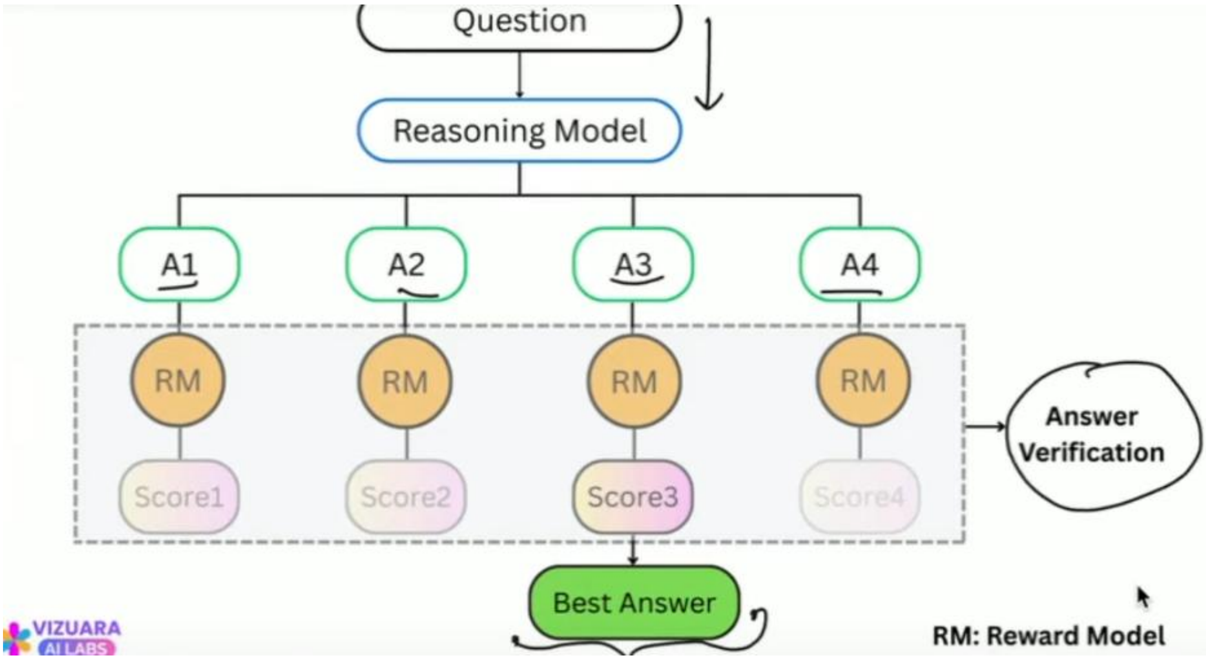
# MODULE 1: Inference Time Compute Scaling - Part 2

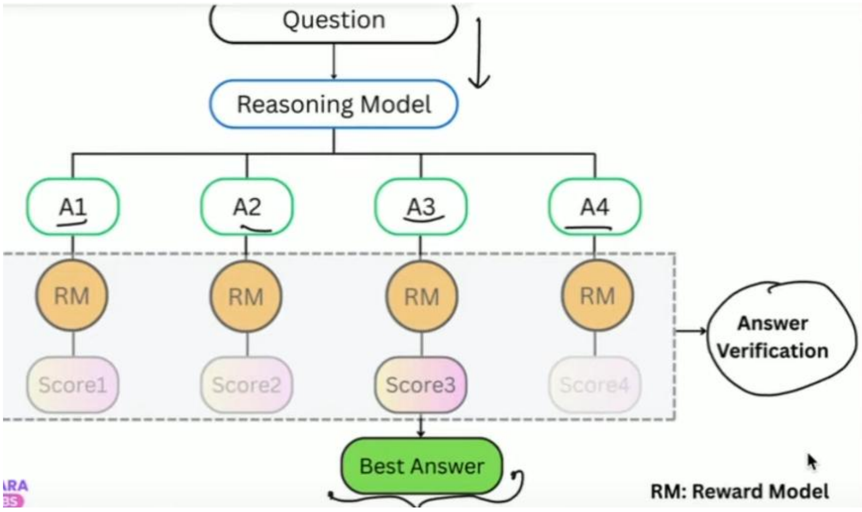
In the previous lecture, we discussed about the methods of Chain of Thought reasoning and zero-shot reasoning. Both these methods are 'prompting' techniques which are used to nudge the LLM to reason before providing an answer.

Today, we will understand about a second category of test-time compute which is called as 'Search against Verifiers'.

## CATEGORY 2: SEARCH AGAINST VERIFIERS

The following diagram explains the basic premise for a verifier:-

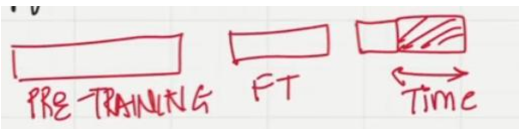




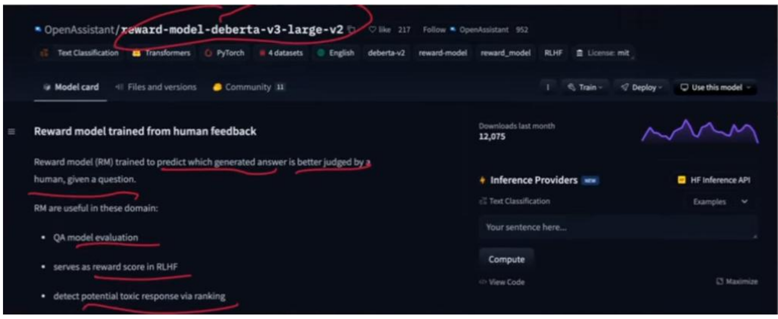

Reasoning Layer

Verification Layer

Analogy: Imagine that you are given the task of selecting the best quality crop in a field. You would pick each crop and verify its quality, only then you would choose the crop. You would not directly pick one crop and be okay with it.




Verification can be done either by humans or by a model. The models who do the verification are called reward models.

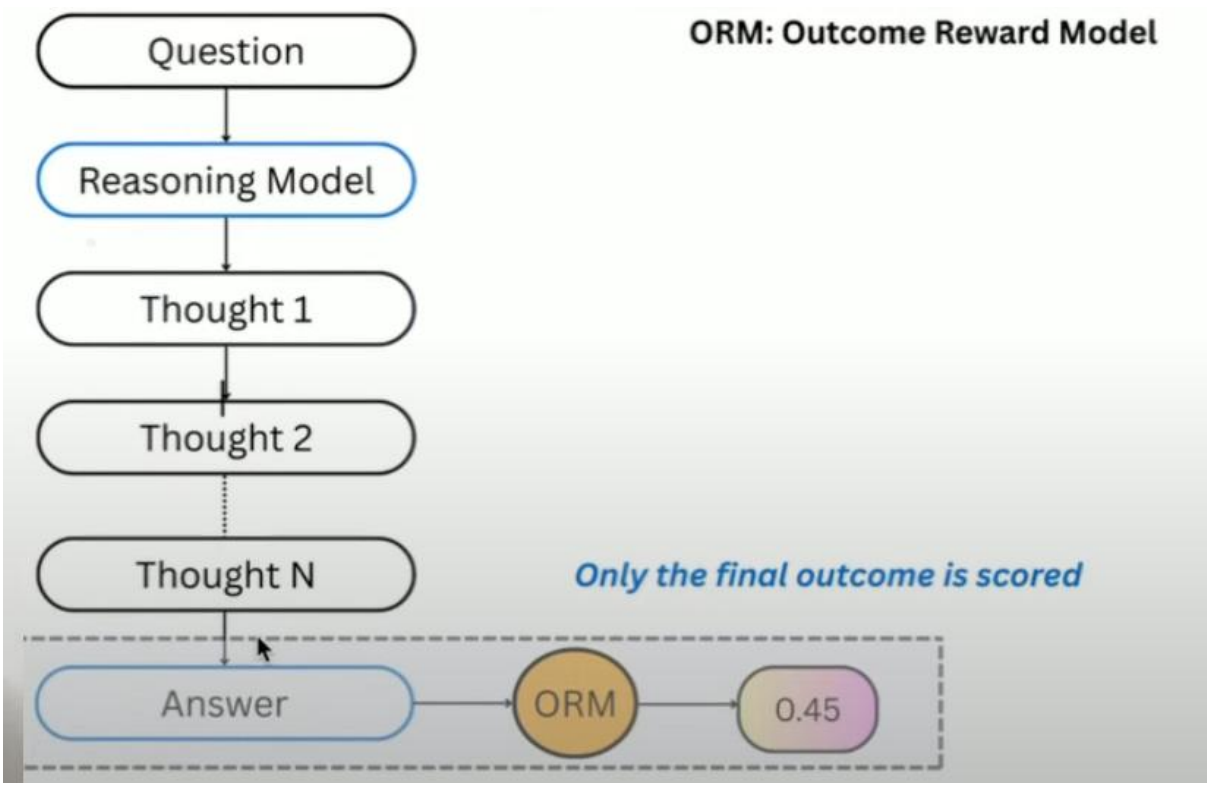



Example of a reward model on Huggingface.

Let us look at reward models in detail now:-

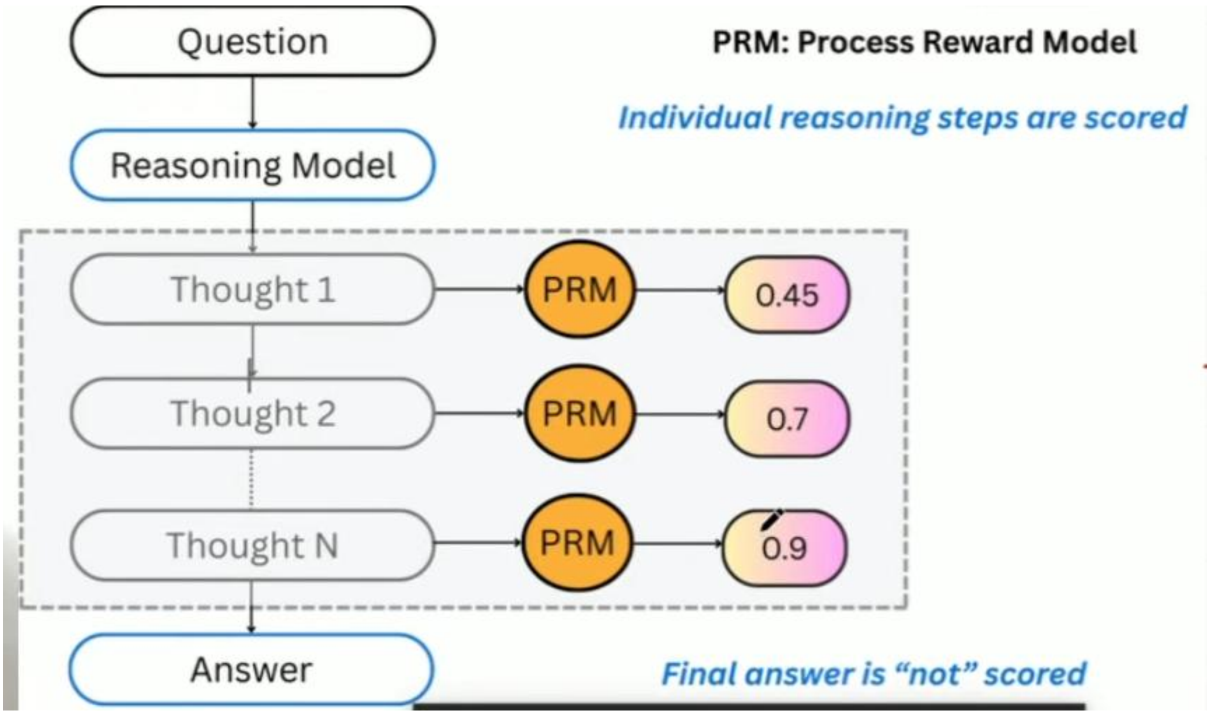
There are 2 types of reward models: (1) Outcome Reward models and (2) Process Reward models.

# Outcome Reward Models:-



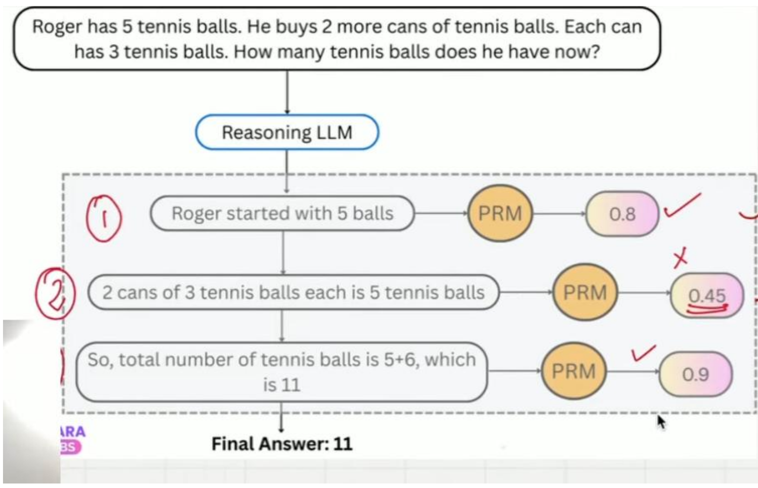

→ Only the final outcome scored.

## Process Reward Models:-




→ Individual reasoning steps scored.

## Let us look at an example of Process Reward Model (PRM):-




→ Correct reasoning gets higher reward

→ Wrong reasoning gets lower reward

The main advantage of using verifiers is that there is no need to fine tune or retrain the LLM which you are using to answer the question.

TYPES OF VERIFIERS:-

We will discuss 3 types of verifiers in this lecture:-

(1) Majority Voting

(2) Best of N

(3) Beam Search

Google DeepMind 2024-8-7

# Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

**Charlie Snell**[♦, 1], **Jaehoon Lee**[2], **Kelvin Xu**[♣, 2] **and Aviral Kumar**[♣, 2]

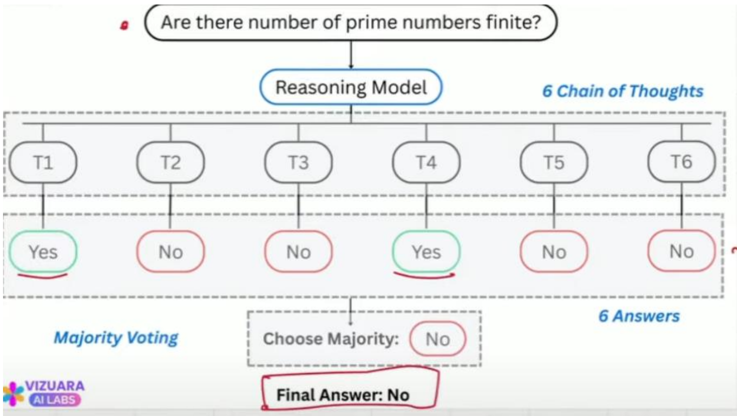
[♣]Equal advising, [1]UC Berkeley, [2]Google DeepMind, [♦]Work done during an internship at Google DeepMind

**Enabling LLMs to improve their outputs by using more test-time computation is a critical step towards building generally self-improving agents that can operate on open-ended natural language. In this paper, we study the scaling of inference-time computation in LLMs, with a focus on answering the question: *if an LLM is allowed to use a fixed but non-trivial amount of inference-time compute, how much can it improve its performance on a challenging prompt?* Answering this question has implications not only on the achievable performance of LLMs, but also on the future of LLM pretraining and how one should tradeoff inference-time and pre-training compute. Despite its importance, little research attempted to understand the scaling behaviors of various test-time inference methods. Moreover, current work largely provides negative results for a number of these strategies. In this work, we analyze two primary mechanisms to scale test-time computation: (1) searching against dense, process-based verifier reward models; and (2) updating the model's distribution over a response adaptively, given the prompt at test time. We find that in both cases, the effectiveness of different approaches to scaling test-time compute critically varies depending on the difficulty of the prompt. This observation motivates applying a "compute-optimal" scaling strategy, which acts to most effectively allocate test-time compute adaptively per prompt. Using this compute-optimal strategy, we can improve the efficiency of test-time compute scaling by more than 4× compared to a best-of-N baseline. Additionally, in a FLOPs-matched evaluation, we find that on problems where a smaller base model attains somewhat non-trivial success rates, test-time compute can be used to outperform a 14× larger model.**

## Majority Voting:

In this method, range of answers are generated by the Large Language model. Whichever answers appears the maximum number of times is selected as the final answer.

So, a verifier is not even required in this method.




6 chain of thoughts are generated.

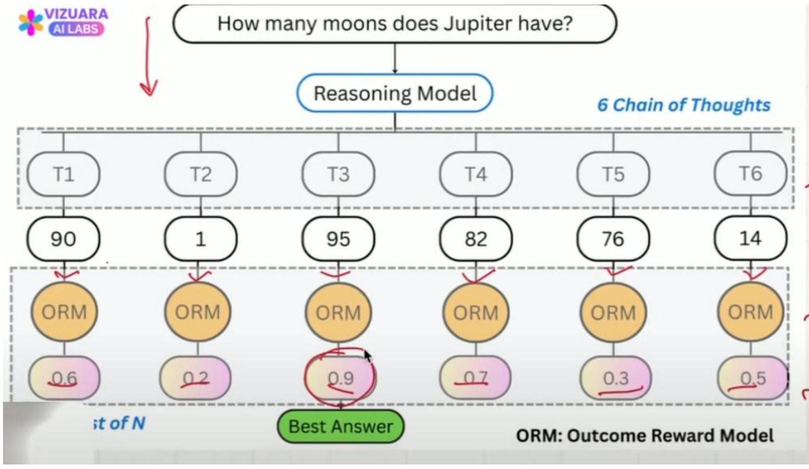
'No' appears maximum (4 times)

This method is also called as 'self-consistency'.

## Best-of-N-samples:

In this method, N samples are generated by the LLM. Then, the verifier chooses the best sample. This method can be done by using either an Outcome Reward Model (ORM) or a Process Reward Model (PRM)

## Using ORM for Best-of-N Sampling:-

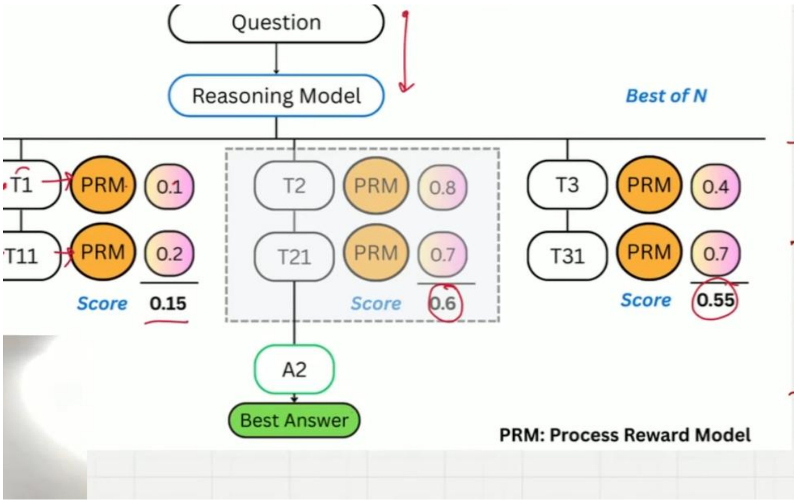



→ 6 samples generated.

→ Verifier scores 6 answ

→ Answer with best su
is selected (95)

## Using PRM for Best-of-N-Sampling:




→ 3 samples generated

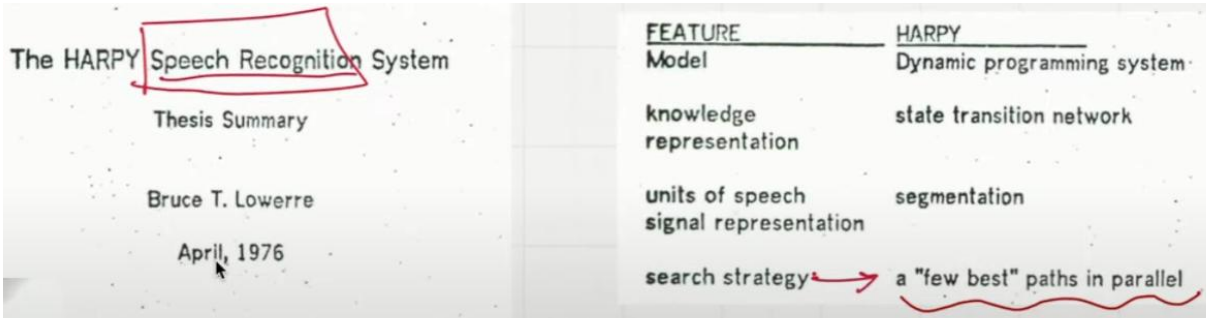
→ PRM evaluates each ste

→ Average from all steps do

→ Answer with best average is selected.

## BEAM SEARCH:-

The beam search algorithm was originally proposed in 1976 in the following paper:-


The HARPY Speech Recognition System

Thesis Summary

Bruce T. Lowerre

April, 1976

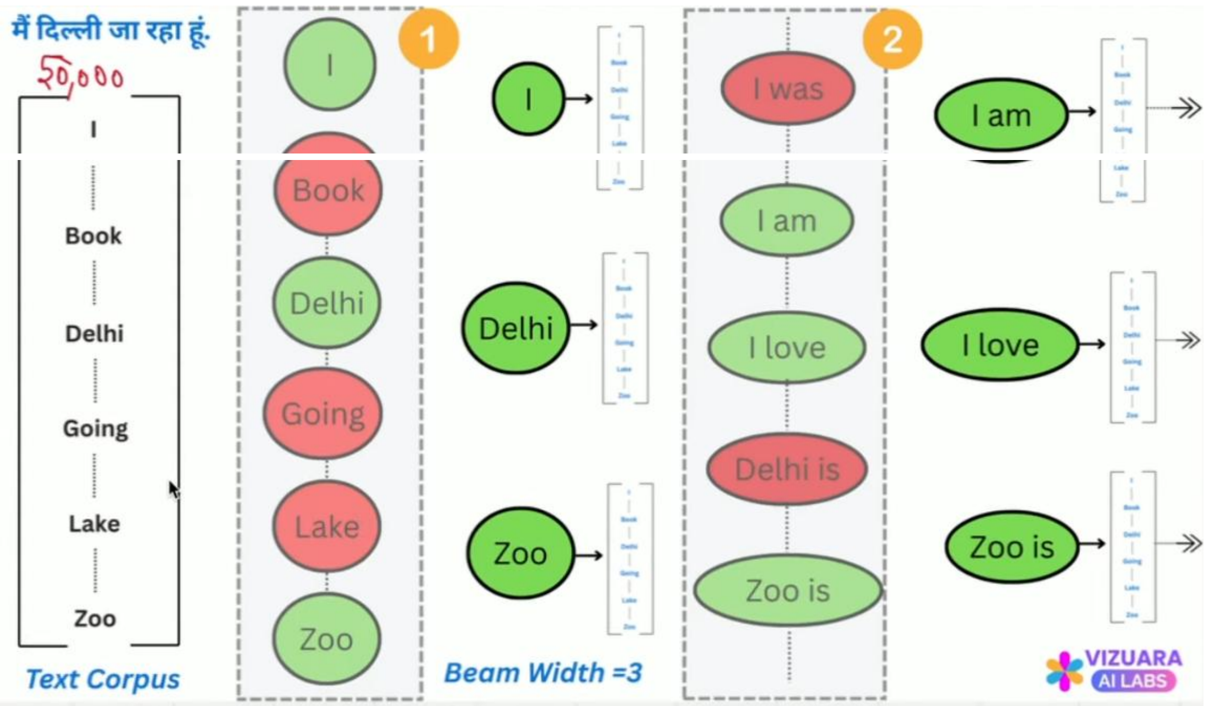
| FEATURE | HARPY |
|---|---|
| Model | Dynamic programming system |
| knowledge representation | state transition network |
| units of speech signal representation | segmentation |
| search strategy → | a "few best" paths in parallel |

We will first understand the basics of beam search and then understand how it is used with reward models.

We will take a simple example. We will learn how sentences are translated using beam search.

Sentence: "मैं दिल्ली जा रहा हूं"

Task: Translate this sentence to English.




Step 0: We are given a text corpus. A list of 50000 words. We have to use only the words in the text corpus for the translation task.

Step 1: We will choose the 3 most probable words which can represent the 1st word in the translated sentence. These words are "I", "Delhi" and "Zoo".

Step 2: For each of these words, we will select the most probable combination of the 1st word and 2nd word in the translated sentence. Now, we select the best 3. These are:
"I am", "I love" and "Zoo is".
Note that, we can eliminate Delhi as the first word at this stage.

Step 3: For each of these pair of words, we select the most probable combination of the combination of the 3 words. Then, we again select the best 3. This is how the process continues.

Here "3" is the beam width.

As we saw from the above example, beam search allows us to explore multiple paths before we come to the answer.

et us see how beam search can be combined with a reasoning LLM a reward model to search many different reasoning paths and e up with an optimum answer.

Let us take this example:-

Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

Let the total number of beams be 4. So, in the first step, the model will generate 4 different thoughts:-

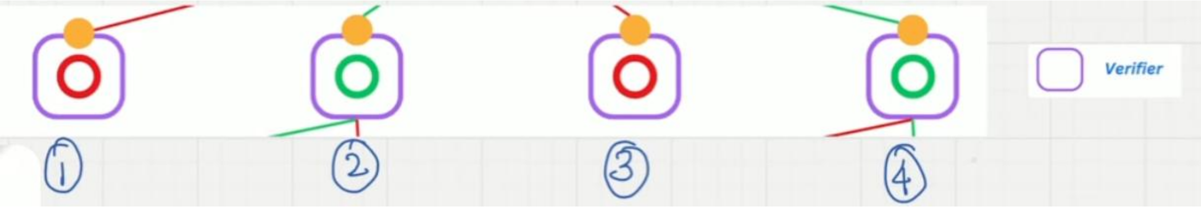
Let us take this example:-

Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

No. of beams → 4

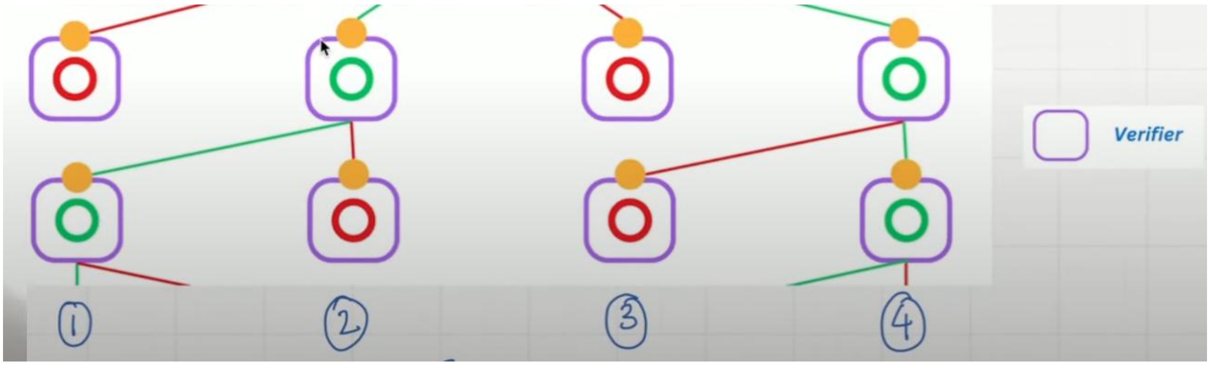
Beam width → 2

Let the total number of beams be 4. So, in the first step, the model will generate 4 different thoughts:-




Out of these 4 thoughts, 2 will be selected (Beam width = 2). These 2 are marked in green.

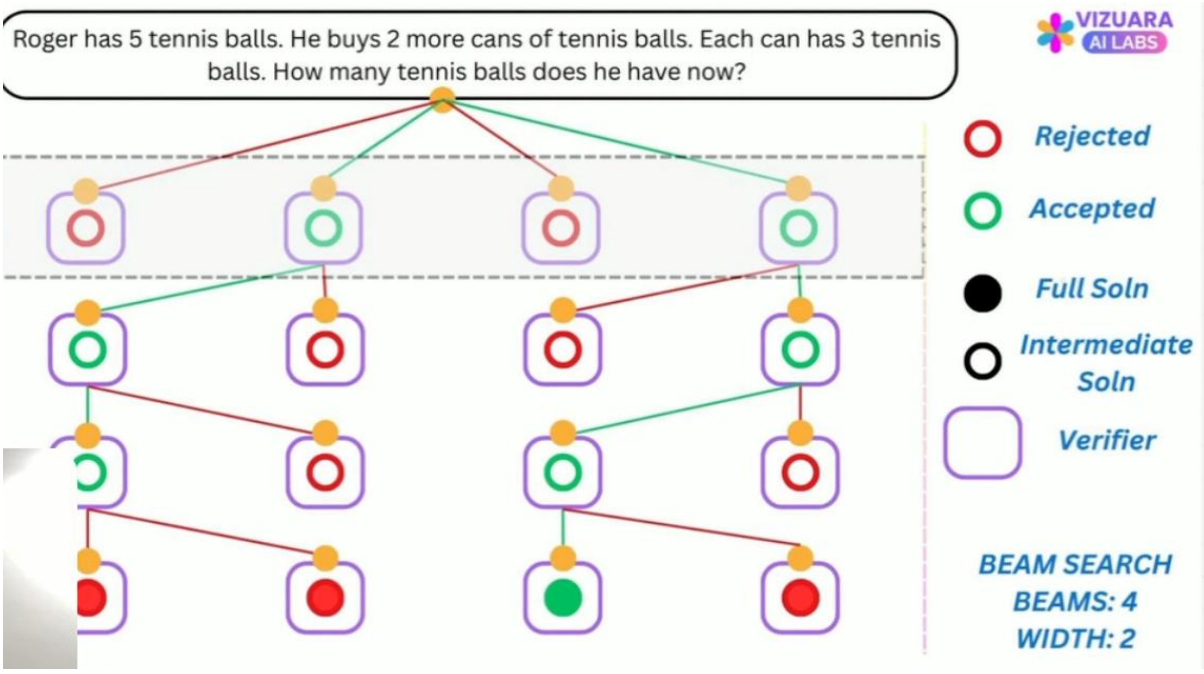
From these 2 thoughts, 2 thoughts each will be selected for the next reasoning step. This is because, at each step, we need a total of 4 thoughts (number of beams = 4)




gain 2 thoughts (marked in green) will be selected and from

n, 2 thoughts will be selected like before.

The final diagram for the beam search looks like the following:-




Note that, at each step there are 4 solutions, out of which 2 (marked in green) are selected.

✏ Hands-on Implementation of Beam-Search with Reward Models:

## Expected Output:-