Caching Historical Embeddings in Conversational Search

OPHIR FRIEDER, Georgetown University
IDA MELE, IASI-CNR
CRISTINA IOANA MUNTEAN, ISTI-CNR
FRANCO MARIA NARDINI, ISTI-CNR
RAFFAELE PEREGO, ISTI-CNR
NICOLA TONELLOTTO, University of Pisa

Rapid response, namely low latency, is fundamental in search applications; it is particularly so in interactive search sessions, such as those encountered in conversational settings. An observation with a potential to reduce latency asserts that conversational queries exhibit a temporal locality in the lists of documents retrieved. Motivated by this observation, we propose and evaluate a client-side document embedding cache, improving the responsiveness of conversational search systems. By leveraging state-of-the-art dense retrieval models to abstract document and query semantics, we cache the embeddings of documents retrieved for a topic introduced in the conversation, as they are likely relevant to successive queries. Our document embedding cache implements an efficient metric index, answering nearest-neighbor similarity queries by estimating the approximate result sets returned. We demonstrate the efficiency achieved using our cache via reproducible experiments based on TREC CAsT datasets, achieving a hit rate of up to 75% without degrading answer quality. Our achieved high cache hit rates significantly improve the responsiveness of conversational systems while likewise reducing the number of queries managed on the search back-end.

CCS Concepts: • **Information systems** → **Specialized information retrieval**; **Search engine architectures and scalability**.

Additional Key Words and Phrases: conversational search, similarity search, caching, dense retrieval

1 INTRODUCTION

Conversational agents, fueled by language understanding advancements enabled by large contextualized language models, are drawing considerable attention [1, 34]. Multi-turn conversations commence with a main topic and evolve with differing facets of the initial topic or an abrupt shift to a new focus, possibly suggested by the content of the answers returned [4, 19].

A user drives such an interactive information-discovery process by submitting a query about a topic followed by a sequence of more specific queries, possibly aimed at clarifying some aspects of the topic. Documents relevant to the first query are often relevant and helpful in answering subsequent queries. This suggests the presence of temporal locality in the lists of results retrieved by conversational systems for successive queries issued by the same user in the same conversation. In support of this claim, Figure 1 illustrates a t-SNE [27] bi-dimensional visualization of dense representations for the queries and the relevant documents of five manually rewritten

Authors' addresses: Ophir Frieder, Georgetown University, ophir@ir.cs.georgetown.edu; Ida Mele, IASI-CNR, ida.mele@iasi.cnr.it; Cristina Ioana Muntean, ISTI-CNR, cristina.muntean@isti.cnr.it; Franco Maria Nardini, ISTI-CNR, francomaria.nardini@isti.cnr.it; Raffaele Perego, ISTI-CNR, raffaele.perego@isti.cnr.it; Nicola Tonellotto, University of Pisa, nicola.tonellotto@unipi.it.

conversations from the TREC 2019 CAsT dataset [4]. As illustrated, there is a clear spatial clustering among queries in the same conversation, as well as a clear spatial clustering of relevant documents for these queries.

We exploit locality to improve efficiency in conversational systems by caching the query results on the client side. Rather than caching pages of results answering queries likely to be resubmitted, we cache documents about a topic, believing that their content will be likewise relevant to successive queries issued by the user involved in the conversation. Topic caching is effective in Web search [20] but, as yet, was never explored in conversational search.

Topic caching effectiveness rests on topical locality. Specifically, if the variety of search domains is limited, the likelihood that past, and hence potentially cached, documents are relevant to successive searches is greater. In the Web environment, search engines respond to a wide and diverse set of queries, and yet, topic caching is still effective [20]; thus, in the conversational search domain where a sequence of searches often focuses on a related if not on the same specific topic, topical caching, intuitively, should have even greater appeal than in the Web environment, motivating our exploration.
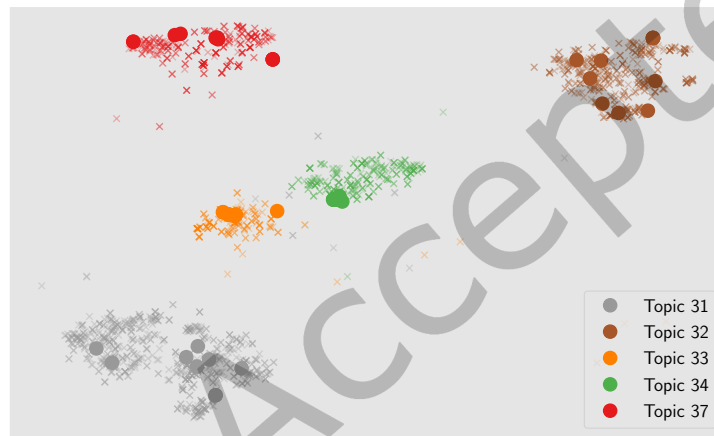

Fig. 1. 2D visualization of conversational queries (•) and corresponding relevant documents (×) for 5 CAsT 2019 topics.

To capitalize on the deep semantic relationship between conversation queries and documents, we leverage recent advances in Dense Retrieval (DR) models [10, 12, 29, 33, 35]. In our DR setting, documents are represented by low-dimension learned embeddings stored for efficient access in a specialised metric index, such as that provided by the FAISS toolkit [11]. Given a query embedded in the same multi-dimensional space, online ranking is performed by means of a top-k nearest neighbor similarity search based on a metric distance. In the worst-case scenario, the computational cost of the nearest neighbor search is directly proportional to the number of documents stored in the metric index. To improve end-to-end responsiveness of the system, we insert a *client-side* metric cache [6, 8] in front of the DR system aimed at reusing documents retrieved for previous queries in the same conversation. We investigate different strategies for populating the cache at cold start and updating its content as the conversation topic evolves.

Our metric cache returns an approximate result set for the current query. Using reproducible experiments based on TREC CAsT datasets, we demonstrate that our cache significantly reduces end-to-end conversational system processing times without answer quality degradation. Typically, we answer a query without accessing the document index since the cache already stores the most similar documents. More importantly, we can estimate the quality of the documents present in the cache for the current query, and based on this estimate, decide if querying

the document index is potentially beneficial. Depending on the size of the cache, the hit rate measured on the CAsT conversations varies between 65% and 75%, illustrating that caching significantly expedites conversational search by drastically reducing the number of queries submitted to the document index on the back-end.

Our contributions are as follows:

- Capitalizing on temporal locality, we propose a client-side document embedding cache C for expediting conversational search systems;
- We innovate means that assess current cache content quality necessitating document index access only needed to improve response quality;
- Using the TREC CAsT datasets, we demonstrate responsiveness improvement without accuracy degradation.

The remainder of the paper is structured as follows: Section 2 introduces our conversational search system architecture and discusses the proposed document embedding cache and the associated update strategies. Section 3 details our research questions, introducing the experimental settings and the experimental methodology. Results of our comprehensive evaluation conducted to answer the research questions are discussed in Section 4. Section 5 contextualizes our contribution in the related work. Finally, we conclude our investigation in Section 6.
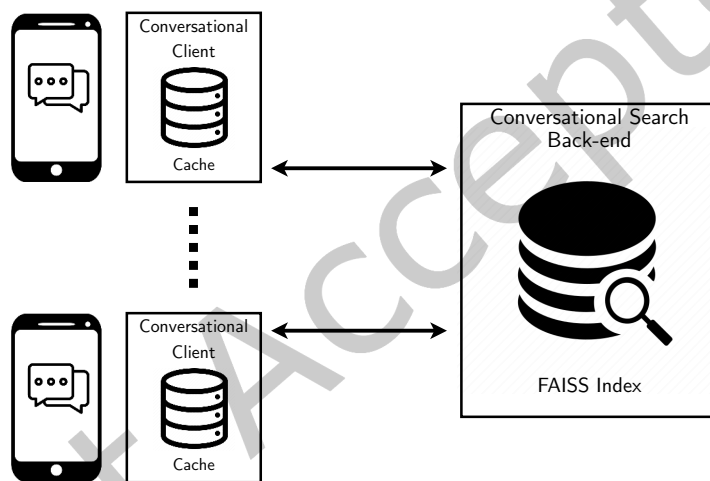

Fig. 2. Architecture of a conversational search system with client-side caching.

2 A CONVERSATIONAL SYSTEM WITH CLIENT-SIDE CACHING

A conversational search system enriched with our client-side caching is depicted in Figure 2. We adopt a typical client-server architecture where a client supervises the conversational dialogue between a user and a search back-end running on a remote server.

We assume that the conversational back-end uses a dense retrieval model where documents and queries are both encoded with vector representations, also known as embeddings, in the same multi-dimensional latent space; the collection of document embeddings is stored, for efficient access, in a search system supporting nearest neighbor search, such as a FAISS index [11]. Each conversational client, possibly running on a mobile device, deals with a single user conversation at a time, and hosts a local cache aimed at reusing, for efficiency reasons, the documents previously retrieved from the back-end as a result of the previous utterances of the ongoing conversation. Reusing previously retrieved, namely cached, results eliminates the additional index access,

reducing latency and resource load. Specifically, the twofold goal of the cache is: 1) to improve user-perceived responsiveness of the system by promptly answering user utterances with locally cached content; 2) to reduce the computational load on the back-end server by lowering the number of server requests as compared to an analogous solution not adopting client-side caching.

In detail, the client handles the user conversation by semantically enriching those utterances that lack context [19] and encoding the rewritten utterance in the embedding space. Online conversational search is performed in the above settings by means of top k nearest neighbor queries based on a metric distance between the embedding of the utterance and those of the indexed documents. The conversational client likewise queries the local cache or the back-end for the most relevant results answering the current utterance and presents them to the requesting user. The first query of a conversation is always answered by querying the back-end index, and the results retrieved are used to populate the initially empty cache. For successive utterances of the same conversation, the decision of whether to answer by leveraging the content of the cache or querying the remote index is taken locally as explained later. We begin by introducing the notation used, continuing with a mathematical background on the metric properties of queries and documents, and with a detailed specification of our client-side cache together with an update policy based on the metric properties of query and document embeddings.

2.1 Preliminaries

Each query or document is represented by a vector in $\mathbb{R}^l$, hereinafter called an *embedding*. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ be a collection of n documents represented by the embeddings $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, where $\phi_i = \mathcal{L}(d_i)$ and $\mathcal{L} : \mathcal{D} \to \mathbb{R}^l$ is a learned representation function. Similarly, let q_a be a query represented by the embedding $\psi_a = \mathcal{L}(q_a)$ in the same multi-dimensional space $\mathbb{R}^l$.

Similarity functions to compare embeddings exist including inner product [12, 24, 29, 35] and euclidean norm [13]. We use STAR [35] to encode queries and documents. Since STAR embeddings are fine-tuned for maximal inner-product search, they cannot natively exploit the plethora of efficient algorithms developed for searching in Euclidean metric spaces.

To leverage nearest neighbor search and all the efficient tools devised for it, maximum inner product similarity search between embeddings can be adapted to use the Euclidean distance. Given a query embedding $\psi_a \in \mathbb{R}^l$ and a set of document embeddings $\Phi = \{\phi_i\}$ with $\phi_i \in \mathbb{R}^l$, we apply the following transformation from $\mathbb{R}^l$ to $\mathbb{R}^{l+1}$ [2, 22]:

$$\bar{\psi}_a = \begin{bmatrix} \psi_a^T / \|\psi_a\| & 0 \end{bmatrix}^T, \quad \bar{\phi}_i = \begin{bmatrix} \phi_i^T / M & \sqrt{1 - \|\phi_i\|^2 / M^2} \end{bmatrix}^T, \tag{1}$$

where $M = \max_i \|\phi_i\|$. In doing so, the maximization problem of the inner product $\langle \psi_a, \phi_i \rangle$ becomes exactly equivalent to the minimization problem of the Euclidean distance $\|\bar{\psi}_a - \bar{\phi}_i\|$. In fact, we have:

$$\min \|\bar{\psi}_a - \bar{\phi}_i\|^2 = \min \left(\|\bar{\psi}_a\|^2 + \|\bar{\phi}_i\|^2 - 2\langle \bar{\psi}_a, \bar{\phi}_i \rangle \right) = \min \left(2 - 2\langle \psi_a, \phi_i / M \rangle \right) = \max \langle \psi_a, \phi_i \rangle.$$

Hence, hereinafter we consider the task of online ranking with a dense retriever as a nearest neighbor search task based on the Euclidean distance among the transformed embeddings $\bar{\psi}$ and $\bar{\phi}$ in $\mathbb{R}^{l+1}$. Intuitively, assuming $l = 2$, the transformation (1) maps arbitrary *query and document* vectors in $\mathbb{R}^2$ into unit-norm *query and document* vectors in $\mathbb{R}^3$, i.e., the transformed vectors are mapped on the surface of the unit sphere in $\mathbb{R}^3$.

To simplify the notation we drop the bar symbol from the embeddings $\bar{\psi} \to \psi$ and $\bar{\phi} \to \phi$, and assume that the learned function $\mathcal{L}$ encodes queries and documents directly in $\mathbb{R}^{l+1}$ by also applying the above transformation.

2.2 Nearest neighbor queries and metric distances

Let δ be a *metric distance function*, $\delta : \mathbb{R}^{l+1} \times \mathbb{R}^{l+1} \to \mathbb{R}$, measuring the Euclidean distance between two embeddings in $\mathbb{R}^{l+1}$ of valid documents and queries; the smaller the distance between the embeddings, the more similar the corresponding documents or queries are.

Given a query q_a, we are interested in retrieving $\text{NN}(q_a, k)$, i.e., the k Nearest Neighbor documents to q_a query according to the distance function $\delta(\cdot, \cdot)$. In the metric space $\mathbb{R}^{l+1}$, $\text{NN}(q_a, k)$ identifies an hyperball $\mathcal{B}_a$ centered on $\psi_a = \mathcal{L}(q_a)$ and with radius r_a, computed as:

$$r_a = \max_{d_i \in \text{NN}(q_a, k)} \delta(\psi_a, \mathcal{L}(d_i)). \tag{2}$$

The radius r_a is thus the distance from q_a of the least similar document among the ones in $\text{NN}(q_a, k)$[1].

We now introduce a new query q_b. Analogously, the set $\text{NN}(q_b, k)$ identifies the hyperball $\mathcal{B}_b$ with radius r_b centered in ψ_b and including the k embeddings closest to ψ_b. If $\psi_a \neq \psi_b$, the two hyperballs can be completely disjoint, or may partially overlap. We introduce the quantity:

$$\hat{r}_b = r_a - \delta(\psi_a, \psi_b), \tag{3}$$

to detect the case of a partial overlap in which the query embedding ψ_b falls within the hyperball $\mathcal{B}_a$, i.e., $\delta(\psi_a, \psi_b) < r_a$, or, equivalently, $\hat{r}_b > 0$, as illustrated[2] in Figure 3.
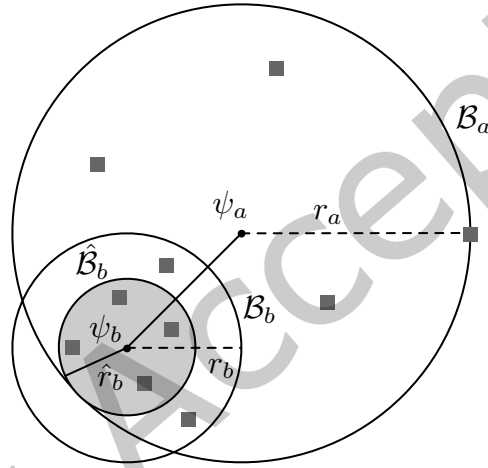

Fig. 3. Overlapping hyperballs for $\text{NN}(q_a, 10)$ and $\text{NN}(q_b, 6)$ with embeddings in $\mathbb{R}^2$. Grey squares represent the embeddings of the 10 nearest neighbor documents to q_a.

In this case, there always exists a hyperball $\hat{\mathcal{B}}_b$, centered on ψ_b with radius $\hat{r}_b$ such that $\hat{\mathcal{B}}_b \subset \mathcal{B}_a$. As shown in the figure, some of the documents in $\text{NN}(q_a, k)$, retrieved for query q_a, may belong also to $\text{NN}(q_b, k)$. Specifically, these documents are all those within the hyperball $\hat{\mathcal{B}}_b$. Note that there can be other documents in $\mathcal{B}_a$ whose embeddings are contained in $\mathcal{B}_b$, but if such embeddings are in $\hat{\mathcal{B}}_b$, we have the *guarantee* that the corresponding documents are the most similar to q_b among all the documents in $\mathcal{D}$ [6]. Our experiments will show that the documents relevant for successive queries in a conversation overlap significantly. To take advantage of such overlap, we now introduce a cache for storing historical embeddings that exploits the above metric properties of dense representations of queries and documents. Given the representation on the current utterance, the proposed cache aims at reusing the embeddings already retrieved for previous utterances of the same conversation for

---

[1]Without loss of generality, we assume that the least similar document is unique, and we do not have two or more documents at distance r_a from q_a.

[2]The figure approximates the metric properties in a local neighborhood of ψ_a on the $(l+1)$-dimensional unit sphere, i.e., in its locally-Euclidean l-dimensional tangent plane.

improving the responsiveness of the system. In the simplistic example depicted in Figure 4, our cache would answer query q_b by reusing the embeddings in $\mathcal{B}_b$ already retrieved for q_a.

2.3 A metric cache for conversational search

Since several queries in a multi-turn conversation may deal with the same broad topic, documents retrieved for the starting topic of a conversation might become useful also for answering subsequent queries within the same conversation. The properties of nearest neighbor queries in metric spaces discussed in the previous subsection suggest a simple, but effective way to exploit temporal locality by means of a metric cache C deployed on the client-side of a conversational DR system.

Algorithm 1: The CACHE pseudo-code

Input : a metric index $\mathcal{M}$, a metric cache C, a query cutoff k, a cache cutoff k_c, a query embedding ψ
Output: a results set $\mathcal{R}$

1 **if** Empty(C) **or** LowQuality(ψ, C) **then**
2     $\mathcal{R} \leftarrow \text{NN}(\mathcal{M}, \psi, k_c)$
3   Insert($C, \mathcal{R}$)
4 $\mathcal{R} \leftarrow \text{NN}(C, \psi, k)$
5 **return** $\mathcal{R}$

Our system for CAChing Historical Embeddings (CACHE) is specified in Algorithm 1. The system receives a sequence of queries belonging to a user conversation and answers them returning k documents retrieved from the metric cache C or the metric index $\mathcal{M}$ containing the document embeddings of the whole collection.

When the conversation is initiated with a query q, whose embedding is ψ, the cache is empty (line 1). The main index $\mathcal{M}$, possibly stored on a remote back-end server, is thus queried for top $\text{NN}(\mathcal{M}, \psi, k_c)$ documents, with cache cutoff $k_c \gg k$ (line 2). Those k_c documents are then stored in the cache (line 3). The rationale of using a cache cutoff k_c much larger than the query cutoff k is that of filling the cache with documents that are likely to be relevant also for the successive queries of the conversation, i.e., possibly all the documents in the conversation clusters depicted in Figure 1. The cache cutoff k_c relates in fact with the radius r_a of the hyperball $\mathcal{B}_a$ illustrated in Figure 3: the larger k_c the larger r_a and the possibility of having documents relevant to the successive queries of the conversation in the hyperball $\mathcal{B}_a$. When a new query of the same conversation arrives, we estimate the quality of the historical embeddings stored in the cache for answering it. This is accomplished by the function LowQuality(ψ, C) (line 1). If the results available in the cache C are likely to be of low quality, we issue the query to the main index $\mathcal{M}$ with cache cutoff k_c and add the top k_c results to C (line 2-3). Eventually, we query the cache for the k nearest neighbor documents (line 4), and return them (line 5).

Cache quality estimation. The quality of the historical embeddings stored in C for answering a new query is estimated heuristically within the function LowQuality(ψ, C) called in line 1 of Algorithm 1. Given the embedding ψ of the new query, we first identify the query embedding ψ_a closest to ψ among the ones present in C, i.e.,

$$\psi_a = \arg\min_{\psi_i \in C} \delta(\psi_i, \psi) \tag{4}$$

Once ψ_a is identified, we consider the radius r_a of the hyperball $\mathcal{B}_a$, depicted in Figure 1, and use Eq. 3 to check if ψ falls within $\mathcal{B}_a$. If this happen, it is likely that some of the documents previously retrieved for ψ_a and stored in C are relevant even for ψ. Specifically, our quality estimation heuristics considers the value $\hat{r} = r_a - \delta(\psi_a, \psi)$ introduced in Eq. 3. If $\hat{r} > \epsilon$, with $\epsilon \geq 0$ being a hyperparameter of the cache, we answer ψ with the k nearest

neighbor documents stored in the cache, i.e., the NN(C, ψ, k) documents; otherwise, we query the main embedding index in the conversational search back-end and update the cache accordingly. This quality test has the advantage of efficiency; it simply requires computing the distances between ψ and the embeddings of the few queries previously used to populate the cache for the current conversation, i.e., the ones that caused a cache miss and were answered by retrieving the embeddings from the back-end (lines 2 and 3 of Algorithm 1).

In addition, by changing the single hyperparameter ϵ that measures the distance of a query from the internal border of the hyperball containing the closest cached query, we can easily tune the quality-assessment heuristic for the specific needs. In the experimental section, we propose and discuss a simple but effective technique for tuning ϵ to balance the effectiveness of the results returned and the efficiency improvement introduced with caching.

3 RESEARCH QUESTIONS AND EXPERIMENTAL SETTINGS

We now present the research questions and the experimental setup aimed at evaluating the proposed CACHE system in operational scenarios. That is, we experimentally assess both the accuracy, namely not hindering response quality, and efficiency, namely a reduction of index request time, of a conversational search system that includes CACHE. Our reference baseline is exactly the same conversational search system illustrated in Figure 2 where conversational clients always forward the queries to the back-end server managing the document embedding index.

3.1 Research Questions

Specifically, in the following we address the following research questions:

- **RQ1**: Does CACHE provide effective answers to conversational utterances by reusing the embeddings retrieved for previous utterance of the same conversation?
  A. How effective is the quality assessment heuristic used to decide cache updates?
  B. To what extent does CACHE impact on client-server interactions?
  C. How much memory CACHE requires in the worst case?
- **RQ2**: How much does CACHE expedite the conversational search process?
  A. What is the impact of the cache cutoff k_c on the efficiency of the system in case of cache misses?
  B. How much faster is answering a query from the cache rather than from the remote index?

3.2 Experimental settings

Our conversational search system uses STAR [35] to encode CAsT queries and documents as embeddings with 769 dimensions[3]. The document embeddings are stored in a dense retrieval system leveraging the FAISS library [11] to efficiently perform similarity searches between queries and documents. The nearest neighbor search is exact, and no approximation/quantization mechanisms are deployed.

Datasets and dense representation. Our experiments are based on the resources provided by the 2019, 2020, and 2021 editions of the TREC Conversational Assistant Track (CAsT). The CAsT 2019 dataset consists of 50 human-assessed conversations, while the other two datasets include 25 conversations each, with an average of 10 turns per conversation. The CAsT 2019 and 2020 include relevance judgements at passage level, whereas for CAsT 2021 the relevance judgments are provided at the document level. The judgments, graded on a three-point scale, refer to passages of the TREC CAR (Complex Answer Retrieval), and MS-MARCO (MAchine Reading

---

[3]STAR encoding uses 768 values but we added one dimension to each embedding by applying the transformation in Eq. 1.

- FILL

COmprehension) collections for CAsT 2019 and 2020, and to documents of MS-MARCO, KILT, Wikipedia, and Washington Post 2020 for CAsT 2021[4].

Regarding the dense representation of queries and passages/documents, our caching strategy is orthogonal w.r.t. the choice of the embedding. The state-of-the-art single-representation models proposed in the literature are: DPR [12], ANCE [29], and STAR [35]. The main difference among these models is how the fine-tuning of the underlying pre-trained language model, i.e., BERT, is carried out. We selected for our experiments the embeddings computed by the STAR model since it employs hard negative sampling during fine-tuning, obtaining better representations in terms of effectiveness w.r.t. ANCE and DPR. For CAsT 2019 and 2020, we generated a STAR embedding for each passage in the collections, while for CAsT 2021, we encoded each document, up to the maximum input length of 512 tokens, in a single STAR embedding.

Given our focus on the efficiency of conversational search, we strictly use manually rewritten queries, where missing keywords or mentions to previous subjects, e.g., pronouns, are resolved by human assessors.

CACHE Configurations. To answer our research questions, we measure the end-to-end performance of the proposed CACHE system on the three CAsT datasets. We compare CACHE against the efficiency and effectiveness of a *baseline* conversational search system with no caching, always answering the conversational queries by using the FAISS index hosted by the back-end (hereinafter indicated as *no-caching*). The effectiveness of no-caching on the assessed conversations of the three CAsT datasets represents an upper bound for the effectiveness of our CACHE system. Analogously, we consider the no-caching baseline always retrieving documents via the back-end as a lower bound for the responsiveness of the conversational search task addressed.

We experiment with two different versions of our CACHE system:

- a *static*-CACHE: a metric cache populated with the k_c nearest documents returned by the index for the first query of each conversation and never updated for the remaining queries of the conversations;
- a *dynamic*-CACHE: a metric cache updated at query processing time according to Alg. 1, where LowQuality(ψ_b, C) returns false if $\hat{r}_b \geq \epsilon$ (see Eq. 3) for at least one of the previously cached queries, and true otherwise.

We vary the cache cutoff k_c in $\{1K, 2K, 5K, 10K\}$ and assess its impact. Additionally, since conversations are typically brief, e.g., from 6 to 13 queries for the three CAsT datasets considered, for efficiency and simplicity of design, we forgo implementing any space-freeing, eviction policy should the client-side cache reach maximum capacity. We assess experimentally that, even without eviction, the amount of memory needed by our *dynamic*-CACHE to store the embeddings of the documents retrieved from the FAISS index during a single conversation suffices and does not present an issue. In addition to the document embeddings, we recall that to implement the LowQuality($\cdot, \cdot$) test, our cache records also the embeddings ψ_a and radius r_a of all the previous queries q_a of the conversation answered on the back-end.

Effectiveness Evaluation. The effectiveness of the *no-caching* system, the static-CACHE, and the dynamic-CACHE are assessed by using the official metrics used to evaluate CAsT conversational search systems [4]: mean average precision at query cutoff 200 (MAP@200), mean reciprocal rank at query cutoff 200 (MRR@200), normalized discounted cumulative gain at query cutoff 3 (nDCG@3), and precision at query cutoffs 1 and 3 (P@1, P@3). Our experiments report the statistically significant differences w.r.t. the baseline system for $p < 0.01$ according to the two-sample t-test.

In addition to these standard IR measures, we introduce a new metric to measure the quality of the approximate answers retrieved from the cache w.r.t. the correct results retrieved form the FAISS index. We define the *coverage* of a query q w.r.t. a cache C and a given query cutoff value k, as the intersection, in terms of nearest neighbor documents, between the top k elements retrieved for the cache C and the exact top k elements retrieved from the
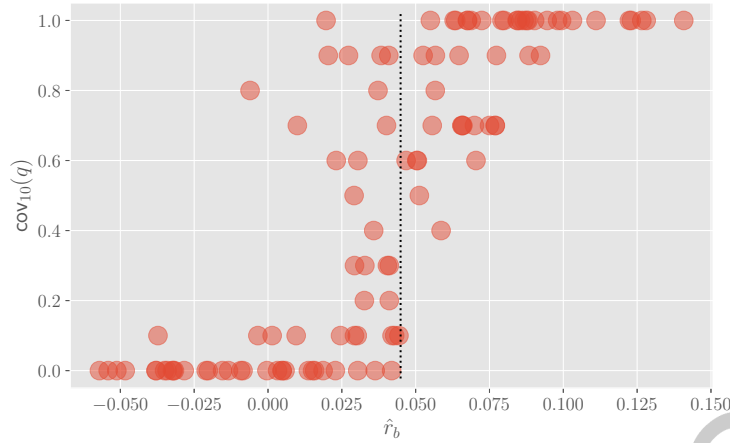
---

[4]https://www.treccast.ai/

Fig. 4. Correlation between $\hat{r}_b$ vs. $\text{cov}_{10}(q)$ for the CAsT 2019 train queries, using static-CACHE, $k = 10$ and $k_c = 1K$. The vertical black dashed line corresponds to $\hat{r}_b = 0.04$, the tuned cache update threshold value ϵ used in our experiments.

whole index $\mathcal{M}$, divided by k:

$$\text{cov}_k(q) = \frac{|\text{NN}(C, \psi, k) \cap \text{NN}(\mathcal{M}, \psi, k)|}{k}, \tag{5}$$

where ψ is the embedding of query q. We report on the quality of the approximate answers retrieved from the cache by measuring the coverage cov_k, averaged over the different queries. The higher cov_k at a given query cutoff k is, greater is the quality of the approximate k nearest neighbor documents retrieved from the cache. Of course $\text{cov}_k(q) = 1$ for a given cutoff k and query q means that we retrieve from the cache or the main index exactly the same set of answers. Moreover, these answers come out to be ranked in the same order by the distance function adopted. Besides measuring the quality of the answers retrieved from the cache vs the main index, we use the metric cov_k also to tune the hyperparameter ϵ.

To this end, Figure 4 reports the correlation between $\hat{r}_b$ vs. $\text{cov}_{10}(q)$ for the CAsT 2019 train queries, using static-CACHE and $k_c = 1K$. The queries with $\text{cov}_{10} \leq 0.3$, i.e., those with no more than three documents in the intersection between the static-CACHE contents and their actual top 10 documents, correspond to $\hat{r}_b \leq 0.04$. Hence, in our initial experiments, we set the value of ϵ to 0.04 to obtain good coverage figures at small query cutoffs. In answering RQ1.A we will also discuss a different tuning of ϵ aimed at improving the effectiveness of dynamic-CACHE at large query cutoffs.

Efficiency Evaluation. The efficiency of our CACHE systems is measured in terms of: i) *hit rate*, i.e., the percentage of queries, over the total number of queries, answered directly by the cache without querying the dense index; ii) *average query response time* for our CACHE configurations and the *no-caching* baseline. The hit rate is measured by not considering the first query in each conversation since each conversation starts with an empty cache, and the first queries are thus compulsory cache misses, always answered by the index. Finally, the query response time, namely latency, is measured as the amount of time from when a query is submitted to the system to the time it takes for the response to get back. To better understand the impact of caching, for CACHE we measure separately the average response time for hits and misses. The efficiency evaluation is conducted on a server equipped with an Intel Xeon E5-2630 v3 CPU clocked at 2.40GHz and 192 GiB of RAM. In our tests, we employ the FAISS[5] Python API v1.6.4. The experiments measuring query response time are conducted by

---
[5]https://github.com/facebookresearch/faiss

using the low-level C++ exhaustive nearest-neighbor search FAISS APIs. We perform this choice to avoid possible overheads introduced by the Python interpreter that comes into play when using the standard FAISS high-level APIs. Moreover, as FAISS is a library designed and optimized for batch retrieval, our efficiency experiments are conducted by retrieving results for a batch of queries instead of a single one. The rationale of doing this relies in the fact that, on a back-end level, we can easily assume that queries coming from different clients can be batched together before being submitted to FAISS. The reported response times are obtained as an average of three different runs.

Available Software. The source code used in our experiments is made publicly available to allow the reproducibility of the results[6].

4 EXPERIMENTAL RESULTS

We now discuss the results of the experiments conducted to answer the research questions posed in Section 3.

4.1 RQ1: Can we provide effective cached answers?

The results of the experiments conducted on the three CAsT datasets with the *no-caching* baseline, static-CACHE, and dynamic-CACHE are reported in Table 1. For each dataset, the static, and dynamic versions of CACHE, we vary the value of the cache cutoff k_c as discussed in Sec. 3.2, and highlight with symbol ▼ the statistical significant differences (two-sample t-test with $p < 0.01$) w.r.t. the *no-caching* baseline. The best results for each dataset and effectiveness metric are shown in bold.

By looking at the figures in the table, we see that static-CACHE returns worse results than *no-caching* for all the datasets, most of the metrics, and cache cutoffs k_c considered. However, in a few cases, the differences are not statistically significant. For example, we observe that static-CACHE on CAsT 2019 with $k_c = 10k$ does not statistically differ from *no-caching* for all metrics but MAP@200. The reuse of the embeddings retrieved for the first queries of CAsT 2019 conversations is thus so high that even the simple heuristic of statically caching the top $10k$ embeddings of the first query allows to answer effectively the following queries without further interactions with the back-end. As expected, we see that by increasing the number k_c of statically cached embeddings from $1K$ to $10K$, we improve the quality for all datasets and metrics. Interestingly, we observe that static-CACHE performs relatively better at small query cutoffs since in column P@1 we have, for 5 times out of 12, results not statistically different from those of *no-caching*. We explain such behavior by observing again Figure 3: when an incoming query q_b is close to a previously cached one, i.e., $\hat{r}_b \geq 0$, it is likely that the relevant documents for q_b present in the cache are those most similar to q_b among all those in $\mathcal{D}$. The larger is query cutoff k, the lower is the probability of the least similar documents among the ones in $\text{NN}(q_b, k)$ residing in the cache.

When considering dynamic-CACHE, based on the heuristic update policy discussed earlier, effectiveness improves remarkably. Independently of the dataset and the value of k_c, we achieve performance figures that are not statistically different from those measured with *no-caching* for all metrics but MAP@200. Indeed, the metrics measured at small query cutoffs result in some cases to be even slightly better than those of the baseline even if the improvements are not statistically significant: since the embeddings relevant for a conversation are tightly clustered, retrieving them from the cache rather than from the whole index in some case reduces noise and provides higher accuracy. MAP@200 is the only metrics for which some configurations of dynamic-CACHE result to perform worse than *no-caching*. This is motivated by the tuning of threshold ϵ performed by focusing on small query cutoffs, i.e., the ones commonly considered important for conversational search tasks [4].

RQ1.A: Effectiveness of the quality assessment heuristic. The performance exhibited by dynamic-CACHE demonstrates that the quality assessment heuristic used to determine cache updates is highly effective. To further

---

[6]https://github.com/hpclab/caching-conversational-search

Table 1. Retrieval performance measured on CAsT datasets with or without document embedding caching. We highlight with symbol ▼ statistical significant differences w.r.t. *no-caching* for $p < 0.01$ according to the two-sample t-test. Best values for each dataset and metric are shown in bold.

		k_c	MAP@200	MRR@200	nDCG@3	P@1	P@3	cov_{10}	Hit Rate
CAsT 2019	no-caching	–	**0.194**	0.647	0.376	0.497	0.495	–	–
	static-CACHE	1K	0.101▼	0.507▼	0.269▼	0.387▼	0.364▼	0.40	100%
		2K	0.112▼	0.567▼	0.304▼	0.428	0.414▼	0.47	100%
		5K	0.129▼	0.588	0.316▼	0.451	0.426▼	0.56	100%
		10K	0.140▼	0.611	0.338	0.486	0.459	0.62	100%
	dynamic-CACHE	1K	0.180▼	0.634	0.365	0.474	0.482	0.91	67.82%
		2K	0.183▼	0.631	0.366	0.480	0.487	0.93	70.69%
		5K	0.186▼	0.652	0.375	0.503	0.499	0.94	74.14%
		10K	0.190	**0.655**	**0.380**	**0.509**	**0.505**	0.96	75.29%
CAsT 2020	no-caching	–	**0.212**	0.622	0.338	0.471	0.473	–	–
	static-CACHE	1K	0.112▼	0.421▼	0.215▼	0.312▼	0.306▼	0.35	100%
		2K	0.120▼	0.454▼	0.236▼	0.351▼	0.324▼	0.41	100%
		5K	0.139▼	0.509▼	0.267▼	0.394	0.370▼	0.48	100%
		10K	0.146▼	0.518▼	0.270▼	0.394▼	0.380▼	0.52	100%
	dynamic-CACHE	1K	0.204▼	0.624	0.339	**0.481**	0.478	0.91	56.02%
		2K	0.203▼	**0.625**	0.336	**0.481**	0.470	0.93	60.73%
		5K	0.208	0.622	**0.341**	0.476	**0.479**	0.94	62.83%
		10K	0.210	**0.625**	0.339	0.476	0.476	0.96	63.87%
CAsT 2021	no-caching	–	**0.109**	0.584	**0.340**	0.449	**0.411**	–	–
	static-CACHE	1K	0.068▼	0.430▼	0.226▼	0.323▼	0.283▼	0.38	100%
		2K	0.072▼	0.461▼	0.240▼	0.348▼	0.300▼	0.42	100%
		5K	0.079▼	0.508▼	0.270▼	0.386	0.338▼	0.51	100%
		10K	0.080▼	0.503▼	0.272▼	0.367▼	0.338▼	0.56	100%
	dynamic-CACHE	1K	0.106	0.577	0.335	0.443	0.409	0.89	61.97%
		2K	0.107	**0.585**	0.338	**0.456**	**0.411**	0.91	63.38%
		5K	0.106	0.584	0.334	0.449	0.407	0.92	66.67%
		10K	0.107	0.584	0.336	0.449	0.409	0.94	67.61%

corroborate this claim, the cov_{10} column of Table 1 reports for static-CACHE and dynamic-CACHE the mean coverage for $k = 10$ measured by averaging Eq. (5) over all the conversational queries in the datasets. We recall that this measure counts the cardinality of the intersection between the top 10 elements retrieved from the cache and the exact top 10 elements retrieved from the whole index, divided by 10. While the cov_{10} values for static-CACHE range between 0.35 to 0.62, justifying the quality degradation captured by the metrics reported in the table, with dynamic-CACHE we measure values between 0.89 and 0.96, showing that, consistently across different datasets and cache configurations, the update heuristics proposed successfully trigger when the content of the cache needs refreshing to answer a new topic introduced in the conversation.

To gain further insights about RQ1.A, we conducted other experiments aimed at understanding if the hyperparameter ϵ driving the dynamic-CACHE updates can be fine-tuned for a specific query cutoff. Our investigation is
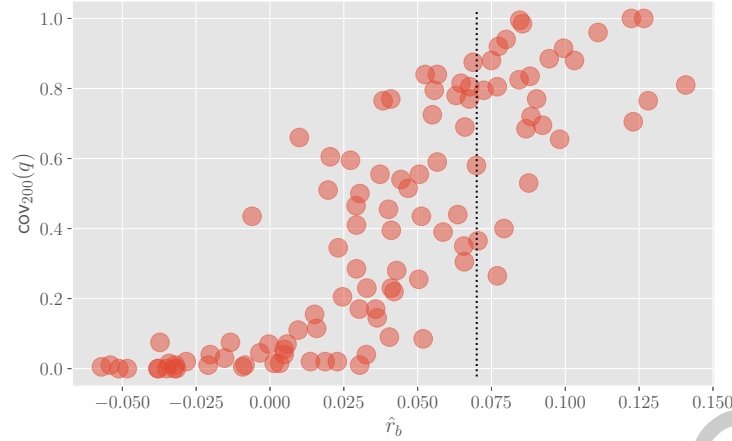
- FILL

Fig. 5. Correlation between $\hat{r}_b$ vs. $\text{cov}_{200}(q)$ for the CAsT 2019 train queries, using static-CACHE and $k_c = 1K$. The vertical black dashed line corresponds to $\hat{r}_b = 0.07$, the tuned cache update threshold value ϵ used in the experiments.

motivated by the MAP@200 results reported in Table 1 that are slightly lower than the baseline for 5 out of 12 dynamic-CACHE configurations. We ask ourselves if it is possible to tune the value of ϵ to achieve MAP@200 results statistically equivalent to those of *no-caching* without losing all the efficiency advantages of our client-side cache.

Similarly to Figure 4, the plot in Figure 5 shows the correlation between the value of $\hat{r}_b$ vs. $\text{cov}_{200}(q)$ for the CAsT 2019 train queries with static-CACHE, $k = 200$ and $k_c = 1K$. Even at query cutoff 200, we observe a strong correlation between $\hat{r}_b$ and the coverage metrics of Eq. 5: most of the train queries with coverage $\text{cov}_{200} \leq 0.3$ have a value of $\hat{r}_b$ smaller than 0.07, with a single query for which this rule of thumb does not strictly hold. Hence, we set $\epsilon = 0.07$ and we run again our experiments with dynamic-CACHE by varying the cache cutoff k_c in $\{1k, 2k, 5k, 10k\}$. The results of these experiments, conducted with the CAsT 2019 dataset, are reported in Table 2. As we can see from the figures reported in the table, increasing from 0.04 to 0.07 the value of ϵ improves the quality of the results returned by the cache at large cutoffs. Now dynamic-CACHE returns results that are always, even for MAP@200, statistically equivalent to the ones retrieved from the whole index by the *no-caching* baseline (according to a two-sample t-test for $p < 0.01$). The improved quality at cutoff 200 is of course paid with a decrease in efficiency. While for $\epsilon = 0.04$ (see Table 1) we measured on CAsT 2019 hit rates ranging from 67.82 to 75.29, by setting $\epsilon = 0.07$ we strengthen the constraint on cache content quality and correspondingly increase the number of cache updates performed. Consequently, the hit rate now ranges from 46.55 to 58.05, witnessing a likewise strong efficiency boost with respect to the *no-caching* baseline.

RQ1.B: Impact of CACHE on client-server interactions. The last column of Table 1 reports the cache hit rate, i.e., the percentage of conversational queries over the total answered with the cached embeddings without interacting with the conversational search back-end. Of course, static-CACHE results in a trivial 100% hit rate since all the queries in a conversation are answered with the embeddings initially retrieved for answering the first query. The lowest possible workload on the back-end is however paid with a significant performance drop with respect to the *no-caching* baseline. With dynamic-CACHE, instead, we achieve high hit rates with the optimal answer quality discussed earlier. As expected, the greater the value of k_c, the larger the number of cached embeddings and the higher the hit rate. With $k_c = 1K$, hit rates range between 56.02% to 67.82%, meaning that even with the lowest cache cutoff experimented more than half of the conversation queries in the 3 datasets are answered

Table 2. Retrieval performance on CAsT 2019 of the *no-caching* baseline and dynamic-CACHE with $\epsilon = 0.07$.

	k_c	MAP@200	MRR@200	nDCG@3	P@1	P@3	cov_{200}	Hit Rate
no-caching	–	0.194	0.647	0.376	0.497	0.495	–	–
dynamic-CACHE	1K	0.193	0.645	0.374	0.497	0.491	0.83	46.55%
	2K	0.193	0.644	0.375	0.497	0.493	0.91	51.15%
	5K	0.194	0.645	0.375	0.497	0.493	0.93	54.02%
	10K	0.194	0.648	0.375	0.497	0.493	0.94	58.05%

directly by the cache, without forwarding the query to the back-end. For $k_c = 10K$, the hit rate value is in the interval $[63.87\% - 75.29\%]$, with more than 3/4 of the queries in the CAsT 2019 dataset answered directly by the cache. If we consider the hit rate as a measure correlated to the amount of temporal locality present in the CAsT conversations, we highlight the highest locality present in the 2019 dataset: on this dataset dynamic-CACHE with $k_c = 1K$ achieves a hit rate higher that the ones measured for $k_c = 10K$ configurations on CAsT 2020 and 2021.

RQ1.C: Worst-case CACHE memory requirements. The memory occupancy of static-CACHE is limited, fixed and known in advance. The worst-case amount of memory required by dynamic-CACHE depends instead on the value of k_c and on the number of cache updates performed during a conversation. The parameter k_c establishes the number of embeddings added to the cache after every cache miss. Limiting the value of k_c can be necessary to respect memory constraints on the client hosting the cache. Anyway, the larger k_c is, the greater the performance of dynamic-CACHE thanks to the increased likelihood that upcoming queries in the conversation will be answered directly, without querying the back-end index. In our experiments, we varied k_c in $\{1K, 2K, 5K, 10K\}$ always obtaining optimal retrieval performances thanks to the effectiveness and robustness of the cache-update heuristic.

Regarding the number of cache updates performed, we consider as exemplary cases the most difficult conversations for our caching strategy in the three CAsT datasets, namely topic 77, topic 104, and topic 117 for CAsT 2019, 2020, and 2021, respectively. These conversations require the highest number of cache updates: 6, 7, 6 for $k_c = 1K$ and 5, 6, 5 for $k_c = 10K$, respectively. Consider topic 104 of CAsT 2020, the toughest conversation for the memory requirements of dynamic-CACHE. At its maximum occupancy, after the last cache update, dynamic-CACHE system stores at most $8 \cdot 1K + 8 \approx 8K$ embeddings for $k_c = 1K$ and $7 \cdot 1K + 7 \approx 70K$ embeddings for $k_c = 10K$. In fact, at a given time, dynamic-CACHE stores the k_c embedding retrieved for the first query in the conversation plus k_c new embeddings for every cache update performed. Indeed, the total number is lower due to the presence of embeddings retrieved multiple times from the index on the back-end. The actual number of cache embeddings for the case considered is $7.5K$ and $64K$ for $k_c = 1K$ and $k_c = 10K$, respectively. Since each embedding is represented with 769 floating point values, the maximum memory occupation for our largest cache is $64K \times 769 \times 4$ bytes ≈ 188 MB. Note that if we consider the case dynamic-CACHE, $k_c = 1K$, achieving the same optimal performance of dynamic-CACHE, $k_c = 10K$ on CAsT 2020 topic 104, the maximum occupancy of the cache decreases dramatically to about 28 MB.

4.2 RQ2: How much does CACHE expedite the conversational search process?

We now answer RQ2 by assessing the efficiency of the conversational search process in presence of cache misses (RQ2.A) or cache hits (RQ2.B).

RQ2.A: What is the impact of the cache cutoff k_c on the efficiency of the system in case of cache misses? We first conduct experiments to understand the impact of k_c on the latency of nearest-neighbor queries performed on the remote back-end. To this end, we do not consider the costs of client-server communications, but only the

retrieval time measured for answering a query on the remote index. Our aim is understanding if the value of k_c impacts significantly or not the retrieval cost. In fact, when we answer the first query in the conversation or dynamic-CACHE performs an update of the cache in case of a miss (lines 1-3 of Algorithm 1), we retrieve from the remote index a large set of k_c embeddings to increase the likelihood of storing in the cache documents relevant for successive queries. However, the query cutoff k commonly used for answering conversational queries is very small, e.g., 1, 3, 5, and $k \ll k_c$. Our caching approach can improve efficiency only if the cost of retrieving from the remote index k_c embeddings is comparable to that of retrieving a much smaller set of k elements. Otherwise, even if we reduce remarkably the number of accesses to the back-end, every retrieval of a large number of results for filling or updating the cache would jeopardize its efficiency benefits.

We conduct the experiment on the CAsT 2020 dataset by reporting the average latency (in msec.) of performing $NN(q, k_c)$ queries on the remote index. Due to the peculiarities of the FAISS library implementation previously discussed, the response time is measured by retrieving the top-k_c results for a batch of 216 queries, i.e., the CAsT 2020 test utterances, and by averaging the total response time (Table 3). Experimental results show that the back-end query response time is approximately 1 second and is almost not affected by the value of k_c. This is expected as exhaustive nearest-neighbor search requires the computation of the distances from the query of all indexed documents, plus the negligible cost of maintaining the top-k_c closest documents in a min-heap. The result thus confirms that large k_c values do not jeopardize the efficiency of the whole system when cache misses occur.

Table 3. Average response time (msec.) for querying the FAISS back-end (no-caching) or the static-CACHE and dynamic-CACHE in case of cache hit.

	k_c			
	1K	2K	5K	10K
no-caching	1,060	1,058	1,061	1,073
static-CACHE	0.14	0.30	0.78	1.59
dynamic-CACHE	0.36	0.70	1.73	3.48

RQ2.B: How much faster is answering a query from the local cache rather than from the remote index? The second experiment conducted aims at measuring the average retrieval time for querying the client-side cache (line 4 of Algorithm 1) in case of hit. We run the experiment for the two caches proposed, i.e., static-CACHE and dynamic-CACHE. While the first one stores a fixed number of documents, the latter employs cache updates that add document embeddings to the cache during the conversation. We report, in the last two rows of Table 3, the average response time of top-3 nearest-neighbor queries resulting in cache hits for different configurations of static-CACHE and dynamic-CACHE. As before, latencies are measured on batches of 216 queries, i.e., the CAsT 2020 test utterances, by averaging the total response time. The results of the experiment show that, in case of a hit, querying the cache requires on average less than 4 milliseconds, more than 250 times less than querying the back-end. We observe that, as expected, hit time increases linearly with the size of the static-CACHE. We also note that dynamic-CACHE shows slightly higher latency than static-CACHE. This is due to the updates of the cache performed during the conversation that add embeddings to the cache. This result shows that the use of a cache in conversational search allows to achieve a speedup of up to four orders of magnitude, i.e., from seconds to few tenths of milliseconds, between querying a remote index and a local cache.

We can now finally answer RQ2, how much does CACHE expedite the conversational search process, by computing the average overall speedup achieved by our caching techniques on an entire conversation. Assuming

that the average conversation is composed of 10 utterances, the *no-caching* baseline that always queries the back-end leads to a total response time of about $10 \times 1.06 = 10.6$ seconds. Instead, with static-CACHE we perform only one retrieval from the remote index for the first utterance while the remaining queries are resolved by the cache. Assuming the use of static-CACHE with 10K embeddings, i.e., the one with higher latency, the total response time for the whole conversation is $1.06 + (9 \cdot 0.00159) = 1.074$ seconds, with an overall speedup of about $9.87\times$ over *no-caching*. Finally, the use of dynamic-CACHE implies possible cache updates that may increase the number of queries answered using the remote index. In detail, dynamic-CACHE with 10K embeddings obtains a hit rate of about 64% on CAsT 2020 (see Table 1). This means that, on average, we forward $1 + (9 \cdot 0.36) = 4.24$ queries to the back-end that cost in total $4.24 \cdot 1.06 = 4.49$ seconds. The remaining cost comes from cache hits. Hits are on average 5.76 and require $5.76 \cdot 0.00348 = 0.002$ seconds accounting for a total response time for the whole conversation of 4.242 seconds. This leads to a speedup of $2.5\times$ with respect to the *no-caching* solution.

The above figures confirm the feasibility and the computational performance advantages of our client-server solution for caching historical embeddings for conversational search.

5 RELATED WORK

Our contribution relates to two main research areas. The first, attracting recently significant interest, is *Conversational Search*. Specifically, our work focuses on the ability of neural retrieval models to capture the semantic relationship between conversation utterances and documents, and, more centrally, with efficiency aspects of neural search. The second related area is *Similarity Caching* that was initially investigated in the field of content-based image retrieval and contextual advertisement.

Neural approaches for conversational search. Conversational search focuses on retrieving relevant documents from a collection to fulfill user information needs expressed in a dialogue, i.e., sequences of natural-language utterances expressed in oral or written form [9, 36]. Given the nature of speech, these queries often lack context and are grammatically poorly formed, complicating their processing. To address these issues, it is natural to exploit past queries and their system response, if available, in a given conversation to build up a *context history*, and use this history to enrich the semantic contents of the current query. The context history is typically used to rewrite the query in a self-contained, decontextualized query, suitable for ad-hoc document retrieval [15, 17, 19, 28, 31]. Lin *et al.* propose two conversational query reformulation methods based on the combination of term importance estimation and neural query rewriting [17]. For the latter, authors reformulate conversational queries into natural and human-understandable queries with a pretrained sequence-to-sequence model. They also use reciprocal rank fusion to combine the two approaches yielding state-of-the-art retrieval effectiveness in terms of NDCG@3 compared to the best submission of Text REtrieval Conference (TREC) Conversational Assistant Track (CAsT) 2019. Similarly, Voskarides *et al.* focus on multi-turn passage retrieval by proposing QuReTeC (Query Resolution by Term Classification), a neural query resolution model based on bidirectional transformers and a distant supervision method to automatically generate training data by using query-passage relevance labels [28]. Authors incorporate QuReTeC in a multi-turn, multi-stage passage retrieval architecture and show its effectiveness on the TREC CAsT dataset.

Others approach the problem by leveraging pre-trained generative language model to directly generate the reformulated queries [18, 26, 32]. Some other studies combine approaches based on term selection strategies and query generation methods [14, 17]. Xu *et al.* propose to track the context history on a different level, i.e., by exploiting user-level historical conversations [30]. They build a structured per-user memory knowledge graph to represent users' past interactions and manage current queries. The knowledge graph is dynamically updated and complemented with a reasoning model that predicts optimal dialog policies to be used to build the personalized answers.

- FILL

Pre-trained language models, such as BERT [5], learn semantic representations called *embeddings* from the contexts of words and, therefore, better capture the relevance of a document w.r.t. a query, with substantial improvements over the classical approach in the ranking and re-ranking of documents [16]. Recently, several efforts exploited pre-trained language models to represent queries and documents in the same dense latent vector space, and then used inner product to compute the relevance score of a document w.r.t. a query.

In conversational search, the representation of a query can be computed in two different ways. In one case, a stand-alone contextual query understanding module reformulates the user query q into a rewritten query $\hat{q}$, exploiting the context history H [9], and then a query embedding $\mathcal{L}(\hat{q})$ is computed. In the other case, the learned representation function is trained to receive as input the query q together with its context history H_q, and to generate a query embedding $\mathcal{L}(q, H_q)$ [23]. In both cases, dense retrieval methods are used to compute the query-document similarity, by deploying efficient nearest neighbor techniques over specialised indexes, such as those provided by the FAISS toolkit [11].

Similarity caching. Similarity caching is a variant of classical exact caching in which the cache can return items that are similar, but not necessarily identical, to those queried. Similarity caching was first introduced by Falchi *et al.*, where the authors proposed two caching algorithms possibly returning approximate result sets for k-NN similarity queries [6]. The two caching algorithms differ in the strategies adopted for building the approximate result set and deciding its quality based on the properties of metric spaces discussed in Section 2. The authors focused on large-scale content-based image retrieval and conducted tests on a collection of one million images observing a significant reduction in average response time. Specifically, with a cache storing at most 5% of the total dataset, they achieved hit rates exceeding 20%. In successive works, the same authors analyzed the impact of similarity caching on the retrieval from larger collections with real user queries [7, 8]. Chierichetti *et al.* propose a similar caching solution that is used to efficiently identify advertisement candidates on the basis of those retrieved for similar past queries [3]. Finally, Neglia *et al.* propose an interesting theoretical study of similarity caching in the offline, adversarial, and stochastic settings [21], aimed at understanding how to compute the expected cost of a given similarity caching policy.

We capitalize on these seminal works by exploiting the properties of similarity caching in metric spaces for a completely different scenario, i.e., dense retrievers for conversational search. Differently from image and advertisement retrieval, our use case is characterized by the similarity among successive queries in a conversation, enabling a novel solution based on integrating a small similarity cache in the conversational client. Our client-side similarity cache answers most of the queries in a conversation without querying the main index hosted remotely. A similar work to our own is the one by Sermpezis *et al.*, where authors propose a similarity-based system for recommending alternative cached content to a user when their exact request cannot be satisfied by the local cache [25]. The contribution is related because it proposes a client-side cache where similar content is looked for, although their focus is on how statically fill the local caches on the basis of user profiles and content popularity.

6 CONCLUSION

We introduced a client-side, document-embedding cache for expediting conversational search systems. Although caching is extensively used in search, we take a closer look at how it can be effectively and efficiently exploited in a novel and challenging setting: a client-server conversational architecture exploiting state-of-the-art dense retrieval models and a novel metric cache hosted on the client-side.

Given the high temporal locality of the embeddings retrieved for answering utterances in a conversation, a cache can provide a great advantage to expedite conversational systems. We initially prove that both queries and documents in a conversation lie close together in the embedding space and that given this specific interaction and query properties, we can exploit the metric properties of distance computations in a dense retrieval context.

We propose two types of caching and compare the results in terms of both effectiveness and efficiency with respect to a no-caching baseline using the same back-end search solution. The first is a static-CACHE which populates the cache with documents retrieved based on the first query of a conversation only. The second, dynamic-CACHE, proposes also an update mechanism that comes in place when we determine, via a precise and efficient heuristic strategy, that the current contents of the cache might not provide relevant results.

The results of extensive and reproducible experiments conducted on CAsT datasets show that dynamic-CACHE achieves hit rates up to 75% with answers quality statistically equivalent to that of the *no-caching* baseline. In terms of efficiency, the response time varies with the size of the cache, nevertheless queries resulting in cache hit are three orders of magnitude faster than those processed on the back-end (accessed only for cache misses by dynamic-CACHE and for all queries by the *no-caching* baseline).

We conclude that our CACHE solution for conversational search is a viable and effective solution, also opening the door for significant further investigation. Its client-side organization permits, for example, to effectively integrate models of user-level contextual knowledge. Equally interesting is the investigation of user-level, personalized query rewriting strategies and neural representations.

ACKNOWLEDGMENTS

REFERENCES

[1] Avishek Anand, Lawrence Cavedon, Hideo Joho, Mark Sanderson, and Benno Stein. 2020. Conversational search. In *Dagstuhl Reports*, Vol. 9.

[2] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the Xbox Recommender System Using a Euclidean Transformation for Inner-Product Spaces. In *Proceedings of the 8th ACM Conference on Recommender Systems* (Foster City, Silicon Valley, California, USA) *(RecSys '14)*. Association for Computing Machinery, New York, NY, USA, 257–264. https://doi.org/10.1145/2645710.2645741

[3] Flavio Chierichetti, Ravi Kumar, and Sergei Vassilvitskii. 2009. Similarity Caching. In *Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Providence, Rhode Island, USA) *(PODS '09)*. Association for Computing Machinery, New York, NY, USA, 127–136. https://doi.org/10.1145/1559795.1559815

[4] Jeffrey Dalton, Chenyan Xiong, Vaibhav Kumar, and Jamie Callan. 2020. CAsT-19: A Dataset for Conversational Information Seeking. In *Proc. SIGIR*. 1985–1988.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL*.

[6] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. 2008. A Metric Cache for Similarity Search. In *Proc. LSDS-IR*. 43–50.

[7] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. 2009. Caching Content-Based Queries for Robust and Efficient Image Retrieval. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (Saint Petersburg, Russia) *(EDBT '09)*. Association for Computing Machinery, New York, NY, USA, 780–790. https://doi.org/10.1145/1516360.1516450

[8] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. 2012. Similarity Caching in Large-Scale Image Retrieval. *Inf. Process. Manage.* 48, 5 (2012), 803–818.

[9] Jianfeng Gao, Chenyan Xiong, Paul Bennett, and Nick Craswell. 2022. Neural Approaches to Conversational Information Retrieval. https://doi.org/10.48550/ARXIV.2201.05176

[10] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-Based Retrieval in Facebook Search. In *Proc. SIGKDD*. 2553–2561.

[11] J. Johnson, M. Douze, and H. Jegou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 03 (2021), 535–547.

- FILL

[12] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proc. EMNLP*. 6769–6781.

[13] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proc. SIGIR*. 39–48.

[14] Vaibhav Kumar and Jamie Callan. 2020. Making Information Seeking Easier: An Improved Pipeline for Conversational Search. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 3971–3980. https://doi.org/10.18653/v1/2020.findings-emnlp.354

[15] Yongqi Li, Wenjie Li, and Liqiang Nie. 2022. Dynamic Graph Reasoning for Conversational Open-Domain Question Answering. *ACM Trans. Inf. Syst.* 40, 4, Article 82 (jan 2022), 24 pages. https://doi.org/10.1145/3498557

[16] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2020. Pretrained Transformers for Text Ranking: BERT and Beyond. arXiv:2010.06467

[17] Sheng-Chieh Lin, Jheng-Hong Yang, Rodrigo Nogueira, Ming-Feng Tsai, Chuan-Ju Wang, and Jimmy Lin. 2021. Multi-Stage Conversational Passage Retrieval: An Approach to Fusing Term Importance Estimation and Neural Query Rewriting. *ACM Trans. Inf. Syst.* 39, 4, Article 48 (aug 2021), 29 pages. https://doi.org/10.1145/3446426

[18] Hang Liu, Meng Chen, Youzheng Wu, Xiaodong He, and Bowen Zhou. 2021. Conversational Query Rewriting with Self-Supervised Learning. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 7628–7632. https://doi.org/10.1109/ICASSP39728.2021.9413557

[19] Ida Mele, Cristina Ioana Muntean, Franco Maria Nardini, R. Perego, Nicola Tonellotto, and Ophir Frieder. 2021. Adaptive utterance rewriting for conversational search. *Inf. Process. Manag.* 58 (2021), 102682.

[20] Ida Mele, Nicola Tonellotto, Ophir Frieder, and Raffaele Perego. 2020. Topical result caching in web search engines. *Inf. Proc. & Man.* 57, 3 (2020).

[21] Giovanni Neglia, Michele Garetto, and Emilio Leonardi. 2022. Similarity Caching: Theory and Algorithms. *IEEE/ACM Transactions on Networking* 30, 2 (2022), 475–486. https://doi.org/10.1109/TNET.2021.3126368

[22] Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *Proc. ICML*. 1926–1934.

[23] Chen Qu, Liu Yang, Cen Chen, Minghui Qiu, W. Bruce Croft, and Mohit Iyyer. 2020. *Open-Retrieval Conversational Question Answering*. Association for Computing Machinery, New York, NY, USA, 539–548. https://doi.org/10.1145/3397271.3401110

[24] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proc. EMNLP*. 3980–3990.

[25] Pavlos Sermpezis, Theodoros Giannakas, Thrasyvoulos Spyropoulos, and Luigi Vigneri. 2018. Soft Cache Hits: Improving Performance Through Recommendation and Delivery of Related Content. *IEEE Journal on Selected Areas in Communications* 36, 6 (2018), 1300–1313. https://doi.org/10.1109/JSAC.2018.2844983

[26] Svitlana Vakulenko, Shayne Longpre, Zhucheng Tu, and Raviteja Anantha. 2021. Question Rewriting for Conversational Question Answering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining* (Virtual Event, Israel) *(WSDM '21)*. Association for Computing Machinery, New York, NY, USA, 355–363. https://doi.org/10.1145/3437963.3441748

[27] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605.

[28] Nikos Voskarides, Dan Li, Pengjie Ren, Evangelos Kanoulas, and Maarten de Rijke. 2020. *Query Resolution for Conversational Search with Limited Supervision*. Association for Computing Machinery, New York, NY, USA, 921–930. https://doi.org/10.1145/3397271.3401130

[29] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *Proc. ICLR*.

[30] Hu Xu, Seungwhan Moon, Honglei Liu, Bing Liu, Pararth Shah, Bing Liu, and Philip Yu. 2020. User Memory Reasoning for Conversational Recommendation. In *Proceedings of the 28th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, Barcelona, Spain (Online), 5288–5308. https://doi.org/10.18653/v1/2020.coling-main.463

[31] Jheng-Hong Yang, Sheng-Chieh Lin, Chuan-Ju Wang, Jimmy J. Lin, and Ming-Feng Tsai. 2019. Query and Answer Expansion from Conversation History. In *TREC*.

[32] Shi Yu, Jiahua Liu, Jingqin Yang, Chenyan Xiong, Paul Bennett, Jianfeng Gao, and Zhiyuan Liu. 2020. *Few-Shot Generative Conversational Query Rewriting*. Association for Computing Machinery, New York, NY, USA, 1933–1936. https://doi.org/10.1145/3397271.3401323

[33] Shi Yu, Zhenghao Liu, Chenyan Xiong, Tao Feng, and Zhiyuan Liu. 2021. Few-Shot Conversational Dense Retrieval. In *Proc. SIGIR*. 829–838.

[34] Hamed Zamani, Johanne R Trippas, Jeff Dalton, and Filip Radlinski. 2022. Conversational Information Seeking. *arXiv:2201.08808* (2022).

[35] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing Dense Retrieval Model Training with Hard Negatives. In *Proc. SIGIR*. 1503–1512.

[36] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W. Bruce Croft. 2018. Towards Conversational Search and Recommendation: System Ask, User Respond. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18)*. Association for Computing Machinery, New York, NY, USA, 177–186. https://doi.org/10.1145/3269206.3271776