Text2Tracks: Generative Track Retrieval for Prompt-based Music Recommendation

Enrico Palumbo[1,*,†], Gustavo Penha[1,†], Andreas Damianou[1], José Luis Redondo García[1], Timothy Christopher Heath[1], Alice Wang[1], Hugues Bouchard[1] and Mounia Lalmas[1]

[1] *Spotify*

Abstract

Music recommender systems have traditionally relied on the user's listening history to provide personalized track recommendations. However, recent advancements in conversational interfaces powered by Large Language Models (LLMs) have enabled users to provide highly specific recommendation requests based on language prompts (e.g. *"Can you recommend some old school rock ballads to relax?"*). In this context, the track recommendation step is addressed in a generative way, i.e. titles of recommended music tracks are generated by the LLM by simply predicting the next textual token (e.g. *"Led Zeppelin - Stairway to Heaven"*). This strategy is sub-optimal for music items because: 1) it relies on generic text tokenization optimized for words rather than for items, 2) it requires an additional entity resolution layer to find the actual track identifier, and 3) the number of decoding steps scales linearly with the length of the artist name and song title, slowing down inference.

In this paper, we frame the task of prompt-based music recommendation as a generative retrieval task, and propose novel effective, and efficient representations of track identifiers that significantly outperform commonly used strategies. We introduce *Text2Tracks*, a generative retrieval model that learns a mapping from a user's music recommendation prompt to the relevant track IDs directly. Through an offline evaluation on three datasets of playlists with language inputs, we find that (1) the strategy to create IDs for music tracks is the most important factor for the effectiveness of *Text2Tracks* and that we can significantly outperform the artist name and track name strategy, (2) provided with the right choice of track identifiers, *Text2Tracks* outperforms sparse and dense retrieval for prompt-based track recommendation, and (3) several design decisions that were successfully applied to generative retrieval do not generalize to the music recommendation domain.

Keywords

generative recommendations, generative retrieval, music recommendation, semantic identifiers

1. Introduction

Conversational assistants such as ChatGPT [1] are receiving massive interest thanks to the groundbreaking abilities of Large Language Models (LLMs) [2, 3, 4]. A prominent feature of modern LLMs is their vast knowledge of media such as songs, films, or books [5]. This world knowledge is extremely useful in cold-start [6] or conversational recommendation scenarios [7], where user preferences are often elicited through natural language prompts more than by tapping into the user's history of interactions.

Conversational Recommenders (CR) [8, 9] have been proposed as a powerful paradigm to elicit user preferences via language prompts, enabling complex and specific content requests, multi-turn refinements, and explanations. Typically, CRs have been implemented as systems comprising many components [10, 11] such as query understanding, item retrieval/recommendation, dialogue management, and response generation. However, modern autoregressive LLMs can effectively handle all of these steps in a zero-shot way by simply predicting the next token [12].

In this context, the track recommendation step, where the LLM generates track identifiers based on a language prompt, can be seen as a Generative Retrieval problem (GR) [13]. GR has shown promising results for question answering and document retrieval tasks [13, 14]. Unlike sparse and dense approaches, GR does not rely on pre-computed indexes for the documents. In GR, a transformer model stores information about all the documents in the catalog within its parameters and directly generates document IDs for an input query. GR is particularly appealing in a conversational

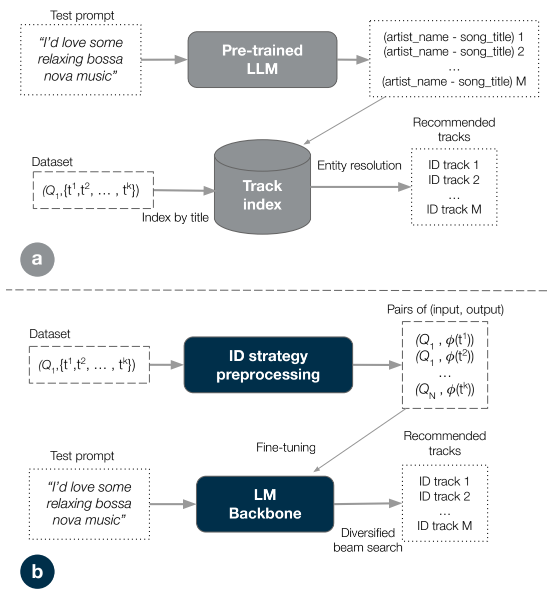

Figure 1: (a) Pre-trained LLMs deal with prompt-based music recommendation by generating the recommended artist name and song title, which are then resolved against an index to find the actual track identifiers. (b) *Text2Tracks* is a generative track retrieval model composed of a component that represents tracks, i.e. the ID strategy ϕ that maps from a track to its ID, and a backbone LM that is fine-tuned with pairs of music recommendation queries and track IDs. At test time *Text2Tracks* generates a set of recommended tracks using a diversified beam search strategy.

interface, as it opens the door to a single fine-tuned LLM handling all aspects of conversational recommendations, generating item identifiers in the same way it generates other textual content, such as follow-up questions, and explanations.

The 1st Workshop on Risks, Opportunities, and Evaluation of Generative Models in Recommender Systems (ROEGEN@RecSys 2024), October 2024, Bari, italy

*Corresponding author, email: enricop@spotify.com

† These authors contributed equally.

A critical question in the GR scenario is how to best represent item identifiers. In the music domain, pre-trained LLMs typically handle this step by generating the artist name and song title in plain text, as this is the most common representation found in their pre-training web data (Figure 1a). Although this approach is intuitive and works well in a zero-shot scenario, modeling tracks through their names has several shortcomings. First, artist and track names do not generally convey meaningful information about the underlying items, i.e. two songs might have a similar title but a completely different genre, style, and mood. Second, given the same artist and song name, finding the corresponding track identifier to play requires an additional entity resolution step, which can be non-trivial, especially for songs that come in many different versions (e.g. live, remastered, acoustic, cover). Third, representing track items by their names requires a number of decoding step that is proportional to their length, which can be very long, making the prediction slow and expensive.

In this paper, we propose to model the prompt-based music recommendation task, where user preferences are expressed through broad-intent natural language queries (e.g. *'Can you recommend some upbeat rock tracks to dance to?'*), as a generative retrieval problem. We propose an approach to generative track retrieval called *Text2Tracks*. Compared to the tasks GR has been tested on so far (document and passage retrieval, question answering), our domain poses a new set of challenges. In our domain, the retrieval units are music tracks typically associated with limited textual descriptions such as just the artist name and track title [15]. Also, music recommendation datasets have a stronger popularity bias with respect to document retrieval and question and answering datasets such as MSMarco [16]—hence, the problem is more about recommending the most canonical and representative items for a certain prompt rather than about semantic matching. Then, we focus on the problem of finding effective and efficient identifiers for music items for a generative retrieval model in the context of a prompt-based music recommendation system.

The proposed *Text2Tracks* uses the hierarchical structure available in the music domain—where tracks are linked to artists—for representing each music track, i.e. the ID strategy. To learn how queries relate to tracks, *Text2Tracks* leverages a language model backbone that maps from queries to track IDs. To better understand the capabilities of *Text2Tracks*, in this paper we address the following research questions:

- **RQ1** Which ID strategy is better suited for the generative retrieval approach to prompt-based music recommendation?
- **RQ2** Is *Text2Tracks* more effective than dense and sparse baselines for prompt-based music recommendation?
- **RQ3** Do the gains coming from previously proposed design decisions for generative retrieval models—constrained decoding, list target optimization, indexing step, and use of synthetic queries—generalize to the domain of music tracks?

Our findings reveal that (1) leveraging an ID strategy that takes into account the hierarchy of music tracks belonging to each artist and adding artists as special tokens provide the best results, outperforming the commonly used artist name and track title approach by 31% in Hits@10 while reducing the number of decoding steps by 7.5 times; (2) provided with this ID strategy, the generative retrieval *Text2Tracks* outperforms dense and sparse retrieval baselines for prompt-based music recommendation with relative gains between 100% and 170% of Hits@10 for the *track* granularity and between 30% and 100% for the *artist* granularity; and (3) a number of previously proposed model design decisions for generative retrieval do not lead to effectiveness gains in this domain, stressing the importance of ad-hoc experimentations for recommendation problems and the music domains.

The key contributions of this work are the *problem framing*, proposing to address prompt-based music recommendation as a generative retrieval problem; *id strategies*, introducing a number of new techniques to model track identifiers in a generative retrieval setting; *experimental analysis*, showing that this approach 1) outperforms typically used strategies for modeling track IDs, 2) is more effective than other track retrieval strategies (i.e. sparse, dense) thanks to the ability of predicting more canonical and popular tracks, and 3) does not benefit from other techniques that work well for generative retrieval in question answering and passage retrieval tasks.

The remainder of the paper is structured as follows. In Section 2 we present related work in the prompt-based recommendation and generative retrieval research areas. In Section 3 we formalize the task and describe the model architecture. In Section 4 we explain the experimental setup and provide implementation details. In Section 5 we report and comment on the results of our experiments. In Section 6 we draw the conclusions and describe future work.

2. Related Work

Prompt-based Recommendation Language-based preferences and textual inputs have always been a core part of recommender systems. Content-based recommender systems [17] have been around for several decades, matching user profiles with item textual metadata such as descriptions, reviews, tags. LLMs have further increased the attention toward the elicitation of language-based preferences for items, thanks to their ability to understand complex requests that rival traditional item-based recommenders, especially in cold-start scenarios [6]. Retrieving and recommending tracks for text-based inputs is inherently challenging [18, 19, 20]. Although music tracks can be described in many ways (e.g. genres, time, writer, artist, mood, speed, instruments, contextual activity, time of the day, etc), we typically have only a limited amount of metadata information describing each track. Additionally, users might want to refine the original query after examining the retrieved tracks, leading to the more challenging task of conversational recommendation [10]. Conversational recommendation is a type of conversational information-seeking activity [8] and is defined as *"a software system that supports its users in achieving recommendation related goals through a multi-turn dialogue"* [10]. Conversational recommenders have the potential to achieve multiple goals such as better eliciting the user information need, recommending certain items, explaining the recommendations, answering questions about the items, etc.

CRs have been typically implemented by pipelines with many components [10, 11] such as query understanding, item retrieval & recommendation, dialogue management, and response generation. With the breakthroughs of LLMs that can potentially do all those tasks, end-to-end approaches have become more popular [21, 22, 12]. One particularly challenging aspect of conversational recommendation with

LLMs is item retrieval, where broad descriptions must be matched against item collection. A common strategy in generative models is to predict the title of the item in an autoregressive fashion: *"Led"* $\rightarrow$ *"Led Zeppelin"* $\rightarrow$ ... $\rightarrow$ *"Led Zeppelin - Stairway to Heaven"* [12, 23], and match the generated titles against the item collection afterward. Even though this strategy works with API-based LLMs, more effective and efficient solutions can be achieved when fine-tuning a model for the prompt-based recommendation task. While research in this space is progressing at unprecedented speed, none of these works models the prompt-based music recommendation task as a generative retrieval problem.

Generative Retrieval Traditional retrieval systems have historically relied on keyword matching, and word frequencies to identify relevant documents for a given query [24]. These systems lack a semantic understanding of words, and documents are not matched if they use different words to express the same concept [25, 26]. Approaches to improve semantic matching between queries and documents in information retrieval have largely benefited [27] from breakthroughs in natural language processing achieved by transformer-based models [28], such as BERT [29] and T5 [30]. For the retrieval task, transformer-based models have been used to augment documents [31], learn representations for sparse retrieval methods [32, 33], and create Bi-encoders [34, 35, 36], i.e. models that encode the query and the document in a shared vector space where retrieval is performed as a nearest neighbor search. Bi-encoders have to encode all the information necessary in a single representation space, which can be suboptimal [37]. Moreover, Bi-encoders have limited robustness to zero-shot scenarios and can be outperformed by traditional lexical matching approaches [38, 39].

Generative retrieval has emerged as a new promising paradigm for semantic search. In GR, transformers act as differentiable search indexes [40, 14, 41, 42], learning to generate relevant document identifiers (IDs) for a specific query. Unlike Cross-encoders, the outputs of the models are not relevance scores for pairs of query and documents, but document IDs that are relevant to the input query. This way, the only input to GR is the query itself. Also unlike Bi-encoders, GR does not output embeddings that are later used to perform similarity calculations; it directly predicts document IDs. Because of the output space, all documents in the collection need to be part of the training set of GR models, as their IDs need to be learned and stored in the model weights, leading to challenges of scalability [43] and ingestion of new documents [44].

Another key point of GR models is assigning IDs for each document in the collection [13, 45, 46, 47]. Adding one new token to the vocabulary that represents each document in the collection can work for smaller collections, but it will quickly become intractable with reasonably sized collections. For this reason, it is crucial to adopt ID strategies that help the model learn the structure of the output space.

Previous work on GR focus on text-rich domains where documents are long and the focus is on semantic matching. In this paper, we propose to frame and study the problem of finding effective ID strategies, focusing on the music domain which has its own set of challenges compared to typical benchmarks.

3. Method

We start by formally defining prompt-based music recommendation as a generative track retrieval task, followed by the main components of the *Text2Tracks* model. Figure 1b displays a diagram of the model, showcasing how the training dataset $\mathcal{D}$ is first pre-processed into pairs of training instances using an ID strategy ϕ to represent the items. Then, the Language Model (LM) is fine-tuned and at test time uses diversified beam search to generate track recommendations. We first describe how the ID strategies work before introducing the LM backbone.

3.1. Prompt-based Music Recommendation

The modeling assumption of this paper is that prompt-based music recommendation can be framed as a generative track retrieval task. The track retrieval task is defined as retrieving a set of tracks relevant to a given music recommendation query. Formally let $\mathcal{D} = \{(Q_i, \{t^1, t^2, ..., t^k\})\}_{i=1}^{N}$ be a dataset composed of relevance labels for music recommendation queries, where Q is the query containing the music recommendation information need and $\{t^1, t^2, ..., t^k\}$ are the tracks that are relevant for this query. For the conversational recommendation setting, $Q^\tau = \{u_0, ..., u_\tau\}$ is the set of utterances from the user until the turn τ of the dialogue. The task is then to learn a function $f(Q)$ that maps from a query Q to a subset of the entire collection of tracks $\mathcal{T}$ in a retrieval-like fashion: $f(Q) \rightarrow \{t^1, t^2, ..., t^m\}$, where $m \ll |\mathcal{T}|$. Following the GR approach [13], we pose that $f(Q)$ is a transformer model with a decoder layer that directly generates the subset of relevant track IDs (see Section 3.3). In this setup, a crucial question is how to model track IDs.

3.2. Text2Tracks: ID strategies

The ID strategy is responsible for generating a string identifier for each item in the collection $\mathcal{T}$. We explore three types of IDs: based on the content of the item, based on integers, and learned ones. Figure 2 describes the three classes of approaches at a high level.

3.2.1. Content-based

This category of IDs uses the textual content of the item as an identifier. The advantage of this type of strategy is that the knowledge stored in the weights of the LM backbone model from its pre-training procedure can be leveraged, as the text metadata is also in the space of natural language. For a single item, each metadata category M has a value. For the tracks t_i, we have the artist name (M^1) and track title (M^2) metadata categories,[1] which leads to the *artist-name-track-title* approach: $\phi(t_i) = concat(value^{M1(t_i)}, value^{M2(t_i)})$. Table 1 shows an example of an item and its respective *artist-name-track-title* representation, where each value of the available metadata is concatenated.

3.2.2. Integer-based

This category of IDs uses integer identifiers to represent the items in the catalog. The most naive one is to assign random

[1]When available, other metadata information could be used to generate a content-based representation, for example, the genres of the artist and track.

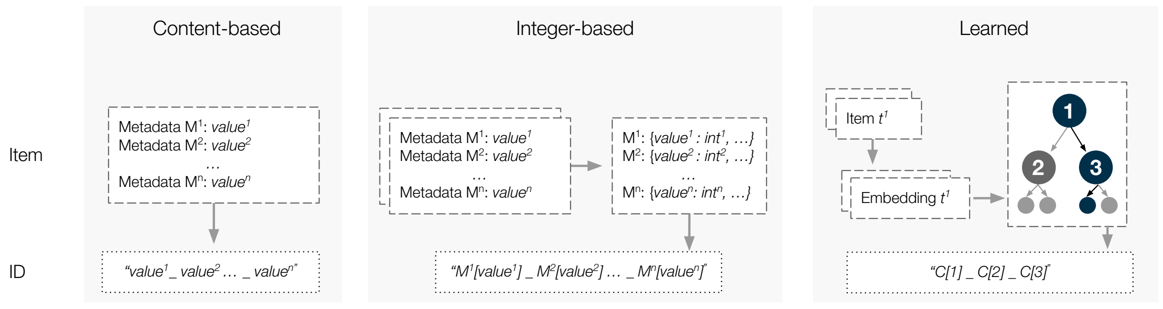

Figure 2: The three categories of ID strategies using "_" as a separator. Content-based strategies use textual metadata associated with the item. Integer-based approaches use random integer values for each metadata, potentially leveraging the hierarchy of metadata available. Learned approaches go from embeddings that represent the item to hierarchically structured tokens.

integers for each item in the collection: *track-int*. When using this strategy we do not add the IDs to the vocabulary of the LM backbone model, as adding millions of tokens to the vocabulary does not scale practically. This means that the LM backbone model will split the IDs into several tokens, and needs to learn to output tokens that when combined represent an item.

To leverage all available metadata we propose to have separate integers to represent each metadata value, which allows for tokens to be shared across items with the same metadata values. Given that the artist and track metadata categories are hierarchical, i.e. tracks belong to a main artist, we first generate random integers for the artist metadata values (M^1) and then we sequentially count in order of appearance in the training dataset the track name metadata values (M^2) of that artist. So for example, if the artist $A1$ has three different tracks t^1, t^2, t^3 they would be represented as follows: "*1_1*", "*1_2*", and "*1_3*"[2]. We refer to this strategy as *artist-int-track-seq*.

To facilitate learning a representation for certain items, we can also add a limited number of tokens related to a metadata category to the vocabulary. In *artist-iid-track-seq* we add the top-K popular artist's integers in terms of appearance in the training set. This allows the model to represent such artists with a single token instead of the combination of the tokens that the tokenizer would otherwise generate. The remainder of the artists (not in the top-K) get their tracks represented as in the previous approach *artist-int-track-seq*.

3.2.3. Learned

This category of IDs learns how to discretize the embeddings that represent each item in the collection. Ideally, similar items in the embedding space share more tokens after this discretization is done by the learned approach. So for example, if two tracks t^1, t^2 share the same genre, a learned approach could create the following hierarchy in their tokens, by using a shared token "*<0>*" in the beginning: "*<0><1>*" and "*<0><2>*". Learned strategies have also the benefit of controlling how many tokens each track is composed of, as well as the number of tracks that have the same tokens. While the other approaches in the paper uniquely identify each item with an ID, learned strategies can hash multiple tracks with the same tokens,[3] which allows for compression.

We test two different ways to encode each item in an embedding space that is used by the learned ID strategy afterward. The first approach is *text-based.* For each track, it creates a textual representation for it by concatenating the train set queries where the track appears. To go from the text to an embedding it uses a sentence encoder model. For example, if a track appears in three train set queries called *rock classics 00s*, *top hits 00s* and *guitar solos to learn*, we concatenate the titles in a single string *rock classics 00s, top hits 00s, guitar solos to learn* and then generate the embedding with a sentence encoder model [48]. The second approach is *cf-based*, and it relies on collaborative filtering to create item embeddings. Each query in the training set leads to a set of tracks, which can be thought of as a user. The co-occurrence of tracks for different queries (users) is leveraged by collaborative filtering models that learn representations based on sequences of items.

For each of the embedding space representations, we propose to use two different algorithms to discretize them. The first is a *hierarchical clustering* approach, in line with previous work [40], where the embedding space is recursively clustered into groups using a KMeans algorithm [49]. A root cluster is assigned a dummy token, and then, at each iteration, tokens are appended to the item representation depending on the cluster the item belongs to, leading to a sequence of m tokens where m is the maximum depth of the hierarchical clustering tree.

The second discretization algorithm proposed is inspired by the use of sparse coding for word sense induction [50]. We learn a sparse coding [51] for each item, which is an approximation of the item $embedding(t)$ as a sparse linear combination of a much smaller set of unit vectors, called the dictionary. After specifying the size of the dictionary, s, and the number of non-zero coefficients in each coding, c, we use a standard dictionary learning algorithm [52] to simultaneously learn the vectors, $v_1 \ldots v_s$, in the dictionary and the coefficients of the coding for each track, $a_1 \ldots a_s$, such that $t \approx a_1 v_1 + \ldots + a_s v_s$ while keeping the magnitude of $a_1, \ldots, a_s$ small. Ordering the non-zero coefficients from largest to smallest magnitude, $|a_{i_1}| \geq \ldots \geq |a_{i_c}|$, we obtain an ID for the track as $< \text{sgn}(a_{i_1}) i_1 > \ldots < \text{sgn}(a_{i_c}) i_c >$. This gives each track an ID with c integer tokens from a lexicon of $2s$ (s positively signed tokens and

[2]In practice, following [45] we start the counts for artists and tracks within that artist from 1000 to avoid problems with unique tokens and mixing the representations of tracks that would become subsets of other track ids.

[3]At test time, if the ID has multiple tracks associated with it, we select the most popular one. It is also possible to uniquely identify each item with learned strategies by adding another disambiguation token.

Table 1
Examples of the IDs generated for each strategy to represent a track for *Text2Tracks*. "<" and ">" delimiters indicate that the string is added to the vocabulary of the model as a new independent token. UI indicates if the strategy uniquely identifies each track. Underline indicates track name and dashed underline indicates artist name.

Category	ID strategy ϕ	UI	Example
content	*artist-name-track-title*	✓	"*The Beatles_Let It Be*"
	track-int	✓	"*1001*"
integer	*artist-int-track-seq*	✓	"*1001_1001*"
	artist-iid-track-seq	✓	"*<1001>_1001*"
learned	*dictionary-encoding*	✗	"*<0><2><3>*"
	hierarchical-clustering		

s negatively signed ones) possible tokens with the valuable property that two tracks whose IDs start with the same subsequence of tokens are likely to have similar representations in the original embedding space.

3.3. Text2Tracks: LM backbone

We propose to fully parameterize the retrieval function $f(Q)$ with a differentiable *seq2seq* transformer model. Following the intuition in [40], we hypothesize that all the necessary information for the generative track retrieval can be learned and stored within the backbone language model's parameters.

3.3.1. Training

When fine-tuning the LM backbone, each train set query Q generates k training instances, where k is the number of relevant tracks $\{t^1, t^2, ..., t^k\}$ for that query. Each track is first mapped to a textual ID by one of the ID strategies, and then the model is trained with the pairs of $(Q, \phi(t))$. For the inputs where we have multiple conversational turns, we concatenate the dialogue utterances into a single query: $Q = concat(\{u_0, ..., u_\tau\})$

An alternative way of fine-tuning the model is to use the entire set of relevant tracks for each training instance. We refer to this as *list targets*, which trains the LM backbone with instances of $(Q, concat(\phi(t^1), \phi(t^2), ..., \phi(t^k)))$, where *concat* concatenates the track IDs using a separator token.

3.3.2. Data augmentation

To increase the amount of training data we propose two ways of generating additional training instances: *synthetic queries* and the *indexing step*. To generate additional *synthetic queries*, we rely on playlist descriptions. For a given description, we choose from 2–4 non-stop words at random to generate a query. The synthetic queries could alleviate lexical gaps by describing the tracks with different terms. The *indexing step* generates additional training instances that go from the concatenation of the available metadata values (artist name and track title) to the IDs: $(concat(value^{M1(e)}, value^{M2(e)}), \phi(t^j))$. The additional training instances can help the model for queries based on artist names.

Table 2
Statistics of the datasets. *u.* stands for unique.

	Curated	MPD100k	CPCD
# queries (train)	75k	80k	448
# queries (dev)	771	10k	-
# queries (test)	523	1k	468
# u. queries	74k	29k	914
# u. artists	140k	53k	3902
# u. tracks	504k	258k	7353
avg. tracks query	12.69	15	15
% test tracks in train	100%	70.50%	65%

3.3.3. Inference

At inference time, track IDs are generated via diversified beam search [53] for the given prompt. For each group, beam search is applied to generate track IDs; however, there is a penalty for homogeneity across the generated tokens from different groups. This means that the model is penalized (as controlled by the homogeneity hyperparameter) when the output tokens are not diverse. Diverse beam search allows *Text2Tracks* to diversify the set of predictions, which is crucial for track recommendation. Optionally, at inference time the model can also resort to *constrained decoding* [41] to avoid the generation of invalid track IDs. All the valid IDs are added to a prefix trie that constrains the LM generation.

4. Experimental Setup

Datasets To answer our three research questions we rely on three datasets: MPD, Curated, and CPCD (statistics shown in Table 2). For the first two datasets, we treat the titles of the playlists as a proxy for the queries Q and the tracks inside the playlist as the relevant tracks for that query. The reasoning behind relying on playlist data is that playlist titles closely resemble broad-intent music searches (e.g. *chill music for studying*).

The Curated dataset comes from a popular music streaming platform. While the train and evaluation queries are a subset of user-generated playlists and tags that describe the tracks, the test queries are a selected set of playlists created by a team of professional music editors in the case of Curated. We guarantee that all the tracks in the test set appear in the train set, and enforce that the queries of the test playlists are not exact matches with the train set queries.

The MPD dataset is a random subset of 100k playlists sampled from the Spotify Million Playlist Dataset Challenge[4]. The playlists have been created by users on the Spotify platform between January 2010 and October 2017.

The third dataset is the Conversational Playlist Curation Dataset (CPCD),[5] a dataset that has been created to evaluate conversational music recommendation systems using a human-human methodology, i.e. one annotator plays the role of the user, and one of the system [11]. We apply on CPCD conversational queries a pre-processing step using GPT3.5 to replace requests for specific artists with more generic music descriptors. We do so because we are not

[4]The MPD dataset is available at https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge
[5]The CPCD dataset is available at https://github.com/google-research-datasets/cpcd/tree/main

interested in evaluating the models for narrow queries targeted at specific artists, but for the effectiveness in track recommendation on broader exploratory intents [54]. For instance, a query such as *"Hey there! I'm hoping to make an "oldies but goodies" playlist. Can you help? Could you incorporate some Beach boys, The Temptations and similar artists?"* becomes *"I'm hoping to make an oldies but goodies playlist. Can you help? I like classic 60s music. Could you incorporate some iconic bands from that era and similar artists?"*. We follow the setup proposed in [11] and concatenate the different turns of the conversations for the track retrieval task.

Baselines & Implementation Our first baseline, Popularity, retrieves the most popular tracks regardless of the query. For the dense (Bi-encoder) and sparse (BM25) baselines, we represent each track by the titles of playlists they appear in the train set. So for example if t_i appears in the playlists "*rock*", "*metal*" and "*guitar solos*" we represent t_i by their concatenation: "*rock, metal, guitar solos*". For BM25 [55] we resort to the default hyperparameters and implementation provided by the PyTerrier toolkit [56]. For the Bi-encoder$_{zs}$ model, we rely on the SentenceTransformers [48] model releases.[6] The library uses Hugginface transformers for the pre-trained models such as MPNet [57]. Specifically, we employ the pre-trained model *all-mpnet-base-v2*. When fine-tuning Bi-encoder$_{ft}$ we rely on the *MultipleNegatives-RankingLoss*.

Text2Tracks uses a T5 [58] base model that was instruction-tuned, known as Flan-T5 [59]. We rely on the Hugginface library to fine-tune it, and unless otherwise stated we use the *flan-t5-base* pre-trained model. We fine-tune it for a total of 20 epochs, with a learning rate of $5e^{-4}$ and a batch size of 64. For the ID strategy *artist-iid-track-seq* we add the top 50k artists as tokens to the vocabulary. Unless otherwise stated, *Text2Tracks* is using the *artist-iid-track-seq* representation and does not use constrained decoding, list targets, indexing step and synthetic queries. For the text-based learned methods for representing IDs, we rely on the *all-MiniLM-L6-v2* model from SentenceTransformers to encode the textual representation of the tracks. For the *cf-based* learned methods, we rely on *word2vec* model [60] to generate embeddings by training it on the sequences of tracks. For KMeans and the dictionary learning approach, we use the *scikit-learn* library.[7]

Evaluation We evaluate our models using the Mean Average Precision (MAP) and the number of relevant items (Hits), with a cut-off thresholds of 10. We follow the pessimistic assumption that only the tracks in the test playlist are relevant, in line with most evaluation setups in the recommender systems literature [61]. Hence, the effectiveness scores are only a lower bound estimate, due to the lack of relevance judgments of all the recommended items. We use Student t-tests with Bonferonni correction and a confidence level of 0.95 to calculate the statistical significance of different models. We calculate the metrics in both the *track* granularity, i.e. relevant tracks were retrieved at the top positions, and the artist granularity, i.e. relevant artists were retrieved at the top positions regardless of the specific tracks returned.

[6]https://www.sbert.net/docs/pretrained_models.html
[7]https://scikit-learn.org/stable/

5. Results

We now go over the results to our three research questions.

5.1. ID strategies (RQ1)

Our first research question is on the "*comparison of different track ID strategy representations*". In Table 3 we see the results for the different strategies to represent IDs. **Our results show that the ID strategy is a deciding factor for the effectiveness of *Text2Tracks*, achieving the best performance with *artist-iid-track-seq*.** When using *artist-iid-track-seq*, tracks from the same artists share tokens, and at prediction time the model is forced to first retrieve the relevant artist, and within that artist predict the relevant track. The results show that the *artist→track* hierarchy provided by *artist-iid-track-seq* is better than trying to learn a hierarchy from the collaborative-filtering information (rows 6 and 7 of Table 3) or from the textual information (rows 4 and 5 of Table 3).[8] By comparing *artist-iid-track-seq* with *artist-int-track-seq* we see that adding the top 50k artists as independent tokens to the vocabulary increases the model's effectiveness.

Besides the learned *cf-based* approaches, the worst-performing ID representation is *track-int* which does not add any prior knowledge to the initial representation of the tracks, and requires the model to learn how queries match groups of tokens (on average 4 tokens) that represent the tracks. When using *artist-name-track-title*, the LM can leverage the "knowledge" stored in its weights regarding the items, e.g. it could already know that the word "*rock*" is semantically close to the rock band "*ACDC*". Even though the number of tokens per track is high when using *artist-name-track-title* (∼15 tokens on average), it is still able to outperform *track-int*.

Overall, our best strategy *artist-iid-track-seq* outperforms the commonly used strategy in pre-trained LLMs *artist-name-track-title* by 31% in Hits@10, while being much more efficient (2 decoding steps vs an average of 15 decoding steps).

5.2. *Text2Tracks* effectiveness (RQ2)

To answer our second research question on "*the effectiveness of* Text2Tracks" we display in Table 4 the results for the task of generative track retrieval for the `Curated`, `MPD100k` and `CPCD` datasets. We see that Text2Tracks outperforms the baselines for all the three datasets with statistical significance, for both prediction granularities (*artist* and *track* levels). **This answers our RQ2 positively, indicating that *Text2Tracks* is an effective approach for track retrieval, outperforming sparse and dense baselines with relative gains between 100% and 170% of Hits@10 for the *track* granularity and between 30% and 100% for the *artist* granularity.** The consistency of the improvements across the three datasets show that *Text2Tracks* is effective both when playlist titles are used as a music recommendation prompts (`Curated`, `MPD100k`) and with more complex conversational music recommendation queries (`CPCD`).

[8]We leave as future work the exploration of better representation spaces to learn ID strategies upon, such as more sophisticated collaborative-filtering methods for learning embeddings, or embeddings coming from the audio information of a track.

Table 3
Results of different ID strategies for the `Curated` dataset (RQ1). Bold denotes the highest effectiveness and superscripts mean statistical significance against the respective ID strategy using Student t-tests with Bonferroni correction. Tracks p. ID is the number of tracks each ID represents on average. Tokens p. track is the average number of tokens the tracks have when tokenizing their IDs. Vocab. size indicates the number of tokens after we added the new tokens to the existing 32.1k vocabulary.

	Category	ID strategy	Tracks p. ID	Tokens p. track	Vocab. size	Hits@10		MAP@10	
						track	*artist*	*track*	*artist*
0	content	*artist-name-track-title*	1.0	14.73	32.1k	0.182^{45}	**0.402**^{145}	0.025^{45}	0.089^{14567}
1	integer	*track-int*	1.0	3.93	32.1k	0.140^{4}	0.239^{45}	0.017	0.039^{45}
2		*artist-int-track-seq*	1.0	6.57	32.1k	0.185^{45}	0.340^{145}	0.027^{45}	0.090^{14567}
3		*artist-iid-track-seq*	1.0	5.91	82.1k	**0.239**^{145}	0.350^{145}	**0.034**^{1456}	**0.120**^{0124567}
4	learned	*cf-based-hierarchical-clustering*	1.8	4.00	147.8k	0.042	0.134	0.006	0.016
5		*cf-based-dictionary-encoding*	3.6	4.00	32.6k	0.076	0.141	0.008	0.022
6		*text-based-hierarchical-clustering*	1.3	4.00	101.6k	0.170^{45}	0.321^{45}	0.020^{45}	0.056^{145}
7		*text-based-dictionary-encoding*	1.7	4.00	32.6k	0.180^{45}	0.358^{145}	0.024^{45}	0.060^{145}

Table 4
Results for the effectiveness of *Text2Tracks* (RQ2). Bold denotes the highest effectiveness for the dataset and superscripts mean statistical significance against the respective model using Student t-tests with Bonferroni correction.

	Model	Hits@10		MAP@10	
		track	*artist*	*track*	*artist*
	`Curated`				
0	Popularity	0.019	0.050	0.003	0.007
1	BM25	0.101^{0}	0.197^{0}	0.015^{0}	0.054^{0}
2	Bi-encoder$_{zs}$	0.065	0.182^{0}	0.010	0.041^{0}
3	Bi-encoder$_{ft}$	0.119^{0}	0.207^{0}	0.013^{0}	0.054^{0}
4	Text2Tracks	**0.239**^{0123}	**0.350**^{0123}	**0.034**^{0123}	**0.120**^{0123}
	`MPD100k`				
0	Popularity	0.121^{23}	0.325^{23}	0.014^{23}	0.061^{23}
1	BM25	0.071^{23}	0.252^{23}	0.008^{23}	0.044^{23}
2	Bi-encoder$_{zs}$	0.023	0.159	0.002	0.027
3	Bi-encoder$_{ft}$	0.022	0.140	0.002	0.024
4	Text2Tracks	**0.330**^{0123}	**0.507**^{0123}	**0.038**^{0123}	**0.113**^{0123}
	CPCD				
0	Popularity	0.002	0.052	0.001	0.006
1	BM25	0.131^{0}	0.265^{0}	0.022^{0}	0.047^{0}
2	Bi-encoder$_{zs}$	0.148^{0}	0.340^{01}	0.025^{0}	0.060^{01}
3	Bi-encoder$_{ft}$	0.222^{012}	0.477^{012}	0.034^{012}	0.081^{012}
4	Text2Tracks	**0.538**^{0123}	**0.613**^{0123}	**0.047**^{0123}	**0.108**^{0123}

Why is *Text2Tracks* more effective than traditional retrieval baselines? We hypothesize that the gains of *Text2Tracks* over traditional sparse and dense baselines come from two main reasons. The first is the lack of textual information for the tracks in such datasets besides the track and artist names. This way all models rely exclusively on the train queries and predicting which track is relevant directly allows for more expressiveness than using similarity between the representations of the query and tracks. See for example the predicted tracks from *Text2Tracks* and Bi-encoder$_{ft}$ for the query "*Christmas Hits*" on the first row of Table 5. While *Text2Tracks* retrieves 3 relevant tracks, Bi-encoder$_{ft}$ is not able to find a relevant artist or track in the top 5 positions. The second one is that *Text2Tracks* can distinguish between popular and non-popular items based on the occurrence of such tracks in the training set. While tokens representing popular artists and tracks are more likely to be generated by *Text2Tracks*, the textual representations of tracks for traditional retrieval methods cannot distinguish two tracks with different popularity values. For example, the first track retrieved for the "*Christmas Hits*" query is "*Mariah Carey - All I Want For Christmas Is You*", a frequent item in the training set.

Table 5
Example of tracks recommended by the proposed *Text2Tracks* compared to the Bi-encoder$_{ft}$ model for the 'Christmas hits' query. In dashed underline we have the predicted artists and tracks of the models that are relevant according to the ground-truth `MPD100k` playlists. *Text2Tracks* can recommend more canonical and relevant tracks and artists in the first positions of the list.

Top 5 tracks predictions	
Text2Tracks	Bi-encoder$_{ft}$
Mariah Carey - All I Want [...]	Marty Stuart - Even Santa [...]
Wham! - Last Christmas	Cascada - Last Christmas
Bing Crosby - White Christmas	Reba McEntire - Silent Night
Andy Williams - It's the Most[...]	Jonny Lang - Santa Claus Is [...]
José Feliciano - Feliz Navidad	Gladys Knight & T. P. - When [...]

How does *Text2Tracks* diversify predictions and how does diversity affect effectiveness? The homogeneity parameter of diverse beam search allows for control of how much the generation of similar predictions across different beam search groups is penalized [53]. We study the effect that this parameter has on the model's effectiveness and on the diversity of the recommended tracks, measured as the Shannon entropy of the list of predicted artists (Figure 3).

We observe that the artist entropy (i.e. diversity) monotonically increases when the homogeneity penalty increases, while the effectiveness has a local maximum at ≈ 0.25. One of the appealing features of *Text2Tracks* is that it is straightforward to sacrifice some effectiveness to increase diversity or vice versa based on the use case at hand.

5.3. Model design choices (RQ3)

Our third research question explores four previously proposed model design decisions for generative retrieval, namely constrained decoding [41], list target optimization [42], indexing step [40], and use of synthetic queries [62], generalize to track recommendation.

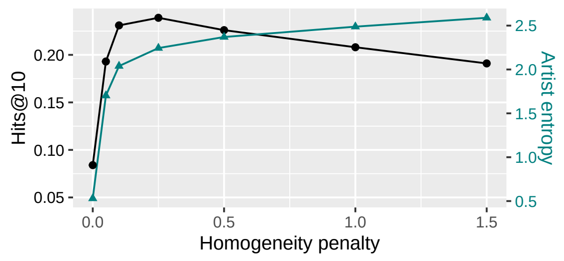

Figure 3: The effect on Hits@10 (●) for the *track* granularity and on the diversity of the artists (▲) when increasing the homogeneity penalty hyperparameter, which applies a penalty for generating tokens that were selected in other beam search groups at prediction with *Text2Tracks* for the `Curated` dataset.

Table 6
Results of different *Text2Tracks* design choices for the `Curated` dataset (RQ3). Superscript ↓ means statistically worse results compared with *Text2Tracks* baseline using Student t-tests with Bonferroni correction.

	Hits@10		MAP@10	
Design choice	*track*	*artist*	*track*	*artist*
Text2Tracks	0.239	0.350	0.034	0.120
w. constr. decoding [41]	0.245	0.354	0.035	0.121
w. list targets [42]	0.078$^{\downarrow}$	0.229$^{\downarrow}$	0.013$^{\downarrow}$	0.040$^{\downarrow}$
w. indexing step [40]	0.231	0.333	0.031	0.121
w. synthetic queries [62]	0.243	0.348	0.032	0.109

First, we look into whether constraining the output space to valid IDs only [14, 41] improves the effectiveness of *Text2Tracks*. We see in Table 6 that there is no statistically significant difference between the model with and without constrained decoding. By checking the model predictions, we see that around 80% of them are valid IDs. The low percentage of ID hallucinations indicates that constrained decoding does not play an important role here, and simply returning more IDs with diversified beam search is enough to reach the best effectiveness.

The second modification we evaluate is training the *Text2Tracks* with lists targets, similar to [42]. As described in Section 3.3 the model is trained to generate the entire playlist in a single sequence. We see in Table 6 that training with list targets significantly degrades model effectiveness.[9]

The third variation is a data augmentation strategy that resembles the DSI's indexing step [40]. As described in Section 3.3 we add training instances that go from the artist name and track title to the track IDs, with the hypothesis that this could help mapping knowledge learned during the LM pre-training to the track IDs space. However, we do not observe a significant difference when using the indexing step against not using them in Table 6.

Finally, we analyze the use of synthetically generated queries [62, 43] to augment the training data. We use the playlist description of the train playlists as the input text for the method to generate synthetic queries.[10] For example, the playlist with the title "*bedtime songs for children*" and description "*Gentle, mellow melodies to help the kids settle down and fall asleep.*", could generate the following synthetic query: "*kids fall asleep*". The results in Table 6 show that such synthetic queries do not improve the effectiveness of *Text2Tracks* significantly. Overall, our findings reveal no statistical improvements coming from the four previously proposed design modifications analyzed here, answering our third research question negatively. These results remark the difference between the domains and tasks where GR is generally applied and the importance of this experimentation tailored to the music domain.

6. Conclusions

In this paper, we propose to address the problem of prompt-based music recommendation through the lenses of generative retrieval and we introduce a generative track retrieval model *Text2Tracks*. Given that tracks are items with scarce text, *Text2Tracks* learns to generate track IDs directly from a textual query as an end-to-end differentiable model. We propose several new strategies to represent track IDs, and show that the track ID representation strategy is a crucial ingredient to obtain high effectiveness. By leveraging the artist-track hierarchy and adding special tokens to the vocabulary we can improve by 31% Hits@10 with respect to the commonly used strategy of generating artist and track names. We then show that for three different playlist datasets with language inputs *Text2Tracks* outperforms dense and sparse retrieval solutions with relative gains ranging between 100–170% in Hits@10 for the *track* granularity and 30–100% for the *artist* granularity. We also show that more complex strategies that led to improvements in other domains such as question answering do not transfer to our problem space.

These findings show that the problem framing that we propose combined with the novel ID strategies that we introduce for music tracks lead to sizable improvements both in terms of effectiveness and efficiency of prompt-based music recommendation. Also, the non-transferability of techniques that have worked well in other domains highlights the non-triviality and importance of our experimentation. Our analysis further strengthens our contribution, showing that the success of *Text2Tracks* is due to its ability to capture the canonicalness and the popularity of tracks, a key ingredient for the prompt-based recommendation problems.

We believe that this work represents a stepping stone in the use of generative retrieval in the music domain, leading to more accurate models that can also greatly simplify music recommendation pipelines based on indexing and retrieving vectors. The generative retrieval approach is particularly appealing in a conversational setup as it paves the way to a fully fine-tuned conversational recommender that can generate track IDs jointly with explanations, follow-up requests, and any other textual content.

In future work, we plan to extend this approach to generate joint track recommendations and textual responses, experimenting with decoder-only models such as Llama [3]. We also aim to test the generalizability of our findings to other media items such as podcasts or books.

References

[1] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray,

[9] A number of additional unsuccessful experiments with list targets were performed: completing train playlists that do not have at least 15 tracks with tracks returned by the Bi-encoder model for the playlist, generation of multiple prediction lists at test time, and combining them with RRF [63], as well as shuffling the order of the tracks inside each playlist for different epochs.

[10] We leave the study of more sophisticated models for generating synthetic queries for tracks as future work. For example, methods that rely on large language models to generate synthetic queries have significant potential.

et al., Training language models to follow instructions with human feedback, Advances in neural information processing systems 35 (2022) 27730–27744.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, arXiv preprint arXiv:2303.08774 (2023).

[3] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, arXiv preprint arXiv:2307.09288 (2023).

[4] J. FitzGerald, S. Ananthakrishnan, K. Arkoudas, D. Bernardi, A. Bhagia, C. D. Bovi, J. Cao, R. Chada, A. Chauhan, L. Chen, et al., Alexa teacher model: Pretraining and distilling multi-billion-parameter encoders for natural language understanding systems, 2022.

[5] G. Penha, C. Hauff, What does bert know about books, movies and music? probing bert for conversational recommendation, in: Proceedings of the 14th ACM conference on recommender systems, 2020, pp. 388–397.

[6] S. Sanner, K. Balog, F. Radlinski, B. Wedin, L. Dixon, Large language models are competitive near cold-start recommenders for language-and item-based preferences, in: Proceedings of the 17th ACM conference on recommender systems, 2023, pp. 890–896.

[7] Y. Deldjoo, Z. He, J. McAuley, A. Korikov, S. Sanner, A. Ramisa, R. Vidal, M. Sathiamoorthy, A. Kasirzadeh, S. Milano, A review of modern recommender systems using generative models (gen-recsys), arXiv preprint arXiv:2404.00579 (2024).

[8] H. Zamani, J. R. Trippas, J. Dalton, F. Radlinski, et al., Conversational information seeking, Foundations and Trends® in Information Retrieval 17 (2023) 244–456.

[9] Y. Deldjoo, J. R. Trippas, H. Zamani, Towards multi-modal conversational information seeking, in: Proceedings of the 44th International ACM SIGIR conference on research and development in Information Retrieval, 2021, pp. 1577–1587.

[10] D. Jannach, A. Manzoor, W. Cai, L. Chen, A survey on conversational recommender systems, ACM Computing Surveys (CSUR) 54 (2021) 1–36.

[11] A. T. Chaganty, M. Leszczynski, S. Zhang, R. Ganti, K. Balog, F. Radlinski, Beyond single items: Exploring user preferences in item sets with the conversational playlist curation dataset, in: Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2023, pp. 2754–2764.

[12] Z. He, Z. Xie, R. Jha, H. Steck, D. Liang, Y. Feng, B. P. Majumder, N. Kallus, J. McAuley, Large language models as zero-shot conversational recommenders, in: Proceedings of the 32nd ACM international conference on information and knowledge management, 2023, pp. 720–730.

[13] Y. Tay, V. Q. Tran, M. Dehghani, J. Ni, D. Bahri, H. Mehta, Z. Qin, K. Hui, Z. Zhao, J. Gupta, et al., Transformer memory as a differentiable search index, 2022.

[14] Y. Wang, Y. Hou, H. Wang, Z. Miao, S. Wu, Q. Chen, Y. Xia, C. Chi, G. Zhao, Z. Liu, et al., A neural corpus indexer for document retrieval, Advances in Neural Information Processing Systems 35 (2022) 25600–25614.

[15] O. Celma, Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space, Springer, 2010.

[16] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, L. Deng, Ms marco: A human generated machine reading comprehension dataset, 2016.

[17] P. Lops, M. De Gemmis, G. Semeraro, Content-based recommender systems: State of the art and trends, Recommender systems handbook (2011) 73–105.

[18] M. Kaminskas, F. Ricci, Contextual music information retrieval and recommendation: State of the art and challenges, Computer Science Review 6 (2012) 89–119.

[19] G. Bonnin, D. Jannach, Automated generation of music playlists: Survey and experiments, ACM Computing Surveys (CSUR) 47 (2014) 1–35.

[20] A. Nanopoulos, D. Rafailidis, M. M. Ruxanda, Y. Manolopoulos, Music search engines: Specifications and challenges, Information Processing & Management 45 (2009) 392–396.

[21] C. Li, H. Hu, Y. Zhang, M.-Y. Kan, H. Li, A conversation is worth a thousand recommendations: A survey of holistic conversational recommender systems, arXiv preprint arXiv:2309.07682 (2023).

[22] X. Wang, X. Tang, W. X. Zhao, J. Wang, J.-R. Wen, Rethinking the evaluation for conversational recommendation in the era of large language models, arXiv preprint arXiv:2305.13112 (2023).

[23] N. De Cao, G. Izacard, S. Riedel, F. Petroni, Autoregressive entity retrieval, arXiv preprint arXiv:2010.00904 (2020).

[24] S. Robertson, H. Zaragoza, et al., The probabilistic relevance framework: Bm25 and beyond, Foundations and Trends® in Information Retrieval 3 (2009) 333–389.

[25] C. Van Gysel, Remedies against the vocabulary gap in information retrieval, arXiv preprint arXiv:1711.06004 (2017).

[26] Z. Ji, Z. Lu, H. Li, An information retrieval approach to short text conversation, arXiv preprint arXiv:1408.6988 (2014).

[27] A. Yates, R. Nogueira, J. Lin, Pretrained transformers for text ranking: Bert and beyond, in: Proceedings of the 14th ACM International Conference on web search and data mining, 2021, pp. 1154–1156.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, 2017.

[29] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[30] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[31] R. Nogueira, J. Lin, A. Epistemic, From doc2query to docTTTTTquery, Online preprint 6 (2019) 2.

[32] T. Nguyen, S. MacAvaney, A. Yates, A unified framework for learned sparse retrieval, in: European Conference on Information Retrieval, Springer, 2023, pp. 101–116.

[33] T. Formal, C. Lassance, B. Piwowarski, S. Clinchant, Splade v2: Sparse lexical and expansion model for information retrieval, arXiv preprint arXiv:2109.10086 (2021).

[34] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, arXiv preprint arXiv:1908.10084 (2019).

[35] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, E. Grave, Unsupervised dense information retrieval with contrastive learning, arXiv preprint arXiv:2112.09118 (2021).

[36] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W.-t. Yih, Dense passage retrieval for open-domain question answering, arXiv preprint arXiv:2004.04906 (2020).

[37] O. Khattab, M. Zaharia, Colbert: Efficient and effective passage search via contextualized late interaction over bert, in: Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, 2020, pp. 39–48.

[38] R. Ren, Y. Qu, J. Liu, W. X. Zhao, Q. Wu, Y. Ding, H. Wu, H. Wang, J.-R. Wen, A thorough examination on zero-shot dense retrieval, arXiv preprint arXiv:2204.12755 (2022).

[39] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, I. Gurevych, Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models, arXiv preprint arXiv:2104.08663 (2021).

[40] Y. Tay, V. Tran, M. Dehghani, J. Ni, D. Bahri, H. Mehta, Z. Qin, K. Hui, Z. Zhao, J. Gupta, et al., Transformer memory as a differentiable search index, Advances in Neural Information Processing Systems 35 (2022) 21831–21843.

[41] M. Bevilacqua, G. Ottaviano, P. Lewis, S. Yih, S. Riedel, F. Petroni, Autoregressive search engines: Generating substrings as document identifiers, Advances in Neural Information Processing Systems 35 (2022) 31668–31683.

[42] X. Chen, Y. Liu, B. He, L. Sun, Y. Sun, Understanding differential search index for text retrieval, arXiv preprint arXiv:2305.02073 (2023).

[43] R. Pradeep, K. Hui, J. Gupta, A. D. Lelkes, H. Zhuang, J. Lin, D. Metzler, V. Q. Tran, How does generative retrieval scale to millions of passages?, arXiv preprint arXiv:2305.11841 (2023).

[44] V. Kishore, C. Wan, J. Lovelace, Y. Artzi, K. Q. Weinberger, Incdsi: Incrementally updatable document retrieval (2023).

[45] W. Hua, S. Xu, Y. Ge, Y. Zhang, How to index item ids for recommendation foundation models, arXiv preprint arXiv:2305.06569 (2023).

[46] Y. Li, N. Yang, L. Wang, F. Wei, W. Li, Multiview identifiers enhanced generative retrieval, arXiv preprint arXiv:2305.16675 (2023).

[47] S. Rajput, N. Mehta, A. Singh, R. H. Keshavan, T. Vu, L. Heldt, L. Hong, Y. Tay, V. Q. Tran, J. Samost, et al., Recommender systems with generative retrieval, arXiv preprint arXiv:2305.05065 (2023).

[48] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019. URL: https://arxiv.org/abs/1908.10084.

[49] D. Sculley, Web-scale k-means clustering, in: Proceedings of the 19th international conference on World wide web, 2010, pp. 1177–1178.

[50] S. Arora, Y. Li, Y. Liang, T. Ma, A. Risteski, Linear algebraic structure of word senses, with applications to polysemy, Transactions of the Association for Computational Linguistics 6 (2018) 483–495.

[51] B. A. Olshausen, D. J. Field, Sparse coding with an overcomplete basis set: A strategy employed by v1?, Vision research 37 (1997) 3311–3325.

[52] J. Mairal, F. Bach, J. Ponce, G. Sapiro, Online dictionary learning for sparse coding, in: Proceedings of the 26th annual international conference on machine learning, 2009, pp. 689–696.

[53] A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, D. Batra, Diverse beam search: Decoding diverse solutions from neural sequence models, arXiv preprint arXiv:1610.02424 (2016).

[54] G. Penha, E. Palumbo, M. Aziz, A. Wang, H. Bouchard, Improving content retrievability in search with controllable query generation, in: Proceedings of the ACM Web Conference 2023, 2023, pp. 3182–3192.

[55] S. E. Robertson, S. Walker, Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval, in: SIGIR'94, Springer, 1994, pp. 232–241.

[56] C. Macdonald, N. Tonellotto, Declarative experimentation ininformation retrieval using pyterrier, in: Proceedings of ICTIR 2020, 2020.

[57] K. Song, X. Tan, T. Qin, J. Lu, T.-Y. Liu, Mpnet: Masked and permuted pre-training for language understanding, Advances in Neural Information Processing Systems 33 (2020) 16857–16867.

[58] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, arXiv preprint arXiv:1910.10683 (2019).

[59] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma, et al., Scaling instruction-finetuned language models, arXiv preprint arXiv:2210.11416 (2022).

[60] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, 2013.

[61] A. Said, A. Bellogín, Comparative recommender system evaluation: benchmarking recommendation frameworks, in: Proceedings of the 8th ACM Conference on Recommender systems, 2014, pp. 129–136.

[62] S. Zhuang, H. Ren, L. Shou, J. Pei, M. Gong, G. Zuccon, D. Jiang, Bridging the gap between indexing and retrieval for differentiable search index with query generation, arXiv preprint arXiv:2206.10128 (2022).

[63] G. V. Cormack, C. L. Clarke, S. Buettcher, Reciprocal rank fusion outperforms condorcet and individual rank learning methods, in: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, 2009, pp. 758–759.