Ideal Query Expansion using Reinforcement Learning

Dissertation submitted in partial fulfilment of the requirements
for the award of the degree of

Master of Technology in **Computer Science**

by

Madhuchchhanda Das

(Roll No. CS2309)

Under the Supervision of

Dr. Debapriyo Majumdar

INDIAN STATISTICAL INSTITUTE KOLKATA

Kolkata, India

June, 2025

Certificate

This is to certify that the dissertation titled **"Ideal Query Expansion using Reinforcement Learning"**, submitted by **Madhuchchhanda Das** to the Indian Statistical Institute, Kolkata, in partial fulfillment of the requirements for the degree of **Master of Technology in Computer Science**, is an original and genuine piece of research conducted under our supervision and guidance. The dissertation meets the academic standards prescribed for the award of the said degree.

To the best of our knowledge, the results presented in this dissertation have not been submitted, either in part or in full, to any other university or institute for the award of any degree or diploma.

.....

Dr. Debapriyo Majumdar
CVPR Unit
Indian Statistical Institute Kolkata-700108 India

Declaration

I hereby declare that the content of this submission is the result of my own work and original ideas. Wherever the ideas or words of others have been used, appropriate citation and referencing have been provided.

I affirm that I have followed all principles of academic honesty and integrity, and have not misrepresented, fabricated, or falsified any data, facts, sources, or ideas in this dissertation.

I understand that any breach of these principles may lead to disciplinary action by the Institute and could also result in legal consequences from the original source if proper citation or necessary permissions have not been obtained.

Madhuchchhanda Das

.....

Madhuchchhanda Das
Roll No.: CS2309
Date:
Place: ISI Kolkata

Acknowledgements

I would like to take this opportunity to express my sincere gratitude to all those who supported and guided me throughout the course of my dissertation. I am especially thankful to my supervisor, Dr. Debapriyo Majumdar, for his invaluable guidance, encouragement, and constant support at every stage of this work. I am also deeply grateful to the Almighty for his grace and blessings, which gave me the strength and perseverance to successfully complete this dissertation.

Madhuchchhanda Das

Abstract

Information retrieval (IR) systems often struggle with short, ambiguous, or underspecified queries, leading to suboptimal document retrieval. Traditional query reformulation methods, such as those based on the Rocchio algorithm, rely on heuristic term selection and relevance feedback but typically apply fixed or manually tuned weights to expanded terms. This limits their adaptability and generalization across diverse query-document contexts.

In this thesis, we propose a novel reinforcement learning (RL)-based framework to dynamically optimize term weighting in reformulated queries. We model the problem as a Markov Decision Process (MDP), where each state represents a query as a vector of term weights. An RL agent learns a policy to assign optimal weights to terms by maximizing a reward signal based on retrieval performance—specifically precision-based metrics like Mean Average Precision (MAP).

Our method is evaluated on benchmark datasets, where it outperforms traditional static approaches by learning query-specific term weighting strategies that generalize well to unseen queries. The approach draws inspiration from earlier optimization techniques such as Dynamic Feedback Optimization in TREC but differs fundamentally by employing a data-driven learning mechanism rather than rule-based reweighting. The results demonstrate that reinforcement learning offers a principled and flexible solution for effective query reformulation in modern IR systems.

Contents

Chapter 1

Introduction

1.1 Introduction

In the field of Information Retrieval (IR), the ability to accurately retrieve documents that closely align with a user's query is crucial for enhancing user satisfaction and system performance. A central challenge in this domain is that users often express their information needs through short, ambiguous, or imprecise queries. As a result, the effectiveness of retrieval systems can be significantly improved by refining these queries a process known as query reformulation.

One widely studied approach to query reformulation is relevance feedback, where information from previously retrieved relevant documents is used to automatically update or expand the original query. Traditionally, relevance feedback has focused on selecting additional terms or adjusting query term weights based on heuristic or statistical methods. However, these methods often lack adaptability and fail to generalize across diverse retrieval tasks.

This thesis explores the use of Reinforcement Learning (RL) to overcome these limitations. Specifically, it models the task of assigning optimal weights to terms

in a reformulated query using an RL framework. By treating query reformulation as a sequential decision-making process, RL agents can dynamically learn which term weight configurations yield better retrieval performance, measured using Mean Average Precision (MAP).

The approach begins by identifying informative terms from documents marked as relevant to a given query. These terms are initially weighted equally. Through interaction with the retrieval system, the RL agent receives feedback in the form of rewards, based on MAP scores, and adjusts term weights to improve performance over time. This adaptive process ensures that the reformulated query better aligns with the user's actual information need.

By integrating reinforcement learning with IR principles, this thesis introduces a novel and adaptive solution to query reformulation. The framework accounts for both term selection and term weighting, enhancing the system's ability to deliver relevant search results in dynamic and diverse environments.

1.2 Motivation

Traditional search systems frequently fall short when queries are under-specified—a common scenario in real-world usage where users may lack the precise keywords or submit queries too brief to capture their true intent. This gap not only degrades retrieval performance but also leaves users in the dark about why certain documents are returned. To address these issues, we need adaptive, transparent reformulation strategies that both improve ranking and explain their behavior.

Heuristic relevance-feedback methods, such as Rocchio's algorithm, adjust queries by reweighting terms according to their frequency in the top-ranked documents. Although these approaches can boost effectiveness, they remain inherently static:

the same term adjustments apply regardless of domain, dataset, or individual user preferences, and they offer little insight into why certain terms were promoted or demoted.

In contrast, Reinforcement Learning (RL) provides a principled framework for both adaptation and explainability. By modeling query construction as a Markov Decision Process (MDP), the system learns which term additions or deletions maximize a retrieval-quality reward (for instance, Precision@K). Each action—adding, removing, or reweighting a specific term—carries an explicit weight update that can be traced back to its impact on the reward signal. As a result, every reformulation step is inherently interpretable: we can inspect the learned policy to see, for a given query context, which terms the agent deemed critical and why.

Implementing this with Deep Q-Networks (DQN) allows our system to continuously refine term weights through interaction with the retrieval environment. Over successive episodes, the agent not only adapts to the peculiarities of different collections and user behaviors but also accumulates a transparent record of term-selection decisions. This record yields twofold benefits: (1) enhanced performance as the policy converges on the most informative query terms, and (2) on-demand explainability, since each query refinement can be decomposed into a sequence of human-readable actions (e.g., "add 'neural-network' because it boosted Precision@10 by 8%," or "remove 'algorithm' as it led to more non-relevant hits").

By fusing Information Retrieval with Reinforcement Learning, this thesis demonstrates a dynamic, performance-driven query construction framework that not only tailors to specific information needs but also provides clear, actionable insights into how and why each term was selected or discarded. The result is a more intelligent, flexible, and user-aware retrieval system—one that both performs better and illuminates its own decision process.

1.3 Problem Statement

Given a collection of documents labeled as relevant or non-relevant with respect to a set of queries, the goal is to automatically construct an optimal query that retrieves relevant documents at higher ranks. Specifically, we consider two closely related problem formulations:

1. **Weighted Query Formulation:** Find a compact weighted query consisting of a limited number of expansion terms, where each term is assigned a weight indicating its relative importance.
2. **Unweighted Query Formulation:** Find a compact unweighted query consisting of a limited number of expansion terms, where each selected term contributes equally to the query.

In both formulations, the constructed query is evaluated based on its ability to retrieve relevant documents early in the ranking. The primary objective is to maximize a ranking-based evaluation metric, namely the Mean Average Precision (MAP), which emphasizes the presence of relevant documents in the top ranks of the retrieved list.

The challenge lies in selecting a small, informative subset of terms from the vocabulary that, when used in the query, leads to high MAP scores—thus ensuring both effectiveness and interpretability of the final query.

1.4 Our Contribution

This research presents several key contributions:

1. **RL-based Query Reformulation Framework:** An end-to-end system that uses a Deep Q-Network (DQN) to learn and assign optimal term weights based on Mean Average Precision (MAP) feedback.

2. **Markov Decision Process Formulation:** The query reformulation task is formally modeled as a Markov Decision Process (MDP), where states represent term weight configurations and actions correspond to weight adjustments.
   For the weighted query reformulation approach, we investigated two strategies: a single-action method, where the agent modifies the weight of one term per step, and a double-action method, where two term weights are adjusted simultaneously in each step.

   In the case of the unweighted query approach, we began with an initial set of 50 candidate terms extracted from relevant documents and aimed to iteratively select a subset of 10 terms that would form a high-quality weighted query through reinforcement learning.

3. **Lucene Integration:** The retrieval engine is implemented using PyLucene, facilitating realistic and scalable evaluation of reformulated queries within a widely used search platform.

4. **Evaluation of Action Strategies:** The study investigates the impact of different strategies and learning rates on convergence speed and retrieval performance.

5. **Adaptability:** The framework is designed to be extensible to various datasets, evaluation metrics (e.g., NDCG, Precision@K), and reinforcement learning algorithms beyond DQN.

1.5 Structure of This Thesis

The remainder of this thesis is organized as follows:

- **Chapter 2: Literature Review** – Reviews foundational and contemporary work in query reformulation, relevance feedback, and applications of reinforcement learning in information retrieval.
- **Chapter 3: Our Approach** – Describes the dataset, preprocessing techniques, term extraction and selection strategy, formulation of the MDP, and the DQN architecture.
- **Chapter 4: Experimental Setup and Results** – Provides implementation details, training parameters, and evaluation of different action strategies and learning rate combinations.
- **Chapter 5: Discussion** – Analyzes experimental outcomes, explores reasons behind observed trends, and discusses strengths, weaknesses, and limitations of the proposed approach.
- **Chapter 6: Conclusion and Future Work** – Summarizes the key contributions, presents conclusions drawn from the results, and outlines future research directions.

Together, these chapters provide a systematic exploration of how reinforcement learning can be effectively applied to query reformulation, improving retrieval quality and adaptability in dynamic information retrieval environments.

Chapter 2

Related Work

This chapter presents a survey of existing literature relevant to the optimization of information retrieval (IR) systems, particularly focused on query reformulation, term weighting in IR. The chapter is divided into sections that explore traditional relevance feedback methods, term weighting strategies, and learning-based approaches, culminating in a discussion on how the proposed work aligns and diverges from previous studies. Traditional approaches to *query expansion* and *term weighting* in information retrieval (IR) have extensively leveraged linguistic heuristics, co-occurrence statistics, and probabilistic modeling. A comprehensive survey by Carpineto and Romano Carpineto and Romano (2012) outlines various automatic query expansion techniques, while Zhai and Lafferty **?** introduced a probabilistic language modeling framework with smoothing for improved term weighting.

With the growing complexity of IR models, there is a corresponding need for interpretable and adaptive retrieval systems. Ribeiro et al. Ribeiro et al. (2016) proposed LIME, a model-agnostic approach for local explanation of predictions, which sparked further interest in transparent decision-making. Singh and Anand **?** extended this

direction to IR, emphasizing local explanations and stability in query-result behavior.

Recent work applies *reinforcement learning* to query reformulation. Nogueira and Cho Nogueira and Cho (2017) first modeled it as a sequential decision-making process, optimizing query quality through action-based term selection. Zeng et al. **?** improved on this by adding reward feedback for selecting relevant terms. Inspired by this line of work, our method treats term selection as a binary decision using RL, introducing penalties for long queries to maintain both effectiveness and interpretability. Recently Jiang et al. (2025) have used LLMs trained by purely reinforcement learning for reformulating the query to improve the performance.

2.1 Relevance Feedback

Relevance feedback has been one of the most studied approaches in IR to improve the quality of search results. A well-known and widely used method is the Rocchio algorithm, which modifies the initial query vector by incorporating terms from known relevant and non-relevant documents. The updated query vector is computed as follows:

$$\vec{Q}_{\text{new}} = \alpha\vec{Q}_{\text{original}} + \beta\frac{1}{|D_r|}\sum_{\vec{d}_i \in D_r}\vec{d}_i - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j \tag{2.1}$$

where D_r and D_{nr} are the sets of relevant and non-relevant documents respectively, and α, β, and γ are weighting parameters. While this approach enhances retrieval effectiveness, it treats all selected terms equally after expansion and does not dynamically optimize individual term weights.

2.2 Dynamic Feedback Optimization

A significant advancement over the standard Rocchio feedback method is the **Dynamic Feedback Optimization** approach introduced in TREC routing tasks by Buckley and Salton (1995). In this work, the goal is to optimize the weights of query terms based on their retrieval performance over a learning set of documents. The method follows a multi-pass strategy to iteratively improve the query formulation:

- In **Step 1**, a feedback query is constructed using Rocchio's method, where the top frequently occurring terms in the relevant documents are added to the query.
- In **Step 2** (optional), terms are evaluated to determine whether their inclusion enhances performance.
- **Step 3** involves performing three reweighting passes, where each term's weight is increased by 50%, 25%, and 12.5% respectively if the change leads to improved recall-precision on the learning set.
- Finally, the optimized query is evaluated on a test set (D3) using average recall-precision.

Experiments showed that this optimization technique resulted in up to a 10% improvement in retrieval performance compared to non-optimized feedback. Furthermore, the study highlighted that performance gains largely stemmed from intelligent reweighting of terms rather than the mere addition of new terms.

2.3 Explainibility in Information Retrieval

As information retrieval systems grow more complex, the need for explainibility understanding how and why a system returns certain results—has become crucial. Earlier IR models used explicit, linguistically motivated features like word counts and part-of-speech tags, which were naturally interpretable. In general, Ribeiro et al. (2016) introduced LIME, a model-agnostic approach for explaining individual predictions of complex classifiers, enhancing transparency in machine learning models. Explainibility of retrieval methods is discussed by Saha et al. (2022) in their survey paper. Recent studies by Verma and Ganguly (2019) have explored local explanation models to interpret IR systems, emphasizing the importance of sample diversity and model stability for generating reliable explanations. Our work aligns with this direction by applying reinforcement learning for term weighting, enabling both effective retrieval and interpretable query reformulation.

Our work aligns with this direction by using term weighting by reinforcement learning, which not only improves retrieval effectiveness but also maintains a degree of transparency through interpretable query term selections and explicit penalties for query length.

2.4 Relation to Our Work

The proposed work draws strong conceptual parallels with the Dynamic Feedback Optimization framework. Both approaches share the objective of constructing an optimally weighted query to retrieve the most relevant documents. However, there are key methodological differences:

- While the Dynamic Feedback Optimization uses a heuristic multi-pass adjustment strategy based on evaluation metrics on a training set, our work formulates the optimization as a Markov Decision Process (MDP), where each query is treated as a state consisting of individual term weights.
- We employ a Reinforcement Learning agent, specifically using techniques such as Deep Q-Learning, to learn an optimal policy that assigns weights to terms based on their impact on retrieval effectiveness.
- Unlike the previous work, which statically evaluates each term weight in isolation, our model considers the dynamic interaction between terms and learns a joint weighting scheme that generalizes across different queries and documents.

By learning from interactions and feedback, our method avoids overfitting to a specific learning set and shows improved generalization capabilities when applied to unseen queries. Moreover, it supports real-time adaptation, whereas the previous approach was primarily designed for offline routing tasks.

2.5 Summary

In summary, traditional relevance feedback techniques laid the groundwork for query reformulation, with Dynamic Feedback Optimization introducing systematic weight tuning for improved retrieval. Our approach builds upon this idea using Reinforcement Learning to learn optimal term weights in a data-driven manner, making it more flexible and potentially more effective in varied and dynamic retrieval settings.

Chapter 3

Our Approach

3.1 Introduction

This chapter elaborates on the methodology followed in this work to address the problem of query reformulation in information retrieval systems. The main objective is to adjust the term weights in a query such that the reformulated query retrieves documents that are most relevant as per the ground truth data.
Two models are proposed:

- A term weighting model that uses reinforcement learning to iteratively adjust term weights to maximize retrieval performance (using metrics MAP).
- A term selection model that learns to choose the most effective subset of 10 terms from the top 50 candidates that best retrieve relevant documents.

This optimization problem is approached using Reinforcement Learning (RL), particularly using a Deep Q-Network (DQN) based function approximation technique.

The core idea is to define a Markov Decision Process (MDP) where each state represents a specific configuration of term weights, and actions correspond to changing the weights. The reward function is based on retrieval effectiveness, measured using MAP (Mean Average Precision).

3.2 Term Extraction and Choosing Suitable Weights

The goal is to construct a reformulated query using top terms extracted from the relevant documents, and assign optimal weights to these terms such that the retrieval results closely match the relevance judgments.

3.2.1 Markov Decision Process (MDP) Formulation

To apply Reinforcement Learning, we model the problem as an MDP with the following components:

State (S)

A state is a vector representing the weights assigned to the 10 selected terms. Each weight is constrained to take one of five values: {0.1, 0.25, 0.50, 0.75, 1.0}. Thus, each term can take 5 values, and the total number of possible states is $5^{10} = 9,765,625$.

Action (A)

An action corresponds to adjusting one or more weights in the current state. Specifically, a weight can be increased or decreased by 0.25. Given the limited number of terms (10) and discrete weight values, we consider 20 possible actions, such as increasing w_1 by 0.25, decreasing w_3 by 0.25, and so on.

Transition Function (T)

Given a state and an action, the next state is obtained by applying the weight change as specified in the action. If a weight is already at its boundary (e.g., 1.0) and the action attempts to increase it, the value is clamped to remain within the defined range.

Reward Function (R)

The reward is defined as the change in MAP (Mean Average Precision) score between the new state and the previous state:

$$R(s, a) = MAP(s') - MAP(s)$$

Here, s is the current state, a is the action, and s' is the resulting state after applying a.

3.2.2 Reinforcement Learning Framework

The objective is to learn an optimal policy π that suggests which action to take from any given state to maximize the expected cumulative reward. Since the state space is large (5^{10}), a tabular Q-learning approach is not feasible. Therefore, we use a function approximator based on a neural network, specifically a Deep Q-Network (DQN).

3.2.3 DQN Architecture

- **Input:** The concatenation of the state and action vectors. That is, a 20-dimensional input.
- **Hidden Layer:** One hidden layer with 64 neurons and ReLU activation.
- **Output:** A single scalar value representing $Q(s, a)$.

The DQN is trained using experience replay. A buffer stores past (state, action, reward, next state) tuples, and minibatches are sampled randomly for training. The target for each sample is:

$$Q_{\text{target}} = r + \gamma \max_{a'} Q(s', a')$$

where r is the reward, γ is the discount factor (set to 0.9), and $Q(s', a')$ is the estimated Q-value of the next state.

3.2.4 Training Strategy

- **Initialization:** For each query, the initial state is set with all weights at 0.5.

- **Epsilon-Greedy Policy:** During training, an ϵ-greedy policy is used to balance exploration and exploitation. Initially, $\epsilon = 1$, and it decays over time to a minimum of 0.05.
- **Episodes:** The RL agent is trained over multiple episodes (e.g., 5 or 10 episodes per query).
- **Steps per Episode:** Each episode consists of multiple steps (e.g., 5000). At each step, the agent:
  - Evaluates current AP.
  - Chooses an action using ϵ-greedy policy.
  - Applies the action to get the new state.
  - Calculates new AP and reward.
  - Stores the transition in the replay buffer.
  - Trains the DQN using minibatches if the buffer has sufficient samples.

3.2.5 Evaluation Metrics

To measure the effectiveness of reformulated queries, we use the following metric:

- **Average Precision (AP):** Measures the average precision across all relevant documents for a query.

This metric help track whether the RL agent is improving the quality of retrieved documents.

3.3 Query Initialization by Binary Weight

The Q-learning agent learns to optimize a binary query representation by exploring the space of term inclusion vectors. The goal is to maximize retrieval performance (e.g., MAP) through iterative updates based on environment feedback.

3.3.1 State Representation

The state at any time step t is represented by a binary vector:

$$s_t = [b_1, b_2, \ldots, b_N] \in \{0, 1\}^N$$

where $b_i = 1$ indicates inclusion of the i^{th} candidate term in the query, and N is the total number of candidate terms derived from relevant documents.

3.3.2 Action Representation

An action a_t corresponds to flipping a single bit in the state vector:

$$a_t = [0, \ldots, 1, \ldots, 0]$$

where the 1 is at the i^{th} position, indicating a toggle of the i^{th} term (either inclusion or exclusion). The next state s_{t+1} is computed as:

$$s_{t+1} = s_t \oplus a_t$$

where $\oplus$ denotes the bitwise XOR operation.

3.3.3 Reward Function and Penalty

The reward is based on the retrieval effectiveness of the query corresponding to the current state. The system uses Mean Average Precision (MAP) as the performance signal. To discourage overly verbose queries, a penalty is applied when the number of selected terms exceeds a predefined threshold (e.g., 10). Let k be the number of selected terms in the current state:

$$k = \sum_{i=1}^{N} b_i$$

The raw reward is the AP (average precision) score of the query:

$$R_{\text{AP}}(s_t)$$

The penalty function $P(k)$ is defined as:

$$P(k) = \begin{cases} 0 & \text{if } k \leq k_{\max} \\ \lambda \cdot \exp(k - k_{\max}) & \text{if } k > k_{\max} \end{cases}$$

where λ is a penalty coefficient, and $k_{\max}$ is the maximum allowed number of terms without penalty.

The final reward is given by:

$$r_t = R_{\text{AP}}(s_t) - P(k)$$

To encourage the agent to make effective transitions, the reward used for learning is computed as the difference between the MAP of the current and next queries:

$$\Delta r_t = r_{t+1} - r_t$$

3.3.4 Q-Function Update

The Q-function $Q(s, a)$ approximates the expected future cumulative reward from taking action a in state s. It is updated using the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

where:

- α is the learning rate,
- γ is the discount factor for future rewards,
- a' represents all possible future actions from s_{t+1}.

In this implementation, the Q-function is approximated using a neural network regressor, which is trained to minimize the squared error between the target Q-values and predicted Q-values:

$$\mathcal{L} = \left(y_t - \hat{Q}(s_t, a_t) \right)^2$$

where:

$$y_t = r_t + \gamma \cdot \max_{a'} \hat{Q}(s_{t+1}, a')$$

3.3.5 Exploration Strategy

An ϵ-greedy policy is used to balance exploration and exploitation. The exploration rate ϵ decays linearly over time from ϵ_{start} to ϵ_{end}:

$$\epsilon_t = \max\left(\epsilon_{\text{end}}, \epsilon_{\text{start}} - \frac{t}{T}(\epsilon_{\text{start}} - \epsilon_{\text{end}})\right)$$

where T is the total number of decay steps.

3.3.6 Training steps

The training process follows a reinforcement learning loop using Q-learning with function approximation. The steps involved are as follows:

1. **Initialization:**
   - Initialize the Q-network using a neural network with random weights.
   - Initialize the state vector $\mathbf{s}_0$ as a binary vector of zeros representing no query terms selected.
   - Set hyperparameters such as ϵ (exploration rate), γ (discount factor), and the replay buffer.
2. **For each training episode:**
   (a) Start with the initial state $\mathbf{s}_0$.
   (b) Repeat for a fixed number of steps:
      i. **Epsilon decay:** Linearly reduce the exploration rate ϵ_t.
      ii. **Action generation:** Enumerate all possible single-bit flip actions $\mathbf{a}_t$ from the current state $\mathbf{s}_t$.

iii. **Action selection:** With probability ϵ_t, select a random action (exploration). Otherwise, select the action that maximizes the Q-value:

$$\mathbf{a}_t = \arg\max_{\mathbf{a}} Q(\mathbf{s}_t, \mathbf{a})$$

iv. **Apply action:** Generate next state $\mathbf{s}_{t+1}$ by flipping the selected bit(s) in $\mathbf{s}_t$.

v. **Create query:** Formulate a Lucene Boolean query using only the terms whose corresponding bits are set to 1 in $\mathbf{s}_{t+1}$.

vi. **Evaluate query:** Compute MAP score and apply a penalty if more than 10 terms are selected:

$$\text{Reward}_t = \text{AP}_{t+1} - \lambda \cdot \exp(\max(0, n_t - 10))$$

where n_t is the number of selected terms and λ is a penalty factor.

vii. **Store transition:** Append the transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ to the replay buffer.

viii. **Update Q-function:** If enough transitions are stored:

- Sample a batch of transitions from the buffer.
- For each transition, compute the target:

$$y = r_t + \gamma \cdot \max_{\mathbf{a}'} Q(\mathbf{s}_{t+1}, \mathbf{a}')$$

- Perform a gradient update to minimize the difference between predicted and target Q-values.

ix. **Update state:** Set $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.

Chapter 4

Experimental Setup and Results

4.1 Dataset Overview

The dataset used in this work is TREC Robust04 which consists of:

- **Query File:** Contains 250 different queries.
- **Document Collection:** A corpus of approximately 500,000 documents.
- **Judgment File:** For each query ID, this file indicates which documents are relevant and which are not. This is referred to as the relevance judgment file.

4.2 Experimental Setup

In this study, all experiments were conducted on the TREC Robust-04 dataset, a widely used benchmark collection for evaluating information retrieval systems. To enable efficient retrieval, we constructed an index of the dataset using `PyLucene`, a Python wrapper for the Apache Lucene search library.

During indexing, we applied standard text preprocessing techniques using Lucene's `EnglishAnalyzer`, which includes stopword removal and stemming to normalize terms and reduce vocabulary size. NLTK was also used to remove stopwords that were not eliminated by PyLucene's EnglishAnalyzer. This preprocessing ensures consistency in term matching during retrieval.

For evaluating retrieval performance, we utilized the official qrels (query relevance judgments) file provided with the dataset, which lists the relevant documents for each query ID. We measured effectiveness using the Mean Average Precision (MAP) metric, calculated over the top 1000 retrieved documents for each query. This metric provides a robust indication of how well the system ranks relevant documents towards the top of the result list.

TABLE 4.1: Experimental Configuration

Parameter	**Value**
Dataset	TREC Robust04
Retrieval Model	Lucene BM25
RL Algorithm	Q-Learning with MLP
Batch Size	16
Replay Buffer Size	1000
Discount Factor (γ)	0.9
Hidden Layer Size	64

4.3 Weighted Query Reformulation

4.3.1 Query Expansion and Initial State

For a given query, we extracted relevant documents using the QRELs and processed them using Lucene's `EnglishAnalyzer`. From relevant documents, we selected the top 10 terms excluding the stop-words as expanded query terms. These terms formed the feature space for optimization, with each term associated with a floating-point weight among $\{0.0, 0.25, 0.5, 0.75, 1.0\}$.

Initially, all term weights were set to 0.5. A weighted Boolean query was then constructed using Lucene's `BoostQuery`, where each term's influence was proportional to its assigned weight.

4.3.2 Our Reinforcement Learning Framework

State Representation: Here the state is the weight of the 10 terms chosen in previous step, i.e. $(w_1, \ldots, w_{10})$, where each w_i belonds to $\{0.0, 0.25, 0.5, 0.75, 1.0\}$.

Action Representation: In this setting, the agent is allowed to modify the weight of exactly one query term per step by a fixed increment or decrement (`±step_size`, set to 0.25). Since we take 10 terms in our query, the action space comprises 20 discrete actions.

If we take the action as incrementing (similarly decrementing) the i-th term by 0.25, the action can be expressed by the 10-dimensional vector:

$$\underbrace{(0, \ldots, 0,}_{i-1 \text{ zeros}}\ 0.25,\ \underbrace{0, \ldots, 0)}_{10-i \text{ zeros}}$$

Q-network: The reinforcement learning component was implemented using a Q-learning paradigm with a function approximator. It is modeled using an MLPRegressor (from `sklearn`) with one hidden layer of 64 neurons. For updating this network, we have explored different learning rates and chosen the best performing learning rate 0.001. The input to the Q-network is a concatenation of the current state (term weights) S and the action vector (change in weights) A and the network outputs a scalar Q-value $Q(S, A)$ estimating the expected reward from state S if we take action A.

Training Setup: The training was performed over 1 episode of 25,000 steps. At each step, the agent:

- Evaluated the current query using Lucene and computed the AP (average precision).
- Chose an action (weight adjustment) using the ε-greedy strategy, where ε decayed from 1.0 to 0.01 over the first 80% of total training steps.
- Updated the query weights and re-evaluated the new query.
- Calculated the reward as the difference in AP (average precision) scores between updated weighted query and previous weighted query.
- Stored the experience, that is the transition tuple (state, action, reward, next state) in the replay buffer to update the Q-function periodically.

Reward Signal: The reward at each step was computed as the change in AP between the current and next query states. This formulation directly incentivizes improvements in retrieval effectiveness, guiding the agent toward optimal term weighting configurations.

TABLE 4.2: Performance of Weighted RL-based Query Reformulation

Steps	MAP (Base Line Query)	MAP (Final RL Query)
5000	0.2479	0.3227
10000	0.2479	0.4085
15000	0.2479	0.4368
20000	0.2479	0.4648
25000	0.2479	0.4803

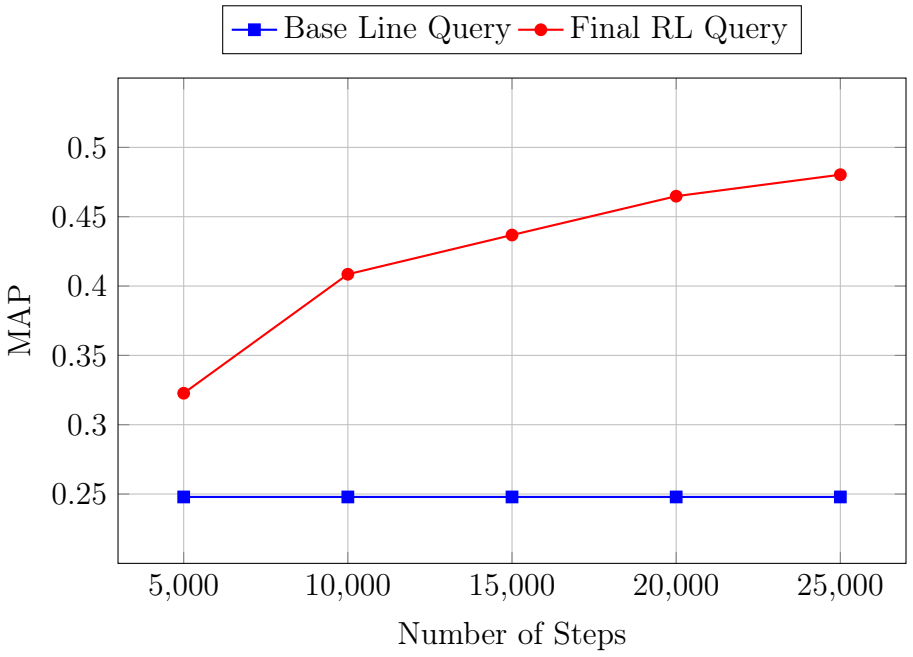

FIGURE 4.1: Comparison of MAP values for Base Line and Final RL Query at different training steps

4.4 Unweighted Query Reformulation

Here each selected term is assigned equal importance during retrieval, that is the weights for each term taken either 0 or 1.

4.4.1 Query Expansion and Initial Setup

For each query id from the TREC Robust dataset, the relevant documnets are taken from qrels file. From those relevant documents, top 20 terms are selected using `Counter` after removing the stopwords. Stopword removal is done using `EnglishAnalyzer` and `nltk` library. Among the chosen 20 terms, the initial weight is given as 0 for each terms.

4.4.2 Reinforcement Learning Framework

State representation: $\mathbf{s} \in \{0,1\}^{20}$, a binary vector indicating which of the 20 candidate terms are included in the current query.

Action representation: Each action toggles (adds or removes) exactly one term in the query. If the agent chooses to toggle the i-th bit, the action is

$$(0, \ldots, 0, \underbrace{1}_{i\text{-th term}}, 0, \ldots, 0).$$

So the total number of action is 20.

Q-Network:

- The Q-function is approximated using a Multi-layer Perceptron (MLP) regressor from `sklearn` of one hidden layer with 64 neurons.
- Input is concatenation of state (length 20) and action (length 20), a vector of length 40.
- The MLP is updated using experience replay with a buffer of size 1000 and a batch size of 16.

- The learning rate is 0.001 and the discount factor γ is set to 0.9.

Training steps: The training is done by taking 25,000 steps. At each step the agent:

- With probability ϵ, choose a random action (exploration), with probability $1 - \epsilon$ it select the action with the highest Q-value (exploitation).
- Apply the action to get the next state.
- Construct the Lucene query from the current and next state.
- Evaluate both queries to compute their AP scores.
- Reward is calculated as the difference in AP (next - current).
- Store the transition $(s, a, r, s^{'})$ in the replay buffer to update the Q-network periodically.
- ϵ-greedy policy is used with linear decay from 1.0 to 0.01.

Reward: The reward signal is designed to encourage high retrieval performance while penalizing overly long queries. So the reward used here is,

$$\text{reward} = \text{EVAL}_{next\,state} - \text{EVAL}_{current\,state}.$$

The evaluation of a query is made such a way such that long queries, which are longer than 10 terms will get penalty.

So the evaluation of a query is measured as

$$\text{EVAL}_{state\,S} = (\text{Average Precision at S}) - (\text{penalty term})$$

where

$$\text{penalty} = \begin{cases} 5e^{\left(\sum_i s_i - 10\right)}, & \text{if } \sum_i s_i > 10, \\ 0, & \text{otherwise.} \end{cases}$$

TABLE 4.3: Performance of Binary RL-based Query Reformulation

# steps	baseline MAP	MAP of Final RL Query	# Terms Selected
5000	0.2479	0.2748	9.26
10000	0.2479	0.2139	8.84
15000	0.2479	0.1813	6.68
20000	0.2479	0.1558	6.05
25000	0.2479	0.1702	4.74

Query ID	Original Query	Modified Query
601	Turkey Iraq water	'water', 'ha', 'dam', 'irrig', 'euphrat', 'region', 'us', 'financ', 'tigri'
602	Czech, Slovak sovereignty	'slovak', 'republ', 'govern', 'minist', 'said'
603	Tobacco cigarette lawsuit	'said', 'compani', 'year', 'damag', 'suit', 'cipollon'
604	Lyme disease arthritis	'lyme', 'patient', 'tryptophan', 'hi', 'mai', 'test'

Query ID	**Original Query**	**Modified Query**
605	Great Britain health care	'health', 'year', 'Britain'
606	leg traps ban	'counti', 'would', 'trap', 'anim', 'fur'
607	human genetic code	'design', 'ship', 'institut', 'construct'
608	taxing social security	'tax', 'would', 'incom'
609	per capita alcohol consumption	'beer', 'drink', 'us', 'brewer'
610	minimum wage adverse impact	'garment', 'state'
611	Kurds Germany violence	'turkish', 'govern'
612	Tibet protesters	'china', 'ha', 'right', 'lama', 'said', 'human'
613	Berlin wall disposal	'wall', 'said', 'new', 'first'
614	Flavr Savr tomato	'us', 'market', 'would', 'produc', 'year', 'engin'
615	timber exports Asia	'export', 'oregon', 'land', 'northwest'
616	Volkswagen Mexico	'year', 'mr', 'mexico', 'market', 'compani', 'volkswagen'
617	Russia Cuba economy	'econom', 'would', 'castro', 'economi'
618	Ayatollah Khomeini death	'ha', 'govern'

Query ID	**Original Query**	**Modified Query**
619	Winnie Mandela scandal	'anc', 'mr', 'ha', 'black', 'apartheid', 'list'
620	France nuclear testing	'nuclear', 'franc', 'ha', 'would', 'polit', 'peopl'

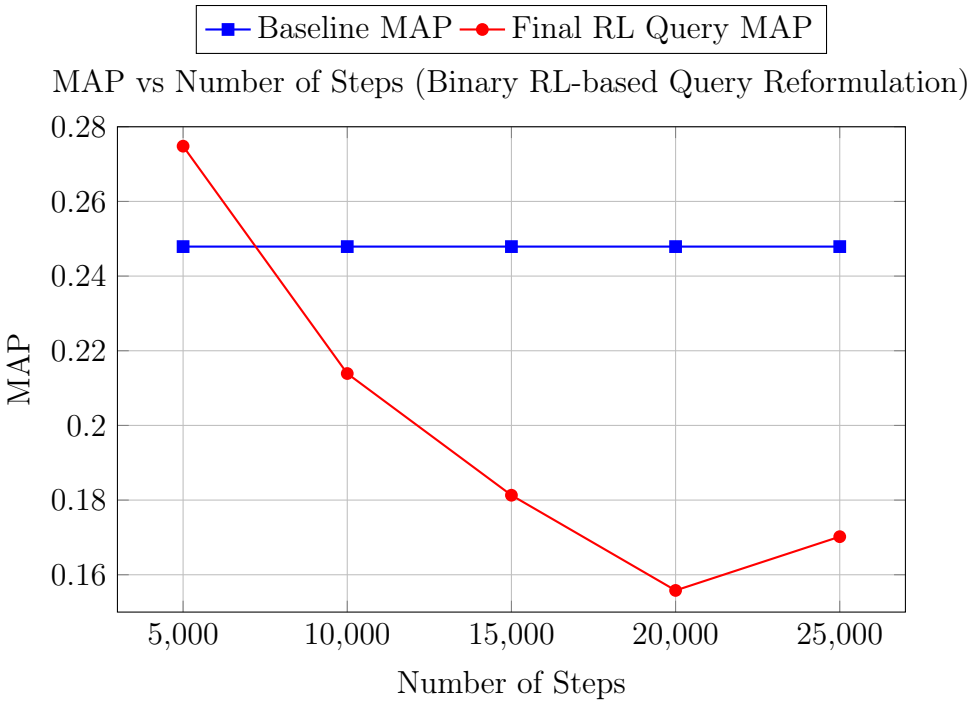

FIGURE 4.2: Comparison of MAP values for Baseline and Binary RL-based Final Query at different training steps

Chapter 5

Conclusion

This thesis investigated the application of Reinforcement Learning (RL) for optimizing query reformulation in information retrieval systems. Traditional approaches to query reformulation—primarily based on heuristic term selection and relevance feedback—often lack adaptability and generalization, especially when dealing with short or ambiguous user queries. To address these limitations, this work proposed a novel framework that leverages Deep Q-Networks (DQN) to optimize both term selection and term weighting in reformulated queries.

The proposed method frames the task as a Markov Decision Process (MDP), where each state represents a current configuration of the query, and actions correspond to selecting terms or adjusting their weights. The system receives feedback in the form of a reward, calculated using the Average Precision (AP) of retrieved documents. This allows the agent to iteratively learn optimal query representations that maximize relevance with respect to the ground truth.

Two primary strategies were explored: (1) unweighted query reformulation, where the agent selects an optimal subset of terms from a pool of candidates, and (2)

weighted query reformulation, where the agent learns to assign appropriate importance values to selected terms. Experiments demonstrated the effectiveness of both approaches. In the unweighted setting, a trade-off between query length and AP was observed; while more compact queries performed reasonably well, longer queries retained higher relevance scores. In the weighted setting, experiments revealed how fine-grained control over term importance can lead to significant improvements in retrieval effectiveness.

The integration with the Lucene search engine using PyLucene enabled realistic evaluations, ensuring that results reflect practical performance. Moreover, the framework's modular design allows it to be extended to different datasets, reward metrics, and RL algorithms.

In conclusion, this thesis presents a principled and adaptable approach to query reformulation, demonstrating that reinforcement learning can effectively learn to improve query quality based on user relevance judgments. By addressing both term selection and term weighting, the proposed system offers a comprehensive solution that bridges classical IR techniques with modern learning-based methods.

Chapter 6

Future Work

While this thesis demonstrates the potential of reinforcement learning in enhancing query reformulation, several promising avenues remain open for future exploration:

- **Alternative RL Algorithms:** This work employs Deep Q-Networks (DQN). Future research could investigate alternative methods such as Policy Gradient, PPO, or Actor-Critic models for better convergence and performance.
- **Reward Function Enhancement:** The current implementation uses MAP as the sole reward signal. Incorporating metrics such as NDCG, Precision@K, or even user feedback can offer a more nuanced learning signal.
- **Dynamic Action Space:** Future models could explore adaptive action strategies where the number and type of term modifications depend on the query complexity or performance gradient.

By pursuing these directions, future work can build upon the foundational framework presented here to develop more intelligent and adaptive information retrieval systems.

References

Buckley, C., Salton, G., 1995. Optimization of relevance feedback weights, in: Proceedings of the Text REtrieval Conference (TREC-4), National Institute of Standards and Technology (NIST), Gaithersburg, MD. pp. 351–357. URL: https://dl.acm.org/doi/10.1145/215206.215383.

Carpineto, C., Romano, G., 2012. A survey of automatic query expansion in information retrieval. ACM Computing Surveys (CSUR) 44, 1–50. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f90f1695f32d88e89773181d4b20e734ed686187.

Jiang, P., Lin, J., Cao, L., Tian, R., Kang, S., Wang, Z., Sun, J., Han, J., 2025. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. URL: https://arxiv.org/abs/2503.00223, arXiv:2503.00223.

Nogueira, R., Cho, K., 2017. Task-oriented query reformulation with reinforcement learning. URL: https://arxiv.org/abs/1704.04572, arXiv:1704.04572.

Ribeiro, M.T., Singh, S., Guestrin, C., 2016. "why should i trust you?": Explaining the predictions of any classifier. URL: https://arxiv.org/abs/1602.04938, arXiv:1602.04938.

Saha, S., Majumdar, D., Mitra, M., 2022. Explainability of text processing and retrieval methods: A critical survey. URL: https://arxiv.org/abs/2212.07126, arXiv:2212.07126.

Verma, M., Ganguly, D., 2019. Lirme: Locally interpretable ranking model explanation. SIGIR , 1281 – 1284URL: https://dl.acm.org/doi/10.1145/3331184.3331377.