used to compute the embeddings are the names of the products. In the Steam dataset, the items of the domain are games. Typically, the names of the games are not closely linked with the concept of a game. Thus, we concatenate the names of the games with tags that accompany and characterize each game. The IDs of the items are not included to the input of the embedding computation.

In the overall process, the collaborative information thus enters the LLM2Sequential model as a sequence of IDs representing the items of the domain that appear in the sequence. In the embedding layer, the sequence of IDs becomes a sequence of embeddings in the following manner. First, the item embedding of each item in the sequence is retrieved from the embedding matrix based on the ID of the item. The item embedding is the embedding previously received by an LLM based on metadata of the item provided as input to the LLM. Second, each item embedding is summed with the positional embedding of the item's position in the sequence to form the final embedding representation. Then, the embeddings are transformed while passing through layers of the model in the feed-forward pass where the collaborative and textual information are fused in the multilayer perceptron of the feed-forward network. Finally, through backpropagation of the gradients that minimize the cost, the weights of each layer are updated leading to the update of the embeddings in the embedding layer as well.

Neighborhood-based Models. To enable neighborhood-based models, and SKNN in particular, to use LLM embeddings, we modified the model such that it can operate on the item embeddings. Specifically, we compute a session embedding for each training session. To compute a score for predicting item i of a prompt session S we take the session embedding e_S of session S and then consider the similarity of e_S with the session embedding $e_{S'}$ of each session S' of the top-k nearest neighboring sessions N_S. We then arrive at Equation 2, which differs from Equation 1 in that the latter computes the similarity between two sessions based on the co-occurence of items in these two sessions, while here we derive the similarity from the session embeddings. We experiment with a number of aggregation methods for producing a session embedding from the individual item embeddings of a session. These aggregation methods are the same as the ones used to produce a session embedding from individual item embeddings in LLMSeqSim (Section 3). In our experiments, we also allow the model to use one embedding combination strategy for the training sessions and a different one for the prompt session.

$$\text{score}(i, S) = \sum_{S' \in N_S} \text{sim}(e_{S'}, e_S) \cdot \mathbf{1}(i \in S') \tag{2}$$

6 Hybrids

The three top-level approaches that we elaborated in Sections 3 (LLMSeqSim), 4 (LLMSeqPrompt), and 5 (LLM2Sequential) have unique characteristics.

- **LLMSeqSim** relies on the semantic similarity provided by an LLM between items and compiles a session embedding with desirable properties across items of a session for producing pertinent next-item recommendations.
- **LLMSeqPrompt** compiles fundamental and domain-specific knowledge into a single model potentially enabling recommendations that share information elements from both origins.
- **LLM2Sequential** is based on incorporating LLM embeddings to the sequential models. This approach introduces meaningful item representations to the training process of sequential models, aiming to boost their learning efficacy towards advanced recommendations.

In this section, we combine the unique characteristics of LLMSeqSim and LLM2Sequential to create stronger hybrids. The hybrids capitalize on the notion of item popularity. Item popularity enables segregating the item space between widely selected items, for which a lot of information is available, and niche items with scant interactions. On the models' side, we have candidate models that —as we will see— perform very well on popular items (e.g., LLM2Sequential), while other models are agnostic to item popularity and base their recommendations on other signals (e.g., LLMSeqSim). Thus, we can use the popularity property of items to select an appropriate model based on a cutoff popularity threshold. This threshold can be applied either directly on the recommendations of a model (Section 6.2) or alternatively on the last item of a prompt session under the premise that the last item is the driving factor for the next item that will follow (Section 6.1). In this paper, we motivate and research the potential of two concrete hybrids[17] based on the aforementioned models and the notion of item popularity. We describe these hybrids below.

6.1 Embedding & Sequential Hybrid

Based on the intuition that similarity-based and content-based approaches can lead to better coverage and novelty [29, 57, 69], while many state-of-the-art recommendation models maintain a bias to popular items, we devise a popularity-based hybrid based on two models. The semantic item recommendation model using LLM embeddings (LLMSeqSim) focuses solely at the semantic representation of items and is oblivious to the notion of item popularity, while LLM2Sequential models are generally good at recommending popular items. In support of this motivation, Figure 9 shows the performance of different models with respect to item popularity. Indeed LLM2BERT4Rec and SKNNEmb exhibit higher hit rates when they recommend for prompt sessions whose last item is popular. In contrast, the hit rate of LLMSeqSim is stable across the popularity of the last item in the prompt sessions.

In practice, we can create a hybrid where LLMSeqSim is used to recommend when encountering unpopular items and LLM2BERT4Rec or SKNNEmb are used to recommend when dealing with popular items at the end of

[17]We also experimented with a hybrid based on embedding model variations, which did not perform better than the individual variations. We share the details in the online material.

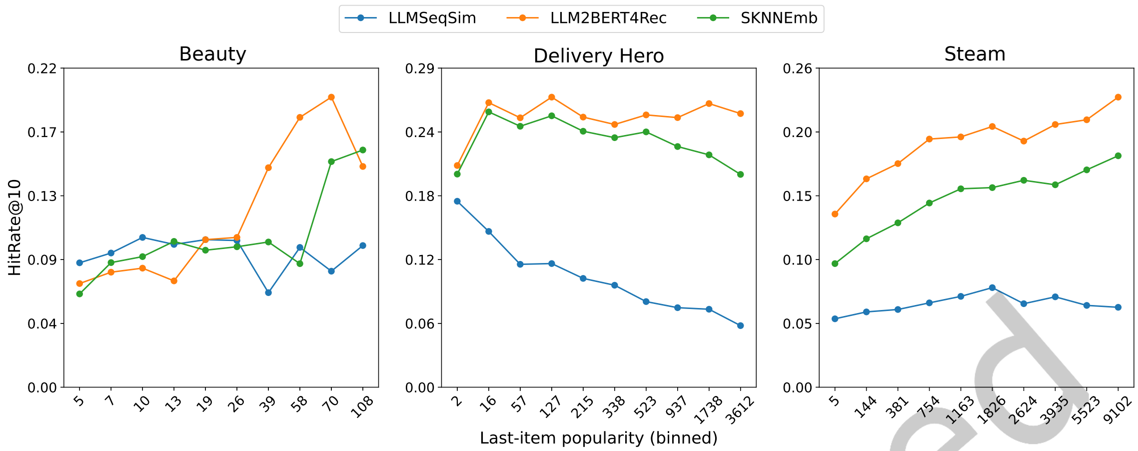

Fig. 9. Performance per last item popularity.

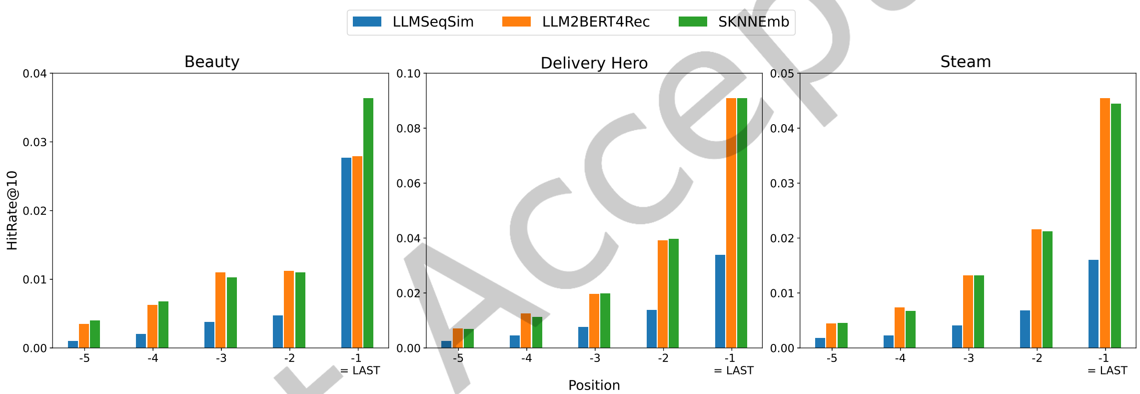

Fig. 10. Performance per item position in session.

a prompt session. Specifically, we apply a cutoff point on the popularity of the last item of a prompt session and use LLMSeqSim to provide recommendations for that prompt session if the popularity of the last item falls under the cutoff point or one of the two aforementioned sequential models otherwise. Our intuition for using the last item as a driver for the prediction is founded on its impact on correct recommendations according to our analysis in Figure 10, which shows the performance of a model if we treat different positions in a prompt session as last. The figure clearly shows that the item in the last position bears the strongest impact on the performance of recommendations. We apply hyperparameter search on the training set to identify the best cutoff point for popularity considering the quantiles of item popularity per dataset.

6.2 Embedding & Popularity Hybrid

The LLM-embedding based recommendation model (LLMSeqSim) leverages a semantically rich representation of items, but completely lacks desirable notions of recommendations, such as the popularity and diversity of items that traditional recommendation models learn to embrace during their training. Therefore, we create a hybrid based on LLMSeqSim that is also aware of popularity.

The hybrid considers popularity by enforcing a threshold on it such that LLMSeqSim does not recommend obscure items. The threshold could be made relative to the number of options that the model has picked for recommendations in order to include highly relevant recommendations of low popularity. But we did not further explore this alternative. As with the previous hybrid (Section 6.1), we apply hyperparameter search on the training set to determine the optimal popularity cutoff point. We also apply a threshold on diversity in order to enforce the semantic diversity of items in the top-k list of recommendations quantified by the items' embedding similarity. Specifically, we remove any item whose pair-wise embedding similarity with any other item in the recommendation list exceeds the threshold. We tried both diversity alone and combined with popularity, but it did not produce any promising results. Therefore, we proceeded with popularity only. We share all results in the online material.

7 Experimental Evaluation

In this section we describe our experimental setup (Section 7.1), the results of our empirical evaluation (Section 7.2), and observations from hyperparameter tuning (Section 7.3). We publicly share the code and data of our experiments to ensure reproducibility.

7.1 Experimental setup

The experimental setup consists of the datasets and their pre-processing (Section 7.1.1), the metrics used to evaluate experimental results (Section 7.1.2), the models that were included in the experiments (Section 7.1.3), the hyperparameter tuning process for executing models with their best configurations (Section 7.1.4), the experimental setup of S3-Rec (Section 7.1.5), TALLRec (Section 7.1.6), and LlamaSeqPromptGenItem (Section 7.1.7) as well as precise information regarding which models were executed on which datasets, including exceptions (Section 7.1.8).

7.1.1 Datasets and Data Splitting. We use the public Amazon Beauty [21] dataset[18], a novel, real-world proprietary dataset from Delivery Hero, and the public Steam dataset[19] from the gaming domain. The Beauty dataset contains product reviews and ratings from Amazon. In line with prior research [3], we pre-processed the dataset so that there are at least five interactions per user and item (p-core = 5). The Delivery Hero dataset contains anonymous QCommerce sessions for dark store and local shop orders.[20] To better simulate a real-world setting, we did not pre-process this dataset, except that we removed sessions with only one interaction. In the Steam dataset we applied p-core = 5 pre-processing and removed interactions that contain items with insufficient metadata (title, genres, and tags). Dataset statistics are provided in Table 2.

We selected a diverse set of datasets to ensure that our findings are not limited to datasets with particular characteristics. The datasets are of different nature, originate from different domains, and have different features. In the left part of Figure 11, we visualize the popularity of items in the sessions. After counting the number of occurrences of each item, we rank them in descending order (the most popular being the first) and then plot the log of the frequency on the log of the rank. The Delivery Hero dataset has a tail of items that appear 1 to 4 times, as it is not p-core = 5. In the right part of the figure, we show the log of the frequency of the sessions' length. The Delivery Hero dataset has shorter sessions than the Steam dataset, and fewer longer sessions (those with more than 40 items) than either of the other two datasets.

[18]https://cseweb.ucsd.edu/~jmcauley/datasets/amazon/links.html, Section *Per-category file*.
[19]https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data; Version 2: Review Data and Item Metadata.
[20]QCommerce is a segment of e-Commerce focusing on fast delivery times on the last mile.

Dataset	# sessions	# items	# classes	# interactions	Avg. length	Density
Beauty p-core = 5	22,363	12,101	220	198,502	8.9	0.073%
Delivery Hero	258,710	38,246	NA	1,474,658	5.7	0.015%
Steam p-core = 5	279,290	11,784	700	3,456,395	12.4	0.105%

Table 2. Dataset statistics

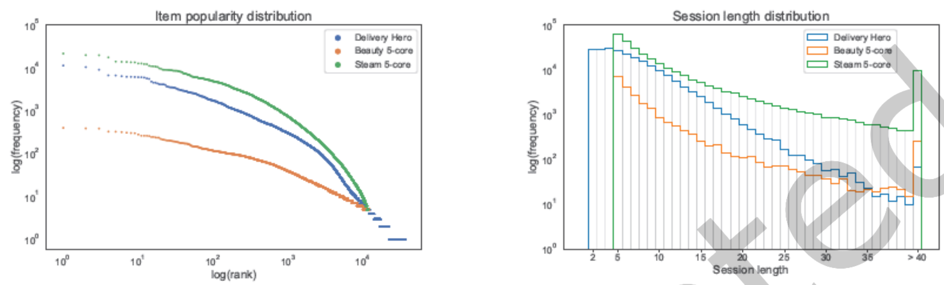

Fig. 11. Distribution of items ranked by popularity (left) and histogram of session length (right) for the datasets.

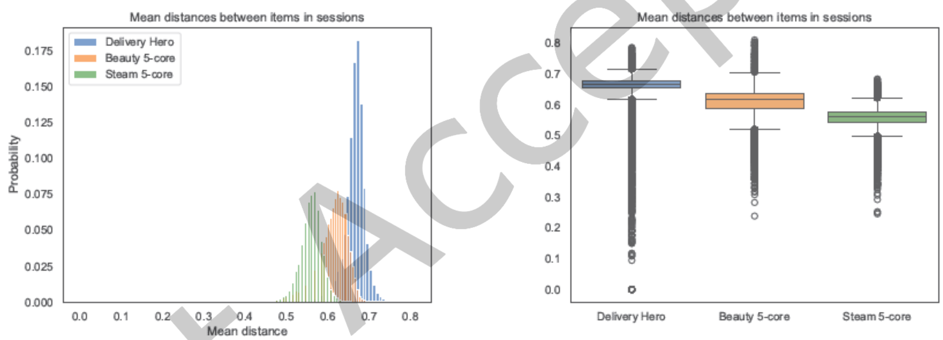

Fig. 12. Distribution of mean embedding distances for the datasets.

Figure 12 sheds more light on the differences between the datasets by focusing on the semantic affinity of the items in each session (i.e., on intra-session diversity). To find the semantic affinity of a session, we obtained the OpenAI embeddings of its items. These being vectors, we calculated the pairwise Euclidean distances of the embeddings of each session, and from the pairwise Euclidean distances we then calculated their mean. Greater distances (and their mean) correspond to smaller semantic affinity, or more diverse session items. In this sense, the mean of the pairwise Euclidean distances is a measure of diversity of a session. In the histograms on the left part of Figure 12, we see that the Delivery Hero dataset has higher and less dispersed mean distances. This is also shown in the corresponding boxplots on the right. Less diverse sessions mean that the recommendation problem has to deal with more similar items and thus may be easier to tackle by recommendation models.

To apply LDA on the data points as explained in Section 3.2, it is required to provide a set of input classes. Recall that the purpose of LDA is to project the data associated with k classes to $k - 1$ dimensions so that the projected data best fit those classes. The Beauty and Steam[21] datasets contain multiple categorical keywords per

[21]We use the genres.

item. We sort these keywords and concatenate them using underscore as a separator to obtain unique classes per item. The Steam dataset also contains tags, which are provided by users. We process these in the same manner as the genre keywords. The Delivery Hero dataset does not readily include metadata that we can use as classes, so we do not apply LDA to it.

To create a training and test set in a sound way, we first split a dataset containing sessions temporally such that all test sessions succeed train sessions in time. Then in the test set, we adopt the leave-one-out approach as in [34, 67], where all but the last interaction of each session represent the prompt (i.e., the ongoing session), and the last interaction serves as the ground truth.

7.1.2 Metrics. We use the standard ranking accuracy metrics NDCG, MRR, and HitRate at the usual cutoff lengths of 10 and 20. Furthermore, we consider the following *beyond-accuracy* metrics to obtain a more comprehensive picture of the performance of the different algorithms: catalog coverage, serendipity, and novelty. *Catalog coverage* represents the fraction of catalog items that appeared in at least one top-*n* recommendation list of the users in the test set [29]. *Serendipity* measures the ratio of correct recommendations per user that are not recommended by a popularity baseline [18]. *Novelty* computes the negative log of the relative item popularity, or self-information [88].

7.1.3 Models. We include both session-based algorithms of different families, GRU4Rec [23], and SKNN [30], as well as three state-of-the-art sequential models, BERT4Rec [67], SASRec [34], and S3-Rec [87]. Notably, S3-Rec introduces a custom pre-training and fine-tuning method that allows a meaningful comparison to our proposed LLM-based approaches, especially with respect to the LLM embedding initialization of neural models (Section 5) and finetuning of LLMs (Section 4). We also provide a comparison with a fine-tuned open-source Llama [68] LLM. In addition, we evaluate TALLRec [4], a state of the art LLM-based approach for sequential recommendations that structures the recommendations as instructions and tunes an LLM through an instruction-tuning process. Both Llama and TALLRec are closely related to our proposed LLM fine-tuning approaches (Section 4). Additionally, we initialize neural models with embeddings retrieved from a pre-trained BERT [13] model to compare against our proposed initialization approach using embeddings retrieved from OpenAI GPT and Google PaLM (Section 5). We refer to those models as BERT2BERT4Rec, BERT2SASRec, and BERT2GRU4Rec. Furthermore, we tested all variants of the SKNN nearest-neighbor method proposed in [53] and report the results of the best SKNN variant in this paper. The complete results can be found in the online material. Moreover, we include the four LLM-based approaches proposed in Sections 3, 4, 5, and 6. Finally, we include a popularity-based baseline (MostPopular) in the experiments.[22]

7.1.4 Hyperparameter Tuning. We systematically tuned all models (except LLMSeqPrompt, LlamaSeqPromptGenItem, S3-Rec, and TALLRec, which we discuss separately) on three validation folds with the Tree Parzen Estimator (TPE) sampler [6], and the total NDCG@20 across the folds as the optimization goal. We chose to optimize for 20 recommendations to provide enough room for the models to manifest their performance differences, although top-10 and top-20 metrics are usually closely related. In the beginning, we explore 40 random configurations, which helps avoid local minima, and then continue by evaluating the suggestions by the TPE sampler. We let the hyperparameter search run for 72 hours, but started multiple hyperparameter search experiments in parallel for each model (two for vanilla neural models and four for LLM2Sequential models) in order to minimize the duration of experiments. We also allow for early stopping of the hyperparameter search if the optimization objective has not been improved for 100 trials. Finally, we prune a trial if its objective value (i.e., NDCG@20) belongs to the bottom 20% of all tested trials up to that point, allowing us to accelerate the hyperparameter search process. We allow pruning only after ten different configurations of a model have been tried.

Overall, our chosen hyperparameter tuning period with early stopping and pruning enabled allows a wide search of the hyperparameter search space for the various models. Table 3 contains the number of different configurations

[22]The consideration of additional traditional baselines and most recently published sequential models is left for future work.

Model	Sec.	# Trials			Description
		BT	DH	ST	
LLMSeqSim	3	591	620	631	Semantic item recommendation model based on OpenAI or Google embeddings
LLMSeqPromptGenItem	4.1	16	14	4	Fine-tuned model to generate the next item
LLMSeqPromptGenList	4.2	5	-	-	Fine-tuned model to generate next-item recommendations
LLMSeqPromptClass	4.3	5	-	-	Fine-tuned model to classify items for next-item recommendations
LLMSeqPromptRank	4.4	5	-	-	Fine-tuned model to rank next-item recommendations
BERT4Rec	5.1	661	52	57	Bidirectional neural model based on the transformer architecture
SASRec	5.1	413	167	146	Unidirectional neural model based on the transformer architecture
GRU4Rec	5.1	622	168	164	Recurrent neural model based on Gated Recurrent Unit
SKNN	5.1	553	377	291	Neighborhood-based sequential model
LLM2BERT4Rec	5.2	1074	118	95	BERT4Rec initialized with LLM embeddings
LLM2SASRec	5.2	1280	272	226	SASRec initialized with LLM embeddings
LLM2GRU4Rec	5.2	1001	381	256	GRU4Rec initialized with LLM embeddings
SKNNEmb	5.2	767	218	128	SKNN utilizing LLM embeddings to compute session similarity
LLMSeqSim &Sequential	6.1	72	72	72	Hybrid of two models that perform well on popular & unpopular items respectively
LLMSeqSim &Popularity	6.2	60	60	60	Semantic item recommendation model enriched with popularity

Table 3. The recommendation models used in the experiments including references to the section where they are described, the number of trials carried out for each dataset (BeauTy, Delivery Hero, STeam) per model in the hyperparameter tuning process, and a short description of each model.

tried per model. Note that the LLM2Sequential neural models manifest a considerably higher number of trials than the respective vanilla neural models in order to account for the two additional hyperparameters they include, i.e. the embedding models and the dimensionality reduction methods, which result in a search space that is eight times larger.

The folds themselves are created by splitting the training data into validation-training and validation-test sets. We do this in a temporal fashion (same as for the train-test split) by splitting the training sessions into four bins, where each bin contains 25% of the training sessions in chronological order. The first fold uses the first bin as the validation-training data, and the second bin as the validation-test data. Then the second fold uses the first two bins as the validation-training data, and the third bin as the validation-test data. Similarly, the third fold uses the first three bins as the validation-training data, and the fourth bin as the validation-test data. By using less data in the first few folds (in comparison to cross-validation), we can more quickly evaluate a configuration as well as preserve the temporal split in our hyperparameter search process.

For the fine-tuning step of LLMSeqPrompt variants we stick to the default parameters, such as learning rate multiplier and number of epochs, recommended by the LLM providers. In the prediction step where we query a fine-tuned model to provide recommendations, we vary the temperature of the model. This hyperparameter, which in GPT takes values between 0 and 2, controls the randomness of the model's output. With a lower temperature the model produces more deterministic completions, while higher temperature makes the completions more random. Note that we treat the different LLMs used in the fine-tuning, i.e., OpenAI ada, OpenAI GPT, and Google PaLM, as hyperparameters and report the best performance per task variant. Thus, the number of trials for an LLMSeqPrompt variant in Table 3 represents the temperature configurations tried for all LLMs. The examined hyperparameter ranges and optimal values for each dataset are reported in the online material.

Finally, the hyperparameter tuning of the hybrids involves the following main parameters: embedding source, embedding dimensionality reduction, sequential model (for the LLMSeqSim &Sequential hybrid), and popularity threshold value.

7.1.5 Training and Evaluation of S3-Rec. The S3-Rec model is different in nature from other models, because it is a pre-training based approach. Nonetheless, we were interested how this particular model would fare in a comparison with the other models in our study. In terms of datasets, we included experiments only for the Beauty and Steam datasets, as these include metadata about item attributes that are required to pre-train S3-Rec. Using the original code of S3-Rec provided by the authors, we pre-trained S3-Rec for 10 epochs, fine-tuned it for 200 epochs with early stopping enabled. As for the other models in our comparison, we used the NDCG and the hit rate as evaluation measures. We note that we determined the metric values using the evaluation protocol from the original paper. S3-Rec's evaluation protocol involves using each session's second-to-last item for validation during fine-tuning. The last item of each session is used for testing. We note that the evaluation protocol (and code) used for the other models, which is more common in the literature, is slightly different and more challenging for the algorithms, as we hide entire sessions from the validation and training data. As a result, the results reported for the S3-Rec model are expected to be higher than what we would observe if the exact same protocol would be applied. As our results will show, however, the accuracy results obtained with S3-Rec fall behind the other models, even though more information was available than for the other models.

7.1.6 Training and Evaluation of TALLRec. For TALLRec, we present experiments only for the Beauty dataset because the model would require many weeks of execution on the Delivery Hero and Steam datasets. Regarding the model's training, we used the same preprocessing and train-validation-test split for Beauty as for LLMSeqPromptGenItem (Section 4.1). To make TALLRec comparable to our work, we converted TALLRec's binary classification problem into a recommendation generation problem. We also used the same system prompt, user prompt template, and assistant prompt template as in LLMSeqPromptGenItem but we adapted it to TALLRec's format.[23] We then trained TALLRec on an EC2 g5.12xlarge instance with 4x24GB A10G GPUs using the default hyperparameter setup provided by the authors.[24]

For the evaluation, we used the same protocol as for LLMSeqPromptGenItem. After resolving out-of-memory issues, we asked the model to produce top-20 recommendations for each test case by repeating each test prompt 20 times. Then, we mapped each recommendation of the model that is not in the product catalog to the closest catalog product via OpenAI embeddings. Finally, we computed the evaluation metrics that we report in the paper.

7.1.7 Fine-tuning and Evaluation of LlamaSeqPromptGenItem. We fine-tuned LlamaSeqPromptGenItem, specifically `meta-textgeneration-llama-3-8b`, on the Beauty and Steam datasets using the LoRA approach for five epochs with 0.0001 learning rate, 32 alpha, 0.05 dropout, and we set the r parameter to 8. In addition, we used 20% of the dataset as validation set. Resource constraints did not allow us to conduct experiments on the Delivery Hero dataset. On the same grounds, we fixed the temperature of the model to 1 and the *top_p* parameter to 0.25, which perform best in practice according to our experience.

7.1.8 Model Executions on Datasets. In principle, we ran all models on all datasets except for fine-tuning models due to budget and time limitations. Specifically, we applied LLMSeqPromptGenList, LLMSeqPromptClass, LLMSeqPromptRank, and TALLRec only on the Beauty dataset, and S3-Rec and LlamaSeqPromptGenItem only on the Beauty and Steam datasets. In addition, we only fine-tuned OpenAI `ada` on LLMSeqPromptGenItem because it is no longer available for fine-tuning.

Besides fine-tuning experiments, we executed again all experiments of the previous paper [20] and in fact managed to tune better a number of baseline models on the Delivery Hero dataset, of which BERT4Rec and SKNN exhibited high performance gains. The performance of SASRec and GRU4Rec also improved moderately. That said, we did observe a small decrease to the performance of SKNN and BERT4Rec in the Amazon Beauty dataset, while GRU4Rec's performance saw a small increase. We attribute the slightly lower performance to a potential modest instability caused by the size of the dataset and to the stochasticity of the hyperparameter search process.

[23] https://github.com/SAI990323/TALLRec/blob/main/finetune_rec.py#L301
[24] https://github.com/SAI990323/TALLRec/blob/main/shell/instruct_7B.sh

Model	Top@10						Top@20					
	nDCG	HR	MRR	CatCov	Seren	Novel	nDCG	HR	MRR	CatCov	Seren	Novel
LLM2SASRec	**0.045**	**0.083**	0.034	0.204	**0.081**	11.660	**0.054**	**0.118**	0.036	0.300	**0.112**	11.844
LLM2BERT4Rec	0.042	0.080	0.031	0.226	0.076	11.645	0.052	**0.118**	0.034	0.328	0.111	11.823
LLMSeqSim &Sequential	0.040	0.074	0.029	0.349	0.072	11.729	0.050	0.116	0.032	0.514	0.110	11.892
LLMSeqSim	0.043	0.068	**0.036**	**0.761**	0.068	**13.755**	0.049	0.090	**0.037**	**0.889**	0.090	13.796
LLMSeqSim &Popularity	0.042	0.065	0.035	0.464	0.065	13.067	0.049	0.093	0.037	0.496	0.092	13.098
SKNNEmb	0.039	0.070	0.030	0.476	0.068	11.855	0.047	0.101	0.032	0.689	0.096	12.090
V_SKNN	0.036	0.063	0.028	0.399	0.059	11.211	0.043	0.090	0.030	0.613	0.083	11.481
LLMSeqPromptRank	0.037	0.068	0.027	0.744	0.068	13.727	0.042	0.089	0.028	0.887	0.089	13.799
LLM2GRU4Rec	0.034	0.065	0.024	0.252	0.062	11.732	0.042	0.097	0.026	0.365	0.091	11.902
BERT2GRU4Rec	0.033	0.060	0.025	0.147	0.056	11.200	0.042	0.095	0.028	0.207	0.087	11.405
BERT4Rec	0.033	0.064	0.023	0.108	0.061	11.789	0.041	0.096	0.025	0.161	0.089	11.937
BERT2SASRec	0.030	0.055	0.022	0.143	0.050	11.096	0.038	0.087	0.024	0.207	0.077	11.307
LLMSeqPromptGenItem	0.032	0.054	0.025	0.548	0.054	13.140	0.037	0.075	0.027	0.728	0.074	13.385
BERT2BERT4Rec	0.024	0.049	0.017	0.084	0.046	11.219	0.033	0.085	0.019	0.132	0.077	11.442
GRU4Rec	0.026	0.051	0.018	0.103	0.046	11.349	0.033	0.080	0.020	0.156	0.073	11.494
SASRec	0.023	0.047	0.016	0.071	0.043	11.081	0.033	0.085	0.018	0.112	0.078	11.316
LLMSeqPromptGenList	0.029	0.053	0.022	0.703	0.053	13.587	0.033	0.069	0.023	0.831	0.069	13.652
LlamaSeqPromptGenItem	0.023	0.041	0.018	0.499	0.041	13.366	0.028	0.061	0.019	0.669	0.060	13.451
S3-Rec	0.020	0.041	–	–	–	–	0.027	0.067	–	–	–	–
TALLRec	0.013	0.024	0.009	0.585	0.024	13.754	0.015	0.035	0.01	0.747	0.034	**13.824**
LLMSeqPromptClass	0.006	0.012	0.005	0.016	0.010	11.041	0.008	0.017	0.005	0.016	0.012	11.072
MostPopular	0.005	0.010	0.003	0.001	0.001	9.187	0.006	0.018	0.003	0.002	0.001	9.408

Table 4. Evaluation results for the Amazon Beauty dataset

7.2 Results and Discussion

Tables 4, 5, and 6 depict the results obtained for the Amazon Beauty, Delivery Hero, and Steam datasets on the hidden test set, respectively. The rows in each table are sorted in descending order according to the NDCG values at 20. We discuss the experimental results in terms of accuracy and beyond accuracy metrics in Sections 7.2.1 and 7.2.2, respectively.

7.2.1 Accuracy metrics. The key observation of the experimental results is that *LLM embeddings boost recommendation model performance in terms of accuracy metrics across models and datasets*. Specifically, sequential models enhanced with LLM embeddings score the best performance across all three datasets demonstrating the superiority of LLM embeddings for the models' learning and training routines. In particular, LLM2SASRec and LLM2BERT4Rec maintain consistently the best performance and alternate on the top spots marking around 45% average improvements in NDCG@20 over their vanilla counterparts on Beauty, and 9% on the Delivery Hero dataset. On Steam, the performance boost is steady but marginal across all sequential models. One potential reason for that could be the quality of embeddings for this particular dataset. As a gaming dataset, Steam features game titles that are semantically non-discriminative. Although we enriched the title with tags that accompany the games, such as `action game` and `multi-player`, these are commonly shared across the games and, therefore, their contribution is ambivalent. Finally, LLM2GRU4Rec achieves a 20% increase in NDCG@20 on Beauty over GRU4Rec, but only marginal improvements in the other two datasets.

In line with our observations regarding LLM embeddings, BERT embeddings also tend to increase the performance of neural recommendation models, but not consistently for all datasets and all models. In most of the cases, LLM embeddings significantly outperform BERT embeddings except for one case, for the DH dataset, where BERT embeddings elevated GRU4Rec's performance significantly stronger than LLM embeddings. In summary, the experiments reveal a real potential for leveraging the semantics of a dataset's item descriptions to uplift

Model	Top@10						Top@20					
	nDCG	HR	MRR	CatCov	Seren	Novel	nDCG	HR	MRR	CatCov	Seren	Novel
LLM2BERT4Rec	**0.101**	**0.180**	0.077	0.240	**0.151**	10.832	**0.120**	**0.253**	**0.082**	0.301	**0.198**	11.028
LLMSeqSim & Sequential	0.099	0.175	0.075	0.276	0.146	10.823	0.117	0.247	0.080	0.354	0.191	11.021
BERT2BERT4Rec	0.098	0.174	0.075	0.208	0.144	10.600	0.116	0.244	0.080	0.281	0.187	10.802
BERT2GRU4Rec	0.098	0.175	0.075	0.260	0.147	10.766	0.116	0.245	0.080	0.333	0.189	10.963
SKNNEmb	0.098	0.166	**0.077**	0.364	0.147	10.938	0.114	0.228	0.082	0.434	0.184	11.076
SKNN	0.098	0.168	0.077	0.361	0.138	10.547	0.113	0.228	0.081	0.438	0.171	10.742
LLM2SASRec	0.096	0.171	0.073	0.275	0.142	10.782	0.113	0.237	0.078	0.358	0.181	10.977
BERT4Rec	0.095	0.169	0.0073	0.278	0.140	10.904	0.112	0.237	0.077	0.364	0.182	11.089
SASRec	0.087	0.153	0.067	0.153	0.126	10.784	0.103	0.217	0.071	0.197	0.163	10.997
LLM2GRU4Rec	0.087	0.154	0.066	0.244	0.127	10.917	0.103	0.218	0.071	0.314	0.163	11.109
GRU4Rec	0.086	0.153	0.066	0.232	0.126	10.869	0.103	0.217	0.070	0.304	0.164	11.058
BERT2SASRec	0.082	0.149	0.061	0.106	0.115	10.319	0.098	0.212	0.066	0.150	0.152	10.599
LLMSeqPromptGenItem	0.063	0.116	0.047	0.400	0.107	12.048	0.070	0.144	0.049	0.611	0.123	13.788
LLMSeqSim & Popularity	0.051	0.091	0.039	0.270	0.090	13.869	0.060	0.126	0.042	0.369	0.122	13.973
LLMSeqSim	0.040	0.073	0.030	**0.684**	0.072	**16.216**	0.047	0.102	0.032	**0.818**	0.099	**16.454**
MostPopular	0.024	0.049	0.017	0.000	0.000	7.518	0.032	0.079	0.019	0.001	0.000	7.836

Table 5. Evaluation results for the Delivery Hero dataset

Model	Top@10						Top@20					
	nDCG	HR	MRR	CatCov	Seren	Novel	nDCG	HR	MRR	CatCov	Seren	Novel
LLM2SASRec	**0.068**	**0.129**	**0.050**	0.258	**0.112**	9.885	**0.085**	**0.197**	**0.054**	0.338	**0.150**	10.026
LLM2BERT4Rec	0.068	0.128	0.049	0.266	0.110	9.662	0.084	0.194	0.054	0.345	0.143	9.862
BERT2BERT4Rec	0.067	0.127	0.049	0.238	0.109	9.765	0.083	0.193	0.053	0.313	0.144	9.955
BERT4Rec	0.066	0.126	0.047	0.242	0.106	9.583	0.083	0.194	0.052	0.317	0.141	9.813
LLMSeqSim & Sequential	0.066	0.127	0.048	0.410	0.109	9.691	0.083	0.192	0.053	0.548	0.142	9.834
LLM2GRU4Rec	0.066	0.124	0.048	0.319	0.108	9.907	0.082	0.188	0.052	0.414	0.141	10.079
GRU4Rec	0.065	0.123	0.048	0.389	0.106	9.976	0.081	0.188	0.052	0.482	0.141	10.161
SASRec	0.065	0.123	0.047	0.168	0.104	9.647	0.081	0.187	0.052	0.232	0.138	9.869
BERT2GRU4Rec	0.064	0.122	0.046	0.183	0.105	9.674	0.080	0.186	0.051	0.242	0.137	9.857
BERT2SASRec	0.060	0.116	0.043	0.106	0.099	9.527	0.076	0.178	0.048	0.142	0.129	9.738
S3-Rec	0.057	0.111	–	–	–	–	0.072	0.173	–	–	–	–
SKNNEmb	0.051	0.099	0.037	0.243	0.072	8.789	0.064	0.151	0.041	0.353	0.093	9.053
S_SKNN	0.048	0.097	0.034	0.178	0.066	8.439	0.062	0.150	0.038	0.263	0.084	8.684
LLMSeqPromptGenItem	0.030	0.062	0.021	0.457	0.057	10.595	0.037	0.089	0.023	0.686	0.068	11.695
MostPopular	0.022	0.042	0.016	0.001	0.000	7.638	0.030	0.075	0.018	0.002	0.001	7.951
LLMSeqSim & Popularity	0.024	0.044	0.017	0.328	0.041	11.349	0.030	0.068	0.019	0.483	0.059	11.397
LLMSeqSim	0.023	0.044	0.017	**0.656**	0.041	**11.871**	0.029	0.067	0.019	**0.786**	0.058	**11.950**
LlamaSeqPromptGenItem	0.014	0.025	0.011	0.301	0.022	11.473	0.019	0.045	0.012	0.424	0.031	11.782

Table 6. Evaluation results for the Steam dataset

the performance of the recommendation models. This potential however depends on the dataset and different models capture the semantics in different ways and to different extents. Most of the time, however, embedding initialization leads to significant performance gains especially when LLM embeddings are used.

TALLRec, S3-Rec, and LlamaSeqPromptGenItem performed average in the evaluation. Specifically, they trail behind the neural models for the Steam dataset and also behind the LLM generation-based approaches for the Beauty dataset.

Furthermore, SKNNEmb outperforms SKNN on Beauty by 9% at NDCG@20 and 12% at HR@20. However, in the other datasets, SKNNEmb only slightly improves over SKNN. A notable observation is that SKNNEmb takes the third place across all models in the Delivery Hero dataset and exhibits the best MRR@10 metric. This result is in line with previous performance comparisons of sequential recommendation models. In [53], for example, it was observed that the rather simple Sequential Rules (SR) method can lead to highly competitive results in terms of the MRR for some datasets, but is not always competitive in terms of Hit Rate. A strong performance on the MRR measure means that an algorithm is able to frequently position one positive item in the recommendation lists, e.g., due to a popularity bias of the algorithm as in the case of the SR method. Overall, while most accuracy measures are typically highly correlated [3], some algorithms often perform particularly well on certain metrics, depending on specific dataset characteristics. In this respect, our results thus exhibit phenomena that are similar to those reported in the literature.

The LLMSeqSim &Popularity hybrid closely trailed the top models with a small uplift in HR@20 compared to LLMSeqSim on Beauty. However, for the other two datasets, it was not competitive, trailing even the popularity baseline on Steam. That said, besides the tie in NDCG@20 on Beauty, the LLMSeqSim &Popularity hybrid was consistently better than LLMSeqSim on the other datasets and even achieved a 27% increase on the Delivery Hero dataset, showing that the consideration of popularity can lead to more attractive recommendations. Similar to the LLMSeqSim &Popularity hybrid, LLMSeqSim only exhibited high performance on the Beauty dataset. In fact, LLMSeqSim topped the MRR metric, and the LLMSeqSim &Popularity hybrid almost matched the performance. Finally, our expectation of high performance on the Steam dataset due to the even higher homogeneity of items in a session than the Beauty dataset according to Figure 12, did not materialize.

Of the fine-tuned models, LLMSeqPromptGenItem is the most noteworthy since it surpasses the performance of GRU4Rec and SASRec on Beauty, while LLMSeqPromptGenList matches the performance of GRU4Rec and SASRec on the same dataset. Note that both models have been fine-tuned on OpenAI GPT in a single epoch, which was enough for the validation loss to converge. We highlight a number of notable aspects of the fine-tuned models in Section 7.3. In the remaining datasets, LLMSeqPromptGenItem is by far inferior to mainstream recommendation models, but also markedly better than simplistic baselines like MostPopular. Interestingly, the implicit ranking of recommendations in LLMSeqPromptGenItem, where we repeatedly ask the model to provide single-item recommendations until we have top-k, was more effective than letting LLMSeqPromptGenList return a ranked list of top-k recommendations. This indication is in line with the performance of LLMSeqPromptRank, which although positioned seemingly high on the performance board, it actually performed worse than LLMSeqSim, which produced the candidate recommendations that LLMSeqPromptRank reranked. Finally, LLMSeqPromptClass performed almost on par with the popularity baseline, showing little to no potential. On the other hand, we barely experimented with the number of clusters, which is a crucial performance factor for this task specification.

7.2.2 Beyond accuracy metrics. We make the following observations for coverage, serendipity, and novelty. The LLMSeqSim model consistently leads to the best catalog coverage and novelty except for Novelty@20 in the Beauty dataset where TALLRec performs better. The performance of LLMSeqSim in terms of catalog coverage and novelty is not too surprising, given the nature of the approach, which is solely based on embedding similarities. Unlike other methods that use collaborative signals, i.e., past user-item interactions, the general popularity of an item in terms of the amount of observed past interactions does not play a role in LLMSeqSim, neither directly nor implicitly. Thus, the model has no tendency to concentrate the recommendations on a certain subset of (popular) items.[25] Finally, the serendipity results are aligned with the accuracy measures across the datasets. This generally confirms the value of personalizing the recommendations to individual user preferences, compared to recommending mostly popular items to everyone.[26]

[25]We recall that the used novelty measure is based on the popularity of the items in the recommendations.
[26]We iterate that our serendipity measure counts the fraction of correctly recommended items that would not be recommended by a popularity-based approach.

Interestingly, LLM2SASRec uplifts all beyond-accuracy metrics compared to SASRec, especially catalog coverage and serendipity where it almost doubles catalog coverage on average and contributes an average of 21% improvement on serendipity across datasets. Also notable is the performance of the LLMSeqSim &Sequential hybrid, which is composed of LLMSeqSim and LLM2BERT4Rec, in terms of catalog coverage. The LLMSeqSim &Sequential contributes a 45% average improvement in catalog coverage compared to LLM2BERT4Rec, while sacrificing only a small margin of performance in terms of accuracy metrics. That said, we note that catalog coverage is a coarse metric that counts an item in the coverage of the item space given just a single occurrence of the item across all top-k recommendations on the test set. Still, the improvement is substantial.

7.3 Observations from Hyperparameter Tuning

Given that our proposed approaches embody a variety of novel technical variations, we devote this section to the relationship between parameters and optimal model performance as revealed by the hyperparameter search process for LLMSeqSim (Section 7.3.1), LLM2Sequential (Section 7.3.2), LLMSeqPrompt (Section 7.3.3), and hybrid models (Section 7.3.4). Table 3 depicts the number of hyperparameter search trials for each model and dataset.

7.3.1 Semantic Item Recommendations via LLM Embeddings (LLMSeqSim). There are four main observations that stand out from the tuning of LLMSeqSim across datasets. First, the use of OpenAI embeddings leads to a clear performance gain compared to embeddings obtained from Google. Second, dimensionality reduction increases accuracy and, third, the application of PCA always results in the best configuration. Finally, while reducing the dimensions of the embeddings always helps, the higher the target number of dimensions is, the higher the accuracy.

7.3.2 LLM-enhanced Sequential Models (LLM2Sequential). For the sequential models enhanced with LLM embeddings, the choice of LLM embedding model per se only leads to marginal performance differences. In fact, both OpenAI and Google embeddings are found in best model configurations depending on the dataset and model. On the other hand, dimensionality reduction consistently contributes to the best model configurations across datasets. Usually, PCA is the optimal reduction method, but LDA and random projection also appear in certain cases. Finally, neural sequential models benefit from low learning and drop rates, while SKNNEmb leverages different types of decay for training and prompt sessions to optimize its performance. SKNNEmb is agnostic to the choice of similarity type, likely due to the normalization of embedding vectors.

7.3.3 Fine-tuned Models (LLMSeqPrompt). A fundamental challenge of fine-tuning regards whether the fine-tuned model will comply with the form of the task at hand. All models that we fine-tuned across datasets closely aligned the form of their responses with the completions provided to them during the fine-tuning process according to the task specifications that we described in Section 4.

We iterate that we applied all four LLMSeqPrompt fine-tuning task variants on Beauty and only LLMSeqPrompt-GenItem on the Delivery Hero and Steam datasets due to budget and time limitations. Section 7.1 includes all information regarding the fine-tuning experiments. In all setups where both GPT and PaLM are fine-tuned, i.e., for all variants in the Beauty, Delivery Hero, and Steam datasets, GPT outperforms PaLM demonstrating an average uplift of 78% in terms of NDCG@20. We include the corresponding results in the online material. For reference, these observations appear to contradict the Chain of Thought Hub [15], which evaluates the reasoning performance of LLMs, where PaLM 2 exhibits noticeably better performance in a variety of reasoning tasks compared to GPT `3.5 turbo`. However, PaLM 2 refers to the `unicorn` version, while we used the `bison` one, which is smaller than `unicorn`. On the other hand, GPT `3.5 turbo` performs substantially better than PaLM `bison` in the LMSys Chatbot Arena leaderboard,[27] which evaluates the user dialogue with an LLM based on user votes.

[27] https://leaderboard.lmsys.org

Dataset	Task	Model	HR@20	NDCG@20	# Uniq. Reco.	Hallucin.	Emb. Sim.
Beauty	LLMSeqPromptGenItem	GPT base	0.029	0.013	6553	72.38%	0.919
		GPT fntd	0.077	0.038	20056	36.11%	0.960
		PaLM base	0.029	0.015	19330	78.63%	0.944
		PaLM fntd	0.043	0.020	21103	56.23%	0.950
	LLMSeqPromptGenList	GPT base	0.021	0.008	28372	82.20%	0.935
		GPT fntd	0.069	0.033	47966	43.35%	0.953
		PaLM base	0.022	0.009	40019	78.85%	0.940
		PaLM fntd	0.042	0.020	32643	21.24%	0.969
DH	LLMSeqPromptGenItem	GPT base	0.022	0.009	41951	67.39%	0.910
		GPT fntd	0.111	0.057	55301	13.79%	0.967
		PaLM base	0.045	0.027	50416	70.85%	0.950
		PaLM fntd	0.050	0.029	43698	37.87%	0.961
Steam	LLMSeqPromptGenItem	GPT base	0.017	0.010	2534	8.41%	0.858
		GPT fntd	0.059	0.032	5701	0.36%	0.889
		PaLM base	0.019	0.013	13303	77.05%	0.870
		PaLM fntd	0.025	0.014	382446	58.87%	0.867

Table 7. Comparison between base and fine-tuned models across datasets for LLMSeqPromptGenItem and LLMSeqPromptGenList in terms of HR@20, NDCG@20, number of unique recommendations, percentage of hallucinations, and embedding similarity between hallucinations and closest match.

In addition, the LLMSeqPromptGenItem task has been used to fine-tune both `ada` and `GPT 3.5 turbo` OpenAI models on the Beauty and Delivery Hero datasets. The results are inconclusive. On the one hand, we observe a notable performance increase of 23% in terms of NDCG@20 in favor of `GPT` on Beauty, but, surprisingly, on Delivery Hero `ada` performed significantly better than `GPT` for the same task achieving 18% higher NDCG@20. Given that the comparison scope is narrow, we can only conclude that the choice of fine-tune model depends on the dataset and that the evolution of models does not necessarily lead to higher performance.

With respect to the *temperature* and *top_p* hyperparameters, we observe that in all setups a high temperature results in lower performance as it allows for more creativity at the cost of item hallucinations. For LLMSeqPromptGenItem and LLMSeqPromptGenList a low but non-zero *temperature* leads to the best performance whereas LLMSeqPromptClass and LLMSeqPromptRank perform best with *temperature* set to zero. A possible explanation for this is that the latter variants do not require any creativity because the recommendation options are contained in the provided prompt.

Finally, to explore the learning capabilities of fine-tuned models, we compare their performance to the corresponding base models. We observe that in all setups the fine-tuned models significantly outperform their non fine-tuned counterparts. For the LLMSeqPromptGenItem and LLMSeqPromptGenList variants, the model generates recommendations that are not part of the prompt. For those, the base model has a much higher rate of hallucinations compared to fine-tuned models as Table 7 shows. Additionally, the hallucinations produced by the fine-tuned models exhibit a higher embedding similarity to the catalog items compared to the base model. These observations indicate that fine-tuned models recall and prioritize concepts that they processed during their fine-tuning compared to the base model, which is not given the same opportunity. Therefore, the base model has to provide recommendations solely based on the information in the prompt.

For the variants LLMSeqPromptClass and LLMSeqPromptRank, the recommendation options are explicitly part of the context such that the model only has to select and/or re-rank the given item names. Thus, hallucinations are uncommon in those variants. Still, the performance advantage of the fine-tuned models over the base models persists. This indicates that the fine-tuned models learn the underlying classification and ranking tasks better. This is an expected finding that is widely acknowledged by literature and practice [1, 68].

All in all, the fine-tuned models perform better for each recommendation task with fewer hallucinations and closer semantic proximity to the actual options compared to the base models. This finding clearly shows that fine-tuning can enable a model to learn not only how to perform a task, but also concepts of a domain to some extent. Thus, it contradicts existing findings and common belief that fine-tuning is only applicable for learning form (instructions) rather than facts [5, 8, 33, 37, 38, 40, 55, 58, 59, 71].

7.3.4 Hybrids. In both hybrids (LLMSeqSim &Sequential, LLMSeqSim &Popularity) the LLMSeqSim model based on the OpenAI embeddings performs better, while for the LLM2Sequential model performance depends on the dataset: the OpenAI embeddings dominate on the Delivery Hero and Steam datasets, and the Google embeddings on the Beauty dataset. With regard to dimensionality reduction, it mainly improves performance compared to the original embedding size.

With respect to the popularity threshold, for LLMSeqSim &Sequential we note that increasing the cutoff point seems to decrease performance. Thus, the hybrid appears to favor LLM2Sequential over LLMSeqSim since a lower threshold results in more recommendations from the former, and less from the latter. In contrast, in LLMSeqSim &Popularity increasing the popularity threshold value appears to increase performance, especially for the Delivery Hero and Steam datasets.

8 Challenges, Limitations, & Conclusions

While large language models offer a variety of new opportunities to build highly effective (sequential) recommender systems, their use, both in industry and academic research, may come with a number of challenges. One central issue in that context is that the most powerful LLMs today are under the control of large organizations and are primarily accessed via APIs. These LLMs are thus black boxes to their users. Moreover, and maybe more importantly, the behavior of these LLMs can change over time and their performance can vary. In an industrial setting, this may represent a major challenge from a quality assurance perspective. In academic settings, the varying behavior of the models can make the exact reproducibility of reported findings almost impossible.

Furthermore, depending on the application use case, an LLM fine-tuning step may be required to achieve satisfactory recommendation performance. In cases where the organization that hosts an LLM provides an API for fine-tuning, this may lead to data protection and privacy issues, as potentially sensitive company-internal data may have to be uploaded for the fine-tuning step. Finally, depending on the particular way an LLM is leveraged to support (sequential) recommendation processes, significant costs may arise with the use of an LLM. Real-world item catalogs can be huge, and fine-tuning may furthermore be required frequently, e.g., in case of use cases with highly dynamic item catalogs. In this context, we recall that for some datasets it may not be sufficient to feed *item names* into the LLM, as they may carry limited semantic information, as in the case of the Steam dataset. In such situations, potentially available longer item descriptions may be used, but this approach may further add to the cost of the fine-tuning process.

In terms of research limitations, our work so far is based on three datasets from two domains (e-commerce, games). The datasets used in our research are however markedly different in terms of various characteristics (e.g., sparsity, size, available metadata, recurrence patterns, frequency of use) and in terms of their nature (e.g., one long sequence of product reviews per user for Amazon Beauty vs. a collection of supermarket purchases per user, each consisting of a sequence of products, in the Delivery Hero dataset). Therefore, we are confident that our findings generalize beyond the conducted experiments. Nonetheless, further experiments are needed to validate the benefits of leveraging LLMs for sequential recommendation in other domains, e.g., in music or news recommendation. In addition, in our current work we have not specifically analyzed the potential of LLM-enhanced sequential recommendations for cold-start situations, where very limited information is available about the preferences of individual users. Also, as another area for future works, it would be interesting to study the use of LLMs for cross-domain recommendation settings. Finally, future work might include the consideration of alternative LLMs or larger versions of existing LLMs, as these models continue to become more powerful and larger in terms of their parameters. One important research question in that context is if larger model sizes will have a significant impact on recommendation accuracy for the different approaches studied in this paper, see also the analyses in [82].

To conclude, Large Language Models and AI assistants like ChatGPT have exhibited disruptive effects in various domains in the past few years. In this work, we have conducted an extensive analysis on the potential benefits of LLMs for the highly relevant class of sequential recommendation problems. Specifically, we have designed, investigated, and systematically assessed different ways of leveraging LLMs in the recommendation process. Our findings show that existing sequential models can particularly benefit from the semantic knowledge encoded in LLMs, leading to substantial accuracy increases compared to previous models. Furthermore, depending on the application, sometimes the LLM-enhanced retrieval of semantically similar items turned out to be highly effective. Our future work includes both the analysis of the proposed methods for additional domains and datasets and the design of alternative ways of building recommendation approaches based on LLMs.

References

[1] OpenAI . 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[2] Mingxiao An, Fangzhao Wu, Chuhan Wu, Kun Zhang, Zheng Liu, and Xing Xie. 2019. Neural News Recommendation with Long- and Short-term User Representations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 336–345.

[3] Vito Walter Anelli, Alejandro Bellogín, Tommaso Di Noia, Dietmar Jannach, and Claudio Pomo. 2022. Top-N Recommendation Algorithms: A Quest for the State-of-the-Art. In *Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization (UMAP '22)*. Association for Computing Machinery, 121–131.

[4] Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023. TALLRec: An Effective and Efficient Tuning Framework to Align Large Language Model with Recommendation. arXiv:2305.00447 [cs.IR]

[5] Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2023. The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A". arXiv:2309.12288 [cs.CL]

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (Eds.), Vol. 24. Curran Associates, Inc.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.

[8] Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. 2020. Recall and Learn: Fine-tuning Deep Pretrained Language Models with Less Forgetting. arXiv:2004.12651 [cs.CL]

[9] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. 714–722.

[10] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1724–1734.

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. Association for Computing Machinery, 191–198.

[12] Sunhao Dai, Ninglu Shao, Haiyuan Zhao, Weijie Yu, Zihua Si, Chen Xu, Zhongxiang Sun, Xiao Zhang, and Jun Xu. 2023. Uncovering ChatGPT's Capabilities in Recommender Systems. In *Proceedings of the 17th ACM Conference on Recommender Systems (RecSys '23)*. ACM.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, NAACL-HLT, Volume 1 (Long and Short Papers). Association for Computational Linguistics, 4171–4186.

[14] Hao Ding, Anoop Deoras, Yuyang (Bernie) Wang, and Hao Wang. 2022. Zero shot recommender systems. In *ICLR 2022 Workshop on Deep Generative Models for Highly Structured Data*.

[15] Yao Fu, Litu Ou, Mingyu Chen, Yuhao Wan, Hao Peng, and Tushar Khot. 2023. Chain-of-Thought Hub: A Continuous Effort to Measure Large Language Models' Reasoning Performance. arXiv:2305.17306 [cs.CL]

[16] Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. 2023. Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System. arXiv:2303.14524 [cs.IR]

[17] Florent Garcin, Christos Dimitrakakis, and Boi Faltings. 2013. Personalized News Recommendation with Context Trees. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. Association for Computing Machinery, 105–112.

[18] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. 2010. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. Association for Computing Machinery, 257–260.

[19] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5). In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys '22)*. Association for Computing Machinery, 299–315.

[20] 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 (RecSys '23)*. 1096–1102.

[21] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proceedings of the 25th International Conference on World Wide Web (WWW '16)*. International World Wide Web Conferences Steering Committee, 507–517.

[22] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. 2023. TabLLM: Few-shot Classification of Tabular Data with Large Language Models. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 206)*. PMLR, 5549–5581.

[23] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR*.

[24] Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. 2023. Learning Vector-Quantized Item Representation for Transferable Sequential Recommenders. In *Proceedings of the ACM Web Conference 2023 (WWW '23)*. Association for Computing Machinery, 1162–1171.

[25] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. 2022. Towards Universal Sequence Representation Learning for Recommender Systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22)*. Association for Computing Machinery, 585–593.

[26] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2023. Large Language Models are Zero-Shot Rankers for Recommender Systems. arXiv:2305.08845 [cs.IR]

[27] Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. 2023. How to Index Item IDs for Recommendation Foundation Models. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*. ACM.

[28] Dietmar Jannach, Lukas Lerche, and Michael Jugovac. 2015. Adaptation and Evaluation of Recommendations for Short-term Shopping Goals. In *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys '15)*. 211–218.

[29] Dietmar Jannach, Lukas Lerche, Iman Kamehkhosh, and Michael Jugovac. 2015. What recommenders recommend: an analysis of recommendation biases and possible countermeasures. *User Modeling and User-Adapted Interaction* 25, 5 (2015), 427–491.

[30] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks Meet the Neighborhood for Session-Based Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. Association for Computing Machinery, 306–310.

[31] Dietmar Jannach, Bamshad Mobasher, and Shlomo Berkovsky. 2020. Research directions in session-based and sequential recommendation. *User Modeling and User-Adapted Interaction* 30, 4 (2020), 609–616.

[32] William B Johnson. 1984. Extensions of Lipshitz mapping into Hilbert space. In *Conference modern analysis and probability, 1984*. 189–206.

[33] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large Language Models Struggle to Learn Long-Tail Knowledge. arXiv:2211.08411 [cs.CL]

[34] Wang-Cheng Kang and Julian J. McAuley. 2018. Self-Attentive Sequential Recommendation. In *Proceedings of the* 18^{th} *IEEE International Conference on Data Mining (ICDM 2018)*. 197–206.

[35] Wang-Cheng Kang, Jianmo Ni, Nikhil Mehta, Maheswaran Sathiamoorthy, Lichan Hong, Ed Chi, and Derek Zhiyuan Cheng. 2023. Do LLMs Understand User Preferences? Evaluating LLMs On User Rating Prediction. arXiv:2305.06474 [cs.IR]

[36] Barrie Kersbergen, Olivier Sprangers, and Sebastian Schelter. 2022. Serenade - Low-Latency Session-Based Recommendation in e-Commerce at Scale. *SIGMOD* (2022).

[37] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (March 2017), 3521–3526. https://doi.org/10.1073/pnas.1611835114

[38] Yuval Kirstain, Patrick Lewis, Sebastian Riedel, and Omer Levy. 2022. A Few More Examples May Be Worth Billions of Parameters. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 1017–1029.

[39] Siqi Lai, Erli Meng, Fan Zhang, Chenliang Li, Bin Wang, and Aixin Sun. 2022. An Attribute-Driven Mirror Graph Network for Session-based Recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1674–1683.

[40] Patrick Lewis, Pontus Stenetorp, and Sebastian Riedel. 2021. Question and Answer Test-Train Overlap in Open-Domain Question Answering Datasets. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (Eds.). Association for Computational Linguistics, Online, 1000–1008. https://doi.org/10.18653/v1/2021.eacl-main.86

[41] Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. 2023. Text Is All You Need: Learning Language Representations for Sequential Recommendation. In *KDD '23: The 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[42] Jinming Li, Wentao Zhang, Tian Wang, Guanglei Xiong, Alan Lu, and Gerard Medioni. 2023. GPT4Rec: A Generative Framework for Personalized Recommendation and User Interests Interpretation. arXiv:2304.03879 [cs.IR]

[43] Ruyu Li, Wenhao Deng, Yu Cheng, Zheng Yuan, Jiaqi Zhang, and Fajie Yuan. 2023. Exploring the Upper Limits of Text-Based Collaborative Filtering Using Large Language Models: Discoveries and Insights. arXiv:2305.11700 [cs.IR]

[44] Xinyi Li, Yongfeng Zhang, and Edward C. Malthouse. 2023. PBNR: Prompt-based News Recommender System. arXiv:2304.07862 [cs.IR]

[45] Xinyi Li, Yongfeng Zhang, and Edward C. Malthouse. 2023. A Preliminary Study of ChatGPT on News Recommendation: Personalization, Provider Fairness, Fake News. arXiv:2306.10702 [cs.IR]

[46] Zhiyu Li, Yanfang Chen, Xuan Zhang, and Xun Liang. 2023. BookGPT: A General Framework for Book Recommendation Empowered by Large Language Model. *Electronics* 12, 22 (2023).

[47] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2015. Personalized tour recommendation based on user interests and points of interest visit durations. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, 1778–1784.

[48] Jianghao Lin, Xinyi Dai, Yunjia Xi, Weiwen Liu, Bo Chen, Xiangyang Li, Chenxu Zhu, Huifeng Guo, Yong Yu, Ruiming Tang, and Weinan Zhang. 2023. How Can Recommender Systems Benefit from Large Language Models: A Survey. arXiv:2306.05817 [cs.IR]

[49] Min Lin, Qiang Chen, and Shuicheng Yan. 2014. Network In Network. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). arXiv:1312.4400

[50] Junling Liu, Chao Liu, Renjie Lv, Kang Zhou, and Yan Zhang. 2023. Is ChatGPT a Good Recommender? A Preliminary Study. arXiv:2304.10149 [cs.IR]

[51] Qijiong Liu, Nuo Chen, Tetsuya Sakai, and Xiao-Ming Wu. 2023. ONCE: Boosting Content-based Recommendation with Both Open- and Closed-source Large Language Models. arXiv:2305.06566 [cs.IR]

[52] Yiding Liu, Weixue Lu, Suqi Cheng, Daiting Shi, Shuaiqiang Wang, Zhicong Cheng, and Dawei Yin. 2021. Pre-trained Language Model for Web-scale Retrieval in Baidu Search. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 3365–3375.

[53] Malte Ludewig and Dietmar Jannach. 2018. Evaluation of Session-based Recommendation Algorithms. *User-Modeling and User-Adapted Interaction* 28, 4–5 (2018), 331–390.

[54] Malte Ludewig, Sara Latifi, Noemi Mauro, and Dietmar Jannach. 2021. Empirical Analysis of Session-Based Recommendation Algorithms. *User Modeling and User-Adapted Interaction* 31, 1 (2021), 149–181.

[55] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning. arXiv:2308.08747 [cs.CL]

[56] Zhiming Mao, Huimin Wang, Yiming Du, and Kam-Fai Wong. 2023. UniTRec: A Unified Text-to-Text Transformer and Joint Contrastive Learning Framework for Text-based Recommendation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 1160–1170.

[57] Marcelo Mendoza and Nicolás Torres. 2020. Evaluating content novelty in recommender systems. *J Intell Inf Syst* 54 (2020). Issue 2.

[58] Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, Hamid Palangi, Guoqing Zheng, Corby Rosset, Hamed Khanpour, and Ahmed Awadallah. 2023. Orca 2:

Teaching Small Language Models How to Reason. arXiv:2311.11045 [cs.AI]
[59] Oded Ovadia, Menachem Brief, Moshik Mishaeli, and Oren Elisha. 2024. Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs. arXiv:2312.05934 [cs.AI]
[60] Aleksandr V. Petrov and Craig Macdonald. 2023. Generative Sequential Recommendation with GPTRec. arXiv:2306.11114 [cs.IR]
[61] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. OpenAI blog.
[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[63] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21, 1, Article 140 (2020).
[64] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (apr 2009), 333–389.
[65] Guy Shani, David Heckerman, and Ronen I. Brafman. 2005. An MDP-Based Recommender System. *Journal of Machine Learning Research* 6, 43 (2005), 1265–1295.
[66] Damien Sileo, Wout Vossen, and Robbe Raymaekers. 2022. Zero-Shot Recommendation as Language Modeling. In *Advances in Information Retrieval*, Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty (Eds.). Springer International Publishing, Cham, 223–230.
[67] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19)*. 1441–1450.
[68] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]
[69] Saúl Vargas and Pablo Castells. 2011. Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems. In *RecSys '11*. 109–116.
[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.
[71] Cunxiang Wang, Xiaoze Liu, Yuanhao Yue, Xiangru Tang, Tianhang Zhang, Cheng Jiayang, Yunzhi Yao, Wenyang Gao, Xuming Hu, Zehan Qi, Yidong Wang, Linyi Yang, Jindong Wang, Xing Xie, Zheng Zhang, and Yue Zhang. 2023. Survey on Factuality in Large Language Models: Knowledge, Retrieval and Domain-Specificity. arXiv:2310.07521 [cs.CL]
[72] Jie Wang, Fajie Yuan, Mingyue Cheng, Joemon M. Jose, Chenyun Yu, Beibei Kong, Xiangnan He, Zhijin Wang, Bo Hu, and Zang Li. 2022. TransRec: Learning Transferable Recommendation from Mixture-of-Modality Feedback. arXiv:2206.06190 [cs.IR]
[73] Lei Wang and Ee-Peng Lim. 2023. Zero-Shot Next-Item Recommendation using Large Pretrained Language Models. arXiv:2304.03153 [cs.IR]
[74] Chuhan Wu, Fangzhao Wu, Mingxiao An, Jianqiang Huang, Yongfeng Huang, and Xing Xie. 2019. Neural News Recommendation with Attentive Multi-View Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (Macao, China) *(IJCAI'19)*. AAAI Press, 3863–3869.
[75] Chuhan Wu, Fangzhao Wu, Suyu Ge, Tao Qi, Yongfeng Huang, and Xing Xie. 2019. Neural News Recommendation with Multi-Head Self-Attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 6389–6394.
[76] Chuhan Wu, Fangzhao Wu, Tao Qi, and Yongfeng Huang. 2021. Empowering News Recommendation with Pre-Trained Language Models. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21)*. Association for Computing Machinery, 1652–1656.
[77] Chuhan Wu, Fangzhao Wu, Tao Qi, Chao Zhang, Yongfeng Huang, and Tong Xu. 2022. MM-Rec: Visiolinguistic Model Empowered Multimodal News Recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22)*. Association for Computing Machinery, 2560–2564.
[78] 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. arXiv:2305.19860 [cs.IR]
[79] Hao Xu, Bo Yang, Xiangkun Liu, Wenqi Fan, and Qing Li. 2022. Category-aware Multi-relation Heterogeneous Graph Neural Networks for Session-based Recommendation. *Knowledge-Based Systems* 251 (2022).
[80] Fan Yang, Zheng Chen, Ziyan Jiang, Eunah Cho, Xiaojiang Huang, and Yanbin Lu. 2023. PALR: Personalization Aware LLMs for Recommendation. arXiv:2305.07622 [cs.IR]
[81] Zheng Yuan, Fajie Yuan, Yu Song, Youhua Li, Junchen Fu, Fei Yang, Yunzhu Pan, and Yongxin Ni. 2023. Where to Go Next for Recommender Systems? ID- vs. Modality-Based Recommender Models Revisited. In *Proceedings of the 46th International ACM SIGIR*

Conference on Research and Development in Information Retrieval (SIGIR '23). Association for Computing Machinery, 2639–2649.

[82] Gaowei Zhang, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, and Ji-Rong Wen. 2024. Scaling Law of Large Sequential Recommendation Models. In *Proceedings of the 18th ACM Conference on Recommender Systems*.

[83] Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2023. Recommendation as instruction following: A large language model empowered recommendation approach. arXiv:2305.07001 [cs.IR]

[84] Qi Zhang, Jingjie Li, Qinglin Jia, Chuyuan Wang, Jieming Zhu, Zhaowei Wang, and Xiuqiang He. 2021. UNBERT: User-News Matching BERT for News Recommendation. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, 3356–3362.

[85] Yuhui Zhang, Hao Ding, Zeren Shui, Yifei Ma, James Zou, Anoop Deoras, and Hao Wang. 2021. Language models as recommender systems: Evaluations and limitations. In *NeurIPS 2021 Workshop on I (Still) Can't Believe It's Not Better*.

[86] Zizhuo Zhang and Bang Wang. 2023. Prompt Learning for News Recommendation *(SIGIR '23)*. Association for Computing Machinery, 227–237.

[87] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020. S3-Rec: Self-Supervised Learning for Sequential Recommendation with Mutual Information Maximization. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 1893–1902.

[88] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. 2010. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences* 107, 10 (2010), 4511–4515.

Received 1 February 2024; revised 15 October 2024; accepted 13 November 2024