Deep Reinforcement Agent for Efficient Instant Search

Ravneet Singh Arora[1], Sreejith Menon[2], Ayush Jain[1] and Nehil Jain[1]
[1]*Bloomberg, U.S.A.*
[2]*Google, U.S.A.*

Keywords: Instant Search, Deep Learning, Reinforcement Learning, Information Retrieval, Search.

Abstract: Instant Search is a paradigm where a search system retrieves answers on the fly while typing. The naïve implementation of an Instant Search system would hit the search back-end for results each time a user types a key, imposing a very high load on the underlying search system. In this paper, we propose to address the load issue by identifying tokens that are semantically more salient toward retrieving relevant documents and utilizing this knowledge to trigger an instant search selectively. We train a reinforcement agent that interacts directly with the search engine and learns to predict the word's importance in relation to the search engine. Our proposed method treats the search system as a black box and is more universally applicable to diverse architectures. To further support our work, a novel evaluation framework is presented to study the trade-off between the number of triggered searches and the system's performance. We utilize the framework to evaluate and compare the proposed reinforcement method with other baselines. Experimental results demonstrate the efficacy of the proposed method in achieving a superior trade-off.

1 INTRODUCTION

Interactivity in search engines has substantially grown in popularity in recent years. To further enrich the user experience, most modern search engines such as Google and Bing provide instant search capabilities (Venkataraman et al., 2016b). Instant search retrieves results on the fly at every keystroke compared to conventional search engines that trigger search at the end of the query. Analyses of query-logs performed by (Cetindil et al., 2012) have shown that the instant search improves user experience by reducing the overall time and effort to retrieve the relevant results and helps users find information when they are not sure of the exact query. This feature is very relevant to mobile applications. Recently, these systems have also become extremely popular in Social Networking websites such as Linkedin (Venkataraman et al., 2016a). Instant Answers is another variation of this paradigm, which is very common in search engines these days. Instant answers allows users to view answers instantly while typing questions such as *"how is weather today?"* This feature is also handy in open-domain question answering where user needs are ambiguous.

The implementation of instant search systems faces a significant challenge in the form of immense load on the back-end search engine. The instant search leads to an increase of tens or up to hundreds of more queries for a single search session. This becomes more severe in the case of longer natural language queries. Managing such load becomes problematic for several reasons: the software or the hardware might not be able to cope with high query throughput during spikes of requests, or it might cause high energy consumption by the servers, or just consume computational resources needed by other processes like indexing.

Several approaches have been proposed to improve the performance and scalability of instant search. Many of these studies are based on designing more efficient index data structures for faster retrieval of results (Bast and Weber, 2006; Fafalios and Tzitzikas, 2011; Li et al., 2012; Ji et al., 2009; Li et al., 2011; Wang et al., 2010). These data structures are examined together with the techniques such as caching (Fafalios et al., 2012) for their ability to improve the search engine query throughput. Caching has been further extensively applied for large-scale traditional search systems in various studies such as (Markatos, 2001; Saraiva et al., 2001; Dean, 2009; Gan and Suel, 2009; Fagni et al., 2006; Long and Suel, 2006). New index data-structures and file sys-

Arora, R., Menon, S., Jain, A. and Jain, N.
Deep Reinforcement Agent for Efficient Instant Search.
DOI: 10.5220/0012178300003598
In *Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2023) - Volume 1: KDIR*, pages 281-288
ISBN: 978-989-758-671-2; ISSN: 2184-3228

Traditional Search	*Flight service from New York to Barcelona Spain*
Instant Search	*Flight service from New York to Barcelona Spain*
Proposed System	*Flight service from New York to Barcelona Spain*

Figure 1: Behaviors of Traditional, Instant and proposed Instant search system for the query *Flight service from New York to Barcelona Spain*. Searches are triggered at tokens marked green. The whole prefix is forwarded to the search engine as a query.

tem formats for boosting the overall speed of search engines have also been explored (Brin and Page, 1998; Dean, 2009).

In this paper, we propose a new method to solve the instant search paradigm's scalability challenges. Our approach stems from the idea that a subset of tokens heavily influences the retrieval of the most relevant results. This subset generally includes keywords that are either topical or tokens that can alter the semantic meaning of the query. We have applied this idea towards training a reinforcement agent that predicts if a typed token is salient and selectively triggers search only for such tokens. This is illustrated in Figure 1. Searches are triggered at tokens marked green. A traditional search system would wait till the last token before issuing the search; an instant search system, on the other hand, queries at every new token. Our proposed approach, in addition to common stopwords, decides to skip the search at word `New` as it is very common and needs more context (`York` in this case) to retrieve the correct answer. Also, since there is only one Barcelona city present in Spain, the word `Spain` does not influence the results returned and hence is skipped. The new approach treats the underlying retrieval engine as a black box and is decoupled from the internal implementation. During the training, the agent updates weights based on the feedback received during its interaction with the search system. This methodology has the following advantages: a) More universal application to a diverse set of modern architectures; b) No need to scale up individual components of complicated search and QA pipelines such as (Yang et al., 2019); c) Easy integration with the existing techniques such as caching.

Reinforcement learning provides the framework to integrate and experiment with different reward functions. Furthermore, there can be a lot of different states based on the decision taken by the algorithm and it is not easy to calculate exact true labels for a pure supervised setting. Recently, reinforcement learning has been successfully applied to an identical problem in the field of Simultaneous Machine Translation (SMT) (Grissom II et al., 2014; Satija and Pineau, 2016; Gu et al., 2016). SMT is defined as the task of producing a real-time translation of a sentence while typing. The goal here is to achieve a good trade-off between the quality and delay of the translation.

We further evaluate the loss in the quality of instant search due to introducing the proposed reinforcement agent. Instant search quality is measured based on the studies that have compared the instant search system with a traditional one (Cetindil et al., 2012; Chandar et al., 2019). Instant search query logs have been analyzed by (Cetindil et al., 2012) to understand the properties of instant search that lead to a better user experience. Recently (Chandar et al., 2019) combined user-query interaction logs with user interviews and proposed new metrics that can evaluate user satisfaction for an instant search system. Both the studies have proposed results-quality and user-effort as the two primary metrics to measure user experience improvement. Quality measures how relevant the search system results are to the user query, whereas Effort captures how quickly the relevant results are retrievable using a search engine. We use these metrics to estimate how well the proposed methods can reduce the overall system load while preserving the performance. Experiments are performed on three different combinations of datasets with two retrieval systems. Our experiments show that the proposed model achieves a superior trade-off by achieving near-optimal performance while reducing the number of triggered searches by 50%.

2 BASELINES

This section introduces the baselines that are evaluated and compared with the proposed model.
Search at Every Token: SET issues search for every new token. This method represents the true instant search paradigm.
Search at Last Token: SLT waits for the entire query and triggers a single search request at the end. This baseline mimics the behavior of a regular retrieval en-

gine.

Skip Stop-Words: SS simply issues a search at every token except the stop-words.

Similarity Matching Pre-Trained Model: SM pre-trained model issues a query only when the query's semantic meaning has changed by more than a certain threshold. We utilize the pre-trained Universal Sentence Encoder model (Cer et al., 2018) to generate an embedding for the query at every new token and compare the similarity with the embedding of the previously searched sub-query. We use *CosineDistance* between sentence embedding vector pairs to measure the similarity. A sentence pair S_1, S_2 is considered to be semantically different if $CosineDistance(S_1, S_2) \geq threshold$ (Gomaa et al., 2013). We treat the *threshold* as a hyper-parameter, and the actual value is later stated in Section 4. Algorithm 1 describes this approach in more detail.

```
Q ← Query ;
N ← Number of tokens in query Q ;
D ← Set of Retrieved Documents ;
q_searched ← Sequence of tokens previously
  searched ;
V_searched ← Embedding Vector of q_searched ;
q_current ← Current sequence of tokens ;
V_current ← Embedding Vector of q_current ;
for i ← 1 to N do
    q_current ← Q[1, i];
    V_current ← GetEmbedding(q_current);
    if CosineDistance (V_searched,
     V_current) ≥ threshold then
        q_searched ← q_current ;
        V_searched ← V_current ;
        D ← RetrieveDocuments(q_current);
    end
end
```

Algorithm 1: Inference using Similarity Matching pre-trained model Method.

3 REINFORCEMENT AGENT

Deep Q Networks: In Q-learning (Watkins and Dayan, 1992), the environment is formulated as a sequence of state transitions (s_t, a_t, r_t, s_{t+1}) of a Markov Decision Process (MDP). At a given time-step t for state s_t, the agent takes an action a_t and in response receives the reward r_t. As a result, the environment transitions into state s_{t+1}. The agent chooses action a_t for the state s_t by referring a state-action value function $Q(s_t, a_t)$, which measures the action's expected long-term reward. The algorithm updates the Q-function by interacting with the environment and obtaining rewards. In large environments, it is impractical to maintain a Q function for a substantially large number of states. DQN (Mnih et al., 2013) solves this problem by approximating $Q(s, a)$ using a deep neural network, which takes state s as input and calculates value for every state/action pair.

Environment: The environment yields new words for the agent and also interacts with the underlying search engine. For a given query, the agent receives a new word x_t from the environment at every time-step t and, in response, takes action a_t. Based on the action, the environment requests the underlying retrieval engine, and the agent is provided feedback in the form of reward r_t. An episode terminates at the last token x_T of the query.

State: The state represents the portion of the query that is already observed by the Agent. For a given query q, let us assume that the agent has received tokens $x_1, \cdots, x_t$ denoted by partial query q'. The environment maintains two sequences of tokens for every q':

- q'_1: the list of tokens $x_1, \cdots, x_{t'}$ used in the last search query submitted to the system.
- q'_2: the list of tokens $x_{t'+1} \cdots, x_t$ the system has seen since it last submitted a search query.

This state formulation allows the agent to learn the overall importance of q' conditioned on already searched sequence q'_1. At every time-step t, the agent receives a new token x_t which is then appended to the unseached sub-sequence q'_2: $q'_2 = q'_2 \cup x_t$. After a search is triggered, q'_2 is appended to the searched sub-sequence q'_1 and q'_2 is cleared back to empty.

Actions: For every new token x_t, the agent chooses one of the following actions:

- **WAIT:** Instant search is not triggered, and the agent waits for the next token.
- **SEARCH:** Typed query q' is issued to the underlying search system, and new results are retrieved. SEARCH action results in following state transition: $q'_1 = q'_1 \cup q'_2$; $q'_2 = \emptyset$

Reward: During training, at every time-step t, the agent receives reward r_t based on (s_t, a_t). The reward function is designed to encourage the agent to improve the search result's quality while keeping the number of searches issued to the underlying retrieval engine low. The agent receives a positive reward if a SEARCH (S) action leads to an improvement in Mean Average Precision (MAP) by more than a fixed threshold R_{th}. Otherwise, a constant penalty of -1 is imposed. The positive reward is directly proportional to the improvement in map: Δ_{MAP}. We treat the threshold R_{th} as a hyper-parameter and the actual value is later stated in Section 4. Since the WAIT (W)

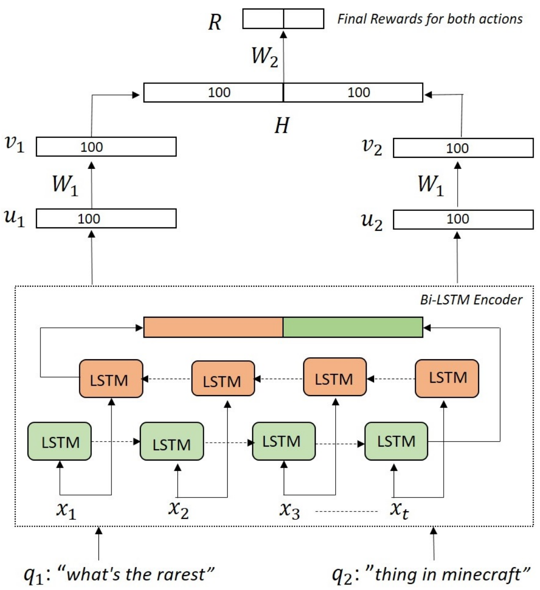

Figure 2: The Bi-LSTM Siamese DQN network for calculating rewards for both WAIT and SEARCH actions. The numbers represent the dimension of outputs generated by each layer.

action does not affect the Quality and Total Searches, the reward is set as 0. The following equation summarizes the reward function:

$$R = \begin{cases} 0, & \text{action} = \mathbf{W} \\ 1 + \Delta\ \text{MAP}, & \text{action} = \mathbf{S} \text{ and } \Delta_{MAP} \geq R_{th} \\ \text{-1}, & \text{action} = \mathbf{S} \text{ and } \Delta_{MAP} < R_{th} \end{cases}$$

Bi-LSTM Deep Q Network: This section describes the base network architecture, as shown in Figure 2 that calculates rewards for a given state. Input to the model is the state, formulated as a pair of sub-queries (q_1, q_2). Input tokens for each sub-query are represented using pre-trained GloVe (Pennington et al., 2014) word-embeddings, that are then passed to a Bi-Directional Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) Siamese Encoder. Since both the sequences have originated from the same query, it is intuitive to apply a Siamese Network that allows the sharing of Bi-LSTM weights. The output vectors for both the sub-queries are concatenated, and the final single feature vector is fed to a fully-connected layer that generates a two-dimensional vector representing the rewards for both the actions. The whole network can be summarized using the below equation:

$$u_1 = f_{Bi\text{-}LSTM}(q_1); u_2 = f_{Bi\text{-}LSTM}(q_2)$$
$$v_1 = relu(W_1 \cdot u_1 + b_1); v_2 = relu(W_1 \cdot u_2 + b_1)$$
$$H = [v_1 \oplus v_2]; R = W_2 \cdot H + b_2$$

Inference: For every typed token during an instant search session, a state is prepared as a pair of sub-queries: prefix of the already searched query and suffix that still needs to be searched. The state is passed as an input to the trained model described in Figure 2. The base model generates rewards for both WAIT and SEARCH actions. The agent picks the action with the best reward, and based on that, the search to the underlying system is either skipped or triggered using the query entered so far. The state for the agent is updated accordingly, and the agent waits for the next token. An episode terminates at the end of the query session.

4 METRICS AND EXPERIMENTAL SETUP

Metrics: We utilize the following metrics to evaluate the performance of the proposed methods.
Average Number of Triggered Searches (TS) - System Load: This metric represents the load on the search system and is measured as the average number of requests made to the search system during an instant search session.
Average Effort: Studies (Cetindil et al., 2012; Chandar et al., 2019) have found the Effort to be a very crucial factor that differentiates an instant search user-experience from a traditional search system. Effort is defined as the minimum number of tokens that a user would have to type to retrieve the best possible ranking of results. Ranking quality is measured using Mean Average Precision (*MAP*) and the best ranking achieves the maximum MAP. Let N_q be the number of tokens in a given query q. n_q is the minimum number of tokens needed to retrieve the best possible ranking for query q. Metric $effort$ is the average effort across all queries in the dataset and is computed as follows:

$$effort = \frac{\sum n_q \leq N_q}{|Q|} \forall q \in Q$$

Quality: We use MAP to capture the quality of the results. MAP is calculated using the open source PyTREC-Eval (Van Gysel and de Rijke, 2018) library.
Evaluation Procedure: To measure the TS vs. Effort trade-off, we simulate an action function in a real instant search session for every query and keep track of both the metrics. The action function returns an action(*WAIT* and *SEARCH*) at every new token based on the decision taken by the method being evaluated. For instance, the SET(subsection 2) method would return *SEARCH* for every token in the query. TS is incremented, and results get updated at every search. For every query, we invoke the action function until the retrieval has achieved the best possible MAP or has reached the last token. The total number of tokens

used to achieve the best MAP is added to the Effort at the end of the query session. For Quality, we keep track of the MAP achieved at every token position for all the queries.
Datasets: We have evaluated the methods on three IR datasets: MS Marco passage ranking(Nguyen et al., 2016), Wiki IR(Frej et al., 2019) 59k version and InsuranceQA (Feng et al., 2015). InsuranceQA is adapted to a pure Document Retrieval task using (Tran and Niedereée, 2018). InsuranceQA is used in order to test how well methods generalize to different domains. To ensure that the underlying search engine can retrieve relevant documents in top 1000 for enough queries, we have reduced the total number of documents to 400k and 500k for MS Marco and Wiki IR, respectively by random sampling. For InsuranceQA, we use the full set of 27,413 documents. The evaluation sets of size 1000 queries are kept unseen for all three datasets.
Retrievers: We conduct experiments using both the BM25(Robertson and Zaragoza, 2009) and semantic-based matching retrieval systems. For semantic retrieval, we use a transformer-based pre-trained sentence encoding model known as Universal Sentence Encoder (USE) (Cer et al., 2018) for representing the queries and documents with embeddings and further use cosine similarity to rank results.
Hyper-Parameters: For SM, we set a threshold of 0.1. For the proposed DQN agent, we trained the model with following settings: future reward $\gamma = 0.05$, $\varepsilon = 1$, $\varepsilon_{decay} = 0.995$, learning rate $\alpha = 0.01$ and $\varepsilon_{min} = 0.7$. Furthermore, weights are learned using Adam optimizer(Kingma and Ba, 2014) with a batch size of 32. Reward threshold R_{th} mentioned in the reward section for determining the action is set to 0 for MS Marco and Wiki IR and 0.0001 for Insurance QA.

5 RESULTS

TS vs. Effort: Table 1 shows the drop in Average Number of Triggered Searches achieved by different methods and compares it with extra Average Effort introduced in the system. The top two rows highlight the absolute values achieved by two basic search systems: SLT, which mimics a traditional search engine, and SET, representing a true instant search system. These systems set the upper and lower bounds respectively on Effort and TS. The bottom three rows list down the percentage change in the metrics introduced by the proposed methods with respect to a true instant search system(SET).

Skip Stop-words method manages to achieve optimum Effort. This can be attributed to the fact that generally, stop-words are not deemed salient in common language usage, allowing SS not to miss a search for any salient words. On the other hand, since SS influences only a limited and fixed set of tokens, the achieved TS is not up to the optimal. Also, the overall performance of SM is quite comparable to SS. Results also show that that the pre-trained model is unable to transfer its knowledge to this new task.

While all the methods are able to retain the Effort within 5% of the SET, the proposed DQN method manages to reduce the overall TS on average by more than 20% across all the datasets compared to other baselines. Compared to a pure instant search system, DQN reduces the overall load by more than 50%. The performance of DQN agent is directly proportional to the training size of the dataset and hence is highest for MS Marco.
Impact on Quality: We have captured the loss of quality in results at every token position by plotting the average MAP over all the queries at every token position for the proposed DQN method and further comparing it with the ideal SET instant search. Figure 3 plots the average MAP(y-axis) at every time-step t(x-axis) for both SET and DQN. The plot shows that the MAP achieved by DQN is very close to that of SET at all the token positions, and hence the loss in quality introduced is minimal.
Subjective Analysis: Lastly, we subjectively analyzed the predictions made on the unseen queries by the learned model. Figure 4 lists queries with tokens at which the search triggered is marked green. We also report the incremental difference in MAP introduced by the triggered search(highlighted as blue) to capture the search action quality.

For MS Marco, besides stop-words, the agent waits for the words *"cost"* , *"install"* and *"purpose"*. MS Marco is a large QA dataset with verbose passages. It is difficult for a basic BM25 algorithm to retrieve a good ranking without additional context early in the query; thus the model decides to wait. For the first InsuranceQA query, the agent decides to execute the search for the token *"a"* as in insurance jargon, "vest a retirement plan" is a common phrase, and a semantic model such as Universal Sentence Encoder does not ignore this as a stop-word. The same is not true for the second sentence where the search is skipped for the same token. For WikiIR's first query, the phrase *"chief justice"* is often present as a whole in documents, and any improvement in ranking contributed by the phrase itself is already captured by the first word *"chief"*. In the second example, the name *"juan carlos"* is unique enough to retrieve relevant documents; therefore, the token *"i"* is skipped.

Table 1: Metrics achieved by different methods. Effort and TS metrics are averaged over all the queries. The top two rows are the absolute values achieved by two base search systems. The bottom three rows list down the % change in the metrics introduced by methods with respect to a true instant search system. *Statistical significance is tested using a two-tailed paired t-test. We mark significant improvements when $p < 0.01$.

Methods	MS Marco - BM25		Wiki IR - BM25		Insurance QA - USE	
	Effort	TS	Effort	TS	Effort	TS
SLT (Regular Search)	10.76	1	5.83	1	8.25	1
SET (Instant Search)	8.24	8.24	4.74	4.74	7.70	7.70
Percentage change in metrics with respect to SET(Pure Instant Search)						
	ΔEffort(%)	ΔTS(%)	ΔEffort(%)	ΔTS(%)	ΔEffort(%)	ΔTS(%)
SS (Baseline)	0	-49.75	0	-22.62	0.59	-39.25
SM (Baseline)	4.00	-45.43	3.24	-26.88	1.50	-40.42
DQN (Proposed)	4.00	**-74.15***	3.94	**-44.88***	1.37	**-55.47***

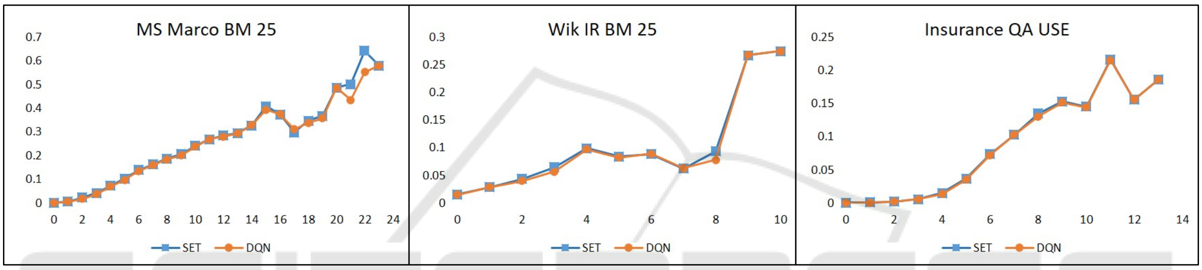

Figure 3: Average MAP achieved by DQN vs. SET at every token position. X-axis is token index and Y-axis is MAP averaged over all the queries.

MS Marco	labor(+0.03) cost to install tile(+0.24) shower(+0.25)
	what is the purpose of the neutral(+0.20) wire(+0.05)
Ins QA	what do vest(+0.04) mean(+0.11) in a(+0.01) retirement(+0.20) plan(+0.25)
	do home insurance(+0.001) cover(+0.006) a toilet(+0.01) leak(+0.15)
Wiki IR	chief(+0.02) justice of new zealand(+0.04)
	juan(+0.02) carlos(+0.04) i of spain(+0.18)

Figure 4: Predictions of DQN Network on unseen queries.

6 CONCLUSION

This paper has introduced a Reinforcement Agent that relieves the load on the back-end search system in an instant search paradigm. Proposed agent achieves the goal by learning word importance based on the search system behavior and utilizes this knowledge towards judiciously issuing searches to the underlying retrieval system. We further evaluated the trade-off between system load and performance. Experiments demonstrate the ability of the proposed agent to achieve near-optimal trade-off.

REFERENCES

Bast, H. and Weber, I. (2006). Type less, find more: fast autocompletion search with a succinct index. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 364–371.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al. (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Cetindil, I., Esmaelnezhad, J., Li, C., and Newman, D. (2012). Analysis of instant search query logs. In *WebDB*, pages 7–12. Citeseer.

Chandar, P., Garcia-Gathright, J., Hosey, C., St. Thomas, B., and Thom, J. (2019). Developing evaluation metrics for instant search using mixed methods methods. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 925–928.

Dean, J. (2009). Challenges in building large-scale information retrieval systems. In *Keynote of the 2nd ACM International Conference on Web Search and Data Mining (WSDM)*, volume 10.

Fafalios, P., Kitsos, I., and Tzitzikas, Y. (2012). Scalable, flexible and generic instant overview search. In *Proceedings of the 21st International Conference on World Wide Web*, pages 333–336.

Fafalios, P. and Tzitzikas, Y. (2011). Exploiting available memory and disk for scalable instant overview search. In *International Conference on Web Information Systems Engineering*, pages 101–115. Springer.

Fagni, T., Perego, R., Silvestri, F., and Orlando, S. (2006). Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Transactions on Information Systems (TOIS)*, 24(1):51–78.

Feng, M., Xiang, B., Glass, M. R., Wang, L., and Zhou, B. (2015). Applying deep learning to answer selection: A study and an open task. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 813–820. IEEE.

Frej, J., Schwab, D., and Chevallet, J.-P. (2019). Wikir: A python toolkit for building a large-scale wikipedia-based english information retrieval dataset. *arXiv preprint arXiv:1912.01901*.

Gan, Q. and Suel, T. (2009). Improved techniques for result caching in web search engines. In *Proceedings of the 18th international conference on World wide web*, pages 431–440.

Gomaa, W. H., Fahmy, A. A., et al. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18.

Grissom II, A., He, H., Boyd-Graber, J., Morgan, J., and Daumé III, H. (2014). Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on empirical methods in natural language processing (EMNLP)*, pages 1342–1352.

Gu, J., Neubig, G., Cho, K., and Li, V. O. (2016). Learning to translate in real-time with neural machine translation. *arXiv preprint arXiv:1610.00388*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ji, S., Li, G., Li, C., and Feng, J. (2009). Efficient interactive fuzzy keyword search. In *Proceedings of the 18th international conference on World wide web*, pages 371–380.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, G., Ji, S., Li, C., and Feng, J. (2011). Efficient fuzzy full-text type-ahead search. *The VLDB Journal*, 20(4):617–640.

Li, G., Wang, J., Li, C., and Feng, J. (2012). Supporting efficient top-k queries in type-ahead search. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 355–364.

Long, X. and Suel, T. (2006). Three-level caching for efficient query processing in large web search engines. *World Wide Web*, 9(4):369–395.

Markatos, E. P. (2001). On caching search engine query results. *Computer Communications*, 24(2):137–143.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L. (2016). Ms marco: A human-generated machine reading comprehension dataset.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Robertson, S. and Zaragoza, H. (2009). *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.

Saraiva, P. C., Silva de Moura, E., Ziviani, N., Meira, W., Fonseca, R., and Ribeiro-Neto, B. (2001). Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 51–58.

Satija, H. and Pineau, J. (2016). Simultaneous machine translation using deep reinforcement learning. In *ICML 2016 Workshop on Abstraction in Reinforcement Learning*.

Tran, N. K. and Niedereée, C. (2018). Multihop attention networks for question answer matching. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 325–334.

Van Gysel, C. and de Rijke, M. (2018). Pytrec_eval: An extremely fast python interface to trec_eval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 873–876.

Venkataraman, G., Lad, A., Guo, L., and Sinha, S. (2016a). Fast, lenient and accurate: Building personalized instant search experience at linkedin. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1502–1511. IEEE.

Venkataraman, G., Lad, A., Ha-Thuc, V., and Arya, D. (2016b). Instant search: A hands-on tutorial. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1211–1214.

Wang, J., Cetindil, I., Ji, S., Li, C., Xie, X., Li, G., and Feng, J. (2010). Interactive and fuzzy search: a dynamic way to explore medline. *Bioinformatics*, 26(18):2321–2327.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Yang, W., Xie, Y., Lin, A., Li, X., Tan, L., Xiong, K., Li, M., and Lin, J. (2019). End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718*.