PATHFINDER: Guided Search over Multi-Step Reasoning Paths

Olga Golovneva[*], **Sean O'Brien, Ramakanth Pasunuru, Tianlu Wang, Luke Zettlemoyer, Maryam Fazel-Zarandi, Asli Celikyilmaz**
FAIR at Meta

Abstract

With recent advancements in large language models, methods like chain-of-thought prompting to elicit reasoning chains have been shown to improve results on reasoning tasks. However, tasks that require multiple steps of reasoning still pose significant challenges to state-of-the-art models. Drawing inspiration from the beam search algorithm, we propose PATHFINDER, a tree-search-based reasoning path generation approach. It enhances diverse branching and multi-hop reasoning through the integration of dynamic decoding, enabled by varying sampling methods and parameters. Using constrained reasoning, PATHFINDER integrates novel quality constraints, pruning, and exploration methods to enhance the efficiency and the quality of generation. Moreover, it includes scoring and ranking features to improve candidate selection. Our approach outperforms competitive baselines on three complex arithmetic and commonsense reasoning tasks by 6% on average. Our model generalizes well to longer, unseen reasoning chains, reflecting similar complexities to beam search with large branching factors.

1 Introduction

Recent progress in large language models (LLMs) has led to a new era in machine reasoning, particularly through the use of prompting methods. These methods, such as chain-of-thought (CoT) Wei et al. (2022), scratchpads Nye et al. (2021), least-to-most Zhou et al. (2023), and program-aided language models (PAL) Gao et al. (2023), typically break down complex tasks into reasoning chains and have shown to improve model performance on tasks such as logical Clark et al. (2020), arithmetic Cobbe et al. (2021) and commonsense Talmor et al. (2021) reasoning.

As tasks that require multiple steps of reasoning become more complex, LLMs begin to struggle with accumulating errors across multiple reasoning steps. Even more challenging is ensuring that each step in a chain is correctly evaluated and contributes positively to the overall reasoning chain and accuracy of the solution. To address these issues, recent work has implemented methods like self-consistency for majority voting Wang et al. (2023), diversifying prompts Li et al. (2023) and Python programs for more accurate reasoning generations Gao et al. (2023). Despite these improvements, the process of creating reasoning chains as a standard autoregressive process still faces challenges due to large search space, sub-optimal assessment and guidance of the reasoning process, especially in complex, multi-step tasks.

In this work, we introduce PATHFINDER, a decoding method designed for the generation and refinement of reasoning chains generated by LLMs. PATHFINDER embodies our approach of dividing the reasoning decoding into two distinct tasks: *candidate generation* and *candidate selection*. For the candidate generation process, PATHFINDER employs a tree-search-based method. It integrates a set of constraints to improve the quality of generated reasoning candidates, along with a pruning

[*]corresponding author at olggol@meta.com

R0-FoMo: Workshop on Robustness of Few-shot and Zero-shot Learning in Foundation Models at NeurIPS 2023.

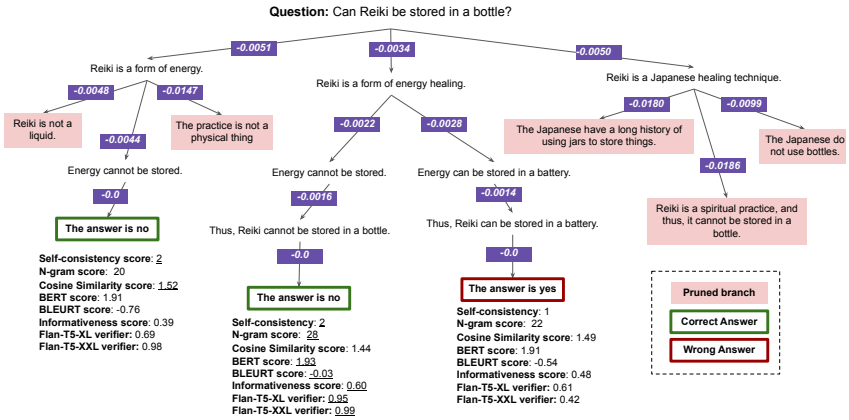

Figure 1: PATHFINDER leverages step-level generic constrained decoding to guide step-by-step reasoning generations. In this example from the StrategyQA dataset, although the reasoning steps are close in n-grams, PATHFINDER prunes less likely branches and chooses more informative ones that explain the question. Branching factor=3, buffer size=3. Numbers in purple rectangles are scores produced by the pruning function governed by Equation 2 using greedy step decoding ($\tau \rightarrow 0$). Highest scores produced for each non-pruned branch are underlined. Details on different candidate selection scores at the leaf of each reasoning path are provided in Section 4.

function for efficient computation and removes subpar candidates as shown in Figure 1. Our model also incorporates an exploration factor to ensure the diversity of reasoning generations. For the candidate selection process, PATHFINDER utilizes a set of novel similarity-based functions that we benchmark against existing LLM-based verifiers. This selection process allows for the selection of more accurate reasoning chains from the candidate pool, thereby refining the quality of the overall reasoning. We conduct extensive experiments across four generation tasks that necessitate multi-step reasoning. Using the small-size LLAMA-7B (Touvron et al., 2023) as a backbone language model, PATHFINDER demonstrates substantial performance improvements across all tasks, highlighting its effectiveness in improving reasoning capabilities of language models. We discuss related research in more detail in the Appendix C.

In summary, we develop PATHFINDER, a new decoding method for effective generation of reasoning traces. Our algorithm is versatile, as it can be applied to a variety of multi-step reasoning generation tasks via decoding time constraints, reducing the need for tuning on costly labeled data. Our extensive experiments demonstrate its effectiveness on several complex tasks.

2 PATHFINDER: Reasoning Decoder

We describe PATHFINDER, a tree-search-based reasoning path generation approach. We first introduce the decoding problem and then describe our approach which incorporates a two step decoding process: *candidate generation* and *selection*.

Decoding. Sequence generation is a task of generating output sequence $\mathbf{y}$ (e.g., reasoning path) given input sequence $\mathbf{x}$ (e.g., question, prompt, etc.). In multi-step reasoning from LLMs, a sequential reasoning chain composed of T steps is generated across multiple timesteps. We denote a reasoning chain as $\mathbf{y} = y_{1:T}$=$[y_1, y_2, \cdots, y_T]$, where each y_t denotes a reasoning step of sequence of distinct tokens. A reasoning chain is autoregressively generated and the decoding consists of solving:

$$\mathbf{y}_* = \arg\max_{\mathbf{y} \in \mathcal{Y}} \log P(\mathbf{y}|\mathbf{x}) \tag{1}$$

where $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_K\}$ is the set of all generated reasoning paths in response to the input $\mathbf{x}$ using generation model $P(\mathbf{y}|\mathbf{x})$.

Candidate Generation. At the core of PATHFINDER is a tree-search algorithm for long form reasoning generation, which generates multiple plausible generation paths. Unlike token-based beam search methods Holtzman et al. (2020), the branching in our approach occurs at the level of reasoning

steps instead of individual tokens. This means that each reasoning step is regarded as a discrete node (see Figure 1). The branching is influenced by varying among a fixed number of sampling parameters (e.g., top-*k*, top-*p*, temperature), allowing PATHFINDER to explore various decoding methods and enabling search over different decoding strategies at each time step. This dynamic decoding property facilitates multi-hop reasoning, resulting in diverse branches.

To trade off quality against compute, we draw a number of candidates from each non-pruned leaf within the reasoning tree as our branching factor at every stage of the process. We continue to sample until all leaves have either reached termination points or have achieved a predefined maximum depth. To avoid over-generation of reasoning step branches, we also introduce a buffer size b, limiting the number of hypothesises stored for each context, and implement pruning methods. In particular, for each hypothesis reasoning step y_t= $[y_t^1, \cdots, y_t^N]$, where y_t^i is the ith token in the sequence on length N, generated in response to the prompt $\mathbf{x}$, we prune branches based on the sequence scores , normalized by the number of tokens:

$$\pi(y_t) = \sum_i \log p_\theta(y_t^i|\mathbf{x}, y_t^{i<})/N^\lambda \tag{2}$$

where $\lambda \in \mathbb{R}$ is a model-specific length penalty parameter , and p_θ is a token generation model. Additionally, similar to Xie et al. (2023), we introduce step sampling temperature τ with annealing factor α used to decay temperature step-by-step as $\tau \rightarrow \alpha\tau$ to add controlled variation in the branches, and sample according to the distribution:

$$p(y_t) \propto \exp(\pi(y_t)/\tau) \tag{3}$$

Candidate Generation Constraints. We enforce additional constraints on reasoning steps to reduce hallucinations. In particular, we force the model to re-generate a reasoning step if one of the following conditions is satisfied: (1) *Repetition constraint*: generated step is similar to one of the previously generated steps or repeats context as determined by the cosine similarity metric. Cosine similarity is computed using the sentence embedding model *all-mpnet-base-v2* HuggingFace, and we force the model to re-generate if similarity value is greater than 0.9; (2) *Contradiction constraint*: generated step contradicts the context (i.e., question and previous steps) as determined by an entailment model. Specifically, we use the model proposed by Laurer et al. (2022) to classify the step into one of three classes: *entailment*, *neutral*, and *contradiction*, and force the model to re-generate if it belongs to the *contradiction* class as determined by the entailment model. If no new steps are generated after $n = 2$ attempts, the branch is pruned.

Candidate Selection. To select a final hypothesis out of a pool of candidates, we experiment with a number of scoring functions of the form:

$$\mathbf{y}_* = \underset{\mathbf{y}_j \in \mathcal{Y}}{\arg\max} \sum_{\mathbf{y}_k \in \mathcal{Y}, \mathbf{y}_k \neq \mathbf{y}_j} S(\mathbf{y}_j, \mathbf{y}_k) \tag{4}$$

where the number of candidate reasoning chains in the pool $\mathcal{Y}$ is limited by the buffer size ($K \leq b$), and S is a similarity function. The intuition is similar to *self-consistency* (Wang et al., 2022) or *wisdom of the crowd* (Suzgun et al., 2022), in the assumption that a solution following from more diverse, generated reasoning chains majority is more likely to be the correct one. In fact, our results support the use of an *N-gram*-based similarity metric . Specifically, if g_j is a set of n-grams for the hypothesis $\mathbf{y}_j$, the *N-gram* similarity function is defined as the number of common n-grams as follows:

$$S(\mathbf{y}_j, \mathbf{y}_k) = |g_j \cap g_k| \tag{5}$$

Candidate selection is a critical component of PATHFINDER. Common techniques are using scorer functions and verifier models. Scorer functions Suzgun et al. (2022); Prasad et al. (2023) help rank fixed set of candidate generations and guide the selection of the final prediction based on some property of the generated text, such as similarity. On the other hand verifier models Li et al. (2023) use external models to explicitly evaluate the correctness of the produced hypothesis, and rank generations based on the faithfulness score. We validate PATHFINDER against verifier models and different similarity-based scorers in our ablation studies in Section 4. We note that the usage of a suitable scoring function is preferred over a verifier model as it would improve runtime and reduce memory load.

Model	GSM8K	StrategyQA	CSQA
GPT-6.7B	2.4	50.0	24.0
MINERVA-8B	**16.2**	-	-
LLAMA-7B	11.0	61.1*	43.3*
LLAMA-7B (self consistency)	15.3*	64.8*	46.9*
FLAN-T5-XL (3B)	13.5	**73.4***	**85.4***
PATHFINDER (LLAMA-7B, N-gram)	11.3	59.0	50.0
PATHFINDER (LLAMA-7B, FLAN-T5-XL)	11.7	60.8	55.1
PATHFINDER (LLAMA-7B, TEXT-DAVINCI-003)	15.4	61.7	56.3

Table 1: Performance of different models on four reasoning benchmark datasets measured with accuracy. Best numbers are **bolded** among models and the second best numbers are underlined. Numbers with an asterisk* are from our evaluations using greedy decoding and CoT prompts provided in Appendix D. For self-consistency scores, we marginalize answer across 16 reasoning chains sampled with temperature $T = 1.0$, *top-k* ($k = 40$) and *top-p* ($p = 0.5$). We note that FLAN-T5 is finetuned on data from both the CSQA and GSM8K datasets and thus will have somewhat inflated performance in comparison to comparably-sized models not trained on these tasks.

3 Experiments: Reasoning Generation

Datasets. We conduct experiments on various benchmark datasets that require complex reasoning skills to reach the final answer: (1) GSM8K (Cobbe et al., 2021), an arithmetic reasoning dataset of 8.5K linguistically diverse grade school math word problems; (2) STRATEGYQA (Geva et al., 2021), a commonsense reasoning dataset of 2,780 questions, annotated with their decomposition and per-step evidence; (3) CSQA (Talmor et al., 2018), a multiple-choice commonsense reasoning dataset of 12,102 questions with one correct answer and four distractor answers.

Backbone LLMs for PATHFINDER. We select two widely-used open-sourced models to generate and evaluate chains of reasoning: LLAMA-7B (Touvron et al., 2023) and FLAN-T5-XL (3B) (Chung et al., 2022). We prompt LLAMA-7B model with chain-of-thought examples (Wei et al., 2022) to generate reasoning steps along with the final answers. We provide specific parameter values and prompt sequences in Appendix D. We also experiment with different methods for candidate selection. In particular, we report results using the following setups: (1) PATHFINDER (LLAMA-7B, N-gram): uses LLAMA-7B model for text generation, and tri-gram similarity for candidate selection; (2) PATHFINDER (LLAMA-7B, FLAN-T5-XL): uses LLAMA-7B model for text generation, and FLAN-T5-XL verifier model for candidate selection; (3) PATHFINDER (LLAMA-7B, TEXT-DAVINCI-003) uses LLaMa-7B model for text generation, and TEXT-DAVINCI-003 verifier model from the family of GPT-3.5 models for candidate selection.

Baselines. We benchmark our approach against leading best models with reported results in the literature, ensuring the model sizes are comparable for fair evaluation[2]. Specifically we compare against GPT-6.7B Wei et al. (2022), LLAMA-7B (Touvron et al., 2023), FLAN-T5-XL (3B) (Fu et al., 2023), and Minerva-8B Lewkowycz et al. (2022). Reported results represent evaluation results on generations produced with CoT prompting and greedy token-level decoding. We also include our own evaluations on a few tasks that to the best of our knowledge are missing in the literature using greedy decoding and prompts provided in Appendix D.

Results. Table 1 compares different LLMs with different decoding methods showing answer accuracy as the evaluation metric. PATHFINDER improves baseline performance on all selected reasoning tasks by 6% on average, but lacks behind the base model with self-consistency applied on STRATEGYQA dataset by 3%. Even simple N-gram-based similarity metric allows to select better paths leading to model improvements with respect to baseline on GSM8K and CSQA datasets. We note that FLAN-T5-XL verifier significantly improves performance on CSQA task, but not that much on the others. This is likely due to the fact that is was trained on this task, while other tasks are significantly harder to evaluate (GSM8K), or not familiar to the model (STRATEGYQA). While overall TEXT-

[2]We also attempted to generate results using the self-evaluation guided decoding method described in Xie et al. (2023) for the LLaMa-7B model. However, the process was slow and computationally expensive: Reducing the beam size to 2 and number of samples at each step to 4, this method took over 4 days to run and achieved an accuracy of 9.5 on GSM8K and 58.3 on StrategyQA.

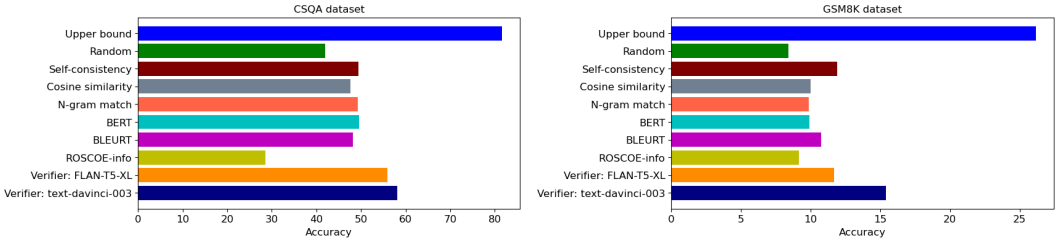

Figure 2: The impact of different similarity-based scoring functions and verifier models (vertical axis) on the accuracy score (horizontal axis) of PATHFINDER, utilizing LLAMA-7B as backbone LLMs, on the CSQA and GSM8K datasets. We use buffer size 128 with branching factor of 8 for CSQA, and buffer size 16 with branching factor of 4 for GSM8K dataset. Scoring functions score and rank hypothesises based on similarity metrics, while verifier model ranks hypothesises based on the generated faithfulness score.

DAVINCI-003 verifier shows better performance, there is a trade-off between the amount of resources needed to run GPT3.5-based evaluations and improvements in performance it could give; Figure 1 shows an example of the generated tree for one of the STRATEGYQA questions. It showcases how PATHFINDER that although the reasoning steps are close in N-grams, PATHFINDER prunes less likely branches and chooses more informative ones that explain the question. Finally, the N-gram scorer selects the correct answer by selecting the branch with higher n-gram similarity to other branches.

4 Ablation Study

How do various candidate selection strategies impact the overall performance? In this section, we evaluate our model using various scorer functions and verifier models, which rank a fixed set of candidates and select the highest-scoring one as the final prediction. We present an upper bound accuracy, which is an accuracy of the "perfect" scorer that would select the correct final answer if present in candidate pool, and contrast the *N-gram*-based scorer with several alternative approaches: *Self-Consistency scorer*, *Cosine Similarity scorer*, BERT *and* BLEURT *scorers*, *Informativeness scorer*, and *Verifier models*. We provide more details on scorer functions construction and ranking methods in Appendix D and Appendix E. We summarize the results in Figure 2. All scorers outperform random selection, with TEXT-DAVINCI-003 results in highest accuracy score of 58.1. At the same time we want to emphasize the gap between the upper bound accuracy and the final accuracy when a scorer function or verifier model is used to rank and select best hypothesis, which clearly shows that the right choice of the candidate selection strategy can significantly boost performance further.

How does the branching factor affect performance? The tree branching factor together with the pruning function significantly influences the diversity of candidates. Intuitively, generating more candidates increases the likelihood of producing at least one correct generation. However, as our scoring models are not perfect, a high volume of noisy candidates could confuse them and escalate the rate of false positives. We asses the *N-gram* scorer performance to comprehend the scorer sensitivity to noise. Figure 3 indicates an optimal branching factor for each buffer size, which supports our hypothesis regarding the scorer function's sensitivity to noise levels. Thus, for tree-search-based step-level decoding it is important to find an optimal value of the branching factor to balance between the diversity of the candidates and the amount of noise. Similar phenomenon was previously observed for beam search token-level decoding, where increasing the number of decoding candidates past a certain point leads to the worse generation quality (Yang et al., 2018; Koehn and Knowles, 2017).

Does an increased buffer size consistently improve performance? To answer this question we empirically investigate the *N-gram* scorer performance. The results summarized in Figure 3 reveal that for small branching factors considered in this study, the accuracy score platoes. Beyond a certain point, increasing the buffer size does not yield more generations because they are limited by the branching factor and generation constrains. In fact, for CSQA experiments shown in Figure 3, at branching factor 8 the average number of hypothesis candidates almost does not change after buffer size 32, and is around 22. The platoe point shifts higher with the increase in the number of generations per node. Throughout our experiments, we observed a consistent improvement in optimal

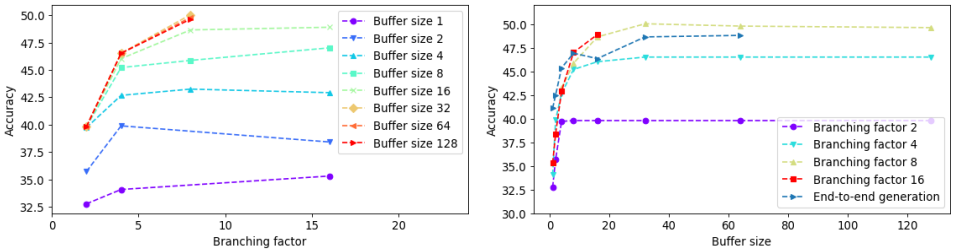

Figure 3: PATHFINDER (LLAMA-7B, N-gram) accuracy scores on CSQA dataset as a function of barnching factor (left) and buffer size (right). On the right figure we also include the results where we use LLAMA-7B to generate the whole reasoning path for each candidate, and then apply tri-gram scorer model on the number of generated candidates corresponding to the buffer size value (denoted as *end-to-end generation*). Decoding process is sensitive to the amount of noise, which results in the existence of an optimal branching factor that maximizes the final accuracy. For each branching factor there is a limit on the maximal number of diverse branches the model can generate, and increasing buffer size after this point does not result in bigger trees, and we observe a platoe in terms of accuracy scores.

		Branching Factor			
Dataset	**End-to-end**	**2**	**4**	**8**	**16**
GSM8K	0.8	0.5	1.6	5.3	-
StrategyQA	0.2	0.2	0.8	1.6	3.3
CSQA	0.07	0.06	0.2	0.6	1.4

Table 2: Average GPU-hours required LLAMA-7B model to generate a reasoning tree for different branching factors br. Batch of 16 samples was used in all datasets. All numbers are calculated for buffer size 8, we also report average GPU-hours for end-to-end generation and sampling size 8.

performance with the an increased buffer size. However, our observations are limited by available computational resources, suggesting that behavior might vary for extreme branching factors.

Do we actually need trees? In Figure 3, along with the tree-search-based results, we also report end-to-end generation performance, i.e., at candidate generation stage, instead of creating a tree we prompt the model to generate a set of reasoning chains, and then apply our candidate selection process. We observe that a sufficient diversity in reasoning steps is necessary for the tree-search-based approach to outperform the end-to-end generation method. Specifically, for the LLAMA-7B model, the tree-search generation approach only outperforms end-to-end generation at a branching factor of 8 or above. Therefore, while tree-search generation has its advantages, it's effectiveness in comparison to end-to-end generation is largely dependent on sufficient diversity in reasoning steps and a relatively high branching factor.

Computational complexity The benefits gained from this approach come at a high computational cost. Assuming a full buffer of c candidates after a few reasoning steps, we generate cb candidate new steps before selection and pruning at each step. If the original reasoning chain required T tokens to generate, PATHFINDER requires $O(bcT)$ tokens. This can be compared against self-consistency with k paths, which requires $O(kT)$ tokens, but can be more easily parallelized with memory split across devices. In practice, we see that, while a simple reasoning path takes under 1 GPU-hour to generate (Table 2), it takes several GPU-hours to generate reasoning tree that would outperform end-to-end generation for complex reasoning tasks. With more effective pruning functions and scorers, we should be able to realize gains with fewer sampled paths. Because our framework is agnostic to the choice of either, we anticipate that better models will allow for a lower branching factor and buffer size, making the method more computationally feasible.

5 Conclusion

We proposed PATHFINDER, a novel decoding method that optimizes reasoning chain generation in large langauge models. We demonstrate notable performance improvements on four multi-step reasoning tasks, emphasizing its versatility and effectiveness. Our approach overcomes traditional limitations, enhancing reasoning capabilities and opening doors for future research.

References

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2017. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945, Copenhagen, Denmark. Association for Computational Linguistics.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? FAccT '21, page 610–623, New York, NY, USA. Association for Computing Machinery.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. *IJCAI*.

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*.

Yao Fu, Litu Ou, Mingyu Chen, Yuhao Wan, Hao Peng, and Tushar Khot. 2023. Chain-of-thought hub: A continuous effort to measure large language models' reasoning performance. *arXiv preprint arXiv:2305.17306*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Olga Golovneva, Moya Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. 2022. Roscoe: A suite of metrics for scoring step-by-step reasoning. *arXiv preprint arXiv:2212.07919*.

Alex Graves. 2012. Sequence transduction with recurrent neural networks.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. *ICLR*.

HuggingFace. sentence-transformers/all-mpnet-base-v2.

Daphne Ippolito, Reno Kriz, João Sedoc, Maria Kustikova, and Chris Callison-Burch. 2019. Comparison of diverse decoding methods from conditional language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3752–3762.

Dan Jurafsky and James H. Martin. 2009. Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition. *Pearson Prentice Hall, Upper Saddle River, N.J.*

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics.

Moritz Laurer, W v Atteveldt, Andreu Casas, and Kasper Welbers. 2022. Less annotating, more classifying–addressing the data scarcity issue of supervised machine learning with deep transfer learning and bert-nli.

Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making large language models better reasoners with step-aware verifier.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4299, Online. Association for Computational Linguistics.

Kris McGuffie and Alex Newhouse. 2020. The radicalization risks of GPT-3 and advanced neural language models. *CoRR*.

Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. 2023. Locally typical sampling. *ACL*.

Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. 2019. Cgmh: Constrained sentence generation by metropolis-hastings sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6834–6842.

Maxwell I. Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. *CoRR*, abs/2112.00114.

Archiki Prasad, Swarnadeep Saha, Xiang Zhou, and Mohit Bansal. 2023. Receval: Evaluating reasoning chains via correctness and informativeness. *arXiv preprint arXiv:2304.10703*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.

Mirac Suzgun, Luke Melas-Kyriazi, and Dan Jurafsky. 2022. Follow the wisdom of the crowd: Effective text generation via minimum bayes risk decoding. *arXiv preprint arXiv:2211.07634*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Alon Talmor, Ori Yoran, Ronan Le Bras, Chandra Bhagavatula, Yoav Goldberg, Yejin Choi, and Jonathan Berant. 2021. Commonsenseqa 2.0: Exposing the limits of AI through gamification. *NeurIPS*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

Sean Welleck, Kianté Brantley, Hal Daumé Iii, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pages 6716–6726. PMLR.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. 2023. Decomposition enhances reasoning via self-evaluation guided decoding.

Yilin Yang, Liang Huang, and Mingbo Ma. 2018. Breaking the beam search curse: A study of (re-)scoring methods and stopping criteria for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3054–3059, Brussels, Belgium. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models.

A Limitations

Our study is limited by the number of models and tasks we used to empirically support the proposed approach. Although PATHFINDER outperforms other baselines on selected tasks, it comes with significant increase in computational complexity. To be able to efficiently use tree-search-based step-level decoding, we would need to develop more effective sampling and scoring techniques that will allow us to achieve high-quality results faster and at lower computational costs.

B Ethics Statement

Our method, PATHFINDER, improves text generation, specifically focused on step-by-step rationale generation from large language models. Thus, it inherits the potential benefits and risks associated with text generation applications Brown et al. (2020). By imposing logical constraints on text generation, we aim to enhance control, consistency, and accuracy, specifically in tasks that require step-by-step reasoning, such as arithmetic reasoning. We would like to note that any language model, even under constraints could potentially be exploited to produce biased, or offensive narratives McGuffie and Newhouse (2020). For in-depth exploration of these risks, we direct the reader to the analysis presented in (Bender et al., 2021).

C Related work

Decoding strategies for text generation. These methods present a continual trade-off between quality and diversity. Traditional deterministic methods such as greedy decoding and beam search Jurafsky and Martin (2009); Graves (2012) offer high-quality results but can lack diversity and are prone to

degeneration. Truncation-based sampling methods such as temperature sampling, top-*k* sampling, top-*p* sampling and locally typical sampling have been used to balance this trade-off Holtzman et al. (2020); Meister et al. (2023). The advent of autoregressive LLMs like GPT has spurred numerous works focusing on various factors such as diversity Ippolito et al. (2019), fluency Holtzman et al. (2020), and constraint satisfaction Anderson et al. (2017); Miao et al. (2019); Welleck et al. (2019); Lu et al. (2021) in decoding strategies. Constrained decoding methods have seen enhancements like grid beam search Anderson et al. (2017) and constrained beam search Hokamp and Liu (2017) that aim at satisfying lexical constraints during generation. Other works such as Metropolis-Hastings sampling-based conditional generation Miao et al. (2019) and tree-based constrained text generation Welleck et al. (2019) seek to address the mismatch between monotonic decoding and satisfying constraints. Contrary to these strategies which often struggle with the balance between quality and diversity, PATHFINDER focuses primarily on reasoning tasks and not open-ended text generation, operating on reasoning steps rather than on individual tokens. By separating out the steps of tree-search-based generation and similarity-based selection, our approach generates diverse reasoning chains and refines them for optimal quality.

Advanced CoT Strategies and Self Consistency. It has been shown that aggregating from diverse CoTs (i.e., multiple reasoning paths for each problem) can effectively enhance end-task performance Wei et al. (2022). Recent works such as self-consistency Wang et al. (2023) and crowd sampling Suzgun et al. (2022) generate multiple reasoning paths and try to find a consensus among the derived answers. Self-consistency has significantly boosted performance in CoT reasoning, even in tasks where CoT prompting traditionally harms performance; crowd sampling has shown gains in non-reasoning tasks like summarization. Our approach bears resemblance to these, but differs in its use of a tree search for candidate generation, its operation on the step level rather than on the full generations, and its use of a novel similarity function.

Other approaches offload some portion of problem-solving to external tools, like training a model to use tools as in Toolformer Schick et al. (2023) or prompting a model to solve a problem with a code interpreter as in PAL Gao et al. (2023). Our approach does not require access to external tools, although it does not prohibit their use. Further, unlike Toolformer our method is training-free.

Our work parallels the study of by Xie et al., which presents a self-evaluation guided stochastic beam search for multi-step reasoning. Their method employs beam search decoding tailored to intermediate steps and guides the searching process by controlling the error of each reasoning step to prevent potential error accumulation.

D Experimental setup

Inference parameters. To run experiments with PATHFINDER we used LLAMA-7B. For step-by-step generations, we applied temperature token sampling with $T = 1.0$, with additional *top-k* ($k = 40$) and *top-p* ($p = 0.5$) truncation to increase the diversity of the samples during tree generation. For end-to-end generation we applied greedy decoding. We fixed the maximum generation length at 128 tokens per step for tree generation, and 512 for the full reasoning generation. We run experiments on 8 GPUs with batch size 16.

Prompt construction. All prompts used for hypothesis generations are listed in Table 3. In particular, for GSM8K dataset we follow prompts from Touvron et al. (2023), for STRATEGYQA and CSQA datasets we follow Wei et al. (2022).

Pruning. In main experiments we applied annealing factor $\alpha = 0.5$, and used step sampling temperature $\tau = 1.0$. $\tau = 0$ corresponds to the maximum likelihood sampling, while $\tau \to \inf$ corresponds to the uniform sampling. This setup allows for higher variation of steps in the beginning of generation, and becomes more strict with depth. We have experimented removing annealing factor and varying $\tau = \{0, 0.5, 1.0, 16\}$ on the train partition of GSM8K dataset, and found the optimal performance at $\tau = 1$ with annealing.

Scoring functions. We contrast the *N-gram*-based scorer with several alternative approaches:

- *Self-Consistency scorer*: The final answer is determined by marginalizing out the sampled reasoning paths to find the most consistent answer in the final answer set (Wang et al., 2022). This method does not take into account reasoning chains, so we modify Equation 4 as

$\mathbf{y}_* = \arg\max_{y_j \in \mathcal{Y}} n_{a_j}$, where $\mathcal{A} = \{a_1, ..., a_b\}$ are all generated answers extracted from corresponding hypothesises $\mathcal{Y}$, and each answer a_j appears n_{a_j} times in $\mathcal{A}$.

- *Cosine Similarity scorer*: The final answer is selected by marginalizing out the total cosine similarity of the reasoning chains, so we modify the Equation 5 as $S(\mathbf{y}_j, \mathbf{y}_k) = \cos(\mathbf{e}_j, \mathbf{e}_k)$, where $\mathbf{e}_j$ is the embedding of the hypothesis reasoning path $\mathbf{y}_j$ as determined by the *all-mpnet-base-v2* HuggingFace sentence embedding model.
- BERT *and* BLEURT *scorers*: Following *wisdom of the crowd* work Suzgun et al. (2022), we try BLEURT (Sellam et al., 2020) and BERTSCORE (Zhang et al., 2019) metrics as similarity function S in Equation 4.
- *Informativeness scorer*: We select the final hypothesis based on the amount of information shared between the source context $\mathbf{c}$ and the reasoning paths, measured through mutual alignment. Specifically, we use the *Info-chain* score defined as $I(\mathbf{y}_j, \mathbf{c})$ in (Golovneva et al., 2022), and revise Equation 4 for this scorer function as $\mathbf{y}_* = \arg\max_{\mathbf{y}_j \in \mathcal{Y}} I(\mathbf{y}_j, \mathbf{c})$.
- *Verifier models*: We select the final hypothesis reasoning path and the answer based on the score provided by a pre-trained verifier model. We use FLAN-T5-XL and TEXT-DAVINCI-003 models to rank the hypothesises and select the one ranked at the top. To score the hypothesises, models are prompted to evaluate the correctness of the reasoning path, then hypothesises are ranked based on returned faithfulness score. We provide more details of the ranking method in Appendix E.

Few-shot prompts used for GSM8K dataset

Answer these questions:

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?
A: There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been 21 - 15 = 6. The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
A: There are originally 3 cars. 2 more cars arrive. 3 + 2 = 5. The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
A: Originally, Leah had 32 chocolates. Her sister had 42. So in total they had 32 + 42 = 74. After eating 35, they had 74 - 35 = 39. The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?
A: Jason started with 20 lollipops. Then he had 12 after giving some to Denny. So he gave Denny 20 - 12 = 8. The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?
A: Shawn started with 5 toys. If he got 2 toys each from his mom and dad, then that is 4 more toys. 5 + 4 = 9. The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?
A: There were originally 9 computers. For each of 4 days, 5 more computers were added. So 5 * 4 = 20 computers were added. 9 + 20 is 29. The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?
A: Michael started with 58 golf balls. After losing 23 on tuesday, he had 58 - 23 = 35. After losing 2 more, he had 35 - 2 = 33 golf balls. The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?
A: Olivia had 23 dollars. 5 bagels for 3 dollars each will be 5 x 3 = 15 dollars. So she has 23 - 15 dollars left. 23 - 15 is 8. The answer is 8.

Few-shot prompts used for StrategyQA dataset

Answer these questions:

Q: Do hamsters provide food for any animals?
A: Hamsters are prey animals. Prey are food for predators. Thus, hamsters provide food for some animals. The answer is yes.

Q: Could Brooke Shields succeed at University of Pennsylvania?
A: Brooke Shields went to Princeton University. Princeton University is about as academically rigorous as the University of Pennsylvania. Thus, Brooke Shields could also succeed at the University of Pennsylvania. The answer is yes.

Q: Yes or no: Hydrogen's atomic number squared exceeds number of Spice Girls?
A: Hydrogen has an atomic number of 1. 1 squared is 1. There are 5 Spice Girls. Thus, Hydrogen's atomic number squared is less than 5. The answer is no.

Q: Yes or no: Is it common to see frost during some college commencements?
A: College commencement ceremonies can happen in December, May, and June. December is in the winter, so there can be frost. Thus, there could be frost at some commencements. The answer is yes.

Q: Yes or no: Could a llama birth twice during War in Vietnam (1945-46)?
A: The War in Vietnam was 6 months. The gestation period for a llama is 11 months, which is more than 6 months. Thus, a llama could not give birth twice during the War in Vietnam. The answer is no.

Q: Yes or no: Would a pear sink in water?
A: The density of a pear is about 0.6g/cm3, which is less than water. Objects less dense than water float. Thus, a pear would float. The answer is no.

Few-shot prompts used for CSQA dataset

Answer these questions:

Q: What do people use to absorb extra ink from a fountain pen?
Answer Choices:
(A) shirt pocket
(B) calligrapher's hand
(C) inkwell
(D) desk drawer
(E) blotter
A: The answer must be an item that can absorb ink. Of the above choices, only blotters are used to absorb ink. The answer is E.

Q: What home entertainment equipment requires cable?
Answer Choices:
(A) radio shack
(B) substation
(C) television
(D) cabinet
A: The answer must require cable. Of the above choices, only television requires cable. The answer is C.

Q: The fox walked from the city into the forest, what was it looking for?
Answer Choices:
(A) pretty flowers
(B) hen house
(C) natural habitat
(D) storybook
A: The answer must be something in the forest. Of the above choices, only natural habitat is in the forest. The answer is B.

Q: Sammy wanted to go to where the people were. Where might he go?
Answer Choices:
(A) populated areas
(B) race track
(C) desert
(D) apartment
(E) roadblock
A: The answer must be a place with a lot of people. Of the above choices, only populated areas have a lot of people. The answer is A.

Q: Where do you put your grapes just before checking out?
Answer Choices:
(A) mouth
(B) grocery cart
(C) super market
(D) fruit basket
(E) fruit market
A: The answer should be the place where grocery items are placed before checking out. Of the above choices, grocery cart makes the most sense for holding grocery items. The answer is B.

Q: Google Maps and other highway and street GPS services have replaced what?
Answer Choices::
(A) united states
(B) mexico
(C) countryside
(D) atlas
A: The answer must be something that used to do what Google Maps and GPS services do, which is to give directions. Of the above choices, only atlases are used to give directions. The answer is D.

Q: Before getting a divorce, what did the wife feel who was doing all the work?
Answer Choices::
(A) harder
(B) anguish
(C) bitterness
(D) tears
(E) sadness
A: The answer should be the feeling of someone getting divorced who was doing all the work. Of the above choices, the closest feeling is bitterness. The answer is C.

Table 3: Prompts used for hypothesis generation per dataset.

E Verifier model

To rank the generated hypothesises we can use pre-trained LLMs to that known to be well calibrated with respect to the True/False questions, and thus were used for self-evaluation (Kadavath et al., 2022). We adopt this approach and extend it to using any external LLM model by prompting it with 5 shots of multiple-choice questions to identify if the generated reasoning is (A) correct or (B) incorrect. We use FLAN-T5 and TEXT-DAVINCI-003 models for evaluation, and extract the probability of the reasoning being correct as a faithfulness score. We note that FLAN-T5 is finetuned on the training partitions of CSQA and GSM8K, and thus will have somewhat inflated performance in comparison to comparably sized models not trained on these tasks. We follow Xie et al. (2023) and use the probability of the option A as a score to rank and select generations.