Conversational Music Retrieval with Synthetic Data

Megan Leszczynski∗†1,2 Ravi Ganti∗2 Shu Zhang∗2 Krisztian Balog∗2
Filip Radlinski2 Fernando Pereira2 Arun Tejasvi Chaganty∗2
1 Stanford University 2 Google Research
mleszczyn@cs.stanford.edu
{gmravi,shzhang,krisztianb}@google.com
{filiprad,pereira,arunchaganty}@google.com

Abstract

Users looking for recommendations often wish to improve suggestions through broad natural language feedback (e.g., “How about something more upbeat?”). However, building such conversational retrieval systems requires conversational data with rich user utterances paired with slates of items that cover a diverse range of preferences. This is challenging to collect scalably using conventional methods like crowd-sourcing. We address this problem with a new technique to synthesize high-quality dialog data by transforming the domain expertise encoded in curated item collections into corresponding item-seeking conversations. The method first generates a sequence of hypothetical slates returned by a system, and then uses a language model to introduce corresponding user utterances. We apply the approach on a dataset of curated music playlists to generate 10k diverse music-seeking conversations. A qualitative human evaluation shows that a majority of these conversations express believable sequences of slates and include user utterances that faithfully express preferences for them. When used to train a conversational retrieval model, the synthetic data yields up to a 23% relative gain on standard retrieval metrics compared to baselines trained on non-conversational and conversational datasets.

1 Introduction

Recent work has made significant advances in the ability of retrieval systems to understand natural language queries and non-textual content (e.g., images, audio) [11, 12, 22]. However, standard retrieval systems still struggle to retrieve items for ambiguous queries (e.g., “Music for focusing”), where the query’s meaning may depend on the user or their context (e.g., writing, meditating). This motivates conversational retrievers, which allow the user to provide natural language feedback (e.g., “How about something more upbeat?”) to steer the system to retrieve the items they are looking for.

Conversational retrievers are challenging to build because they require conversational training data that pairs multiple turns of rich and diverse natural language feedback with retrieved items. One possibility is to generate data in a crowd-sourced Wizard-of-Oz setup [2, 14, 18, 24]: here, one person acts a user looking for items, while another acts as a wizard that recommends items given the user’s queries and feedback. A key limitation of this approach is that both people need some domain expertise: the wizard to find relevant items to suggest, and the user to provide varied and meaningful feedback. For many domains, few people have this expertise, leading to shallow conversations.
In contrast, curated item collections (e.g., playlists, recipe books) are widely available on the internet. These collections capture the domain expertise of their creators who pick a coherent set of items. Moreover, they often include detailed metadata (e.g., titles, descriptions and tags) with attributes a user may refer to in their preferences (e.g., “upbeat music”, “healthy recipes”). Motivated by these properties, we ask: can we leverage curated item collections to generate conversations?

There are two key features missing in item collections: (1) multiple turns with slates of items to recommend to the user, and (2) user utterances describing their feedback for each slate. We solve these problems in two steps: First, we observe that, in an ideal conversation, each turn should make a coherent change that brings the user closer to their target slate. Following Göpfert et al. [8], we represent items, collections, and slates as vectors in a shared embedding space, and generate a sequence of slates by pivoting around item collections towards the target slate. Second, we use a dialog inpainting language model [5] to generate conversational user utterances that express preferences for each slate by prompting the model with metadata from corresponding item collections (Figure 1).

We use this approach to create a dataset of 10,000 synthetic music-seeking conversations that cover a breadth of domain expertise, from conversations about Japanese pop music to those about electro-swing music. In a qualitative evaluation, we find a majority of the data contain believable sequences of slates with user utterances that faithfully express preferences for them. We also evaluate this data quantitatively by measuring its impact on training conversational music retrievers. Our proposed approach yields up to a 23% relative Hits@10 gain compared to baselines trained on non-conversational and conversational datasets.

Related work. Traditional recommender systems [15,16] use matrix factorization techniques to personalize results from large quantities of user log data. In contrast, conversational or interactive recommender systems [3] can reduce their dependence on logs by allowing users to interact with them and provide direct feedback. However, because of a lack of conversational training data, conversational recommenders are often trained using reinforcement learning techniques [3,17,28,30] or using supervised learning on scripted dialogue flows [9]. A notable exception is the ReDial dataset [13], a large conversational movie recommendation dataset collected by paid experts in a Wizard-of-Oz setting. Our work also benefits from recent advances in conversational systems [4,6,26,27,28,29,29,29], retrieval modeling [13,25], and content modeling [10,11,19,21].

2 From Curated Item Collections to Item-Seeking Conversations

Curated item collections contain substantial domain expertise: they not only group items (songs) \( x \in X \) into coherent collections (playlists) \( z \subset X \), but also provide valuable metadata about each
collection φ(z) (playlist titles and descriptions). However, they lack two key features present in an item-seeking conversation: a sequence of slates of items s_t ⊂ X presented to the user in each turn t (instead of a fixed collection of items), and corresponding user utterances u_t that express a preference for each slate (instead of the non-conversational metadata). We address these problems by using a dataset of item collections Z to first generate sequences of slates and then generate user utterances for them.

**Generating sequences of slates.** We are guided by following desirable properties on the slates a user might see in an ideal conversation: (P.1) a slate, s_t, should maintain continuity with its previous turn’s s_{t-1} (if the user first asked for party music, they are more likely to ask for pop music than meditation music); (P.2) the change between subsequent slates should be coherent, corresponding to natural language feedback from the user—we expect that an item collection z_t approximates this change; and finally, (P.3) each turn should bring the user closer to their target slate s* (if the user ultimately wants good party music, we expect slates to include more high energy music in later turns).

To realize these properties, we follow Göpfert et al. [8] and represent items x, item collections z_t and slates s_t as vectors (x, z_t and s_t) in a shared embedding space \( \mathbb{R}^d \). We use P.1 and P.2 to model \( \tilde{s}_t \) as a linear combination: \( \tilde{s}_t = \alpha s_{t-1} + \beta z_t \), and use P.3 to choose \( \alpha \) and \( \beta \) to minimize the distance to \( s^* \).

We sample \( s_0 \) and \( \tilde{s}_t \) from Z to ensure that sequences start and end in coherent slates. Empirically, we found that sampling \( z_t \) from shrinking neighborhoods of \( s^* \) works well\(^4\). We plan to further explore this in future work. Finally, we can retrieve the items in \( s_t \) using the item neighbors of \( \tilde{s}_t \).

**Generating user utterances.** Once we have a sequence of slates, we need to generate corresponding user utterances: suppose that \( s_{t-1} \) contained “80s Sing-Alongs” songs and \( s_t \) “Power-Up Drum & Bass” songs; \( u_t \) should contain feedback like “Now drum and bass songs to get the party going”.

To achieve this, we use a dialog inpainter [5]—a T5-based language model trained to predict missing utterances in a conversation. We set up the conversation with system utterances that describe each slate \( s_t \) using the corresponding collection’s metadata \( \phi(z_t) \)—e.g., “I’ve added songs described as sick party drum and bass songs.”—and use a dialog inpainter to fill in the “missing” user utterances\(^5\).

**Application to conversational retrieval.** Our ultimate goal is to build a conversational item retriever where users can interact with the system by providing feedback over multiple turns. Given a user utterance \( u_t \) and the history of previous utterances and slates \( H_t = (u_1, s_1, \ldots, u_{t-1}, s_{t-1}) \), a conversational retriever predicts a new slate of items that ranks the user’s target \( s^* \) highest.\(^6\)

We model the task by learning a ranking function \( \rho_t : \mathcal{X} \rightarrow \mathbb{R} \) using a dual encoder architecture \([7, 13, 20]\), which has shown to be effective on similar tasks. Dual encoders independently embed queries \( q \) and targets \( x \) into dense vectors, and compute the ranking function using cosine similarity: \( \rho(x; q) = \text{embed}(x) \cdot \text{embed}(q) \). We follow prior work \([5, 25]\) and use item metadata to provide \((u_t, H_t)\) as a query, and sample items from \( s_t \) to be the target (see Appendix A.5 for model input details).

## 3 Evaluation

Our evaluation aims to validate two claims: (1) our approach generates high quality synthetic data, and (2) training on the synthetic data improves retrieval performance. We use our approach to generate 10,000 music-seeking conversations from 19,156 expert-curated playlists (Z) from a proprietary dataset. Each conversation consists of six turns, where each turn \( t \) has a user utterance \( u_t \) and slate \( s_t \).

### 3.1 Dataset Quality

We begin by qualitatively evaluating the synthetic data. One author manually rated over 100 turns on a three point scale according to the following questions: (1) how believable is the turn, with respect to the target?, (2) how well-phrased is the utterance?, and (3) how well does the utterance

---

\(^1\)\(\alpha \) and \(\beta \) can be solved in closed-form when the vectors in the embedding space are of unit-norm.

\(^2\)See Appendix A.1 for further details on the embedding space and sampling procedure.

\(^3\)See Appendix A.2 for examples of the templates used to generate the system utterances.

\(^4\)We assume that the user’s target is fixed throughout the conversation.

---
match their preference? We want sequences of slates that are realistic, user utterances that resemble those of real users, and user utterances that faithfully represent the preferences for the slates. We find that the majority of turns are high quality on each of the dimensions, suggesting that the synthetic data is sufficiently high quality to use for training data (Figure 2).

3.2 Retrieval Performance

We now measure the quantitative impact of the synthetic data when used to train conversational retrievers. We first describe the experimental setup. See Appendix C for more details.

Evaluation Dataset. As there is no existing data for the conversational music retrieval task, we evaluate on a test set of the synthetic data, which consists of 856 conversations. We report a standard retrieval metric, Hits@10, by comparing the song ranking at each turn with the target slate, excluding songs that have been seen in the history up to that turn. Hits@10 is 1 iff any of the top-10 retrieved songs are labeled relevant. All models use the same conversation history from the dataset, as opposed to building conversation history using model predictions from previous turns.

Model implementation. We initialize our dual encoder from a pretrained T5 1.1-Base checkpoint [23] and finetune on the synthetic data by randomly sampling a turn $t$ for each conversation, and using $q_t = (u_t, H_t)$ as a query and a randomly sampled song from $s_t$ as a target. We train the model using a contrastive loss with in-batch negatives. At inference time, we precompute the song embeddings and use nearest neighbor search to retrieve the top 10 songs for each query $q_t$.

Baselines. We compare against two dual encoders trained on variants of the query: NonConversational, where the query $q$ is the playlist title and seed songs from the playlist, and NoHistory, where query $q = u_t$ for a randomly sampled turn $t$. We also compare against two dual encoders trained on variants of the synthetic dataset: RandomSequence, where the slates $s_t$ are equal to a randomly sampled playlist $z_t$ at each turn $t$, and UserTemplates, where the user utterances are templated instead of generated by the inpainter. The training target is a randomly sampled song $x$ from the playlist (NonConversational) or from $s_t$ (all others). While the training sets differ between the baselines, all baselines are evaluated on the same set of queries, including the conversation history, at test time.

Figure 2 (right) compares our model to baselines. We observe that our model achieves up to a 23% relative gain over the best baseline on Hits@10 averaged over turns. The gains over the UserTemplates model suggest it is useful to train on conversational user utterances rather than templated user utterances. Moreover, the gains over the NoHistory model suggest that training with a conversation history is also useful. Overall, the results suggest that our model is better able to use natural language feedback to improve retrieval. As future work, we are interested in scaling up the qualitative evaluation of the synthetic data and evaluating the retrieval performance on real conversational item retrieval datasets.

4 Discussion

We introduced a general technique to convert item collections into synthetic conversations and demonstrated the benefits of building conversational retrieval models trained on such data for the
music domain. In the future, we plan to more extensively evaluate our models and explore how to incorporate user information—either from user logs or explicit user intent—to further personalize conversational retrieval systems. Finally, while language models are known to propagate the harmful biases in their training data [1], by synthetically generating data we have the opportunity to filter and mitigate these biases—this is an important direction for future work.

References


Appendix

A Data Generation Details

We provide additional implementation details for generating sequences of slates and generating user utterances described in Section 2.

A.1 Slate Sequence Generation

We generate six turns for each conversation using hyperparameters that we found generated reasonable sequences in preliminary experiments. We randomly sample the starting point $s_0$ from the neighborhood of the target $s^*$ using a neighborhood of size $k = 256$, using the softmax of similarity to the target as the sampling distribution. To ensure that the starting point is not too close to the target, we also require that $s_0$ is not in the top 50 neighbors of $s^*$. We then sample $z_t$ from the neighborhood of $s^*$, decreasing $k$ each turn until $k = 1$ on the last turn to ensure that the conversation reaches the user’s target slate. Specifically, we sample from neighborhood sizes of $\{128, 64, 32, 16, 1\}$ and use the softmax over the similarity to the previous turn’s slate $s_{t-1}$ as the sampling distribution to encourage each $z_t$ to be relevant to the previous slate. Finally, we sample the top 20 items that are neighbors of $z_t$ to retrieve the items in the slate $s_t$ for each turn. We are interested in understanding the sensitivity of our approach to these hyperparameters in future work.
Table 1: Examples of templates used to instantiate system utterances. We fill in $description$ with the description for playlist $z_t$. 

<table>
<thead>
<tr>
<th>Preference</th>
<th>System Utterance Template</th>
</tr>
</thead>
</table>
| More       | Of course! Let me also add some songs described as $description$. How can I improve the vibe?  
Got it, I will also add some songs described as $description$. Describe how I can improve the vibe.  
Sure, I will also add some songs described as $description$. How can I improve this playlist now?  
Definitely, let me also add more songs described as $description$. What else can I do to make this playlist better?  
Ok, let me go ahead and also add some songs described as $description$. How can I improve this playlist specifically? |
| Less       | Got it, let me remove some songs described as $description$. How would you like to improve this playlist now?  
Sure, I will remove some songs described as $description$. How can I improve the vibe?  
Ok, let me remove some songs described as $description$. What can I do to make this playlist better?  
Of course! I will remove some songs described as $description$. How can I improve this playlist now?  
Sounds good. Let me remove songs described as $description$. Describe how I can improve the vibe. |

A.2 Utterance Generation

After generating sequences of slates, we use a dialog inpainter to generate corresponding user utterances. The dialog inpainter takes as input a sequence of system utterances. We create the system utterances by filling in templates with metadata for each item collection $z_t$. We include examples of the templates used for the system utterances to generate music-seeking conversations in Table [1]. We support two types of preferences—more or less—which we determine based on the value of the weight $\beta$ in the linear combination (i.e., positive $\beta$ means more).

B Model Inputs

We describe how we represent the inputs to the non-conversational and conversational dual encoders.

B.1 Non-conversational dual encoder

We use a non-conversational dual encoder to learn the joint embedding space to generate the sequences of slates in Section [2]. We also use this model as one of the baselines in Section [1]. Here, we discuss how we represent the inputs, the query $q$ and the targets $x$, to the dual encoder. 

The query $q$ is a textual representation of an item collection $z$, generated using item collection metadata and item metadata for a sample of "seed" items in $z$. The target $x$ is a textual representation of a random sampled item in $z$, generated using the item metadata. 

Concretely, for music retrieval, the query input is as follows:

```
playlist_title [SEP] seed_song_title by artist_names from album_title
```

In our experiments, we use five seed songs and concatenate a description of each seed song to the end of the query with a separator token. Similarly, the target input is defined as:

```
song_title by artist_names from album_title
```
B.2 Conversational dual encoder

We now discuss how we represent the inputs to the conversational dual encoder. Recall that we define the conversation history at turn $t$ as $H_t = (u_1, s_1, \ldots, u_{t-1}, s_{t-1})$. For the conversational dual encoder, query $q \overset{\text{def}}{=} (u_t, H_t)$, where $u_t$ is the user utterance at turn $t$. The target $x$ is a sampled item from the slate $s_t$.

To construct the query, we observe that the user utterances $u$ are already text. We then represent slate $s$ textually using the metadata associated with the top-k items from $s$. Finally, we concatenate the textual representation of each term in $H_t$ with the utterance $u_t$. Note that we use reverse concatenation (i.e., the last turn occurs at the beginning of the query and the first turn occurs at the end of the query), so that the position of the last utterance $u_t$ is independent of the turn $t$. The targets use the same representation as the non-conversational model (Section B.1).

Specifically, for the conversational music retriever, the query input, at time $t$ is as follows:

$u_t$, [SEP] song_description$_{t-1}$, [SEP] utterance$_{t-1}$, ..., [SEP] song_description$_1$, [SEP] utterance$_1$

Where "utterance" represents the user utterance $u$ and "song_description" represents a textual description of the song sampled from the slate $s$ that matches the target input encoding. In our experiments we use the top 3 songs from the slate $s$ at each turn and concatenate the song descriptions with a separator token (shown with the top 1 song for space).

C Retrieval Experiment Details

We provide additional details for the retrieval results in Section 3.2.

C.1 Data preprocessing

One failure mode of the dialog inpainter is occasionally copying text from the input prompts. We want to reduce these cases as it can lead to user utterances that are less conversational and less realistic. To account for this, we filter the test split of the synthetic data to remove conversations that have a high substring overlap between a user utterance and the system utterances (i.e., longest common substring is greater or equal to 75 characters). This filters about 14% of conversations, resulting in a test set of 856 conversations.

C.2 Baselines

We include the templates used to instantiate the user utterances for the UserTemplates baseline in Table 2. We assign preferences as described in Appendix A.2.

<table>
<thead>
<tr>
<th>Preference</th>
<th>User Utterance Template</th>
</tr>
</thead>
</table>
| More       | Add some songs described as $description$.  
Can you also add some songs described as $description$.  
I want more songs that can be described as $description$.  
Can I have some songs described as $description$.  
More $description$. |
| Less       | Fewer songs described as $description$.  
Please no songs described as $description$.  
I don’t want songs described as $description$.  
Remove songs described as $description$.  
Less $description$. |

Table 2: Examples of templates used to instantiate user utterances for the UserTemplates baseline. We fill in $description$ with the description for playlist $z$. 


C.3 Training details

We finetune all models (including baselines) from a T5.1.1-Base checkpoint for 500k steps with constant learning rate 0.001, dropout rate 0.1, and batch size 512. We use TPUv3 chips to train the non-conversational model and TPUv4 chips to train our model and all other baselines.

C.4 Evaluation details

We use a query token length of 1024 tokens and a target token length of 128 tokens and preprocess the input as described in Section B.2 for all models. We use the validation set of the synthetic data to select the best checkpoint for our model and baselines. We run inference using the full retrieval corpus, and select the checkpoint that achieves the highest Hits@10 averaged over turns. We report Hits@10 results using the best checkpoint for each model over the test set of the synthetic data.