When does Retrieval Work on Reasoning-Intense Tasks?

Quan Shi[1], Howard Yen[1], Mengzhou Xia[1],
Shunyu Yao[1,2], Danqi Chen[1], Karthik Narasimhan[1],
[1]Princeton University, [2]OpenAI,
Correspondence: benshi@princeton.edu

Abstract

Retrieval-Augmented Generation (RAG) is widely used in knowledge-intensive tasks, such as open-domain QA. However its effectiveness in reasoning-intensive tasks like mathematical problem-solving and code generation remains unclear. In this paper, we investigate the impact of incorporating oracle algorithmically similar documents into the context on performance in reasoning-intensive tasks. Our study spans five diverse code and mathematics tasks. Results vary dramatically across tasks, ranging from 66% performance degradation to 50% improvement. We further observe that RAG particularly helps in cases where high-level algorithmic decisions dominate model errors, but fails to fix algorithm implementation, or problem misunderstanding errors. Our results point to future research that focus on enhancing model utilization of retrieved documents and exploring retrieval methods for reasoning-intensive tasks.

1 Introduction

Retrieval-Augmented Generation (RAG) has emerged as a powerful paradigm for enhancing language models through external knowledge retrieval during inference, demonstrating remarkable success in knowledge-intensive tasks like open-domain question answering and fact verification (Guu et al., 2020; Lewis et al., 2020). Yet despite this success, RAG's effectiveness for reasoning-intensive tasks, particularly in coding and mathematical problem-solving, remains under-explored. Recent investigations into reasoning-based RAG have yielded contradictory results as the performance changes varies significantly across settings, models, and curated corpora (Wang et al., 2024b; Su et al., 2024; Shi et al., 2024). This raises a crucial question: Under what conditions can we expect retrieval to enhance model performance on reasoning tasks?

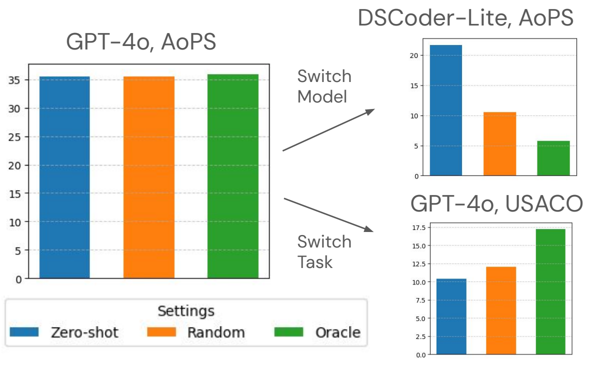


Figure 1: Under different models and tasks, the effect of RAG varies wildly. Tasks instances that benefit from retrieval largely benefit from fixing algorithm selection errors.

To address this question, we conduct a systematic evaluation of reasoning-based RAG's performance across 5 diverse coding and mathematical tasks, spanning various difficulty levels and problem-solving paradigms. We first curate sets of documents corresponding to each task instance based on oracle-selected algorithmic topic tags and human-curated matches like (Su et al., 2024), simulating a near-perfect reasoning-based retriever. Then, models are provided these *algorithmically optimal* documents to isolate the model's ability to leverage retrieved content, removing retrieval accuracy as a confounding factor. Our investigation show that performance impacts vary dramatically across tasks and models, ranging from 66% degradation to 50% improvement in relative performance. Notably, we discover that increasing the number of retrieved documents rarely improves performance, suggesting that SOTA models cannot leverage a large number of retrieved documents on reasoning-heavy tasks in contrast to their effectiveness on knowledge-intensive tasks (Wang et al., 2024a; Qi et al., 2019).

To understand these varying results, we con-

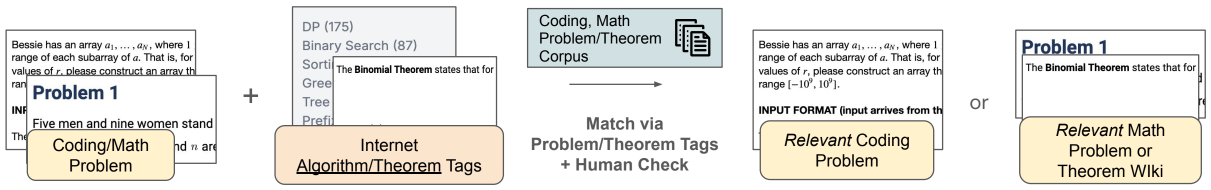

Problem 1
Coding/Math Problem
+
DP (175)
Binary Search (87)
The Binomial Theorem states that for
Internet Algorithm/Theorem Tags
Coding, Math Problem/Theorem Corpus
Match via Problem/Theorem Tags + Human Check
Relevant Coding Problem
or
Problem 1
Relevant Math Problem or Theorem WIki

Figure 2: Overview of oracle document collection process.

duct a qualitative analysis of model outputs, finding that most errors can be categorized as either failure to understand the problem, failure to select the correct algorithm/data structure to solve the problem, or failure to implement the selected algorithm/data structure. We find a distinct shift in error distribution between tasks where reasoning RAG proves beneficial versus those where it falls short. Successful cases of RAG-assisted problem solving predominantly occur when the original zero-shot errors stem from incorrect algorithm or data structure selection. This pattern suggests that current models primarily leverage retrieved documents at a high-level, algorithmic planning stage rather than for fine-grained implementation details or deeper reasoning steps. This suggests that future efforts to enhance downstream performance should prioritize improving models' ability to effectively utilize retrieved documents, rather than solely focusing on developing better retrieval mechanisms, which is what much contemporary work focuses on (Su et al., 2024; Wang et al., 2024b).

To summarize, our core contributions are:

- A curated dataset [1] of oracle documents matched to task instances by high overlap in algorithmic similarity.
- A comprehensive evaluation of oracle reasoning-based RAG across diverse coding and mathematical tasks, establishing approximate RAG upper bounds.
- A dataset of human annotated error mode analysis on model outputs, showing that current reasoning-based RAG systems primarily benefit high-level algorithmic planning rather than implementation details.

[1]See code + datasets here: `https://github.com/benshi34/LCBRetrieval`

2 Related Work

Retrieval Augmented Generation Retrieval Augmented Generation (RAG) (Guu et al., 2020; Lewis et al., 2020) has emerged as a significant approach to address the knowledge limitations of closed-domain language models (Ram et al., 2023). While its efficacy has been notable in knowledge-intensive tasks that require synthesis of knowledge over large knowledge bases (Asai et al., 2023; Zhou et al., 2022; Borgeaud et al., 2022; Khandelwal et al., 2019), its performance on reasoning-intensive tasks has not been well documented in recent literature.

Retrieval in Code + Math CodeRAG (Wang et al., 2024b) is one of the first efforts to broadly examine the ability of retrieval to improve coding performance on tasks such as SWE-Bench (Jimenez et al., 2023), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021), obtaining largely negative results. However, selected tasks, such as were not selected with challenging model reasoning in mind BRIGHT (Su et al., 2024) builds a widely scoped retrieval benchmark encompassing many reasoning tasks, such as coding, math, and science-based QA to measure retrieval based on algorithmic similarity.

Code + Math Benchmarks Early evaluations include GSM8k (Cobbe et al., 2021), HumanEval (Chen et al., 2021), However, with the rapid development of frontier models, performance in many benchmarks have been quite saturated. In our work, we select from benchmarks with generally poor baseline performance with larger room for improvement, such as USACO (Shi et al., 2024), LiveCodeBench (Jain et al., 2024), TheoremQA (Chen et al., 2023), and AoPs, a collection of olympiad math problems similar to MATH (Hendrycks et al., 2021).

3 Experiment Setup

Our objective is to evaluate the maximal retrieval performance on reasoning-intensive tasks, condi-

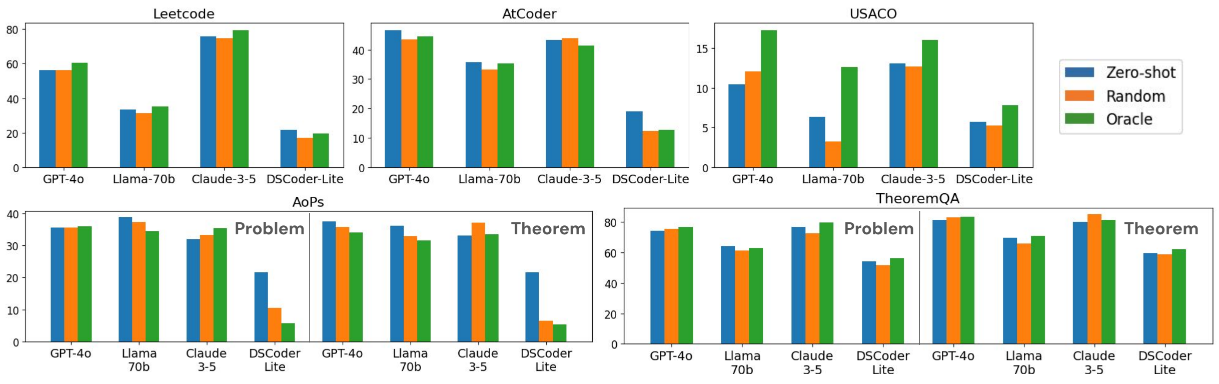


Figure 3: Performance overview of models on LeetCode, AtCoder, USACO, TheoremQA, and AoPS task sets.

tioned on relevance determined by algorithmic similarity. To extend prior work (Su et al., 2024; Wang et al., 2024b), we introduce a broader dataset of coding and math problems annotated with oracle algorithmic labels, enabling the exploration of concrete upper bounds that remain underexamined in previous studies. Specifically, we select five diverse sources of coding and math problems and analyze model performance with and without retrieval, leveraging gold-standard relevance documents during inference.

Tasks For evaluation, we select a suite of representative, challenging reasoning-intensive tasks, centered on coding and math. For coding, this includes competition-style programming problems obtained from Leetcode, AtCoder (Jain et al., 2024), and USACO (Shi et al., 2024). For math, this includes TheoremQA (Chen et al., 2023), as well as AoPS (Su et al., 2024), a subset of the problems in the MATH (Hendrycks et al., 2021) dataset with human-labelled algorithm + relevance tags.

Document Criteria We define document relevance based on algorithmic similarity, similar to (Su et al., 2024). A document is considered relevant if it contains the core algorithm or data structure required to solve the target problem. For example, a document containing telescoping series problems would be relevant to a new telescoping series problem, but not to problems involving geometric series. For coding tasks, the reference documents consist of problem statements paired with their solutions. Math task documents contain either problem-solution pairs or relevant theorem wiki pages, depending on the experimental setting. We summarize the document corpus in Appendix A.1.

Obtaining Oracle Relevance Documents We establish oracle relevance labels through a two-step process, as illustrated in Figure 2. For USACO and Leetcode, we first utilize problem tags from `https://usaco.guide/`, where documents sharing the most tags with the target task are initially selected. For math problems, relevant wiki pages serve as oracle labels. All selections then undergo human verification to confirm algorithmic relevance, with irrelevant problems removed from the test set.

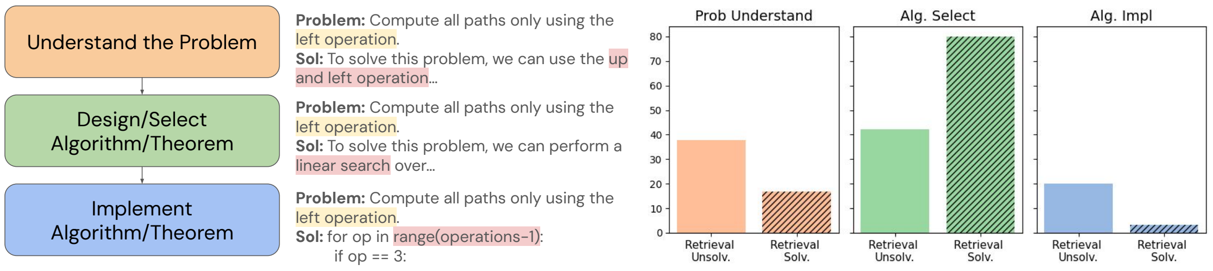
Inference Settings At inference time, we evaluate each model's ability to generate either the correct code for coding problems, or the correct numerical answer for math problems. Numerical answers are verified through exact match, and code answers are verified through correctness defining test cases. 3 different settings are analyzed: zero-shot, random retrieval, and oracle retrieval. We analyze up to 5 documents retrieved for each task.

Models We evaluate on a suite of closed and open models representing the frontier of coding/math performance, including GPT-4o, llama-3.1-70b-instruct (Llama-70b), claude-3.5-sonnet, and deepseek-coder-v2-lite (Deepseek-Coder-Lite).

4 Results

Full results can be found in Figure 3. We summarize the core findings below.

Improvements are Inconsistent The impact of retrieval with gold-relevant documents varies widely across tasks. For example, USACO shows significant improvements over the random baseline for all models tested. In contrast, tasks like LeetCode and TheoremQA exhibit marginal gains, somewhat inconsistent between models. Notably, tasks such as AtCoder and AoPS not only fail to benefit from retrieval but often experience performance *drops*.



Problem: Compute all paths only using the left operation.
Sol: To solve this problem, we can use the up and left operation...

Problem: Compute all paths only using the left operation.
Sol: To solve this problem, we can perform a linear search over...

Problem: Compute all paths only using the left operation.
Sol: for op in range(operations–1):
    if op == 3:

Figure 4: Distribution of error types (Algorithm Implementation, Selection, and Problem Understanding) in cases where RAG succeeded versus failed. RAG predominantly resolves Algorithm Selection errors while showing limited effectiveness in addressing Implementation errors.

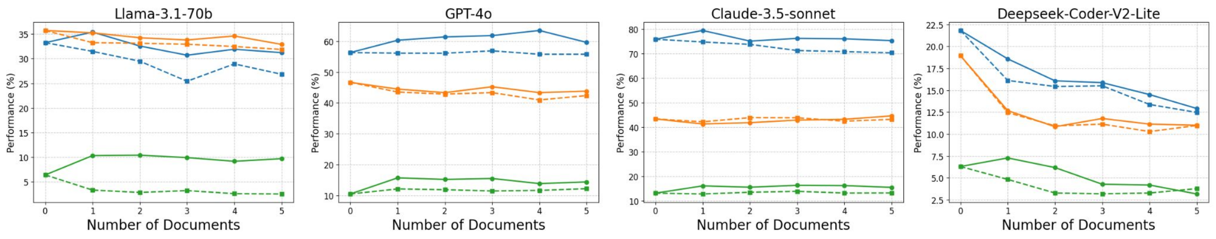


Figure 5: Performance of the model as a factor of number of documents provided at inference time (input length).

Varying Robustness to Noisy Contexts. The performance of certain models, particularly LLaMA and DeepSeek, degrades significantly when more documents are appended as context. This is in contrast to larger closed-source models, such as Claude and GPT-4o, which are largely unaffected by additional, potentially less relevant documents in context. This suggests that these models struggle to filter relevant information effectively, making them more sensitive to noise in the retrieved content.

4.1 Qualitative Analysis

Why does the effectiveness of retrieval vary so much between tasks? To better understand this, we examine a representative set of problems newly solved (n=30) and unsolved by RAG (n=45) from all problem sources evaluated. For these cases, we analyze the model's original errors, categorize them into distinct error modes, and evaluate how retrieval either mitigated or failed to address these issues. A full summary of results[2] can be found in the appendix in Figure 6.

Error Categorization. Through our analysis, we identified three primary error modes:

- **Algorithm Implementation:** The model identifies the correct algorithm but fails to implement it correctly.
- **Algorithm Selection:** The model fails to select the appropriate algorithm to solve the problem.
- **Problem Understanding:** The model misinterprets or overlooks key problem specifications, leading to incorrect algorithm design and implementation.

It is important to note that these error classes are not mutually exclusive. Instead, they represent distinct points of failure that a model may encounter along its path to solving a problem. For instance, a model must first understand the problem correctly; without this foundation, subsequent steps, such as selecting an algorithm, are likely to fail. However, understanding the problem does not guarantee correct algorithm selection, as the model may still choose an inefficient or inappropriate approach. We summarize the findings of qualitative error analysis below.

Algorithmic RAG Fixes Selection Errors We observe that problems benefiting from retrieval predominantly fall under the **Algorithm Selection** category. In particular, many of these problems were originally marked by time limit exceeded (TLE) errors, where the model lacked the knowledge to select an efficient algorithm. Retrieval often provides the model with relevant algorithmic strategies

[2]See full dataset at https://huggingface.co/datasets/benshi34/qual-analysis-reasoning-retrieval

or examples. For instance, in problem "farmer john has no large brown cow" from USACO, the model initially used a linear search, but switched to binary search given a document suggesting such optimization was necessary.

RAG Fixes Problem Understanding Errors When Contexts Align. In some cases, retrieval aids Problem Understanding errors when the retrieved documents contain problems with similar structures or specifications. For instance, in problem "2023_AIME_II_Problems/Problem_9" from AoPS, the model initially misunderstood the labeling of edges in a trapezoid. However, seeing correct reasoning about edge labeling caused the model to understand the problem as intended.

RAG Struggles with Fixing Implementation Errors. In contrast, retrieval is less effective in cases where the model struggles with **Algorithm Implementation** or **Problem Understanding**. Despite having highly relevant material in context, the model fails to utilize it to correct its solution. For example, in 'decremental-string-concatenation' from LeetCode, the model fails to keep track of the previous state while implementing its designed DP solution. Despite a similarly structured DP solution retrieved in context, the model fails to apply a fix to the current solution, repeating the same mistake.

5 Conclusion

In this paper, we provide insights into the circumstances in which algorithmic retrieval helps downstream performance on difficult reasoning tasks, such as coding and math. We find that retrieval is most helpful when it can address high-level algorithm selection errors, providing models with the necessary algorithmic insights. However, retrieval is less effective at resolving low-level implementation issues or addressing fundamental misunderstandings of the problem. We highlight two core directions for future work in reasoning-based retrieval:

Model Usage of Retrieved Documents As seen in the qualitative analysis dataset, models often remain anchored to their initial solution approaches even when retrieved documents contain clearly helpful information. Even with optimal retrieval and careful prompting, models frequently fail to incorporate retrieved information into their problem-solving process, suggesting that utilizing documents for complex reasoning tasks may be out of distribution for current frontier models. Future work should focus on developing models that can more flexibly adapt their strategies based on retrieved information.

Alternative Definitions of Utility In this work, we define "oracle retrieval" based on algorithmic similarity as constructed by humans. However, our analysis suggests that models may have their own, less intuitive utility functions for determining document relevance. For instance, a model's struggles with a dynamic programming problem might stem from data structure choices rather than algorithm implementation. Future research should investigate these alternative definitions of document utility to better align retrieval strategies with how models actually leverage external information.

6 Limitations

Tasks Considered We do not discount the possibility that the insights obtained are specific to our tasks selected, and do not generalize to other tasks. However, since coding and math problems are generally of similar formats, we hope that our task selection is general enough to broadly apply to these reasoning problems and beyond.

Models Utilized We had only considered four models, 2 open and 2 closed of varying sizes, which limits our generalization to other types and sizes of models. Additionally, we could not the best of open models available, like llama-405b, due to compute restraint.

Scale of Analysis Due to compute restraints, we only averaged Pass@1 results over 3 trials, which introduces some amount of variance into our reported model performances. Additionally, we only analyzed 54 total problems for qualitative analysis: there may be small result changes given different subsets selected.

7 Potential Risks

We do not believe this work contains any substantial risks. This work intends to analyze and show the downstream effects of retrieval augmented generation given optimal retrieval in code and math problems, and does not introduce any new systems that could be malicious.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. Theoremqa: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7889–7901.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Peng Qi, Xiaowen Lin, Leo Mehr, Zijian Wang, and Christopher D Manning. 2019. Answering complex open-domain questions through iterative query generation. *arXiv preprint arXiv:1910.07000*.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.

Quan Shi, Michael Tang, Karthik Narasimhan, and Shunyu Yao. 2024. Can language models solve olympiad programming? *arXiv preprint arXiv:2404.10952*.

Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Haisu Liu, Quan Shi, Zachary S Siegel, Michael Tang, et al. 2024. Bright: A realistic and challenging benchmark for reasoning-intensive retrieval. *arXiv preprint arXiv:2407.12883*.

Minzheng Wang, Longze Chen, Fu Cheng, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, et al. 2024a. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5627–5646.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2024b. Coderag-bench: Can retrieval augment code generation? *arXiv preprint arXiv:2406.14497*.

Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. 2022. Docprompting: Generating code by retrieving the docs. *arXiv preprint arXiv:2207.05987*.

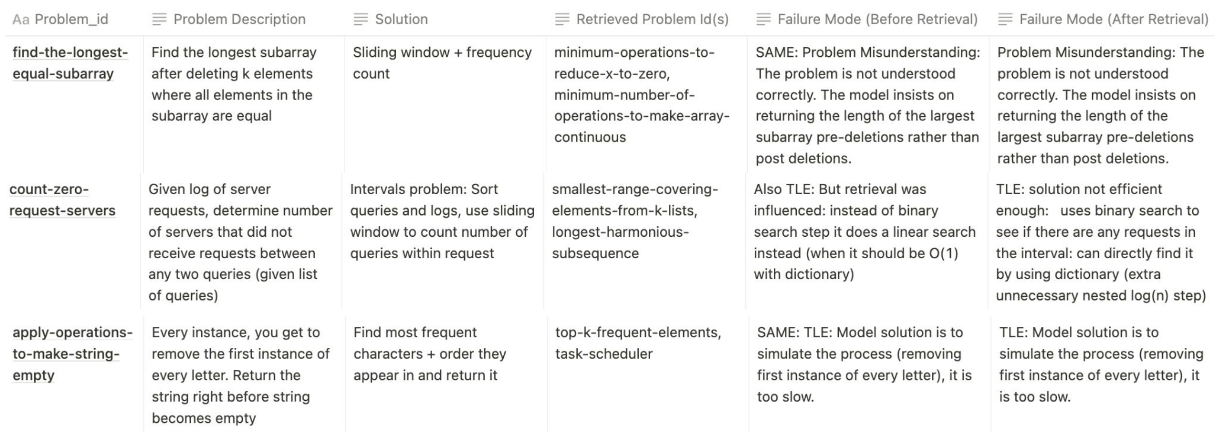
A Experiment Details

A.1 Model Endpoints

The model endpoints used were gpt-4o-2024-08-06, llama-3.1-70b, DeepSeek-Coder-V2-Lite-Instruct, claude-3-5-sonnet-20240620.

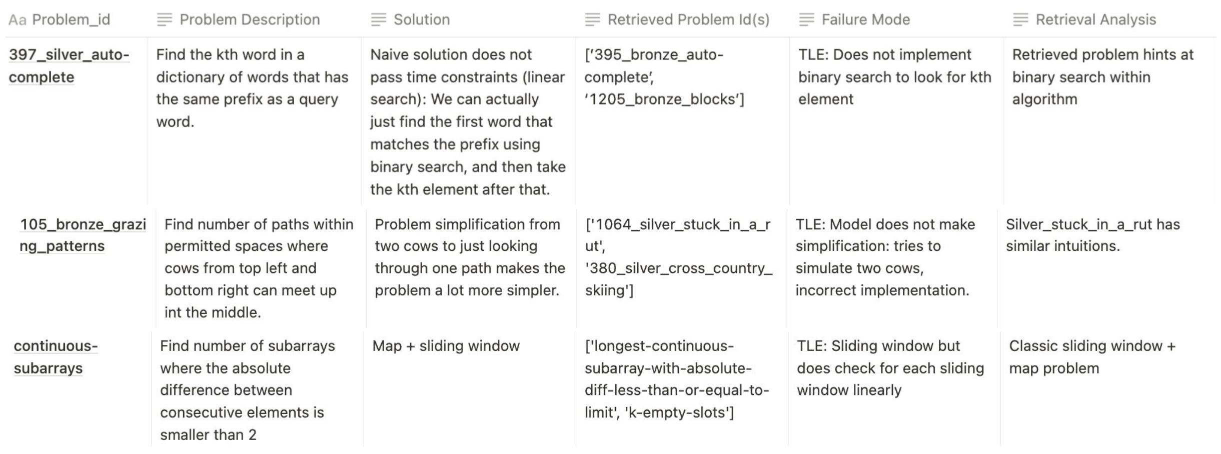
Statistic	Corpus	Value
Num Docs	Math Coding	560 3,247
Mean Doc Length	Math Coding	1,234 words 876 words
Text Contents	Math Coding	Natural Language, LaTeX Natural Language, Code
Document Types	Math Coding	Wikis, Problem + Solutions Problem + Solutions
Source	Math Coding	AoPS, TheoremQA LeetCode, AtCoder, USACO

Table 1: Stats of the document pool in which we selected the oracle documents from.



Aa Problem_id	Problem Description	Solution	Retrieved Problem Id(s)	Failure Mode (Before Retrieval)	Failure Mode (After Retrieval)
find-the-longest-equal-subarray	Find the longest subarray after deleting k elements where all elements in the subarray are equal	Sliding window + frequency count	minimum-operations-to-reduce-x-to-zero, minimum-number-of-operations-to-make-array-continuous	SAME: Problem Misunderstanding: The problem is not understood correctly. The model insists on returning the length of the largest subarray pre-deletions rather than post deletions.	Problem Misunderstanding: The problem is not understood correctly. The model insists on returning the length of the largest subarray pre-deletions rather than post deletions.
count-zero-request-servers	Given log of server requests, determine number of servers that did not receive requests between any two queries (given list of queries)	Intervals problem: Sort queries and logs, use sliding window to count number of queries within request	smallest-range-covering-elements-from-k-lists, longest-harmonious-subsequence	Also TLE: But retrieval was influenced: instead of binary search step it does a linear search instead (when it should be O(1) with dictionary)	TLE: solution not efficient enough: uses binary search to see if there are any requests in the interval: can directly find it by using dictionary (extra unnecessary nested log(n) step)
apply-operations-to-make-string-empty	Every instance, you get to remove the first instance of every letter. Return the string right before string becomes empty	Find most frequent characters + order they appear in and return it	top-k-frequent-elements, task-scheduler	SAME: TLE: Model solution is to simulate the process (removing first instance of every letter), it is too slow.	TLE: Model solution is to simulate the process (removing first instance of every letter), it is too slow.

Figure 6: Sample of error classification analysis performed on problems that are solved via retrieval.

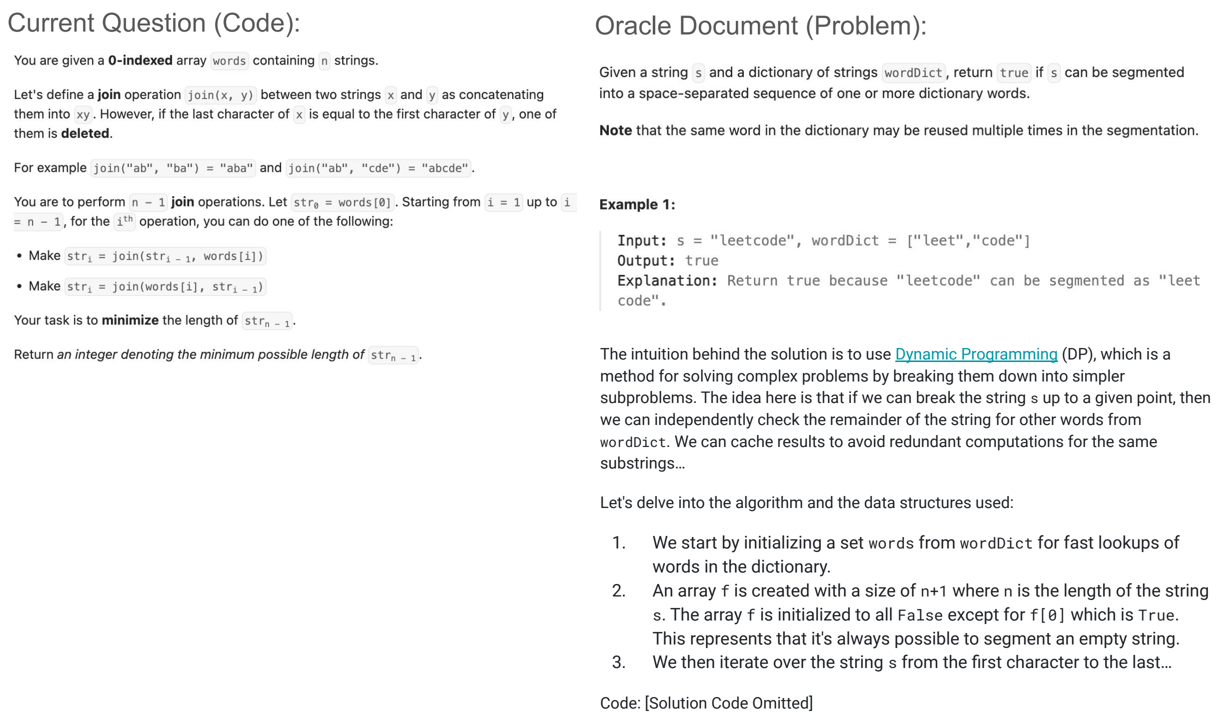


Aa Problem_id	Problem Description	Solution	Retrieved Problem Id(s)	Failure Mode	Retrieval Analysis
397_silver_auto-complete	Find the kth word in a dictionary of words that has the same prefix as a query word.	Naive solution does not pass time constraints (linear search): We can actually just find the first word that matches the prefix using binary search, and then take the kth element after that.	['395_bronze_auto-complete', '1205_bronze_blocks']	TLE: Does not implement binary search to look for kth element	Retrieved problem hints at binary search within algorithm
105_bronze_grazing_patterns	Find number of paths within permitted spaces where cows from top left and bottom right can meet up int the middle.	Problem simplification from two cows to just looking through one path makes the problem a lot more simpler.	['1064_silver_stuck_in_a_rut', '380_silver_cross_country_skiing']	TLE: Model does not make simplification: tries to simulate two cows, incorrect implementation.	Silver_stuck_in_a_rut has similar intuitions.
continuous-subarrays	Find number of subarrays where the absolute difference between consecutive elements is smaller than 2	Map + sliding window	['longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit', 'k-empty-slots']	TLE: Sliding window but does check for each sliding window linearly	Classic sliding window + map problem

Figure 7: Sample of error classification analysis performed on problems that are not solved via retrieval.

Benchmark	Num Problems	Correctness
TheoremQA (Problem)	206	Exact Match
TheoremQA (Theorem)	78	Exact Match
AoPS(Problem)	111	Exact Match
AoPS(Theorem)	94	Exact Match
USACO	307	Test Case
LeetCode	181	Test Case
AtCoder	210	Test Case

Table 2: Statistics of benchmarks used in the study. This table summarizes the number of problems and methods of correctness determination for each benchmark.



Current Question (Code):

You are given a **0-indexed** array `words` containing `n` strings.

Let's define a **join** operation `join(x, y)` between two strings `x` and `y` as concatenating them into `xy`. However, if the last character of `x` is equal to the first character of `y`, one of them is **deleted**.

For example `join("ab", "ba") = "aba"` and `join("ab", "cde") = "abcde"`.

You are to perform `n - 1` **join** operations. Let $\texttt{str}_0$ `= words[0]`. Starting from `i = 1` up to `i = n - 1`, for the $\texttt{i}^{\texttt{th}}$ operation, you can do one of the following:

- Make $\texttt{str}_\texttt{i}$ `= join(`$\texttt{str}_{\texttt{i - 1}}$`, words[i])`
- Make $\texttt{str}_\texttt{i}$ `= join(words[i], `$\texttt{str}_{\texttt{i - 1}}$`)`

Your task is to **minimize** the length of $\texttt{str}_{\texttt{n - 1}}$.

Return *an integer denoting the minimum possible length of* $\texttt{str}_{\texttt{n - 1}}$.

Oracle Document (Problem):

Given a string `s` and a dictionary of strings `wordDict`, return `true` if `s` can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

```
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet
code".
```

The intuition behind the solution is to use Dynamic Programming (DP), which is a method for solving complex problems by breaking them down into simpler subproblems. The idea here is that if we can break the string `s` up to a given point, then we can independently check the remainder of the string for other words from `wordDict`. We can cache results to avoid redundant computations for the same substrings…

Let's delve into the algorithm and the data structures used:

1. We start by initializing a set `words` from `wordDict` for fast lookups of words in the dictionary.
2. An array `f` is created with a size of `n+1` where `n` is the length of the string `s`. The array `f` is initialized to all `False` except for `f[0]` which is `True`. This represents that it's always possible to segment an empty string.
3. We then iterate over the string `s` from the first character to the last…

Code: [Solution Code Omitted]

Figure 8: Sample Problem-Document Pair