Aho-Corasick Look-ahead Adaptive Weighted Rejection Sampling

Bilin Sun

April 2025

1 Motivation and challenges

An autoregressive LLM implicitly determines a distribution $\mathcal{D}$ over the sequence space $\Sigma^* := \bigsqcup_{i\in\mathbb{N}} \Sigma^i$ where $\Sigma \subset \mathbb{N}$ is the set of admissible token IDs.[1]

We aim to tackle a special case of controlled generation. *Controlled generation* refers to the problem of sampling from $\mathcal{D}_L = \mathcal{D}$ defined by

$$\mathcal{D}_L(s) = \frac{1}{Z}\mathcal{D}(s)\mathbb{1}_{s\in L}$$

where $Z = \sum_{s\in L} \mathcal{D}(s)$ and $L \subset \Sigma^*$ is the set of acceptable sequences. In our case, $L = \{s \mid p \text{ is not a consecutive subsequence of } s, \forall p \in P\}$ for a given set $P \subset \Sigma^*$ of banned sequences (each of which spells a forbidden phrase).

(1) Because these phrases consist of more than one token each, as we will see in the next section, simply banning one of the tokens, e.g. by masking out the logit, will not work.

(2) There may not be a unique way for the same phrase to be spelled out. For our original project, we would like to ban the phrase `linarith`, but as it turns out, it can be spelled as `lin` `arith` and `l` `inar` `ith`. When the phrase appears together with a leading space, it may even be spelled as ` l` `inar` `ith` (note the extra space).

(3) We would like to ensure the model generates proofs with probabilities *proportional to original*, except for proofs that use sledgehammers which should be set to be zero.

[1] Formally: Let and $\overline{\mathcal{D}}$ be the distribution over $\Sigma^{\mathbb{N}}$ we get from infinitely sampling from the LLM. Let $\tau : \Sigma^{\mathbb{N}} \to \mathbb{N}$ be the stopping time defined by the first occurrence of the end of sequence ([EOS]) found in the sequence, assuming the LLM does always issue an [EOS] in finite time. Let $f_\tau : \Sigma^{\mathbb{N}} \to \Sigma^*$ be the truncation function $s \mapsto s_{1:\tau(s)}$. Then $\mathcal{D} = \overline{\mathcal{D}} \circ f^{-1}$, the pushforward measure on Σ^*.

2 Why existing methods aren't appropriate

- Sequential monte carlo (SMC) methods, e.g. `https://openreview.net/pdf?id=xoXn62FzD0`, as well as importance sampling and naïve rejection sampling, over full (i.e. up to EOS) sequences, can sample many wasted proofs containing the sledgehammer before getting a proof without it. In my experiments, the LLM generates `linarith` with probability $> 1 - 1 \times 10^{-10}$ for some problems.
- Works like GeLaTo and Ctrl-G train additional models for an approximation, which involves significantly more efforts and only get approximations.
- Adaptive weighted rejection sampling, e.g. `https://arxiv.org/pdf/2504.05410`, addresses the low efficiency of SMC in our scenario, but existing works primarily deal with banned sequences of length 1.

3 Iterations of my algorithm

- If we naïvely ban the first token (e.g. `lin`) alone (e.g. by setting the logit to $-\infty$ or by setting the probability to 0), we would hurt the model's ability to output innocent phrases that also start with `lin`.
- If we naïvely ban the last token (e.g. `arith`) alone, we may still have the LLM generate the first few tokens of the phrase disproportionally frequently in an attempt to invoke the sledgehammer, only to be stopped halfway; as a result, it will be generating innocent phrases starting with the same first few tokens disproportionally often. Hence, some kind of rejection sampling over more than one is certainly necessary.
- As a naïve attempt at adaptive weighted rejection sampling, we can, for every token, sample tokens from the LLM but don't commit them (i.e. append them to the prompt for all future generations) yet, keep sampling sequentially into the a buffer until we reach the maximum possible length of all banned sequences, and check for matches with banned sequences. If there's no match, we commit these tokens sampled so far; if there's a match, we perform rejection sampling by masking the probability of this sequence to 0, and then renormalize the probabilities for all other sequences. By not sampling examples until EOS (most of which may be highly similar), this spends less wasted compute than SMC. However, this does not guarantee correctness; for instance, it's possible to sample a few tokens ending with `lin`, and then sample another few tokens starting with `arith`; `linarith` isn't detected because it's split across aboundary.
- For correctness, we adapt the Aho-Corasick string match algorithm. Given a predefined set of sequences (in our case the banned phrases), Aho-Corasick builds a trie, traverses text, moving along the trie accordingly,

and detects any match. For our purpose, we keep a FIFO queue of uncommitted tokens. When a newly generated token is added to the end of the queue, we according to the Aho-Corasick algorithm on the trie: if the new token moves us to a node on the trie with a larger depth, that means we may still be on the way to a banned token, and we cannot commit any token yet; if the new token moves us to a node with no greater depth than the current node, then we know the oldest token in the queue is guaranteed not to be starting a banned phrase, so we pop and commit the oldest token[2]; if the new token completes a banned phrase, we need to abandon tokens in the uncommitted FIFO queue, set the probability of the phrase to 0, renormalize probabilities, dial back the trie node to that at the last committed token, and sample again starting from the last committed token.

- Our algorithm is not complete with additional data structures to cache the probabilities over which we carry out zero-ing out and renormalization. We design a "FIFO tree" structure (to be explained later...) for this purpose, and manage memory carefully to avoid leaks.

4 Algorithm properties

Our algorithm provably handles challenge (1) correctly, and trivially solves (2). It satisfies (3) in a special sense (to be explained later...).

Further, time complexity-wise, it can significantly outperform SMC methods (`https://arxiv.org/pdf/2504.05410` has relevant analysis).

Moreover, this algorithm is compatible with techniques from many of the previous examples, like the normalization constant estimator from `https://arxiv.org/pdf/2504.05410`. By integrating those techniques together with our look-ahead algorithm, we can get even better runtime guarantees or satisfy (3) in more general senses.

[2] Actually could be multiple tokens; I'll clarify this later