ETS: Efficient Tree Search for Inference-Time Scaling

Hooper, Coleman, et al. "ETS: Efficient Tree Search for Inference-Time Scaling." *ArXiv.org*, 2025

Seobin Lee

Contents

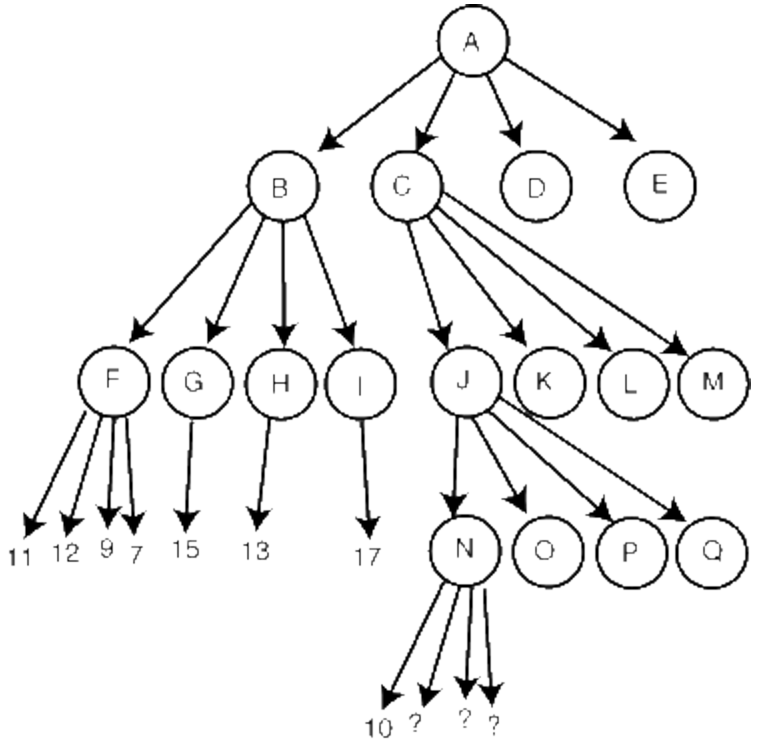
Background

- Test-time scaling is important!
  - A smaller model with increased compute allocation at test time can outperform a larger pretrained model

- How to do this?
  - cf. iterative self-refinement
  - Generate multiple candidate solutions and evaluate them using a scoring criterion, either after sampling or progressively during the search, to guide the selection toward high-quality final responses
    - Best-of-N, tree search
    - Typical scoring method: using a Process Reward Model (PRM)

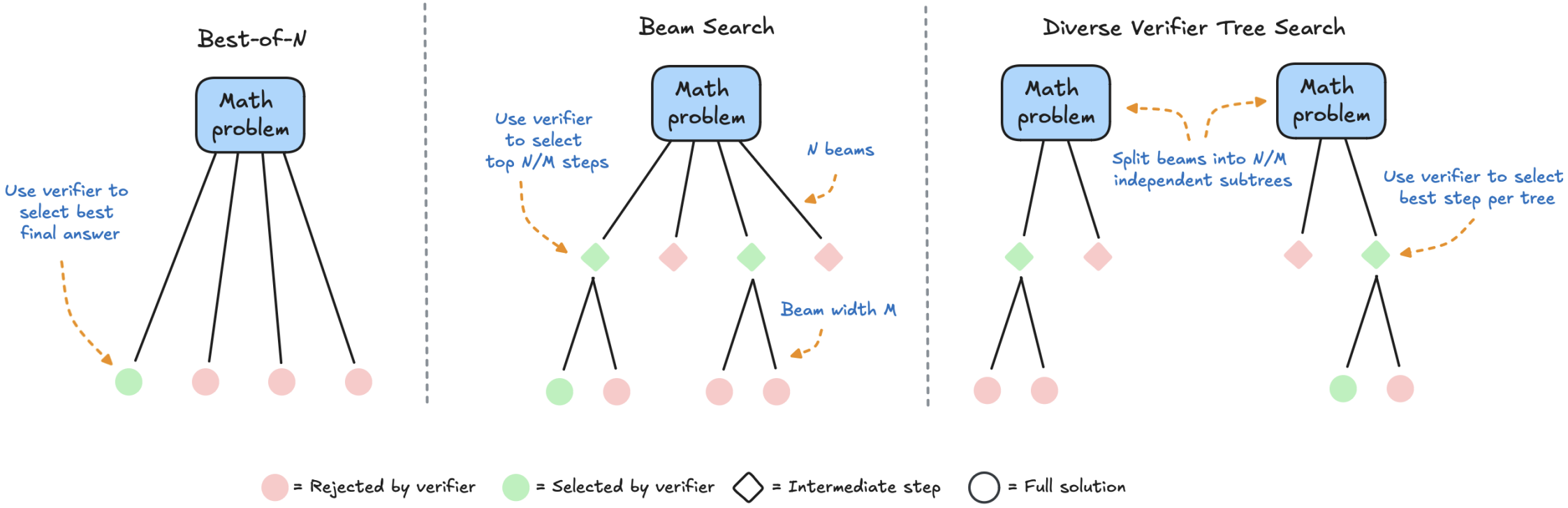
Background

- Tree: undirected graph in which any two vertices are connected by exactly one path



4 Background

- Tree search:



Ref: https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute

Background

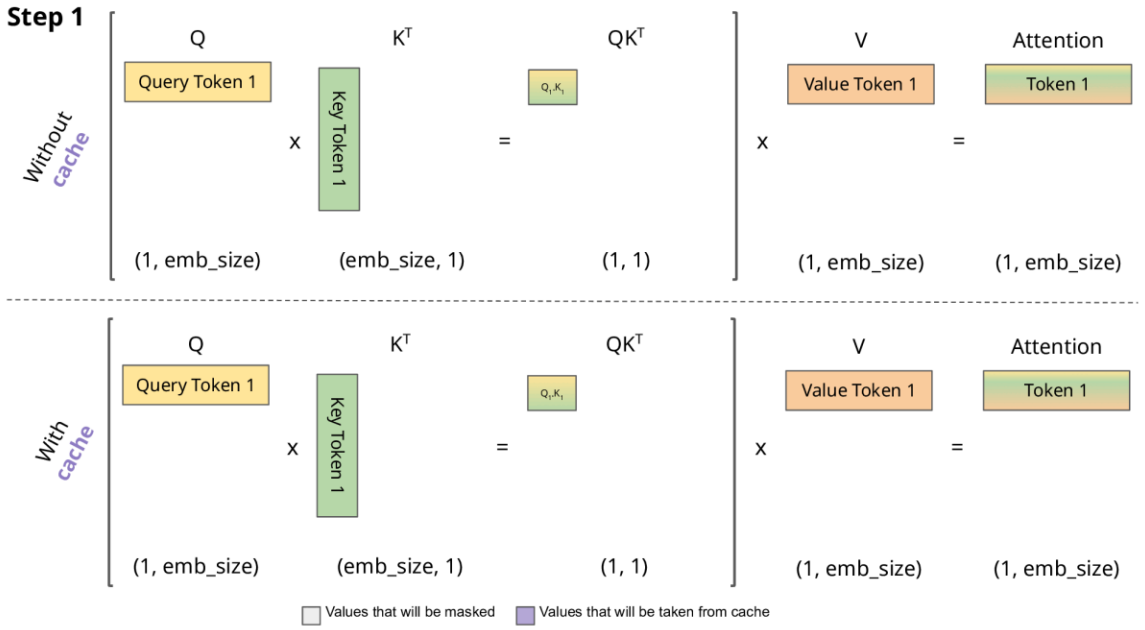
- However, the problem is…
  - The increased inference costs are substantial, as many trajectories must be generated and scored before a final answer is selected

Background

- Then, our goal must be to extend test-time while performing floating point operations efficiently
  - **No!**
  - The actual inference costs of search are not necessarily proportional to the number of model calls or FLOPs
    - LLM inference is typically memory bandwidth-bound
    - The number of memory operations required for search is not necessarily correlated with FLOPs

Background

- LLM inference is typically memory bandwidth-bound
  - Dominant contributor: model weights, **KV cache**



Ref: https://medium.com/@joaolages/kv-caching-explained-276520203249

Background

- LLM inference is typically memory bandwidth-bound
  - Dominant contributor: model weights, **KV cache**
  - For short sequence lengths, the model weights are typically the dominant contributor to memory consumption and bandwidth
  - For longer context lengths and batched inference, the KV cache becomes the main memory bottleneck

Background

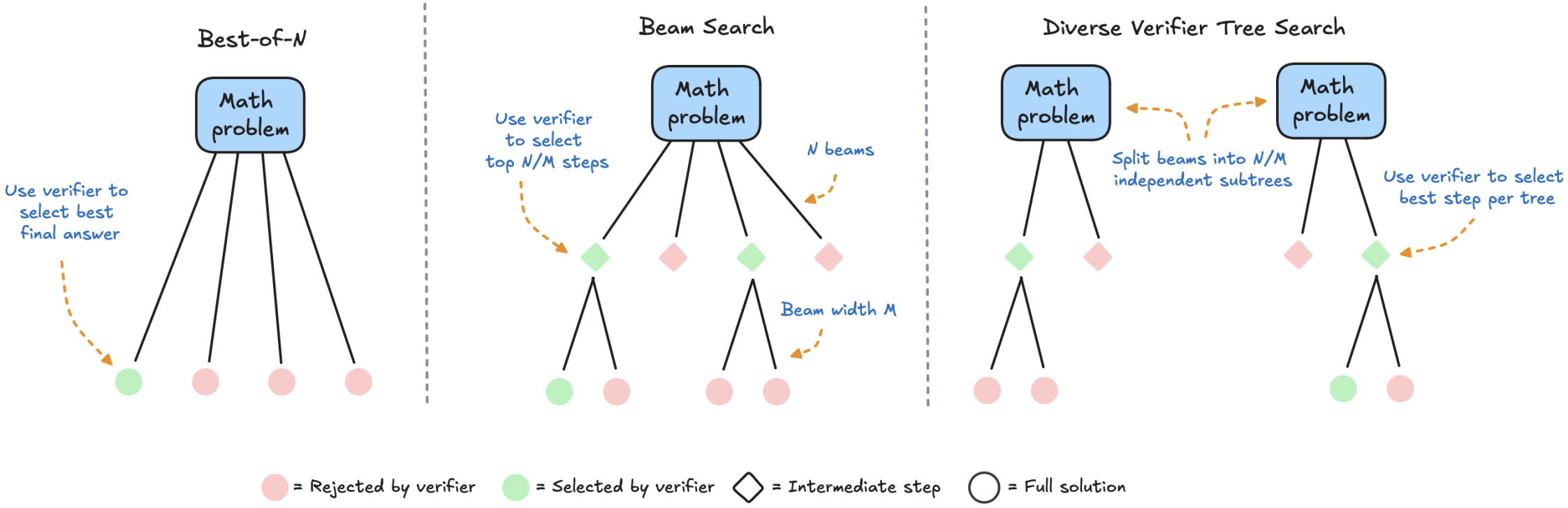
- For performing tree search, KV cache sharing between trajectories has a substantial impact on the memory consumption and bandwidth
  - With two trajectories which share the KV cache for most of their previous steps, this will require substantially fewer memory operations when performing further generations than if each trajectory has an entirely separate KV cache state

- If the KV cache for the sequences is too large to fit in memory, then the number of sequences that can be run in parallel will be constrained and the search process gets fragmented into multiple successive iterations
  - It leads to performing more memory operations for the model weights, since the model weights need to be loaded for each fragment.

- While the KV cache state can be reused for earlier steps in the search when generating later steps, if the memory requirements are too great, then the KV cache for the earlier steps would be de-allocated and would need to be recomputed, which would increase latency

Background

- Tree search:



Ref: https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute

Background

- The capacity to perform extensive KV-cache sharing implies that search trajectories within the tree overlap significantly
- There exists a **trade-off** between promoting diverse trajectories for higher accuracy and maximizing KV-cache reuse
- Finding the **right balance in this trade-off** is crucial for both efficiency and performance

Method

- Measure the efficiency costs and benefits of retaining…
  - a set of trajectories **vs** each separate trajectory
  - Reason: the efficiency cost of retaining each trajectory depends on the other trajectories that are retained or pruned

Method

- Reward term:

$$\frac{\sum_{i \in S} W_i}{\sum_{i \in A} W_i}$$

  - $W_i = \text{ceil}\left(N \frac{\exp(R_i / T_R)}{\sum_{k \in A} \exp(R_k / T_R)}\right)$ (weight of a trajectory i)
  - N: the total number of continuations that need to be sampled
  - R_i: the reward for trajectory i computed using a PRM
  - T_R: a temperature parameter that controls how balanced the sampling is
  - A: the set of all trajectories

- We want to find a subset $S \subset A$ that achieves high $\frac{\sum_{i \in S} W_i}{\sum_{i \in A} W_i}$

Method

- Cost term:

$$\frac{|V_S|}{|V_A|}$$

  - V_A: the set of all nodes in the tree before pruning the tree
  - V_S: the set of all nodes in the tree after pruning the tree

- We want to find a subset $S \subset A$ that achieves low $\frac{|V_S|}{|V_A|}$

Method

- Since we have two optimization goals:

$$\max_{S}\left(\frac{\sum_{i\in S} W_i}{\sum_{i\in A} W_i} - \lambda_b \frac{|V_S|}{|V_A|}\right)$$

  - $\lambda_b > 0$ is a hyperparameter
  - Add the additional constraint that $|S| \geq 1$ to ensure that we always retain at least one leaf node

- Is that enough?
  - According to the authors, **no!**

Method

- If we naively enforce KV sharing, this will also consequently prune out diverse trajectories which were crucial for attaining high accuracy

- Existing approaches for sampling continuations lead to many redundant or similar continuations
  - Even if some of these continuations are not exact duplicates, they can still have a similar semantic meaning

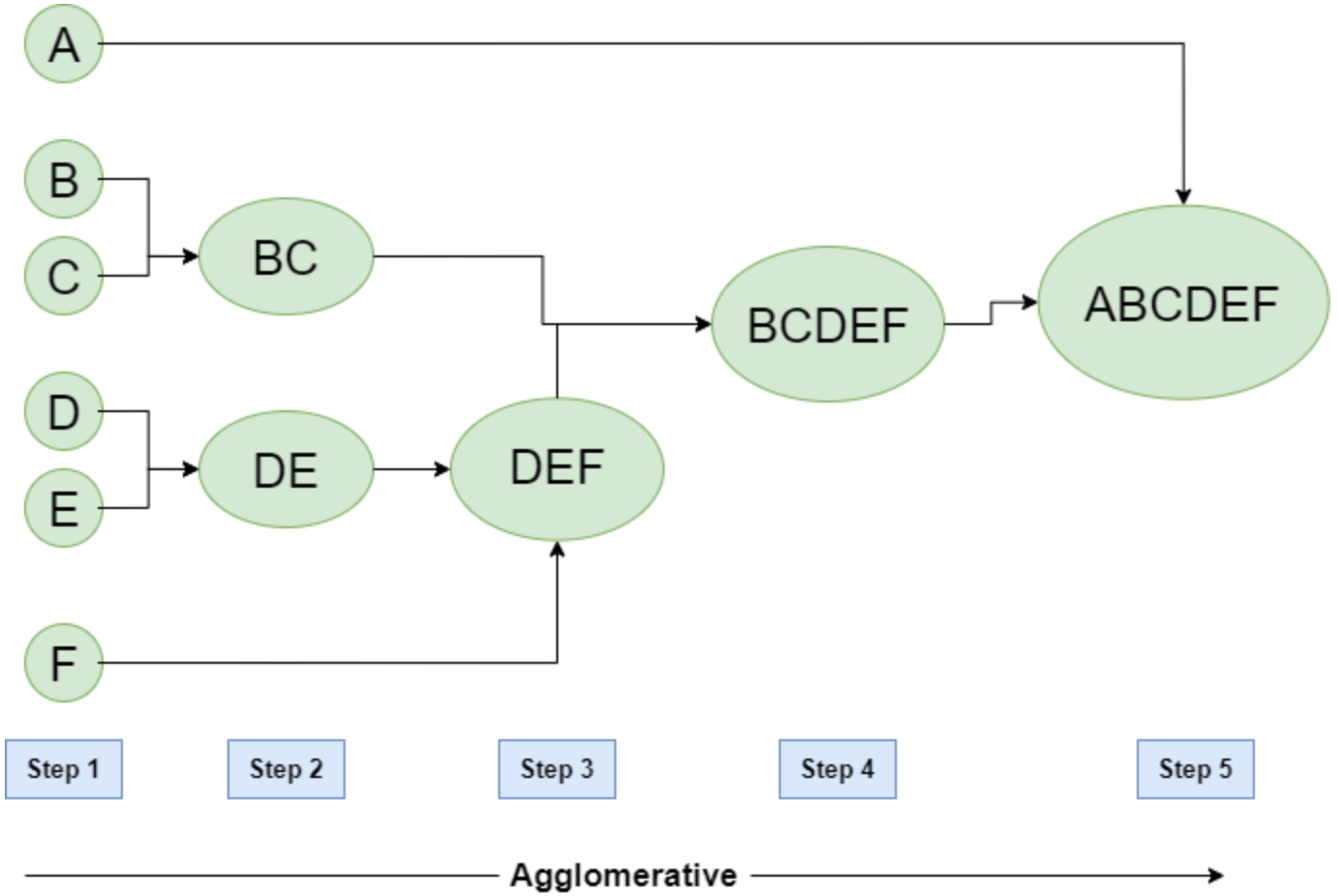
- The objective is to preserve maximal coverage of the original semantic space after pruning

Method

- However, we can only estimate whether retaining a given trajectory improves coverage when considering the other trajectories that are retained

- Solution:
  - Embed the last step for each sequence using a BERT model finetuned for embedding math sentences
  - Cluster these embeddings using hierarchical agglomerative clustering from Scipy based on cosine similarity

Method



Ref: https://www.geeksforgeeks.org/hierarchical-clustering/

Method

- Diversity term:

$$\frac{|C_S|}{|C_A|}$$

  - C_A: the total number of clusters
  - C_S: the clusters covered by the selected trajectories

Method

- Final optimization problem:

$$\max_{S}\left(\frac{\sum_{i\in S} W_i}{\sum_{i\in A} W_i} - \lambda_b \frac{|V_S|}{|V_A|} + \lambda_d \frac{|C_S|}{|C_A|}\right)$$

- It is an integer linear programming problem, so they leveraged the Pulp optimization library using the CBC solver

Method

- After pruning out trajectories, how many continuations to sample from each of the remaining trajectories determines by:

$$W_i' = \text{ceil}\left(N \frac{\exp(R_i/T_R)}{\sum_{k \in S} \exp(R_k/T_R)}\right)$$

Baseline

- REBASE
  - Given the question x, balance temperature $T_b > 0$, and sampling number of solutions N, it samples N instances of the first step for the question
    - Let the sampling budget of depth 0, B_0, to N at initialization
  - In the i-th iteration, the PRM assigns the rewards to all the nodes at depth i
    - After that, the algorithm examines whether the solutions up to depth i are complete
    - Supposing there are C_i completed solutions, it updates the sampling budget using $B_i \leftarrow B_{i-1} - C_i$
      - If $B_i = 0$, the process ends, and it obtains N solutions
    - For all the nodes n_j with reward $R(n_j)$ in the depth i of the tree, it calculates the expansion width of the n_j as:

$$W_j = \mathrm{Round}\left(B_i \frac{\exp\left(R(n_j)/T_R\right)}{\sum_k \exp(R(n_k)/T_R)}\right)$$

Baseline

- REBASE
  - When the balance temperature T_b is small, this method encourages more exploitation
  - When T_b is large, it encourages exploration
  - This method produces more diverse trajectories than standard beam search and attains higher accuracy for the same efficiency budget defined by FLOPs or number of model calls
  - It has substantial inference overheads due to the reduction in KV cache sharing from sampling in a more balanced manner when deciding which trajectories to expand

Experiments

- As mentioned previously, FLOP count is not a good metric for memory-bound problems and may even be misleading

- Experiment:
  - They profiled throughput on 100 samples from the MATH500 test set on NVIDIA H100 NVL GPUs, with the Llemma-34B model and Llemma-Reward 34B PRM each on a separate GPU
  - Results were collected by running evaluation using 8 parallel threads
    - It is analogous to the attainable throughput with a batch size of 8 for serving use-cases
  - To estimate FLOPs, they leveraged the approximation that the number of FLOPs is proportional to the number of tokens generated, which holds for short context length

Experiments

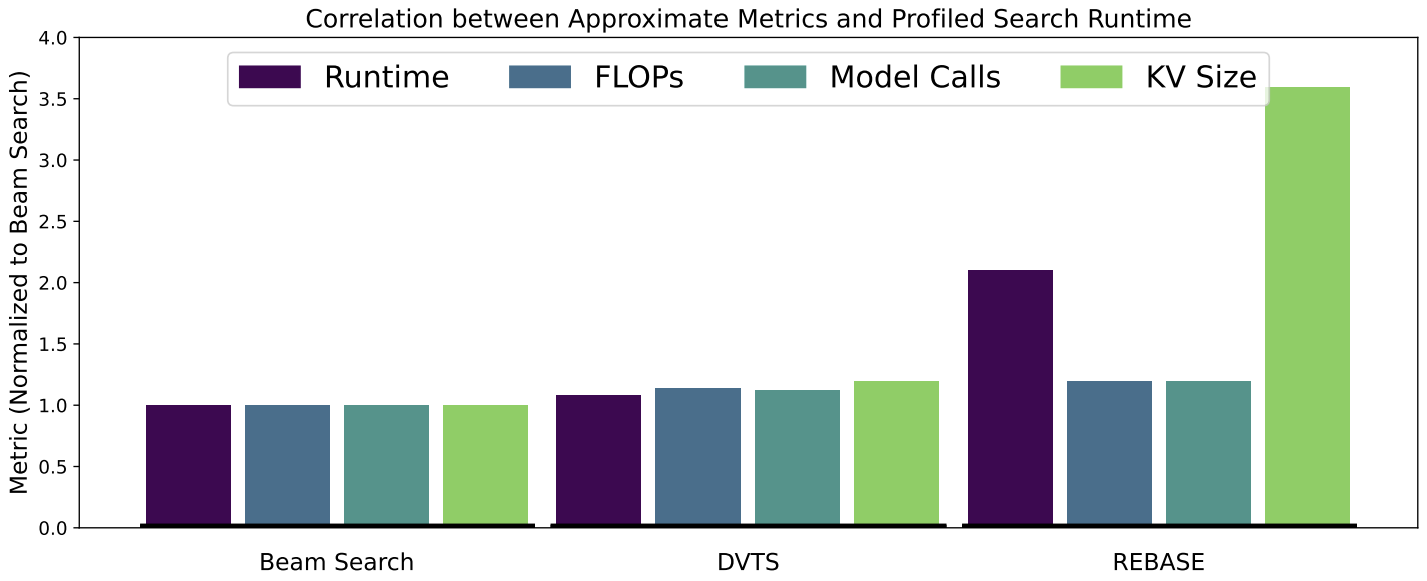


Figure 2. Correlation between approximate efficiency metrics and profiled runtime. We report profiled Runtime as well FLOPs), number of model calls, and total KV Cache Size ("KV Size") as well as profiled runtime. We measure each metric for Beam Search, DVTS, and REBASE for the Llemma-34B model with a width of 256, and we report each metric normalized to the value for Beam Search. For Beam Search and DVTS, we retain $\sqrt{N}$ trajectories at each step, where N is the width of the search. As can be seen, REBASE has similar FLOPs and number of model calls compared to beam search and DVTS, but it exhibits significantly higher runtime. The increased runtime is due to its increased KV cache size. This clearly shows that FLOPs and number of model calls are not necessarily the right proxy metrics to use when assessing search efficiency.

Experiments

- They leveraged the open-source REBASE code for the balanced sampling implementation

- They used SGLang to serve KV cache reuse

Experiments

```
dimensions = ["Clarity", "Originality", "Evidence"]

@function
def multi_dimensional_judge(s, path, essay):
  s += system("Evaluate an essay about an image.")
  s += user(image(path) + "Essay:" + essay)
  s += assistant("Sure!")

  # Return directly if it is not related
  s += user("Is the essay related to the image?")
  s += assistant(select("related", choices=["yes", "no"]))
  if s["related"] == "no": return

  # Judge multiple dimensions in parallel
  forks = s.fork(len(dimensions))
  for f, dim in zip(forks, dimensions):
    f += user("Evaluate based on the following dimension:" +
      dim + ". End your judgment with the word 'END'")
    f += assistant("Judgment:" + gen("judgment", stop="END"))

  # Merge the judgments
  judgment = "\n".join(f["judgment"] for f in forks)

  # Generate a summary and a grade. Return in the JSON format.
  s += user("Provide the judgment, summary, and a letter grade")
  s += assistant(judgment + "In summary," + gen("summary", stop=".")
                 + "The grade of it is" + gen("grade"))

  schema = r'\{"summary": "[\w\d\s]+\.", "grade": "[ABCD][+-]?"\}'
  s += user("Return in the JSON format.")
  s += assistant(gen("output", regex=schema))

state = multi_dimensional_judge.run(...)
print(state["output"])
```

Handle chat template and multi-modal inputs

Select an option with the highest probability

Fetch result; Use Python control flow

Runtime optimization: KV Cache Reuse (Sec. 3)

Multiple generation calls run in parallel

Fetch generation results

Runtime optimization: API speculative execution (Sec. 5)

Runtime optimization: fast constrained decoding (Sec. 4)

Run an SGLang program

Figure 2: The implementation of a multi-dimensional essay judge in SGLang utilizes the branch-solve-merge prompting technique [40]. Primitives provided by SGLang are shown in red.

Experiments

- Hyperparameter
  - For the proposed method:
    - $T_R = 1.0$
    - $\lambda_d = 1$
    - They sweeped over $\lambda_b \in [1,2]$ and select the largest value of λ_b which doesn't degrade accuracy by greater than 0.2%
  - For the REBASE:
    - $T_R = 0.2$ (default setting)

Experiments

- Model
  - Llemma-34B model fine-tuned on Metamath along with the Llemma-34B PRM from the REBASE paper
  - Mistral-7B model fine-tuned on Metamath as well as the corresponding Mistral-7B PRM from the Math-Shepherd paper

    cf. Math-Shepherd paper: it aims to automate the data generation process to facilitate training PRMs

Experiments

- Search width
  - 16, 64, 256
  - For the proposed method & REBASE:
    - As in the REBASE paper, they reduced the search width each time a retained trajectory completes
  - For the beam search & DVTS:
    - The label '(fixed)' indicates that 4 trajectories were retained at every step, while '(root(N))' denotes that $\sqrt{N}$ trajectories were retained at each step, where N is the initial width of the search

Experiments

- Datasets:
  - MATH500, GSM8K

- Select the final answer with weighted majority voting using the final PRM score for each trajectory as the weight

Experiments

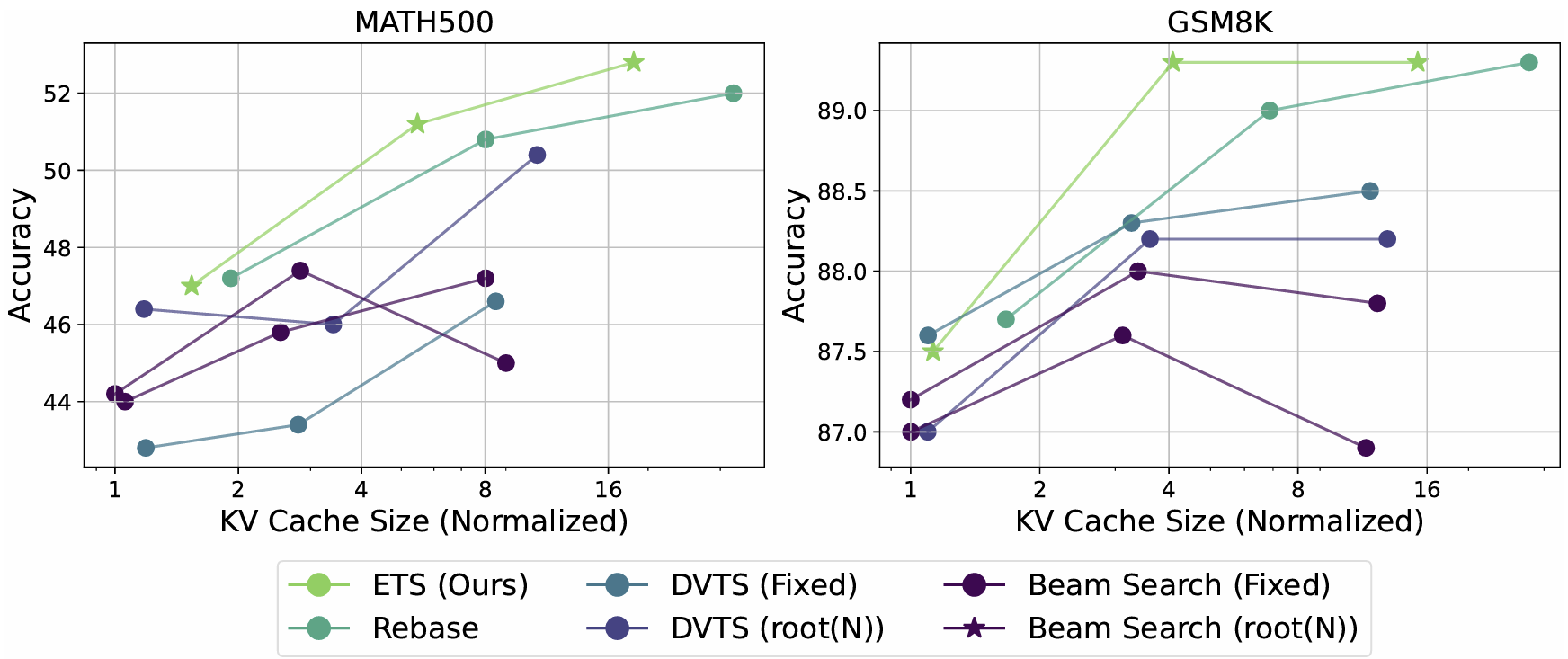


Figure 3. Accuracy versus efficiency trade-off curves for different search strategies with the Llemma-34B model. We report results for search widths of 16, 64, and 256 across all methods. We provide baseline results for Beam Search and DVTS (both with retaining a fixed number of trajectories as well as $\sqrt{N}$ trajectories at each step) (Snell et al., 2024; Beeching et al., 2024), as well as for REBASE (Wu et al., 2024). Our results demonstrate how our method allows for improved efficiency relative to REBASE, while maintaining the accuracy benefits due to retaining necessary diverse trajectories.

Experiments

Table 1. Accuracy versus KV cache size for REBASE as well as ETS. Results are provided for MATH500 and GSM8K for the Llemma-34B and Mistral-7B-SFT models. We report the KV cache size reduction ("KV Red.") for each width (relative to REBASE), where higher is better since it implies a reduction in memory consumption.

Method	Width=16		Width=64		Width=256	
	Acc.	KV Red.	Acc.	KV Red.	Acc.	KV Red.
Llemma-34B						
REBASE	47.2	1×	50.8	1×	52.0	1×
ETS	**47.0**	**1.2×**	**51.2**	**1.5×**	**52.8**	**1.8×**
Mistral-7B-SFT						
REBASE	38.8	1×	43.4	1×	42.4	1×
ETS	**39.4**	**1.3×**	**43.2**	**1.3×**	**42.2**	**1.7×**

MATH500

Method	Width=16		Width=64		Width=256	
	Acc.	KV Red.	Acc.	KV Red.	Acc.	KV Red.
Llemma-34B						
REBASE	87.7	1×	89.0	1×	89.3	1×
ETS	**87.5**	**1.5×**	**89.3**	**1.7×**	**89.3**	**1.8×**
Mistral-7B-SFT						
REBASE	88.6	1×	89.1	1×	90.1	1×
ETS	**88.3**	**1.2×**	**89.2**	**1.6×**	**89.6**	**1.3×**

GSM8K

Experiments

- They measured throughput for REBASE as well as the proposed method on H100 NVL GPUs
  - Benchmark [4,8,16,32] parallel threads and select the best configuration for each method
    - The number of parallel threads is representative for the serving scenario with a batch size equal to the number of threads
  - They ran benchmarking with the main LLM (Llemma-34Bmodel) and the PRM each on a separate H100 NVL GPU, and for the proposed method, they co-located the embedding model on the same GPU as the reward model
  - Beam width: 256
  - Dataset: MATH500 test set

Experiments

Table 2. Throughput for our approach relative to REBASE (Wu et al., 2024). Results were measured on NVIDIA H100 GPUs using the Llemma-34B model, evaluated on 100 samples from the MATH500 test set (with the accuracy reported for the full test set). We report throughput improvements using a beam width of 256. We also include the reduction in KV cache size (normalized to REBASE), as well as the accuracy for each approach.

Method	Accuracy	KV Reduction	Throughput
REBASE	52.0	1×	1×
ETS	**52.8**	**1.8**×	**1.4**×

Experiments

- Ablation study
  - Impact of the diversity term:
    - For the results with only KV cache sharing term, they fix $\lambda_d = 0$ and sweep over $\lambda_b \in [0.75, 1.25]$ selecting the largest value of λ_b which doesn't degrade accuracy by greater than 0.2%

Experiments

Table 3. Ablation for our methodology. We include results on MATH500 for different beam widths with the Llemma-34B model, and we report KV budget estimates. We compare ETS with only applying the KV budget term in the cost model ("ETS-KV").

Method	Width=16		Width=64		Width=256	
	Acc.	KV Red.	Acc.	KV Red.	Acc.	KV Red.
REBASE	47.2	1×	50.8	1×	52.0	1×
ETS-KV	**47.2**	**1.3**×	51.4	1.3×	52.8	1.7×
ETS	47.0	1.2×	**51.2**	**1.5**×	**52.8**	**1.8**×

Conclusion

- They proposed a tree search method that balances accuracy and efficiency by promoting KV cache sharing while preserving semantic diversity
- Unlike prior approaches, the proposed method explicitly models KV-sharing costs, going beyond standard efficiency metrics like FLOPs
- The proposed method with beam width 256 achieves 1.8× smaller KV cache size and 1.4× higher throughput than REBASE, with minimal accuracy degradation and without requiring custom kernel implementations

Limitations

- For Table 3, it is only at a beam width of 256 that the diversity term shows a clearly positive effect

- From the perspective of service providers who need to support many users simultaneously, it is unclear whether the proposed method is practical
  - This raises the question of whether the method can have meaningful industrial impact given its computational demands

- One of the known issues with beam search is that it tends to produce repetitive responses
  - Although the proposed method introduces a diversity term into the optimization, it remains fundamentally a variant of beam search, leaving open the question of how effectively it mitigates this inherent flaw

Thank You!