LEARNING TO COORDINATE MULTIPLE REINFORCEMENT LEARNING AGENTS FOR DIVERSE QUERY REFORMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a method to efficiently learn diverse strategies in reinforcement learning for query reformulation in the tasks of document retrieval and question answering. In the proposed framework an agent consists of multiple specialized sub-agents and a meta-agent that learns to aggregate the answers from sub-agents to produce a final answer. Sub-agents are trained on disjoint partitions of the training data, while the meta-agent is trained on the full training set. Our method makes learning faster, because it is highly parallelizable, and has better generalization performance than strong baselines, such as an ensemble of agents trained on the full data. We show that the improved performance is due to the increased diversity of reformulation strategies.

1 INTRODUCTION

Reinforcement learning has proven effective in several language processing tasks, such as machine translation (Wu et al., 2016; Ranzato et al., 2015; Bahdanau et al., 2016), question-answering (Wang et al., 2017a; Hu et al., 2017), and text summarization (Paulus et al., 2017). In reinforcement learning efficient exploration is key to achieve good performance. The ability to explore in parallel a diverse set of strategies often speeds up training and leads to a better policy (Mnih et al., 2016; Osband et al., 2016).

In this work, we propose a simple method to achieve efficient parallelized exploration of diverse policies, inspired by hierarchical reinforcement learning (Singh, 1992; Lin, 1993; Dietterich, 2000; Dayan & Hinton, 1993). We structure the agent into multiple sub-agents, which are trained on disjoint subsets of the training data. Sub-agents are co-ordinated by a meta-agent, called aggregator, that groups and scores answers from the sub-agents for each given input. Unlike sub-agents, the aggregator is a generalist since it learns a policy for the entire training set.

We argue that it is easier to train multiple sub-agents than a single generalist one since each sub-agent only needs to learn a policy that performs well for a subset of examples. Moreover, specializing agents on different partitions of the data encourages them to learn distinct policies, thus giving the aggregator the possibility to see answers from a population of diverse agents. Learning a single policy that results in an equally diverse strategy is more challenging.

Since each sub-agent is trained on a fraction of the data, and there is no communication between them, training can be done faster than training a single agent on the full data. Additionally, it is easier to parallelize than applying existing distributed algorithms such as asynchronous SGD or A3C (Mnih et al., 2016), as the sub-agents do not need to exchange weights or gradients. After training the sub-agents, only their actions need to be sent to the aggregator.

We build upon the works of Nogueira & Cho (2017) and Buck et al. (2018b). Therefore, we evaluate the proposed method on the same tasks they used: query reformulation for document retrieval and question-answering. We show that it outperforms a strong baseline of an ensemble of agents trained on the full dataset. We also found that performance and reformulation diversity are correlated (Sec. 5.5).

Our main contributions are the following:
A simple method to achieve more diverse strategies and better generalization performance than a model average ensemble.

Training can be easily parallelized in the proposed method.

An interesting finding that contradicts our, perhaps naive, intuition: specializing agents on semantically similar data does not work as well as random partitioning. An explanation is given in Appendix F.

2 Related Work

The proposed approach is inspired by the mixture of experts, which was introduced more than two decades ago (Jacobs et al., 1991; Jordan & Jacobs, 1994) and has been a topic of intense study since then. The idea consists of training a set of agents, each specializing in some task or data. One or more gating mechanisms then select subsets of the agents that will handle a new input. Recently, Shazeer et al. (2017) revisited the idea and showed strong performances in the supervised learning tasks of language modeling and machine translation. Their method requires that output vectors of experts are exchanged between machines. Since these vectors can be large, the network bandwidth becomes a bottleneck. They used a variety of techniques to mitigate this problem. Anil et al. (2018) later proposed a method to further reduce communication overhead by only exchanging the probability distributions of the different agents. Our method, instead, requires only scalars (rewards) and short strings (original query, reformulations, and answers) to be exchanged. Therefore, the communication overhead is small.

Previous works used specialized agents to improve exploration in RL (Dayan & Hinton, 1993; Singh, 1992; Kaelbling et al., 1996). For instance, Stanton & Clune (2016) and Conti et al. (2017) use a population of agents to achieve a high diversity of strategies that leads to better generalization performance and faster convergence. Rusu et al. (2015) use experts to learn subtasks and later merge them into a single agent using distillation (Hinton et al., 2015).

The experiments are often carried out in simulated environments, such as robot control (Brockman et al., 2016) and video-games (Bellemare et al., 2013). In these environments, rewards are frequently available, the states have low diversity (e.g., same image background), and responses usually are fast (60 frames per second). We, instead, evaluate our approach on tasks whose inputs (queries) and states (documents and answers) are diverse because they are in natural language, and the environment responses are slow (0.5-5 seconds per query).

Somewhat similarly motivated is the work of Serban et al. (2017). They train many heterogeneous response models and further train an RL agent to pick one response per utterance.

3 Method

3.1 Task

We describe the proposed method using a generic end-to-end search task. The problem consists in learning to reformulate a query so that the underlying retrieval system can return a better result.

Following Nogueira & Cho (2017) and Buck et al. (2018b) we frame the task as a reinforcement learning problem, in which the query reformulation system is an RL-agent that interacts with an environment that provides answers and rewards. The goal of the agent is to generate reformulations such that the expected returned reward (i.e., correct answers) is maximized. The environment is treated as a black-box, i.e., the agent does not have direct access to any of its internal mechanisms. Figure 1-(b) illustrates this framework.

3.2 System

Figure 1-(c) illustrates the new agent. An input query $q_0$ is given to the $N$ sub-agents. A sub-agent is any system that accepts as input a query and returns a corresponding reformulation. Thus, sub-agents can be heterogeneous.
Figure 1: a) A vanilla search system. The query $q_0$ is given to the system which outputs a result $a_0$. b) The search system with a reformulator. The reformulator queries the system with $q_0$ and its reformulations $\{q_1, \ldots, q_N\}$ and receives back the results $\{a_0, \ldots, a_N\}$. A selector then decides the best result $a_i$ for $q_0$. c) The proposed system. The original query is reformulated multiple times by different reformulators. Reformulations are used to obtain results from the search system, which are then sent to the aggregator, which picks the best result for the original query based on a learned weighted majority voting scheme. Reformulators are independently trained on disjoint partitions of the dataset thus increasing the variability of reformulations.

Here we train each sub-agent on a partition of the training set. The $i$-th agent queries the underlying search system with the reformulation $q_i$ and receives a result $a_i$. The set $\{(q_i, a_i)\,|\,0 \leq i \leq N\}$ is given to the aggregator, which then decides which result will be final.

3.3 SUB-AGENTS

The first step for training the new agent is to partition the training set. We randomly split it into equal-sized subsets. For an analysis of how other partitioning methods affect performance, see Appendix F. In our implementation, a sub-agent is a sequence-to-sequence model (Sutskever et al., 2014; Cho et al., 2014) trained on a partition of the dataset. It receives as an input the original query $q_0$ and outputs a list of reformulated queries $(q_i)$ using beam search.

Each reformulation $q_i$ is given to the same environment that returns a list of results $(a_1^i, \ldots, a_K^i)$ and their respective rewards $(r_1^i, \ldots, r_K^i)$. We then use REINFORCE (Williams, 1992) to train the sub-agent. At training time, instead of using beam search, we sample reformulations.

Note that we also add the identity agent (i.e., the reformulation is the original query) to the pool of sub-agents.

3.4 META-AGENT: AGGREGATOR

The aggregator receives as inputs $q_0$ and a list of candidate results $(a_1^i, \ldots, a_K^i)$ for each reformulation $q_i$. We first compute the set of unique results $a_j$ and two different scores for each result: the accumulated rank score $s_j^A$ and the relevance score $s_j^R$.

The accumulated rank score is computed as $s_j^A = \sum_{i=1}^{N} \frac{1}{\text{rank}_{i,j}}$, where $\text{rank}_{i,j}$ is the rank of the $j$-th result when retrieved using $q_i$. The relevance score $s_j^R$ is the prediction that the result $a_j$ is relevant to query $q_0$. It is computed as:

$$s_j^R = \sigma(W_2\text{ReLU}(W_1z_j + b_1) + b_2),$$

where

$$z_j = [f_{\text{CNN}}(q_0); f_{\text{BOW}}(a_j); f_{\text{CNN}}(q_0) - f_{\text{BOW}}(a_j); f_{\text{CNN}}(q_0) \odot f_{\text{BOW}}(a_j)],$$

and

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$
W₁ ∈ ℝ^{4D×D} and W₂ ∈ ℝ^{D×1} are weight matrices, b₁ ∈ ℝ^{D} and b₂ ∈ ℝ^{1} are biases. The brackets in [x; y] represent the concatenation of vectors x and y. The symbol ⊙ denotes the element-wise multiplication, σ is the sigmoid function, and ReLU is a Rectified Linear Unit function (Nair & Hinton, 2010). The function f_{CNN} is implemented as a CNN encoder followed by average pooling over the sequence (Kim, 2014). The function f_{BOW} is the average word embeddings of the result. At test time, the top-K answers with respect to s_j = s_j^A s_j^R are returned.

We train the aggregator with stochastic gradient descent (SGD) to minimize the cross-entropy loss:

\[ L = - \sum_{j \in J^*} \log(s_j^R) - \sum_{j \not\in J^*} \log(1 - s_j^R), \]

where J^* is the set of indexes of the ground-truth results. The architecture details and hyperparameters can be found in Appendix B.

4 DOCUMENT RETRIEVAL

We now present experiments and results in a document retrieval task. In this task, the goal is to rewrite a query so that the number of relevant documents retrieved by a search engine increases.

4.1 ENVIRONMENT

The environment receives a query as an action, and it returns a list of documents as an observation/state and a reward computed using a list of ground truth documents. We use Lucene\(^2\) in its default configuration as our search engine, with BM25 as the ranking function. The input is a query and the output is a ranked list of documents.

4.2 DATASETS

To train and evaluate the models, we use three datasets:

TREC-CAR: Introduced by Dietz et al. (2017), in this dataset the input query is the concatenation of a Wikipedia article title with the title of one of its section. The ground-truth documents are the paragraphs within that section. The corpus consists of all of the English Wikipedia paragraphs, except the abstracts. The released dataset has five predefined folds, and we use the first four as a training set (approx. 3M queries), and the remaining as a validation set (approx. 700k queries). The test set is the same used evaluate the submissions to TREC-CAR 2017 (approx. 1,800 queries).

JEOPARDY: This dataset was introduced by Nogueira & Cho (2016). The input is a Jeopardy! question. The ground-truth document is a Wikipedia article whose title is the answer to the question. The corpus consists of all English Wikipedia articles.

MSA: Introduced by Nogueira & Cho (2017), this dataset consists of academic papers crawled from Microsoft Academic API.\(^3\) A query is the title of a paper and the ground-truth answer consists of the papers cited within. Each document in the corpus consists of its title and abstract.

4.3 REWARD

Since the main goal of query reformulation is to increase the proportion of relevant documents returned, we use recall as the reward: \( R@K = \frac{|D_K \cap D^*|}{|D^*|} \), where \( D_K \) are the top-K retrieved documents and \( D^* \) are the relevant documents. We also experimented using as a reward other metrics such as NDCG, MAP, MRR, and R-Precision but these resulted in similar or slightly worse performance than Recall@40. Despite the agents optimizing for Recall, we report the main results in MAP as this is a more commonly used metric in information retrieval. For results in other metrics, see Appendix A.

\(^1\)In the preliminary experiments, we found CNNs to work better than LSTMs (Hochreiter & Schmidhuber, 1997).

\(^2\)https://lucene.apache.org/

\(^3\)https://www.microsoft.com/cognitive-services/en-us/academic-knowledge-api
<table>
<thead>
<tr>
<th></th>
<th>TREC-CAR</th>
<th>Jeopardy</th>
<th>MSA</th>
<th>Training (Days)</th>
<th>FLOPs ($\times 10^{18}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25</td>
<td>11.3</td>
<td>7.1</td>
<td>3.1</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PRF</td>
<td>11.6</td>
<td>12.2</td>
<td>3.4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>RM3</td>
<td>12.0</td>
<td>12.6</td>
<td>3.1</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>RL-RNN (Nogueira &amp; Cho, 2017)</td>
<td>12.8</td>
<td>15.0</td>
<td>4.1</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>RL-10-Ensemble</td>
<td>13.0</td>
<td>16.1</td>
<td>4.4</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>RL-10-Full</td>
<td>14.1</td>
<td>28.4</td>
<td>4.9</td>
<td>1</td>
<td>2.3</td>
</tr>
<tr>
<td>RL-10-Bagging</td>
<td>14.1</td>
<td>28.7</td>
<td>5.0</td>
<td>1</td>
<td>2.3</td>
</tr>
<tr>
<td>RL-10-Sub</td>
<td>14.3</td>
<td>29.6</td>
<td>5.5</td>
<td>1</td>
<td>2.3</td>
</tr>
<tr>
<td>RL-10-Sub (Pretrained)</td>
<td>14.4</td>
<td>29.8</td>
<td>5.4</td>
<td>$10^*+1$</td>
<td>4.6</td>
</tr>
<tr>
<td>RL-10-Full (Extra Budget)</td>
<td>14.8</td>
<td>30.3</td>
<td>5.6</td>
<td>10</td>
<td>23.0</td>
</tr>
<tr>
<td>RL-10-Full (Ensemble 10 Aggregators)</td>
<td>17.7</td>
<td>33.0</td>
<td>6.1</td>
<td>10</td>
<td>23.0</td>
</tr>
<tr>
<td>RM3 + BERT Aggregator</td>
<td>35.5</td>
<td>50.0</td>
<td>8.7</td>
<td>10</td>
<td>23.0</td>
</tr>
<tr>
<td>RL-10-Sub + BERT Aggregator</td>
<td>36.4</td>
<td>51.2</td>
<td>10.2</td>
<td>10</td>
<td>23.0</td>
</tr>
<tr>
<td>Best System of TREC-CAR 2017 (MacAvaney et al., 2017)</td>
<td>14.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: MAP scores on the test sets of the document retrieval datasets. Similar results hold for other metrics (see Appendix A). *The weights of the agents are initialized from a single model pretrained for ten days on the full training set.

### 4.4 Baselines

**BM25:** We give the original query to Lucene with BM25 as a ranking function and use the retrieved documents as results.

**PRF:** This is the pseudo relevance feedback method (Rocchio, 1971). We expand the original query with terms from the documents retrieved by the Lucene search engine using the original query. The top-N TF-IDF terms from each of the top-K retrieved documents are added to the original query, where N and K are selected by a grid search on the validation data.

**Relevance Model (RM3):** This is our implementation of the relevance model for query expansion (Lavrenko & Croft, 2001). The probability of adding a term $t$ to the original query is given by:

$$P(t|q_0) = (1 - \lambda) P'(t|q_0) + \lambda \sum_{d \in D_0} P(d) P(t|d) P(q_0|d), \quad (4)$$

where $P(d)$ is the probability of retrieving the document $d$, assumed uniform over the set, $P(t|d)$ and $P(q_0|d)$ are the probabilities assigned by the language model obtained from $d$ to $t$ and $q_0$, respectively. $P'(t|q_0) = \frac{tf(t,d)}{|d|}$, where $tf(t,d)$ is the term frequency of $t$ in $d$. We set the interpolation parameter $\lambda$ to 0.65, which was the best value found by a grid-search on the development set.

We use a Dirichlet smoothed language model (Zhai & Lafferty, 2001) to compute a language model from a document $d \in D_0$:

$$P(t|d) = \frac{tf(t,d) + uP(t|C)}{|d| + u}, \quad (5)$$

where $u$ is a scalar constant ($u = 1500$ in our experiments), and $P(t|C)$ is the probability of $t$ occurring in the entire corpus $C$.

We use the $N$ terms with the highest $P(t|q_0)$ in an expanded query, where $N = 100$ was the best value found by a grid-search on the development set.

**RL-RNN:** This is the sequence-to-sequence model trained with reinforcement learning from Nogueira & Cho (2017). The reformulated query is formed by appending new terms to the
original query. The terms are selected from the documents retrieved using the original query. The agent is trained from scratch.

**RL-N-ENSEMBLE:** We train \( N \) RL-RNN agents with different initial weights on the full training set. At test time, we average the probability distributions of all the \( N \) agents at each time step and select the token with the highest probability, as done by Sutskever et al. (2014).

### 4.5 Proposed Models

We evaluate the following variants of the proposed method:

**RL-N-FULL:** We train \( N \) RL-RNN agents with different initial weights on the full training set. The answers are obtained using the best (greedy) reformulations of all the agents and are given to the aggregator.

**RL-N-BAGGING:** This is the same as RL-N-Full but we construct the training set of each RL-RNN agent by sampling with replacement \( D \) times from the full training set, which has a size of \( D \). This is known as the bootstrap sample and leads to approximately 63% unique samples, the rest being duplicates.

**RL-N-SUB:** This is the proposed agent. It is similar to RL-N-Full but the multiple sub-agents are trained on random partitions of the dataset (see Figure 1-(c)).

**BERT AGGREGATOR:** We experimented replacing our simple aggregator with BERT (Devlin et al., 2018), which recently achieved the state-of-the-art in a wide range of textual tasks. Using the same notation used in their paper, we feed the query as sentence A and the document text as sentence B. We truncate the document text such that concatenation of query, document, and separator tokens have a maximum length of 512 tokens. We use a pretrained BERT\textsubscript{LARGE} model as a binary classification model, that is, we feed the [CLS] vector to a single layer neural network and obtain the probability of the document being correct. We obtain the final list of documents by ranking them with respect these probabilities. We train it with the same objective used to train our aggregator (Equation 3). To compare how well our proposed reformulation agents perform against the best non-neural reformulation method, we implemented two variants of the system described here. One is when the initial list of candidate documents \( a_j \) is given by RM3 (RM3 + BERT Aggregator), and the other is by RL-10-Sub (RL-10-Sub + BERT Aggregator).

### 4.6 Results

A summary of the document retrieval results is shown in Table 1. We estimate the number of floating point operations used to train a model by multiplying the training time, the number of GPUs used, and 2.7 TFLOPS as an estimate of the single-precision floating-point of a K80 GPU.

Since the sub-agents are frozen during the training of the aggregator, we pre-compute all \((q_0, q_i, a_i, r_i)\) tuples from the training set, thus avoiding sub-agent or environment calls. This reduces its training time to less than 6 hours \((0.06 \times 10^{18} \text{ FLOPs})\). Since this cost is negligible when compared to the sub-agents’, we do not include it in the table.

The proposed methods (RL-10-\{Sub, Bagging, Full\}) have 20-60% relative performance improvement over the standard ensemble (RL-10-Ensemble) while training ten times faster. More interestingly, RL-10-Sub has a better performance than the single-agent version (RL-RNN), uses the same computational budget, and trains on a fraction of the time. Lastly, we found that RL-10-Sub (Pretrained) has the best balance between performance and training cost across all datasets.

Compared to the top-performing system in the TREC-CAR 2017 Track (MacAvaney et al., 2017), an RL-10-Full with an ensemble of 10 aggregators yields a relative performance improvement of approximately 20%.

By replacing our aggregator with BERT, we improve performance by 50-100% in all three datasets (RL-10-Sub + BERT Aggregator). This is a remarkable improvement given that we used BERT
without any modification from its original implementation. Without using our reformulation agents, the performance drops by 3-10% (RM3 + BERT Aggregator).

For an analysis of the aggregator’s contribution to the overall performance, see Appendix C.

**NUMBER OF SUB-AGENTS:** We compare the performance of the full system (reformulators + aggregator) for different numbers of agents in Figure 2. The performance of the system is stable across all datasets after more than ten sub-agents are used, thus indicating the robustness of the proposed method. For more experiments regarding training stability, see Appendix D.

## 5 QUESTION-ANSWERING

To further assess the effectiveness of the proposed method, we conduct experiments in a question-answering task, comparing our agent with the active question answering agent proposed by Buck et al. (2018b).

The environment receives a question as an action and returns an answer as an observation, and a reward computed against a ground truth answer. We use either BiDAF (Seo et al., 2016) or BERT (Devlin et al., 2018) as a question-answering system. Given a question, it outputs an answer span from a list of snippets. We use as a reward the token level F1 score on the answer (see Section 5.3 for its definition).

We follow Buck et al. (2018b) to train BiDAF and BERT. We emphasize that their parameters are frozen when we train and evaluate the reformulation system. Training and evaluation are performed on the SearchQA dataset (Dunn et al., 2017). The data contains Jeopardy! clues as questions. Each clue has a correct answer and a list of 50 snippets from Google’s top search results. The training, validation and test sets contain 99,820, 13,393 and 27,248 examples, respectively.

### 5.1 Basielines and Benchmarks

We compare our agent against the following baselines and benchmarks:

**BiDAF/BERT:** The original question is given to the question-answering system without any modification (see Figure 1-(a)).

**AQA:** This is the best model from Buck et al. (2018b). It consists of a reformulator and a selector. The reformulator is a subword-based sequence-to-sequence model that produces twenty reformulations of an input question using beam search. The answers for the original question and its reformulations are obtained from BiDAF. These are given to the selector which then chooses one of the answers as final (see Figure 1-(b)). The reformulator is pretrained on translation and paraphrase data.

### 5.2 Proposed Methods

**AQA-N-{FULL, SUB}**: Similar to the RL-N-{Full, Sub} models, we use AQA reformulators as the sub-agents followed by an aggregator to create AQA-N-Full and AQA-N-Sub models, whose sub-agents are trained on the full and random partitions of the dataset, respectively. For the training and hyperparameter details, see Appendix B.2.
<table>
<thead>
<tr>
<th></th>
<th>Dev F1</th>
<th>Oracle</th>
<th>Test F1</th>
<th>Oracle</th>
<th>Training (Days)</th>
<th>FLOPs ($\times 10^{18}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiDAF (Seo et al., 2016)</td>
<td>37.9</td>
<td>-</td>
<td>34.6</td>
<td>-</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>$R^3$ (Wang et al., 2017a)</td>
<td>-</td>
<td>-</td>
<td>55.3</td>
<td>-</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Re-Ranker (Wang et al., 2017b)</td>
<td>-</td>
<td>-</td>
<td>60.6</td>
<td>-</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>DS-QA (Lin et al., 2018)</td>
<td>-</td>
<td>-</td>
<td>64.5</td>
<td>-</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>BERT (Devlin et al., 2018)</td>
<td>71.2</td>
<td>-</td>
<td>69.1</td>
<td>-</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>AQA (Buck et al., 2018b)</td>
<td>47.4</td>
<td>56.0</td>
<td>45.6</td>
<td>53.8</td>
<td>10</td>
<td>4.6</td>
</tr>
<tr>
<td>BiDAF + AQA-10-Sub</td>
<td>51.7</td>
<td>66.8</td>
<td>49.0</td>
<td>61.5</td>
<td>1</td>
<td>4.6</td>
</tr>
<tr>
<td>BiDAF + AQA-10-Full</td>
<td>51.0</td>
<td>61.2</td>
<td>48.4</td>
<td>58.7</td>
<td>1</td>
<td>4.6</td>
</tr>
<tr>
<td>BiDAF + AQA-10-Full (extra budget)</td>
<td>51.4</td>
<td>61.3</td>
<td>50.5</td>
<td>58.9</td>
<td>10</td>
<td>46.0</td>
</tr>
<tr>
<td>BERT + AQA-10-Sub</td>
<td>71.2</td>
<td>76.8</td>
<td>69.1</td>
<td>75.4</td>
<td>1</td>
<td>4.6</td>
</tr>
</tbody>
</table>

Table 2: Main result on the question-answering task (SearchQA dataset). We did not include the training cost of the aggregator (0.2 days, 0.06 $\times 10^{18}$ FLOPs).

5.3 Evaluation Metrics

**F1:** We use the macro-averaged F1 score as the main metric. It measures the average bag of tokens overlap between the prediction and ground truth answer. We take the F1 over the ground truth answer for a given question and then average over all of the questions.

**Oracle:** Additionally, we present the oracle performances, which are from a perfect aggregator that predicts $s^R_j = 1$ for relevant answers and $s^R_j = 0$, otherwise.

5.4 Results

Results are presented in Table 2. When using BiDAF as the underlying Q&A system, the proposed method (AQA-10-{Full, Sub}) have both better F1 and oracle performances than the single-agent AQA method, while training in one-tenth of the time. Even when the ensemble method is given ten times more training time (AQA-10-Full, extra budget), our method achieves a higher performance.

We achieve the state-of-the-art on SearchQA by using BERT without any modification from its original implementation. Our reformulation strategy (BERT + AQA-10-Sub), however, could not improve upon this underlying Q&A system. We conjecture that, although there is room for improvement, as the oracle performance is 5-7% higher than BERT alone, the reformulations and answers do not contain enough information for the aggregator to discriminate good from bad answers. One possible way to fix this is to give the context of the answer to the aggregator, although in our experiments we could not find any successful way to use this extra information.

**Original Query Contribution:** We observe a drop in F1 of approximately 1% when the original query is removed from the pool of reformulations, which shows that the gains come mostly from the multiple reformulations and not from the aggregator falling back on selecting the original query.

5.5 Query Diversity

Here we evaluate how query diversity and performance are related. For that, we use four metrics (defined in Appendix E): pCos, pBLEU, PINC, and Length Std.

Table 3 shows that the multiple agents trained on partitions of the dataset (AQA-10-Sub) produce more diverse queries than a single agent with beam search (AQA) and multiple agents trained on the full training set (AQA-10-Full). This suggests that its higher performance can be partly attributed to the higher diversity of the learned policies.
Method | pCos ↓ | pBLEU ↓ | PINC ↑ | Length Std ↑ | F1 ↑ | Oracle ↑
---|---|---|---|---|---|---
AQA | 66.4 | 45.7 | 58.7 | 3.8 | 47.7 | 56.0
AQA-10-Full | 29.5 | 26.6 | 79.5 | 9.2 | 51.0 | 61.2
AQA-10-Sub | 14.2 | 12.8 | 94.5 | 11.7 | 51.4 | 61.3

Table 3: Diversity scores of reformulations from different methods. For pBLEU and pCos, lower values mean higher diversity. Notice that higher diversity scores are associated with higher F1 and oracle scores.

6 CONCLUSION

We proposed a method to build a better query reformulation system by training multiple sub-agents on partitions of the data using reinforcement learning and an aggregator that learns to combine the answers of the multiple agents given a new query. We showed the effectiveness and efficiency of the proposed approach on the tasks of document retrieval and question answering. One interesting orthogonal extension would be to introduce diversity on the beam search decoder (Vijayakumar et al., 2016; Li et al., 2016), thus shedding light on the question of whether the gains come from the increased capacity of the system due to the use of the multiple agents, the diversity of reformulations, or both.

REFERENCES


Christopher Stanton and Jeff Clune. Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. *PloS one*, 11(9):e0162235, 2016.


APPENDIX A DOCUMENT RETRIEVAL: RESULTS ON MORE METRICS

Following Dietz et al. (2017), we report the results on four standard TREC evaluation measures: R-Precision (R-Prec), Mean-average Precision (MAP), Reciprocal Rank (MRR), and Normalize Discounted Cumulative Gain (NDCG). We also include Recall@40 as this is the reward our agents are optimizing for. The results for TREC-CAR, Jeopardy, and MSA are in Tables 4, 5, 6, respectively.

APPENDIX B HYPERPARAMETERS

B.1 DOCUMENT RETRIEVAL TASK

SUB-AGENTS: We use mini-batches of size 256, ADAM (Kingma & Ba, 2014) as the optimizer, and learning rate of $10^{-4}$.

AGGREGATOR: The encoder $f_{d_0}$ is a word-level two-layer CNN with filter sizes of 9 and 3, respectively, and 128 and 256 kernels, respectively. $D = 512$. No dropout is used. ADAM is the optimizer with learning rate of $10^{-4}$ and mini-batch of size 64. It is trained for 100 epochs.

B.2 QUESTION-ANSWERING TASK

SUB-AGENTS: We use mini-batches of size 64, SGD as the optimizer, and learning rate of $10^{-3}$.

AGGREGATOR: The encoder $f_{q_0}$ is a token-level, three-layer CNN with filter sizes of 3, and 128, 256, and 256 kernels, respectively. We train it for 100 epochs with mini-batches of size 64 with SGD and learning rate of $10^{-3}$.

APPENDIX C AGGREGATOR ANALYSIS

C.1 CONTRIBUTION OF THE AGGREGATOR VS. MULTIPLE REFORMULATORS

To isolate the contribution of the Aggregator from the gains brought by the multiple reformulators, we use the aggregator to re-rank the list of documents obtained with the rewrite from a single reformulator (RL-RNN Greedy + Aggregator). We also use beam search or sampling to produce $K$ rewrites from a single reformulator (RL-RNN $K$ Sampled/Beam + Aggregator). The $K$ lists of ranked documents returned by the environment are then merged into a single list and re-ranked by the Aggregator.

The results are shown in table 7. The higher performance obtained with ten rewrites produced by different reformulators (RL-10-Sub) when compared 20 sampled rewrites from a single agent (RL-RNN 20 Sampled + Aggregator) indicates that the gains the proposed method comes mostly from the pool of diverse reformulators, and not from the simple use of a re-ranking function (Aggregator).

C.2 ABLATION STUDY

To validate the effectiveness of the proposed aggregation function, we conducted a comparison study on the TREC-CAR dataset. We present the results in Table 8. We notice that removing or changing the accumulated rank or relevance score functions results in a performance drop between 0.4-1.4% in MAP. The largest drop occurs when we remove the aggregated rank ($s_j = s_j^A$), suggesting that the rank of a document obtained from the reformulation phase is a helpful signal to the re-ranking phase.

Not reported in the table, we also experimented concatenating to the input vector $z_i$ (eq. 2) a vector to represent each sub-agent. These vectors were learned during training and allowed the aggregator to distinguish sub-agents. However, we did not notice any performance improvement.
<table>
<thead>
<tr>
<th></th>
<th>R@40</th>
<th>MAP</th>
<th>R-Prec</th>
<th>MRR</th>
<th>NDCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25</td>
<td>27.5</td>
<td>11.3</td>
<td>9.8</td>
<td>21.0</td>
<td>27.4</td>
</tr>
<tr>
<td>PRF</td>
<td>28.6</td>
<td>11.7</td>
<td>10.1</td>
<td>21.7</td>
<td>27.2</td>
</tr>
<tr>
<td>RM3</td>
<td>29.7</td>
<td>12.1</td>
<td>10.5</td>
<td>22.5</td>
<td>27.2</td>
</tr>
<tr>
<td>RL-RNN</td>
<td>31.6</td>
<td>12.7</td>
<td>10.9</td>
<td>23.6</td>
<td>28.3</td>
</tr>
<tr>
<td>RL-10-Ensemble</td>
<td>31.7</td>
<td>12.8</td>
<td>11.0</td>
<td>23.8</td>
<td>28.3</td>
</tr>
<tr>
<td>RL-RNN Greedy + Aggregator</td>
<td>32.0</td>
<td>12.8</td>
<td>11.1</td>
<td>23.2</td>
<td>29.0</td>
</tr>
<tr>
<td>RL-RNN 20 Sampled + Aggregator</td>
<td>32.5</td>
<td>12.0</td>
<td>11.2</td>
<td>23.1</td>
<td>29.3</td>
</tr>
<tr>
<td>RL-RNN 20 Beam + Aggregator</td>
<td>32.3</td>
<td>12.9</td>
<td>11.1</td>
<td>23.0</td>
<td>29.2</td>
</tr>
<tr>
<td>RL-10-Full</td>
<td>35.2</td>
<td>14.1</td>
<td>12.0</td>
<td>24.5</td>
<td>29.5</td>
</tr>
<tr>
<td>RL-10-Bagging</td>
<td>35.9</td>
<td>14.1</td>
<td>12.1</td>
<td>24.6</td>
<td>29.7</td>
</tr>
<tr>
<td>RL-10-Sub</td>
<td>36.7</td>
<td>14.2</td>
<td>12.1</td>
<td>25.0</td>
<td>29.5</td>
</tr>
<tr>
<td>RL-10-Sub (Pretrained)</td>
<td>36.9</td>
<td>14.4</td>
<td>12.3</td>
<td>25.0</td>
<td>29.8</td>
</tr>
<tr>
<td>RL-10-Full (Extra Budget)</td>
<td>37.7</td>
<td>14.8</td>
<td>12.5</td>
<td>25.3</td>
<td>29.1</td>
</tr>
<tr>
<td>RL-10-Full (Ensemble of 10 Aggregators)</td>
<td>39.5</td>
<td>17.7</td>
<td>13.5</td>
<td>26.3</td>
<td>30.8</td>
</tr>
<tr>
<td>RM3 + BERT Aggregator</td>
<td>50.9</td>
<td>35.5</td>
<td>32.1</td>
<td>51.1</td>
<td>45.2</td>
</tr>
<tr>
<td>RL-10-Full + BERT Aggregator</td>
<td>52.0</td>
<td>36.4</td>
<td>32.6</td>
<td>52.0</td>
<td>46.6</td>
</tr>
<tr>
<td>Best System of TREC-CAR 2017 (MacAvaney et al., 2017)</td>
<td>-</td>
<td>14.8</td>
<td>11.6</td>
<td>22.3</td>
<td>22.8</td>
</tr>
</tbody>
</table>

Table 4: Results on more metrics on the test set of the TREC-CAR dataset.

<table>
<thead>
<tr>
<th></th>
<th>R@40</th>
<th>MAP</th>
<th>R-Prec</th>
<th>MRR</th>
<th>NDCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25</td>
<td>23.0</td>
<td>7.1</td>
<td>3.8</td>
<td>7.1</td>
<td>10.5</td>
</tr>
<tr>
<td>PRF</td>
<td>29.7</td>
<td>12.2</td>
<td>7.8</td>
<td>12.2</td>
<td>16.0</td>
</tr>
<tr>
<td>RM3</td>
<td>30.5</td>
<td>12.6</td>
<td>8.1</td>
<td>12.6</td>
<td>16.5</td>
</tr>
<tr>
<td>RL-RNN</td>
<td>33.7</td>
<td>15.0</td>
<td>10.0</td>
<td>15.0</td>
<td>19.1</td>
</tr>
<tr>
<td>RL-10-Ensemble</td>
<td>35.2</td>
<td>16.1</td>
<td>10.8</td>
<td>16.1</td>
<td>20.3</td>
</tr>
<tr>
<td>RL-RNN Greedy + Aggregator</td>
<td>42.0</td>
<td>21.2</td>
<td>14.9</td>
<td>21.2</td>
<td>25.8</td>
</tr>
<tr>
<td>RL-RNN 20 Sampled + Aggregator</td>
<td>42.4</td>
<td>21.5</td>
<td>15.1</td>
<td>21.5</td>
<td>26.1</td>
</tr>
<tr>
<td>RL-RNN 20 Beam + Aggregator</td>
<td>42.3</td>
<td>21.4</td>
<td>15.0</td>
<td>21.4</td>
<td>26.0</td>
</tr>
<tr>
<td>RL-10-Full</td>
<td>52.1</td>
<td>28.4</td>
<td>20.5</td>
<td>28.4</td>
<td>33.7</td>
</tr>
<tr>
<td>RL-10-Bagging</td>
<td>52.5</td>
<td>28.7</td>
<td>20.8</td>
<td>28.7</td>
<td>34.0</td>
</tr>
<tr>
<td>RL-10-Sub</td>
<td>53.5</td>
<td>29.7</td>
<td>21.5</td>
<td>29.7</td>
<td>35.0</td>
</tr>
<tr>
<td>RL-10-Sub (Pretrained)</td>
<td>54.0</td>
<td>29.8</td>
<td>21.6</td>
<td>29.8</td>
<td>35.2</td>
</tr>
<tr>
<td>RL-10-Full (Extra Budget)</td>
<td>54.4</td>
<td>30.3</td>
<td>22.1</td>
<td>30.3</td>
<td>35.8</td>
</tr>
<tr>
<td>RM3 + BERT Aggregator</td>
<td>60.3</td>
<td>50.0</td>
<td>31.0</td>
<td>44.4</td>
<td>48.9</td>
</tr>
<tr>
<td>RL-10-Full + BERT Aggregator</td>
<td>61.5</td>
<td>36.4</td>
<td>32.2</td>
<td>46.0</td>
<td>50.1</td>
</tr>
</tbody>
</table>

Table 5: Results on more metrics on the test set of the Jeopardy dataset.
<table>
<thead>
<tr>
<th></th>
<th>R@40</th>
<th>MAP</th>
<th>R-Prec</th>
<th>MRR</th>
<th>NDCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25</td>
<td>12.7</td>
<td>3.1</td>
<td>6.0</td>
<td>15.4</td>
<td>9.1</td>
</tr>
<tr>
<td>PRF</td>
<td>13.2</td>
<td>3.4</td>
<td>6.4</td>
<td>16.2</td>
<td>9.7</td>
</tr>
<tr>
<td>RM3</td>
<td>12.3</td>
<td>3.1</td>
<td>6.0</td>
<td>15.0</td>
<td>8.9</td>
</tr>
<tr>
<td>RL-RNN</td>
<td>15.1</td>
<td>4.1</td>
<td>7.3</td>
<td>18.8</td>
<td>11.2</td>
</tr>
<tr>
<td>RL-10-ensemble</td>
<td>15.8</td>
<td>4.4</td>
<td>7.7</td>
<td>19.7</td>
<td>11.7</td>
</tr>
<tr>
<td>RL-RNN Greedy + Aggregator</td>
<td>16.1</td>
<td>4.5</td>
<td>7.8</td>
<td>20.1</td>
<td>12.0</td>
</tr>
<tr>
<td>RL-RNN 20 Sampled + Aggregator</td>
<td>16.4</td>
<td>4.6</td>
<td>7.9</td>
<td>20.5</td>
<td>12.2</td>
</tr>
<tr>
<td>RL-RNN 20 Beam + Aggregator</td>
<td>16.2</td>
<td>4.5</td>
<td>7.9</td>
<td>20.3</td>
<td>12.1</td>
</tr>
<tr>
<td>RL-10-Full</td>
<td>17.4</td>
<td>4.9</td>
<td>8.4</td>
<td>21.9</td>
<td>13.0</td>
</tr>
<tr>
<td>RL-10-Bagging</td>
<td>17.6</td>
<td>5.0</td>
<td>8.5</td>
<td>22.1</td>
<td>13.2</td>
</tr>
<tr>
<td>RL-10-Sub</td>
<td>18.9</td>
<td>5.5</td>
<td>9.2</td>
<td>23.9</td>
<td>14.2</td>
</tr>
<tr>
<td>RL-10-Sub (Pretrained)</td>
<td>19.1</td>
<td>5.4</td>
<td>9.1</td>
<td>24.0</td>
<td>14.2</td>
</tr>
<tr>
<td>RL-10-Full (Extra Budget)</td>
<td>19.2</td>
<td>5.6</td>
<td>9.3</td>
<td>24.3</td>
<td>14.4</td>
</tr>
<tr>
<td>RM3 + BERT Aggregator</td>
<td>24.7</td>
<td>8.7</td>
<td>12.0</td>
<td>36.4</td>
<td>20.9</td>
</tr>
<tr>
<td>RL-10-Full + BERT Aggregator</td>
<td>25.9</td>
<td>10.2</td>
<td>13.4</td>
<td>37.8</td>
<td>21.8</td>
</tr>
</tbody>
</table>

Table 6: Results on more metrics on the test set of the MSA dataset.

<table>
<thead>
<tr>
<th></th>
<th>TREC-CAR</th>
<th>Jeopardy</th>
<th>MSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>RL-RNN</td>
<td>10.8</td>
<td>15.0</td>
<td>4.1</td>
</tr>
<tr>
<td>RL-RNN Greedy + Aggregator</td>
<td>10.9</td>
<td>21.2</td>
<td>4.5</td>
</tr>
<tr>
<td>RL-RNN 20 Sampled + Aggregator</td>
<td>11.1</td>
<td>21.5</td>
<td>4.6</td>
</tr>
<tr>
<td>RL-RNN 20 Beam + Aggregator</td>
<td>11.0</td>
<td>21.4</td>
<td>4.5</td>
</tr>
<tr>
<td>RL-10-Sub</td>
<td>12.3</td>
<td>29.7</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Table 7: Multiple reformulators vs. aggregator contribution. Numbers are MAP scores on the dev set. Using a single reformulator with the aggregator (RL-RNN Greedy/Sampled/Beam + Aggregator) improves performance by a small margin over the single reformulator without the aggregator (RL-RNN). Using ten reformulators with the aggregator (RL-10-Sub) leads to better performance, thus indicating that the pool of diverse reformulators is responsible for most of the gains of the proposed method.

<table>
<thead>
<tr>
<th>Aggregator Function</th>
<th>MAP</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_j = s_j^A s_j^R ) (proposed, Section 3.4)</td>
<td>12.3</td>
<td>-</td>
</tr>
<tr>
<td>( z_j = f_{CNN}(q_0)</td>
<td></td>
<td>f_{BOW}(a_j) ) (eq. 2)</td>
</tr>
<tr>
<td>( s_j^A = \sum_{i=1}^{N} \mathbb{1}_{a_i = a_j} )</td>
<td>11.7</td>
<td>-0.6</td>
</tr>
<tr>
<td>( s_j = s_j^A )</td>
<td>11.1</td>
<td>-1.2</td>
</tr>
<tr>
<td>( s_j = s_j^R )</td>
<td>10.9</td>
<td>-1.4</td>
</tr>
</tbody>
</table>

Table 8: Comparison of different aggregator functions on TREC-CAR. The reformulators are from RL-10-Sub.
Table 9: Partitioning strategies and the corresponding evaluation metrics. We notice that the random strategy generally results in the best quality sub-agents, leading to the best scores on both of the tasks.

### Appendix D Training Stability of Single vs. Multi-Agent

Reinforcement learning algorithms that use non-linear function approximators, such as neural networks, are known to be unstable (Tsitsiklis & Van Roy, 1996; Fairbank & Alonso, 2011; Pirotta et al., 2013; Mnih et al., 2015). Ensemble methods are known to reduce this variance (Freund, 1995; Breiman, 1996a:b). Since the proposed method can be viewed as an ensemble, we compare the AQA-10-Sub’s F1 variance against a single agent (AQA) on ten runs. Our method has a much smaller variance: 0.20 vs. 1.07. We emphasize that it also has a higher performance than the AQA-10-Ensemble.

We argue that the higher stability is due to the use of multiple agents. Answers from agents that diverged during training can be discarded by the aggregator. In the single-agent case, answers come from only one, possibly bad, policy.

### Appendix E Diversity Metrics

Here we define the metrics used in query diversity analysis (Sec. 5.5):

**PCos**: Mean pair-wise cosine distance: $\frac{1}{N} \sum_{n=1}^{N} \frac{1}{|Q^n|} \sum_{q,q' \in Q^n} \cos(\#q, \#q')$, where $Q^n$ is a set of reformulated queries for the $n$-th original query in the development set and $\#q$ is the token count vector of $q$.

**PBLEU**: Mean pair-wise sentence-level BLEU (Chen & Cherry, 2014):

$\frac{1}{N} \sum_{n=1}^{N} \frac{1}{|Q^n|} \sum_{q,q' \in Q^n} \text{BLEU}(q, q')$.

**PINC**: Mean pair-wise paraphrase in k-gram changes (Chen & Dolan, 2011):

$\frac{1}{N} \sum_{n=1}^{N} \frac{1}{|Q^n|} \sum_{q,q' \in Q^n} \frac{1}{K} \sum_{k=1}^{K} 1 - \frac{|\text{k-gram}_q \cap \text{k-gram}_{q'}|}{|\text{k-gram}_q|}$, where $K$ is the maximum number of k-grams considered (we use $K = 4$).

**LENGTH STD**: Standard deviation of the reformulation lengths: $\frac{1}{N} \sum_{n=1}^{N} \text{std}(\{|q^n_i|\}_{i=1}^{|Q^n|})$

### Appendix F On Data Partitioning

Throughout this paper, we used sub-agents trained on random partitions of the dataset. We now investigate how different data partitioning strategies affect final performance of the system. Specifically, we compare the random split against a mini-batch K-means clustering algorithm (Sculley, 2010).

**Balanced K-means Clustering** For K-means, we experimented with three types of features: average question embedding (Q), average answer embedding (A), and the concatenation of these two (Q+A). The word embeddings were obtained from Mikolov et al. (2013).

The clusters returned by the K-means can be highly unbalanced. This is undesirable since some sub-agents might end up being trained with too few examples and thus may have a worse generalization performance than the others. To address this problem, we use a greedy cluster balancing algorithm as a post-processing step (see Algorithm 1 for the pseudocode).
Algorithm 1 Cluster Balancing

1: Given: desired cluster size $M$, and a set of clusters $C$, each containing a set of items.
2: sort $C$ by descending order of sizes
3: $C_{\text{remaining}} \leftarrow$ shallow_copy($C$)
4: for $c$ in $C$
5: remove $c$ from $C_{\text{remaining}}$
6: while $c$.size $< M$
7: item $\leftarrow$ randomly select an item from $c$
8: move item to the closest cluster in $C_{\text{remaining}}$
9: sort $C_{\text{remaining}}$ by descending order of sizes
10: end while
11: end for
12: return $C$

Evaluation Metric

In order to gain insight into the effect of a partitioning strategy, we first define three evaluation metrics. Let $\pi_i$ be the $i$-th sub-agent trained on the $i$-th partition out of $K$ partitions obtained from clustering. We further use $s_{ij}$ to denote the score, either F-1 in the case of question answering or R@40 for document retrieval, obtained by the $i$-th sub-agent $\pi_i$ on the $j$-th partition.

Out-of-partition score computes the generalization capability of the sub-agents outside the partitions on which they were trained:

$$E_i[E_{j \neq i}[s_{ij}]] = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{K-1} \sum_{j \neq i} s_{ij}.$$  

This score reflects the general quality of the sub-agents. Out-of-partition variance computes how much each sub-agent’s performance on the partitions, on which it was not trained, varies:

$$E_i[V_{j \neq i}[s_{ij}]] = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{K-2} \sum_{j \neq i} (s_{ij} - E_{i[j \neq i]}[s_{ij}])^2. \quad (6)$$

It indicates the general stability of the sub-agents. If it is high, it means that the sub-agent must be carefully combined in order for the overall performance to be high. Out-of-partition error computes the generalization gap between the partition on which the sub-agent was trained and the other partitions:

$$E_i[e_i] = \frac{1}{N} \sum_{i=1}^{N} (s_{ij} - E_{j \neq i}[s_{ij}]).$$

This error must be low, and otherwise, would indicate that each sub-agent has overfit the particular partition, implying the worse generalization.

Result

We present the results in Table 9. Although we could obtain a good result with the clustering-based strategy, we notice that this strategy is highly sensitive to the choice of features. Q+A is optimal for SearchQA, while $A$ is for TREC-CAR. On the other hand, the random strategy performs stably across both of the tasks, making it a preferred strategy. Based on comparing $Q$ and $Q+A$ for SearchQA, we conjecture that it is important to have sub-agents that are not specialized too much to their own partitions for the proposed approach to work well. Furthermore, we see that the absolute performance of the sub-agents alone is not the best proxy for the final performance, based on TREC-CAR.

Appendix G Reformulation Examples

Table 10 shows four reformulation examples by various methods. The proposed method (AQA-10-Sub) performs better in the first and second examples than the other methods. Note that, despite the large diversity of reformulations, BiDAF still returns the correct answer.

In the third example, the proposed method fails to produce the right answer whereas the other methods perform well. In the fourth example, despite the correct answer is in the set of returned answers, the aggregator fails to set a high score for it.
Table 10: Examples for the qualitative analysis on SearchQA. In **bold** are the reformulations and answers that had the highest scores predicted by the aggregator. We only show the top-5 reformulations of each method. For a detailed analysis of the language learned by the reformulator agents, see Buck et al. (2018a).