Improving Branching Language via Self-Reflection

Kolby Nottingham
University of California Irvine
Irvine, CA 92697

Rachel Dong
Riot Games
Santa Monica, CA 90064

Ben Kasper
Riot Games
Santa Monica, CA 90064

Wesley Kerr
Riot Games
Santa Monica, CA 90064

Abstract

While most language is formatted linearly, applications such as planning, trees of thought, and branching narrative are represented in a tree structure. Generating branching outputs from a language model (LM) is trivial, but representing trees of text in a one dimensional input is problematic. This makes popular self-reflection methods of improvement prohibitive for branching language. In this work, we address this limitation by proposing a new method for improving trees of branching language. Our method iterates between reflecting on sampled paths through a tree and resampling problematic subtrees. We evaluate our method on a branching narrative task with the objective of improving every path through the tree. Our method creates narrative that is preferred 60% more than unmodified narrative trees by an LM judge. Our method also scales to tree depths that cause naive methods of self-reflection to fail.

1 Introduction

Self-Reflection methods improve the quality of language model (LM) outputs by iteratively critiquing and revising generated responses [Bai et al., 2022, Shinn et al., 2023, Wang et al., 2024]. Unlike many other blackbox optimization algorithms, self-reflection is conditioned on its previous model outputs. This detail is inconsequential for applications in which the LM is generating linear language that can be maintained in the LM context. However, many LM use cases are emerging that utilize outputs other than linear language.

Popular applications such as planning, trees of thought, and branching narrative require generating outputs in a tree structure. Many language model agents utilize a tree search over generated future plans to help select the next best action to take [Nottingham et al., 2023, Liu et al., 2023, Zhou et al., 2024]. The reasoning method, Tree of Thoughts, searches over a tree structure of reasoning paths to find the best response Yao et al. [2024]. Branching narrative, popular in interactive text adventures and choose-your-own-adventure style novels, have a long history of using trees of branching language for entertainment [Li and Riedl, 2010, Li et al., 2013, Ammanabrolu et al., 2020, Leandro et al., 2024]. Unlike examples that use branching language for search, branching narrative prioritizes overall tree quality as opposed to the quality of the best path through the tree.

Unfortunately, naively applying self-reflection to improving branching language requires representing large trees of text in-context and fails as the depth of the tree increases. Additionally, the naive approach requires regenerating the entire tree at each iteration of self-reflection even though much of the tree may not require modification. To address these concerns, we propose a new method for applying self-reflection to branching language outlined in Figure 1. Our method starts by sampling a random path through the tree. It then critiques that path and selects a node to start an edit. It

Language Gamification Workshop 2024.

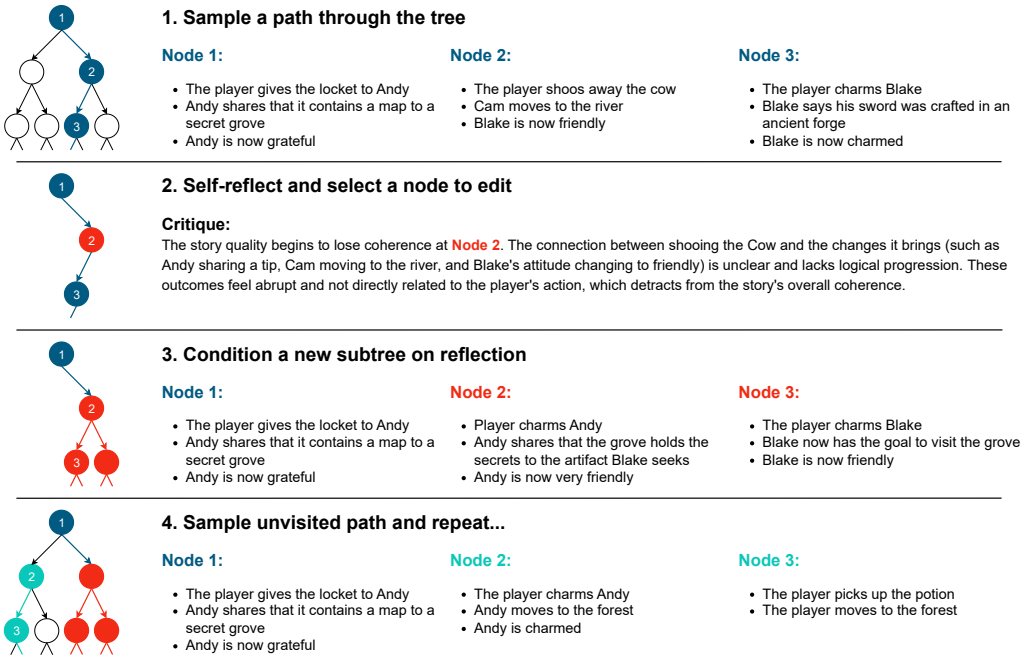

Figure 1: Our method identifies and resamples problematic subtrees to improve overall tree quality. This tree represents alternative paths through a narrative. Each node in the tree corresponds to player action and plot points that occur at that node. By critiquing and resampling subtrees, we efficiently improve the overall story that the tree represents.

resamples the subtree beginning at the selected node, conditioned on the generated critique. This process is repeated until all nodes of the tree have been visited.

We test our method on generating branching narrative events and compare to baseline methods using a LM judge prompted to asses narrative quality. We find that trees generated using our method are preferred over trees without self-reflection by 60% when using an LM judge. We also compare to a naive implementation of self-reflection that critiques entire trees in-context represented as json. While this naive approach is competitive with our method on shallow trees, errors quickly emerge as tree depth increases.

2 Related Work

2.1 Self-Reflection

Bai et al. [2022] helped popularize self-reflection with the method they call *Constitutional AI* that iteratively critiques and revises output based on instructions. A more recent method called *Mixture of Agents* works by iteratively generating outputs from an ensemble of LMs and then critiquing and and revising those outputs [Wang et al., 2024]. Many language model agents utilize reflection methods in interactive environments by leveraging environment feedback as a critique [Shinn et al., 2023, Zhao et al., 2023, Majumder et al., 2023, Nottingham et al., 2024]. All of these applications of self-reflection target linear language outputs and are not designed with branching language in mind. We compare our method to naive adaptations of self-reflection by representing an entire tree of outputs in json.

2.2 Tree-based Generation

Tree-based search using branching outputs from LMs are another popular method for improving LM outputs. For example, *Language Agent Tree Search* and *Reason for Future, Act for Now* utilize

Monte Carlo Tree Search (MCTS) via prompting LMs for language and embodied tasks [Zhou et al., 2024, Liu et al., 2023]. *Tree of Thoughts* uses a similar tree search to reason about general language tasks [Yao et al., 2024]. The above methods prioritize finding the best path in a tree as opposed to improving all paths. However, LMs are also used to generate branching outputs in which all paths matter. For example, Nottingham et al. [2023] use an LM to generate a directed acyclic graph to guide a reinforcement learning policy to explore potential subgoals.

2.3 Narrative Generation

Narrative generation has long utilized branching structures such as trees or directed acyclic graphs to represent potential ordering of plot points in narrative [Li and Riedl, 2010, Li et al., 2013, Ammanabrolu et al., 2020]. In the past, LMs needed a significant amount of assistance to generate coherent narrative [Ammanabrolu et al., 2020, Rashkin et al., 2020]. However, modern LMs are much more proficient at generating stories and branching narrative Leandro et al. [2024]. While modern LMs continue to improve in this regard, they still often output uninteresting or generic stories. An approach such as self-reflection can have a big impact on improving overall story quality.

3 Method

We believe that self-reflection will be a powerful tool for improving branching language. However, naively representing a tree in the context of a language model with a technique such as json, tuples, or path enumeration performs poorly. Also, unlike linear output, not all paths in a tree are dependent on each other. This means that we can modify problematic portions of a tree while leaving the rest untouched, potentially speeding up the self-reflection process.

Algorithm 1 Subtree Self-Reflection

Require: X
  $visited \leftarrow \emptyset$
  while $|visited| < |X|$ **do**
    $path \leftarrow SamplePath(X, visited)$
    $critique, x \leftarrow Reflect(path, visited)$
    if $x \neq null$ **then**
      $X' \leftarrow Resample(x, critique)$
      $X \leftarrow Update(X, X')$
      $visited \leftarrow visited \cup path \cup X'$
    else
      $visited \leftarrow visited \cup path$
    end if
  end while
  return X

With this in mind, we design a new method for applying self-reflection to trees by regenerating problematic subtrees while conditioning on critiques of paths through the tree. A single iteration of this process is detailed in Algorithm 1. We begin with a tree X, from which we randomly sample a $path$, or sequence of nodes, through the tree. We prompt a LM with the $path$ to generate a textual $critique$ and select the most problematic node, labeled x, for editing. We then resample a subtree, X', starting from x conditioned on the $critique$. The LM can also opt to skip resampling a subtree if the $critique$ is positive. This process repeats until all subtrees have been visited.

We mark all nodes that occurred in a $path$ or were part of a regenerated subtree, X', as $visited$. In Algorithm 1, $SamplePath$ and $Reflect$ are conditioned on $visited$ nodes. This is to indicate that $SamplePath$ will only return a $path$ if at least one node in the $path$ has not been $visited$. Likewise, $Reflect$ will not return a $visited$ node as x. Once all nodes are $visited$ the iteration of self-reflection terminates and the fully updated tree is returned.

The naive implementation of self-reflection we consider requires $O(1)$ time complexity when critiquing a tree and $O(n^d)$ time complexity when resampling a tree, where n is the branching factor and d is the depth and the time complexity measures the number of calls to the LM. Our method requires worst case $O(n^{d-1})$ time complexity when critiquing a tree and worst case $O(n^d)$ time complexity when resampling a tree. However, both values are empirically far smaller since the more critiques we generate, the fewer nodes we need to resample. we can also resample subtrees concurrently and our method may choose to skip entire subtrees, further reducing the time it takes us to perform self-reflection. In our experiments, our method is actually 9% faster than the naive method.

Our method also improves space complexity with respect to the context length of the LM. A naive implementation would maintain the entire tree, $O(n^d)$, during the critique step. However, our method only includes tree paths in-context which has a complexity of $O(d)$.

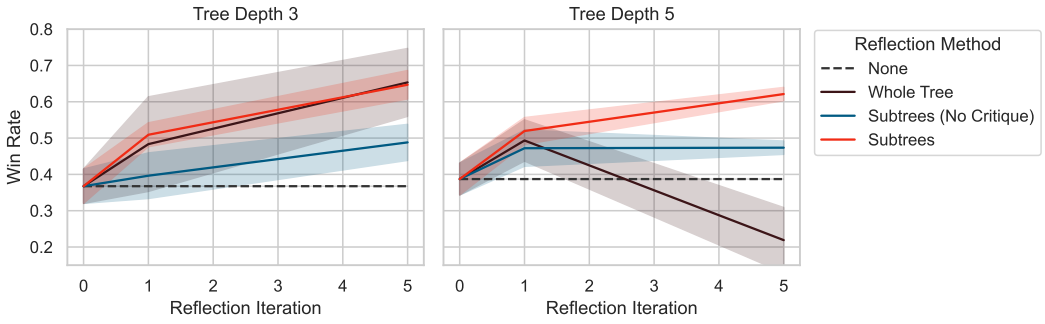

Figure 2: Our method, labeled *Subtrees*, is the only method that successfully scales to improving deeper trees. An naive self-reflection method that reflects on the *Whole Tree* at once is competitive with our method at shallow depths but is less efficient and does not scale as far. Resampling subtrees with *No Critique* in-context utilizes our method for identifying subtrees to resample but ablates including the generated critique in-context when resampling.

4 Experimental Results

We compare our method to several baselines on a branching narrative task. This task involves continuing a story by generating a tree of narrative events that branch around character decisions. At each node, the LM selects 1-3 child nodes from a set of potential character actions. The LM must also select from a set of potential effects that update the state based on that action. Each path through the branching narrative represents a story continuation.

We evaluate a narrative tree by performing pairwise comparisons between randomly sampled paths from different tree generation methods. A LM judge is prompted to summarize each path and select the continuation that is the most interesting, coherent, and logical. The same criteria are used in all self-reflection methods we evaluate on. We generate and evaluate on 10 different story initializations. The initial tree, critiques, edits, and judging are done by GPT-4o with a temperature of 0.7.

Figure 2 reports win rates between our method and baselines after multiple iterations of self-reflection. All methods are able to outperform originally generated trees with no self-reflection. We first compare to a naive baseline that includes the whole tree in-context in json, labeled *Whole Tree*. This method performs comparably to ours on small trees but fails to scale to larger trees and takes 9% longer to run. It also consistently has higher variance across the 10 story initializations we evaluate on as indicated by the error regions in Figure 2.

We also ablate including the generated $critique$ when resampling a subtree, labeled *No Critique*. The fact that *No Critique* improves over the baseline, demonstrates that our method successfully identifies poorly performing nodes to resample. However, including the critique greatly improves performance.

5 Discussion & Conclusion

We develop a method for efficiently performing self-reflection on large trees of branching language. By iteratively critiquing linear paths through the tree and only editing problematic subtrees, our method scales to deeper trees, maintains lower variance, and is 9% faster when compared to baselines.

Our method is especially well suited when applied to the objective of improving overall tree quality, as is the case in our branching narrative task. However, the majority of applications that generate trees of text are used for search, which only optimizes for the best path through the tree. We believe that our method can also be applied to tree search for language by critiquing and resampling subtrees throughout the search. We leave the investigation of this application to future work.

As use cases for LMs continue to grow, non-linear language generation will continue to become more common. LMs will need to be able to process branching language as well as language in graph-based or multiple spatial dimensions. Our work represents early steps in this direction as we investigate how well LMs can process non-linear language.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In *International Conference on Machine Learning*, pages 26311–26325. PMLR, 2023.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *arXiv preprint arXiv:2309.17382*, 2023.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Forty-first International Conference on Machine Learning*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Boyang Albert Li and Mark O. Riedl. An offline planning approach to game plotline adaptation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2010. URL https://api.semanticscholar.org/CorpusID:11061580.

Boyang Albert Li, Stephen Lee-Urban, George Johnston, and Mark O. Riedl. Story generation with crowdsourced plot graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2013. URL https://api.semanticscholar.org/CorpusID:16270356.

Prithviraj Ammanabrolu, Wesley Cheung, William Broniec, and Mark O. Riedl. Automated storytelling via causal, commonsense plot ordering. *ArXiv*, abs/2009.00829, 2020. URL https://api.semanticscholar.org/CorpusID:221446544.

Jorge Leandro, Sudha Rao, Michael Xu, Weijia Xu, Nebojsa Jojic, Chris Brockett, and Bill Dolan. Geneva: Generating and visualizing branching narratives using llms. In *IEEE Conference on Games 2024*, August 2024. URL https://www.microsoft.com/en-us/research/publication/geneva-generating-and-visualizing-branching-narratives-using-llms/.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

Kolby Nottingham, Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Sameer Singh, Peter Clark, and Roy Fox. Skill set optimization: Reinforcing language model behavior via transferable skills. In *Forty-first International Conference on Machine Learning*, 2024.

Hannah Rashkin, Asli Celikyilmaz, Yejin Choi, and Jianfeng Gao. Plotmachines: Outline-conditioned generation with dynamic plot state tracking. *ArXiv*, abs/2004.14967, 2020. URL https://api.semanticscholar.org/CorpusID:216868683.