Learning Relevant Questions for Conversational Product Search using Reinforcement Learning

Ali Montazeralghaem
Center for Intelligent Information Retrieval
College of Information and Computer Sciences
University of Massachusetts Amherst
Amherst, MA
montazer@cs.umass.edu

James Allan
Center for Intelligent Information Retrieval
College of Information and Computer Sciences
University of Massachusetts Amherst
Amherst, MA
allan@cs.umass.edu

ABSTRACT

We propose RelQuest, a conversational product search model based on reinforcement learning to generate questions from product descriptions in each round of the conversation, directly maximizing any desired metrics (i.e., the ultimate goal of the conversation), objectives, or even an arbitrary user satisfaction signal. By enabling systems to ask questions about user needs, conversational product search has gained increasing attention in recent years. Asking the *right* questions through conversations helps the system collect valuable feedback to create better user experiences and ultimately increase sales. In contrast, existing conversational product search methods are based on an assumption that there is a set of effectively pre-defined candidate questions for each product to be asked. Moreover, they make strong assumptions to estimate the value of questions in each round of the conversation. Estimating the true value of questions in each round of the conversation is not trivial since it is unknown. Experiments on real-world user purchasing data show the effectiveness of RelQuest to generate questions that maximize standard evaluation measures such as NDCG.

1 INTRODUCTION

Customers search for products on Internet platforms such as Amazon and eBay to make purchases online. However, in most cases, search queries are imprecise and these platforms unsurprisingly fail to find the user's desired product on a first attempt. By enabling these platforms to ask clarifying questions about user needs, conversational product search has gained increasing attention in recent years [3, 36–38].

A conversation is initiated by the user's query and in each round of the conversation the system asks a question aiming to winnow out incorrect items, driving the search toward the user's target item. Ideally, every question is related to the need and clearly moves toward the target, but since the actual target item is unknown in the conversation, systems typically fall back on strategies that they hope accomplish the goal, strategies that to some degree assume that the user already knows all attributes of the target item and can easily answer questions about them.

In that context, existing conversational product search models typically work by deriving a "question" from a pre-defined database of attribute-value pairs: finding out that the user wants a "color=blue" item may discriminate between all possible targets that match the query. These systems then generally impose intuitive strategies for selecting questions based on the goal of dropping items with undesired attribute values. For instance, Zou et al. [37] applied Generalized Binary Search (GBS) to find an attribute that best splits the probability mass of predicted user preferences closest to two halves for the remaining of the products during each question. However, these strategies are not inherently in line with the ultimate goal of conversational product search, which increases the performance of retrieval at the end of the conversation.

To make an efficient conversation with the user, the system should be able to generate a relevant question in each round of the conversation aiming to maximize the retrieval performance at the end of the conversation. However, the true value for each question in each round of the conversation is unknown. This is because of the tension between immediate payoff and potentially delayed value of a question in a conversation. In other words, we should wait until the end of the conversation, when the system wins or loses (i.e., finding the target product or not), to assign a value for each asked question. Furthermore, different conversations can be designed for each product search based on different scenarios and strategies and the value of each question is based on questions the system asked before.

We present RelQuest, a conversational product search system that derives questions from the descriptions of top-ranked products and that learns a strategy for driving the conversation by directly maximizing any desired metric or objective. RelQuest uses reinforcement learning to learn both how to generate relevant questions and how to target the specified objective. In each round of the conversation, RelQuest first retrieves some products based on the initial query and subsequent question-answer pairs. Then, based on the ambiguity in the retrieved products, it generates a question to clarify user needs. Since RelQuest generates questions dependent on a conversation, we utilize a pretrained autoencoder as the retrieval model to find products considering exact matching, semantic matching, and order of terms simultaneously.

RelQuest's question generator is designed to be a policy network that takes the context of the conversation and retrieved products and generates a clarifying question. It has been shown that there are two types of deep models that can capture different levels

WSDM '22, February 21–25, 2022, Phoenix, AZ, USA

of matching in retrieval tasks [6]. Inspired by this, we design two types of deep policy networks: 1) Representation-focused, and 2) Interaction-focused policy network to make sure that our policy network has the ability to generate relevant questions in a conversation. Given the answer to the generated question, we include a discriminator to estimate the usefulness of that question as a reward function to guide the question generator. The workflow of RelQuest is shown in Figure 1. As a side-product of the way RelQuest generates questions, it has the ability to provide suggested answers to the user for each generated question that enables the user to choose one of these suggested answers.

An important advantage of RelQuest compared to the most existing methods is that it can be optimized for different evaluation metrics, such as average precision or normalized discounted cumulative gain (NDCG) [8]. In the ideal case, the reward function can be easily modeled by various user satisfaction signals.

The core contributions of this work are: 1) We design RelQuest, a reinforcement learning approach for generating relevant questions for conversational product search that optimizes arbitrary desired metrics or objectives; 2) RelQuest appears to be first attempt at generating questions from product descriptions in the conversational product search; 3) We propose an autoencoder to generate a code layer for each product that enables our approach to generate question dependent on a conversation's state; and, 4) We show that RelQuest outperforms competitive baselines using real-world user purchasing data.

2 RELATED WORK

2.1 Conversational Product Search and Recommendation

Belkin et al. [2] was a earliest work that proposed an interactive information retrieval system that utilized script-base conversational interaction for search. Product search in e-commerce is another research area [21]. This area has gained much more popularity recently with the advent of intelligent conversational systems and the process of neural approaches in the natural language process (NLP). Yang et al [33, 34] proposed an approach to predict the next question in conversations. A Multi-Memory Network (MMN) architecture for conversational search and recommendation was proposed by Zhang et al. [36]. However, their model can predict the questions in the training which is not compatible with the nature of the conversational recommendation system. A Belief Tracker model was developed by Sun et al. [28] to derive facet-value pairs from user utterances during the conversation. To decide between asking a question or recommending an item, they also proposed a policy network. However, depending on the target item, their model does not predict a particular value for each question and they have simply considered a constant reward for each question. Recently, Lei et al. [14] showed that the interaction between conversation and recommendation can improve the performance of these systems substantially. They also proposed a policy network to decide between asking a question or recommending an item. However, their model cannot predict a specific value for each question based on the target item and they simply considered a constant reward for each question. Zou et al. [37] proposed a question-based recommendation method that is able to ask users to express their preferences over descriptive item features. However, they assumed that there is

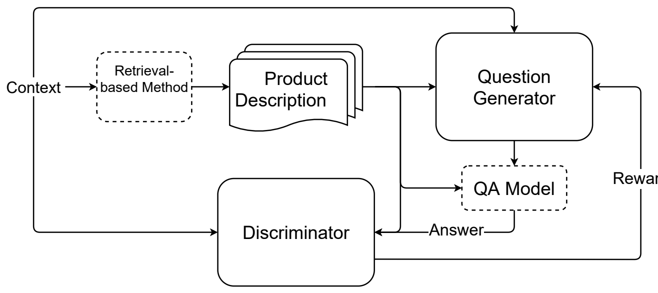


Figure 1: The workflow of RelQuest to generate questions.

a question pool for each product. Moreover, in each round of the conversation, they applied Generalized Binary Search (GBS) to find the entity that best splits the probability mass of predicted user preferences closest to two halves for the rest of the products. This assumption is not in line with the goal of the conversation product search which is improving retrieval performance.

2.2 Deep Reinforcement Leaning

Reinforcement learning (RL) is a machine learning approach that optimizes an agent's actions in relation to the desired reward [29]. RL algorithms have shown an impressive potential for tackling a wide range of complex tasks, from game playing [26] to robotic manipulation [19], due to the advancement of deep learning. One of the most famous achievements of deep RL is Google's DeepMind research on the game of Go [25, 27].

The agent and the environment are the key roles in RL. The environment is the world that is visible to the agent and that the agent can interact with. The agent sees observations in the setting at every stage of interaction and takes an action based on these observations and then depending on his actions, the agent earns a reward. The reward is a measure to show how good or bad are the actions taken by the agent. The agent's final goal is to maximize the cumulative reward.

3 METHODOLOGY

In this section, we outline the problem setup to generate questions for conversational product search and then lay out our RelQuest approach to solve this problem. Our approach consists of two main components: 1) a question generator; 2) and a discriminator. Figure 1 is an overview of our approach.

3.1 Problem Statement and Motivation

Let $Q_0 = \{q_1, q_2, ..., q_m\}$ be a query with m terms issued by a user u to initiate a conversation with the system, and $P = \{p_1, p_2, ..., p_k\}$ be a set of k products.

In each round of the conversation, the system asks a question about user needs and then according to the answer of the question, we update the initial query Q_0 as a context of the conversation c. For the first round of the conversation, the context is equal to the initial query $c = Q_0$. Given a context in each round of the conversation, we initially retrieve a ranked list of products $R = \{p_1, p_2, ..., p_N\} \subset P$ by a retrieval system (section 3.4) that considers exact and semantic matching simultaneously to increase the performance of the system [6, 17]. Then, we feed this ranked list to a generator module (section 3.2) to generate a clarifying question Q that maximize the *ultimate goal* of the product search. Given the generated question, a QA model (section 3.5) answers the generated question by considering the target product p_u. Then, we use a discriminator module (section 3.3) that gets the context, the answer of the question, and a ranked list of products and outputs a reward signal to evaluate the generated question. The output of the discriminator is treated as a reward to update the question generator parameters using REINFORCE algorithm [32].

In this task, the reward is measured based on the performance of the retrieval. The reward can be designed based on different levels of relevance, ranging from algorithmic and topical relevance to motivational relevance. The reward function can be replaced with any desired retrieval metric function such as average precision (AP), and normalized discounted cumulative gain (NDCG) [8]. In the ideal case, it can measure user satisfaction signals captured from user interaction with the system or questionnaires. In these cases, the reward function is non-differentiable. Reinforcement learning (RL) is an effective approach to maximize non-differentiable metrics through policy gradient.

3.2 A Reinforcement Learning Approach for Question Generator

The question generator gets the context c and the output of the retrieval model R to generate a clarifying question. In each round of the conversation, the question generator produces a question that directly maximizes the ultimate goal of the product search i.e., optimizing the evaluation metrics or any desired objective. Note that the ground truth for each question in each conversation is unknown. This is because different conversations can be designed for each product search based on different scenarios. In other words, we should wait until the end of the conversation, when the system wins or loses (i.e., finding the target product or not), to assign a value for each asked question. Therefore, we cannot use supervised learning for this task and we utilize reinforcement learning to train a question generator that maximizes a reward function.

In reinforcement learning, there is an agent that takes an action in each round of the conversation and gets a reward signal from an environment (e.g., discriminator). The reward is a number that tells the agent how good or bad was the taken action. The agent tries to figure out the best actions to take or the optimal way to behave in the environment in order to carry out his task in the best possible way to gain more reward. In this setting, our system would be able to generate questions that ultimately maximize the evaluation metrics. In the following, we describe how we model an agent as our question generator, the action of the agent, the state, and the reward function.

Agent: The agent in this task is a question generator that takes the state in each round of the conversation, which includes the current context c of the conversation and the output of the retrieval model for a specific user and generate a clarifying question.

State: At the first of the conversation, the state is equal to the context, which is the initial query $c_0 = Q_0$, and top retrieved products R_c for a specific user u as follows:

$$s_0 = (c_0, R_{c_0}, u). \tag{1}$$

In each round of the conversation, given the generated question Q by the agent, we update the context by the answer to the question

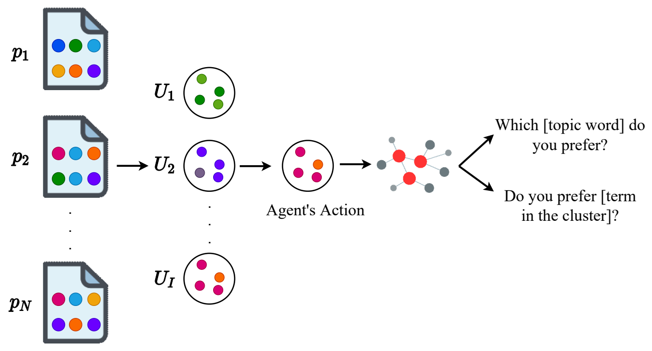

p_1
p_2
p_N
U_1
U_2
U_I
Agent's Action
Which [topic word] do you prefer?
Do you prefer [term in the cluster]?

Figure 2: The agent selects one of the clusters over product descriptions and then generates a clarification question.

which is generated by the QA model (i.e., $\text{QA}(Q, p_u)$). So, the state in the following rounds of the conversation is $s_t = (c_t, R_{c_t}, u)$ where

$$c_t = c_{t-1} + \text{QA}(Q, p_u) \tag{2}$$

and p_u is a knowledge source about user preferences (i.e., the target product).

Reward: Given a state s_t and action Q_t, the reward $R(s_t, Q_t)$ is a number that is generated by the environment to guide the question generator. The reward is computed based on the evaluation function and any desired objective by the discriminator which is described in section 3.3.

Agent's Action: Question generation is based on the amount of ambiguity in the retrieved products. In other words, there should be some ambiguous aspects in the retrieved products that the agent needs to generate questions about them to clarify. To do this, given the current context, we first pick the top retrieved products in each round of the conversation and then cluster terms in their description as shown in Figure 2. Each cluster is supposed to be an aspect that the agent can ask about it. Therefore, by giving the current context and top retrieved products, the agent's action is to sample a cluster from existing clusters. Then, a question can be generated according to the selected cluster.

A question is a combination of interrogatives, topic words, and ordinary words [31]. We find a topic word for the selected cluster and use two question templates to generate the final format as follows:

- Do you prefer [term in the cluster]?
- Which [topic word] do you prefer?

The first template belongs to the "yes/no" questions [37] in case that the selected cluster has just one term. The second template is about user preference over a topic word. To find a topic word for the selected cluster, we utilize WordNet [5] which is a lexical database of semantic relations between words. If we find more than one topic word for the selected cluster, we choose one that has semantic relationships with the majority of terms in the cluster. Also, if we did not find any topic word in the WordNet, we choose one of the terms in the selected cluster that has semantic relationships with the majority of terms in the cluster by using cosine similarity of embedding vectors of terms. One can use other resources to find more accurate topic words for a cluster of terms [4, 15]. For the second template, we can show the terms in the selected cluster as suggested answers to the user. For example, a selected cluster can be "Samsung, iPhone, Huawei". In this case, generated question in a round of the conversation is "Which brand do you prefer?" where "brand" is a topic word for the selected cluster and suggested answers are: "Samsung, iPhone, Huawei". Note that the user's answer may be different from the terms in the suggestion. We use K-Means clustering which is very popular and widely used in a variety of applications to cluster terms in the product descriptions (see section 4.3.2 for more details).

Policy Network: A policy is a rule used by an agent to decide what actions to take. In each round of the conversation, the input of the policy network is the current state and the agent should predict a probability distribution over the clusters. According to this probability distribution, the agent can take an action which is choosing a cluster. We explain how we design the policy network's input, architecture, and output in the rest of this section.

Loss Function and Optimization: Let H be the number of asked questions in a conversation. Our objective is to find a policy θ that create a sequence of questions τ in a conversation with the user to find the target product as follows:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G_\tau] \tag{3}$$

where G_τ is a discounted sum of rewards. The discounted return from time t, G_t, is the discounted sum of rewards starting from time t:

$$G_t = \sum_{k=0}^{H-1} \gamma^k R_{t+k}(s_{t+k}, Q_{t+k}) \tag{4}$$

where γ, $0 \le \gamma \le 1$ is the discount rate and $R(s_t, Q_t)$ is the reward function which is defined in section 3.3. An advantage of using the discount rate is that we can wait until the end of the conversation and then assign a value for every individual-generated question based on whether the agent finds the target product or not.

We need to model a policy that generates a sequence of questions that maximize the total rewards $\theta^* = \arg\max_\theta J(\theta)$. We use REINFORCE algorithm [32] to compute policy gradients as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) G], \tag{5}$$

where the parameters of the policy network can be updated as $\theta^t \longleftarrow \theta^{t-1} + \alpha \nabla_\theta J(\theta)$ where α is the learning rate.

3.3 Discriminator

Discriminator takes the answer of the generated question, product descriptions, and the current context and returns a reward as a signal that tells the question generator how good or bad was the generated question.

To generate relevant questions we consider two metrics. The first metric is related to the level of ability of the discriminator to distinguish between the target product and other products. Specifically we evaluate the quality of the conversation at the end of the conversation. A conversation will be ended when either the agent finds the target product (i.e., the rank of the target product is 1) or the number of asked questions equals the maximum number of questions that the agent can ask. So, we define a reward function for the first metric as follows:

$$R_1(s_t, Q_t) = \begin{cases} \text{eval}(u, c_t, \text{QA}(Q_t, p_u), R) & \text{If conversation ended} \\ 0 & \text{Otherwise} \end{cases}$$

where $\text{eval}(u, c_t, \text{QA}(Q_t, p_u), R)$ measures the performance of the retrieval. This function can be replaced with any desired retrieval metric function such as average precision (AP), and normalized discounted cumulative gain (NDCG) [8]. Note that we compute the value of each question in the conversation by Eq. (4). For example, consider that we can retrieve the target product at rank 1 at the end of the conversation. In this case, $R_1(s_t, Q_t)$ is equal to 1 and we assign a discounted positive value for each generated question in the conversation by Eq. (4).

The second metric measures the difference between the answer of the generated question and the previous answers in the conversation history. In other words, we want to make sure that the answer of the generated question reveals as much as possible unknown information about the target products. Therefore, we use the informativeness metric proposed by Qi et.al [20] as follows:

$$R_2(s_t, Q_t) = 1 - \max_{0<k<t} \text{Prec}(\text{QA}(Q_t, p_u), \text{QA}(Q_k, p_u)) \tag{6}$$

where Prec(., .) represents unigram precision between the answer of the generated question and a previously given answer in the conversation. Intuitively, if the answer of the question has more overlap with any of the previously revealed answers, then this answer has less information about the target product.

Finally, the reward for a generated question can be computed as a linear combination of two objectives as follows:

$$R(s_t, Q_t) = R_1(s_t, Q_t) + R_2(s_t, Q_t). \tag{7}$$

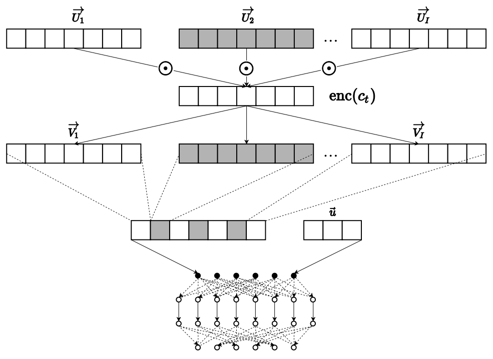
3.4 Retrieval System with an Autoencoder

The quality of the generated question depends on the products retrieved by RelQuest's retrieval system. Therefore, we need to build a retrieval system that considers *exact and semantic matching* between the context of the conversation and product descriptions [6]. More importantly we need to generate question *dependently* in a conversation. In other words, we want to generate a question while it depends on the previously generated questions. That means our retrieval system should be able to consider the order of words in the context of the conversation in the ranking. In summary, three constraints should be met by our retrieval system: 1) exact matching; 2) semantic matching; and 3) the order of word in the context [7]. To satisfy these constraints, we use two separate components in our retrieval system: 1) an autoencoder to consider semantic matching and the order of terms; and 2) a conventional retrieval model to boost exact matching in the final ranking.

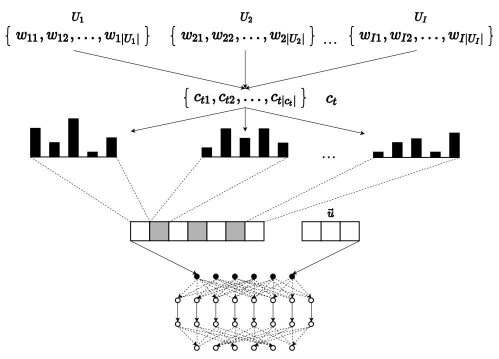
3.4.1 Pretrained Autoencoder. Autoencoders are a type of self-supervised learning model to learn a low-dimensional representation (i.e., code layer) of sequence data, aiming to remember the order and semantic meaning of words.

An autoencoder consists of an encoder, a code layer, and a decoder. Given a product description $p_i = \{w_1, w_2, ..., w_T\}$, we convert each word w_i to an embedding vector $\vec{w_i} = EM[w_i] \in R^{1 \times D}$, where EM is a pretrained word embedding matrix. This process produces a sequence of embedding vectors for each product $\vec{p_i} = \{\vec{w_1}, \vec{w_2}, ..., \vec{w_T}\}$.

Given these embedding vectors, we use a Bidirectional Long Short Term Memory (BiLSTM) as our encoder to obtain a dense



(a) Representation-focused policy network.



U_1
U_2
U_I
$\{w_{11}, w_{12}, \ldots, w_{1|U_1|}\}$
$\{w_{21}, w_{22}, \ldots, w_{2|U_2|}\}$
$\{w_{I1}, w_{I2}, \ldots, w_{I|U_I|}\}$
$\{c_{t1}, c_{t2}, \ldots, c_{t|c_t|}\}$
c_t
$\vec{u}$

(b) Interaction-focused policy network.

Figure 3: Two types of deep policy networks.

representation (code layer) enc(.) for each product as follows:

$$\text{enc}(p_i) = \text{BiLSTM}_{\text{enc}}(\vec{p_i}). \quad (8)$$

Likewise, we employ another BiLSTM as our decoder that takes the output of the encoder (i.e., encoding representation) and reconstructs the original data as follows:

$$\text{dec}(p_i) = \text{BiLSTM}_{\text{dec}}(\text{enc}(p_i)). \quad (9)$$

where $\text{dec}(p_i)$ is the output of the decoder. The goal of decoding is to confirm whether the encoding representation is valid.

We train the autoencoder by minimizing the difference between the output of the decoder $\text{dec}(p_i)$ and the input of the encoder p_i using cross-entropy loss function [18]. When the training of the autoencoder is completed, we only use the encoder to convert each product or query to a dense representation.

Although the autoencoder has the ability to detect exact matching in the retrieval, we found that the model can get better performance when we boost autoencoder results by a conventional retrieval model (RM) like TF-IDF, BM25, or KL-divergence retrieval module [13]. So, given a context c and a product description p_i, we compute the retrieval score as:

$$\text{score}(c, p_i) = \cos(\text{enc}(c), \text{enc}(p_i)) \times \text{RM}(c, p_i) \quad (10)$$

where $\cos(.,.)$ is the cosine similarity function between two representations and $\text{RM}(.,.)$ is a conventional retrieval model (TF-IDF function in our experiments). Therefore, given the context in each round of the conversation, we retrieve the products by Eq. (10).

3.5 QA Model

Given the generated question in each round of the conversation, we need to answer this question by using user simulation. Recent work [28, 36–38] assume that the user knows all attributes of the target item and will respond to the questions with full knowledge. In more details, the user will respond with "yes" if there is a predefined aspect in the target item description and "no" if the aspect is absent. So, this approach can be used for "yes/no" questions. However, our model can generate another type of questions (see section 3.2). So, instead of using this rule, we utilize a question answering model (i.e., QA model) to answer the generated question Q by using the knowledge source about user preference p_u. To do this, we utilize a pretrained QA model which is proposed by Yu et al. [35]. They proposed QANet which is an end-to-end machine-reading and question answering model. We utilize QANet in our experiments since this model is both accurate and fast in inference. Instead of using a recurrent neural network (RNN), QANet uses two components: 1) convolution which models local interactions and 2) self-attention to model global interactions. Given the generated question Q and description of the target product p_u, QANet outputs an answer A.

3.6 Policy Network

Policy network's input: The input of the policy network is the state $s_t = (c_t, R_{c_t}, u)$ where c_t is the context of the conversation at timestamp t. We retrieve N top products based on their scores computed by a retrieval system $R_{c_t} = \{p_1, ..., p_N\}$. Then we collect all the words in the description of these products in one set $W_{c_t} = \{w_1, w_2, ..., w_M\}$. This words set will be used to generate I clusters by K-means clustering algorithm $\{U_1, U_2, ..., U_I\} = \text{K-means}(W_{c_t})$. Then, these clusters will be fed to the policy network with the context of the conversation.

Policy network's architecture: Given the clusters and the current context, we design a deep policy network to select one of the clusters. Inspired by the literature [6, 16] we propose two types of policy network: 1) Representation-focused, and 2) Interaction-focused.

3.6.1 *Representation-focused policy network.*

In the representation-focused policy network, we aim to learn a representation for each cluster according to the context representation. In this case, the agent can select the best action based on the similarity between representatives of the clusters and the context. The architecture of the representation-focused policy network is depicted in Figure 3a.

In representation-focused policy network each cluster has a representation which is the average of embedding vectors of its words $\vec{U_i} = \frac{\sum_{w \in U_i} \vec{w}}{|U_i|}$. To compare the representation of each cluster with context, we compute the context representation by using the encoder (i.e., Eq. (8)) in the autoencoder. This will help the model to produce different representations based on the order of words in the context.

In the next step, we compute element-wise multiplication of each cluster representation and context representation $\vec{V_i} = \vec{U_i} \odot \text{enc}(c_t)$. This is important to consider a representation for each user to ask the personalized question in the conversation [36]. Therefore, we utilize an embedding vector $\vec{u_i}$ for each user u_i. Finally, the input of the representation-focused policy network is the concatenation of all cluster representation and the user embedding as:

$$\psi(s_t) = [\vec{V_1}||\vec{V_2}||...||\vec{V_I}||\vec{u}]. \tag{11}$$

3.6.2 Interaction-focused policy network. Inspired by deep relevance matching model (DRMM) [6], we design an interaction-focused policy network that models the interaction between clusters and the context by using matching histogram mapping. The architecture of the interaction-focused policy network is depicted in Figure 3b. In this model, for each cluster we consider a matching histogram with 5 bins $\{[-1, -0.5), [-0.5, 0), [0, 0.5), [0.5, 1), [1, 1]\}$. Then for each term in a cluster and the context, we compute the cosine similarity between their vectors and accumulate the count of local interactions in each bin. In this case, each cluster can be displayed by a fixed-length (e.g., 5) vector. Since the range of local interactions in each bin can be different, we need to normalize the count value of each bin. For this reason, we apply logarithm over the count value in each bin [6].

After computing the fixed-length vectors for all clusters as described above, the input of the interaction-focused policy network would be the concatenation of these vectors and the user embedding same as Eq. (11).

The interaction-focused policy network can distinguish between exact matching signals and semantic similarity matching signals by assigning a separate bin to the exact matching (i.e, last bin $[1, 1]$).

Policy network's output: The output of the policy network for both architectures (i.e., the representation-focused and interaction-focused) is designed with a feed-forward neural network which is composed of : 1) the input layer $\vec{z_0} = \psi(s_t)$, $l-1$ hidden layer, and the output layer $\vec{z_l}$. Each hidden layer $\vec{z_i}$ is a fully-connected layer $\vec{z_i} = \varphi(W_i.\vec{z_{i-1}} + b_i), \quad 1 \leq i \leq l-1$, where φ is a non-linear activation function. The output layer $\vec{z_l}$ also is a fully-connected layer, but for the output layer we used a softmax function as an activation function. The output of the softmax function is a probability distribution over possible outputs (i.e., all clusters). Therefore, if we represent the parameters of the policy network by θ, the probability of a cluster given the current state is computed as $\pi_\theta(U_i|s_t) = \frac{e^{z_{l_i}}}{\sum_{j=1}^{I} e^{z_{l_j}}}$.

4 EXPERIMENTS

4.1 Datasets

Following previous work on this task [1, 36], we use the Amazon product dataset. This dataset contains millions of products and customers and rich metadata such as descriptions of products, multi-level product descriptions, categories, and reviews for products[1]. There are 24 sub-datasets of different product types. In this study, we use *Cell Phones & Accessories*, *Health & Personal Care*, and *Movies & TV* in our experiments. Table 1 shows the basic statistics of these three datasets. We randomly select 70% of data for each user in the training/validation set and keep the other 30% to make the test set.

Initial Query Construction. Following Zhang et al. [3, 36], we used a three-step paradigm of product search to construct the initial request Q_0 for each user u which purchased a item p_u: 1) extract the multi-level category of information of product from the metadata, 2) concatenate the terms in this information, and 3) remove stopwords and duplicate words. All the queries associated with the purchased item can be considered as the initial query which is issued by the user. The initial queries do not reveal the specific information of the purchased items. Examples of queries are "health personal care dietary supplement vitamin", "cell phone accessory international charger", "tv movies" in each category.

[1]https://nijianmo.github.io/amazon/index.html

4.2 Baselines

We compare RelQuest with three groups of baselines: 1) word based retrieval, 2) embedding based retrieval, and 3) conversational based retrieval. In particular the baselines are: (1) **BM25** [22]: An effective and widely-used retrieval method to rank items based on term frequency, inverse document frequency of query terms and item description length; (2) **Rocchio** [23]: An approach to form a new query by maximizing its similarity to relevant items and minimizing its similarity to non-relevant items. BM25 [22] function is used for weighting terms; (3) **MultiNeg** [9]: An approach to increase the performance of the product search using non-relevant results from multiple negative models; (4) **LSE** [30]: The latent semantic entity which is a non-personalized product search model; (5) **HEM** [1]: The hierarchical embedding model which is a personalized product search approach; (6) **AVLEM** [3]: A paradigm for conversation product search based on negative feedback. AVLEM identifies users' preferences by showing results and collecting feedback on the aspect-value pairs of the non-relevant items; (7) **Qrec** [37]: A question-based recommendation method which directly queries users on the automatically extracted entities in relevant documents. We set the number of questions in this model to 5 same as RelQuest; and (8) **Sem**: This is similar to our model, but with the difference that in each round of the conversation we pick the cluster that is most similar to the current context. We compute the similarity between each cluster and the context by using cosine similarity between vectors of their words.

4.3 Experimental and parameter setting

4.3.1 ***Evaluation Measures.*** For evaluating the performance of the models, we use mean reciprocal rank (MRR) and normalized discounted cumulative gain (NDCG) at 10 [8]. Note that we assume that for each conversation we have a target item. Therefore, in this case, MRR is equal to mean average precision (MAP). Statistically significant differences of performance are determined using two-tailed paired t-test at 95% confidence level ($p\_value < 0.05$). We tuned all hyper-parameters on the validation set.

4.3.2 ***Parameter Setting.*** We implemented and trained RelQuest using Tensorflow [2]. Our code is available at http://suppressed-for-review. The parameters of the policy network in the question generator are trained with Adam optimizer [10] according to the back-propagation algorithm [24]. The learning rate in our experiments was selected from $[1e-3, 5e-4, 1e-5]$. We set the batch size to 8 since in reinforcement learning larger batch size can reduce GPU utilization. For the policy network, we use 3 hidden layers with 3000, 1000, and 10 hidden units. For the first two, we use Rectified Linear Units (ReLU) as an activation function and for the last one,

[2]https://www.tensorflow.org/

Table 1: Basic statistics of the experimental datasets, where l(Request) is the average length of initial requests.

Dataset	#Users	#Items	#Reviews	#Queries	#l(Request)	#User-Query pairs (Training/Testing)
Health & Personal Care	38,609	18,534	346,355	779	8.25±2.16	231,186/282
Cell Phones & Accessories	27,879	10,429	194,439	165	5.93±1.57	114,177/665
Movies & TV	123,960	50,052	1,697,524	248	5.31±1.61	241,436/5,209

Table 2: Comparison of proposed models (trained to maximize NDCG as a first part of the reward function) and baselines. Thegraphics superscript ▲ indicates that the improvements over all baselines are statistically significant.

Model Type	Model Name	Health & Personal Care		Cell Phones & Accessories		Movies & TV	
		MRR	NDCG	MRR	NDCG	MRR	NDCG
Word Based Retrieval	BM25	0.055	0.053	0.065	0.077	0.009	0.008
	Rocchio	0.055	0.053	0.065	0.077	0.009	0.009
	MultiNeg	0.046	0.048	0.062	0.076	0.015	0.016
Embedding Based Retrieval	LSE	0.157	0.195	0.098	0.084	0.025	0.027
	HEM	0.189	0.201	0.115	0.116	0.030	0.030
Conversational Based Retrieval	AVLEM	0.260	0.305	0.154	0.177	0.035	0.038
	Sem	0.289	0.398	0.242	0.262	0.059	0.103
	Qrec	0.296	0.419	0.214	0.238	0.070	0.127
Our Approach	RelQuest-Rep	0.311▲	0.466▲	**0.312**▲	**0.326**▲	0.106▲	0.170▲
	RelQuest-Int	**0.333**▲	**0.497**▲	0.281▲	0.301▲	**0.118**▲	**0.185**▲

we use the softmax activation function to generate a probability distribution over the output of the policy network. The discount factor in Eq. (4) was set to 0.99 since it has been shown this value works well in the reinforcement learning [11]. For each user, we consider an embedding vector with a size of 100 which is randomly initialized and learned in the training. The number of clusters in the K-Means algorithm was set to 10. To use the K-Means algorithm we need to have an embedding vector for each term. To train embedding vectors we used the word2vec algorithm by skip-gram strategy. The size of embedding vectors was set to 100. For each dataset, we used review data to train the word2vec algorithm. The reason is users often criticize or praise different aspects of a product in their reviews. Training embedding vectors over review data helps the K-means algorithm to detect clusters more appropriately. The maximum number of questions that the agent can ask was set to 5 and we used 10 top documents in each round of the conversation to generate questions. We truncated each product description to have 100 words at most. Also, all stopwords are removed from queries and product descriptions.

Autoencoder training. For each dataset, we trained an autoencoder and use it in the question generation process. The parameters of autoencoder are trained with Adam optimizer. The learning rate is set to 0.001 and for each encoder and decoder, we use a 1-layer bidirectional LSTM (BiLSTM) with 50 hidden units. After 50 epochs we saved the encoder and use it as a part of our retrieval model.

4.4 Results and Discussion

4.4.1 ***Comparison with the Baselines.*** In the first experiments, we evaluate RelQuest with two type of deep policy networks i.e., RelQuest-Rep and RelQuest-Int against baselines. Note that in this experiment, we use NDCG@10 in our reward function for training RelQuest.

The results of this experiment are reported in Table 2. The first observation is that the results of word based retrieval models are worse than other baselines. BM25 model cannot achieve high performance since there are no significant correlations between user purchases and the term matching between queries and product descriptions [1, 30]. The performance of the Rocchio and Multi-Neg also can show that even feedback information does not help because of low performance in the retrieved items. LSE and HEM are able to detect semantic matching between words. According to the results in Table 2, semantic matching improves the performance. Therefore, in RelQuest, we consider semantic matching by learning a code layer for each product which is able to remember semantic meaning and order of words.

HEM outperforms LSE in all cases since HEM is a personalized product search approach and as discussed by previous studies [1, 3] personalized question generation enables models to perform better. RelQuest also takes advantage of the personalized generation since we learn an embedding vector for each specific user. Therefore, RelQuest is capable of asking personalized questions based on the knowledge of the user.

All conversational product search models outperform word and embedding based retrieval models. This shows the importance of the interaction with users in the product search. AVLEM is a conversation product search driven by non-relevant items. According to the result, AVLEM outperforms LSE and HEM, since AVLEM lets the system asks questions of non-relevant results.

Qrec is an approach that learns the informativeness of questions based on GBS algorithm. However, this algorithm is not in line with the goal of the product search. According to the results, RelQuest can achieve better results in terms of MRR and NDCG. The reason is that RelQuest optimizes directly NDCG as a part of our reward function. By maximizing NDCG in RelQuest, other related metrics also increase, but in our opinion, NDCG is a more important metric for this task since we need to show top retrieved products to the user. Therefore, in RelQuest, we tried to maximize this metric.

Both versions of RelQuest outperform Sem in all cases. This shows the importance of selecting the best cluster in each round of the conversation aiming to maximize the performance of the conversation at the *end* of it. This is because the Sem model just selects

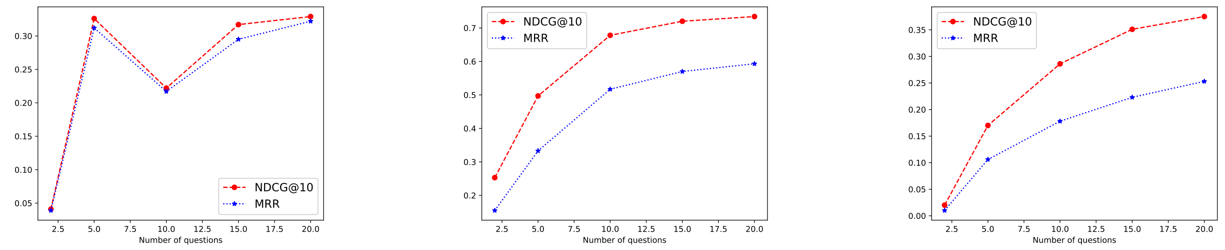


Figure 4: Impact of the number of questions on the performance of RelQuest, Cell Phones & Accessories (left), Health & Personal Care (middle), and Movies & TV (right).

the most similar cluster to the context in each round regardless of the performance of the conversation at the end of it.

We also observe that in two datasets interaction-focused policy network achieves better performance compared to the representation focused version. The reason could be related to the number of parameters in these two types of policy networks. The number of parameters in the interaction-focused policy network is less than the representation-focused policy network. Also, the interaction-focused policy network can capture the relation between terms in the cluster and the context individually. However, in the representation-focused policy network, we use the representation of the cluster (i.e., the average of embedding vectors of its words). However, we see different observations in Cell phones & Accessories dataset. The reason is related to the number of queries in this dataset which is less than two other datasets. In this case, the representation-focused policy network can achieve better performance since this network can remember more interaction between users and the agent.

4.4.2 *Analysis of the Number of Asked questions.*

In this experiment, we let our model to generate different number of questions to see the performance of the system by asking more and fewer questions. We set the maximum number of questions in our system to $\{2, 5, 10, 15, 20\}$. Figure 4 shows the result of this experiment in three datasets and with two metrics. According to this figure, in two datasets ("Health & Personal Care" and "Movies & TV"), by asking more questions the preference of the system increases. These results are reasonable since we are collecting more feedback from users. However, the performance of the system in the first dataset ("Cell Phones & Accessories") decreases after asking 5 more questions. The reason can be related to the number of queries in this dataset. According to Table 1, the number of queries in this dataset is less than two other datasets. However, in this dataset also the system recovers the performance after asking 5 more questions.

Note that although increasing the number of questions will increase the efficiency of the model, but this will definitely reduce user satisfaction. Therefore, an interesting future research diction is improving the performance of the system by asking a few questions.

4.4.3 *Analysis of Cumulative Reward and Evaluation Metric.*

To show the learning curve of the question generator in RelQuest, we depicted the cumulative reward that the question generator has earned in the training in Figure 5 (left). In this experiment, we just report the results of the cell phones & accessories dataset for the sake of space. However, we had similar observations for other datasets. According to this figure, The learning process is divided into two parts. In the first part, the question generator learns to earn

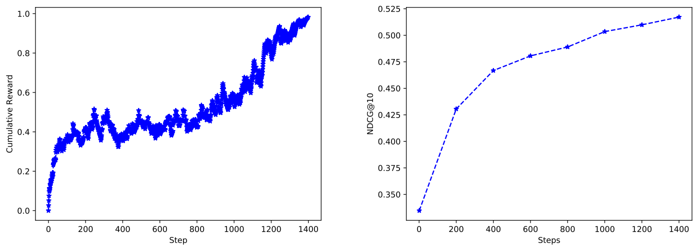


Figure 5: The cumulative reward over training (left), The value of NDCG@10 in the training as a part of reward function (right).

easy rewards in the environment after some steps. This observation is compatible with the results of playing a game by an agent [12]. In the second part of the learning, the agent starts to earn more difficult rewards by increasing the steps.

We also want to make sure that the agent is maximizing the evaluation metric as a part of our reward function in the training. For this reason, we depicted NDCG@10, which we used as a part of our reward function, in Figure 5 (right). According to this figure, the question generator is trying to enrich the context of the conversation by asking the right questions in the conversation which increases NDCG@10 in the training. The performance of the question generator depends on the performance of the retrieval model and the QA model. Proposing more accurate models for these two will cause that the question generator can earn more rewards.

By comparing Figure 5 in the left and right we can see the effect of the informativeness metric in the reward function. In other words, if we want to generate a type of question, we just need to use our objective in the reward function in an appropriate way. Then, the model tries to maximize this reward function.

5 CONCLUSIONS AND FUTURE WORK

We have presented RelQuest, a conversational product search model based on reinforcement learning to generate questions from product descriptions in each round of the conversation, questions chosen to maximize any desired metric that reflects the ultimate the goal of the conversation. We described an autoencoder based process for retrieving candidate products at each step of the conversation. We showed that by using this autoencoder we can boost the retrieval performance and this will help the question generator to earn more rewards in the training. The core of RelQuest is generating questions from the current state of the conversation and a set of candidate products. The goal is to find a question that disambiguates the set of products in a way that optimizes the target metric or objective. We show that RelQuest is able to maximize a combination of objectives as a reward function. Overall, we have shown that RelQuest is a successful and more realistic approach to conversational product search. It does not depend upon a manually curated database of possible questions (though could use them) and learns to pose questions based on any target metric that makes sense in context.

6 ACKNOWLEDGEMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and also in part by Amazon.com. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] Qingyao Ai, Yongfeng Zhang, Keping Bi, Xu Chen, and W Bruce Croft. 2017. Learning a hierarchical embedding model for personalized product search. In *SIGIR'17*. 645–654.
[2] Nicholas J Belkin, Colleen Cool, Adelheit Stein, and Ulrich Thiel. 1995. Cases, scripts, and information-seeking strategies: On the design of interactive information retrieval systems. *Expert systems with applications* 9, 3 (1995), 379–395.
[3] Keping Bi, Qingyao Ai, Yongfeng Zhang, and W Bruce Croft. 2019. Conversational product search based on negative feedback. In *CIKM'19*. 359–368.
[4] Haw-Shiuan Chang, Ziyun Wang, Luke Vilnis, and Andrew McCallum. 2017. Distributional inclusion vector embedding for unsupervised hypernymy detection. *arXiv preprint arXiv:1710.00880* (2017).
[5] Christiane Fellbaum. 2010. WordNet. In *Theory and applications of ontology: computer applications*. Springer, 231–243.
[6] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM international on conference on information and knowledge management*. 55–64.
[7] Ayyoob Imani, Amir Vakili, Ali Montazer, and Azadeh Shakery. 2019. An Axiomatic Study of Query Terms Order in Ad-hoc Retrieval. In *European Conference on Information Retrieval*. Springer, 196–202.
[8] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.
[9] Maryam Karimzadehgan and ChengXiang Zhai. 2011. Improving retrieval accuracy of difficult queries through generalizing negative document language models. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 27–36.
[10] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR'15* (San Diego, CA, USA).
[11] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.
[12] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. 2020. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4501–4510.
[13] John Lafferty and Chengxiang Zhai. 2001. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. 111–119.
[14] Wenqiang Lei, Xiangnan He, Yisong Miao, Qingyun Wu, Richang Hong, Min-Yen Kan, and Tat-Seng Chua. 2020. Estimation-action-reflection: Towards deep interaction between conversational and recommender systems. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 304–312.
[15] Berenike Litz, Hagen Langer, and Rainer Malaka. 2009. Sequential supervised learning for hypernym discovery from Wikipedia. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*. Springer, 68–80.
[16] Zhengdong Lu and Hang Li. 2013. A deep architecture for matching short texts. *Advances in neural information processing systems* 26 (2013), 1367–1375.
[17] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. 2017. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 257–266.
[18] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* (2017).
[19] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. 2017. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073* (2017).
[20] Peng Qi, Yuhao Zhang, and Christopher D Manning. 2020. Stay hungry, stay focused: Generating informative and specific questions in information-seeking conversations. *arXiv preprint arXiv:2004.14530* (2020).
[21] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.
[22] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*. Springer, 232–241.
[23] J. J. Rocchio. 1971. Relevance feedback in information retrieval. In *The Smart retrieval system - experiments in automatic document processing*, G. Salton (Ed.). Englewood Cliffs, NJ: Prentice-Hall, 313–323.
[24] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533.
[25] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.
[26] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.
[27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354–359.
[28] Yueming Sun and Yi Zhang. 2018. Conversational recommender system. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 235–244.
[29] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[30] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. 2016. Learning latent vector spaces for product search. In *CIKM'16*. 165–174.
[31] Yansen Wang, Chenyi Liu, Minlie Huang, and Liqiang Nie. 2018. Learning to ask questions in open-domain conversational systems with typed decoders. *arXiv preprint arXiv:1805.04843* (2018).
[32] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
[33] Liu Yang, Minghui Qiu, Chen Qu, Jiafeng Guo, Yongfeng Zhang, W Bruce Croft, Jun Huang, and Haiqing Chen. 2018. Response ranking with deep matching networks and external knowledge in information-seeking conversation systems. In *SIGIR'18*. 245–254.
[34] Liu Yang, Hamed Zamani, Yongfeng Zhang, Jiafeng Guo, and W Bruce Croft. 2017. Neural matching models for question retrieval and next question prediction in conversation. *arXiv preprint arXiv:1707.05409* (2017).
[35] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541* (2018).
[36] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W Bruce Croft. 2018. Towards conversational search and recommendation: System ask, user respond. In *CIKM'18*. 177–186.
[37] Jie Zou, Yifan Chen, and Evangelos Kanoulas. 2020. Towards question-based recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 881–890.
[38] Jie Zou and Evangelos Kanoulas. 2019. Learning to ask: Question-based sequential Bayesian product search. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 369–378.