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 are normally 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.

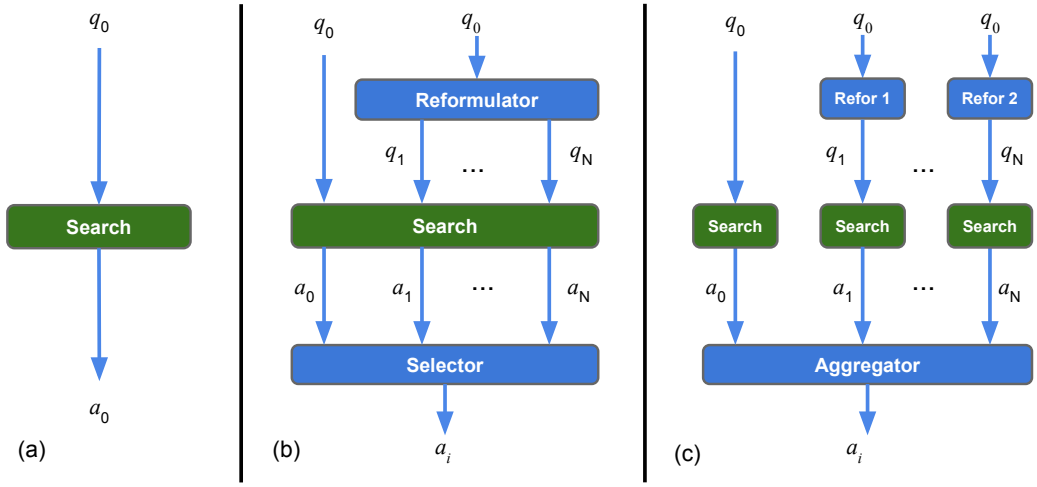

Figure 1: **a)** A vanilla search system. The query q_0 is given to the system which outputs a result a_0. **b)** The search system with a reformulator. The reformulator queries the system with q_0 and its reformulations $\{q_1, ...q_N\}$ and receives back the results $\{a_0, ..., a_N\}$. A selector 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 $(a_i^1, .., a_i^K)$ and their respective rewards $(r_i^1, ..r_i^K)$. 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 $(a_i^1, ..a_i^K)$ 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:

$$s_j^R = \sigma(W_2 \text{ReLU}(W_1 z_j + b_1) + b_2), \tag{1}$$

where

$$z_j = f_{\text{CNN}}(q_0)||f_{\text{BOW}}(a_j)||f_{\text{CNN}}(q_0) - f_{\text{BOW}}(a_j)||f_{\text{CNN}}(q_0) \odot f_{\text{BOW}}(a_j), \tag{2}$$

$W_1 \in \mathbb{R}^{4D \times D}$ and $W_2 \in \mathbb{R}^{D \times 1}$ are weight matrices, $b_1 \in \mathbb{R}^D$ and $b_2 \in \mathbb{R}^1$ are biases. The symbol $||$ denotes the concatenation operation, 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[1] 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:

$$L = -\sum_{j \in J^*} \log(s_j^R) - \sum_{j \notin J^*} \log(1 - s_j^R), \tag{3}$$

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[3] as our search engine. 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 & Ben (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.[4] 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] The ranking function is BM25.

[4] https://www.microsoft.com/cognitive-services/en-us/academic-knowledge-api

	TREC-CAR	Jeopardy	MSA	Training (Days)	FLOPs ($\times 10^{18}$)
Lucene	9.4	7.1	3.1	N/A	
PRF	9.8	12.2	3.4	N/A	
RM3	10.2	12.6	3.1	N/A	
RL-RNN (Nogueira & Cho, 2017)	10.8	15.0	4.1	10	2.3
RL-10-Ensemble	10.9	16.1	4.4	10	23.0
RL-RNN Greedy + Aggregator	10.9	21.2	4.5	10	2.3
RL-RNN 20 Sampled + Aggregator	11.1	21.5	4.6	10	2.3
RL-RNN 20 Beam + Aggregator	11.0	21.4	4.5	10	2.3
RL-10-Full	12.2	28.4	4.9	1	2.3
RL-10-Bagging	12.2	28.7	5.0	1	2.3
RL-10-Sub	12.3	29.7	5.5	1	2.3
RL-10-Sub (Pretrained)	12.5	29.8	5.4	10*+1	4.6
RL-10-Full (Extra Budget)	12.9	30.3	5.6	10	23.0

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 10 days on the full training set.

4.4 BASELINES

LUCENE: We give the original query to Lucene 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:

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

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. $P'(t|q_0) = \frac{\text{tf}(t \in q)}{|q|}$, where $\text{tf}(t, d)$ is the term frequency of t in d. We set the interpolation parameter λ 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 $d \in D_0$:

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

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.

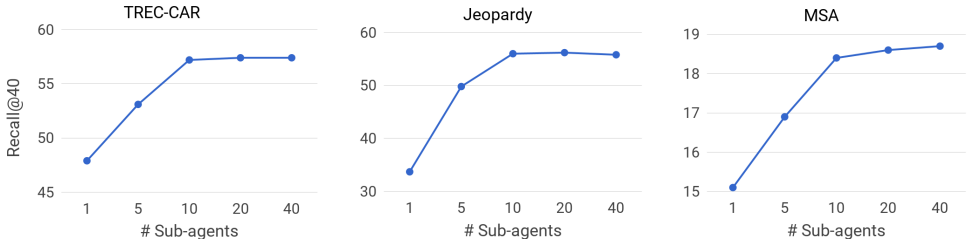

Figure 2: Overall system's performance for different number of sub-agents.

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)).

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 (0.06×10^{18} 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.

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.

	Dev		Test		Training	FLOPs
	F1	Oracle	F1	Oracle	(Days)	($\times 10^{18}$)
BiDAF (Seo et al., 2016)	37.9	-	34.6	-	N/A	
R^3 (Wang et al., 2017a)	-	-	55.3	-	N/A	
Re-Ranker (Wang et al., 2017b)	-	-	60.6	-	N/A	
AQA (Buck et al., 2018b)	47.4	56.0	45.6	53.8	10	4.6
AQA-10-Sub	51.7	66.8	49.0	61.5	1	4.6
AQA-10-Full	51.0	61.2	48.4	58.7	1	4.6
AQA-10-Full (extra budget)	51.4	61.3	50.5	58.9	10	46.0

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 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 BiDAF as the question-answering system (Seo et al., 2016). 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. We emphasize that BiDAF's 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 BASELINES AND BENCHMARKS

We compare our agent against the following baselines and benchmarks:

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

RE-RANKER AND R^3: Re-Ranker is the best model from Wang et al. (2017b). They use an answer re-ranking approach to reorder the answer candidates generated by a base Q&A model, R^3 (Wang et al., 2017a). We report both systems' results as a reference. To the best of our knowledge, they are currently the best systems on SearchQA. R^3 alone, without re-ranking, outperforms BiDAF by about 20 F1 points.

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 reformulations and their answers 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.

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.

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 $s_j^R = 1$ for relevant answers and $s_j^R = 0$, otherwise.

5.4 Results

Results are presented in Table 2. 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.

The best model outperforms BiDAF, which is used in our environment, by almost 16 F1 points. In absolute terms, the proposed method does not reach the performance of the Re-Ranker or underlying R^3 system. It is important to realize, though, that these are orthogonal issues: any Q&A system, including R^3, could be used as environments, including re-ranking post-processing. We leave this as a future work.

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.

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

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*, 2018.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996a.

Leo Breiman. Bias, variance, and arcing classifiers (technical report 460). *Statistics Department, University of California*, 1996b.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Christian Buck, Jannis Bulian, Massimiliano Ciaramita, Wojciech Gajewski, Andrea Gesmundo, Neil Houlsby, and Wei Wang. Analyzing language learned by an active question answering agent. *arXiv preprint arXiv:1801.07537*, 2018a.

Christian Buck, Jannis Bulian, Massimiliano Ciaramita, Andrea Gesmundo, Neil Houlsby, Wojciech Gajewski, and Wei Wang. Ask the right questions: Active question reformulation with reinforcement learning. In *Proceedings of ICLR*, 2018b.

Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 362–367, 2014.

David L Chen and William B Dolan. Collecting highly parallel data for paraphrase evaluation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 190–200. Association for Computational Linguistics, 2011.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13(1):227–303, 2000.

Laura Dietz and Gamari Ben. Trec car: A data set for complex answer retrieval. *http://trec-car.cs.unh.edu*, 2017.

Matthew Dunn, Levent Sagun, Mike Higgins, Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

Michael Fairbank and Eduardo Alonso. The divergence of reinforcement learning algorithms with value-iteration and function approximation. *arXiv preprint arXiv:1107.4606*, 2011.

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121 (2):256–285, 1995.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

Minghao Hu, Yuxing Peng, and Xipeng Qiu. Reinforced mnemonic reader for machine comprehension. *CoRR, abs/1705.02798*, 2017.

Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. Pacrr: A position-aware neural ir model for relevance matching. *arXiv preprint arXiv:1704.03940*, 2017.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Victor Lavrenko and W Bruce Croft. Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 120–127. ACM, 2001.

Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.

Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Rodrigo Nogueira and Kyunghyun Cho. End-to-end goal-driven web navigation. In *Advances in Neural Information Processing Systems*, pp. 1903–1911, 2016.

Rodrigo Nogueira and Kyunghyun Cho. Task-oriented query reformulation with reinforcement learning. *arXiv preprint arXiv:1704.04572*, 2017.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Adaptive step-size for policy gradient methods. In *Advances in Neural Information Processing Systems*, pp. 1394–1402, 2013.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Joseph John Rocchio. Relevance feedback in information retrieval. *The SMART retrieval system: experiments in automatic document processing*, pp. 313–323, 1971.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pp. 1177–1178. ACM, 2010.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

Iulian V Serban, Chinnadhurai Sankar, Mathieu Germain, Saizheng Zhang, Zhouhan Lin, Sandeep Subramanian, Taesup Kim, Michael Pieper, Sarath Chandar, Nan Rosemary Ke, et al. A deep reinforcement learning chatbot. *arXiv preprint arXiv:1709.02349*, 2017.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Satinder P Singh. Reinforcement learning with a hierarchy of abstract models. In *AAAI*, pp. 202–207, 1992.

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.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

JN Tsitsiklis and B Van Roy. An analysis of temporal-difference learning with function approximationtechnical. Technical report, Report LIDS-P-2322). Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1996.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*, 2016.

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, and Jing Jiang. R3: Reinforced reader-ranker for open-domain question answering. *arXiv preprint arXiv:1709.00023*, 2017a.

Shuohang Wang, Mo Yu, Jing Jiang, Wei Zhang, Xiaoxiao Guo, Shiyu Chang, Zhiguo Wang, Tim Klinger, Gerald Tesauro, and Murray Campbell. Evidence aggregation for answer re-ranking in open-domain question answering. *arXiv preprint arXiv:1711.05116*, 2017b.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 334–342. ACM, 2001.

APPENDIX A DOCUMENT RETRIEVAL: RESULTS ON MORE METRICS

Following Dietz & Ben (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_{q_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 ($s_j = s_j^R$), 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.

	R@40	MAP	R-Prec	MRR	NDCG
Lucene	25.7	9.4	8.3	17.7	15.4
PRF	26.8	9.8	8.6	18.4	16.1
RM3	28.0	10.2	9.0	19.2	16.8
RL-RNN	29.8	10.8	9.4	20.3	17.8
RL-10-Ensemble	30.1	10.9	9.5	20.5	18.0
RL-RNN Greedy + Aggregator	30.2	10.9	9.6	20.5	18.0
RL-RNN 20 Sampled + Aggregator	30.7	11.1	9.7	20.8	18.3
RL-RNN 20 Beam + Aggregator	30.5	11.0	9.6	20.7	18.2
RL-10-Full	33.9	12.2	10.5	22.8	20.2
RL-10-Bagging	34.1	12.2	10.6	22.9	20.3
RL-10-Sub	34.9	12.3	10.6	23.2	20.5
RL-10-Sub (Pretrained)	35.1	12.5	10.8	23.5	20.8
RL-10-Full (Extra Budget)	35.9	12.9	11.0	24.1	21.1
RL-10-Full (Ensemble of 10 Aggregators)	37.7	14.3	12.6	25.8	23.0
2017 Winner Entry (Hui et al., 2017)	-	17.1	13.0	26.0	25.8

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

	R@40	MAP	R-Prec	MRR	NDCG
Lucene	23.0	7.1	3.8	7.1	10.5
PRF	29.7	12.2	7.8	12.2	16.0
RM3	30.5	12.6	8.1	12.6	16.5
RL-RNN	33.7	15.0	10.0	15.0	19.1
RL-10-Ensemble	35.2	16.1	10.8	16.1	20.3
RL-RNN Greedy + Aggregator	42.0	21.2	14.9	21.2	25.8
RL-RNN 20 Sampled + Aggregator	42.4	21.5	15.1	21.5	26.1
RL-RNN 20 Beam + Aggregator	42.3	21.4	15.0	21.4	26.0
RL-10-Full	52.1	28.4	20.5	28.4	33.7
RL-10-Bagging	52.5	28.7	20.8	28.7	34.0
RL-10-Sub	53.5	29.7	21.5	29.7	35.0
RL-10-Sub (Pretrained)	54.0	29.8	21.6	29.8	35.2
RL-10-Full (Extra Budget)	54.4	30.3	22.1	30.3	35.8

Table 5: Results on more metrics on the test set of the Jeopardy dataset.

	R@40	MAP	R-Prec	MRR	NDCG
Lucene	12.7	3.1	6.0	15.4	9.1
PRF	13.2	3.4	6.4	16.2	9.7
RM3	12.3	3.1	6.0	15.0	8.9
RL-RNN	15.1	4.1	7.3	18.8	11.2
RL-10-Ensemble	15.8	4.4	7.7	19.7	11.7
RL-RNN Greedy + Aggregator	16.1	4.5	7.8	20.1	12.0
RL-RNN 20 Sampled + Aggregator	16.4	4.6	7.9	20.5	12.2
RL-RNN 20 Beam + Aggregator	16.2	4.5	7.9	20.3	12.1
RL-10-Full	17.4	4.9	8.4	21.9	13.0
RL-10-Bagging	17.6	5.0	8.5	22.1	13.2
RL-10-Sub	18.9	5.5	9.2	23.9	14.2
RL-10-Sub (Pretrained)	19.1	5.4	9.1	24.0	14.2
RL-10-Full (Extra Budget)	19.2	5.6	9.3	24.3	14.4

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

	TREC-CAR	Jeopardy	MSA
RL-RNN	10.8	15.0	4.1
RL-RNN Greedy + Aggregator	10.9	21.2	4.5
RL-RNN 20 Sampled + Aggregator	11.1	21.5	4.6
RL-RNN 20 Beam + Aggregator	11.0	21.4	4.5
RL-10-Sub	12.3	29.7	5.5

Table 7: Multiple reformulators vs. aggregator contribution. 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.

Aggregator Function	MAP	Diff
$s_j = s_j^A s_j^R$ (proposed, Section 3.4)	12.3	-
$z_j = f_{\text{CNN}}(q_0)\|\|f_{\text{BOW}}(a_j)$ (eq. 2)	11.9	-0.4
$s_j^A = \sum_{i=1}^{N} \mathbb{1}_{a_i=a_j}$	11.7	-0.6
$s_j = s_j^A$	11.1	-1.2
$s_j = s_j^R$	10.9	-1.4

Table 8: Comparison of different aggregator functions on TREC-CAR. The reformulators are from RL-10-Sub.

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

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: $\frac{1}{N}\sum_{n=1}^{N}\text{std}\big(\{|q_i^n|\}_{i=1}^{|Q|}\big)$

	SearchQA				TREC-CAR			
	$E_i[e_i] \downarrow$	$E_i[E_{j\neq i}[s_{ij}]] \uparrow$	$E_i[V_{j\neq i}[s_{ij}]] \downarrow$	F1↑	$E_i[e_i] \downarrow$	$E_i[E_{j\neq i}[s_{ij}]] \uparrow$	$E_i[V_{j\neq i}[s_{ij}]] \downarrow$	R@40↑
Q	9.9	52.0	**1.1**	53.3	15.3	50.4	5.9	50.0
A	22.0	50.1	3.9	51.4	**1.3**	**57.0**	0.3	**56.9**
Q+A	**9.0**	50.5	1.2	**53.4**	1.8	56.2	0.3	56.5
Rand.	9.5	**53.8**	**1.1**	**53.4**	1.9	**57.0**	**0.2**	**57.1**

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 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

```
Given: desired cluster size M, and a set of clusters C, each containing a set of items.
sort C by descending order of sizes
C_remaining ← shallow_copy(C)
for c in C do
    remove c from C_remaining
    while c.size < M do
        item ← randomly select an item from c
        move item to the closest cluster in C_remaining
        sort C_remaining by descending order of sizes
    end while
end for
return C
```

Evaluation Metric In order to gain insight into the effect of a partitioning strategy, we first define three evaluation metrics. Let π_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 π_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:

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

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:

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

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.

Method	Query	Reference / Answer from BiDAF (F1)
Jeopardy!	The name of this drink that can be blended or on the rocks means "daisy" in Spanish	
SearchQA	name drink blended rocks means daisy spanish	margarita
AQA	What name drink blended rocks mean daisy spanish?	margarita tequila daisy (0.33)
	What rock drink name means daisy spanish?	**margarita tequila daisy mentioned (0.20)**
	What name drink blended rocks means daisy spanish?	margarita tequila daisy mentioned (0.20)
	What rock drinks name means daisy spanish?	margarita tequila daisy mentioned (0.20)
	What name drink blended rock means daisy spanish?	margarita tequila daisy mentioned (0.20)
AQA-10-Full	What is drink name name drink daisy daisy? me	margarita eater jun (0.33)
	What name is drink spanish?	margarita eater jun (0.33)
	What is daisy blender rock daisy spanish?? daisy spanish?	**cocktail daisy margarita spanish (0.26)**
	rock name name	cocktail daisy margarita spanish (0.25)
	What name drink blended st st st st st ship ship	cocktail daisy margarita spanish (0.26)
AQA-10-Sub	Where is name drink??	margarita (1.0)
	What is drink blended rock?	**margarita (1.0)**
	rock definition name	margarita (1.0)
	What is name drink blended rock daisy spanish 16 daisy spanish?	margarita similarity (0.5)
	Nam Nam Nam Nam Nam Nam Nam drink	tequila (0.0)
Jeopardy!	A graduate of Howard University, she won the Nobel Prize for literature in 1993	
SearchQA	graduate howard university , nobel prize literature 1993	toni morrison
AQA	Nobel university of howard university?	toni morrison american novelist (0.5)
	Nobel university of howard university in 1993?	toni morrison american novelist (0.5)
	Nobel graduate literature in 1993?	**toni morrison american novelist (0.5)**
	Nobel university graduate howard university 1993?	princeton (0.0)
	Nobel university for howard university?	columbia (0.0)
AQA-10-Full	Another university start howard university starther	toni morrison american novelist (0.5)
	university howard car?	**toni morrison american novelist (0.5)**
	What is howard graduate nobel?	toni morrison american novelist (0.5)
	What is howard howard university?	toni morrison american novelist (0.5)
	Where graduated howard university noble literature literature?	american novelist morrison (0.16)
AQA-10-Sub	**Where is howard university??**	**toni morrison (1.0)**
	The nobel university?	toni morrison (1.0)
	What name howard howard university?	toni morrison (1.0)
	This howard?	toni morrison american novelist (0.5)
	1993?	howard cornell universities (0.0)
Jeopardy!	For Bill Gates, it computes to own 2 models, the 959 and the 911, from this manufacturer	
SearchQA	bill gates , computes 2 models , 959 911 , manufacturer	porsche
AQA	**Bill gates iin computes older models?**	**porshe (1.0)**
	Bill gates in compute gates how old are they?	porshe (1.0)
	Bill gates bill gates computes mod?	porshe (1.0)
	Bill gates computes 2 models pics of 959?	porshe (1.0)
	Bill gates in compute gates how old is it?	porshe (1.0)
AQA-10-Full	Another model start bill bette	porshe (1.0)
	What is an bill gates 100 car?	porshe (1.0)
	What is bill bill bill bill gates computes?	porshe (1.0)
	What is manufacturer?	**porshe (1.0)**
	bill bill gats sa computes 2 bill gats?	porshe (1.0)
AQA-10-Sub	Where is bill gates manufacturer?	bill gates (0.0)
	A bill gates?	**bill gates (0.0)**
	The model?	bill gates (0.0)
	What is bill gates model?	sports car (0.0)
	What model bill gates 9 58 model 9 gates?	sports car (0.0)
Jeopardy!	The first written mention of this capital's name was in a 1459 document of Vlad the Impaler	
SearchQA	first written mention capital 's name 1459 document vlad impaler	bucharest
AQA	First film was written by 1459 vlad impaler?	bucharest castle (0.5)
	First film was written by 1459 vlad impalter?	bucharest castle (0.5)
	First film was written by 1459 vlad impal?	bucharest castle (0.5)
	First film was written by 1459 vlad impalot?	bucharest castle (0.5)
	First film was written in 1459?	**bucharest national capital (0.33)**
AQA-10-Full	What is capital vlad impaler?	bucharest (1.0)
	First referred capital vlad impaler impaler?	bucharest (1.0)
	capital	romania 's largest city capital (0.0)
	Another name start capital	romania 's largest city capital (0.0)
	capital capital vlad car capital car capital?	**romania 's largest city capital (0.0)**
AQA-10-Sub	Where is vla capital capital vlad impalers?	bucharest (1.0)
	What capital vlad capital document document impaler?	bucharest (1.0)
	Another capital give capital capital	**bulgaria , hungary , romania (0.0)**
	capital?	bulgaria , hungary , romania (0.0)
	The name capital name?	hungary (0.0)

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).