Dynamic Retriever Selection in RAG Systems: An RL Approach to User-Centric NLP

Parth Sharma
AI/ML services
Genpact
Bengaluru, Karnataka
Parth.Sharma@genpact.com

Aman Kaif Mohammad
AI/ML services
Genpact
Bengaluru, Karnataka
Mohammad.Amankaif @genpact.com

2024 International Conference on Electrical and Computer Engineering Researches (ICECER) | 979-8-3315-3973-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICECER62944.2024.10920369

***Abstract*— This paper investigates a novel use of Reinforcement Learning (RL) to dynamically choose the best retriever in Retrieval-Augmented Generation (RAG) systems with the goal of improving the performance of natural language processing tasks. RAG systems combine retrieval techniques with pre-trained language models to produce responses that are accurate within their context, but a static retriever selection system results in inefficiencies, in a dynamically changing environment where user preferences and the document corpus evolves. We attempt to solve this problem by proposing an RL-based solution to enable dynamic retriever selection based on document context and user feedback. The reinforcement learning agent is able to adjust to user preferences and a changing document corpus by utilizing Q-Learning. The methodology covers issue formulation, agent architecture and training procedures. Our experiments validate our RL-based approach's performance characterized by metrics like user satisfaction and response accuracy. The discussion highlights the strengths and limitations of the Reinforcement Learning approach and suggests future research directions. This experiment underlines the potential of adaptive mechanisms in NLP, showcasing RL's capability to revolutionize RAG applications by creating more responsive and user-tailored NLP systems. (*Abstract*)**

Keywords—GenAI, RAG, LLM, RL

I. Introduction

A. Context of the technology

The RAG (Retrieval-Augmented Generation) [1] framework has emerged as a powerful tool in the field of natural language understanding and generation. It leverages a large corpus of documents to provide contextually rich responses to user queries. However, with the ever-changing nature of document sets and user expectations, the static selection of a retriever can lead to suboptimal performance. This paper addresses this challenge by implementing an RL agent that can adapt to changes and select the most effective retriever for the current context.

The answers generated by a RAG framework depend on the quality of the context passed to the LLM to answer from, this in turn depends on how the original corpus of documents were chunked (split into pieces) and what retriever was used to retrieve those chunks. Different retrievers perform differently with documents of varying domains, hence the need for this dynamic approach.

Reinforcement Learning [2] is a learning framework that enables learning the optimal action in a stationary as well as a stochastic environment if there is an intuitive "reward" feedback for each action it takes from the state it is present in, it is useful in situations where there are many ways or paths to achieve your goal, and each action has a notion of "goodness". Q Learning [2] is a subset of the RL paradigm, which is what we have used here for our retrieval agent as the state space for this problem is very limited and this wouldn't be computation or memory expensive.

II. Methodology

A. Core Components/Paradigms

1) Reinforcement Learning (RL):

Reinforcement Learning (hereby referred to as RL) is a computational approach to understanding and automating goal-directed learning and decision-making. It is distinguished by an agent learning to behave in an environment by performing actions and observing the results of these actions. The core components of a Reinforcement Learning system are:

a) Agent:

The RL agent is the decision-maker that interacts with the environment. It is responsible for selecting actions based on its policy to maximize cumulative reward over time.

b) Environment:

The environment is the external system with which the agent interacts. It is typically modelled as a Markov Decision Process (MDP) and provides feedback to the agent in the form of states and rewards.

c) Policy:

A policy is a strategy employed by the agent that defines the action to be taken in each state. It can be deterministic or stochastic and is often represented as a probability distribution over actions.

d) Reward Signal:

The reward signal is a critical component that defines the goal in an RL problem. It is a scalar feedback signal that tells the agent how well it is doing at a given time. The agent's objective is to maximize the total reward it receives in the long run.

e) Value Function:

The value function estimates the expected cumulative future reward that can be obtained from a state or state-action pair. It helps in evaluating the desirability of states and guiding the agent to make better decisions.

f) Model of the Environment (optional):

Some RL approaches incorporate a model of the environment, which predicts how the environment will respond to certain actions. This can be used for planning by simulating outcomes of actions without taking them.

g) Learning Algorithm:

The learning algorithm updates the agent's policy based on its experiences. It uses the observed transitions (state, action, reward, next state) to make the policy better at achieving the goal. Popular algorithms include Q-Learning, Policy Gradients, and Actor-Critic methods.

The combination of these components enables an RL agent to learn from direct interaction with its environment without human guidance or exhaustive instructions, driving advancements in areas such as robotics, game playing, and autonomous systems.

2) Retrieval Augmented Generation:

Retrieval Augmented Generation (RAG) [1] frameworks are an innovative approach in the field of natural language processing that combines the strengths of pre-trained language models with retrieval-based components to enhance the quality of generated text. The core components of a RAG framework include:

a) Retrieval Component:

The retrieval component is responsible for fetching relevant information from a large corpus or database, which can be a collection of documents, passages, or other structured data. This is typically accomplished using a dense vector space model where queries and documents are embedded into a high-dimensional space to facilitate efficient similarity search.

b) Language Model (Generator):

The language model, often a transformer-based neural network like GPT or T5, is pre-trained on a vast corpus of text and fine-tuned for specific generation tasks. This component serves as the generator, producing text that is coherent and contextually relevant by integrating the retrieved information.

c) Augmentation Mechanism:

This mechanism effectively combines the outputs of the retrieval component with the language model. It ensures that the retrieved information is appropriately incorporated into the generation process, often using attention or fusion strategies to blend the external knowledge with the language model's internal representations.

d) Post-Processing:

Outputs generated by the RAG framework may undergo post-processing to ensure coherence, fluency, and adherence to task-specific requirements. This can involve additional steps like re-ranking generated responses, refining answers, or applying grammar corrections.

In summary, the RAG framework marries the power of large-scale language models with information retrieval to enhance the generation of text, providing a significant advantage in tasks that require external knowledge or factual correctness.

B. Environment and Setup Overview

1) Problem Formulation:

- Define the environment: The RAG application and its document set.
- Establish the state space: There is only one state for now as the information we could gain from questions asked and documents in the corpus is assumed to be conveyed by the human score input itself.
- Define the action space: The set of available retrievers.
- Establish the reward function: User scores for the responses, which reflect user satisfaction, this is later ensembled with other metrics such as time taken for entire pipeline and retrieved context relevance.

2) Retriever Selection Agent:

- Describe the RL agent architecture: Tabular Q Learning.
- During training, a baseline is achieved by using a small set of questions and inferencing each question through the entire pipeline using each available retriever. After this baseline is achieved, adaptive exploration is employed where the average score for the last n iterations decides if the exploration should be increased, and by how much, to re-learn the retrievers' weights.

3) Implementation:

- After being trained and being in a stable state (not currently re-learning), after a question is asked, the retriever with the max score is selected, the RAG application uses the specified retriever to retrieve context, the LLM answers based on the retrieved context and the user rates the answer generated.
- The retrievers available currently are **FullText (BM25)** [3], **Semantically Reranked**, **xKNN**, **HNSW** [4], **Hybrid Search** (Combination of FullText and HNSW), as provided by leading cloud services.

- The documents in the corpus were semantically chunked and stored on a vector database for this experiment.

4) Experimentation:

- The agent reward consists of the human score as well as some other metrics which are context relevancy and answer relevancy.
- Design experiments to test the performance of the RL agent in various scenarios.

C. Implementation

1) Available Methods

This problem is very similar to a multi armed bandit scenario [5] in a stochastic environment. Some methods that are employed to train agents for multi armed bandits are UCB (Upper Confidence Bound) [6] and constant alpha Q learning [2].

The constant alpha helps in continuous learning as the learning step usually decays with episodes but here, we want the agent to learn continuously, so we keep the learning rate a constant, which also means that there is a probability that the agent will never converge and keep changing the weights, which is fine for us.

We set up a Q-Learning agent with one state and 5 actions, the actions being all the available retriever methods. It uses the regular Q-learning update rule to learn:

$$Q(s,a) = Q(s,a) + \alpha\big(R_{t+1} + \gamma * max\, Q'(s',a) - Q(s,a)\big) \quad (1)$$

Where:

$Q(s,a)$	The current state value pair in the table that is being updated
α	Learning rate (hyperparameter)
R_{t+1}	The reward gotten in current time step.
γ	Discount coefficient to give less importance to future rewards.
$\max Q'(s',a)$	The max q table state-pair value, from the state the current action takes us to.

The reward received by the agent ranges from 0 to 1, with 0.5 weightage given to the user's rating and 0.5 to automatically calculated metrics like: Time taken for the chain to complete, Context Relevancy and Answer Relevancy, explained later. RAGAS [7] (A RAG evaluation framework, written in python) is used to calculate Context Relevancy and Answer Relevancy, as described in their work. The formulae Ragas uses for these metrics are mentioned below:

Context Relevancy:

$$\frac{|RelevantSentencesInRetrievedContext|}{|TotalSentencesInContext|} \quad (2)$$

Answer Relevancy:

A Language Model is used to generate questions to the answer that we have received, then the similarities of these questions to the original question are used to formulate this metric.
In Equation (3), v1 and v2 are vectors generated by embedding models where the angle between them is always $\leq 90°$.

$$\text{cosSimilarity}(\text{v1, v2}) = \frac{\text{v1.v2}}{||v1||\,||v2||} \in [0,1] \quad (3)$$

$$\text{Answer Relevancy} = \frac{1}{N}\sum_{i=1}^{N} \text{cosSimilarity}\big(E_{gi}, E_o\big) \quad (4)$$

Where:

E_{gi}	The embedding vector of the generated question i.
E_o	The embedding vector of the original question.
N	The number of generated questions, which is 3 by default for the library used

2) Dataset Creation

A small dataset was created to obtain baseline weights for the retriever methods at hand. The dataset consisted of question-answer pairs from a single domain (Financial Operating Procedure documents with lots of structured data and industry specific terms, in this case) created by domain experts where the answers were known to be in the documents. Ten questions were chosen for this use case as during experimentation that was enough to form a clear baseline and separation in preference weights (for 5 retrieval methods). How this dataset was used to obtain the baseline is outlined in the next section.

3) Training

In the training phase, to obtain the baseline from where continuous learning takes place, a small dataset is used. Each question from the dataset is answered once by all the available actions, then answered a few times (3 here) by choosing the actions the agent deems are most optimal at that time step, and the baseline is formed after the routine mentioned above is over. The epsilon decays at the designated decay rate throughout the baseline training.
The number of weight-altering iterations the agent goes through for the baseline would look like:

$$\mathrm{N_q} * \mathrm{N_a} + \mathrm{N_q} * 3 \quad (4)$$

Where:

$\mathrm{N_q}$	Number of questions
N_a	Number of actions

The equation for exponential epsilon decay is as follows:

$$\varepsilon = \varepsilon * e^{decay\ rate}$$

Where:

- *decay rate* is a chosen hyperparameter < 0.

4) Algorithm for engine baseline training

Definitions

available_retrievers:	The retrievers that are available to the engine
engine:	The RL engine
q table:	The table that the engine uses to maintain the weights/preferences
epsilon:	The epsilon that the engine uses to decide exploration and exploitation of the available retrievers. This is 1 at the beginning of baseline training
decay_rate:	The epsilon decay rate used when not in a retraining phase
reward:	The reward the engine uses to decide whether something was a bad action or good action.

context relevancy, answer relevancy, time taken ∈ [0, 1]
reward ∈ [0, 3]

Flow for agent learning cycle (5)

```
1  user rates the generated answer
2  context relevancy is calculated
3  answer relevancy is calculated
4  time taken is calculated
5  reward = 0.5*(user_rating) +
       0.5*(context_relevancy + answer_relevancy
       + time_taken)/3
6  q table is updated given the reward and retriever
7  epsilon is updated according to the decay_rate
```

Flow for baseline training

```
1  repeat for question in baseline_questions
2      repeat for retriever in available_retrievers
3          engine answers question using retriever
4          agent learning cycle (5) is run

5  repeat for question in baseline_questions
6      repeat for counter in 0 to 3
13         engine chooses retriever based on q table
14         engine answers question using retriever
15         agent learning cycle (5) is run
```

5) Continuous learning

We employed a combination of exploration techniques to ensure that our RL agent could adequately balance the trade-off between exploring new strategies and exploiting known rewarding actions. The exploration methods incorporated into our agent's learning algorithm included:

Epsilon-Greedy Strategy (Modified with RBED (Reward Based Epsilon Decay) [8][2]): An epsilon-greedy policy was used during the initial baseline training phase; the epsilon decay strategy is then changed to an adaptive one later, after the baseline is formed. When the average of the past *n* rewards falls below a certain threshold (a benchmark for the expected quality of answer), the exploration is increased using a slightly modified version of RBED (reward-based epsilon decay).

RBED (Reward Based Epsilon Decay) [8][2]: In RBED, the epsilon is lowered to exploit more on a per episode basis, based on the maximum reward it achieved in the last episode. In our case not only do we want to exploit more given an unexpectedly quicker convergence to the correct retrieval method, but we also want to increase exploration if we get consecutive bad ratings. The range of epsilon which RBED will assign has to be tweaked though, since given a specific retrain convergence period, after a set of bad rewards, a minimum epsilon and learning rate exist that guarantees convergence to the best retrieval method (given there exists a best method).

The changes made to accommodate for the above-mentioned points:

For a retrain period after bad responses –

- An epsilon starting point of 0.7, retrain decay of -0.04, with a retrain period of 50 iterations seemed to work well enough during experimentation and always converge within 2 retrain iterations, mostly within 1 (for our document corpus and number of available retrievers). For more than 2 iterations, we can say that no retrieval method is currently giving satisfactory answers.
- The given epsilon starting point of 0.7 could be brought down by using RBED mapped to an epsilon range of [x, 0.7], more experimentation would be required for this.

a) Algorithm for engine evaluation

Definitions

question:	The question the user has asked
avg_window:	The window over which the average of past rewards is taken, to gauge user satisfaction.
avg_reward:	The average of rewards calculated over the last *avg_window* rewards.
retrain_threshold:	If the average of the rewards over the avg_window goes below this threshold the engine enters the retraining phase.
retrain flow:	The algorithm for the retrain phase as explained below.
epsilon:	The epsilon that the engine uses to decide exploration and exploitation of the available retrievers. This continues from the value of epsilon after baseline training.
reward:	The reward the engine uses to decide whether something was a bad action or good action.
decay_rate:	The epsilon decay rate used when not in a retraining phase.

context relevancy, answer relevancy, time taken ∈ [0, 1]
reward ∈ [0, 3]

Flow for evaluation

```
1   repeat while engineUp:
2       the avg_reward is calculated
3       if avg_reward < retrain_threshold:
4           retrain = true
5       else:
6           retrain = false
7       if not retrain:
8           engine chooses retriever given epsilon
9           engine answers question using chosen retriever
10          user rates the generated answer
11          context relevancy is calculated
12          answer relevancy is calculated
13          time taken is calculated
            reward = 0.5*(user_rating) +
                    0.5*(context_relevancy +
                    answer_relevancy + time_taken)/3
15          q table is updated given the reward and
                    chosen retriever
16          epsilon is updated according to the decay_rate
17      else:
18          enter retrain flow
```

b) Algorithm for retraining

Definitions

question:	The question the user has asked.
retrain_i	The counter that keeps track of iterations in the retraining phase
retrain_iterations	The max number of iterations the engine must go through before entering the evaluation phase again.
retrain_decay_rate:	The decay rate used specifically for the retraining phase of the engine.

context relevancy, answer relevancy, time taken ∈ [0, 1]
reward ∈ [0, 3]

Flow for retraining

```
1   epsilon is bumped up from current value to 0.7
2   retrain_i = 0
3   repeat while retrain_i < retrain_ iterations:
4       engine chooses retriever given epsilon
5       engine answers question using chosen retriever
6       user rates the generated answer
7       context relevancy is calculated
8       answer relevancy is calculated
9       time taken is calculated
10      reward = 0.5*(user_rating) +
                0.5*(context_relevancy +
                answer_relevancy + time_taken)/3
11      q table is updated given the reward and
                chosen retriever
12      epsilon is updated according to the
                retrain_decay_rate
```

III. EXPERIMENTS AND RESULTS

A. Experiment Details

A typical learning cycle consists of the following events:

- The curated dataset is used to complete baseline training
- The user asks the engine a question.
- The question is then answered using the current best choice of retriever according to the engine, given the weights, state of the document corpus and question asked.
- The user rates the answer on a scale of 1 to 5.
- The engine then readjusts its preferences or weights of the retrievers based on how far off the received reward was from what the engine expected, using the Q-Learning update rule.

The experiment starts with the agent having all retrievers' weights as 0. The agent then goes through baseline training, after baseline training (80 iteration according to equation (4)), at 90 iterations the document corpus composition is changed by adding Financial Operating Procedures and then questions pertaining to those documents were asked to measure the retrievers' performance and user satisfaction with regards to the agent's ability to adapt.

To measure performance, the metrics we will be monitoring are the average user rating, cost incurred, and time taken.

Cost incurred	Some search methods return more chunks than required and hence cost more, due to increased input-token cost of the LLM.
Time taken	This is the time taken for the retriever to return the fetched chunks.

Definitions

max_epsilon:	Starting value of epsilon.
min_epsilon:	Minimum exploration maintained throughout.
decay_rate:	The decay rate used in baseline as well as evaluation phases
reward_threshold:	Retraining starts after the average of the last n rewards falls below this category.
retrain_iterations:	Minimum number of iterations before which the engine can enter retraining again.

Configuration:

max_epsilon	1.0
min_epsilon	0.05
decay_rate	-0.02
retrain_decay_rate	-0.04
learning_rate	0.3
retrain_iterations	50
reward_threshold	0.73

Figure 1 shows the reward and exploration-epsilon on the y-axis versus number of iterations on the x-axis. We can see the average reward increasing with iterations and the epsilon reaching stable lower values, indicating better user satisfaction and convergence to a method/retriever, respectively.

As seen in Figure 2, the agent manages to relearn the best retriever within ~150 iterations. The initial variance in weights you see is due to all the actions/retrievers having the same weight to start with before baseline training. The baseline training finishes at 80 iterations as calculated by equation (4), after which you can see the agent change its retriever preferences from Semantic to FullText given the change in questions asked and context documents. FullText search was observed to perform better with tasks that include niche terms or a lot of abbreviations, and the agent learns this during the retraining session.

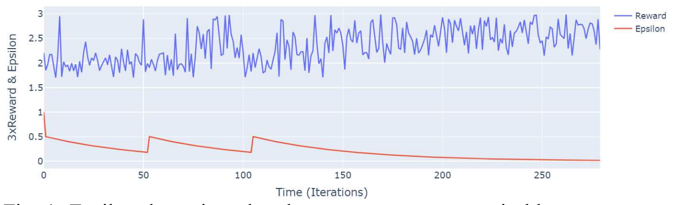

Fig. 1. Epsilon decay in red and average user response in blue, over agent iterations.

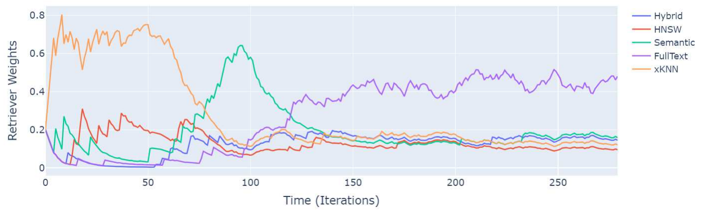

Fig. 2. Retriever preferences over iterations

IV. Conclusion

The engine was able to relearn retriever preferences in a dynamic environment with a changing document corpus and multiple users, while providing 21% higher average user reward and 16% lesser cost incurred (input-token cost) as compared to using a single retriever (HNSW in this experiment), over 300 question-answer iterations.
The iterations the agent takes to relearn the weights can be lowered further by experimenting with RBED and retrain decay rate parameters.

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", 12 Apr 2021, arXiv:2005.11401. Available: https://arxiv.org/abs/2005.11401

[2] R. S. Sutton, & A. G. Barto, (2018). "Reinforcement learning: An introduction." MIT press.

[3] S. E. Robertson, Z. Hugo, (2009). "The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval." 3. 333-389. 10.1561/1500000019.

[4] Y. A. Malkov, D. A. Yashunin. "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs", 2016, arXiv:1603.09320.

[5] P. Auer, N. Cesa-Bianchi, Y. Freund and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," Proceedings of IEEE 36th Annual Foundations of Computer Science, Milwaukee, WI, USA, 1995, pp. 322-331, doi: 10.1109/SFCS.1995.492488.

[6] P. Auer. "Using confidence bounds for exploitation-exploration trade-offs." Journal of Machine Learning Research, 3(Nov):397–422, 2002.

[7] S. Es, J. James, L. Espinosa-Anke, S. Schockaert. "RAGAS: Automated Evaluation of Retrieval Augmented Generation", 2023, arXiv:2309.15217.

[8] A. Maroti, "RBED: Reward Based Epsilon Decay**"**, 30 Oct 2019, arXiv:1910.13701. Available: https://arxiv.org/abs/1910.13701