Article

iRAT: Replanning and Controlled Retrieval for Robust LLM Reasoning

Zeeshan Ali , Praneeth Vadlapati * , Aryan Singh

Posted Date: 22 July 2025

doi: 10.20944/preprints202507.1289.v2

Keywords: large language models; artificial intelligence; chain-of-thought reasoning; uncertainty-aware language models; reasoning in LLMs; context-aware reasoning; LLM reasoning frameworks

Article

iRAT: Replanning and Controlled Retrieval for Robust LLM Reasoning

Zeeshan Ali [†], Praneeth Vadlapati [*,†] and Aryan Singh

University of Arizona, USA

[*] Correspondence: praneethv@arizona.edu

[†] Equal contributions.

Abstract

Large Language Models (LLMs) have demonstrated significant capabilities in answering questions using techniques such as Chain of Thought (CoT) and Retrieval-Augmented Generation (RAG). CoT enables step-by-step reasoning to improve accuracy, while RAG supplements LLMs with relevant external information. Retrieval-Augmented Thoughts (RAT) combines CoT and RAG to provide a more robust factual foundation and coherence in reasoning chains. However, RAT is limited in its ability to handle uncertainty and lacks replanning, often resulting in unnecessary retrievals, inefficiencies, and globally inconsistent reasoning. To address these limitations, we introduce iRAT, a novel reasoning framework that enhances RAT through retrieval control and replanning. iRAT dynamically evaluates uncertainty in initial responses, employs controlled and filtered retrievals to obtain only the most relevant context, revises thoughts to align with new content, and uses replanning to correct previous thoughts. Evaluations demonstrated that iRAT outperforms RAT in HumanEval, MBPP, and GSM8K datasets, while reducing retrievals by a considerable amount. The source code is available at github.com/prane-eth/iRAT. The fine-tuned model used for replanning is available at huggingface.co/zeeshan5k/iRATReasoningChainEvaluatorv2.

Keywords: large language models; artificial intelligence; chain-of-thought reasoning; uncertainty-aware language models; reasoning in LLMs; context-aware reasoning; LLM reasoning frameworks

Introduction

Background

Large Language Models (LLMs) are recognized for their effectiveness in addressing user queries based on information available through training, fine-tuning, or in-context learning. Among the key techniques to enhance their capabilities is "Chain of Thought" (CoT) [1], also known as "reasoning," which prompts LLMs to generate intermediate reasoning steps prior to generating final responses. CoT outperforms few-shot prompting with enhanced response accuracy. Another technique is Retrieval-Augmented Generation (RAG) [2], which supplements LLMs with new information through retrievals from external sources. However, recent findings on reasoning reveal an illusion of thinking in LLMs when facing complex tasks [3].

Literature Review

Previous research on Retrieval-Augmented Thoughts (RAT) [4,5], referred to as "old-RAT", utilized CoT combined with RAG, which mitigated hallucinations and incoherent reasoning in the LLMs, increasing response accuracy. Old-RAT generates an initial draft, divides it into reasoning steps, and retrieves external knowledge at each step to iteratively refine the reasoning process. This approach substantially mitigates hallucinations and improves factual grounding. By incorporating retrieval rather than relying solely on the LLM's knowledge base, old-RAT achieves improved performance across various reasoning tasks. However, old-RAT lacks a mechanism to assess uncertainty, leading to unnecessary retrievals and thoughts, which reduces its efficiency at scale. Furthermore, it fails to optimize reasoning globally and does not employ a model to update previous thoughts when new thoughts contradict them, which implies it lacks end-to-end trajectory optimization.

Self-RAG [6] introduces a reflective framework where an LLM dynamically decides whether to retrieve, generate, or critique at each step using specialized reflection tokens. This adaptive mechanism improves factual accuracy, enables generalization across different tasks, and maintains low overhead during inference. While effective, the reflection token training process may exhibit instability, and the framework demands significant computational resources during initial training. Additionally, the quality of the generated reflection signals significantly affects the performance and renders the system sensitive to prompt and domain variations. RAG2 [7] improves factual grounding in the medical domain using a rationale-based approach, where the LLM generates intermediate rationales to guide retrieval queries and filters retrieved results using a perplexity-based scoring model. RAG2 ensures a balanced use of multiple corpora to mitigate source bias and improve reliability. However, the system is limited by its domain specificity, limited filtering capacity (handling one snippet at a time), and elevated pipeline complexity due to additional rationale.

Solution

This study introduces iRAT, an enhanced retrieval-augmented reasoning framework derived from old-RAT, and improves reasoning through retrieval control policies and dynamic replanning mechanisms to reduce unnecessary retrievals, filter undesirable results, dynamically correct intermediate reasoning inaccuracies, and adapt inference pipelines for complex, long-horizon tasks. iRAT is designed to be a robust and resource-efficient retrieval-augmented thinking framework capable of adapting to complex tasks, enabling higher accuracy and resource efficiency in real-world applications compared to old-RAT.

Methods

Initial Draft Generation

The first stage in iRAT employs an LLM to generate an initial draft for each query. The system employed an open-source model Llama 3.3 (70B) [8], which is known for its performance despite its small size. Each query undergoes a validation process to identify and mitigate potentially harmful content, including malicious patterns and unsupported characters.

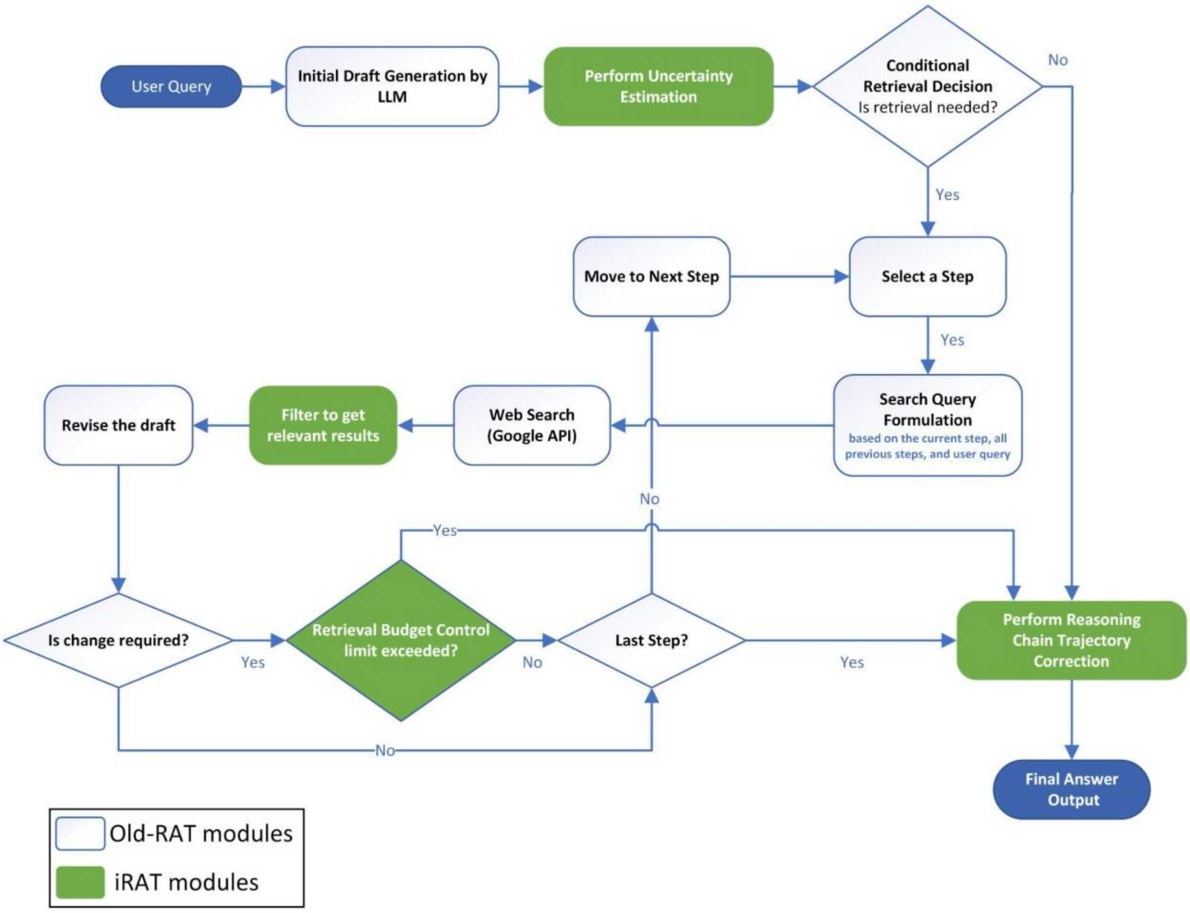
User Query
Initial Draft Generation by LLM
Perform Uncertainty Estimation
Conditional Retrieval Decision
Is retrieval needed?
No
Yes
Move to Next Step
Select a Step
Yes
Search Query Formulation
based on the current step, all previous steps, and user query
Web Search (Google API)
Filter to get relevant results
Revise the draft
No
Yes
Is change required?
Yes
Retrieval Budget Control limit exceeded?
No
Last Step?
Yes
Perform Reasoning Chain Trajectory Correction
No
Final Answer Output
Old-RAT modules
iRAT modules

Figure 1. iRAT architecture diagram. Available at: https://github.com/prane-eth/iRAT/blob/main/assets/iRAT-Full-architecture.jpeg.

Uncertainty Estimation

This step measures the model's confidence in answering a query. The embedding model all-MiniLM-L6-v2 [9] was selected due to its established performance and small size. This process involves generating three initial responses to the query, encoding drafts into embeddings using the model, and calculating pairwise cosine similarities of the embeddings to measure the consistency of the responses. The average of these pairwise similarity scores represents a self-consistency score, also known as "certainty." The uncertainty is calculated as 1 - average_consistency .

Retrieval

Retrieval Decision

This module triggers retrieval only when uncertainty exceeds a threshold of 30%. This step enables selective retrieval to maintain accuracy while reducing resource usage when the model's confidence is high, allowing the optimization of cost and latency.

Retrieval-Based Revision with Budget Control

If retrieval is triggered, a process similar to that of old-RAT is employed to revise the draft in multiple steps. The text is divided into chunks to create multiple steps. At each step, a search query is formulated for the chunk using the selected LLM. The queries are used to fetch paragraphs from the web to update the chunk. Budget control policy is enforced to allow only one retrieval per chunk. While old-RAT generates one chunk per paragraph, iRAT further minimizes retrievals by merging consecutive chunks, subject to a limit of 500 characters per chunk. Budget control is essential because excess retrievals increase computational and financial costs.

The Google Search API is used to retrieve the top 10 most relevant results according to Google. To maintain fairness, URLs containing HumanEval, MBPP, and GSM8K datasets are excluded to ensure the model does not receive solutions from the selected datasets. Unlike old-RAT, the system supports retrievals from websites such as StackOverflow and Stack Exchange pages by utilizing their public API. The system mitigates unsupported URLs, such as YouTube and PDF files, to prevent retrieval errors.

Result Filtering

SEO spamming [10] in Google's Search Results may result in irrelevant, low-value, or malicious pages that have the potential to mislead LLMs. Most pages include non-informative elements such as headers and advertisements, potentially interfering with model comprehension. Long content in a web page might lead to information overload, potentially degrading LLM response accuracy and increasing inference costs. Hence, spam URLs are filtered based on the page's URL and domain using Google Safe Browsing API [11] and Malicious URLs Dataset [12]. Paragraphs are extracted from page content, and the new "Attention-Retrieval" method selects relevant paragraphs.

The Attention-Retrieval method employs pre-trained re-ranking models to generate ranks and scores of retrieved paragraphs based on the query. The model ms-marco-MiniLM-L6-v2 [13] was selected due to its optimal model size and the scores on the official web page [14]. Top 8 most relevant paragraphs are predicted using the model, and the results are further filtered to select paragraphs above a threshold of 50% of score. Consecutive paragraphs are merged, subject to a limit of 500 characters per paragraph. Similar to old-RAT, the response is revised based on each selected paragraph.

Replanning

To address the challenge of global end-to-end optimization, this module reviews all the steps and generates feedback to enhance previous steps to align the whole chain. This module employs an ensemble of a reward model and DeepSeek-R1-Distill-Qwen-1.5B [15], quantized and fine-tuned using LoRA [16]. This module mitigates error propagation and reduces contradictions and inaccuracies in prior steps, unlike the old-RAT process. The fine-tuned model used for replanning is available at huggingface.co/zeeshan5k/iRATReasoningChainEvaluatorv2.

Final Evaluation

The system gets evaluated using HumanEval [17] and MBPP [18] datasets for coding tasks, and GSM8K [19] dataset for mathematical reasoning tasks. System evaluation employed the pass@k metric [20] on HumanEval and MBPP datasets, and the Exact Match (EM) metric [21] to match the answers on the GSM8K dataset. The pass@k metric measures the model's code passing all the test cases provided, in the first "k" attempts, while the Exact Match compares the dataset's answers to the model's answers. An additional metric that was introduced was the average number of retrievals required to answer a query. This step compares iRAT with old-RAT on the same machine using the same model to enable a fair comparison.

Results and Discussion

Performance

The performance and comparison of old-RAT and iRAT across HumanEval, MBPP, and GSM8K datasets are summarized in the tables below.

Coding task results

Table 1. HumanEval and MBPP result comparison of old-RAT and iRAT.

Method	HumanEval pass@1 score	MBPP pass@1 score
Old-RAT	63.41%	63.64%
iRAT	79.27%	76.36%
Improvement	15.86%	12.72%

Mathematical Reasoning Task Results

Table 2. GSM8K result comparison of old-RAT and iRAT.

Method	GSM8K Exact Match score
Old-RAT	81.35%
iRAT	89.39%
Improvement	8.04%

The bar charts below compare the scores visually.

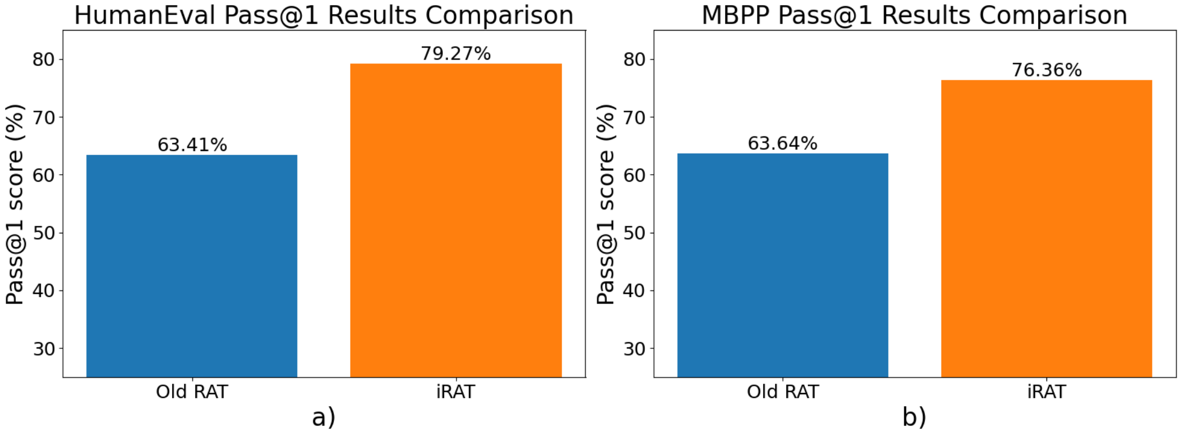

Figure 2. Bar charts comparing pass@1 scores of old-RAT and iRAT in (a) HumanEval dataset and (b) MBPP dataset.

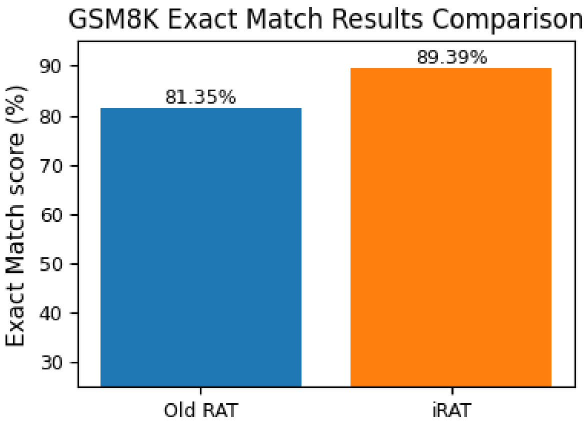

Figure 3. Bar charts comparing Exact Match (EM) scores of old-RAT and iRAT in GSM8K dataset.

doi:10.20944/preprints202507.1289.v2

Usage of Retrievals

A comparison of the average retrievals used per query using old-RAT and iRAT for all three datasets is presented in a table below.

Table 3. Average retrievals per query for RAT and iRAT.

Dataset	Average Retrievals (old-RAT)	Average Retrievals (iRAT)	Reduction in retrievals
HumanEval	4.46	3.16	29.15%
MBPP	5.24	3.36	35.88%
GSM8K	3.43	1.76	48.69%

Discussion

In HumanEval, the largest increase in accuracy and the smallest reduction in retrievals were observed. For MBPP, a relatively moderate improvement in accuracy and a moderate reduction in retrievals were observed. GSM8K demonstrated a smaller accuracy gain accompanied by the largest reduction in retrievals. A greater improvement in performance was observed in coding tasks compared to mathematical tasks, likely due to the old-RAT already achieving over 80% accuracy on the latter, suggesting a limited room for further improvement. A substantial reduction in retrievals was noted for mathematical tasks compared to coding tasks. iRAT demonstrated its potential to enhance accuracy through replanning while reducing retrievals. Notably, a greater reduction in retrievals corresponds with a smaller performance gain. Importantly, the reduction in retrievals did not negatively impact performance, indicating the effectiveness of trajectory correction.

Limitations and Future Work

Future work may explore a self-reflection process using the selected base model itself, as it possesses a larger knowledge base than the Chain Evaluator model. Old-RAT references the use of vector databases. However, their source code employs Google Search, a procedure that iRAT also adopts to enable fair comparison. Future work may compare the performance of both old-RAT and iRAT using vector databases, and also compare both systems for queries that require a significantly larger number of reasoning steps. Similar to old-RAT, iRAT has been tested using English datasets, although future work could extend it to multilingual datasets. The source code was not designed for commercial deployment, as it has not been evaluated under high-concurrency conditions. Future work may enable support for more websites and PDF files. iRAT has been experimented on coding and mathematical reasoning tasks, and may also be experimented across diverse problem domains.

Conclusions

This study introduced iRAT, an enhanced reasoning framework developed upon old-RAT, incorporating new modules designed to address its limitations. iRAT improves reasoning accuracy and resource efficiency through controlled retrieval and replanning mechanisms. These enhancements enable iRAT to selectively leverage external knowledge sources and revise intermediate reasoning through replanning. Experimental results indicate that the system improves correctness and coherence in multi-step reasoning tasks. iRAT improved accuracy while enhancing control over resource consumption through the reduction of unnecessary retrievals and processing overhead, rendering it more suitable for real-world applications compared to CoT, RAG, and old-RAT. While several challenges and opportunities for future work remain, iRAT represents a novel advancement in adaptable reasoning frameworks.

Author Contributions: Zeeshan Ali: Architecture, uncertainty evaluation, and Chain Evaluator model. Praneeth Vadlapati: Pipeline, result-filter module, evaluation, most of the code and paper, and team leadership. Aryan Singh: Retrieval module with budget control, dataset analysis, MBPP pre-processing, pipeline wireframe, and bug fixing in budget control. Alvaro Arteaga: User input scanning, and the idea of spam website filter.

Acknowledgments: We sincerely thank Dr. Leonard D. Brown at the University of Arizona for suggesting improvements to this work. We thank Alvaro Arteaga (aarteaga@arizona.edu) at the University of Arizona for contributions to the source code.

Appendix

Examples of thoughts generated

Example 1: (From HumanEval Dataset)

Explanation of the task: The dataset mentions a function template with a description. The model is expected to write the code according to the description and pass all the mentioned test cases.

As mentioned, the workflow of the iRAT is:

Initial draft generation of user query —> Revise draft through retrievals —> Get feedback from Chain Evaluator model -—> Generate a final answer based on the query and the feedback.

Query:

Write functional code in Python according to the description. Start your code with:

```
from typing import List
def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
```

Old-RAT answer:

```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) -> bool:

   # First, we sort the list of numbers in ascending order

   numbers.sort()

   # Then, we iterate over the sorted list to check the difference between adjacent numbers

   for i in range(len(numbers) - 1):

         # If the difference between any two adjacent numbers is less than or equal to the threshold, we return
```

```
True
      if numbers[i + 1] - numbers[i] <= threshold:
          return True
  # If we have checked all pairs of adjacent numbers and haven't found any that are closer than the threshold,
we return False
  return False


# Test cases
print(has_close_elements([1.0, 2.0, 3.0], 0.5))   # False
print(has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3))   # True
```

iRAT revised draft:

## Introduction

The problem requires us to determine if any two numbers in a given list are closer to each other than a given threshold. This can be achieved by using the `math.isclose()` function in Python, which checks if two values are close to each other.

## Solution

To solve this problem, we can use a simple function in Python that iterates over the list of numbers and checks if any two numbers are closer to each other than the given threshold. Here is the Python code to achieve this:

~~~
```python
from typing import List
import math


def has_close_elements(numbers: List[float], threshold: float) -> bool:
 for i in range(len(numbers)):
     for j in range(i + 1, len(numbers)):
         if math.isclose(numbers[i], numbers[j], abs_tol=threshold):
             return True
 return False


# Example usage:
numbers = [1.0, 2.8, 3.0, 4.0, 5.0, 2.0]
threshold = 0.3
result = has_close_elements(numbers, threshold)
~~~

print(result)   # Output: True

```

## Explanation

The provided function `has_close_elements` takes a list of numbers and a threshold value as input. It iterates over the list of numbers and checks if any two numbers are closer to each other than the given threshold using the `math.isclose()` function. If it finds any two numbers that are closer to each other than the threshold, it returns True. Otherwise, it returns False.

## Advice

To use this function, simply call it with a list of numbers and a threshold value as arguments. The function will return a boolean value indicating whether any two numbers are closer to each other than the threshold. Note that this function uses the `math.isclose()` function, which uses a relative or absolute tolerance to compare the values. The `abs_tol` parameter is used to specify the absolute tolerance.

**Evaluator Feedback**:

The reasoning chain is correct but the code provided is not functional. For example, the function does not return the expected boolean values when tested with the sample inputs.

**iRAT final answer**:

```
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:

for i in range(len(numbers)):

    for j in range(i + 1, len(numbers)):

        if abs(numbers[i] - numbers[j]) < threshold:

            return True

return False

# Example usage:

numbers = [1.0, 2.8, 3.0, 4.0, 5.0, 2.0]

threshold = 0.3

result = has_close_elements(numbers, threshold)

print(result)   # Output: True

numbers = [1.0, 2.0, 3.0]

threshold = 0.5
```
```

```
result = has_close_elements(numbers, threshold)

print(result)    # Output: False
```

Example 2: (From MBPP Dataset)

Query:

Write a function to find the largest subset where each pair is divisible.?

Sample test case:

assert largest_subset([1, 3, 6, 13, 17, 18], 6) == 4

Old-RAT answer:

## Step 1: Define the problem and the goal

The goal is to write a function that finds the largest subset of a given list of numbers where each pair of numbers is divisible by a given divisor.

## Step 2: Identify the key elements of the problem

The key elements are the list of numbers, the divisor, and the requirement that each pair of numbers in the subset must be divisible by the divisor.

## Step 3: Determine the approach to solve the problem

To solve this problem, we can use a dynamic programming approach. We will iterate over the list of numbers and for each number, we will check if it can be added to the existing subset without violating the divisibility condition.

## Step 4: Write the function

~~~
```python
def largest_subset(nums, divisor):
    """
    Finds the largest subset of a given list of numbers where each pair of numbers is divisible by a given divisor.
    Args:
        nums (list): A list of integers.
        divisor (int): The divisor that each pair of numbers in the subset must be divisible by.
    Returns:
        int: The size of the largest subset that meets the condition.
    """
    if not nums:
        return 0
~~~

```
    nums.sort()
    dp = [1] * len(nums)

    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] % nums[j] == 0 and nums[i] % divisor == 0:
                dp[i] = max(dp[i], dp[j] + 1)

    return max(dp)
```

## Step 5: Test the function

```python
print(largest_subset([1, 3, 6, 13, 17, 18], 6))    # Output: 4
```

The final answer is: 4

iRAT - revised draft:

## Introduction to the Problem

The problem asks us to find the largest subset of a given set of distinct positive integers such that every pair of elements in this subset satisfies the condition that one element is divisible by the other. This means for any two elements in the subset, either the first element is divisible by the second or the second is divisible by the first. If there are multiple solutions, we can return any of them.

## Understanding the Problem

To efficiently find a subset where any two numbers divide each other, we should think about what makes a divisible subset work. A key insight is that if we sort our numbers, we can build this subset more systematically. When numbers are sorted, we only need to check if a larger number is divisible by a smaller one (not both ways). This approach simplifies the problem and allows us to use dynamic programming to solve it. The problem is related to finding subsets with specific properties, similar to the problem of finding subsets where the sum of elements is a prime number, as discussed in the provided text.

## Approach to the Solution

We start by sorting the given list of numbers. Then, we initialize a dynamic programming table where each entry represents the size of the largest subset that can be formed using the numbers up to that index. We fill up the table by iterating over the list of numbers. For each number, we check all previous numbers to see if the current number is divisible by the previous number. If it is, we update the size of the largest subset that can be formed using the current number. This method is similar to the approach used to calculate the number of subsets of a set that add up to a prime number, where the formula $C_i[s] = C_{i-1}[s] + C_{i-1}[s-i] + (i == s)$ is used to efficiently calculate the result.

## Implementation Details

Let's break down how to solve this step by step. First, we sort the list of numbers. Then, we initialize the dynamic programming table with all elements set to 1, since a single number is always a subset of size 1. We then iterate over the sorted list, and for each number, we check all previous numbers. If the current number is divisible by a previous number, we update the corresponding entry in the table to be the maximum of its current value and the value of the previous entry plus 1. This is because we can extend the subset that ends at the previous number by appending the current number to it.

## Reconstructing the Subset

After filling up the table, we find the entry that corresponds to the maximum value in the table. This entry tells us where the largest subset ends. We then reconstruct the subset by backtracking from the last number to the beginning of the list, appending each number to the subset if it is divisible by the previous number in the subset. The final size of the subset is the answer to the problem.

## Final Answer

Here is the Python function that implements the above approach:

~~~
```python
def largest_subset(nums):
    if not nums:
        return 0
    nums.sort()
    dp = [1] * len(nums)

    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] % nums[j] == 0:
                dp[i] = max(dp[i], dp[j] + 1)

    max_length = max(dp)
    subset = []
    i = dp.index(max_length)
    while max_length > 0:
        subset.append(nums[i])
~~~

```
        max_length -= 1
        for j in range(i):
            if nums[i] % nums[j] == 0 and dp[j] == max_length:
                i = j
                break
    return len(subset)


assert largest_subset([1, 3, 6, 13, 17, 18]) == 4
```

This solution has a time complexity of O(n^2) due to the nested loops, where n is the number of elements in the input list. The space complexity is O(n) for the `dp` table.

Evaluator Feedback:
The reasoning is concise and correct, explaining the steps of solving the Largest Subset problem. It provides an efficient code.

iRAT - final answer:
To solve the problem of finding the largest subset where each pair is divisible, we can follow these steps:

1. Sort the list of numbers in ascending order.
2. Initialize a dynamic programming table where each entry represents the size of the largest subset that can be formed using the numbers up to that index.
3. Fill up the table by iterating over the list of numbers. For each number, check all previous numbers to see if the current number is divisible by the previous number. If it is, update the size of the largest subset that can be formed using the current number.
4. After filling up the table, find the entry that corresponds to the maximum value in the table. This entry tells us where the largest subset ends.
5. Reconstruct the subset by backtracking from the last number to the beginning of the list, appending each number to the subset if it is divisible by the previous number in the subset.

Here is the Python function that implements the above approach:

```python
def largest_subset(nums, n):
    if not nums:
        return 0
    nums.sort()
```

~~~
    dp = [1] * len(nums)


    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] % nums[j] == 0:
                dp[i] = max(dp[i], dp[j] + 1)


    max_length = max(dp)
    subset = []
    i = dp.index(max_length)


    while max_length > 0:
        subset.append(nums[i])
        max_length -= 1
        for j in range(i):
            if nums[i] % nums[j] == 0 and dp[j] == max_length:
                i = j
                break


    return len(subset)


assert largest_subset([1, 3, 6, 13, 17, 18], 6) == 4
```
~~~

This solution has a time complexity of O(n^2) due to the nested loops, where n is the number of elements in the input list. The space complexity is O(n) for the `dp` table.

The function takes two parameters: `nums` which is the list of numbers, and `n` which is not used in the function. The function returns the size of the largest subset where each pair is divisible.

The provided unit test `assert largest_subset([1, 3, 6, 13, 17, 18], 6) == 4` checks if the function returns the correct result for the given input. The expected output is 4, which is the size of the largest subset where each pair is divisible.

Example 3: (From GSM8K Dataset)

Query Explanation:

GSM8K dataset contains questions and solutions that mention exact answers after "####" at the end. Each question is provided to the model, which is expected to return a response to match the expected answer.

Query:

If a bag of marbles costs $20 and the price increases by 20% of the original price every two months, how much would a bag of marbles cost after 36 months?

At the end, write the final answer after "####" at last.
For example, if the answer is ABC, write "#### ABC"

Old-RAT answer:

To solve this problem, we first need to calculate the price increase every 2 months, which is 20% of the original price of $20.

20% of $20 = 0.20 * $20 = $4
So, every 2 months, the price of the bag of marbles increases by $4.
Since we want to find the price after 36 months, we need to calculate how many 2-month periods are in 36 months:

36 months / 2 months = 18 periods
Now, we can calculate the total price increase after 18 periods:

$4 * 18 = $72

However, this is not the correct way to calculate the price increase, as the 20% increase is applied to the original price every 2 months, not added as a fixed amount.

Let's recalculate:

Original price = $20
Price after 2 months = $20 + (20% of $20) = $20 + $4 = $24
Price after 4 months = $24 + (20% of $24) = $24 + $4.80 = $28.80
Price after 6 months = $28.80 + (20% of $28.80) = $28.80 + $5.76 = $34.56

We can see a pattern emerging. The price increase is 20% of the previous price every 2 months.

To simplify the calculation, we can use the formula for compound interest:

A = P * (1 + r)^n

doi:10.20944/preprints202507.1289.v2

where:
A = final price
P = original price = $20
r = monthly interest rate = 20%/2 = 10% per month (compounded every 2 months)
n = number of periods = 36 months / 2 months = 18 periods

However, since the increase is compounded every 2 months, we should adjust the formula accordingly:

A = P * (1 + 0.20)^n

where:
n = number of 2-month periods = 18

A = $20 * (1 + 0.20)^18
A = $20 * (1.20)^18
A = $20 * 45.4737
A ≈ $909.47

However, considering the instructions provided and the necessity to follow the format to the letter, it seems there was an initial misunderstanding in directly applying the provided text to solve the given problem. The original text discusses CAM caps in the context of commercial real estate leases, which does not directly apply to calculating the cost of a bag of marbles over time with a percentage increase. The correct approach involves recognizing the problem as one of compound interest or geometric progression, where the price increases by a fixed percentage at regular intervals.

Given the mistake in directly applying the provided formula and recognizing the error in calculation approach, let's correct the methodology for calculating the future price of the marbles based on the correct understanding of compound interest or percentage increase over time:

The price increases by 20% every 2 months. To find the price after 36 months, we apply the increase correctly:

- After 2 months: $20 * 1.20 = $24
- After 4 months: $24 * 1.20 = $28.80
- After 6 months: $28.80 * 1.20 = $34.56
- Continuing this pattern for 18 periods (36 months), we calculate the final price using the formula A = P * (1 + r)^n, where r = 0.20 (20% increase), and n = 18 (number of 2-month periods).

Correctly calculating this:
A = $20 * (1.20)^18

Using a calculator for the exact value:

A ≈ $20 * 45.4737

A ≈ $909.47

The final answer is: #### $909.47

iRAT - final answer:

To find the cost of a bag of marbles after 36 months, given that it initially costs $20 and its price increases by 20% of the original price every two months, we first need to understand the pattern of the price increase.

The price increase is 20% of the $20 original price. To find 20% of $20, we multiply $20 by 0.20, which equals $4. This $4 increase happens every two months.

Since the price increases every two months, over 36 months, the price will increase 36 / 2 = 18 times.

The correct approach to calculating the total price after 36 months involves adding the $4 increase to the price every two months, for a total of 18 increases. The formula to calculate the final price after 18 increases would be: $20 (original price) + $4 (increase every 2 months) * 18 (number of increases).

Performing the calculation: $20 + $4 * 18 = $20 + $72 = $92.

Therefore, after 36 months, the bag of marbles would cost $92.

$92

Datasets

HumanEval: OpenAI's code generation dataset comprises 164 Python programming problems, each with a function signature, docstring, body, and multiple unit tests. It is used to evaluate the correctness of program synthesis from natural language descriptions. The dataset covers coding problems related to language comprehension, algorithms, and basic mathematics, with some tasks resembling introductory software engineering interview questions. It includes columns "prompt," "canonical_solution," "test," and "entry_point".

MBPP: Google's code generation dataset that includes 974 Python programming problems designed to be solvable by beginner programmers. Each problem consists of an English task description, a code solution, and three automated test cases. Old-RAT evaluated the test set from index 11 to 175, and the same has been used for iRAT.

GSM8K: OpenAI's mathematical reasoning dataset that consists of 8790 high-quality and linguistically diverse mathematical word problems. It contains questions and answers. Answers include a reasoning followed by a number after "####" at the end, which is used to compare with the model's response.

MS MARCO: Microsoft's dataset that contains 1 million real user queries, each query paired with at most 10 results containing paragraphs, URLs, and the selections of relevant paragraphs. We use this to evaluate re-ranking models on coding tasks prior to implementation in the result-filtering module. More information is mentioned in a section below.

Malicious URLs Dataset: A vast dataset of 651,191 URLs, including 428103 benign or safe URLs, 96457 defacement URLs, 94111 phishing URLs, and 32520 malware URLs. All non-safe URLs were used to filter the query results in the result-filtering module.

Hardware and Software

Table A1 summarizes the hardware configurations of the Virtual Machines (VMs) that were used for evaluating old-RAT and iRAT. Table A2 lists the Python libraries used for this project and their respective versions.

Table A1. Hardware details.

Name	Value
Platform	Azure
VM name	B4as_v2
vCPUs	4
Memory	16 GB
GPU	N/A

Table A2. Python (v3.12.3) - packages used.

Package	Version
beautifulsoup4	4.13.4
cohere	5.15.0
datasets	3.6.0
google-api-python-client	2.174.0
gradio	5.35.0
html2text	2025.4.15
html5lib	1.1
human-eval	1.0.3
IPython	9.4.0
jupyter	1.1.1
langchain	0.3.26
langchain-community	0.3.27
last_layer	0.1.33
loguru	0.7.3
lxml	6.0.0
matplotlib	3.10.3
numpy	2.3.1
openai	1.93.0
pysafebrowsing	0.1.4
python-dotenv	1.1.1
readability-lxml	0.8.4.1
requests	2.32.4
sentence-transformers	5.0.0
simple-cache	0.35

tiktoken	0.9.0
transformers	4.53.0

Evaluation of Re-Ranking Models for Coding Tasks

The Attention-Retrieval method was evaluated on coding tasks using pre-trained re-ranking models, which assign ranks and scores based on query and paragraphs. A coding-related subset was extracted from the MARCO dataset [22] based on the occurrence of coding-related keywords in the queries. The dataset includes 2511 training rows and 310 validation rows. Scores were computed using the R-Precision metric, which measures the proportion of relevant paragraphs within the top-R-ranked results, where R denotes the number of ground-truth relevant paragraphs for a given query. For each query, the model was used to predict scores for each paragraph, with the top R scoring paragraphs selected. The accuracy of selecting the correct paragraphs was computed.

Table A3. Accuracy on the coding subset of MARCO.

Model	Parameters	Estimated R-Precision Score
ms-marco-MiniLM-L6-v2 [13]	22.7M	**78.71%**
ms-marco-MiniLM-L4-v2 [23]	**19.2M**	76.45%
mxbai-rerank-xsmall-v1 [24]	70.8M	73.87%

Models pre-trained on the MARCO dataset outperformed the larger “mxbai” model on the MARCO-derived task. These results demonstrate the suitability of the evaluated models for coding tasks. Considering the trade-off between model size and performance, ms-marco-MiniLM-L6-v2 was selected for integration into the result-filtering module. The evaluation code is available at: https://github.com/prane-eth/iRAT/blob/main/notebooks/Result-filter/AR_evaluate.py#L33.

Keywords Used to Extract Coding-Related Rows from MARCO

python, java, JS, c++, css, html, typescript, php, Swift, kotlin, perl, sql, matlab, objective-c, c#, fortran, cobol, vba, groovy, haskell, clojure, f#, solidity, xml, json, yaml, protobuf, graphql, django, laravel, node.js, tensorflow, pytorch, keras, numpy, scikit-learn, sklearn, hadoop, kubernetes, docker, ansible, terraform, azure, AWS, gcp, linux, unix, Git, github, gitlab, bitbucket, jenkins, travis, circleci, maven, gradle, webpack, babel, redux, data structure, object-oriented, functional programming, algorithm, API, REST, GraphQL, microservices, serverless.

Chain Evaluator Model - Initial Experiment with Reinforcement Learning (RL)

As part of our experimental investigation, we trained a supervised reward model to evaluate reasoning chains generated by an LLM with robust factual grounding. Each chain was rated on a scale from 1 to 5 based on its logical coherence and factual accuracy. However, the reward model, based on the RoBERTa-base architecture, demonstrated limited capacity to learn meaningful reward signals. Upon further analysis, we determined that the model lacked the embedded world knowledge required to interpret and assess the reasoning chains accurately. For instance, evaluating a reasoning chain that explains why the sky appears blue necessitates foundational understanding of physical phenomena such as light scattering and atmospheric composition—knowledge that the relatively small reward model did not possess.

Unlike the LLM, which implicitly encodes such knowledge through extensive pretraining, the reward model operated in isolation, relying primarily on surface-level token patterns without sufficient contextual depth. As a result, it was unable to reliably distinguish between valid and flawed reasoning. This mismatch in knowledge and interpretive capacity ultimately led us to abandon the reward model approach. Instead, we employed the fine-tuned LLM directly as an evaluator, which demonstrated substantially improved alignment with reasoning accuracy and coherence.

doi:10.20944/preprints202507.1289.v2

References

1. J. Wei et al., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” 2023, arXiv. [Online]. Available: http://arxiv.org/abs/2201.11903
2. Y. Gao et al., “Retrieval-Augmented Generation for Large Language Models: A Survey,” 2024, arXiv. [Online]. Available: http://arxiv.org/abs/2312.10997
3. P. Shojaee, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar, “The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity,” 2025, arXiv. doi: 10.48550/arXiv.2506.06941.
4. Z. Wang, A. Liu, H. Lin, J. Li, X. Ma, and Y. Liang, “RAT: Retrieval Augmented Thoughts Elicit Context-Aware Reasoning and Verification in Long-Horizon Generation,” in NeurIPS 2024 Workshop on Open-World Agents, 2024. [Online]. Available: https://openreview.net/forum?id=5QtKMjNkjL
5. “RAT on GitHub.” [Online]. Available: https://github.com/CraftJarvis/RAT
6. A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, “Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection,” 2023, arXiv. [Online]. Available: https://paperswithcode.com/paper/self-rag-learning-to-retrieve-generate-and
7. J. Sohn et al., “Rationale-Guided Retrieval Augmented Generation for Medical Question Answering,” 2024, arXiv. [Online]. Available: https://paperswithcode.com/paper/rationale-guided-retrieval-augmented
8. Meta AI, “meta-llama/Llama-3.3-70B-Instruct,” HuggingFace. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct
9. Sentence Transformers, “all-MiniLM-L6-v2,” Hugging Face. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
10. J. Bevendorff, M. Wiegmann, M. Potthast, and B. Stein, “Is Google Getting Worse? A Longitudinal Investigation of SEO Spam in Search Engines,” in Advances in Information Retrieval, N. Goharian, N. Tonellotto, Y. He, A. Lipani, G. McDonald, C. Macdonald, and I. Ounis, Eds., Cham: Springer Nature Switzerland, 2024, pp. 56–71.
11. Google, “Safe Browsing Lookup API (v4),” Google Developers. [Online]. Available: https://developers.google.com/safe-browsing/v4/lookup-api
12. M. Siddhartha, “Malicious URLs dataset,” Kaggle. [Online]. Available: https://paperswithcode.com/dataset/malicious-urls-dataset
13. Cross Encoder, “ms-marco-MiniLM-L6-v2,” Hugging Face. [Online]. Available: https://huggingface.co/cross-encoder/ms-marco-MiniLM-L6-v2
14. “MS MARCO Scores - Pretrained Models,” Sentence Transformers. [Online]. Available: https://sbert.net/docs/cross_encoder/pretrained_models.html#ms-marco
15. Deepseek AI, “DeepSeek-R1-Distill-Qwen-1.5B,” Hugging Face. [Online]. Available: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
16. E. J. Hu et al., “LoRA: Low-Rank Adaptation of Large Language Models,” 2021, arXiv. [Online]. Available: http://arxiv.org/abs/2106.09685
17. OpenAI, “HumanEval.” Hugging Face, 2023. [Online]. Available: https://huggingface.co/datasets/openai/openai_humaneval
18. Google Research, “MBPP.” Hugging Face, 2023. [Online]. Available: https://huggingface.co/datasets/google-research-datasets/mbpp
19. OpenAI, “GSM8K.” Hugging Face, 2021. [Online]. Available: https://huggingface.co/datasets/openai/gsm8k

20. S. Kulal et al., “SPoC: Search-based Pseudocode to Code,” in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/7298332f04ac004a0ca44cc69ecf6f6b-Paper.pdf
21. P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “SQuAD: 100,000+ Questions for Machine Comprehension of Text,” in Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, J. Su, K. Duh, and X. Carreras, Eds., Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. doi: 10.18653/v1/D16-1264.
22. Microsoft, “MS MARCO (v2.1).” Hugging Face, 2018. [Online]. Available: https://huggingface.co/datasets/microsoft/ms_marcoms-marco
23. Cross Encoder, “ms-marco-MiniLM-L4-v2,” Hugging Face. [Online]. Available: https://huggingface.co/cross-encoder/ms-marco-MiniLM-L4-v2
24. Mixedbread AI, “mxbai-rerank-xsmall-v1.” Hugging Face, 2024. [Online]. Available: https://huggingface.co/mixedbread-ai/mxbai-rerank-xsmall-v1

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.