Universiteit Leiden
The Netherlands

Bachelor Data Science and Artificial Intelligence

Optimizing Agentic Retrieval-Augmented Generation using Reinforcement Learning-Based Methods

Nataliia Bagan

Supervisors:
Zhaochun Ren
Jujia Zhao

External supervisors:
Zhengliang Shi

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 01/07/2025

Abstract

Retrieval-Augmented Generation (RAG) has become a popular method for solving the problem of automated question answering (QA). However, naïve RAG is less effective in QA tasks involving multiple steps or reasoning. Agentic RAG has been recently introduced as a solution to this. This bachelor's thesis explores the integration of Reinforcement Learning (RL) techniques into agentic RAG systems for multi-hop question answering. Specifically, we investigate how Monte-Carlo Tree Search (MCTS) can be employed to guide an autonomous agent in decomposing complex user queries into subqueries, retrieving relevant information, and synthesizing accurate answers. We improve the pipeline through 5 approaches and analyze the usage of multiple LLMs. We find that with all improvements, MCTS reaches and slightly surpasses previous baselines using iterative retrieval, improving accuracy from 8.8% to 11.2% on multi-hop datasets, such as HotpotQA.

Glossary

Term	**Description**	**Type**
Symbols		
x	User query	str
y	Model's answer	str
Z	Knowledge base of documents in RAG	List[str]
z	Retrieved document	str
Abbreviations		
CEM	Cover Exact Match (score)	
EM	Exact Match (score)	
LLM	Large Language Model	
LM	Language Model	
MCTS	Monte-Carlo Tree Search	
QA	Question Answering	
RAG	Retrieval-Augmented Generation	
RL	Reinforcement Learning	
UCT	Upper Confidence Bounds applied to Trees (score)	

Contents

1 Introduction

Large Language Models (LLMs) show remarkable capabilities in natural language understanding and generation. One of the most prominent applications of LLMs is answering user questions, particularly in tasks that require reasoning or access to world knowledge, such as open-domain question answering (ODQA). Various techniques have been proposed to improve LLMs' performance in these settings, including prompting strategies like Chain-of-Thought (CoT), which encourages step-by-step reasoning [1].

Despite these advancements, LLMs still struggle with complex queries that require integrating knowledge from multiple sources. Retrieval-Augmented Generation (RAG) has emerged as a powerful solution to this challenge [2]. In RAG, the model retrieves relevant documents from an external knowledge source (e.g., Wikipedia) and conditions its answer on both the original question and the retrieved information. Variants of this method, such as IRCoT, combine retrieval with CoT-style prompting to improve intermediate reasoning [3].

However, as tasks grow more complex, such as those that require high contextual performance, scalability, and multi-step reasoning, traditional RAG pipelines are often insufficient. Various enhancements have been introduced, for instance Chain-of-Retrieval Augmented Generation (CoRAG) and Iter-RetGen, which allow for iterative RAG and dynamic reformulation of the user query, similar to CoT [4], [5]. Such approaches have led to the development of agentic RAG, where an autonomous agent coordinates the retrieval and generation processes [6]. This extended RAG workflow often meets stricter baselines in multi-hop QA, where answering a query involves multiple inference steps and combining evidence from various sources. For example, the agent may break a complex question into sub-questions, retrieve documents for each one, and then synthesize the information into a final answer.

Training such agents and RAG systems remains a challenge. It is often unclear how to best optimize the retrieval strategies or evaluate the long-term effects of each retrieval decision. Reinforcement Learning (RL) usually offers a promising framework to address this challenge [7], [8]. By modeling the agent as a decision-maker receiving feedback in the form of rewards, we can use RL techniques to improve its behavior iteratively. Techniques such as Monte-Carlo Tree Search (MCTS) and the actor-critic algorithm allow the agent to optimize its performance over multiple steps. Some recent works, such as Search-R1 and RAG-Star, have investigated this direction [9], [10].

RAG, especially agentic RAG integrated with RL techniques, is still a novel research area. Much remains unexplored, particularly regarding how agents should be trained and how various RL methods perform in this setting. In this thesis, we study agentic RAG through the lens of Reinforcement Learning. We test several techniques built around Monte-Carlo Tree Search (MCTS) and evaluate their effectiveness in improving RAG systems in the multi-hop QA tasks. We compare their performance against existing agentic RAG baselines, such as Search-R1 [9].

1.1 Problem formulation

Our main research question is:

How can RL-based methods, specifically MCTS and reward modeling, improve the answer quality of agentic RAG models in multi-hop QA tasks?

That is, given a user query x, how can we improve the final answer y generated by the RAG system to achieve higher performance on standard QA evaluation metrics like exact match, accuracy, and F1 score?

To investigate this, we further break it down into the following sub-questions:

RQ1: How to implement MCTS for query planning in agentic RAG systems and what is its effect on the quality and efficiency of the RAG model?

RQ2: How do MCTS-enhanced agentic RAG systems perform compared to established baselines like Search-R1 on benchmark multi-hop QA datasets in terms of final answer accuracy, exact match, and F1 score?

1.2 Thesis overview

Section 2 (Related work) delves deeper into the theoretical background of the thesis, specific RL strategies used, and existing relevant research. Section 3 (Methodology) explains the overview of the techniques and approaches used in the thesis. Section 4 (Implementation) details the algorithms used and implemented technically. Section 5 (Experiments) describes the details of the conducted experiments, introduces the considered datasets and benchmarks, and shows their outcome. Section 6 (Discussion) gives more profound insights into the observed results. Section 7 (Conclusions) summarizes the work and suggests further improvements and ideas.

2 Related work

2.1 Reinforcement Learning

Reinforcement learning (RL) is a framework where an agent learns to make decisions by interacting with an environment, aiming to maximize cumulative reward over time. A policy guides the agent's actions, while a reward function evaluates the outcomes. RL is well-suited for tasks requiring sequential decision-making and exploration strategies.

RL has been used in natural language processing in areas such as dialogue systems, recommender systems, and question answering. For instance, LLaMA-Berry utilizes Monte-Carlo Tree Search (MCTS) to improve reasoning in LLMs for mathematical question-answering [11]. In the context of RAG systems, RL provides a way to optimize document retrieval and answer generation dynamically through reward-guided training.

Below are some RL methods relevant to this work.

2.1.1 Monte-Carlo Tree Search

Exhaustive tree search in search problems is computationally expensive and often redundant. Monte-Carlo Tree Search (MCTS) provides a practical alternative, balancing exploration and exploitation in large search spaces. It involves four main steps:

1. Selection Starting from the root node, the algorithm recursively selects child nodes based on a selection policy until it reaches a node that is not fully expanded or a leaf node.

2. Expansion If the selected node is not a terminal state, one or more child nodes are added to the tree. These new nodes represent possible future states that extend the current path.

3. Simulation A simulation (also called a rollout) is performed from the newly added node to estimate the potential outcome. Child nodes are explored randomly until a terminal state is reached or a predefined depth is met. The reward of the initially selected node is estimated.

4. Backpropagation The result of the simulation is propagated back up the tree. Each node along the path from the expanded node to the root updates its reward based on the following node's reward.

This way, the value of a non-terminal node can be approximated using rollouts, and thus, not the entire search space must be explored. Backpropagation step refines the estimated values. Selection and expansion steps guide the exploration of the subspace.

2.1.2 Actor-critic algorithms

Actor-critic algorithms use two separate models: the actor, which selects actions (e.g., which document to retrieve), and the critic, which evaluates those actions by providing a reward signal. This dynamic feedback loop helps train the agent more effectively, especially in complex, multi-step tasks. For instance, Search-R1 uses an actor-critic setup to iteratively refine its retrieval and reasoning process in RAG [9].

Sometimes these algorithms are enhanced with reverse curriculum learning. It starts training from states that are close to the goal and gradually moves the initial states further away [12]. In the context of RAG, this could mean providing the agent with increasingly less contextual information for a complex question, helping it learn the retrieval strategy step-by-step.

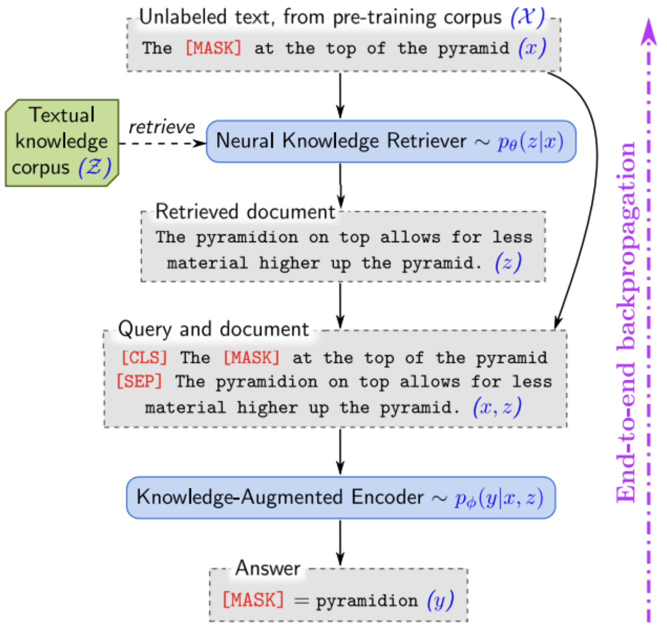
2.2 Retrieval-Augmented Generation

RAG systems have two key components: the retrieval module and the generation model. The retrieval module identifies and retrieves the most relevant documents (z_i) from a knowledge base (Z) based on the given user's query (x). Consequently, the generation model uses these retrieved documents as context to produce an educated answer (y) (Figure 1). RAG was first introduced as a knowledge-augmented extension of encoder [2], and was later adapted for seq2seq (encoder-decoder) architecture [13], therefore enabling various generative module architectures.

2.2.1 RAG and search

Iterative strategies for reasoning and knowledge-intensive tasks have shown promising results, such as Chain-of-Thought (CoT), Interleaved Retrieval with Chain-of-Thought reasoning (IRCoT), Chain-of-Retrieval Augmented Generation (CoRAG), Iter-RetGen, and Search-in-the-Chain [1], [3], [4], [5], [14]. These approaches guide the model through intermediate steps of reasoning, retrieval, and retrieval-augmented generation rather than expecting an answer after one iteration.

Search methods can be used to explore such multi-step reasoning paths. In a tree-based setup, each node can represent a reasoning step, sub-query, or a step to answer a multi-hop question. For example, RAG-Star applies MCTS to generate and evaluate possible decompositions of a complex



Unlabeled text, from pre-training corpus $(\mathcal{X})$
The [MASK] at the top of the pyramid (x)
Textual knowledge corpus $(\mathcal{Z})$
retrieve
Neural Knowledge Retriever $\sim p_\theta(z|x)$
Retrieved document
The pyramidion on top allows for less material higher up the pyramid. (z)
Query and document
[CLS] The [MASK] at the top of the pyramid [SEP] The pyramidion on top allows for less material higher up the pyramid. (x, z)
Knowledge-Augmented Encoder $\sim p_\phi(y|x, z)$
Answer
[MASK] = pyramidion (y)
End-to-end backpropagation

Figure 1: Retrieval-Augmented Generation (RAG) pipeline. Source: [2].

query [10]. During simulation, each candidate sub-query is evaluated locally, and the outcomes are scored to guide the selection of the most promising inference paths.

2.2.2 RAG and Reinforcement Learning

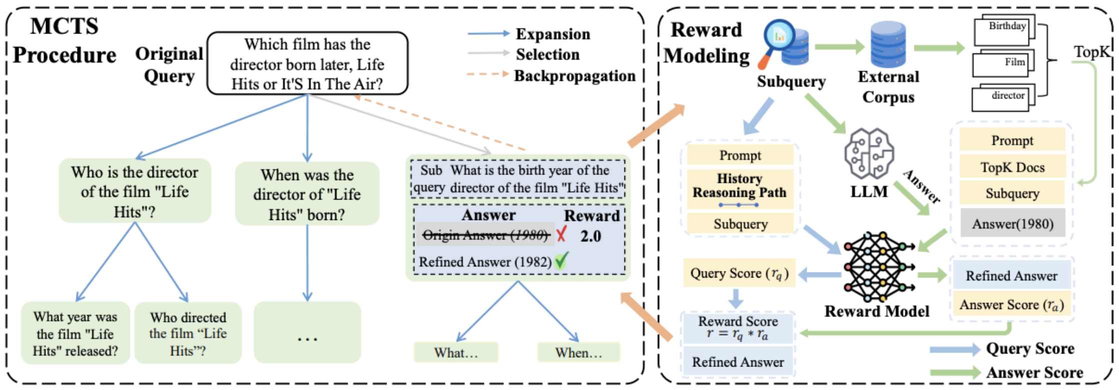
RL can be used to improve both the retrieval and generation components of RAG systems. Reward modeling enables self-reflection: an agent can evaluate the quality of its retrieved context and generated answers and adjust its behavior accordingly. The reward can be either a fixed function, like exact string matching (EM), or a separate reward model can be trained, as in an actor-critic environment, discussed in Section 2.1.2.

Several methods have explored this combination, such as Self-RAG and RAG-DDR [7], [8]. In particular, Search-R1 uses an actor-critic setup where the LLM is prompted to iteratively decompose and answer multi-hop questions, with the critic providing dynamic rewards [9]. RAG-Star, in turn, leverages MCTS to reason over potential query decompositions [10].

Search-R1. Search-R1 uses an engineered prompt to enforce iterative retrieval to improve performance on complex multi-hop questions. This prompt instructs a language model to repeat the RAG procedure, reformulate sub-queries when needed, and ultimately provide a final answer. This way, the agent, which is the Search-R1 model itself, conducts multi-step reasoning, retrieves evidence, and generates responses in a unified system. This thesis aims to compare whether the MCTS procedure outperforms the described iterative prompting algorithm.

Moreover, the Search-R1 study applies an actor-critic algorithm to train Qwen2.5, later referred to as Search-R1-Qwen2.5, for this specific setup. This RL-based fine-tuning helps the model yield better answers.

RAG-Star. RAG-Star combines search-based reasoning with retrieval and generation [10]. It uses MCTS to guide retrieval by searching through different query decomposition paths. Each simulation tests candidate sub-queries by retrieving documents and evaluating the quality of intermediate answers. Promising paths are reinforced during backpropagation, helping the system prioritize more effective strategies, ultimately leading to more precise and informative responses.



MCTS Procedure
Original Query
Which film has the director born later, Life Hits or It'S In The Air?
Expansion
Selection
Backpropagation
Who is the director of the film "Life Hits"?
When was the director of "Life Hits" born?
Sub What is the birth year of the query director of the film "Life Hits"
Answer
Reward
~~Origin Answer (1980)~~ ✗
2.0
Refined Answer (1982) ✓
What year was the film "Life Hits" released?
Who directed the film "Life Hits"?
…
What…
When…
Reward Modeling
Subquery
External Corpus
Birthday
Film
director
TopK
Prompt
History Reasoning Path
Subquery
LLM
Answer
Prompt
TopK Docs
Subquery
Answer(1980)
Query Score (r_q)
Reward Model
Refined Answer
Answer Score (r_a)
Reward Score $r = r_q * r_a$
Refined Answer
Query Score
Answer Score

Figure 2: Framework of the RAG-Star approach. Source: [10].

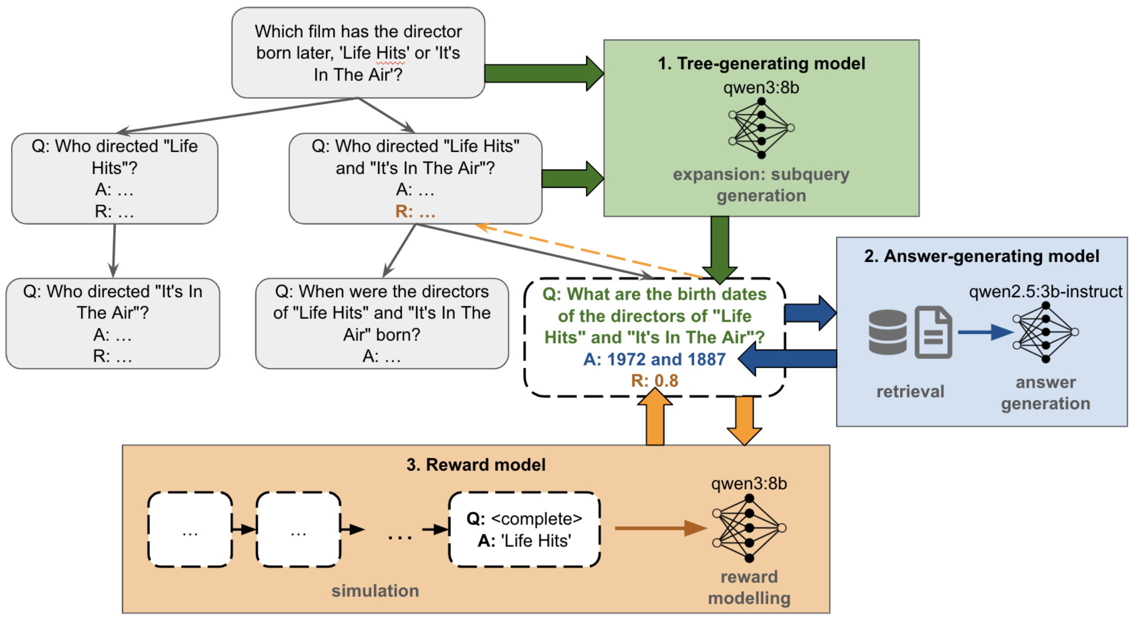
However, RAG-Star has multiple directions for improvement to consider. Its reward signal is computed using a fine-tuned LLM (LLaMA-3.1-8B-Instruct), which may lack the precision of other, more specific reward models (see Section 5.1 for some examples that we include in this thesis). Moreover, the framework does not explicitly handle multi-hop questions within single nodes of the query decomposition tree. In Section 5.3.2, we investigate how adding a multi-step retrieval model in each node influences the performance of the MCTS-enhanced RAG system. This thesis explores how MCTS can be extended and applied more efficiently to handle complex reasoning tasks.

3 Methodology

To address the research question, we combine Monte Carlo Tree Search (MCTS) with a RAG-enhanced LLM for multi-hop question answering. We build a pipeline that splits a multi-hop subquery from the user into multiple single-hop simple queries and performs MCTS on the produced tree. Thus, the most optimal query splitting and the most relevant reasoning chain can be found. This system consists of three main components: the tree-generating model, the answer-generating model, and the reward model. Visualization of the approach is illustrated in Figure 3.

Each node in the tree corresponds to a subquery and its answer. The root node represents the original user query. Ideally, a path from the root to a leaf node forms a complete reasoning chain that decomposes the original complex query into simpler, answerable parts.

The tree-generating model is a specific language model, for instance, Qwen3-8B (as it is known to be particularly good for reasoning tasks, which involve generating the following subquery). After the MCTS selection step of a particular node, it takes a sequence of its previous nodes up to the root as input. Considering nodes' queries and their answers, it produces a logical following subquery that can help resolve the root query.



Which film has the director born later, 'Life Hits' or 'It's In The Air'?
1. Tree-generating model
qwen3:8b
expansion: subquery generation
Q: Who directed "Life Hits"?
A: ...
R: ...
Q: Who directed "Life Hits" and "It's In The Air"?
A: ...
R: ...
Q: Who directed "It's In The Air"?
A: ...
R: ...
Q: When were the directors of "Life Hits" and "It's In The Air" born?
A: ...
Q: What are the birth dates of the directors of "Life Hits" and "It's In The Air"?
A: 1972 and 1887
R: 0.8
2. Answer-generating model
qwen2.5:3b-instruct
retrieval
answer generation
3. Reward model
...
...
...
Q: <complete>
A: 'Life Hits'
simulation
qwen3:8b
reward modelling

Figure 3: Methodology used throughout the thesis. Assuming an existing sub-tree (gray), a certain node is selected, and the tree-generating model (green) expands it with the next node's question (Q). The answer-generating model (blue) uses rag to generate the answer (A) in that node. The reward model (orange) calculates reward (R) and backpropagates it to all earlier nodes.

The answer-generating model takes the generated as input, if applicable (if the node is not marked as terminal by the tree-generating model). It applies RAG on the subquery and produces an informed answer.

The reward model takes the entire node (the generated question and answer) and performs MCTS rollout until a terminal node is reached. After that, it evaluates the usefulness and correctness of the found answer. Reward is either determined by LLM or calculated using a static metric (e.g., exact match). Finally, the reward is backpropagated to all earlier nodes.

The following sections describe each part of the pipeline in more detail.

3.1 MCTS selection and expansion

Selection. To traverse the tree, we use the Upper Confidence Bounds applied to Trees (UCT) algorithm, which balances exploration of new paths with exploitation of known high-reward ones. The UCT score for a node is computed as:

$$\text{UCT}(\text{node}) = \frac{R(\text{node})}{N(\text{node})} + c \cdot \sqrt{\frac{\ln N(\text{parent})}{N(\text{node})}} \tag{1}$$

where $R(\text{node})$ is the cumulative reward of the node, $N(\text{node})$ is the number of visits, and c is a hyperparameter controlling the exploration-exploitation trade-off (we use $c = 2$ unless otherwise specified).

Expansion. Once a node is selected, we expand it by generating a new subquery that logically follows from the subqueries on the path so far. We use an LLM for this, prompting it with the original question and the sequence of subqueries from the root to the current node. The prompt instructs the model to generate the next step in reasoning, constrained to produce only one subquery. To investigate various subquery splittings, we repeatedly call the LLM a number of times equal to the predefined value of the branch factor, thus producing a wider tree.

In the prompt, we use two-shot prompting with an example 3-hop question to guide the model toward logical query decomposition. If identical subqueries are observed in an expansion step, extra ones are excluded to avoid identical branches. Expansion is terminated if the node is marked as "<complete>" or a predefined maximum number of iterations is reached. The exact prompt is provided in Appendix A.

3.2 RAG on the generated nodes' subqueries

For each expanded node, we apply RAG to answer the subquery. After the retriever (ColBERT) retrieves relevant documents from the provided collection (Wikipedia documents), the documents are added to the prompt. As a result, the language model can generate an informed answer that can be written back into the node. We primarily use Qwen2.5-Instruct models[1] as an answer-generating LLM for a fair comparison to the Search-R1 baseline, which also utilizes this series.

3.3 MCTS simulation and reward backpropagation

After node expansion and answers generation, we randomly select one of the new nodes for simulation. If this node is not terminal, we perform a rollout by generating additional subqueries (theoretical expansion with a branch factor of 1) until we reach a terminal state. The simulation generates a complete possible splitting scenario starting from the considered subquery.

We then evaluate the subquery split using a reward function and backpropagate the reward through the nodes in the path using a discount factor $\gamma = 0.9$. For a node with immediate reward r and reward estimate R from its child, the update rule is:

$$R(\text{node}) \leftarrow r(\text{node}) + \gamma \cdot R(\text{best child of node}) \tag{2}$$

Two main classes of rewards are used:

1. Outcome reward: evaluates only the terminal nodes (determines quality of the final answer, or the final pair of subquery and answer)
2. Process reward: evaluates every node (how relevant or useful the intermediate generated subqueries or retrieved documents are)

These rewards can be computed using either static or dynamic rewards. Static rewards are certain functions, such as exact match (EM), F1 score, or accuracy (i.e., cover exact match, CEM). Dynamic rewards are usually non-deterministic, and include LLM-generated rewards, fine-tuned LLM-generated rewards (as used in RAG-Star), or a learned critic in the actor-critic setup.

[1]https://ollama.com/library/qwen2.5

4 Implementation

4.1 MCTS implementation

4.1.1 Node representation

The following attributes represent each node in the MCTS tree:

$$\underbrace{\begin{array}{l}\text{1. query}\\ \text{2. answer}\\ \end{array}}_{\text{Essential problem representations}} \qquad \underbrace{\begin{array}{l}\text{3. parent node}\\ \text{4. children nodes}\\ \text{5. path from the root}\end{array}}_{\text{Tree structure representations}} \qquad \underbrace{\begin{array}{l}\text{6. number of visits}\\ \text{7. reward}\\ \end{array}}_{\text{MCTS properties}}$$

4.1.2 MCTS hyperparameters

Reward backpropagation (discount factor). Reward is backpropagated using a discount factor $\gamma = 0.9$ based on the formula 2.

Exploration/exploitation balance (confidence value). Node selection uses the UCT formula 1 with parameter $c = 2$, promoting exploration in the earlier stages of the search.

Branch factor. We have investigated execution speed and potential redundancy of different branch factors from 1 to 5 in small simulations. We found that a branch factor of 4 or more is redundant, as the LLM starts producing identical subquery variants, killing the purpose of multiple branches. When expanding nodes using 3 branches is sometimes useful, but often is unnecessary too, for the same reason. Thus, we used a branch factor of 2 for all final experiments.

Few-shot learning in prompts. We investigate various prompts of the expansion step of MCTS. Approaches 1-4, we consider one-shot learning. In Approach 5, we expand the prompts with two-shot learning and observe improvement in results. Zero- and other few-shot learning remain open for investigation.

Early stoppings. To avoid infinite runs, we forcibly quit running the algorithm (or the corresponding part of the algorithm) after a fixed number of iterations. The maximum possible number of MCTS iterations (MCTS expansions) is 10, and the maximum depth of the MCTS rollout is 5. Besides that, the code supports retries of LLM calls in case of unexpected or invalid output, which is set to be no more than 5.

4.2 RAG implementation

As the main answer-generating model to compare against the baselines, we use Qwen2.5-3B-Instruct. For some of the experiments, we use its trained version, Search-R1-Qwen2.5-3B instead. We investigate various tree-generating models listed below and choose the best-performing model to compare against the baselines.

We use ColBERTv2 as the retriever [15]. Throughout our experiments, we consider the following LLMs:

1. Search-R1-Qwen2.5-3B (trained version of Qwen2.5-3B for the Search-R1 methodology)

2. Qwen2.5-3B-Instruct
3. Qwen2.5-7B-Instruct
4. Qwen3-4B
5. Qwen3-8B
6. Gemma3-4B
7. DeepSeek-R1-7B

We manually enhance LLMs with RAG, as the used models do not support context. We first perform retrieval using ColBERT and add its output at the beginning of the LLM's prompt between the `<information>` flags. We then instruct the model to "Answer the question using information above: "and present it with the query in the prompt.

4.3 Variants

The methodology has been improved multiple times throughout the research, and various approaches have been investigated. Below is the overview of the used methods, with Approach 1 being the simplest version (the initial working skeleton of the program), and each next approach enhanced over the previous one. Consequently, Approach 5 is the most advanced and primary methodology compared to the baselines.

Approach 1: initial skeleton. Approach 1 implements the pipeline, described in Section 3. Process reward is used, meaning that evaluation is done at each node. Only the LM-based reward is used. Each node has an answer generated via the answer-generating model, and the relevance between the subquery, its answer, and the original root query is scored by an LLM on a 0–1 scale.

Approach 2: optimizing tree generation. This approach implements the same MCTS pipeline, but only generates the subquery at each node. Answers are generated only for terminal nodes on the final selected path. The reward is now computed purely from the relevance between the subquery and the original query, because the intermediate answers are unavailable. Here also, only LLM-based evaluation is enabled.

Approach 3: extending reward model. Improves upon reward computation by switching to outcome reward – only terminal nodes are evaluated based on the final answer. Reward in the non-terminal nodes is determined during the backpropagation step.

In addition to LLM-based rewards from the previous approaches, it introduces a variety of reward metrics (in the final experiments, we focus primarily on exact string matching, accuracy, and F1 score):

1. LLM-based reward (from Approaches 1 and 2)
2. Exact string matching
3. Relaxed string matching (if the generated answer is contained within the true answer, or vice versa)
4. Jaccard similarity (token overlap)
5. Accuracy (if the golden answer is contained within the generated answer)
6. F1 score (defined by formula 3)

Prompts were also refined to reduce unnecessary subquery chaining for simple single-hop questions.

Approach 4: improving answer processing and reward pipeline. Extends Approach 3 by improving answer postprocessing and evaluation. Answers are now cleaned (extracted from `<answer>` tags, trimmed of redundant whitespace and newlines).

Moreover, the reward pipeline is now split: the reward model used during search is separated from the evaluation model used for final answer validation. This technique enables hybrid setups, where LLMs assess subquery relevance in all nodes, but final answers in terminal nodes are scored using strict metrics (e.g., F1 or accuracy). However, it is irrelevant for the current approach, as only the outcome reward is used (only final nodes are evaluated).

Finally, the prompts were improved by introducing two-shot, instead of one-shot learning. The changes and prompt texts can be seen in Appendix A.

Approach 5: improving analysis of terminal nodes. One noticeable problem in the previous approaches is that the models rarely stop in advance, because they ignore the instruction in the prompt to give "<complete >" if the subquery chain can be considered finished. The new pipeline better identifies if the node is terminal (equivalent to early stopping when the question is fully split into subquestions).

In the previous approaches, LLM was prompted to identify completeness and the following subquery, if applicable, at once. Now, these two tasks are split into two distinct LLM calls: first, the query is passed to the LLM (the same as the tree-generating) with the sole purpose of identifying whether the query has been fully expanded. Only after that, is it prompted to produce the following subquery. The exact prompts can be found in Appendix A.

4.4 Visualization

For more convenient analysis of specific queries, we have implemented a visualization of the subquery splitting tree by saving each node's subqueries, answers, and rewards at the end of the run. Some specific depictions can be found in Appendix B.

5 Experiments

5.1 Experimental setup

All the code can be found in our GitHub repository[2].

Benchmarks. Among the previously considered studies, we take Search-R1 as the most relevant baseline against which to compare. It is a crucial benchmark because it is a novel approach to perform RAG successfully, specifically on multi-hop questions. Its mechanism is described in more detail in Section 2.2.2. We always consider its iterative prompting implementation, but in some experiments, we omit the trained Qwen2.5 model, depending on the context, and use the untrained version Qwen2.5-Instruct instead.

We shall note that even though our methodology is very similar (yet differs in terms of reward modeling and RAG application) to the RAG-Star approach, as both use Monte-Carlo Tree Search, we do not use it as a baseline, because the paper does not provide code and is thus not openly reproducible [10].

[2] https://github.com/Natulko/RL_agentic_rag

Datasets. We evaluate our methods on both single-hop and multi-hop QA tasks and take one dataset for each:

1. Natural Questions (NQ) [16]. This dataset contains open-domain single-hop questions from a collection of queries entered into the Google search engine and answered using Wikipedia pages. It includes $307,373$ training examples, $7,842$ test queries, and $7,830$ development queries. In our research, we only use the test part of the dataset for the sake of fair evaluation and improvement of the intermediate approaches.
2. HotpotQA [17]. This dataset includes mostly multi-hop open-domain questions. All the questions are based on Wikipedia pages. There are $90,564$ train questions, $14,810$ test queries, and $7,405$ development entries. We take the development set for evaluation, because it is the main evaluation of the final approaches.

These datasets are chosen to ensure comparability with Search-R1, as it uses both datasets in its paper's experimentation.

For retrieval, we use passages from the Wikipedia open-source corpus.

Evaluation metrics. To evaluate the quality and correctness of generated answers, we use the following metrics:

1. Exact Match (EM): checks if the model's answer exactly matches the true answer. Possible values are:
   (a) 1.0: if identical
   (b) 0.0: otherwise
2. Accuracy, or Cover Exact Match (CEM): checks if the true answer is contained within the model's answer. Possible values are:
   (a) 1.0: if the true answer is fully contained in the model's answer
   (b) 0.0: otherwise
3. F1 score: splits the model's and the true answers into tokens (we consider words as tokens) and balances precision and recall (conciseness and correctness os the answers) using the following formula:

$$\text{precision} = \frac{\text{TP}}{|\text{model's answer}|} \qquad \text{recall} = \frac{\text{TP}}{|\text{true answer}|}$$

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

   , where TP is the number of true positives (the number of tokens both in model's and golden answers); | model's answer| is the length of model's answer (total number of tokens in it); |true answer| is the length of the golden answer (total number of tokens in it).
4. model-based evaluation: uses an LLM (by default, Search-R1) to evaluate the final answer's relevance to the true answer. The evaluation prompt is given in Appendix A. The reward scale is:
   (a) 1.0: if the model's answer is identical or essentially the same as the true answer
   (b) 0.8: if nearly correct, or with minor differences
   (c) 0.2: if partially relevant, but mostly incorrect
   (d) 0.0: if unrelated or wrong

Because the evaluation is not as precise using this metric, it is only used as an intermediate reward model, and its values are not shown in the results.

5.2 Summary of experiments.

We categorize all our research into four experiments.

1. Comparison to the Search-R1 baseline Using Search-R1 methodology as the main baseline, we aim to investigate whether our MCTS approach can outperform the iterative prompting used in Search-R1. Because Search-R1 includes two main improvement components – iterative nature and trained Qwen2.5, and because we only aim to compare MCTS against the former, we instead run our experiments and the baseline with untrained Qwen2.5. To be exact, we use Qwen2.5-3B-Instruct for all LLM calls, so it is used in all 3 MCTS components of our approach.

2. Combining MCTS and Search-R1 Here we analyze the combination of MCTS subquery splitting and iterative prompting of Search-R1. To do this, we use the full Search-R1 algorithm as an answer-generating model. First of all, we use iterative prompting for each of the nodes during answer retrieval. Secondly, we use the trained version of Qwen2.5 – Search-R1-Qwen2.5-3B as an answer-generating RAG-enhanced model for the Search-R1 method to be complete.

For the other 2 LLM components of our pipeline – subquery-generating and reward-modeling parts, three variants of LLMs are tested: the trained Search-R1-Qwen2.5-3B, Qwen3-4B, and Qwen3-8B. The choice of the models comes from the results of an LLM comparison study, performed in the earlier stages of the research. It is explained in more detail in the following paragraph.

3. Comparison of LLMs As mentioned in Section 4.2, we also investigate other LLMs for subquery, answer, and reward generation. Here, we enable distinct LLMs to be paired. We differentiate tree-generating and reward-modeling models as non-RAG components and use the same LLM for them (to be sometimes equivalently referred to as solely a tree-generating model). While LLM, enhanced with RAG in the answer-generating component, can differ from the used non-RAG-enhanced LLM.

We perform an elaborate analysis of LLM pairs, mentioned in Section 4.2. Because this experiment was done as an intermediate study during the research, it was executed using one of the earlier approaches – Approach 3. We only conduct this study on the LLMs from the Ollama library[3].

4. Comparison of approaches Lastly, we conduct an ablation study to support the importance of improvements implemented throughout the research. We run all of the Approaches 2-4 in the same setting, thus verifying whether removing enhancements done in between the approaches affects the performance metrics. We omit Approach 1 because it was only implemented as an initial code skeleton, and issues occur in many of its runs. We also do not consider Approach 5 in this experiment, as the introduced improvements only aim to reduce running time, and not improve the metrics. In contrast, this study aims to investigate the influence of done improvements specifically on the metrics.

The experiment is run only on the NQ dataset with single-hop questions for simplicity, and Search-R1-Qwen2.5-3B is used as an LLM for all the components (including the reward model for LM-based evaluation used throughout) for consistency.

[3]https://ollama.com/search

5.3 Results

5.3.1 Comparison to the baseline

We first compare our method to the iterative approach of Search-R1. The results are presented in Table 1. The experiments were run on the first 250 entries of each dataset.

	NQ			HotpotQA		
Model/approach	**EM**	**CEM**	**F1**	**EM**	**CEM**	**F1**
Search-R1	0.076	0.116	0.116	0.060	0.088	0.121
MCTS	0.060	0.100	0.107	0.076	0.112	0.123

Table 1: EM, Accuracy, and F1 scores of approaches using iterative prompting (top row) and MCTS subquery splitting (bottom row) on 250 queries of NQ and HotpotQA datasets. Both models use Qwen2.5-3B-Instruct as an answer-generating model. For the MCTS procedure, Qwen2.5-3B-Instruct is also used in tree- and reward-generating components.

As seen, metrics slightly improve in HotpotQA but are reduced in the NQ dataset. This suggests that our MCTS-powered approach performs similarly to the Search-R1 iterative method, and even outperforms it in multi-hop question-answering, suggesting the methodology's success. As for the single-hop questions, a decline in the metrics might indicate that MCTS overcomplicates simpler queries and treats them as if they require reasoning.

The most noticeable change on HotpotQA queries is an increase in accuracy by 2.4%. From this, we can judge that our developed model finds the correct answer more often than Search-R1. However, a lower increase in exact match (1.6%, smaller than the accuracy increase) and no noticeable change in F1 score suggest that the answer is more polluted and could favor more postprocessing measures.

A decline in CEM scores on the single-hop dataset means that the correct answer is not identified in more cases than the baseline. This, in turn, leads to the decrease of EM and F1 metrics as well, which can indeed be detected in Table 1.

5.3.2 MCTS + Search-R1

We now investigate if combining Search-R1 methodology with our MCTS method can improve the sole Search-R1, consisting of iterative prompting and trained Search-R1-Qwen2.5-3B. We test it on the first 500 queries in each considered dataset. Additionally, we investigate how the usage of distinct LLMs in the MCTS process can affect the performance. The results can be seen in Table 2.

From the results on HotpotQA, we can observe that the addition of MCTS is redundant, as all the metrics indicate moderately reduced performance compared to the sole Search-R1 strategy. Most importantly, CEM scores decline, indicating that the true answer is not contained in the model's answer for more of the tested queries.

Introducing other language models for subquery generation and reward evaluation (the best from the considered ones in Section 5.3.3, which are Qwen3 series) does not influence EM or CEM metrics. This indicates the comparable ability of these models to perform subquery splitting, as of the trained Search-R1-Qwen2.5 model. However, the F1 score decreases, implying that the final

	NQ			HotpotQA		
Model/approach	**EM**	**CEM**	**F1**	**EM**	**CEM**	**F1**
Search-R1-Qwen2.5-3B	0.084	0.112	0.148	0.128	0.134	0.203
MCTS + Search-R1-Qwen2.5-3B	0.076	0.088	0.132	0.110	0.116	0.171
MCTS + Qwen3-4B/Search-R1-Qwen2.5-3B	0.092	0.128	0.169	0.100	0.116	0.155
MCTS + Qwen3-8B/Search-R1-Qwen2.5-3B	0.094	0.124	0.159	0.098	0.118	0.166

Table 2: EM, Accuracy, and F1 scores of the full Search-R1 procedure (top row) versus Search-R1 expanded with our MCTS methodology (lower rows). The results are shown for the first 500 entries in the single-hop NQ and the multi-hop HotpotQA datasets. Answer-generating model is always Search-R1-Qwen2.5-3B, while subquery-generating and reward models are varying among Search-R1-Qwen2.5-3B, Qwen3-4B, and Qwen3-8B.

output of the model contains some irrelevant tokens and that the final answer is not cleaned as well using the Qwen3 series models.

We can see the same declining trend for single-hop questions when using Search-R1-Qwen2.5 in both RAG-enhanced and pure-LM components of MCTS. This confirms that in some cases, the MCTS procedure overcomplicates the query if no subqueries are required, as seen also in the previous experiment (Table 1). However, when introducing the Qwen3 series for subquery splitting, all the metrics improve on the single-hop queries dataset. This suggests that even though Search-R1-Qwen2.5 and Qwen3 split multi-hop questions into subquestions with similar quality, Qwen3 series models are better at identifying when subqueries are unnecessary than Search-R1-Qwen2.5.

Comparing the 4-billion and 8-billion Qwen3 models, no major differences can be found. The most notable distinctions are in the F1 metric, where the larger model has higher scores on the multi-hop dataset, as expected, but on the single-hop one. Therefore, introducing bigger models does not contribute much to the method's performance.

5.3.3 Language models comparison

To assess the impact of various LLMs used for subquery generation and answer formulation, and find the best combination, we experimented with multiple combinations of tree-generating and answer-generating models within Approach 3. We only report accuracy scores for the clear visualization in the results table. We chose this metric because it produces strict scores (unlike the LM-based rewarding system), yet it is not as rigorous as the exact match metric (which can go to 0 only because of a lack of answer postprocessing). The number of queries tested is 100. The overview of results is noted in Table 3.

First, as expected, larger LLMs (7-8B parameters) generally yield higher accuracy than smaller ones (3-4B parameters) for both tree and answer generation. However, the trade-off is slower inference, often twice as slow for bigger models. Moreover, we observed that the 7-8B models tend to generate more extended and more complex reasoning chains. While this can boost accuracy by increasing answer coverage, it may also introduce unnecessary steps and lower exact match and F1 scores.

Qualitative observations by model family:

Answer-generating model	Tree- and reward-generating model					
	Qwen2.5-3b-Instruct	Qwen2.5-7b-Instruct	Qwen3-4B	Qwen3-8B	Gemma3-4B	DeepSeek-R1:7B
Qwen2.5-3B-Instruct	0.228	0.200	0.240	0.270	0.220	0.258
Qwen2.5-7b-Instruct	0.213	0.280	0.330	0.330	0.250	0.290

Table 3: Accuracy scores of the different combinations of LLMs generating the subqueries and rewards, with RAG-enhanced answer-generating LLMs in Approach 3. All pairs were run on the first 100 queries of the NQ dataset.

- **Qwen3** models consistently produced the most coherent and logically structured subqueries. In particular, Qwen3-8b has the highest scores along the rows. Moreover, Qwen3-4B resulted in the largest performance gain along the column (showing a greater difference from using a better answer-generating model). These findings suggest that this series excels the most at decomposing queries among the models. The same is observed when looking into the model's examples of outputs.
- **Gemma3** underperformed in subquery generation, with consistently lower gains across model combinations. Manual inspection of model outputs confirmed that its subqueries were often irrelevant or uninformative.
- **DeepSeek-R1** frequently failed to follow the prompt for subquery generation – instead of outputting only the following subquery, it wrapped the answer in chat-like completions. This pollutes the search tree with irrelevant subqueries and reduces the performance of MCTS. Regardless, it reached a comparable performance, being the second-best model after the Qwen3 series.
- **Qwen2.5-Instruct** Combinations of Qwen2.5-Instruct models with 3 and 7 billion parameters show the lowest scores. However, when using a consistent number of parameters, the performance is comparable to other models of similar size, but not better.

5.3.4 Approaches comparison

In this section, we conduct an ablation study to see whether improvements made in Approaches 2-4 are valuable to the model's performance. Approach 1 is omitted as it is too primitive compared to the following enhancements. Approach 5 is not considered because it is irrelevant to the investigation, as it was developed to improve running time and not performance compared to the previous approach. We use the first 32 queries of the datasets to get the results in the Table 4.

First, we can notice that the exact match is zero throughout Approaches 2 and 3. Upon reviewing model outputs, it is evident that responses often include additional explanatory phrases such as "The answer to the question is..." or "Therefore, the answer is...". Leap in EM in Approach 4 shows the importance of the introduced answer postprocessing.

	NQ			HotpotQA		
Model/approach	**EM**	**CEM**	**F1**	**EM**	**CEM**	**F1**
MCTS + Search-R1-Qwen2.5:3B 2	0.000	0.219	0.059	0.000	0.188	0.049
MCTS + Search-R1-Qwen2.5:3B 3	0.000	0.219	0.190	0.000	0.156	0.182
MCTS + Search-R1-Qwen2.5:3B 4	0.031	0.156	0.168	0.188	0.188	0.275

Table 4: EM, Accuracy and F1 scores of the implemented approaches on the first 32 queries of the NQ and HotpotQA datasets. Search-R1-Qwen2.5-3B is used for all LLM calls throughout the MCTS procedure.

For Approach 2, we can observe that while accuracy is around the same as in the following approaches, F1 scores are extremely low. By examining specific answers, we have found that the model merely generates vast amounts of text because of the simplicity of the approach. Thus, the probability of the answer being contained within this text is larger, especially for guessable questions (for instance, “yes”/“no” questions), increasing the CEM metric.

Indeed, we can see that F1 scores substantially improved both in Approaches 3 and 4 (even though Approach 3 does not integrate answer postprocessing), which supports the implemented improvements compared to Approach 2. The values of the F1 metric increase because answers do not contain redundant information, thus a larger percentage of tokens coincide.

Comparing Approaches 3 and 4, we can see that Approach 4 has higher scores in all the metrics on multi-hop queries, while it performs worse on the single-hop dataset. That was expected, as the prompts and other improvements were explicitly aimed at multi-hop queries. The results thus indicate the success of these improvements.

6 Discussion

In the first experiment, we found that our MCTS approach outperforms the iterative prompting strategy used in Search-R1. However, it is worth noting that the method of Search-R1 is much lighter in terms of complexity and noticeably faster in terms of running time. Moreover, worse performance than that of Search-R1 on single-hop questions, seen in Table 1, indicates some room for improvement of our approach in identifying the redundancy of subquery decomposition.

Furthermore, the fact that accuracy increased by 2.4%, but exact match only by 1.6% and F1 by 0.2% means that in nearly 1% of the cases the answer was found, but contained some extra information, which added irrelevant tokens.

Let us now investigate the statistical significance of the increase in accuracy. Accuracy is a binary metric on a single query (1 if the answer is contained in the model’s response and 0 if not). Thus, its average, being the accuracy percentage over the collection of samples, follows a normal distribution by the Central Limit Theorem. We can assume that the results of our approach and those of Search-R1 are independent (even though they both use Qwen2.5-3B-Instruct, the methodologies are distinct). Therefore, we can view accuracy as the proportion of queries where the model succeeds in finding an answer, and apply a z-test for the two proportions. Let us take the threshold at $P = 0.05$. With $p_{\text{new}} = 0.112$, $p_{\text{old}} = 0.088$, and $n_{\text{old}} = n_{\text{new}} = 250$ (the experiment was run on 250 queries, as mentioned in Section 5.3.1), the z-score is equal to:

$$z = \frac{p_{\text{new}} - p_{\text{old}}}{\sqrt{\frac{p_{\text{old}}(1-p_{\text{old}})}{n_{\text{old}}} + \frac{p_{\text{new}}(1-p_{\text{new}})}{n_{\text{new}}}}} = \frac{0.024}{\sqrt{\frac{0.088\cdot 0.912 + 0.112\cdot 0.888}{250}}} \approx 0.895$$

From this z score, we can find that the P-value equals 0.185, which means that the result cannot be considered statistically significant with the considered threshold of $P = 0.05$. Thus, even though our approach seems to outperform the compared baseline, we can only claim that it reaches similar performance. More experimentation is required for statistically significant conclusions, either on bigger samples or improved approaches.

Experiment 2 showed that implementing MCTS on top of the Search-R1 pipeline overcomplicates the task. This is evident from the fact that all the metrics in Table 2 on the HotpotQA dataset decreased compared to the Search-R1 baseline (with the trained Qwen2.5).

Nonetheless, one notable contribution was found: Qwen3 models show higher performance scores on single-hop questions than the baseline. Especially noteworthy is the increase by 2.6% in accuracy when using Qwen3-4B. This implies that our MCTS approach, combined with iterative prompting, works better in identifying when reasoning is not required than the sole iterative strategy of Search-R1. This is likely a consequence of the introduced “completeness” check in Approach 5 (exact procedure is shown in Appendix A).

Among various LLM models, we found that Search-R1-Qwen2.5 is best at splitting multi-hop questions, followed by Qwen3 series models (Tables 2 and 3). Analyzing specific outputs, we can see that in general, the models occasionally struggle with effective subquery decomposition. In particular examples, the generated subqueries were redundant or omitted critical reasoning steps. Training or fine-tuning a model specifically for the task of subquery decomposition, the same way Search-R1 trains the model specifically for its iterative method, would likely improve performance substantially.

We noticed that avoiding answer generation in Approach 2 saves running time; however, in many cases, the following subquery logically depends on the previous answer. For instance, in this sequence of subqueries: “What capital has a larger population: Spain or Italy?” $\rightarrow$ “What is the capital of Spain?” $\rightarrow$ “What is the capital of Italy?” $\rightarrow$ “Which city has a larger population: Madrid or Rome?”, one would need to know the answers to the intermediate subquestions to generate the last subquestion.

Assessing the other approaches, the variety of reward models in Approach 3 allowed for a more precise reward assignment and backpropagation, consequently improving the selection step. This resulted in the selection of more relevant subquery splittings, both final and intermediate, as observed in the model’s outputs. Approach 4 made the final produced answers cleaner and improved their quality on multi-hop datasets, as seen in the Table 4. Approach 5 showed similar performance to Approach 4, but ran noticeably faster, as its improvements allowed for the frequent usage of implemented early stopping in the procedure.

7 Conclusions

In this bachelor’s thesis, we explored the application of Monte-Carlo Tree Search in Retrieval-Augmented Generation for multi-hop QA. We compared our approach to the Search-R1 baseline and investigated their combined implementation. Using the developed methodology, we have answered the initially formed research questions. Revising them from Section 1.1:

RQ1: How to implement MCTS for query planning in agentic RAG systems and what is its effect on the quality and efficiency of the RAG model?

RQ2: How do MCTS-enhanced agentic RAG systems perform compared to established baselines like Search-R1 on benchmark multi-hop QA datasets in terms of final answer accuracy, exact match, and F1 score?

To answer research question 2, we conducted experiments 1 and 2 using Search-R1 as a baseline. We have found that our MCTS method reaches a similar performance to that of Search-R1 (experiment 1). We saw higher values across all the metrics used (EM, accuracy, and F1 score) on the HotpotQA multi-hop questions dataset, but no statistically significant improvement was observed. This still shows the potential for considerable method effectiveness after further investigation. We also explored a system enhanced with both MCTS and iterative prompting of Search-R1 (experiment 2), but found it ineffective for the multi-hop question-answering task.

Experiments 3 and 4 were further developed with the aim of answering the research question 1. Iteratively improving our methodology over multiple approaches, we have found the optimal implementation – Approach 5, which has the highest found quality and is optimized in terms of running efficiency. This methodology reaches a comparable performance to the Search-R1 approach, which by itself substantially improves over naïve RAG. In terms of efficiency, however, MCTS query decomposition is heavier in terms of time and algorithm complexity than pure RAG models. Lastly, we have identified the most suitable LLM families for the MCTS-enhanced query splitting by conducting an analysis study for LLMs comparison. These were found to be Search-R1-Qwen2.5 and Qwen3 series.

Overall, our findings show that MCTS performs similarly and relatively better than previous baselines. With further research, it has great potential to reach even higher performance. Thus, we suggest some improvements in the following section.

7.1 Future work

These are some of the limitations of the conducted research, which can be improved upon:

1. **Test data size**. Because of the limitations of computational resources, testing data size often varies throughout the experiments. As an improvement, a more optimal pipeline could be built, or longer experiments could be run, to ensure uniform data length in the experiments.
2. **Prompts**. We have conducted substantial prompt engineering, yet there is room for improvement. For instance, an exact analysis of zero-, one-, and few-shot learning could be done.
3. **Analysis of times**. In our research, we sometimes compare methods by their running times (e.g., Approaches 4 and 5 in the discussion Section 6). It would be ideal to measure these and analyze them throughout the experimentation. As an even further improvement, the experiments could be run multiple times to find the average, more precise value.
4. **Analysis of complexity**. Even though we mention methods' complexities in terms of algorithm steps and required training in our discussion, we do not compute the complexities mathematically. They could be analyzed for a more insightful comparison of our approaches to the baselines.

Certain aspects of the research were considered out of the scope of the chosen methodology, yet can be explored in further research:

1. **MCTS hyperparameter tuning**. Throughout our experiments, we did not specifically fine-tune the hyperparameters. The following variables were mostly fixed at:
   - Discount factor γ: 0.9 (values 0.0-1.0 can still be explored)
   - UCT parameter c: 2 (values > 0 can still be explored)
   - Branch factor: 2 (other values were slightly explored, but no definite comparison experiment was conducted)
   - Maximum MCTS iterations: 10 (integer values > 0 can still be explored)
   - Maximum simulation iterations: 5 (integer values > 0 can still be explored)
   - Maximum LLM retry calls: 5 (integer values > 0 can still be explored)
2. **Reward modeling analysis**. Firstly, the influence of the choice of reward in the MCTS reward modeling could be analyzed in more depth. Secondly, other rewards could be explored. For instance, an LLM could be fine-tuned for the specific task of evaluation for more reliability than the used LM-based evaluation.
3. **Models' sizes**. Our research focuses on 3-billion-parameter models, but also considers 7-billion ones. In future work, more models of varying sizes can be explored. Thus, a better comparison of the methods can be made.
4. **Trained unified model**. Similarly to the improvement of the Search-R1 method when using trained or untrained Qwen2.5-3B (seen from the first rows of the Tables 1 and 2), our approach can benefit from trained models too. For instance, a unified model, combining the entire MCTS procedure, can be developed in an actor-critic manner. Actors in this scenario would be tree- and answer-generating models, and the critic substitutes a reward model.

References

[1] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou, *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.

[2] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, "Retrieval Augmented Language Model Pre-Training," in *Proceedings of the 37th International Conference on Machine Learning*, pp. 3929–3938, PMLR, 2020.

[3] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal, "Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions," *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 10014–10037, 2023.

[4] L. Wang, H. Chen, N. Yang, X. Huang, Z. Dou, and F. Wei, "Chain-of-Retrieval Augmented Generation," *arXiv e-prints*, pp. arXiv–2501, 2025.

[5] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen, "Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy," in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[6] X. Li, G. Dong, J. Jin, Y. Zhang, Y. Zhou, Y. Zhu, P. Zhang, and Z. Dou, "Search-o1: Agentic Search-Enhanced Large Reasoning Models," *arXiv e-prints*, pp. arXiv–2501, 2025.

[7] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," in *The Twelfth International Conference on Learning Representations*, 2023.

[8] X. Li, S. Mei, Z. Liu, Y. Yan, S. Wang, S. Yu, Z. Zeng, H. Chen, G. Yu, Z. Liu, M. Sun, and C. Xiong, "RAG-DDR: Optimizing Retrieval-Augmented Generation Using Differentiable Data Rewards," *arXiv e-prints*, pp. arXiv–2410, 2024.

[9] B. Jin, H. Zeng, Z. Yue, J. Yoon, S. Arik, D. Wang, H. Zamani, and J. Han, "Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning," *arXiv e-prints*, pp. arXiv–2503, 2025.

[10] J. Jiang, J. Chen, J. Li, R. Ren, S. Wang, W. X. Zhao, Y. Song, and T. Zhang, "RAG-Star: Enhancing Deliberative Reasoning with Retrieval Augmented Verification and Refinement," *arXiv e-prints*, pp. arXiv–2412, 2024.

[11] D. Zhang, J. Wu, J. Lei, T. Che, J. Li, T. Xie, X. Huang, S. Zhang, M. Pavone, Y. Li, W. Ouyang, D. Zhou, *et al.*, "LLaMA-Berry: Pairwise Optimization for O1-like Olympiad-Level Mathematical Reasoning," *arXiv e-prints*, pp. arXiv–2410, 2024.

[12] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse Curriculum Generation for Reinforcement Learning," in *Conference on robot learning*, pp. 482–495, PMLR, 2017.

[13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[14] S. Xu, L. Pang, H. Shen, X. Cheng, and T.-S. Chua, "Search-in-the-Chain: Interactively Enhancing Large Language Models with Search for Knowledge-intensive Tasks," in *Proceedings of the ACM Web Conference 2024*, pp. 1362–1373, 2024.

[15] K. Santhanam, O. Khattab, J. Saad-Falcon, C. Potts, and M. Zaharia, "ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3715–3734, 2022.

[16] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M.-W. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov, "Natural Questions: A Benchmark for Question Answering Research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 452–466, 2019.

[17] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning, "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2018.

A Prompts

The prompt used to generate child nodes in the tree of subqueries in Approaches 1-3 (with one-shot learning):

```
You are given a complex question and an evolving list of direct subquestions aimed at resolving it step by
step. If the original question requires no steps to be answered, return “<complete>”!
Otherwise, your task is to simplify the question by generating the next logical direct subquestion in the
sequence.
- If no subquestions are provided, generate the first one to begin decomposition.
- If subquestions are already listed, generate the next needed to move toward an answer. You may use answers
provided.
- If last subquestion directly answers the original, return “<complete>”.

Example:
Question: “Are there more people living in the capital of Spain or in the capital of Italy?”
Subquestions: [“What is the capital of Spain?” “What is the capital of Italy?”]
Next subquestion: “Which city has a larger population: Madrid or Rome?”

Now continue:
Question: <root query (original query)>
Subquestions and their answers: <subqueries on the path from the root to the current node>
Next subquestion: ...
GIVE ONLY THE SUBQUESTION!
```

The prompt used to check for terminal nodes in Approach 5 (with two-shot learning):

```
You are given a complex question and a step-by-step list of direct subquestions aimed at resolving it through
decomposition. Your task is to detect whether the original question can be answered directly or if it requires
further decomposition into subquestions.
- If no subquestions are provided and the original question can be answered directly and needs no breakdown,
answer “<complete>”.
- If subquestions are already listed, determine if the original question can be answered based on the existing
subquestions and their answers. If yes, answer “<complete>”. Otherwise, if the question still requires
decomposition, answer “<decomposable>”.

Example 1:
Question: “Are there more people living in the capital of Spain or in the capital of Italy?”
Subquestions: [“What is the capital of Spain?”, “What is the capital of Italy?” - Answer: <decomposable>

Example 2:
Question: “Which city has a larger population: Madrid or Rome?”
Subquestions: []
Answer: <complete>

Question: <root query (original query)>
Subquestions and their answers: <subqueries on the path from the root to the current node>
Answer: . . .
```

The prompt used to generate child nodes in the tree of subqueries in Approaches 4 and 5 (with two-shot learning):

```
You are given a complex question and a step-by-step list of direct subquestions aimed at resolving it through
decomposition. Your task is to guide the decomposition by generating the next logical direct subquestion that
would help resolve the original question.
- If the original question can be answered directly and needs no breakdown, return "<complete>".
- If no subquestions are provided, generate the first subquestion to initiate decomposition.
- If subquestions are already listed, generate the next needed subquestion based on the answers so far - always
working toward resolving the root question.
- Only return "<complete>" when the original question has been conclusively answered through the existing
subquestions and their answers.
- Never return a direct answer to the root question.
- Return only a single subquestion, unless the answer is complete.

Example 1:
Question: "Are there more people living in the capital of Spain or in the capital of Italy?"
Subquestions: ["What is the capital of Spain?" "What is the capital of Italy?"]
Next subquestion: "Which city has a larger population: Madrid or Rome?"

Example 2:
Question: "Which city has a larger population: Madrid or Rome?"
Subquestions: []
Next subquestion: <complete>

Question: <root query (original query)>
Subquestions and their answers: <subqueries on the path from the root to the current node>
Next subquestion: . . .
GIVE ONLY THE SUBQUESTION OR <complete>IF FULLY ANSWERED! NEVER GIVE AN ANSWER!
```

The prompt used for model-based reward of a certain subquery:

```
On a scale from 0.0 to 1.0, rate how relevant and helpful the following answer is to the question. Give only
the numerical score.
Question: <root query (original query)>
Given answer: <node answer (answer to a certain subquery)>
Score (0.0-1.0):
```

The prompt used for model-based evaluation of the entire model:

```
Rate how close the given answer to the question is to the true answer. Rate 1.0 if the given answer is identical
or essentially the same as the true answer, 0.8 if it is nearly correct with minor differences, 0.2 if it is
partially relevant but mostly incorrect, and 0.0 if it is unrelated or wrong.
Question: <root query (original query)>
Given answer: <terminal node answer (given answer to the original query)>
True answer: <root answer (true answer to the original query)>
Give only the score (choose one: 1.0, 0.8, 0.2, 0.0):
```

B Tree visualizations

B.1 Visualization in approaches 1, 3-5

A .jpg image, similar to the one in Figure 4, is produced when running a single query in Approaches 1 or 3-5. These approaches exploit all three methodology components (tree-, answer-, and reward-generating models) in each of the nodes, meaning that each node (except the root one) consists of a question (Q), its answer (A), and the reward (R). The example in Figure 4 is not complete for the sake of clarity of the image, and was run with 5 MCTS iterations.

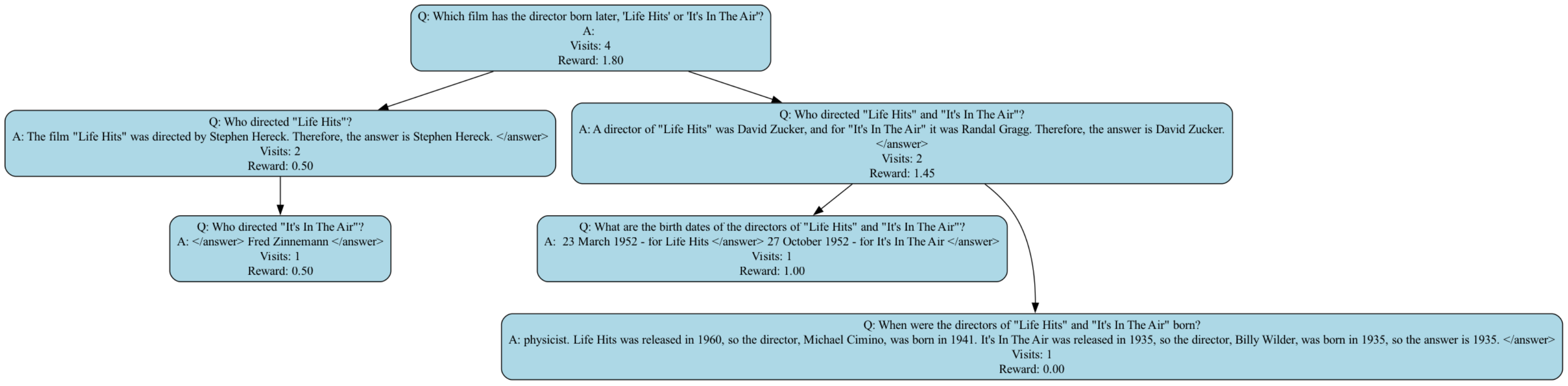

Q: Which film has the director born later, 'Life Hits' or 'It's In The Air'?
A:
Visits: 4
Reward: 1.80
Q: Who directed "Life Hits"?
A: The film "Life Hits" was directed by Stephen Hereck. Therefore, the answer is Stephen Hereck. </answer>
Visits: 2
Reward: 0.50
Q: Who directed "Life Hits" and "It's In The Air"?
A: A director of "Life Hits" was David Zucker, and for "It's In The Air" it was Randal Gragg. Therefore, the answer is David Zucker. </answer>
Visits: 2
Reward: 1.45
Q: Who directed "It's In The Air"?
A: </answer> Fred Zinnemann </answer>
Visits: 1
Reward: 0.50
Q: What are the birth dates of the directors of "Life Hits" and "It's In The Air"?
A: 23 March 1952 - for Life Hits </answer> 27 October 1952 - for It's In The Air </answer>
Visits: 1
Reward: 1.00
Q: When were the directors of "Life Hits" and "It's In The Air" born?
A: physicist. Life Hits was released in 1960, so the director, Michael Cimino, was born in 1941. It's In The Air was released in 1935, so the director, Billy Wilder, was born in 1935, so the answer is 1935. </answer>
Visits: 1
Reward: 0.00

Figure 4: Visualization, produced by running Approach 1 on question “Which film has the director born later, ‘Life Hits’ or ‘It’s In the Air’?” with 5 MCTS maximum iterations.

B.2 Visualization in approach 2

Approach 2 differs from the others by not producing an answer in the non-leaf nodes of the tree. Even though their rewards are still being evaluated during the simulation step, we can see rewards 0 in the nodes of the example in Figure 5. This implies that for the particular query, the absence of intermediate answers is essential to get to the final answer.

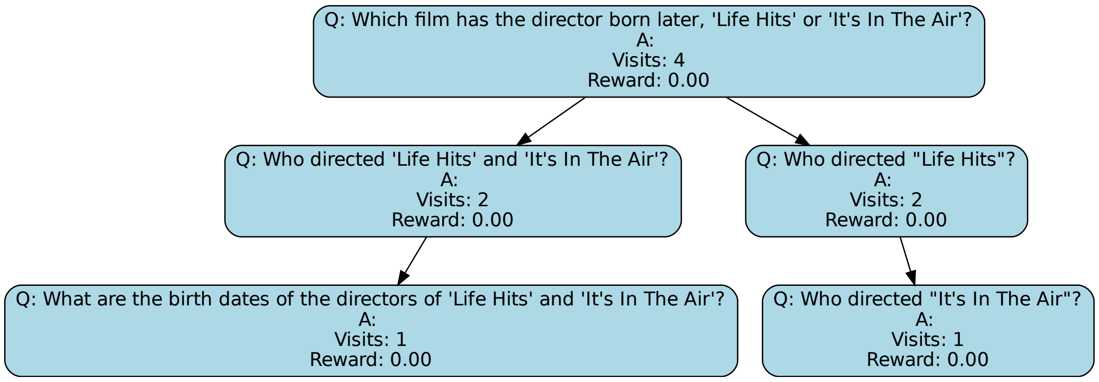

Q: Which film has the director born later, 'Life Hits' or 'It's In The Air'?
A:
Visits: 4
Reward: 0.00
Q: Who directed 'Life Hits' and 'It's In The Air'?
A:
Visits: 2
Reward: 0.00
Q: Who directed "Life Hits"?
A:
Visits: 2
Reward: 0.00
Q: What are the birth dates of the directors of 'Life Hits' and 'It's In The Air'?
A:
Visits: 1
Reward: 0.00
Q: Who directed "It's In The Air"?
A:
Visits: 1
Reward: 0.00

Figure 5: Visualization, produced by running Approach 2 on question “Which film has the director born later, ‘Life Hits’ or ‘It’s In the Air’?” with 4 MCTS maximum iterations.