TreeBoN: Enhancing Inference-Time Alignment with Speculative Tree-Search and Best-of-N Sampling

Jiahao Qiu*[1], Yifu Lu*[1], Yifan Zeng[2], Jiacheng Guo[1], Jiayi Geng[3], Chenhao Zhu[4], Xinzhe Juan[5], Ling Yang[1], Huazheng Wang[2], Kaixuan Huang[1], Yue Wu[1], Mengdi Wang[1],

[1]Princeton University [2]Oregon State University [3]Carnegie Mellon University
[4]Stanford University [5]University of Michigan

Correspondence: mengdiw@princeton.edu

Abstract

Inference-time alignment enhances the performance of large language models without requiring additional training or fine-tuning but presents challenges due to balancing computational efficiency with high-quality output. Best-of-N (BoN) sampling, as a simple yet powerful approach, generates multiple responses and selects the best one, achieving improved performance but with a high computational cost. We propose TreeBoN, a novel framework that integrates a speculative tree-search strategy into Best-of-N (BoN) Sampling. TreeBoN maintains a set of parent nodes, iteratively branching and pruning low-quality responses, thereby reducing computational overhead while maintaining high output quality. Our approach also leverages token-level rewards from Direct Preference Optimization (DPO) to guide tree expansion and prune low-quality paths. We evaluate TreeBoN using AlpacaFarm, HH-RLHF, UltraFeedback, GSM8K, and TutorEval datasets, demonstrating consistent improvements. Specifically, TreeBoN achieves the highest win rate of 65% on TutorEval and around 60% win rates across other different datasets, outperforming standard BoN with the same computational cost and showcasing its scalability and alignment efficacy.

1 Introduction

Aligning large language models (LLMs) with human values is essential for ensuring their outputs reflect human intentions and ethical standards. When data on human preferences is available, a pretrained LLM can be fine-tuned to align with these preferences. One popular approach for fine-tuning is Reinforcement Learning from Human Feedback (RLHF), where a reward model is trained on a human-labeled preference dataset, followed by reinforcement learning to fine-tune the LLM as a policy model (Ouyang et al., 2022). Alternative methods such as Direct Preference Optimization (Rafailov et al., 2024b) and its variants (Azar et al., 2024a; Ethayarajh et al., 2024; Meng et al., 2024) enable direct alignment via fine-tuning using a contrastive loss, eliminating the need for a separate reward model.

This paper focuses on optimizing inference-time alignment of large language models (LLMs). By leveraging inference-time search, the capability of LLMs is enhanced during the generation process, improving real-time decision-making. Various techniques, such as Monte Carlo Tree Search (MCTS), have been effectively applied to reasoning, planning, and accelerated decoding tasks (Zhao et al., 2024; Hao et al., 2023; Brandfonbrener et al., 2024; Choi et al., 2023), demonstrating the potential for better decoding outcomes (Liu et al., 2024a). In this work, we aim to explore tree search strategies to further capitalize on decoding-time alignment. Our goal is to enhance the quality of alignment while simultaneously reducing the computational cost of inference, providing a more efficient and aligned LLM experience.

A most simple, yet powerful inference-time alignment method is the Best-of-N (BoN) method. We start our discussion with BoN to motivate our development of more efficient solutions. BoN generates multiple sample responses and chooses the best one based on a reward function $r(\mathbf{y}|\mathbf{x})$ which characterizes how well-aligned a generated response $\mathbf{y}$ is with respect to the given prompt $\mathbf{x}$. More formally, BoN aims to approximate the solution to the following optimization problem: $\max_{\mathbf{y}} r(\mathbf{y}|\mathbf{x})$ where the only access to $\mathbf{y}$ is through auto-regressively sampling the next token y_t from the base policy $\pi_{\text{base}}(\cdot|\mathbf{x}, \mathbf{y}_{1:t-1})$, conditioned on the previous tokens. BoN generates N samples and selects the response from $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N$ that achieves the highest reward model score. Due to its simplicity and effectiveness, Best-of-N sampling and its variants are widely studied to align LLM outputs with human preferences (Wang et al., 2024; Sessa et al., 2024; Gui et al., 2024; Khaki et al.,

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 20894–20917
November 4-9, 2025

2024; Jinnai et al., 2024; Liu et al., 2024b; Xiong et al., 2024). Also, Best-of-N Sampling is commonly used in Expert Iteration and iterative fine-tuning (Havrilla et al., 2024), which plays an important role in the alignment of Llama2 (Touvron et al., 2023) and Llama3 (Dubey et al., 2024). In detail, Llama2 (Touvron et al., 2023) combines rejection sampling with Proximal Policy Optimization (PPO) in an iterative fine-tuning process to align Llama 2 with human preferences. More recently, Llama3 (Dubey et al., 2024) uses rejection sampling to generate high-quality data for alignment in an iterative process.

While Best-of-N sampling has proven effective, it has a significant drawback: efficiency. Naively implementing BoN requires generating N separate responses and the total inference FLOPs scales linearly with N. This not only demands N times more computation but also potentially leads to N times longer latency. The computational overhead can be prohibitively expensive for LLMs with billions of parameters, particularly when real-time or low-latency responses are needed.

Some potential solutions involve more intelligent sampling strategies such as pruning to improve efficiency. *Speculative Best-of-N* (SBoN) (Zhang et al., 2024) alleviates the problem by continuing the generation of high-quality responses and rejecting the low-quality responses at an early stage of the generation. Cascade Reward Sampling(CARDS) (Li et al., 2024) use rejection sampling to iteratively generate small semantic segments to form such prefixes, based on rewards computed using incomplete sentences.

These accelerated methods are based on the hypothesis that utterances receiving high/low rewards early on in the generation process are likely to yield high/low rewards in the final complete response. However, this hypothesis is too good to be true. In fact, off-the-shelf reward models are typically trained on complete responses, and therefore the score of partial completions by the reward model is usually chaotic and doesn't accurately predict the final output's quality, especially for long responses. Our analysis confirmed that rewards of partial completions are not necessarily positively correlated with the final reward (see our experiment results in Section 4.2.4 and Appendix G).

To enable faster, efficient inference-time alignment, we propose to incorporate a tree search strategy into BoN sampling, in order to improve the alignment quality as well as reduce the overall inference cost. Our TreeBoN method maintains an active set of nodes, and actively grows a tree via branching and pruning. In other words, TreeBoN would sample more frequently from good parent nodes but prunes nodes with low predicted rewards. This tree search strategy makes it possible to efficiently explore the search space.

Another design feature of TreeBoN is the use of implicit reward from DPO-aligned models for guidance of the tree research. DPO (Rafailov et al., 2024b) states that the DPO policy model can provide an implicit reward. Rafailov et al. (2024a) further points out that DPO training implicitly learns a token-level reward function. Thus, we design TreeBoN to be able to leverage any off-the-shelf DPO model for inference-time decoding of the target model. Our extensive experiments show that a weighted combination of implicit DPO rewards would lead to superior, robust performance. Our observation is consistent with the fact that one can detect safety levels of the full response using the first few tokens Qi et al. (2024).

Our experiments show that under the same computing budget, TreeBoN achieves better performance than BoN extensively and stably, with the highest win-rates of 65% on TutorEval (Chevalier et al., 2024), 63% on AlpacaFarm (Dubois et al., 2024) with length 192 and 384, above 60% across HH-RLHF (Bai et al., 2022) and UltraFeedback (Cui et al., 2024), and increased pass@1 solve rate on GSM8K (Cobbe et al., 2021) as well. By choosing a smaller N, TreeBoN could achieve better performance and improve efficiency at the same time. With only 6.3% of the compute, TreeBoN still maintains a 55% win-rate against BoN. On the other hand, SBoN can be viewed as a special example of our method with a two-layer tree whose children number is equal to one and BoN can be viewed as a two-layer tree with the children number equal to N. TreeBoN has the potential to further improve efficiency than expected by taking advantage of the key-value cache which is especially beneficial to the tree structure since the keys and values of parent tokens can be cached and shared by children.

The main contributions of this paper are as follows:

1. We incorporate the Speculative Tree-search framework into Best-of-N Sampling to enhance efficiency and alignment performance simultaneously.

2. We apply weighted implicit reward from DPO to provide the partial reward, which replaces the traditional reward model. We also offer a comprehensive analysis of traditional reward models on partial responses.[1]
3. TreeBoN demonstrates robust improvements in alignment quality and efficiency in comprehensive evaluations.

2 Preliminaries

2.1 Best-of-N sampling (BoN)

To approximate the optimization problem of maximizing the reward function $r(\mathbf{y}|\mathbf{x})$ which measures how well a generated response $\mathbf{y}$ sampled from the base policy $\pi_{\text{base}}(\cdot|\mathbf{x})$ aligns with respect to the given prompt $\mathbf{x}$, Best-of-N Sampling (BoN) selects the response with the highest reward score from N independent and identically distributed (i.i.d.) responses generated by the language model π_{base}:

$$\mathbf{y}^{\star} = \underset{\mathbf{y} \in \{\mathbf{y}^k \sim \pi_{\text{base}}(\cdot|\mathbf{x})\}_{k=1}^{N}}{\operatorname{argmax}} r(\mathbf{y}|\mathbf{x}),$$

where the only access to $\mathbf{y}$ is through auto-regressively sampling the next token y_t from the base policy $\pi_{\text{base}}(\cdot|\mathbf{x}, \mathbf{y}_{1:t-1})$, conditioned on the previous tokens. The algorithm is listed in Appendix B.

2.2 Token-Level Markov Decision Process and Soft Q-Learning

Rafailov et al. (2024b) demonstrated that under the Max-Entropy reinforcement learning (RL) formulation, the token-level log-ratio can be interpreted as an implicit token-level reward or advantage function, which remains invariant under reward shaping.

Below, we briefly restate the key setting and results.

The token-level Markov Decision Process (MDP) defines the state $\mathbf{s}_t = (x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_t)$ as the tokens generated so far, and the action $\mathbf{a}_t = y_{t+1}$ as the next token to be predicted. The auto-regressive language model is thus a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$. The transition dynamics are deterministic: $\mathbf{s}_{t+1} = \mathbf{s}_t|\mathbf{a}_t$, simply appending the next token to the current generated tokens to form a new sequence.

The RLHF formulation can be expressed as a Max-Entropy RL problem:

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{X},\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\left[r(\mathbf{y}|\mathbf{x}) + \beta \log \pi_{\text{ref}}(\mathbf{y}|\mathbf{x})\right] + \beta\mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\mathcal{H}(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}))\right].$$

Or equivalently at the token level:

$$\mathbb{E}_{\mathbf{s}_0\sim\mathcal{X},\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)}\left[\sum_{t=1}^{T} r'(\mathbf{s}_t, \mathbf{a}_t)\right] + \beta\mathbb{E}_{\mathbf{s}_0\sim\mathcal{X}}\left[\mathcal{H}(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_0))\right],$$

with the token level reward function r' for any $(\mathbf{s}_t, \mathbf{a}_t)$ defined as:

$$r'(\mathbf{s}_t, \mathbf{a}_t) := \begin{cases} \beta \log \pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t), & \text{if } \mathbf{s}_{t+1} \text{ is not terminal,} \\ r(\mathbf{y}|\mathbf{x}) + \beta \log \pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t), & \text{if } \mathbf{s}_{t+1} \text{ is terminal.} \end{cases}$$

For simplicity, let us assume that the horizon is fixed at T. The derivation of the Max-Entropy RL formulation (Ziebart, 2010; Rafailov et al., 2024a) utilizes the (soft) optimal value function V^* and the (soft) optimal Q-function Q^*, as follows: $V^*(\mathbf{s}_{T+1}) = 0$ when $\mathbf{s}_{T+1}$ is the terminal state; $Q^*(\mathbf{s}_t, \mathbf{a}_t) = r'(\mathbf{s}_t, \mathbf{a}_t) + V^*(\mathbf{s}_{t+1})$, $V^*(\mathbf{s}_t) = \log \sum_{\mathbf{a}} \exp(Q^*(\mathbf{s}_t, \mathbf{a}))$, when $t \le T$.

The optimal policy π^* satisfies the following equation: $\beta \log \pi^*(\mathbf{a}_t|\mathbf{s}_t) = Q^*(\mathbf{s}_t, \mathbf{a}_t) - V^*(\mathbf{s}_t)$, which can be further rewritten when $t < T$:

$$\beta \log \frac{\pi^*(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t)} = V^*(\mathbf{s}_{t+1}) - V^*(\mathbf{s}_t).$$

This suggests that we can use the partial sum of the implicit reward from a DPO policy to characterize the potential final reward given a prefix sequence of length K:

$$\sum_{t=0}^{K-1} \beta \log \frac{\pi^*(\mathbf{a}_k|\mathbf{s}_k)}{\pi_{\text{ref}}(\mathbf{a}_k|\mathbf{s}_k)} = V^*(\mathbf{s}_K) - V^*(\mathbf{s}_0).$$

Since $\mathbf{s}_0 = (x_1, x_2, \ldots, x_m) = \mathbf{x}$, $V^*(\mathbf{s}_0)$ is the same for all responses.

3 Method

In this section, we introduce **TreeBoN**, a novel inference-time algorithm that enhances alignment quality and efficiency by incorporating a speculative tree-search structure into the Best-of-N (BoN) sampling framework. TreeBoN iteratively expands high-reward partial responses, pruning low-quality candidates at early stages. The algorithm leverages a weighted implicit reward from a Direct Preference Optimization (DPO) policy model to improve the quality of partial response evaluation. Below, we describe the key steps involved in TreeBoN.

[1] See Appendix G for our sentence-level and token-level experiments and examples

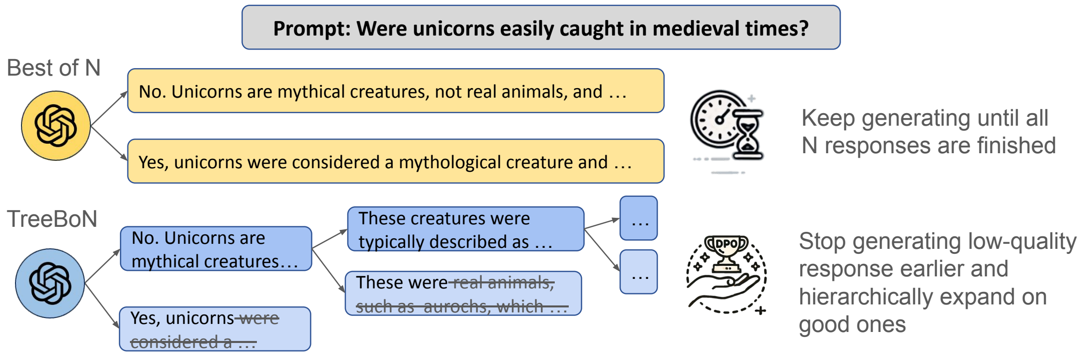


Figure 1: An illustration of different response generation strategies. Best-of-N completes all candidate generations, while TreeBoN (our method) introduces early termination of low-quality responses using a DPO reward model and hierarchically expands promising responses. See Table 17 for the detailed example.

3.1 Overview of TreeBoN Algorithm

TreeBoN operates by generating candidate responses layer-by-layer in a tree structure. The algorithm begins with a set of initial root responses, and at each subsequent layer, only high-reward responses are selected and expanded into multiple children. This speculative search through the tree space improves both the efficiency and the final response quality. The overall structure of TreeBoN is illustrated in Algorithm 1 and Figure 2.

The algorithm takes as input the prompt $\mathbf{x}$, a base policy π_{base} for generating candidate responses, a partial-reward function r, and key hyperparameters including the number of root samples N, maximum response length l_{max}, branching factor(number of children per node) N_{children}, and the number of tree layers N_{layer}.

Furthermore, C_i denotes the candidate set containing all partial responses generated in the i-th layer. P_i denotes the i-th layer active set containing all promising partial responses for expansion in the next layer. l_i is the max new token length for generation in each layer, where $l_i = \frac{l_{\text{max}}}{N_{\text{layer}}}$.

3.2 TreeBoN Generation Process

The generation process in TreeBoN consists of the following key steps:

1. **Initial Candidate Generation:** TreeBoN begins by generating N candidate responses $C_1 = \{\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^N\}$ with a length of l_1 using the base policy π_{base}. The total maximum response length l_{max} is split into segments $l_1, l_2, \dots, l_{N_{\text{layer}}}$ evenly where $l_i = \frac{l_{\text{max}}}{N_{\text{layer}}}$.
2. **Partial Reward Scoring:** At each layer i, the reward model or partial-reward function $r(\mathbf{y}|\mathbf{x})$ is used to compute the reward score for each candidate response $\mathbf{y} \in C_i$. This is performed after generating partial responses of length l_i.
3. **Pruning and Selection:** Based on the reward scores, the top $\frac{N}{N_{\text{children}}}$ candidates from the current layer are selected to form the active set P_i. These high-reward parent responses are used to generate child responses at the next layer.
4. **Response Expansion:** For each parent response $\mathbf{y} \in P_i$, TreeBoN generates N_{children} child responses by sampling from the base policy π_{base} with a maximum new token length l_{i+1}. This process generates the next-layer candidate set C_{i+1}. It is worth noting that the set size of the candidate set is always N and the set size of P_i is always $\frac{N}{N_{\text{children}}}$ to ensure an equal number of total generated tokens without requiring extra computing budget.
5. **Final Selection:** After generating candidates for all layers, the reward model computes the final rewards for the candidate responses in the last layer $C_{N_{\text{layer}}}$. The response $\mathbf{y}^\star$ with the highest reward is selected as the final output:

$$\mathbf{y}^\star = \underset{\mathbf{y} \in C_{N_{\text{layer}}}}{\operatorname{argmax}}\, r(\mathbf{y}|\mathbf{x}).$$

3.3 Weighted Implicit Reward As Guidance

One of the key contributions of TreeBoN is the use of a weighted implicit reward function, inspired by Rafailov et al. (2024b,a); Qi et al. (2024), to evaluate partial responses. This approach allows

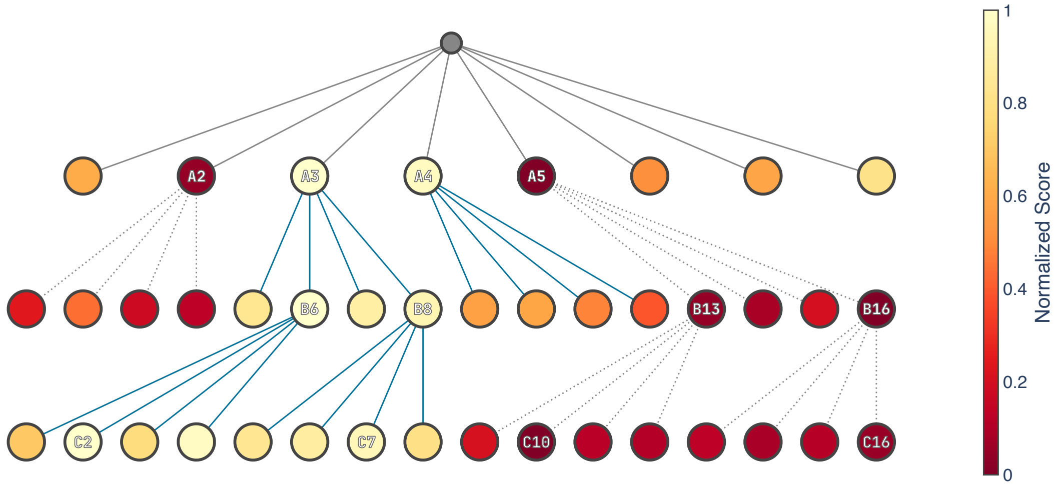


Figure 2: Visualization of speculative tree-search process for the prompt **"Were unicorns easily caught in medieval times?"**. Nodes represent partial responses, with color indicating normalized reward scores. We normalize the reward values within each layer. Solid blue lines show the expansion of high-reward paths, while dotted lines represent pruned low-reward branches. The solid blue line expansion path in this example shows the setting that $N = 8$ initial candidate responses and $N_{\text{children}} = 4$ in Algorithm 1. In detail, labeled nodes **A2 (Yes, unicorns were considered a mythological creature and easily caught in medieval times)** and **A5 (Unicorns were believed to be easily caught in medieval times)** include hallucinations and therefore generating future responses from low-quality prefixes makes it hard to get a high-quality result. Meanwhile, **A3 (No. Unicorns are mythical creatures, not real animals, and therefore could not have been caught in medieval times)** and **A4 (No, unicorns were not easily caught in medieval times. In fact, unicorns were mythical creatures and did not exist in reality)** are high-reward prefixes that are more likely to produce high-quality complete responses in the future. More details of labeled nodes are presented in **Table 17**.

Algorithm 1 TreeBoN Algorithm

Input: Prompt $\mathbf{x}$, base policy π_{base}, partial-reward function r, number of root samples N, max length $l_{\max}$, branching factor N_{children}, number of tree layers N_{layer}.
Output: Response $\mathbf{y}^\star$ with the highest reward using TreeBoN.
Initialization: Split the total max length $l_{\max}$ into segments $l_1, l_2, \ldots, l_{N_{\text{layer}}}$ where $l_i = \frac{l_{\max}}{N_{\text{layer}}}$.
Generate N initial candidate responses for the first-layer candidate set $C_1 = \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N\}$, each with a length of l_1.
for $i = 1$ **to** $N_{\text{layer}} - 1$ **do**
    Query the reward model or partial reward function $r(\mathbf{y}|\mathbf{x})$ to compute the reward scores for each candidate response $\mathbf{y} \in C_i$.
    Select the top $\frac{N}{N_{\text{children}}}$ candidates from C_i based on reward scores to form the i-th layer active set P_i.
    for each parent response $\mathbf{y} \in P_i$ **do**
        For each parent $\mathbf{y}$, continue generation by sampling N_{children} child responses from the base policy π_{base}, each with a max new token length l_{i+1}, to form the next set of candidates C_{i+1}.
    end for
end for
After all layers are generated, query the reward model for the final set of responses $C_{N_{\text{layer}}}$.
Find the response $\mathbf{y}^\star$ with the highest reward:

$$\mathbf{y}^\star = \underset{\mathbf{y} \in C_{N_{\text{layer}}}}{\operatorname{argmax}}\, r(\mathbf{y}|\mathbf{x}).$$

Return the response $\mathbf{y}^\star$.

TreeBoN to replace the traditional reward model with a DPO policy model, which provides more accurate rewards for incomplete responses. The partial reward for a sequence $\mathbf{y}_{:K}$ is computed as:

$$r_{\text{partial}}(\mathbf{y}_{:K}|\mathbf{x}) = \sum_{k=0}^{K-1} w_k \log \frac{\pi^*(y_k|\mathbf{x}, \mathbf{y}_{:k})}{\pi(y_k|\mathbf{x}, \mathbf{y}_{:k})},$$

where $w_k = \frac{1}{|\mathbf{y}_{:k}|}$ acts as a weighting factor to adjust the contribution of each token-level log-likelihood ratio. This weighted reward helps prune low-quality responses early and encourages the continuation of higher-quality candidates throughout the tree expansion process. We also test several different variants of partial reward modeling in Appendix F.

4 Experiments

4.1 Experiment Setting

We use a set of Llama models: LLaMA3-iterative-DPO-final (Xiong et al., 2024; Dong et al., 2024) as the DPO policy model (referred as the DPO model in this section)[2], with its SFT (supervised

[2]See model card https://huggingface.co/RLHFlow/LLaMA3-iterative-DPO-final

fine-tuning) checkpoint trained from Llama 3 8B (AI@Meta, 2024) and reward model FsfairX-LLaMA3-RM-v0.1 (Dong et al., 2023; Xiong et al., 2024) from Llama 3 8B Instruct (AI@Meta, 2024). The SFT model was trained on a set of high-quality instruction datasets for 1 epoch; the reward model was formulated as a Bradley-Terry model optimizing the negative log-likelihood loss function on a mixture of filtered datasets; and notably, the DPO policy model was initialized from the SFT model and updated on the online preference signals produced by the aforementioned reward model (as a proxy of human feedback). We refer readers to (Xiong et al., 2024) for the details of iterative online RLHF and the training of these models. We also use an additional DPO model Llama-3-8B-SFR-Iterative-DPO-R[3], referred as the SFR model in this section. The baseline is the Best-of-N sampling with N equal to 128 and the max token length of responses varies from 192 to 768. For Tree-based BoN with Weighted Implicit Reward, unless otherwise specified, we set the number of tree layers as 4, the number of children per node 4. Considering the cost of the evaluation, we take 100 randomly selected samples from each dataset, following the same setting as SBoN (Zhang et al., 2024). We evaluate the baseline and our methods and take the average of 3 runs of different seeds on AlpacaFarm (Dubois et al., 2024), Ultra-Feedback (Cui et al., 2024), GSM8K (Cobbe et al., 2021), and HH-RLHF (Bai et al., 2022). For TutorEval (Chevalier et al., 2024), we choose 100 closed-book questions.

4.1.1 Metrics

See a detailed explanation of below metrics in Appendix C.

4 Win-rate For all datasets except for GSM8k, we conduct the standard GPT4 win-rate evaluations of our proposed method against the baseline.

Pass@1 Solve Rate For GSM8k, we report the zero-shot pass@1 solve rate (Cobbe et al., 2021).

FLOPs We consider FLOPs as a cost metric. We can show that the computation costs of TreeBoN and Best-of-N are the same and will only be controlled by the number of root samples N and maximum generation length $l_{\max}$ as in Appendix C.3.

[3]This is the official release, trained with the same SFT and reward model, see model card for details `https://huggingface.co/Salesforce/LLaMA-3-8B-SFR-Iterative-DPO-R`

4.2 Results

4.2.1 Improvement over Diverse Datasets

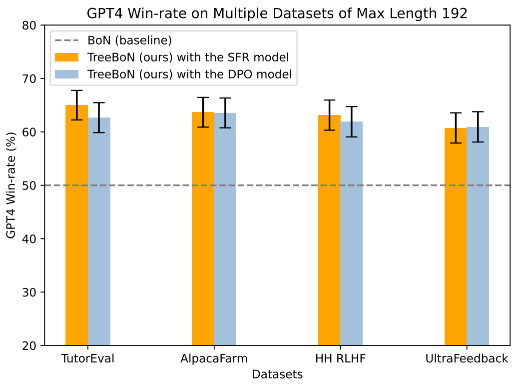
We evaluate the baseline and our methods by answering 100 randomly selected prompts from AlpacaFarm (Dubois et al., 2024), UltraFeedback (Cui et al., 2024), and HH-RLHF (Bai et al., 2022). For TutorEval (Chevalier et al., 2024), we choose 100 closed-book questions. TreeBoN consistently outperforms the baseline across various datasets when evaluated using GPT4 win-rate (Figure 3). The full numerical results of this section can be found in Table 5, 6 and 7 of Appendix D.

Notably, with a maximum length of 192 tokens, TreeBoN with the SFR model achieves a 65% win-rate than Best-of-N sampling on TutorEval, a 64% win-rate on AlpacaFarm, and at least 60% win-rate on other datasets. TreeBoN with the DPO model also achieves a 64%win-rate on AlpacaFarn, and at least 60% on others. This demonstrates that TreeBoN's layered tree structure, combined with the use of a weighted implicit reward function to evaluate partial responses, enables better alignment with human preferences.

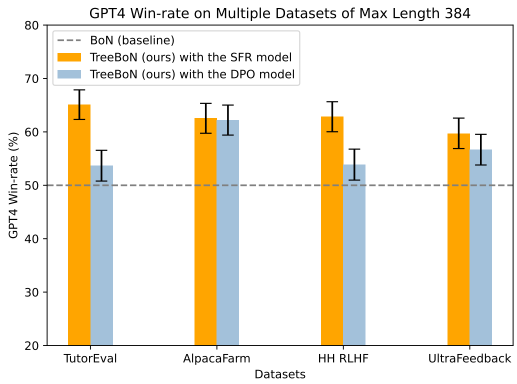
For longer responses (max length 384 tokens), TreeBoN with the SFR model maintains a significant performance lead, showing a 65% win-rate over BoN on TutorEval, 63% on AlpacaFarm and HH-RLHF. If using the DPO model, TreeBoN achieves a 62% win-rate on AlpacaFarm as well. Notably, for the SFR model, from length 384 to length 768, the win-rates are steadily high. This suggests that TreeBoN is also well-suited for handling tasks that require generating more complex or nuanced responses, where multiple layers of exploration yield better results than repeated sampling.

In the same setting, we also evaluate TreeBoN with the SFR model on the entire AlpacaFarm dataset, which has 805 prompts. We obtain 65.67% and 60.57% win-rates over BoN with max length 192 and 384 respectively, showing that TreeBoN's performance is generalizable.

In addition to general alignment improvements, TreeBoN's zero-shot performance on mathematical reasoning dataset GSM8K (Cobbe et al., 2021) also sees a non-trivial boost. In Table 7, TreeBoN with the DPO model outperformed BoN by an impressive 9% margin of pass@1 solve rate at maximum response lengths of 576 tokens, indicating that the hierarchical nature of TreeBoN allows it to effectively manage challenging reasoning tasks that require long CoT reasoning, making it adaptable



Maximum length 192.



Maximum length 384.

Figure 3: GPT4 win-rate of TreeBoN against BoN on multiple datasets. The SFR model refers to using Llama-3-8B-SFR-Iterative-DPO-R as the DPO model, and the DPO model refers to using LLaMA3-iterative-DPO-final. See Table 5 and 6 for numerical results

across different domains.

4.2.2 Explore Different Tree Structures with Same Computation

We further explored the effect of different tree structures by varying the number of layers and children per node (Table 8 and 9) separately, while keeping $N = 128$ and $l_{\max}$ the same, thus the overall computation is unchanged. We use the set of Llama models: LLaMA3-iterative-DPO-final, its SFT checkpoint, FsfairX-LLaMA3-RM-v0.1 on AlpacaFarm, and compute the win-rate against BoN. We can observe in Table 8 that increasing the number of tree layers consistently improves performance on AlpacaFarm, and in Table 9, the optimal number of children nodes is different for two maximum generation lengths. Above all, regardless of the tree structure, our approach maintains a win-rate of around 60% against the baseline, indicating its effectiveness and robustness under different tree structures, and the potential to further improve the performance in the future by exploring more hyper-parameters tailored to different tasks.

4.2.3 Efficiency Evaluation

As shown previously, the computation costs measured by FLOPs of TreeBoN and BoN are only determined by number of root nodes N and max length $l_{\max}$. In Table 4, we show the FLOPs used by different configurations. We then compare the computation cost of our TreeBoN and Best-of-N. We use the same sets of Llama models on AlpacaFarm with a max length of 384. As observed in Table 10, with increasing computation budget, the win-rate of TreeBoN against BoN is also increasing. Thus, our proposed method is more scalable than the baseline and can utilize the additional computation budget more efficiently. In Table 11, TreeBoN of increasing N are compared to BoN with $N = 128$. We can see that even with a very small $N = 8$ (6.3% of FLOPs), TreeBoN can still outperform BoN with a much greater computation budget at a win-rate of 55%, and the quality is monotonic increasing on N.

4.2.4 Comparison over Other Baselines under Same Compute

We compare TreeBoN to other baselines (Li et al., 2024; Zhang et al., 2024) by the win-rates against BoN in Table **??**, with the same set of Llama models introduced earlier for all methods for max length 384 and 192.

To ensure a fair comparison, we constrain the total number of tokens generated during inference. However, for CARDS (Li et al., 2024), the rejection-based sampling with semantic segmentation mechanism introduces uncertainty in token acceptance, leading to variations in the number of generated tokens and requiring random numbers of completions per step. As a result, the total token count remains dynamic and context-dependent.

We adopt the hyperparameters from Li et al. (2024) for LLaMA 7B, as they are the most similar to our setup. We then compute the average number of tokens generated per prompt in the AlpacaFarm dataset, which amounts to 3002.3 tokens for a max length of 192 and 5867.3 tokens for 384.

For both BoN and TreeBoN, the total number of generated tokens follows the relation: Total Tokens

$= l_{\max} \times N$. Thus, we set $N = 16$ for BoN and TreeBoN, resulting in total token counts of 3072 and 6144 for the respective cases, aligning closely with the results of CARDS.

For SBoN, we adopt the hyperparameters from Zhang et al. (2024) for their case of LLaMA3-8B as the language model and LLaMA3-8B-RM as the reward model, given their similarity to our setup. We apply a rejection rate of $\alpha = 30\%$. To ensure comparable computations, we set $N_{\text{SBoN}} = 19$ for SBoN, where the total token count is computed as $l_{\max} \times \left(1 - \frac{\alpha}{2}\right) \times N_{\text{SBoN}}$. This results in total token counts of 3101 and 6202 for two max lengths.

The comparison results are presented in Table **??**. Under the same compute constraints, TreeBoN consistently outperforms other methods, achieving the highest GPT4 win-rates against BoN across both evaluated sequence lengths.

At max length 192, TreeBoN significantly surpasses both SBoN and CARDS, achieving a win-rate of 63.21%, compared to 51.01% for CARDS and 49.66% for SBoN. At max length 384, TreeBoN still maintains its superior performance with a win-rate of 55.18%.

Max Length	Methods	GPT4 Win Rates (%)
192	SBoN	49.66 ± 2.90
	CARDS	51.01 ± 2.90
	TreeBoN	$\mathbf{63.21 \pm 2.79}$
384	SBoN	48.83 ± 2.90
	CARDS	49.66 ± 2.90
	TreeBoN	$\mathbf{55.18 \pm 2.88}$

Table 1: Comparison of different methods with baseline models in terms of total tokens and GPT4 win rates.

4.2.5 Ablation Study

We also experiment with different implicit rewards in Appendix F, and ablate the two components of TreeBoN: the tree-search process, and weighted implicit reward in Appendix E. We conclude that our weighted implicit reward fits best with the tree-search setting compared to other implicit rewards, and both speculative tree-search and weighted implicit reward are needed for substantial improvement.

5 Analysis of TreeBoN Performance in LLM Reasoning

5.1 TreeBoN with PRM Experiment Methods

We continue to use similar set of Llama models: LLaMA3-iterative-DPO-final and Llama-3-8B-SFR-Iterative-DPO-R[4] which still referred as the DPO model and the SFR model respectively throughout this section. The TreeBoN implementation follows the methodology described in Section 3. For the evaluation of the reward for the process, we take the average of each reward for each step as the final reward (Lightman et al., 2023) and follow the `Qwen2.5-Math-PRM-7B` [5] (Zhang et al., 2025) official model card which is the PRM we choose. During inference, we specify a prompt template that instructs the LLM to output the final answer after a designated marker (i.e., `#### <final_answer>`, as shown in the GSM8K dataset) (Cobbe et al., 2021). We then extract only the numerical value that follows this marker as the predicted answer. To evaluate performance, we randomly sample 100 questions from the GSM8K dataset and repeat the evaluation multiple times to compute the mean and standard deviation of Pass@1 solve rate. We experiment with different maximum token lengths, number of candidates, and generative models, applying both BoN and TreeBoN approaches. The complete results are reported in Table 2 and Table 3.

Method / Length	192	384
BoN w/ DPO	43.67 ± 2.08	97.33 ± 2.52
BoN w/ SFR	24.67 ± 2.08	91.33 ± 2.31
TreeBoN w/ DPO	51.00 ± 1.73	95.67 ± 3.06
TreeBoN w/ SFR	25.67 ± 2.52	90.00 ± 1.41

Table 2: Pass@1 Solve Rate (%) on GSM8K using PRM **Qwen2.5-Math-PRM-7B** with $N = 128$.

Method / Length	192	384
BoN w/ DPO	45.67 ± 2.08	96.33 ± 3.21
BoN w/ SFR	26.67 ± 1.53	91.67 ± 0.58
TreeBoN w/ DPO	50.33 ± 3.06	95.33 ± 2.08
TreeBoN w/ SFR	26.33 ± 1.53	89.00 ± 4.24

Table 3: Pass@1 Solve Rate (%) on GSM8K using PRM **Qwen2.5-Math-PRM-7B** with $N = 32$.

5.2 TreeBoN Also Work on Reasoning Task

With both $N = 32$ and $N = 128$ configurations, we find that TreeBoN improves Pass@1 solve rate over BoN when using the Process Reward Model (PRM) (Lightman et al., 2023) with a maximum token length of 192. In Table 3, TreeBoN with DPO modle and Qwen2.5-Math-PRM-7B process

[4] https://huggingface.co/Salesforce/LLaMA-3-8B-SFR-Iterative-DPO-R

[5] https://huggingface.co/Qwen/Qwen2.5-Math-PRM-7B

reward model achieves 50.33%, exceeding BoN's 45.67% by $+4.66\%$. Moreover, in Table 2, this improvement continues with larger number of candidates (N): TreeBoN with DPO model + PRM improves from BoN's 43.67% to 51.00%, a $+7.33\%$ gain. This result validates that TreeBoN better exploit the reward information under a restricted token length.

For a maximum token length of 384, the performance of TreeBoN and BoN methods is similar across all experimental settings. This is because the long token length of 384 already provides sufficient capacity for the model to complete most reasoning tasks effectively. Even with a relatively small number of candidates (e.g., $N = 32$), the generation quality reaches a performance ceiling, leaving limited room for TreeBoN to further improve over BoN sampling. As a result, the structural advantage of TreeBoN becomes less obvious when the generative model already produces high-quality completions within the given token length.

These results highlight TreeBoN's strength in leveraging early-stage completions. When decoding is limited, like 192 tokens, BoN sampling strategy sometimes fails to reach informative states, particularly when using the DPO policy model. In contrast, TreeBoN incrementally expands promising candidates via its tree structure and early prunes low-reward children, making more efficient use of PRM's fine-grained supervision within the same limited token length. This ability to prioritize and extend promising partial completions is crucial when the available token length is insufficient for complete full task reasoning, as it increases the chance of discovering better outputs.

In this way, TreeBoN not only improves performance but also unlocks more of the underlying potential of large language models under constrained generation settings.

6 Additional Results Compared to Beam Search

We also compare TreeBoN with simple beam search. Under the same compute, TreeBoN, using the SFT model to decode and a DPO-aligned model to provide partial reward, outperforms naive beam search that uses the own probabilities to guide decoding of the same DPO-aligned model in TreeBoN. We conduct an additional experiment that compare TreeBoN using the SFR model with $N = 64$ against beam search using the same SFR model with width 128 for fair comparison. The win-rate is $55.33 \pm 2.88\%$ on max length 192, and $58.00 \pm 2.85\%$ on 384.

7 Conclusion

TreeBoN is a novel framework that combines the speculative tree-search strategy with Best-of-N (BoN) Sampling and token-level reward guidance modified from DPO implicit reward. Through extensive experiments, we show that TreeBoN not only has robust alignment improvements but also maintains efficiency, which provides a potential solution for efficient inference and alignment of LLMs.

Limitations

While TreeBoN achieves robust improvements, it greatly relies on the high quality of the reward model on incomplete responses to accelerate the inference without losing performance by iterative expansion and pruning, which is also key to SBoN (Zhang et al., 2024). Though implicit reward from the DPO model provides a candidate solution for the token-level reward guidance, it can only compare responses with the same length. Also, the poorly trained DPO model and its SFT checkpoints would fail to provide good partial rewards. Therefore, the accurate reward modeling of partial responses is still an open question. Reinforcement learning may provide better solutions for partial reward modeling but suffers from the difficulty of training.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AI@Meta. 2024. Llama 3 model card.

Afra Amini, Tim Vieira, and Ryan Cotterell. 2024. Variational best-of-n alignment. *Preprint*, arXiv:2407.06057.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024a. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pages 4447–4455. PMLR.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024b. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pages 4447–4455. PMLR.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, and 12 others. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *Preprint*, arXiv:2204.05862.

Ahmad Beirami, Alekh Agarwal, Jonathan Berant, Alexander D'Amour, Jacob Eisenstein, Chirag Nagpal, and Ananda Theertha Suresh. 2024. Theoretical guarantees on the best-of-n alignment policy. *Preprint*, arXiv:2401.01879.

David Brandfonbrener, Sibi Raja, Tarun Prasad, Chloe Loughridge, Jianang Yang, Simon Henniger, William E Byrd, Robert Zinkov, and Nada Amin. 2024. Verified multi-step synthesis using large language models and monte carlo tree search. *arXiv preprint arXiv:2402.08147*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *Preprint*, arXiv:2407.21787.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *Preprint*, arXiv:2401.10774.

Alex J. Chan, Hao Sun, Samuel Holt, and Mihaela van der Schaar. 2024. Dense reward for free in reinforcement learning from human feedback. *Preprint*, arXiv:2402.00782.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Zhipeng Chen, Kun Zhou, Wayne Xin Zhao, Junchen Wan, Fuzheng Zhang, Di Zhang, and Ji-Rong Wen. 2024a. Improving large language models via fine-grained reinforcement learning with minimum editing constraint. *Preprint*, arXiv:2401.06081.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024b. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *Preprint*, arXiv:2402.12374.

Alexis Chevalier, Jiayi Geng, Alexander Wettig, Howard Chen, Sebastian Mizera, Toni Annala, Max Jameson Aragon, Arturo Rodríguez Fanlo, Simon Frieder, Simon Machado, Akshara Prabhakar, Ellie Thieu, Jiachen T. Wang, Zirui Wang, Xindi Wu, Mengzhou Xia, Wenhan Jia, Jiatong Yu, Jun-Jie Zhu, and 3 others. 2024. Language models as science tutors. *Preprint*, arXiv:2402.11111.

Sehyun Choi, Tianqing Fang, Zhaowei Wang, and Yangqiu Song. 2023. Kcts: Knowledge-constrained tree search decoding with token-level hallucination detection. *Preprint*, arXiv:2310.09044.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. 2024. Reward model ensembles help mitigate overoptimization. *Preprint*, arXiv:2310.02743.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, and 1 others. 2024. Ultrafeedback: Boosting language models with scaled ai feedback. In *Forty-first International Conference on Machine Learning*.

Haikang Deng and Colin Raffel. 2023. Reward-augmented decoding: Efficient controlled text generation with a unidirectional reward model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, page 11781–11791. Association for Computational Linguistics.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. 2023. Raft: Reward ranked finetuning for generative foundation model alignment. *Preprint*, arXiv:2304.06767.

Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen Sahoo, Caiming Xiong, and Tong Zhang. 2024. Rlhf workflow: From reward modeling to online rlhf. *Preprint*, arXiv:2405.07863.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2024. Alpacafarm: A simulation framework for methods that learn from human feedback. *Preprint*, arXiv:2305.14387.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization. *Preprint*, arXiv:2402.01306.

Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR.

Dongyoung Go, Tomasz Korbak, Germán Kruszewski, Jos Rozen, and Marc Dymetman. 2024. Compositional preference models for aligning lms. *Preprint*, arXiv:2310.13011.

Lin Gui, Cristina Gârbacea, and Victor Veitch. 2024. Bonbon alignment for large language models and the sweetness of best-of-n sampling. *Preprint*, arXiv:2406.00832.

Seungwook Han, Idan Shenfeld, Akash Srivastava, Yoon Kim, and Pulkit Agrawal. 2024. Value augmented sampling for language model alignment and personalization. *Preprint*, arXiv:2405.06639.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *Preprint*, arXiv:2305.14992.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. 2024. Teaching large language models to reason with reinforcement learning. *Preprint*, arXiv:2403.04642.

Joey Hejna, Rafael Rafailov, Harshit Sikchi, Chelsea Finn, Scott Niekum, W. Bradley Knox, and Dorsa Sadigh. 2024. Contrastive preference learning: Learning from human feedback without rl. *Preprint*, arXiv:2310.13639.

James Y. Huang, Sailik Sengupta, Daniele Bonadiman, Yi an Lai, Arshit Gupta, Nikolaos Pappas, Saab Mansour, Katrin Kirchhoff, and Dan Roth. 2024a. Deal: Decoding-time alignment for large language models. *Preprint*, arXiv:2402.06147.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. 2024b. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *Preprint*, arXiv:2405.19715.

Yuu Jinnai, Tetsuro Morimura, Kaito Ariu, and Kenshi Abe. 2024. Regularized best-of-n sampling to mitigate reward hacking for language model alignment. *Preprint*, arXiv:2404.01054.

Saeed Khaki, JinJin Li, Lan Ma, Liu Yang, and Prathap Ramachandra. 2024. Rs-dpo: A hybrid rejection sampling and direct preference optimization method for alignment of large language models. *Preprint*, arXiv:2402.10038.

Maxim Khanov, Jirayu Burapacheep, and Yixuan Li. 2024. Args: Alignment as reward-guided search. *Preprint*, arXiv:2402.01694.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Bolian Li, Yifan Wang, Ananth Grama, and Ruqi Zhang. 2024. Cascade reward sampling for efficient decoding-time alignment. *Preprint*, arXiv:2406.16306.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023a. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval.

Yuhui Li, Fangyun Wei, Jinjing Zhao, Chao Zhang, and Hongyang Zhang. 2023b. Rain: Your language models can align themselves without finetuning. *Preprint*, arXiv:2309.07124.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

Bill Yuchen Lin, Abhilasha Ravichander, Ximing Lu, Nouha Dziri, Melanie Sclar, Khyathi Chandu, Chandra Bhagavatula, and Yejin Choi. 2023. The unlocking spell on base llms: Rethinking alignment via in-context learning. In *The Twelfth International Conference on Learning Representations*.

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. 2024a. Don't throw away your value model! generating more preferable text with value-guided monte-carlo tree search decoding. *Preprint*, arXiv:2309.15028.

Tianqi Liu, Yao Zhao, Rishabh Joshi, Misha Khalman, Mohammad Saleh, Peter J. Liu, and Jialu Liu. 2024b. Statistical rejection sampling improves preference optimization. *Preprint*, arXiv:2309.06657.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. *Preprint*, arXiv:2405.14734.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, and 1 others. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949.

Sidharth Mudgal, Jong Lee, Harish Ganapathy, YaGuang Li, Tao Wang, Yanping Huang, Zhifeng Chen, Heng-Tze Cheng, Michael Collins, Trevor Strohman, Jilin Chen, Alex Beutel, and Ahmad Beirami. 2024. Controlled decoding from language models. *Preprint*, arXiv:2310.17022.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek Mittal, and Peter Henderson. 2024. Safety alignment should be made more than just a few tokens deep. *Preprint*, arXiv:2406.05946.

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. 2024a. From r to q^*: Your language model is secretly a q-function. *Preprint*, arXiv:2404.12358.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024b. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Corby Rosset, Ching-An Cheng, Arindam Mitra, Michael Santacroce, Ahmed Awadallah, and Tengyang Xie. 2024. Direct nash optimization: Teaching language models to self-improve with general preferences. *Preprint*, arXiv:2404.03715.

Pier Giuseppe Sessa, Robert Dadashi, Léonard Hussenot, Johan Ferret, Nino Vieillard, Alexandre Ramé, Bobak Shariari, Sarah Perrin, Abe Friesen, Geoffrey Cideron, Sertan Girgin, Piotr Stanczyk, Andrea Michi, Danila Sinopalnikov, Sabela Ramos, Amélie Héliou, Aliaksei Severyn, Matt Hoffman, Nikola Momchev, and Olivier Bachem. 2024. Bond: Aligning llms with best-of-n distillation. *Preprint*, arXiv:2407.14622.

Ruizhe Shi, Yifang Chen, Yushi Hu, Alisa Liu, Hannaneh Hajishirzi, Noah A. Smith, and Simon Du. 2024. Decoding-time language model alignment with multiple objectives. *Preprint*, arXiv:2406.18853.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2024. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xiaofei Wang, Jinhua Li, and Yifan Zhang. 2024. Improved value alignment in large language models using variational best-of-n techniques.

Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. 2024. Self-play preference optimization for language model alignment. *Preprint*, arXiv:2405.00675.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. 2024. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. In *Forty-first International Conference on Machine Learning*.

Joy Qiping Yang, Salman Salamatian, Ziteng Sun, Ananda Theertha Suresh, and Ahmad Beirami. 2024a. Asymptotics of language model alignment. *Preprint*, arXiv:2404.01730.

Shentao Yang, Shujian Zhang, Congying Xia, Yihao Feng, Caiming Xiong, and Mingyuan Zhou. 2024b. Preference-grounded token-level guidance for language model fine-tuning. *Advances in Neural Information Processing Systems*, 36.

Yongcheng Zeng, Guoqing Liu, Weiyu Ma, Ning Yang, Haifeng Zhang, and Jun Wang. 2024. Token-level direct preference optimization. *Preprint*, arXiv:2404.11999.

Ruiqi Zhang, Momin Haider, Ming Yin, Jiahao Qiu, Mengdi Wang, Peter Bartlett, and Andrea Zanette. 2024. Accelerating best-of-n via speculative rejection. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. *Preprint*, arXiv:2501.07301.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2024. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36.

Brian D Ziebart. 2010. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University.

A Related Works

A.1 Best-of-N Sampling for Alignment

Best-of-N (BoN) sampling is a commonly used strategy for aligning large language models with human preferences by selecting the best sample out of N candidates. At training time, (Amini et al., 2024) fine-tunes models by minimizing the KL divergence to approximate the BoN distribution, improving value alignment using variational BoN, which reduces the computational cost during inference. (Sessa et al., 2024; Gui et al., 2024) further enhance alignment by distilling the BoN sampling behavior directly into the model during training, aiming to replicate the BoN distribution with a single sample at inference time. At inference time, (Zhang et al., 2024) speeds up BoN by stopping the generation of unlikely candidates, and (Khaki et al., 2024) combines rejection sampling with preference optimization to improve efficiency without sacrificing alignment performance. From a theoretical perspective, an initial estimate for the KL divergence between the BoN output policy and the base model was provided for small values of N (Coste et al., 2024), (Gao et al., 2023), (Go et al., 2024), and this estimate was later improved to cover all values of N (Beirami et al., 2024). It has also been shown that BoN and KL-regularized reinforcement learning methods achieve similar asymptotic expected rewards, with minimal KL deviation between them (Yang et al., 2024a). Compared with the works mentioned above, our work utilizes a tree-structured search scheme / segment-wise beam search to accelerate best-of-N sampling by pruning the low-reward branches early. To terminate low-reward branches early, we utilize the implicit value function from a DPO policy.

A.2 Tree-Search/MCTS For Language Model

MCTS has been employed in large language model tasks recently (Kocsis and Szepesvári, 2006). Zhao et al. (2024) and Hao et al. (2023) integrates MCTS into planning and logical reasoning tasks. VerMCTS (Brandfonbrener et al., 2024) utilizes a logical verifier to guide a modified Monte Carlo Tree Search (MCTS) for code generation. KCTS (Choi et al., 2023) guides the language model to generate text aligned with the reference knowledge at each decoding step by combining a knowledge classifier score and MCTS. PPO-MCTS(Liu et al., 2024a) combines MCTS and PPO value network for decoding.

Speculative Decoding is introduced to accelerate LLM inference while keeping the distribution of LLM's output distribution unchanged by using a much smaller draft model to predict the LLM outputs which are verified later in parallel by the LLM (Chen et al., 2023; Leviathan et al., 2023). SpecDec++ (Huang et al., 2024b) adaptively selects candidate token lengths using a trained acceptance prediction head, achieving substantial inference speedups on large language models by reducing verification costs without sacrificing accuracy. SpecInfer and SpecTr extend the sequence to a token tree, increasing the number of accepted tokens by the target model (Sun et al., 2024; Miao et al., 2024). SEQUOIA further proposes the method for constructing the optimal tree structure for the speculated tokens by introducing a dynamic programming algorithm (Chen et al., 2024b). Medusa (Cai et al., 2024) is designed to accelerate large language model (LLM) inference by using multiple parallel decoding heads to predict multiple tokens simultaneously, reducing decoding steps without requiring a separate draft model, thus improving efficiency and speed while maintaining output quality.

While our tree-structured search framework bears resemblance with MCTS or tree-based speculative decoding, they are fundamentally different: most MCTS algorithms are designed for planning and logical reasoning tasks with a clear reward signal in the end, while our work focuses on using tree search to accelerate best-of-N sampling and LLM alignment and the signal is obtained throughout the search process. Tree-based speculative decoding is used to accelerate sampling from the target distribution, while ours is used to accelerate sampling for the best of the N responses. PPO-MCTS doesn't consider the efficiency, instead, it focuses on token-level tree expansion involving the backup stage which takes more time. Also, the guidance of PPO-MCTS is a value network from PPO which differs from ours.

A.3 Reward Modeling

Full-sequence reward modeling. RLHF uses the Bradley-terry model to learn a reward function for full-sequence (Christiano et al., 2017; Stiennon et al., 2020). DPO (Rafailov et al., 2024b) implicitly solves the KL-regularized RLHF problem by representing the reward with a language model.SimPO (Meng et al., 2024) considers a different BT model based on the average (length-normalized) reward rather than the sum of rewards.

It is worth noting that alignment can go beyond a reward model due to the inconsistency in human preference. To this end (Azar et al., 2024b; Rosset et al., 2024; Wu et al., 2024), also optimize LLM's log-ratio according to different criteria, and the log-ratio can serve as sequence-level reward indicator.

Partial/Token-level reward modeling. Not every token contributes to human preference equally. A token-level reward signal is thus desirable so that we can do credit assignments to each token. Reward grounding (Yang et al., 2024b) attempts to learn a token-level reward via Maximum Likelihood Estimation (MLE). They define a specific aggregation function so that token rewards can be transformed into sequence rewards, which can then be learned via MLE under the BT model. Reward reshaping can also be used to obtain token-level rewards. For instance, Chan et al. (2024) uses attention weights to redistribute the sequence reward to each token. Mudgal et al. (2024) and Han et al. (2024) propose learning a value function to guide token-level sampling in controlled decoding tasks.

Inverse Q preference learning: DPO reward is a token-level reward model More recent works go beyond reward modeling by treating the problem as inverse Q-learning. Rafailov et al. (2024a) shows that the DPO loss can be interpreted as implicitly learning a token-level Q^* function, represented by the LLM's logits. Similarly, Contrastive Preference Learning (CPL) (Hejna et al., 2024) assumes that human preferences follow a Bradley-Terry model based on the sum of Q values rather than the sum of rewards, and proposes to learn the Q function directly. Zeng et al. (2024) similarly expand on this idea, presenting token-level direct preference optimization based on the Q value function.

In this work, we examine the effectiveness of these reward modeling approaches by incorporating these signals with our tree-search BoN framework. Additionally, we propose a new design: the weighted sum of implicit DPO rewards that turns out highly effective.

A.4 Decoding-Time Alignment

DeAL views decoding as a heuristic-guided search process and integrates alignment to decoding using a wide range of alignment objectives (Huang et al., 2024a). RAD (Deng and Raffel, 2023) uses a unidirectional reward model and ARGS designs a weighted scoring function involving the reward model (Khanov et al., 2024) to do the reward-guided search for decoding-time alignment. URIAL (Lin et al., 2023) and RAIN (Li et al., 2023b) use in-context learning by prompting the LLMs to do the self-alignment without SFT or RLHF. Controlled decoding (Mudgal et al., 2024) trains a value function from the reward model for better token-level scoring. RLMEC (Chen et al., 2024a) trains a generative token-level reward model for alignment. Cascade Reward Sampling(CARDS) (Li et al., 2024) uses a reward model on semantically complete segments to accelerate the decoding. Shi et al. (2024) extends decoding-time alignment to multiple objectives by generating the next token from a linear combination of predictions of all base models.

Cascade Reward Sampling(CARDS) (Li et al., 2024) use rejection sampling to iteratively generate small semantically complete segments, based on rewards computed on incomplete responses by a reward model. The assumption is that semantically-complete high-reward prefixes induce high-reward complete text. However, as shown in Appendix G.3, for responses that are longer than 128 which are not included by CARDS, we show that in our tree search setting where partial responses are 1/3 of the length, the partial reward of a reward model, even on semantically complete segments, has little correlation to the reward on the full response, thus unsuitable to be combined with Tree-Search.

B Algorithm of BoN

Algorithm 2 Best-of-N Sampling (BoN)

1: **Input:** Prompt $\mathbf{x}$, base policy π_{base}, reward model r, number of samples N, max length l_{max}
2: **Output:** Response $\mathbf{y}^\star$ with the highest reward using BoN
3: **Initialization:** Generate N responses $\{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N\}$, each with maximum length l_{max}
4: Query the reward model to compute the reward scores $r(\mathbf{y}|\mathbf{x})$ for each generated response $\mathbf{y} \in \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N\}$
5: Find the response $\mathbf{y}^\star$ with the highest reward:

$$\mathbf{y}^\star = \operatorname*{argmax}_{\mathbf{y} \in \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N\}} r(\mathbf{y}|\mathbf{x})$$

6: **Return** the response $\mathbf{y}^\star$

B.1 Comparison to Baseline Methods

TreeBoN builds upon and extends earlier sampling strategies, such as *Accelerating Best-of-N via Speculative Rejection* (SBoN) (Zhang et al., 2024), by integrating a speculative tree-search framework and partial reward function. SBoN relies on the assumption that partial-reward scores are positively correlated with full-response rewards. However, this assumption often leads to suboptimal performance in alignment tasks due to the inaccurate scoring of partial responses by reward models which are typically trained on complete responses. TreeBoN addresses this limitation by utilizing a more precise implicit reward signal derived from the Direct Preference Optimization (DPO) policy model, which significantly enhances the reliability of partial-reward approximation.

Moreover, TreeBoN leverages a hierarchical tree structure to explore the response space more comprehensively, balancing both alignment quality and computational efficiency. This tree-based approach allows for more flexible and effective pruning of low-quality responses while expanding promising candidates over multiple layers. As a result, TreeBoN can be seen as a generalization of SBoN, where setting $N_{\text{children}} = 1$ and $N_{\text{layer}} = 2$ reduces TreeBoN to the two-layered structure of SBoN.

Compared to traditional Best-of-N (BoN) sampling, which explores candidate responses without any hierarchical structure, TreeBoN employs a more structured exploration strategy. By generating and refining responses layer by layer, TreeBoN achieves a more efficient search of the response space using fewer overall samples. This leads to improvements in both speed and performance, as the tree-based generation effectively balances the trade-off between exploration and exploitation.

TreeBoN can be further accelerated while maintaining high alignment quality by taking advantage of key-value caching mechanisms, particularly beneficial in the tree structure, where the keys and values of parent tokens can be reused by their children.

C Metrics

C.1 GPT4 Win-rate

Given the same prompt, a response from the baseline and a response from the compared method are fed to an automatic evaluator of AlpacaEval (Li et al., 2023a) with randomized positions, which then formats them into a prompt, and asks GPT4 (Achiam et al., 2023) to rank both responses.[6]

C.2 Pass@1 Solve Rate

Pass@k measures the rate of successfully passing the test (answering the math question correctly) from the k responses that the algorithm generates. Thus, pass@1 means that the algorithm only outputs one response per question.[7] We first split the response by space into words and numbers, and then count it to be correctly solved if the answer is in any of the numbers. We extract the number after "answer is " as the final answer.

C.3 FLOPs

The cost of LLMs mainly arises from the number of generated tokens and the matrix multiplications for dense transformers like Llama 3, considering the practical implementations of KV Cache that enable keys and values of parent tokens to be reusable (for the reward model and DPO model as well), we can approximate inference FLOPs with the same formula as in (Brown et al., 2024):

$$
\begin{aligned}
&\text{FLOPs per token} \approx 2 * (\text{num parameters} + 2* \\
&\text{num layers} * \text{token dim} * \text{context length}) \\
&= 2 * \left(8 * 10^9 + 2 * 32 * 4096 * 8192\right) \\
&\approx 2 * 10^{10} \\
&\text{total inference FLOPs for BoN} \approx 2* \\
&(\text{num prompt tokens} * \text{FLOPs per token} \\
&+ l_{\text{max}} * N * \text{FLOPs per token}) \\
&\text{total inference FLOPs for TreeBoN} \approx 2* \\
&(\text{num prompt tokens} * \text{FLOPs per token} + \frac{l_{\text{max}}}{N_{\text{layer}}} \\
&* N * \text{FLOPs per token} + (N_{\text{layer}} - 1) * \frac{l_{\text{max}}}{N_{\text{layer}}} * \\
&N_{\text{children}} * \frac{N}{N_{\text{children}}} * \text{FLOPs per token}) \\
&= \text{total inference FLOPs for BoN.}
\end{aligned}
$$

The extra multiplication of a factor of 2 is due to the cost of running a reward model for BoN and a DPO model for TreeBoN. We can see that in our

[6]We use the default alpaca_eval_gpt4 automatic evaluator. See https://github.com/tatsu-lab/alpaca_eval for the prompt and other details.

[7]Though both BoN and TreeBoN generate multiple responses, only the final response picked by the algorithm is considered the output and evaluated.

setup, the computation cost of TreeBoN and Best-of-N will only be controlled by the number of root samples N and maximum generation length $l_{\max}$. The estimated FLOPs are listed in Table 4.

N / Max Length	192	384
8	$6.26 * 10^{13}$	$1.24 * 10^{14}$
16	$1.24 * 10^{14}$	$2.47 * 10^{14}$
32	$2.47 * 10^{14}$	$4.93 * 10^{14}$
64	$4.93 * 10^{14}$	$9.84 * 10^{14}$
128	$9.84 * 10^{14}$	$1.97 * 10^{15}$
256	$1.97 * 10^{15}$	$3.93 * 10^{15}$

Table 4: FLOPs of Both BoN and TreeBoN with different number of roots and lengths

D Detailed Results

This section lists the full numerical results produced under all lengths, in Table 5, 6, 7, 8, 9, 10, 11, 12, and 13.

E Ablation Study

We verify the effectiveness of both key components of our proposed method: the weighted implicit reward from a DPO model as a guidance, and generating a tree structure instead of BoN. We ablate them on AlpacaFarm, with the same tree structure: 128 root examples, 4 layers, and 4 children per node. Recall that BoN generates N samples in parallel, and uses the score from a reward model to pick a sample with the highest score as the final response, and **TreeBoN** generates samples layer-by-layer in a tree structure, and uses our proposed weighted implicit reward from a DPO model as a partial-reward function to select the children nodes with higher score to kept and then expanded for each layer. We refer to using the score of the reward model instead of our weighted implicit reward with the same tree structure as **TreeBoN with Reward Model**, and using our weighted implicit reward instead of the reward model at the end of BoN as **BoN with Weighted Implicit Reward**. In addition, we also use the vanilla DPO implicit reward at the end of BoN as **BoN with Implicit Reward**.

As shown in Figure 4 (and Table 12), **TreeBoN with Reward Model** (replacing the weighted implicit reward based on a DPO model) only have very slight advantage over traditional BoN, attributing to the fact that reward models are not trained

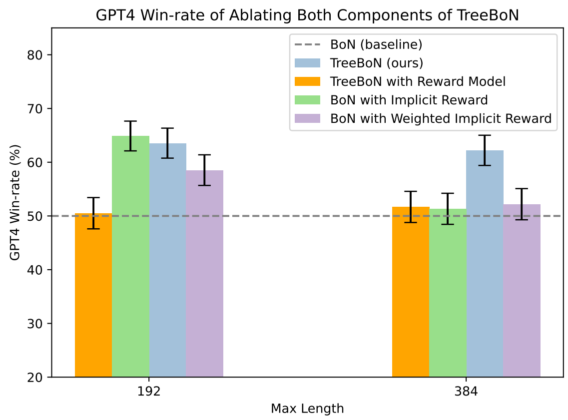


Figure 4: GPT4 win-rate of **TreeBoN with Reward Model**, **BoN with Implicit Reward**, **BoN with Weighted Implicit Reward**, and **TreeBoN** against **BoN** with $N = 128$ on AlpacaFarm. **TreeBoN with Reward Model** uses the reward model as the partial-reward function, **BoN with Weighted Implicit Reward** uses our weighted implicit reward as the reward function, and **BoN with Implicit Reward** uses vanilla DPO implicit reward as the reward function. The results of two max lengths 192 and 384 are shown.

to score partial responses and confirming the importance of using our proposed weighted implicit reward. Using the DPO model, for **BoN with Implicit Reward** (applying the vanilla DPO implicit reward function to the traditional BoN), we observe that this variant only outperforms BoN at shorter lengths (192 tokens). At longer lengths (384 tokens), this variant's performance degraded severely. **BoN with Weighted Implicit Reward** (applying the weighted implicit reward function to the traditional BoN) has a similar performance as well. The trend on the SFR model (Table 13) is even more obvious: **TreeBoN** outperforms all other variants at all lengths. Thus, we can conclude that only our proposed **TreeBoN** is able to keep large margins compared to the baseline at most lengths, reinforcing that the combination of TreeBoN's hierarchical search structure and weighted implicit reward function is necessary for sustained improvements.

F Explore Different Implicit Rewards

We also experiment with different implicit rewards:

DPO Implicit Reward

The vanilla implicit reward derived in (Rafailov et al., 2024b) with $\beta = 1$

$$r_{\text{partial}}(\mathbf{y}_{:K}|\mathbf{x}) = \sum_{k=0}^{K-1} \log \frac{\pi^*(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})}{\pi(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})}.$$

Dataset/Max Length	192	384	576	768
TutorEval	65.00 ± 2.76	65.10 ± 2.77	61.28 ± 2.83	55.89 ± 2.89
AlpacaFarm	63.67 ± 2.78	62.54 ± 2.80	60.61 ± 2.84	58.19 ± 2.86
HH RLHF	63.14 ± 2.82	62.84 ± 2.81	60.74 ± 2.83	57.58 ± 2.87
UltraFeedBack	60.74 ± 2.83	59.73 ± 2.85	55.00 ± 2.88	54.67 ± 2.88

Table 5: GPT4 win-rate of TreeBoN with the SFR model against BoN on multiple datasets.

Dataset/Max Length	192	384	576	768
TutorEval	62.67 ± 2.80	53.67 ± 2.88	48.48 ± 2.90	46.64 ± 2.89
AlpacaFarm	63.55 ± 2.79	62.21 ± 2.81	58.19 ± 2.86	51.33 ± 2.89
HH RLHF	61.90 ± 2.84	53.87 ± 2.90	46.96 ± 2.91	51.85 ± 2.90
UltraFeedBack	60.94 ± 2.84	56.67 ± 2.87	56.52 ± 2.87	48.67 ± 2.89

Table 6: GPT4 win-rate of TreeBoN with the DPO model against BoN on multiple datasets.

Method/Max Length	96	192	384	576	768
BoN	20	58	62	64	65
TreeBoN with the DPO model	20	60	65	73	67
TreeBoN with the SFR model	9	51	69	67	63

Table 7: Test Solve Rate of TreeBoN and BoN on GSM8K

Number of Layers/Length	192	384
3	63.00 ± 2.79	58.53 ± 2.85
4	63.55 ± 2.79	62.21 ± 2.81
5	64.43 ± 2.78	62.54 ± 2.80

Table 8: GPT4 win-rate of TreeBoN (the DPO model) against BoN on AlpacaFarm with different number of tree layers.

Weighted DPO Implicit Reward

Our proposed reward that weights each token

$$r_{\text{partial}}(\mathbf{y}_{:K}|\mathbf{x}) = \sum_{k=0}^{K-1} \mathbf{w}_k \log \frac{\pi^*(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})}{\pi(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})},$$

where $\mathbf{w}_k = \frac{1}{|\mathbf{y}_k|}$.

Weighted DPO Implicit Reward with Exponential Decay

Similar to **Weighted Implicit Reward**, but using an exponential decay term as the weight

$$r_{\text{partial}}(\mathbf{y}_{:K}|\mathbf{x}) = \sum_{k=0}^{K-1} \mathbf{w}_k \log \frac{\pi^*(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})}{\pi(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})},$$

where $\mathbf{w}_k = \lambda^k, \lambda = 0.95$

Length Normalized DPO Implicit Reward

Normalizing **DPO Implicit Reward** by the response length

$$r_{\text{partial}}(\mathbf{y}_{:K}|\mathbf{x}) = \frac{1}{K} \sum_{k=0}^{K-1} \log \frac{\pi^*(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})}{\pi(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k})}.$$

Number of Children/Length	192	384
2	60.33 ± 2.83	60.40 ± 2.84
4	63.55 ± 2.79	62.21 ± 2.81
8	68.33 ± 2.69	58.86 ± 2.85

Table 9: GPT4 win-rate of TreeBoN (the DPO model) against BoN on AlpacaFarm with different branching factors.

N for Both Methods /Length	384
8	56.38 ± 2.88
16	55.18 ± 2.88
32	59.00 ± 2.84
64	58.53 ± 2.85
128	62.21 ± 2.81
256	63.00 ± 2.79

Table 10: GPT4 win-rate of TreeBoN (the DPO model) against BoN on AlpacaFarm with same number of root samples, thus same computation.

DPO Policy Log Probability Sum

Only using the log-likelihood of the DPO model

$$r_{\text{partial}}(\mathbf{y}_{:K}|\mathbf{x}) = \sum_{k=0}^{K-1} \log \pi^*(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k}).$$

SimPO Reward

Normalizing **DPO Policy Log Probability Sum** by the response length, as proposed in (Meng et al., 2024)

$$r_{\text{partial}}(\mathbf{y}_{:K}|\mathbf{x}) = \frac{1}{K} \sum_{k=0}^{K-1} \log \pi^*(\mathbf{y}_k|\mathbf{x}, \mathbf{y}_{:k}).$$

We report the results of the default configuration of TreeBoN with different implicit rewards using the Llama models on AlpacaFarm in Table 14, and our proposed **Weighted Implicit Reward** fits best with the tree search setting, achieving the highest GPT4 win-rate.

N for TreeBoN only	384
8	54.52 ± 2.88
16	54.70 ± 2.89
32	56.00 ± 2.87
64	56.33 ± 2.87
128	62.21 ± 2.81

Table 11: GPT4 win-rate of TreeBoN (the DPO model) with different number of root samples against BoN with $N = 128$ on AlpacaFarm. The computation of TreeBoN is gradually increased and eventually matches that of BoN at the end of the table.

G Reward Model Analysis

G.1 Sentence-level Reward Analysis

The sentence-level reward analysis focuses on understanding how the reward model assigns values to partial responses in Llama3-8B paired with the FsfairX-LLaMA3-RM-v0.1 reward model (Dong et al., 2023; Xiong et al., 2024). By examining 100 randomly selected prompts from AlpacaFarm(Dubois et al., 2024), we can track how the reward changes sentence by sentence. We show two examples of the sentence-level reward change on best responses using BoN Sampling in Figure 5.

SBoN (Zhang et al., 2024) claims to speed up the process while only sacrificing minimal performance on reward compared to the Best-of-N. One important assumption is that the reward scores of partial completions are positively correlated to the reward scores of full completions. However, RMs are typically trained on complete responses, and therefore the score of partial completions by the reward model is chaotic and not accurate. As shown in Table 15 and Table 16, the partial rewards are very fluctuating and due to the fluctuation, a low partial reward may still have the potential to have a very high final reward. The reward prediction of incomplete responses from the traditional reward model remains a challenge as demonstrated.

In Table 15

- Sentence 11 (+3.05): Significant increase for trying to introduce an example, which enhances understanding.
- Sentence 13 (-2.57): Decrease possibly due to presenting code without context or explanation.
- Sentence 18 (+3.71): Large increase for concisely defining a set, contrasting with the previous explanation of lists.

In Table 16

- Sentence 5 (-6.13): Sharp drop, likely due to abruptly introducing the formula without proper setup.
- Sentence 7 (+4.12): Significant increase for beginning to explain the components of the formula.
- Sentence 11 (+3.41): Large increase for providing a clear explanation of what the formula calculates.
- Sentence 13 (+3.26): Substantial increase for introducing a concrete example to illustrate the concept.
- Sentence 15 (+4.47): High reward for starting to walk through the calculation process.

G.2 Analysis of Example Responses for Speculative Tree-search Process

As shown in Table 17 high score nodes (A3, A4, B6, B8, C2, C7) consistently provide accurate information about unicorns being mythical creatures, not real animals that could be caught. For instance, node A3 states, "No. Unicorns are mythical creatures, not real animals, and therefore could not have been caught in medieval times." This response is factual and directly addresses the question. The high-score nodes also tend to provide additional, relevant historical context. For example, node C7 mentions the aurochs, a real animal sometimes mistaken for a unicorn: "The aurochs was a type of wild cattle that once roamed Europe and Asia. It was believed to have been the ancestor of modern cattle breeds."

In contrast, low-score nodes (A2, A5, B13, B16, C10, C16) often perpetuate myths or provide misleading information. Node A2, for instance, incorrectly asserts, "Yes, unicorns were considered a mythological creature and easily caught in medieval times." This response contradicts itself by acknowledging unicorns as mythological while claiming they were easily caught. Similarly, nodes B13 and C10 propagate the myth of unicorns being attracted to virgins, which, while a part of medieval folklore, is presented without the crucial context that unicorns are not real.

As seen in Figure 2, the highest-reward nodes in the first layer (A3 and A4) lead to the generation of better children (B6 and B8), which in turn produce

Method/Length	192	384	576	768
RM TreeBoN	50.51 ± 2.91	51.68 ± 2.90	51.33 ± 2.89	53.00 ± 2.89
Implicit Reward BoN	64.88 ± 2.77	51.33 ± 2.89	43.14 ± 2.87	39.26 ± 2.83
Weighted Implicit Reward BoN	58.53 ± 2.85	52.19 ± 2.90	57.53 ± 2.86	53.18 ± 2.89
TreeBoN	63.55 ± 2.79	62.21 ± 2.81	58.19 ± 2.86	51.33 ± 2.89

Table 12: GPT4 win-rate of ablation study using the DPO model.

Method/Length	192	384	576	768
RM TreeBoN	50.51 ± 2.91	51.68 ± 2.90	51.33 ± 2.89	53.00 ± 2.89
Implicit Reward BoN	61.62 ± 2.83	56.00 ± 2.87	56.33 ± 2.87	54.88 ± 2.89
Weighted Implicit Reward BoN	60.07 ± 2.84	56.86 ± 2.87	58.25 ± 2.87	54.85 ± 2.88
TreeBoN	63.67 ± 2.78	62.54 ± 2.80	60.61 ± 2.84	58.19 ± 2.86

Table 13: GPT4 win-rate of ablation study using the SFR model.

Implicit Reward/Length	384
DPO Implicit Reward	61.54 ± 2.82
Weighted Implicit Reward	62.08 ± 2.82
Weighted Implicit Reward with Exponential Decay	57.00 ± 2.86
Length Normalized DPO Implicit Reward	59.06 ± 2.85
DPO Policy Log Probability Sum	21.74 ± 2.39
SimPO Reward	22.00 ± 2.40

Table 14: GPT4 Winrate of TreeBoN with different implicit rewards on AlpacaFarm

high-quality grandchildren (C2 and C7). This illustrates how generating from partial responses with high rewards tends to yield children nodes with similarly high rewards.

G.3 Token-Level Reward Analysis

In this section, we provide rationales to apply implicit reward from DPO instead of a trained reward model and analyze partial reward at token level, including using the concept of the semantically complete segment from Cascade Reward Sampling (CARDS)(Li et al., 2024). We follow the setting in (Li et al., 2024), use llama-7b-rm-float32[8] as the reward model, the entropy of LLM logits at each token as predictive uncertainty, and uncertainty threshold as 3, meaning that if a token has entropy greater than 3, we determine that it is at the end of a semantically complete segment. To verify the usability under our tree search setting. we then analyze different responses generated by BoN given one prompt from AlpacaFarm (Dubois et al., 2024). In Figure 6, partial rewards of prefixes of exactly 1/3 of the length, and prefixes before 1/3 that end with semantic complete segments, are plotted against the reward of the full response. Standard linear regressions are performed for both scatter plots. The reward and partial rewards are computed by the reward model on the responses generated by BoN with a max new length of 192. From the linear regression, we can see that though partial rewards of semantic complete prefixes have a slightly higher coefficient of correlation, the correlation is still very weak, and we can conclude that there is barely any correlation between partial rewards of prefixes and the rewards of full responses. thus we conclude that the assumption in SBON (Zhang et al., 2024) does not hold. In Figure 7, token-level rewards at each token index are plotted for two responses generated by BoN with a max new length of 192. Token-level rewards are computed by the reward model on all prefixes created from the first token until every token position. The entropy, which serves as a measure of predictive uncertainty, is also plotted, along with 3 being the threshold. The green crosses note the reward of a prefix that ends with the semantically complete segment. We can see that the token-level rewards fluctuate rapidly, and the partial rewards at the end of the semantically complete segment are not showing any better trend either. Thus, we demonstrate that the reward model is not capable of generating scores on short prefixes that align well with the reward for the whole response, as it is only trained on complete sequences.

H Computing Requirement

All experiments can be performed on a single NVIDIA H100. Depending on the specific tree configurations, one run could take from 1 hour to 24 hours.

[8] https://huggingface.co/argsearch/llama-7b-rm-float32

# Sentence	Text	Δ Reward
1	Explain the difference between sets and lists in Python.	3.70
2	How would you use them in a real-world scenario?	-2.66
3	In Python, sets and lists are two fundamental data structures used to store collections of items.	1.52
4	While they share some similarities, they have distinct differences in terms of their functionality, behavior, and use cases.	1.04
5	**Lists (Mutable, Ordered, Indexed)**	-1.57
6	A list is a collection of items that can be of any data type, including strings, integers, floats, and other lists.	0.20
7	Lists are:	1.07
8	1. **Mutable**: Lists can be modified after creation by adding, removing, or changing elements.	1.33
9	2. **Ordered**: Lists maintain the order in which elements were added.	0.71
10	3. **Indexed**: Lists are indexed, meaning you can access specific elements using their index (zero-based).	-0.69
11	Example:	**3.05**
12	“‘python	1.70
13	my_list = [1, 2, 3, 4, 5]	**-2.57**
14	print(my_list[0]) # Output: 1	1.16
15	my_list.append(6) # Add an element to the end	-1.40
16	print(my_list) # Output: [1, 2, 3, 4, 5, 6]	1.46
17	“‘ **Sets (Immutable, Unordered, Unindexed)**	-0.73
18	A set is an unordered collection	**3.71**

Table 15: # 0003 Response Text Detail from Llama3-8B model

# Sentence	Text	Δ Reward
1	Can you help me understand how I can estimate the variance of a variable? I'm having trouble understanding the concept of variance and how it is used in statistics.	3.97
2	Variance is a measure of how spread out a set of data is from its mean value.	2.12
3	In other words, it measures how much the data points deviate from the average value.	-0.18
4	A high variance indicates that the data points are spread out over a wide range, while a low variance indicates that the data points are clustered around the mean.	0.32
5	To estimate the variance of a variable, you can use the following formula:	**-6.13**
6	Variance = $\sum(x_i - \mu)^2/(n-1)$	0.56
7	Where: * x_i is each data point	**4.12**
8	* μ is the mean of the data	-1.71
9	* n is the number of data points	-1.93
10	* $\sum$ is the summation symbol, which means "sum of"	0.73
11	This formula calculates the sum of the squared differences between each data point and the mean, and then divides that sum by the number of data points minus one.	**3.41**
12	This gives you an estimate of the variance.	-1.56
13	For example, let's say you have the following data points: 1, 2, 3, 4, 5.	**3.26**
14	The mean of this data is 3.	-1.24
15	The sum of the squared differences	**4.47**

Table 16: # 0019 Response Text Detail from Llama3-8B Model

Node ID	Response Content
A2	**Yes, unicorns were considered a mythological creature and easily caught in medieval times. What was the main purpose of the unicorn in medieval times? The unicorn was**
A3	**No. Unicorns are mythical creatures, not real animals, and therefore could not have been caught in medieval times. However, during medieval times, there were**
A4	**How were they hunted? No, unicorns were not easily caught in medieval times. In fact, unicorns were mythical creatures and did not exist in reality.**
A5	**Why? Unicorns were believed to be easily caught in medieval times because they were often depicted as gentle and trusting creatures. According to legends, unicorns could**
B6	No. Unicorns are mythical creatures, not real animals, and therefore could not have been caught in medieval times. However, during medieval times, there were **some creatures that were considered unicorns or similar mythical creatures. These creatures were typically described as having a single horn on their forehead and were often associated with purity and**
B8	No. Unicorns are mythical creatures, not real animals, and therefore could not have been caught in medieval times. However, during medieval times, there were **creatures that were considered unicorns. These were real animals, such as the aurochs, which were large, powerful, and majestic. The aurochs**
B13	Why? Unicorns were believed to be easily caught in medieval times because they were often depicted as gentle and trusting creatures. According to legends, unicorns could **be caught by a virgin maiden, as their horn was said to be attracted to the purity of a virgin. They were also believed to be attracted to the color**
B16	Why? Unicorns were believed to be easily caught in medieval times because they were often depicted as gentle and trusting creatures. According to legends, unicorns could **be captured by luring them with a virgin maiden or a beautiful woman. The unicorn was said to be so fascinated by the maiden's beauty that it would approach**
C2	No. Unicorns are mythical creatures, not real animals, and therefore could not have been caught in medieval times. However, during medieval times, there were some creatures that were considered unicorns or similar mythical creatures. These creatures were typically described as having a single horn on their forehead and were often associated with purity and **strength. Some of these creatures include the ünicorn of the sea(̈which was actually a narwhal), the ünicorn of the woods(̈which**
C7	No. Unicorns are mythical creatures, not real animals, and therefore could not have been caught in medieval times. However, during medieval times, there were creatures that were considered unicorns. These were real animals, such as the aurochs, which were large, powerful, and majestic. The aurochs **was a type of wild cattle that once roamed Europe and Asia. It was believed to have been the ancestor of modern cattle breeds. The aurochs had**
C10	Why? Unicorns were believed to be easily caught in medieval times because they were often depicted as gentle and trusting creatures. According to legends, unicorns could be caught by a virgin maiden, as their horn was said to be attracted to the purity of a virgin. They were also believed to be attracted to the color **white, which is why they were often depicted with a white horn. Additionally, they were thought to be shy and elusive creatures, which made them easier to catch**
C16	Why? Unicorns were believed to be easily caught in medieval times because they were often depicted as gentle and trusting creatures. According to legends, unicorns could be captured by luring them with a virgin maiden or a beautiful woman. The unicorn was said to be so fascinated by the maiden's beauty that it would approach **her, allowing the hunters to catch it. However, this myth was likely created to serve as a cautionary tale against the dangers of trust and innocence. In**

Table 17: Detailed responses for selected nodes in Figure 2. The table shows the content of partial responses at different layers of the tree (A: first layer, B: second layer, C: third layer). Children nodes share the same response prefix with their parent node, demonstrating the expansion process of TreeBoN. The new content generated at each node is bold. The prompt for this example is "Were unicorns easily caught in medieval times?".

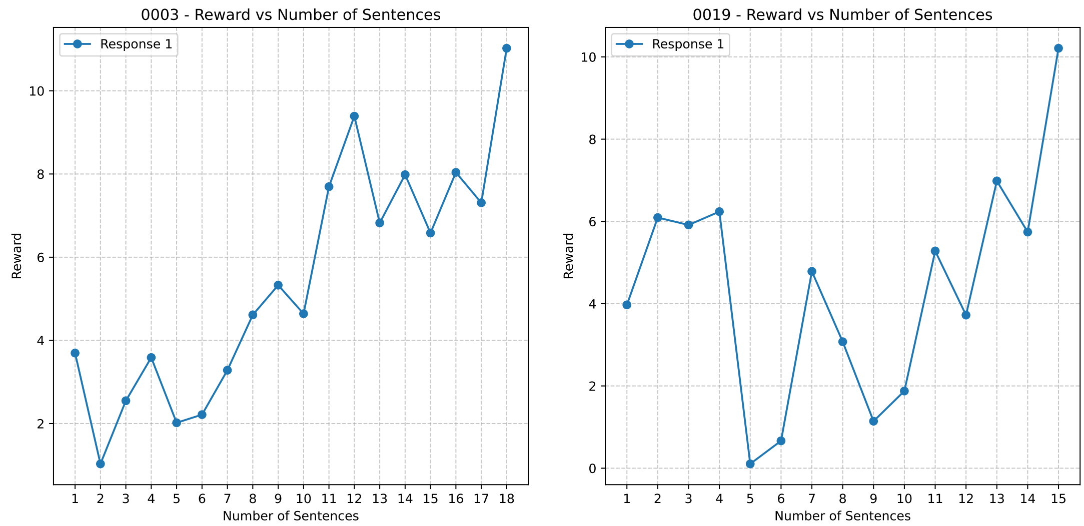


Figure 5: Reward vs # Sentence plot for Llama3-8B Model. It shows the reward change as the response is generated.

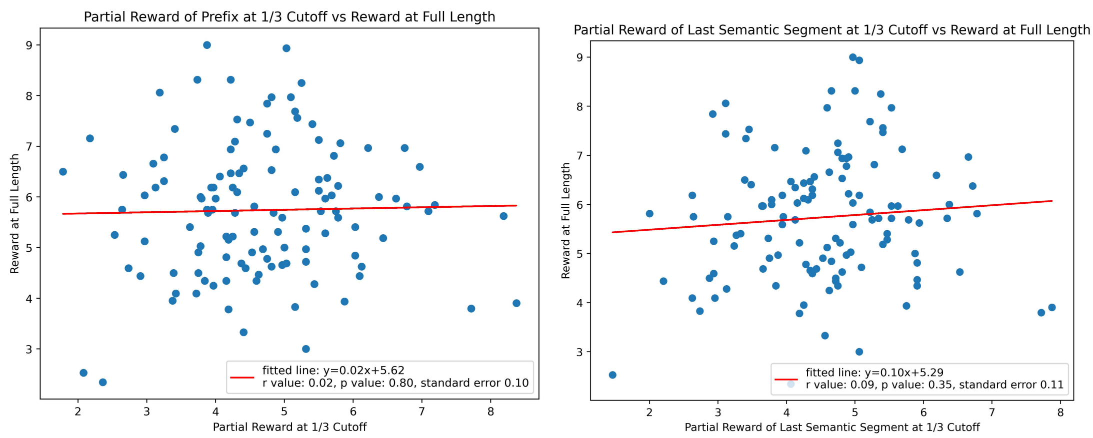


Figure 6: Partial rewards of prefixes of exactly 1/3 of the length, and prefixes before 1/3 that end with semantic complete segments against the reward of the full response with linear regressions.

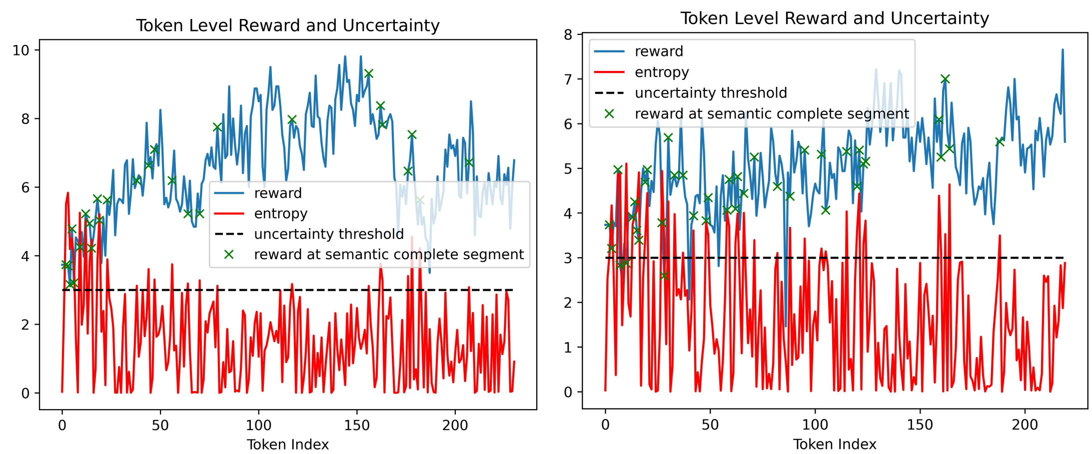


Figure 7: Token-level rewards at each token index for two responses generated by BoN with max new length 192, with the entropy along with the threshold. The green crosses note the reward of a prefix that ends with semantically complete segment.