UNDERSTANDING AI PATENT SEARCH: DISCRETE ACTION SPACE EXPLORATION FOR BOOLEAN QUERY GENERATION

by

Jordan Rodriguez

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2026

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Master's Committee, we certify that we have read the thesis prepared by Jordan Rodriguez, titled "Understanding AI Patent Search: Discrete Action Space Exploration for Boolean Query Generation" and recommend that it be accepted as fulfilling the thesis requirement for the Master's Degree.

Marvin Slepian
Marvin Slepian (Apr 27, 2026 16:45:50 PDT)

Dr. Marvin Slepian — Date: 04/27/2026

Mihai Surdeanu (Apr 27, 2026 17:50:03 PDT)

Dr. Mihai Surdeanu — Date: 04/27/2026

Eduardo Blanco

Dr. Eduardo Blanco — Date: 04/27/2026

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to the Graduate College.

We hereby certify that we have read this thesis prepared under our direction and recommend that it be accepted as fulfilling the Master's requirement.

Marvin Slepian
Marvin Slepian (Apr 27, 2026 16:45:50 PDT)

Dr. Marvin Slepian — Date: 04/27/2026
Master's Thesis Committee Co-Chair
Department of Medicine, Department of Biomedical Engineering, ACABI

Mihai Surdeanu (Apr 27, 2026 17:50:03 PDT)

Dr. Mihai Surdeanu — Date: 04/27/2026
Master's Thesis Committee Co-Chair
Department of Computer Science

ACKNOWLEDGMENTS

Thank you to Dr. Slepian, for offering me my first view into the world of research, and motivating the original work that led to this project. To Dr. Surdeanu, who offered the seed of an implementation idea and much guidance throughout, always believing in me to continue on. To Asiful, for his work that laid the foundation for this implementation and his advice along the way.

I would also like to thank my friends, those that I met in college and those that I have known for much longer. In particular Federico, my closest friend and companion walking along a parallel road at every step. Bennett, for patiently listening to my successes, failures, and other soliloquys along the way over many cups of tea.

And for my family: Thank you for believing in me for longer than I can remember, even in those moments when I did not believe in myself.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Patents play a vital role in protecting innovation and shaping economic growth. Inventors and patent examiners rely on effectively searching existing patents to determine whether proposed patents are novel, or already covered by prior art. AI-based patent search methods can offer utility in this process, but operate as black-boxes and lack the transparency and explainability required in a legal setting. We present an approach to dynamically refine explainable boolean queries which recreate the results of the AI-based search. To reduce the complexity of optimization on such a large search space of possible candidate terms we use ranking and filtering to narrow the broad search space into a discrete action space of candidate terms. We apply three optimization techniques to the resulting action space: Reinforcement Learning (with PPO), greedy search, and Monte-Carlo Tree Search. We find that greedy search was the most effective in our context.

Understanding AI Patent Search: Discrete Action Space Exploration for Boolean Query Generation

1 Introduction

The United States Patent and Trademark Office (USPTO) receives hundreds of thousands of patent applications every year (U.S. Patent and Trademark Office Patent Technology Monitoring Team (PTMT)). Each new application must be evaluated to determine whether its contribution is novel, or already captured by some prior art. This evaluation of novelty requires the use of search tools including AI search tools based on deep learning. However, machine-learning based search algorithms use opaque and proprietary methods which do not allow scrutiny and explanation. This is undesirable in a legal process where clarity and explanation are crucial to decision making.

To address this problem, we attempt to bridge the gap between AI search tools and traditional boolean queries by dynamically optimizing a boolean query which replicates the results of an AI patent search system. A sufficiently accurate boolean query reconstruction can act as an explanation, empowering creators and patent examiners in their understanding of the underlying logic of the systems. Figure 1 contains an example of a patent and the corresponding query our baseline system would generate based on a collection of patents relevant to it. To dynamically optimize a boolean query, we begin with a static baseline query, and iteratively add and remove terms using a discrete action space. We rank and filter candidate terms to transform the broad search space of all possible terms to add to or remove from the query into a discrete action space where the top candidates from each field are considered as possibilities. Having created a discrete action space, we evaluated three approaches for reward optimization: Reinforcement Learning (RL), a greedy tree search, and Monte-Carlo Tree Search (MCTS).

Patent: Mitochondrially targeted antioxidants (US-7109189-B2)
Abstract: The invention provides mitochondrially targeted antioxidant compounds comprising a lipophilic cation moiety covalently coupled to a glutathione peroxidase mimetic. These compounds can be used to treat patients who would benefit from the reduction of oxidative stress.
Query:

```
(cpc:A61P43/00 (ti:hydroxypheophorbide OR ti:aldolase OR ti:mitochondria OR
ti:mitoquinone)) OR (cpc:A61K31/185 (ti:malignancies OR ti:neuropathy OR ti:snakebite
OR ti:poisoning)) OR (cpc:C07F9/5442 (ti:mitochondria OR ti:mitoquinone)) OR
(cpc:C07H15/26 ti:selenol) OR (cpc:C07F9/58 ti:polyisoprenyl) OR (cpc:C07F9/091
ti:d609) OR (cpc:C07C2601/16 ti:decrease) OR (cpc:C07H15/18 ti:fucosidase) OR
(cpc:A61K31/00 ti:poisoning) OR (cpc:A61K31/44 ti:cataracts) OR (cpc:A61K31/662
ti:pro) OR (cpc:A61K31/66 ti:renal)
```

Figure 1: An example patent was used as input to an AI patent search tool, returning 50 neighbor patents. The boolean query below was generated to retrieve that particular set of 50 neighbors, reproducing the results of the AI search. The query contains combinations of Cooperative Patent Classification (cpc) codes and terms in the titles of patents. The generated query is a reflection of the common features of the neighbor patents, and a functional explanation of the AI search results.

2 Background

To understand the problem context and proposed solutions, we will outline the importance of patent search, the role of boolean queries in patent search, and the limitations in explainability of modern deep learning approaches to patent search. Then we will explore prior applications of RL and MCTS which motivate their application as potential solutions to the task of boolean query optimization.

2.1 Patent Law: Novelty and Prior Art

Intellectual property is enshrined in the U.S. constitution as a means for inventors to have exclusive rights to discoveries for a limited time (USC, 1788). Patents are a form of intellectual property that protects particular processes, machines, manufactures, and compositions of matter that are novel and non-obvious (U.S. Congress, 1952a,b). In applying for a patent, a key element is searching through the existing literature to determine that the proposed invention is **novel**, meaning that no previous patent or publicly available material contains the ideas of the proposed patent (U.S. Congress, 1952b; United States Patent and Trademark Office, 2019). To determine this, it is necessary to search through **prior art**, the existing body of literature including published patents. Prior art search is critical for both

inventors, who must find the relevant work that precedes their invention and ensure novelty, and examiners, who must go through a similar review process when determining whether to accept a proposed patent (United States Patent and Trademark Office, 2019). We will be discussing two forms of widely used prior art search: boolean queries and deep learning search tools.

2.2 Interpretability and Deep Learning in Patent Search

Boolean queries are a method of retrieving documents from a database by filtering them based on whether they contain specific words (terms). Terms in a boolean query can be combined with operators such as `AND` and `OR`, creating a specification of which combinations of terms are acceptable. For example, one could retrieve all documents that contain either 'wireless' and 'smartphone', or simply the term 'camera': `('wireless' AND 'smartphone') OR 'camera'`. Patent professionals use boolean queries to retrieve relevant documents in prior art search, with the advantage that the results returned by a boolean query are intrinsically explainable: there are precise terms in the document that match the query.

In recent years, there has been much advancement in deep learning approaches to patent analysis, including for ideation, classification, and valuation (Krestel et al., 2021). Prior art search has been improved with the use of deep learning to quantify similarity between documents, or terms, for the purpose of boolean query expansion (Helmers et al., 2019) (Aras et al., 2018) (Krishna et al., 2019). These advances and others power the underlying functionality of a tool such as Google Patents (Google), which can retrieve relevant patents for prior art search. However, although these methods may be effective at finding relevant documents, their functionality is opaque. As with many commercial search systems, the implementation details of Google Patents are not publicly available. Unlike with a boolean query, the reasoning behind a particular retrieved patent is not apparent when using deep learning search tools.

Attempts have also been made to create systems that integrate explainability into models structurally, through knowledge graphs for explainable recommendations (Chen and Deng, 2023), or graphs which connect dependent claims to better understand prior art for novelty analysis (Jang et al., 2023). These systems represent substantial progress in the application

of AI to patent novelty analysis and prior art search, and offer explanations for the results obtained. However, the approach proposed in this paper, namely constructing a boolean query as a means of explaining the result of an AI search, can be applied to any search system to better understand the results. This can be widely applied by professionals in conjunction with any search tool.

2.3 Reinforcement Learning for Query Generation

Reinforcement Learning (RL) is a paradigm of machine learning that allows applications where the reward is not integrable, or dissociated from the actions of the model. This is achieved by the creation of a discrete state space, actions, and a reward function. An agent learns to maximize the reward function by taking actions and observing the resulting changes in state and, ultimately, reward.

RL has been used in approaches to generate SQL queries from natural language (Ortiz et al., 2018) (Zhong et al., 2017). These models are used to learn optimal queries for database systems, but their approaches do not apply directly to the problem outlined here, where we attempt to create a query from terms found in collections of documents.

The most structurally similar to the approach outlined in this paper of boolean query generation is **Query Rewriting** models, which attempt to rewrite a user-generated query to more effectively retrieve desired documents. These models must use similar techniques to select terms to include in a query. Buck et al. (2018) use a black-box IR system to find the optimal rephrase for a question in a QA setting. Their approach is to create a sequence-to-sequence model for paraphrasing, which learns to optimize the quality of answers returned from the system when used to paraphrase the question.

In a similar vein, Nogueira & Cho (2017) use a similar sequence-generation approach to generate a query from terms found in (1) an original input query and (2) the set of documents retrieved from that initial query, using a language model to find the most effective and probable terms based on the query and documents. Others have used the approach of using RL to fine-tune an LLM to rephrase a user query in order to optimize accuracy of the query in retrieving relevant documents (Nguyen et al., 2025) (Jiang et al., 2025).

These approaches show that RL can be applied to the problem of filtering a large search space of terms. In our application, we will attempt to use an RL agent to choose terms from a filtered set of candidates.

2.4 Monte-Carlo Tree Search for Broad Search Spaces

Our proposed approach applies tree-search algorithms to a constructed action space, attempting to maximize reward with the minimum computational cost. One approach to this problem of exploring a broad action space is **Monte-Carlo Tree Search** (MCTS). MCTS is particularly well suited to this task because of the fact that we can evaluate the quality of alterations to the query and by evaluating performance of the boolean query. MCTS was originally developed in the context of the game Go where a key difficulty in computationally optimizing play is the large search space of potential moves, which makes brute-force search ineffective. This mirrors the difficulty in the query generation problem of choosing between a large number of possible candidate terms to add and remove. To address this problem, MCTS uses a heuristic balancing exploration of new potential moves and exploitation of the most effective known solution. From a particular board state, a root node is created. Each child of the root node represents the state of the board after one more move is executed, and the tree can grow deeper as more moves are simulated. The value of each new node is estimated by a simulation, executing a random series of moves (**rollout**) until the game terminates and a winner is determined (Coulom, 2007). To determine which moves warrant a deeper exploration, a reward function (**UCB1**) is used which balances the estimated score of all children of the node with the number of times the node has been visited, prioritizing new exploration on nodes that (1) have high expected return, and (2) have been visited infrequently (Kocsis and Szepesvári, 2006).

$$UCB1 = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

where w_i is the cumulative reward from node i, n_i is the number of visits to node i, N is the number of visits to the parent, and c is a tunable exploration constant. A higher value of c

encourages more exploration of univisited nodes, while a lower value prioritizes exploitation by expanding known high-scoring nodes.

Later Go-playing algorithms such as AlphaGo exceed human ability, using deep-learning methods to estimate the value of a particular board state rather than random rollouts (Silver et al., 2016). In our application, the reward of a particular query can be computed directly by executing the query rather than estimated, removing the need for random rollouts or a neural classifier to estimate value. We adapt MCTS to boolean query optimization where nodes represent query states and actions correspond to term additions or removals.

3 Boolean Query Generation Task

3.1 Problem Statement

The USPTO created a publicly accessible challenge on Kaggle (United States Patent and Trademark Office, 2024) which contains the dataset used in this task. For a certain target patent, 50 relevant neighbors from an AI search tool are generated. Given the target patent and the set of neighbors, the task is to create a boolean query to retrieve as many of these neighbors as possible. The boolean query reproducing the set of retrieved patents is taken as a functional explanation of the results of the AI search. This set of neighbors constitutes the **gold** data which the query should retrieve, and success will be measured using F1. F1 is the harmonic mean of two evaluation metrics: **Precision** and **Recall**. Precision measures how many of the documents retrieved by the query were from the desired gold set, and Recall measures how many of the desired documents were retrieved overall.

$$\text{Precision} = \frac{TP}{TP + FP} \quad \text{Recall} = \frac{TP}{TP + FN}$$

where TP represents *True Positives*, patents retrieved by the query and present in the gold set, FP represents *False Positives*, patents retrieved by the query but not in the gold set, and FN represents *False Negatives*, patents in the gold set not retrieved by the query.

F1 is computed by combining the two:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\,TP}{2\,TP + FP + FN}$$

A model that prioritizes only one at the expense of the other will be penalized. To optimize for F1, the query must be (1) broad enough to effectively retrieve the gold patents, and (2) narrow enough that most of what it retrieves is relevant.

The dataset contains 4000 target patents each with associated neighbors. 2400 of this set were used for training, 800 for development, and the remaining 800 as a testing set.

3.2 Static Baseline

As a baseline on the query generation task and a starting point for the iterative query optimization system, we used a deterministic system from prior work which creates a query for the set of neighbor patents using linguistic methods (Islam et al., 2026). This baseline approach uses TF-IDF ranking to select the top-k most relevant terms from each of the neighboring documents and constructs a conjunctive sub-query for each neighbor. That is to say, this sub-query is the two terms appearing in two selected fields of the patent (from the set of fields: title, abstract, description, claims, cpc) which are relatively uncommon in the whole corpus of documents, but which are common in this particular patent. These terms are combined using the `AND` operator to create the sub-query, so it is likely that each sub-query will retrieve the document it was created from. For example, patent 1 from the set of neighbors could contribute the subquery `('wireless' AND 'camera')`, patent 2 the sub-query `('wireless' AND 'smartphone')`, and so on. These sub-queries are then combined disjunctively (using the `OR` operator) to create a query for the whole set: `('wireless' AND 'camera') OR ('wireless' AND 'smartphone') OR (...)`. Finally, the query is simplified to combine redundant terms using distributive laws. Some sub-queries may contain duplicated terms, and these can be combined. In this case, our example query could be reduced to `'wireless' AND ('camera' OR 'smartphone')`.

4 Proposed Approach

Our proposed approach is an iterative environment that transforms the broad search space of potential terms to add to the query into a discrete action space using a two-part ranking system to filter to a fixed set of candidates. First, we will address how the query is con-

structed as more terms are added to the baseline, then the construction of the actions and rewards, and finally the three systems created to optimize the rewards in this environment: a Reinforcement Learning agent, greedy tree search, and Monte-Carlo Tree Search.

4.1 Query Construction

Our query formula is similar to the baseline, in that each document contributes a sub-query to the larger query. Each subquery takes the form:

```
term_1 AND ... AND term_n
```

Where `term_1`...`term_n` are terms from a particular document. These subqueries are then combined into a larger query using disjunction:

```
subquery_1 OR ... OR subquery_n
```

Distributive laws are then applied to remove redundant terms to reduce the query length. For example, the two subqueries joined by disjunction `(car AND train) OR (car AND plane)` can be reduced to `(car AND (train OR plane))`. Finally, the negated portion of the query is appended to the end in the form:

```
AND NOT (term_1 OR ... OR term_n)
```

This construction means that as new terms are added from the candidates, they are added to the subquery corresponding to the document they were taken from.

4.2 Action Space

Taking the statically generated baseline query as a starting point, we construct an adaptive environment with a discrete action space that represents possible alterations to the query. Table 1 contains the full set of actions. Each action contains a candidate term to add to or remove from the query, dynamically updated based on the documents retrieved at the previous iteration. This is possible because we have access to the gold data of neighbor patents, and can select terms likely to increase the accuracy. The potential space of query terms to add and remove is large, so to express the query-construction task as possible

discrete actions, we filter and rank candidate terms at each step of the process. For example, to find candidate terms to add to the query, we collect potential terms from the set of false negatives of the current query— patents that were in the gold set but not retrieved by the current query. Each document from the set of false negatives contributes a candidate term based on the top ranked term from each of its fields. The ranking system used to filter these terms is described below, combining TF-IDF ranking and embedding similarity with the existing query. Once a scoring system has been established, the top term from each document can be put into a list and ranked. From there, the top k terms from each field are possible actions that the environment offers (k=4 was used after some emperical evaluation). Once an action is selected, the corresponding term is added to the query and the query is executed again, retrieving a new set of documents with new terms. The reward provided by the environment is the difference in F1 score between the newly altered query and the previous query.

Action IDs	**Description**
0–3	Add positive term from `clm` (terms 0–3)
4–7	Add positive term from `cpc` (terms 0–3)
8–11	Add positive term from `ti` (terms 0–3)
12–15	Add positive term from `detd` (terms 0–3)
16	Add a negative term
17	Remove a positive term
18	Remove a negative term
19	Terminate

Table 1: The action space of the iterative query optimization environment. Actions represent potential alterations to the boolean query. Abbreviations `clm` `cpc`, `ti`, and `detd` represent the patent fields claims, cooperative patent classification codes, title, and description respectively. Candidate terms are filtered and ranked using the process outlined in Section 4.3. Optimization methods including Reinforcement Learning, greedy search, and MCTS were applied to the action space to find the best actions in each situation.

4.3 Candidate Term Scoring

Candidate terms are ranked based on a combination of TF-IDF (Term Frequency-Inverse Document Frequency) scores and embedding similarity to the overall query. TF-IDF is defined here as:

$$\text{TF-IDF}(t, d, D) = f_{t,d} \times \log \frac{N}{n_t}$$

Where t is a particular term (word), d is a particular document (patent), D is the set of all documents in the corpus, $f_{t,d}$ is the number of times term t appears in document d, N is the number of total documents, and n_t is the number of documents in D containing term t.

For computational efficiency, the TF-IDF scores of the top 5 scoring terms in each of the fields of each of the documents were cached. The inverted version of this cache, which maps a particular term to its score in each of the documents (if that term was in the top 5), was used to approximate the aggregate TF-IDF score of a term across a subset of documents. The TF-IDF scores of candidate terms are computed across distinct subsets depending on the context. For each of the types of candidate terms, there are distinct subsets used in this computation:

1. Terms to add to the query. These terms were collected from the false negatives, with their TF-IDF score computed by subtracting their score across the all the retrieved documents from their score across the false negatives.

2. A term to add to the query with a negation (`AND NOT term`). These terms are collected from the false positives, ranked based on their score in the false positives minus their score in gold set.

3. A term to remove from the positive portion the query. These terms are ranked by their score in the false positive documents minus their score in the gold set.

4. A term to remove from the negated portion of the query. These terms are assigned their score across the false negatives minus their score across the gold.

This approach seeks to maximize the likelihood that adding terms and removing terms will cause the query to accept new documents from the gold set, and remove false positives.

Embedding cosine similarity was used as another scoring mechanism. Embeddings were used to map terms and groups of terms to a semantic vector space. PatentSBERTA, a sentence transformer model trained on patent data, was applied (Bekamiri et al., 2024). Each

term in the query was assigned an embedding, and the mean was used as the whole query embedding. For CPC terms, they were replaced with the embedding for their whole description. A mapping was created of cpc codes to their descriptions (including the hierarchical portions where applicable, ie B1007 may be be 'shovels' while B10073 could be 'with retractable handles', so it would concatenate to 'shovels with retractable handles' for B10073). The candidate terms were similarly assigned embeddings, and a particular candidate term's score would be its cosine similarity to the query embedding:

$$\text{sim}(\mathbf{q}, \mathbf{t}) = \frac{\mathbf{q} \cdot \mathbf{t}}{\|\mathbf{q}\|\|\mathbf{t}\|}$$

Where q would be the query embedding vector, and t the term embedding vector.

When terms were being ranked relative to one another as candidates, the two scoring systems were combined using Reciprocal Rank Fusion (Cormack et al., 2009). For the whole set of candidate terms, two ranks were assigned based on their scores in TF-IDF across the relevant subset of documents and their embedding cosine similarity to the existing query. The two scores were combined using the formula for Reciprocal Rank Fusion to create one overall ranking:

$$RRFScore(c \in C) = \sum_{r \in R} \frac{1}{k + r(c)}$$

Where C is the set of current candidates, R is the set of two rankings, $r(c)$ is the rank of c in r, and k is a constant (in this case, 60).

This combined score ensures candidates are both lexically relevant and semantically aligned with the query. Once this RRF Score has been computed, candidate terms are selected from each of the fields of the patent, the top k (=4) scoring terms from each field being accessible as options to be added to the query.

All three approaches share a common action space and reward function, including the approach used to rank and filter candidate terms. The RL agent additionally requires a state representation to condition its policy; the greedy and MCTS approaches treat the problem as a black-box reward optimization, requiring only the ability to execute an action and observe the resulting F1.

4.4 Reinforcement Learning Agent

Figure 2 illustrates the iterative loop of the Reinforcement Learning agent in optimizing the query. The environment provides a state, action space, and reward. Unlike the other two approaches, which iterate using only the reward as signal to optimize at each step, the RL agent has access the information in the state, which allows it to learn associations between the TF-IDF and embedding scores of candidate terms, and the corresponding reward of adding those candidate terms.

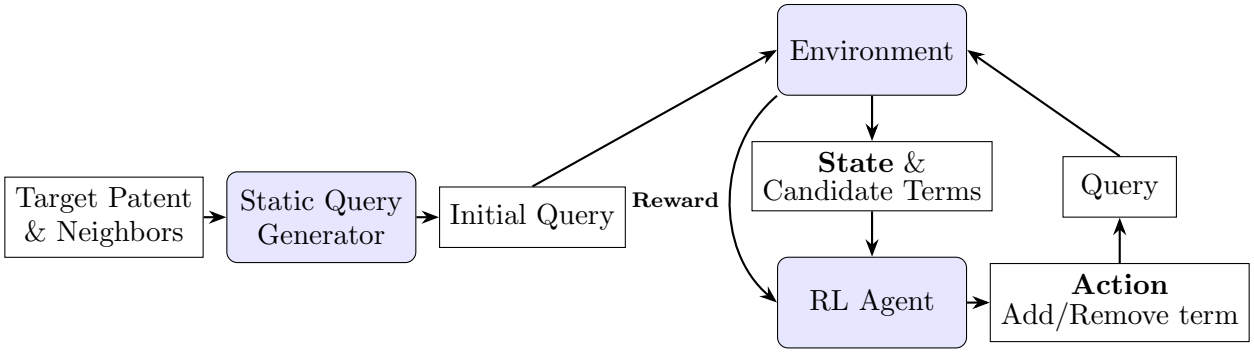

Figure 2: Architecture of the boolean query generation environment for the Reinforcement Learning agent. The system initializes a query using the baseline static query generator, then iteratively refines it using the actions selected by the agent. At each iteration, the agent chooses between possible candidate terms to add, collected from the documents intended to be retrieved.

- **State.** The state includes information about the current query and candidate terms: The number of true positives, false positives, and false negatives obtained from executing the current query, the length of the current query when reduced (including operators), number of terms in the query, and two scores for each of the 16 candidate terms: their TF-IDF score, and their embedding similarity to the existing query.

- **Actions.** There are 20 possible actions, based on the candidate terms selected: for each of the 4 fields available (title, description, cpc, and claims), the top 4 ranked terms are available to add. Further, there is an option to add a negative term (preceeded by `NOT`), an option to remove a positive term, and an option to remove a negative term. Finally, the option to terminate the current episode.

- **Reward.** The reward at each iteration is the difference in F1 from executing the action.

Once this environment is created, Proximal Policy Optimization (PPO) (Schulman et al., 2017) is used to train the model over the course of 10,000 timesteps using the training set.

4.5 Greedy Tree Search

Figure 3 illustrates the approach of greedy tree search. Using the action space of the constructed environment, the greedy tree search attempts all actions available at a particular state. The rewards are computed by re-executing the new query, and the highest reward action is chosen. This process repeats until no actions have a positive reward, and the terminate action is chosen.

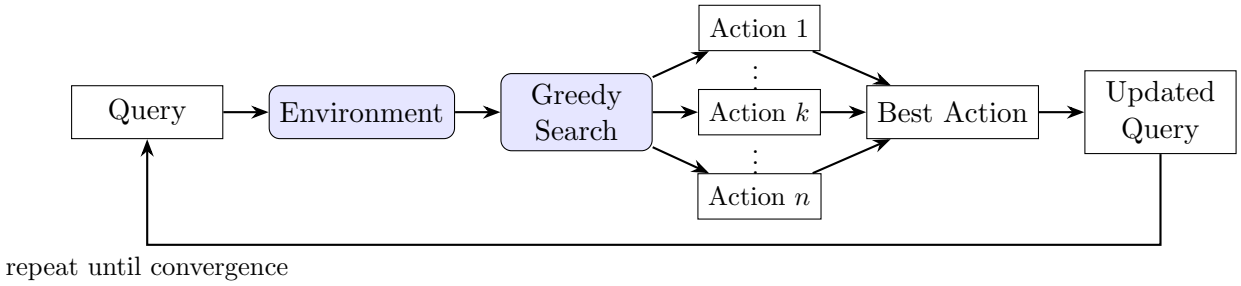

Figure 3: Greedy tree search evaluates all candidate actions at each step and selects the one yielding the highest F1, iterating until no improvement is found.

4.6 Monte-Carlo Tree Search

Monte-Carlo Tree Search, as explained in Section 2.4, is suited to the task of exploring branching possibilities in the search space of possible queries. The algorithm creates a tree of possible states, and after exploring and estimating the average value of each action, chooses the one highest average reward action, before iterating again and exploring further. Figure 4 illustrates an example of MCTS. The algorithm proceeds by exploring possible actions, creating a tree of potential new states with exploration and exploitation balanced by the UCB1 reward function (see Section 2.4). Nodes with higher UCB1 either have higher expected reward, or have not been explored as much as other nodes. The constant c balances between these two factors. In our case, to calibrate with the rewards (marginal changes in F1), we set $c = 0.2$. In our implementation, nodes of the tree each represent a query, and the reward for a particular child is the cumulative change in F1 from the root to that child.

The algorithm iterates by propagating the children of the node with the highest UCB1 score, iterating and updating the expected average reward of each child of the root. After a fixed number of iterations (in our case, 60), the child of the root with the highest expected score is selected as the next action. The remaining piece of the tree underneath that child is preserved for the next iteration of the algorithm. This approach, unlike the greedy search, has the potential to choose actions that may not be the optimal next move, but lead to a higher average reward down the line based on the continued exploration.

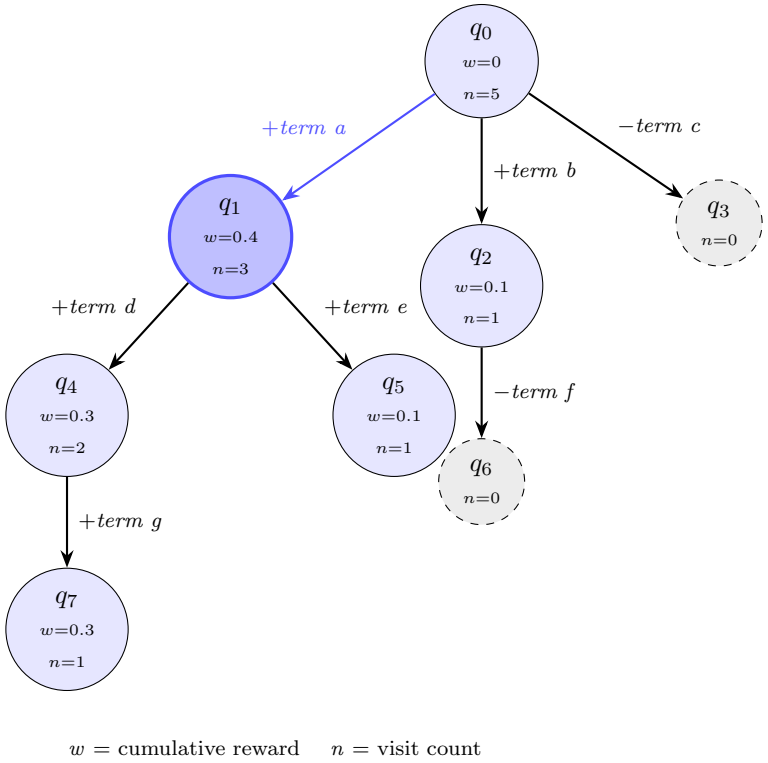

Figure 4: MCTS builds a search tree over query states. Nodes represent query states annotated with cumulative reward w and visit count n. UCB1 guides exploration, expanding high-reward and under-visited nodes. The highlighted blue node (q_1) is the action with the highest expected reward, which would be selected given this tree. Subsequent exploration would begin with q_1 as the root.

5 Results

Each of the three optimization approaches over the action space, Reinforcement Learning, greedy search, and Monte-Carlo Tree Search, were tested on the 800 patent development set. The results are displayed in Table 2. A box plot of the F1 score distribution of each setting

Method	F1
Static Baseline	0.557
RL (PPO)	0.557
Greedy Search	0.643
MCTS	0.590

Table 2: Comparison of query optimization methods by mean F1 score in an evaluation over 800 examples.

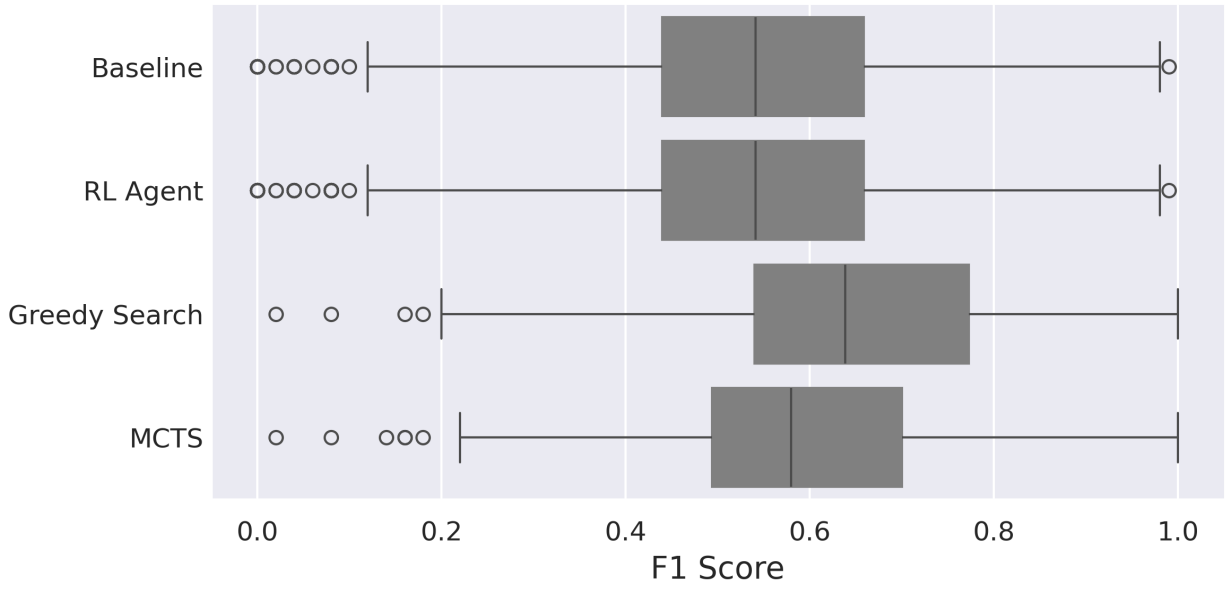

Figure 5: Boxplot illustrating the distribution of F1 for the various optimization methods. Each was tested over the 800 patent development set.

is shown in Figure 5. The best performing method was greedy search, with an average F1 of 0.643 across all runs, an increase of 0.086 in F1. The worst-performing optimization method was the RL agent, with a roughly equivalent performance to baseline. This is explored below. Of the two tree search methods, the simpler greedy search method was higher scoring than MCTS.

5.1 Reinforcement Learning Training Progression

Figure 6 contains line graphs which show the relative proportions of actions selected throughout the training process. The agent tended to favor early termination more as training progressed. This increasingly conservative approach seems to indicate that the ac-

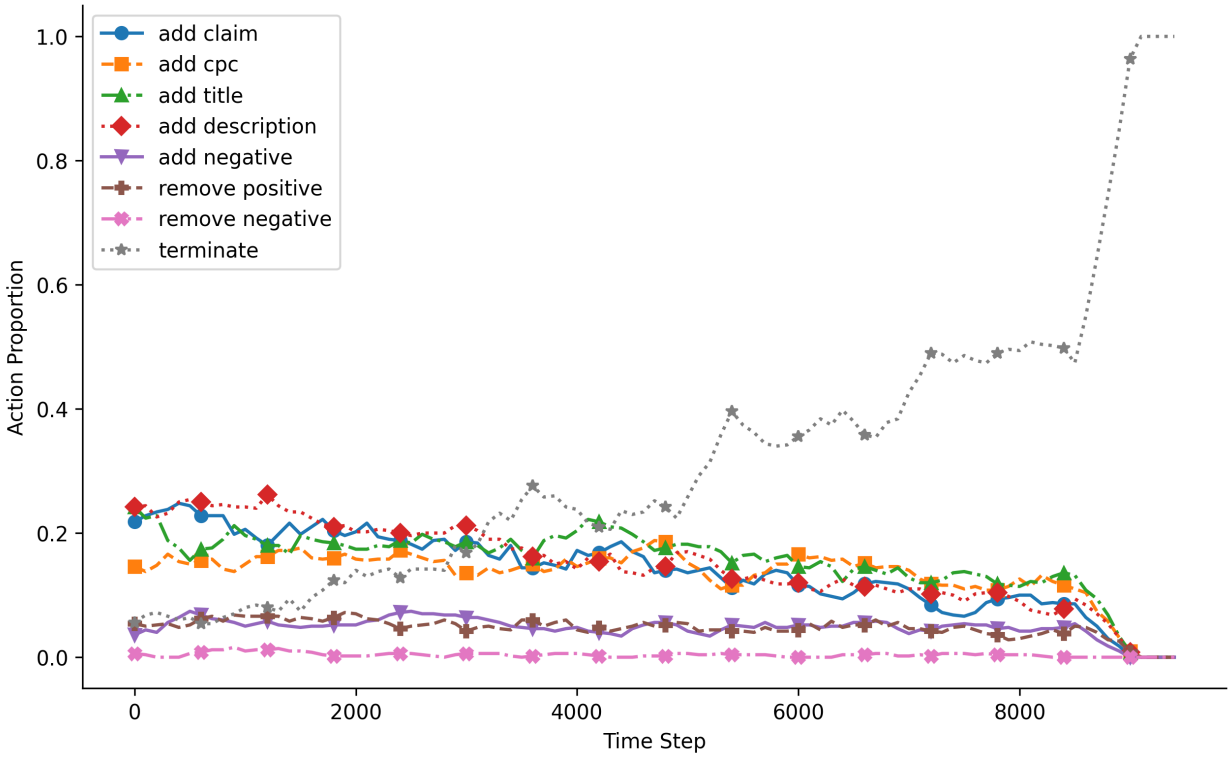

Figure 6: Action proportions selected by the Reinforcement Learning agent throughout the training process. Actions to add terms are grouped based on the field they came from in the original patent. Over time, the model converges toward selecting the terminate action.

tions which add and remove from the query do not tend to increase the reward substantially enough to incentivize them. Considering that the RL agent had access to information about the current query as well as scores for the candidate terms, it seems that these pieces of information are not sufficient to learn a policy that correctly maps the information to the correct candidate to maximize reward. In a sparse reward environment, that proxy information for the potential value of a candidate term is not enough.

5.2 Tree Search Method Comparison

Although both tree search methods increased performance over the baseline, greedy search showed a higher improvement than Monte-Carlo Tree Search. This is an interesting result, considering that MCTS has been demonstrated to be effective at optimization problems in large search spaces. However, there are a few potential explanations for this difference. Firstly, the cost of evaluating a query was a limiting factor in the exploration constant of the

algorithm. With a budget of 60 iterations and 20 potential actions, the depth of exploration is limited. To counterbalance this fact, the low c constant encouraged exploitation rather than exploration (because upon finding a high-reward child, it would be best to continue branching to its children), but this is a difficult balance to tune, because if the highest scoring child is not explored then the model cannot find the optimum that greedy search would have found. Another limiting factor in such a sparsely populated reward environment is that the average reward of a child could easily be skewed by encountering negative or low-scoring children, even if one particular path would be the highest scoring, that path may not have the highest average reward, which is used in MCTS to select the next action. Ultimately, the higher compute cost of evaluating many potential branches, then discarding the majority of the work at each iteration, did not result in a greater capacity to optimize the reward.

To illustrate the differences between the two methods, an example patent with the corresponding queries and actions performed is included in Figure 7. In this example, the greedy search performed more actions than MCTS did. Although the benefit of MCTS is that it may explore each potential action more deeply before making a selection, with such a large action space it did not find the actions that greedy found at each iteration that would make progressive improvements to the query. Greedy performed more actions, because at each iteration, there was another action to take with assured performance improvement. Patterns like this where MCTS was more conservative than greedy explain some of the gap in performance between the two systems.

6 Conclusion

Prior art search is an important part of the patent application process, yet the AI tools increasingly used for this task offer little transparency into their results. We address this by framing boolean query reconstruction as a means of explaining black-box patent search, constructing a dynamic discrete environment and evaluating three optimization approaches: Reinforcement Learning, greedy search, and Monte-Carlo Tree Search. Greedy search proved most effective, achieving an F1 of 0.643 on the development set, suggesting that the sequential nature of query construction favors exhaustive local search over learned policies or broader tree exploration. These results demonstrate that boolean query reconstruction is

Patent: Mitochondrially targeted antioxidants (US-7109189-B2)
Abstract: The invention provides mitochondrially targeted antioxidant compounds comprising a lipophilic cation moiety covalently coupled to a glutathione peroxidase mimetic. These compounds can be used to treat patients who would benefit from the reduction of oxidative stress.
Baseline Query:

```
(cpc:A61P43/00 (ti:hydroxypheophorbide OR ti:aldolase OR ti:mitochondria OR
ti:mitoquinone)) OR (cpc:A61K31/185 (ti:malignancies OR ti:neuropathy OR ti:snakebite
OR ti:poisoning)) OR (cpc:C07F9/5442 (ti:mitochondria OR ti:mitoquinone)) OR
(cpc:C07H15/26 ti:selenol) OR (cpc:C07F9/58 ti:polyisoprenyl) OR (cpc:C07F9/091
ti:d609) OR (cpc:C07C2601/16 ti:decrease) OR (cpc:C07H15/18 ti:fucosidase) OR
(cpc:A61K31/00 ti:poisoning) OR (cpc:A61K31/44 ti:cataracts) OR (cpc:A61K31/662
ti:pro) OR (cpc:A61K31/66 ti:renal)
```

MCTS Actions:

```
Add ‘detd:carbonylaminoethylsulfonic, Add ‘detd:tritylsulfanyl’
```

MCTS Final Query

```
(cpc:A61K31/185 (ti:malignancies OR ti:poisoning OR ti:neuropathy OR ti:snakebite))
OR (cpc:A61P43/00 (ti:mitoquinone OR ti:aldolase OR ti:mitochondria OR
ti:hydroxypheophorbide)) OR (cpc:C07F9/5442 (ti:mitoquinone OR ti:mitochondria)) OR
(cpc:C07F9/58 ti:polyisoprenyl) OR (cpc:A61K31/66 ti:renal) OR (cpc:A61K31/44
ti:cataracts) OR (cpc:C07F9/091 ti:d609) OR (detd:tritylsulfanyl) OR (cpc:A61K31/662
ti:pro) OR (cpc:A61K31/00 ti:poisoning) OR (cpc:C07C2601/16 ti:decrease) OR
(cpc:C07H15/26 ti:selenol) OR (detd:carbonylaminoethylsulfonic) OR (cpc:C07H15/18
ti:fucosidase)
```

Greedy Actions:

```
Add ‘clm:absorbing’; Add ‘detd:fcot’; Add ‘ti:hydroxypheophorbide’,
Add ‘clm:carboranyl’ Add ‘detd:carbonylaminoethylsulfonic’,
Add ‘ti:diaminoalkyne’ Add ‘ti:shikimate’, Add ‘ti:arsenoxide’,
Add ‘ti:diethylaminobenzenethiosulphonic’, Add ‘detd:dihexylporphyrin’,
Add ‘detd:ccccv’, Add ‘clm:heterolytic’, Add ‘detd:rutschow’, Add ‘detd:gluoh’,
Add ‘detd:bonalfa’, Add ‘detd:benzylphloretin’, Add ‘cpc:A61K31/105’
```

Greedy Final Query:

```
(cpc:A61P43/00 (ti:hydroxypheophorbide OR ti:aldolase OR ti:mitochondria OR
ti:mitoquinone)) OR (cpc:A61K31/185 (ti:malignancies OR ti:neuropathy OR ti:snakebite
OR ti:poisoning)) OR (cpc:C07F9/5442 (ti:mitochondria OR ti:mitoquinone)) OR
(cpc:C07H15/26 ti:selenol) OR (ti:diaminoalkyne) OR (detd:rutschow) OR
(detd:benzylphloretin) OR (cpc:C07F9/58 ti:polyisoprenyl) OR (ti:arsenoxide) OR
(cpc:C07F9/091 ti:d609) OR (detd:ccccv) OR (detd:gluoh) OR (cpc:C07C2601/16
ti:decrease) OR (ti:lipophenol) OR (cpc:C07H15/18 ti:fucosidase) OR (cpc:A61K31/00
ti:poisoning) OR (cpc:A61K31/44 ti:cataracts) OR
(detd:diethylaminobenzenethiosulphonic) OR (detd:bonalfa) OR (cpc:A61K31/662 ti:pro)
OR (detd:dihexylporphyrin) OR (detd:carbonylaminoethylsulfonic) OR (cpc:A61K31/105)
OR (ti:shikimate) OR (clm:carboranyl) OR (clm:heterolytic) OR (cpc:A61K31/66
ti:renal)
```

Figure 7: An example of a baseline query, actions taken by the two tree search methods, and the final resulting queries. For the terms added to the query, the prefixes indicate the field: ‘clm’ for claims, ‘cpc’ for the Cooperative Patent Classification codes, ‘ti’ for title, ‘detd’ for description. The final query from greedy search has an F1 of 0.82, compared to the original F1 of 0.58. For the same patent, MCTS processing resulted in a final F1 of 0.64.

a viable explanation method for AI patent search, applicable to any search system without modification.

6.1 Limitations and Future Work

In this application, the failure of RL because of the convergence towards early termination seems to suggest that the model could not sufficiently learn from the sparse examples of positive actions that contribute to query effectiveness. To address this concern, other RL algorithms could be applied, potentially off-policy approaches such as DQN, which can use rewards repeatedly with replay buffers to better optimize in such a sparse reward landscape.

The disparity in performance between the two tree search algorithms could potentially be alleviated by making alterations to MCTS. One possibility is to further tune the hyperparameter C and runtime compute budget, allowing the algorithm as is to potentially explore more successfully. Another potential alteration is to change the algorithm to only account for positive rollouts, excluding the influence of negatively scoring children that reduced the likelihood of picking otherwise highly-scoring nodes. Another possibility is applying another search algorithm, such as beam search.

Besides the learning algorithm limitations, there is also the question of scope. The experiments conducted here used a limited index of the entire patent dataset. To test the system more robustly, the queries could be applied to a larger index of patent documents. Beyond that, for the applied purpose of prior art search, patents are not the only relevant material to be retrieved. Future work could expand to search systems that include not only patents from the USPTO but other patent agencies, and other forms of disclosure. Incorporating more diverse sources of potential prior art would involve new document types, structures, and linguistic patterns to consider, but would ultimately greatly increase the practical utility for prior art search.

References

U.S. Constitution, Article I, Section 8, 1788. URL https://constitution.congress.gov/browse/article-1/section-8/. Ratified June 21, 1788.

H. Aras, R. Türker, D. Geiss, M. Milbradt, and H. Sack. Get your hands dirty: Evaluating word2vec models for patent data. In *International Conference on Semantic Systems*, 2018. URL https://api.semanticscholar.org/CorpusID:52189726.

H. Bekamiri, D. S. Hain, and R. Jurowetzki. Patentsberta: A deep nlp based hybrid model for patent distance and classification using augmented sbert. *Technological Forecasting and Social Change*, 206:123536, 2024. ISSN 0040-1625. doi:https://doi.org/10.1016/j.techfore.2024.123536. URL https://www.sciencedirect.com/science/article/pii/S0040162524003329.

C. Buck, J. Bulian, M. Ciaramita, W. Gajewski, A. Gesmundo, N. Houlsby, and W. Wang. Ask the right questions: Active question reformulation with reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=S1CChZ-CZ.

H. Chen and W. Deng. Interpretable patent recommendation with knowledge graph and deep learning. *Scientific Reports*, 13:2586, Feb. 2023. ISSN 2045-2322. doi:10.1038/s41598-023-28766-y.

G. V. Cormack, C. L. A. Clarke, and S. Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA, 2009. Association for Com-

puting Machinery. ISBN 9781605584836. doi:10.1145/1571941.1572114. URL https://doi.org/10.1145/1571941.1572114.

R. Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, H. J. Van Den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games*, volume 4630, pages 72–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-75537-1 978-3-540-75538-8. doi:10.1007/978-3-540-75538-8_7.

Google. Google patents. https://patents.google.com. Accessed: 2025.

L. Helmers, F. Horn, F. Biegler, T. Oppermann, and K.-R. Müller. Automating the search for a patent's prior art with a full text similarity search. *PLOS ONE*, 14(3):e0212103, Mar. 2019. ISSN 1932-6203. doi:10.1371/journal.pone.0212103.

M. A. Islam, R. Vacareanu, J. Rodriguez, M. Surdeanu, and M. J. Slepian. Boolean explanations bring interpretability to artificial intelligence search in patent prior art assessment. *PLOS One*, 2026. Under review.

H. Jang, S. Kim, and B. Yoon. An eXplainable AI (XAI) model for text-based patent novelty analysis. *Expert Systems with Applications*, 231:120839, Nov. 2023. ISSN 09574174. doi:10.1016/j.eswa.2023.120839.

P. Jiang, J. Lin, L. Cao, R. Tian, S. Kang, Z. Wang, J. Sun, and J. Han. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=u9JXu4L17I.

L. Kocsis and C. Szepesvári. Bandit Based Monte-Carlo Planning. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning:*

ECML 2006, volume 4212, pages 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-45375-8 978-3-540-46056-5. doi:10.1007/11871842_29.

R. Krestel, R. Chikkamath, C. Hewel, and J. Risch. A survey on deep learning for patent analysis. *World Patent Information*, 65:102035, June 2021. ISSN 01722190. doi:10.1016/j.wpi.2021.102035.

A. Krishna, Y. Jin, C. Foster, G. Gabel, B. Hanley, and A. Youssef. Query Expansion for Patent Searching using Word Embedding and Professional Crowdsourcing, Nov. 2019.

D. A. Nguyen, R. K. Mohan, S. Yang, P. S. Akash, and K. C.-C. Chang. Minielm: A lightweight and adaptive query rewriting framework for e-commerce search optimization. In *Findings of the Association for Computational Linguistics: ACL 2025*, page 6952–6964. Association for Computational Linguistics, 2025. doi:10.18653/v1/2025.findings-acl.363. URL http://dx.doi.org/10.18653/v1/2025.findings-acl.363.

R. Nogueira and K. Cho. Task-oriented query reformulation with reinforcement learning. In M. Palmer, R. Hwa, and S. Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 574–583, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi:10.18653/v1/D17-1061. URL https://aclanthology.org/D17-1061/.

J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning State Representations for Query Optimization with Deep Reinforcement Learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, DEEM'18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450358286. doi:10.1145/3209889.3209890. URL https://doi.org/10.1145/3209889.3209890.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham,

N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan. 2016. ISSN 0028-0836, 1476-4687. doi:10.1038/nature16961.

United States Patent and Trademark Office. Patent process overview, Jan. 2019. URL https://www.uspto.gov/patents/basics/patent-process-overview. Last updated September 24, 2025.

United States Patent and Trademark Office. USPTO — explainable AI for patent professionals. Kaggle Competition, 2024. URL https://www.kaggle.com/competitions/uspto-explainable-ai/overview.

U.S. Congress. Inventions patentable, 35 U.S.C. 101, 1952a. Patent Act of 1952.

U.S. Congress. Conditions for patentability; novelty, 35 U.S.C. 102, 1952b. Patent Act of 1952.

U.S. Patent and Trademark Office Patent Technology Monitoring Team (PTMT). U.S. Patent Statistics Summary Table, Calendar Years 1963 to 2020, 05/2021 update. https://www.uspto.gov/web/offices/ac/ido/oeip/taf/us_stat.htm.

V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017. URL https://arxiv.org/abs/1709.00103.