Approximately Aligned Decoding

Daniel Melcer[*]
Khoury College of Computer Sciences
Northeastern University
Boston, MA, USA
melcer.d@northeastern.edu

Sujan Gonugondla[*]
Meta Superintelligence Labs
New York, NY, USA
sujan@meta.com

Pramuditha Perera
AWS NGDE
New York, NY, USA
pramudi@amazon.com

Haifeng Qian[*]
Nvidia
Santa Clara, CA, USA
haifengq@nvidia.com

Wen-Hao Chiang
AWS NGDE
New York, NY, USA
cwenhao@amazon.com

Yanjun Wang
AWS NGDE
New York, NY, USA
yanjunw@amazon.com

Nihal Jain
AWS NGDE
New York, NY, USA
nihjain@amazon.com

Pranav Garg
AWS NGDE
New York, NY, USA
prangarg@amazon.com

Xiaofei Ma
AWS NGDE
New York, NY, USA
xiaofeim@amazon.com

Anoop Deoras
AWS NGDE
New York, NY, USA
adeoras@amazon.com

Abstract

It is common to reject undesired outputs of Large Language Models (LLMs); however, current methods to do so require an excessive amount of computation to re-sample after a rejection, or distort the distribution of outputs by constraining the output to highly improbable tokens. We present a method, Approximately Aligned Decoding (AprAD), to balance the distortion of the output distribution with computational efficiency, inspired by algorithms from the speculative decoding literature. AprAD allows for the generation of long sequences of text with difficult-to-satisfy constraints, while amplifying low probability outputs much less compared to existing methods. We show through a series of experiments that the task-specific performance of AprAD is comparable to methods that do not distort the output distribution, while being much more computationally efficient.

1 Introduction

Large Language Models (LLMs) are able to perform many complex text manipulation tasks, and embody an incredible amount of world knowledge, but their output may contain undesirable elements such as syntactically-incorrect code, hallucinated PII, profanity, or failed tool calls. These issues, which we collectively refer to as errors or constraint violations for the remainder of the paper, may be detected with incremental parsers, regular expression matching, or even simple substring searches.

[*]Work performed while at Amazon
Code available at https://github.com/amazon-science/Approximately-Aligned-Decoding.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Each individual task that a LLM is used for may have a unique set of constraints. However, re-training a LLM to accommodate the constraints of every task is expensive, and may still not fully protect against violations. Therefore, the community has developed several methods to mitigate constraint violations without the need to retrain the language model. However, many existing methods deviate severely from the original output distribution, or have intractable performance for certain constraints.

Our contributions are as follows. First, we analyze several existing methods for avoiding constraint violations in text generated from autoregressive language models, and compare the strengths and weaknesses of each method. Second, we present a method, Approximately Aligned Decoding (AprAD), that allows for a useful midpoint in the tradeoff between computational efficiency and maintenance of the output distribution, without the need for any additional training or fine-tuning step. At its core, AprAD uses a procedure from the speculative sampling literature to determine backtracking behavior after encountering an error generation. Finally, we run a series of experiments, demonstrating that our method obtains excellent task-specific performance on both synthetic and real-world domains, without introducing an unreasonable level of inference overhead.

1.1 Related Work

Language models based on a Transformer architecture [32] have steadily become more popular, with high parameter counts, in consumer chatbot products such as OpenAI ChatGPT [23] and Anthropic Claude [2], or code generation tools such as GitHub Copilot [10] and Amazon Q Developer [5].

Sampling-based methods. Several LLM tools have introduced output control features such as schema-restricted generation [24]. For those willing to run local inference on a language model, however, there are a vast array of methods for constraining the output of a model to follow a template [3, 4, 21, 22, 27, 29], produce syntactically valid code [12, 18, 28, 30, 33], or conform to various poetry constraints [26]. These works often use *constrained generation* [6, 9] to control their outputs. As we will discuss, while constrained generation is often effective, it may result in undesirable *probability amplification*. Other methods avoid probability amplification at the expense of additional computation; for example, rejection sampling and Adaptive Sampling with Approximate Expected Futures (ASAp) [25]. A related method [1] translates the constraint into a circuit and re-samples sequences in the neighborhood of error samples to obtain a non-error sample.

Posterior estimation methods. Another class of methods [14, 34, 35], avoids errors by estimating the posterior probability of an error occurring for a given prefix, and decreasing the probability of generating prefixes that are more likely to lead to an error. These methods are usually able to quickly generate a sample with little amplification of low-probability outputs, but rely on an accurate estimator of the posterior probability of an error, which may not always be available.

Speculative decoding. A LLM's autoregressive nature can lead to high inference latency, even without constraint following. One method to combat this, *speculative decoding* [13, 19], reduces latency by transforming the sequential generation problem into a parallelizable verification problem. Several extensions such as Medusa [7] and EAGLE [15, 16] have improved the latency and efficiency of speculative decoding, and a variant, Mentored Decoding [31] further increases the speed of speculative decoding by allowing for a controlled deviation from the LLM's probability distribution.

2 Preliminaries

We first describe autoregressive language models and their properties. We then discuss speculative decoding, a method closely related to the algorithm that we will introduce.

2.1 Autoregressive Language Models

Algorithm 1 Generation with an autoregressive model

procedure Generate($P, x_{1\ldots n}$) ▷ *Initial $x_{1\ldots n}$ is the prompt*
  while Stopping condition not met **do** ▷ *Typically special* EOS *token, and length limit*
    Sample one token $x_{n+1} \sim P(\cdot|x)$
    Increment n
  return x

We assume that a vocabulary $\mathcal{V}$ of tokens is provided. An autoregressive language model is a function approximator trained to predict $P(x_n|x_{1\ldots n-1})$; the conditional probability of token $x_n \in \mathcal{V}$, given existing tokens $x_{1\ldots n-1} \in \mathcal{V}^*$. Algorithm 1 describes repeated sampling from a language model.

Note that there are several other methods for token selection; i.e. greedy selection, beam search, etc. While we focus on sampling, the techniques we present may also be applicable to other methods.

2.2 Speculative Decoding

Algorithm 2 Speculative sampling procedure

```
procedure SpecSample(P, S, n, x_{1...m})                    ▷ x_{n+1...m} are from SSM
    for i ∈ [n + 1 ... m] do                                ▷ May be vectorized instead of iterative loop
        r ← P(x_i|x_{1...i−1}) / S(x_i|x_{1...i−1})         ▷ P(·) and S(·) already calculated and cached
        with probability r do                               ▷ Always if r ≥ 1
            continue                                        ▷ Accept x_i
        else                                                ▷ Reject x_i, sample a replacement token
            R(t) = max(0, P(t|x_{1...i−1}) − S(t|x_{1...i−1}))   ▷ Calculate residuals
            return x_{1...i−1}, Sample(Normalize(R(·)))
    return x_{1...m}, Sample(P(·|x_{1...m}))                ▷ Accepted whole sequence, can sample x_{m+1}
```

Autoregressive LLMs can require considerable computational resources to evaluate. Due to the sequential nature of inference, additional parallel resources often have limited effect to decrease generation latency.

Speculative decoding [7, 13, 16, 19] decreases latency by recasting the autoregressive generation problem as one of parallelizable verification. This method assumes the existence of a small speculative model (SSM) S that approximates the LLM output, using fewer computational resources.

Given input tokens $x_{1\ldots n}$, the SSM is sampled for m tokens, resulting in tokens $x_{n+1\ldots m}$. Then, the LLM P is used to compute $P(x_{i+1}|x_{1\ldots i})$ for $i \in [n \ldots m]$ all in parallel.

Briefly, Algorithm 2 takes as input the sample from the SSM, and the token generation probability distributions from both the SSM and LLM. It determines how much of a prefix of the sample $x_{n\ldots m}$ to keep or discard, such that it is as if the resulting prefix $x_{n\ldots k}$ for $k \in [n, m]$ is drawn from the LLM. When the SSM's distribution and LLM's distribution are similar to each other, more of the prefix has a higher probability of being kept. Additionally, because the probabilities $P(\cdot|x_{1\ldots k})$ have already been computed, Algorithm 2 samples a new token x_{k+1}.

We later show that Algorithm 2 is useful for a different domain, with a somewhat different notion of quality and efficiency: violation-free generation.

3 Problem Statement and Existing Approaches

Error sets. *Error Set* $\mathcal{B} \subset \mathcal{V}^*$ is the set of strings containing errors. Without loss of generality, we assume that if string $x_{1\ldots n} \in \mathcal{B}$, then all strings with $x_{1\ldots n}$ as a prefix are also members of $\mathcal{B}$; i.e. adding more text does not negate an error. Careful design is required when, for example, profane words are substrings of benign words [8], or where un-parseable code can be made valid with additional text. Any error set may be theoretically be transformed into one that satisfies this assumption: if $x_{1\ldots n}$ is invalid, but additional text may be added to make it valid, the error set should contain $x_{1\ldots n} \circ \text{EOS}$ where EOS is the end-of-sequence token, but the error set should *not* contain $x_{1\ldots n}$. However, in many domains, it may be difficult to determine if additional text may be added to make a string valid.

Black-box constraints. $\mathcal{B}$ will often be infinite size; we treat it as a black-box indicator function. Some constraints, such as those expressible by a context-free grammar, may permit more efficient implementations [6, 9, 18]; these efficiencies largely apply to all sampling-based error-free generation approaches equally, so we do not consider their impact separately.

We define the probability distribution obtained by sampling P, except for any elements of $\mathcal{B}$:

$$\hat{P}^{\mathcal{B}}(w) = \begin{cases} w \in \mathcal{B} & 0 \\ w \notin \mathcal{B} & \frac{P(w)}{\sum_{w \notin \mathcal{B}} P(w)} \end{cases} \tag{1}$$

Problem 1. *Given an autoregressive language model P over alphabet $\mathcal{V}$, and error set $\mathcal{B} \subset \mathcal{V}^*$, provide a method to sample from $\hat{P}^{\mathcal{B}}$.*

Dense error sets. Rejection sampling is the most straightforward method for sampling from $\hat{P}^{\mathcal{B}}$; however, it may require a large number of evaluations as $\sum_{w \in \mathcal{B}} P(w)$ approaches 1. For example, consider a domain where each token has, approximately, some non-zero probability p of being an error—where the language model has a somewhat consistent error rate per token. If d tokens are generated, an output has approximately a $(1-p)^d$ probability of being error-free; thus requiring on average $\frac{1}{(1-p)^d}$ generations. We consider such domains, where the probability of generating an error approaches 1 for long generations, to have *dense* error sets.

3.1 Existing Approach: Constrained Generation

Running Example. We introduce a running example to illustrate the effects of several error-free decoding methods. There are two possible tokens, A and B, and the task is to generate sequences of length 2. All token probabilities are $1/2$, resulting in a probability of $1/4$ to generate each of the four sequences initially. Sequence AA is marked as an error, meaning that there are three possible non-error sequences. The ideal re-normalized probability is therefore $1/3$ for each remaining sequence.

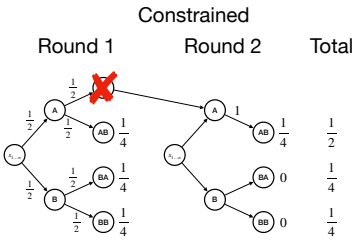


Figure 1: The entire probability mass of AA is shifted to AB.

Constrained generation attempts to solve the error-free generation problem by using a greedy algorithm: avoid selecting any tokens that immediately lead to an error.[1]

Probability amplification. Problematically, constrained generation often greatly *amplifies* low-probability samples by committing to a given prefix, even if the most probable sequences beginning with that prefix are errors. Figure 1 shows that in our running example, the probability of AB is significantly raised. Note that this distortion is even worse in low-entropy scenarios; if $P(\text{B}|\text{A})$ were lowered to 0.0001, there would still be a $1/2$ probability to sample AB. This amplification effect compounds exponentially for longer sequences.

3.2 Existing Approach: ASAp—Sampling Without Replacement

Adaptive Sampling with Approximate Expected Futures (ASAp) [25] is a technique to sample exactly from the distribution of $\hat{P}^{\mathcal{B}}$. If ASAp encounters an error during sampling, it adds it to set $B \subseteq \mathcal{B}$. Because B is finite, the conditional probabilities $\hat{P}^{B}(x_i|x_{1\ldots i-1})$ can be tractably calculated, allowing for the algorithm to sample from $\hat{P}^{B}$. The sampling process repeats with this new distribution until an error-free sample is found.

In the limit of repeated samples, B will approach $\mathcal{B}$, and therefore, $\hat{P}^{B}$ will approach $\hat{P}^{\mathcal{B}}$. Importantly, if $x \sim \hat{P}^{B}$ is sampled such that $x \notin \mathcal{B}$, this sample may be accepted, even though $B \neq \mathcal{B}$.

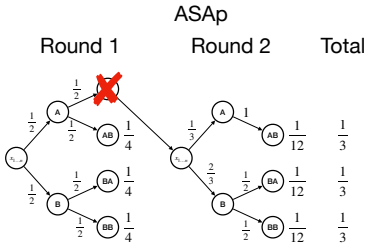


Figure 2: The probability mass of AA is distributed evenly.

In Figure 2, after sequence AA is added to set B (in the $1/4$ of cases where it is initially sampled), the conditional probabilities are recalculated as shown in Round 2. Each sequence correctly has an equal chance of being sampled in the second round.

While ASAp succeeds in cases where there are only a small number of errors that comprise the majority of the probability mass, its generation speed suffers when there are a large number of errors—each error must be discovered before it is added to B. In dense probability sets, its performance

[1] If every token leads to an immediate error, it is necessary to backtrack, though many implementations assume that there is always some token available that satisfies the constraint. Our implementation of constrained generation attempts all top-k tokens, and backtracks only if these are exhausted.

characteristics are similar to rejection sampling, as there are an exponential number of error sequences that must be discovered as generation length increases.

3.3 Existing Approaches: Posterior Estimation

We note several additional methods that, although they use different formalizations and implementations from each other, rely on a similar core idea to approximate $\hat{P}^{\mathcal{B}}$. In all cases, for any given prefix $x_{1\ldots n}$, these methods create an estimator of the likelihood of a valid sequence being generated for each prefix. This posterior probability estimation is used to sample from $\hat{P}^{\mathcal{B}}$. The differences lie in how they perform the estimation:

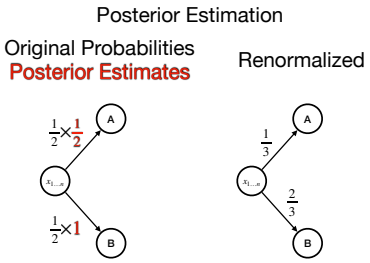


Figure 3: An accurate posterior estimator corrects the probabilities before sampling.

FUDGE [34] involves training a discriminator, usually a neural network or combination of several networks, to directly estimate this probability. SMC Steering [14] creates this estimate using Monte Carlo sampling. This method additionally incorporates optimizations such as sampling without replacement, and aggressive pruning of low-probability branches. In contrast, Ctrl-G [35] first distills a LLM into a Hidden Markov Model (HMM) with a tractable number of states (thousands or tens of thousands). If the constraint can be expressed as a Deterministic Finite Automaton (DFA) over tokens, Ctrl-G takes the product of the DFA and HMM, and calculates the probability of an error in this product system.

Gen-C [1] contains elements of both posterior-estimation and sampling-based generation, relying on both sampling the neighborhood of an error, and constructing a large constraint circuit.

While these methods exhibit impressive results on many tasks, they may face issues in domains where the posterior probability is close to 1, or where the posterior is uncorrelated with the prefix content, leading to incorrect estimates. We further discuss considerations for method selection in Section 6.1.

4 Method

An alternate view of constrained generation and ASAp is that, after encountering an error, constrained generation reuses almost the entire sample, while ASAp reuses none of it (besides to adjust probabilities in its next iteration). A natural alternative is to reuse *some* of the sample—enough to avoid excessive computation, but not enough to cause severe probability amplification.

How much of the sample should be reused? A strategy that reuses a fixed number of tokens, or that backtracks a fixed percentage of the generation length, is unlikely to effectively adapt to a variety of tasks or error sets. The prefix selection strategy should avoid discarding too much useful information, or backtracking to the middle of a low-entropy sequence. Fortunately, the prefix selection algorithm from speculative sampling presents an excellent strategy for backtracking behavior.

4.1 Speculative Sampling as a Prefix Selection Algorithm

In ASAp, where B is the set of observed errors so far, let $x = (x_1, \ldots, x_n)$ be a trace drawn from $\hat{P}^B$, such that $x \in \mathcal{B}$. ASAp will add x to B and sample again. We observe that $\hat{P}^B$ and $\hat{P}^{B\cup\{x\}}$ are almost always near-identical distributions, with $\hat{P}^{B\cup\{x\}}$ generally as a "more accurate" distribution because it incorporates an additional error sample.

Our method uses the sample $x \sim \hat{P}^B$ to approximate a sample $x' \sim \hat{P}^{B\cup\{x\}}$, in a similar manner to how speculative decoding uses a sample from a SSM to approximate a sample from a LLM—rather than the probability distributions being generated by two separate models, the distributions are both created from the same model, before and after adjusting for a violating sample. By evaluating $\text{SpecSample}(x, \hat{P}^B, \hat{P}^{B\cup\{x\}})$, our method obtains a prefix of x that can be used as a starting point for sampling again. Because the distributions of $\hat{P}^B$ and $\hat{P}^{B\cup\{x\}}$ are so close to each other, this prefix is usually most of the length of x, especially when the language model is relatively high-entropy. This process is given as Algorithm 3; we refer to it as Approximately Aligned Decoding, or AprAD.

Algorithm 3 Approximately Aligned Decoding (AprAD)

procedure ApproxAlignedDecoding($P, \mathcal{B}, x_{1\ldots n}$) ▷ *See implementation notes in Appendix H*
  ▷ $x_{1\ldots n}$ *is prompt* ◁
  $\hat{P}^B \leftarrow P$ ▷ *Adjusted probability distribution*
  $m \leftarrow n$ ▷ *Current token index*
  while Stopping condition not met **do**
    Sample one token $x_{m+1} \sim \hat{P}^B(\cdot|x_{1\ldots m})$
    Increment m
    if $x_{1\ldots m} \in \mathcal{B}$ **then**
      ▷ *Probabilities before update are queried and cached (Appendix H.3)* ◁
      $\hat{P}^{B\cup\{x\}} \leftarrow$ AddBadSample($\hat{P}^B, x_{1\ldots m}$) ▷ *Same prob. adjustment as in ASAp*
      $x_{1\ldots m} \leftarrow$ SpecSample($\hat{P}^{B\cup\{x\}}, \hat{P}^B, n, x_{1\ldots m}$) ▷ *Algorithm 2—$m$ decreases*
      $\hat{P}^B \leftarrow \hat{P}^{B\cup\{x\}}$
  return $x_{1\ldots m}$
procedure AddBadSample($\hat{P}^B, x_{1\ldots m}$)) ▷ *See implementation notes in Appendix–Algorithm 6*
  ▷ *In practice, only adjust* $x_{n+1\ldots m}$ ◁
  $\hat{P}^{B\cup\{x\}} \leftarrow \hat{P}^B$
  for $x_i \in (x_m, \ldots, x_1)$ **do**
    ▷ *Note that token sequence is reversed* ◁
    ▷ *Remove probability of* $x_{1\ldots m}$*, without changing probability of any other sequence* ◁
    $\hat{P}^{B\cup\{x\}}(x_i|x_{1\ldots i-1}) \leftarrow \hat{P}^B(x_i|x_{1\ldots i-1}) - \hat{P}^B(x_{i\ldots m}|x_{1\ldots i-1})$
    Renormalize $\hat{P}^{B\cup\{x\}}(\cdot|x_{1\ldots i-1})$
  return $\hat{P}^{B\cup\{x\}}$

In the running example (Figure 4), $\frac{P^{\{\text{AA}\}}(\text{A})}{P^{\{\}}(\text{A})} = \frac{1/3}{1/2} = 2/3$, so AprAD keeps A in $2/3$ of cases. The remaining $\frac{1}{3}$ is distributed to other tokens at the same token index; in this case, only B.

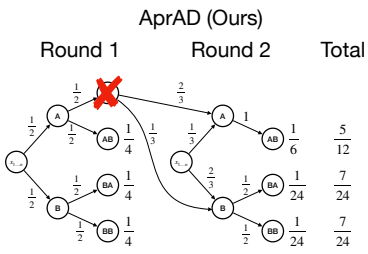


Figure 4: AprAD acts as a midpoint between constrained decoding and ASAp.

We note that AprAD still amplifies some sequence probabilities because it only invokes SpecSample after discovering an error. In the speculative decoding case, SpecSample would also be invoked if AB was directly generated by the SSM—equivalent to Round 1 in this example—but AprAD accepts AB instantly. Because the algorithm cannot iterate through every possible suffix string, it does not check whether AA contains an error except in the cases that AA is actually sampled, leading to AB being slightly overrepresented in the output probability distribution.

The resulting amplification is significantly less compared to constrained decoding, as some of the probability mass is transferred outside of the immediately neighboring sequences. Importantly, AprAD largely avoids the most extreme cases of probability amplification—if $\frac{\hat{P}^{B\cup\{x\}}(\text{A})}{\hat{P}^B(\text{A})}$ were very low, such as would arise if $P(\text{B}|\text{A}) \ll 0.5$, the procedure would be unlikely to re-select A after backtracking. This stands in contrast to constrained decoding, which always selects AB in round 2, no matter the value of $P(\text{B}|\text{A})$.

An extended illustration of the cause of probability amplification is provided in Appendix B. In Appendix C, we argue that the amplification with AprAD is bounded to a factor of 2 per each backtrack operation performed, rather than the unbounded amplification of constrained decoding.

5 Evaluation

We first exhibit AprAD's closer adherence to the ideal distribution by simulating several environments where the ideal distribution is known in advance, in Section 5.1. We additionally demonstrate that AprAD invokes the language model fewer times than ASAp.

In Section 5.2, we extend our results to a domain with a dense text-based error set. Finally, in Section 5.3, we show that our method succeeds on a task using a more complex code-based constraint.

5.1 Simulated Model with Known Ideal Distribution

We construct a testbench with a simulated language model that always returns one of three tokens (A, B, and C) with equal probability. We mark k sequences of length 3 as errors, and use the sampling method under test to sample 10000 sequences of length 3. The ideal distribution is trivial to compute—probability $\frac{1}{27-k}$ for every non-error sequence. To measure how a sampling process compares to the ideal distribution, we compute the KL-divergence between the observed distribution and ideal. Additionally, we include the *Generation Ratio*, a measure of how many times the model must be evaluated, including all backtracking, relative to output length (see Appendix A.2 for more detail).

The results are shown in Table 1, indicating that our method approximates the ideal distribution much more closely than constrained generation, with a significantly lower generation ratio than ASAp.

5.2 Lipograms (Excluded Vowels)

A piece of text that avoids using a given letter is called a lipogram; those where the excluded letter is a vowel tend to be difficult to create, especially for unconstrained LLMs [26].

We use Mistral-7B-Instruct-v0.2 [11] to generate text, where generation of a given vowel is considered an error. We prompt the LLM to perform one of five simple tasks (detailed in Appendix A). Each task is appended to instructions to avoid using the given vowel, for a total of 25 prompts. Each sampling method is used to generate a completion of up to 200 tokens. Generation is terminated if the process reaches 2000 model invocations, and the last sequence before an error was detected is returned.

Using a blind evaluation process, human raters score each completion on quality, regardless of if the constraint was followed. The raters also score completions on constraint following intent; i.e. if the model answers by selecting appropriate words that avoid the given letter, versus misspelling words, using lookalike or accented characters, etc. Completions that include the banned letter automatically receive the lowest constraint intent score. Additional information is provided in Appendix A.

The results of this evaluation are provided in Table 2, and a sample of the outputs are provided in Figure 5. All outputs and rater scores are included in the supplemental material, and additional examples are provided in Appendix I.

As shown by these results, AprAD consistently produces high-quality outputs, nearly matching the readability of unconstrained generation. Additionally, it consistently follows the intent of a constraint—while all methods except for unconstrained generation follow the constraint, constrained generation often does so in an undesirable manner, rather than by selecting appropriate words that do not contain the banned letter. Finally, AprAD is able to complete the generation or make substantial progress within the alotted computation limit, while ASAp struggles to generate more than a handful of tokens with the same inference budget.

	ASAp		Constrained		AprAD (Ours)	
Error Set	KL-div	Ratio	KL-div	Ratio	KL-div	Ratio
$\emptyset$	**0.0014**	**1.000**	**0.0014**	**1.000**	**0.0014**	**1.000**
AAA	**0.0014**	1.020	0.0075	**1.000**	0.0046	1.004
AAA, AAC	**0.0012**	1.041	0.0429	**1.000**	0.0157	1.013
AAA, ACC	**0.0013**	1.042	0.0138	**1.000**	0.0093	1.009
AAA, CCC	**0.0010**	1.044	0.0155	**1.000**	0.0074	1.010
AAA, AAB, ABA, BAA	**0.0013**	1.093	0.0504	**1.000**	0.0224	1.024
A** except AAC	**0.0014**	1.232	0.3836	**1.113**	0.1540	1.205
*** except AAA, AAB, ABA, BAA	**0.0000**	3.644	0.1771	**1.670**	0.0521	2.142
*** except AAA, BAA	**0.0000**	5.701	**0.0000**	**1.784**	**0.0000**	2.653

Table 1: KL-Divergence and generation ratios for simulated task with various error sets. Lower is better for both. Stars in the error set are wildcards; i.e. AB* means ABA, ABB, and ABC.

Method	Quality (1-5)	Constraint Intent (1-3)	Gen. Ratio
Constrained	3.56 ± 0.34	2.32 ± 0.18	$\mathbf{1.00 \pm 0.00}$
AprAD (Ours)	4.52 ± 0.23	$\mathbf{2.84 \pm 0.11}$	4.20 ± 0.91
ASAp	1.72 ± 0.23	2.36 ± 0.16	321.00 ± 79.43
Unconstrained	$\mathbf{4.68 \pm 0.17}$	1.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$

Table 2: Lipogram evaluation results, including subjective rater evaluations of output quality and adherence to the constraint, and standard error (1σ). Higher is better for Quality and Constraint Intent; lower is better for Generation Ratio. Note that constraint intent is automatically set to 1 if the banned letter is detected in the output. Generation is stopped after 2000 model invocations and the output provided as-is; for ASAp, this often resulted in very short outputs and thus extreme generation ratios and low quality scores. Several raters interpreted these short outputs as violating constraint intent.

Summarize the history of artificial intelligence without using the letter "A". AprAD (Ours)

The history of intelligent technology unfolds through key milestones in computing. In the mid-20th century, pioneering minds nurtured the seeds of this field through the development of logistic models, which evolved into expert systems in the 1960s. These systems, relying on heuristics, could solve complex problems in fields like medicine or meteorology.

Summarize the history of artificial intelligence without using the letter "A". Unconstrained

In the e**a**rly d**a**ys, brilli**a**nt minds pondered on cre**a**ting m**a**chines that could mimic hum**a**n intelligence. These pioneers, including **A**l**a**n Turing **a**nd M**a**rvin Minsky, theorized **a**nd experimented. They **a**imed to cre**a**te intelligent m**a**chines, using logic **a**nd m**a**them**a**tics.

Summarize the history of artificial intelligence without using the letter "A". ASAp

History of Intelligence

Summarize the history of artificial intelligence without using the letter "A". Constrained

In the mid-20th century, scientists delved deep into investiging compution l modeles for intelligencen modeled on the workings of the brn. Pioneers such как E.N. Turing, Clrk Quincy Shrffngton, иnddd W. Ross Quilp, and Marvin Minsky led ths fld. They explored logic-gated programs, thm brgth mchr to lf understnding the cptures of th gnntl mind.

Figure 5: Representative generation samples for all four methods, using Mistral-7B-Instruct-v0.2. Appearance of the banned letter is bolded, and non-ASCII characters (all Cyrillic in this example) are colored red and underlined. Full samples are provided in Appendix I.

5.3 BigCodeBench Hallucination Avoidance

API hallucination avoidance represents a significantly more complex constraint- instead of using standard string operations to detect lipogram constraint violations, hallucination avoidance depends on a program's imports, variable assignments, the local installation environment, etc.

We evaluate the effectiveness of each error-free sampling method on a code generation task, where the generator avoids API hallucinations. We use a wrapper around the Pyright language server [20] to detect hallucinated API calls in partial Python programs, with moderate post-processing such that the detector tends towards false negatives rather than false positives. For example, even if name `foo` is never defined, the detector does not consider the incomplete program "`example(foo.bar`" as an error, because it is possible to later add text that binds `foo`, such as "`for foo in baz)`". Text where a hallucination is detected by this program is designated as an error.

The methods are compared based on their performance on BigCodeBench v0.1 [36], a benchmark that focuses on practical programming tasks, often requiring the use of common libraries.[2]

[2]BigCodeBench uses the Apache 2.0 license. We observed instances where unconstrained LLMs use libraries present in the testing environment that aren't explicitly imported in the prompt. To better align the generation environment with the testing environment, we add all imports present in the testing environment to the prompt.

For all sampling methods, we use Starcoder2 [17], in the 7B and 15B model sizes. We generate 5 samples for each task, with temperature 0.8, and a top-p of 0.95. In addition to evaluating the pass@1 and pass@5 rates on execution-based tests, we log the specific error if execution fails. A NameError or UnboundLocalError indicates that the generation included an hallucinated API call, so we track the rate at which these errors *do not* occur. Several errors, such as AttributeError, may indicate either a hallucination or a logic error (such as improper None values), so we exclude such ambiguous errors.

Note that all methods use the same random seed, so the outputs only diverge when the detector activates. Table 3 shows the results for all tasks where the outputs diverge; Table 4 (Appendix) also includes tasks for which all methods return identical results. As the results show, the pass rate of AprAD is higher than constrained generation, with a much lower generation overhead than ASAp.

Size	Method	Pass@1	Pass@5	!NameErr@1	!NameErr@5	Gen. Ratio
15b	Unconstrained	0.21 ± 0.01	0.50 ± 0.01	0.83 ± 0.01	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
	AprAD (Ours)	$\mathbf{0.26 \pm 0.01}$	$\mathbf{0.54 \pm 0.01}$	$\mathbf{0.98 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	1.08 ± 0.01
	ASAp	$\mathbf{0.26 \pm 0.01}$	$\mathbf{0.54 \pm 0.01}$	$\mathbf{0.98 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	1.56 ± 0.12
	Constrained	0.22 ± 0.01	0.51 ± 0.01	0.93 ± 0.01	$\mathbf{1.00 \pm 0.00}$	1.02 ± 0.00
7b	Unconstrained	0.12 ± 0.01	0.35 ± 0.01	0.80 ± 0.01	$\mathbf{0.99 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
	AprAD (Ours)	0.14 ± 0.01	0.38 ± 0.01	$\mathbf{0.95 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	1.06 ± 0.01
	ASAp	$\mathbf{0.15 \pm 0.01}$	$\mathbf{0.39 \pm 0.01}$	$\mathbf{0.95 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	1.47 ± 0.07
	Constrained	0.12 ± 0.01	0.35 ± 0.01	0.89 ± 0.01	$\mathbf{0.99 \pm 0.00}$	1.02 ± 0.00

Table 3: Subset of tasks where at least one trial results in a different output for any method: 233 tasks (20.4%) for 15b, 304 tasks (26.7%) for 7b. For both model sizes, of the tasks where at least one model output is different, an average of 1.5/5 outputs are different. Lower is better for generation ratio; higher is better for all others. AprAD approaches the task performance of ASAp, with a generation ratio close to that of constrained generation. Ratio includes standard error (1σ). See Appendix A.2.

6 Discussion

As introduced in Section 3, there are several approaches to control the output of a LLM. We further discuss considerations when selecting a specific method.

6.1 Posterior Estimation-Based Methods

While posterior estimation-based techniques excel at many tasks, they tend to struggle when the probability of a constraint violation does not necessarily depend on a given text prefix. For example, the probability of a LLM generating long sequences of text without the letter 'e' is close to 0 regardless of the prompt or prefix, and mostly depends on the arbitrary behavior deep within a language model. It is unlikely that a learned discriminator or a hidden Markov model would capture this constraint, and it would require an extraordinary number of Monte Carlo samples to accurately calculate the posterior probability.

In contrast, code generation may be a more appropriate domain for posterior-estimation based methods. For example, a misleading comment that mentions a specific method all but ensures that this method will be generated on the next line. FUDGE may be able to learn that a LLM is more likely to hallucinate in some domains or with specific libraries. SMC Steering could work with a sufficient number of Monte Carlo samples, although this may be computationally expensive. For Ctrl-G, a large enough HMM could plausibly capture some of the conditions that lead to a hallucinated API call. However, neither FUDGE nor Ctrl-G would be able to quickly adapt to changes in the local environment or task.

6.2 A Spectrum of Sampling-Based Methods

Sampling-based methods are able to generate text that does not violate a constraint, even in domains where it is difficult to estimate the posterior error probability. As discussed in Section 4, the sampling behavior of AprAD lies at a midpoint between constrained generation and ASAp.

A user may wish for even further granularity in the conformance-speed tradeoff of sampling-based methods. We propose a new hyperparameter, h, and modify Line 3 of Algorithm 2 by setting r to

$\left(\frac{P(x_i|x_{1\ldots i-1})}{S(x_i|x_{1\ldots i-1})}\right)^h$; r controls the probability that a specific token in the prefix is kept after a violation occurs. When $h = 1$, this is equal to unmodified AprAD. When $h = 0$, r will always equal 1—this is equal to constrained generation. As $h \to \infty$, r will approach 0—this approaches the behavior of ASAp. We conjecture that values between these extremes allow for fine-grained control of the conformance-speed tradeoff, though we leave a more comprehensive analysis to future work.

6.3 Search Algorithms

We note that while AprAD excels at generating text that excludes an error set, it will not necessarily drive the generation process towards a specific solution. For such domains, it may be beneficial to combine our method with a search algorithm. For example, in a theorem proving context, a MCTS-like process could be used to direct a higher-level search, while AprAD is used for generating text that does not contain invalid tactic applications. We leave these combinations as future work.

7 Limitations

While our method enables generation in new domains, it still exhibits a measure of probability amplification, and additional overhead compared to constrained generation. On even denser error sets, it may be necessary to use constrained generation, or possibly the modification proposed in Section 6.2. Additionally, due to variations in testing environments and code optimization levels, our main performance metric was generation ratio rather than wall-clock time.

8 Conclusion

As our experiments show, Approximately Aligned Decoding is an effective method to generate sequences under dense language model constraints. It is straightforward to implement, requires no separate training step, introduces a manageable amount of inference overhead, and performs well on a variety of real-world and synthetic tasks.

References

[1] Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. 2025. Controllable Generation via Locally Constrained Resampling. In *The Thirteenth International Conference on Learning Representations*. International Conference on Learning Representations, Singapore, 10 pages. https://openreview.net/forum?id=8g4XgC8HPF

[2] Anthropic. 2024. Meet Claude. https://www.anthropic.com/claude

[3] Ben Athiwaratkun, Shiqi Wang, Mingyue Shang, Yuchen Tian, Zijian Wang, Sujan Kumar Gonugondla, Sanjay Krishna Gouda, Rob Kwiatowski, Ramesh Nallapati, and Bing Xiang. 2024. Token Alignment via Character Matching for Subword Completion. arXiv:2403.08688 [cs.CL] https://arxiv.org/abs/2403.08688

[4] Automorphic. 2023. Trex. automorphic-ai.

[5] AWS, Inc. 2024. AI Coding Assistant - Amazon Q Developer - AWS. https://aws.amazon.com/q/developer/.

[6] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. arXiv:2403.06988 [cs.LG] https://arxiv.org/abs/2403.06988

[7] 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. arXiv:2401.10774 [cs.LG] https://arxiv.org/abs/2401.10774

[8] Darryl Francis. 2020. The Scunthorpe Problem. *Word Ways* 53, 2 (May 2020), 4 pages. https://digitalcommons.butler.edu/wordways/vol53/iss2/12

[9] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2024. Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning. arXiv:2305.13971 [cs.CL] https://arxiv.org/abs/2305.13971

[10] Github, Inc. 2023. GitHub Copilot · Your AI Pair Programmer. https://github.com/features/copilot.

[11] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[12] Evan Jones. 2023. Llama : Add Grammar-Based Sampling. https://github.com/ggerganov/llama.cpp/pull/1773.

[13] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. arXiv:2211.17192 [cs.LG] https://arxiv.org/abs/2211.17192

[14] Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka. 2023. Sequential Monte Carlo Steering of Large Language Models using Probabilistic Programs. arXiv:2306.03081 [cs.AI] https://arxiv.org/abs/2306.03081

[15] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. arXiv:2406.16858 [cs.CL] https://arxiv.org/abs/2406.16858

[16] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. arXiv:2401.15077 [cs.LG] https://arxiv.org/abs/2401.15077

[17] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. arXiv:2402.19173 [cs.SE] https://arxiv.org/abs/2402.19173

[18] Daniel Melcer, Nathan Fulton, Sanjay Krishna Gouda, and Haifeng Qian. 2024. Constrained Decoding for Code Language Models via Efficient Left and Right Quotienting of Context-Sensitive Grammars. arXiv:2402.17988 [cs.PL] https://arxiv.org/abs/2402.17988

[19] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 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 (ASPLOS '24)*. ACM, San Diego, USA, 13 pages. https://doi.org/10.1145/3620666.3651335

[20] Microsoft. 2019. Pyright. https://github.com/microsoft/pyright

[21] Microsoft. 2023. Guidance. Microsoft.

[22] Microsoft. 2023. TypeChat. https://microsoft.github.io/TypeChat/.

[23] OpenAI. 2024. ChatGPT. https://openai.com/chatgpt/overview/

[24] OpenAI. 2024. Introducing Structured Outputs in the API. https://openai.com/index/introducing-structured-outputs-in-the-api/

[25] Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. 2024. Grammar-Aligned Decoding. arXiv:2405.21047 [cs.AI] https://arxiv.org/abs/2405.21047

[26] Allen Roush, Sanjay Basu, Akshay Moorthy, and Dmitry Dubovoy. 2023. Most Language Models can be Poets too: An AI Writing Assistant and Constrained Text Generation Studio. arXiv:2306.15926 [cs.CL] https://arxiv.org/abs/2306.15926

[27] Rahul Sengottuvelu. 2023. Jsonformer: A Bulletproof Way to Generate Structured JSON from Language Models.

[28] Grant Slatton. 2023. Added Context Free Grammar Constraints · Grantslatton/Llama.Cpp@007e26a. https://github.com/grantslatton/llama.cpp/commit/007e26a99d485007f724957fa8545331ab8d50c3.

[29] SRI. 2023. LQML. SRI Lab, ETH Zurich.

[30] Wannita Takerngsaksiri, Chakkrit Tantithamthavorn, and Yuan-Fang Li. 2023. Syntax-Aware On-the-Fly Code Completion. arXiv:2211.04673 [cs]

[31] Vivien Tran-Thien. 2024. An Optimal Lossy Variant of Speculative Decoding. https://huggingface.co/blog/vivien/optimal-lossy-variant-of-speculative-decoding

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] https://arxiv.org/abs/1706.03762

[33] Brandon T. Willard and Rémi Louf. 2023. Efficient Guided Generation for Large Language Models. arXiv:2307.09702 [cs.CL]

[34] Kevin Yang and Dan Klein. 2021. FUDGE: Controlled Text Generation With Future Discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Virtual, 3511–3535. https://doi.org/10.18653/v1/2021.naacl-main.276

[35] Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. 2024. Adaptable Logical Control for Large Language Models. arXiv:2406.13892 [cs.CL] https://arxiv.org/abs/2406.13892

[36] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. 2024. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. arXiv:2406.15877 [cs.SE] https://arxiv.org/abs/2406.15877

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Analysis of existing methods in Sections 3, 6; Method is presented in Section 4; Experimental support in Section 5.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Discussion in Section 7, as well as error set assumptions in Section 3.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: Paper is primarily empirical.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Open-weight models were used, datasets are fully described, sampling parameters are provided in Section 5, additional implementation details in supplemental material

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: Not possible at this time, but extensive implementation details are described in appendix.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Described in Section 5 and appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All experiments report 1σ SEM when relevant. Pandas library function used to calculate error. Distribution of errors is unknown.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix G.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Research ethics were followed.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: While our method discusses generative AI, it does not present a forseeable direct path to negative applications beyond those already well-known regarding LLMs.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Paper does not release data or models with high misuse risk.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: BigCodeBench is cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: Paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: Crowdsourcing was not used. Details about human rating provided in Appendix A.1.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Paper did not involve risks to humans.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLM usage described in methods section.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Lipogram evaluation details

We provide the following prompts to the language model, as well as the relevant special tokens to delimit user instructions and chat turns.

1. Write a story without using the letter "[A/E/I/O/U]".
2. Describe elephants without using the letter "[A/E/I/O/U]".
3. Provide instructions to tie a tie without using the letter "[A/E/I/O/U]".
4. Critique the Mona Lisa without using the letter "[A/E/I/O/U]".
5. Summarize the history of artificial intelligence without using the letter "[A/E/I/O/U]".

Each prompt is combined with each vowel, resulting in 25 prompts. With four sampling methods, this results in 100 total generations.

During sampling, we use a top-k of 20, and temperature of 0.8. 200 tokens was chosen as short enough to be quickly read by the human raters, and long enough to discern the sample quality. 2000 tokens was chosen as 10 times the output length, to prevent infinite computation.

A.1 Rater Instructions and Details

We create a file that only contains the 100 prompt-completion pairs, without information on which method generated each completion. All samples are shuffled in random order.

We selected four AI research colleagues not otherwise directly involved in the implementation or experimental evaluation of this method as human raters, to evaluate 25 samples each. The labels of which method corresponded to each output were hidden from the raters. We provided the following instructions to the raters:

> This file contains a set of prompts, and responses using one of several methods. Each prompt contains a constraint to not use a specific letter. Irrespective of whether the response follows the constraint, rate the response quality on a scale of 1-5 in the "Score" column, noting that generation is always cut off after 200 tokens.
>
> Additionally, rate how well the response follows the intent of the constraint in the "Follows Intent" column. Examples of not following the intent include working around the constraint by excessively dropping letters, using unnecessary accents, writing Unicode lookalike letters, or responding in a foreign language, rather than through selecting appropriate words that satisfy the constraint. This column is pre-filled with 'X' if the output contains the banned letter. Otherwise, write 1 if it violates the intent, 2 if it is ambiguous, and 3 if it does not.

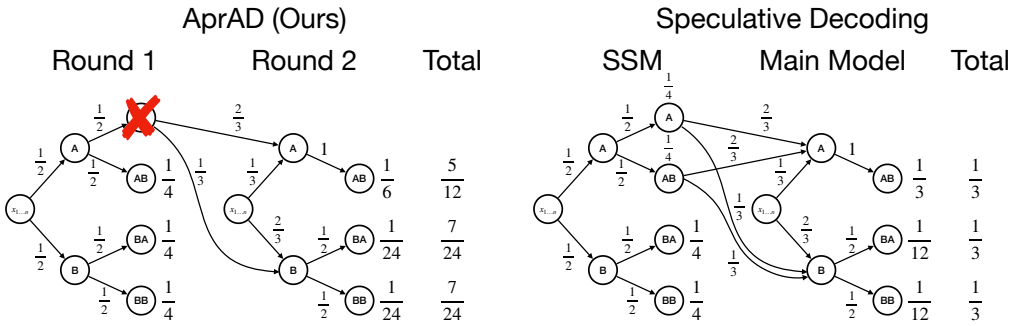
We additionally highlighted the presence of non-ASCII lookalike letters to the human raters. The complete model outputs, and the scores that each rater assigned, are provided in the supplementary material. Additional example outputs are provided in Appendix I.

A.2 Generation Ratio

The computation time for all methods is dominated by LLM evaluation time. This time is highly sensitive to variations in the testing environment and specific implementations, so we selected generation ratio as a more portable and accurate comparison between methods.

Generation ratio measures the total number of LLM invocations compared to the final output length, including all backtracking. For example, if given sampling method generates 9 tokens, backtracks to token index 7, re-uses the computed probabilities to select a new eighth token, and then generates 2 more tokens to obtain 10 tokens in all, the generation ratio is 1.1 (11 model invocations for 10 output tokens).

We note that for the BigCodeBench results, a data error caused the denominator of the generation ratio to reflect the number of tokens in the output after converting from tokens, to text, back to tokens, rather than reflecting the original number of generated tokens. This causes the generation ratios to vary slightly from their true value (usually by 0.001-0.005); to be overly conservative, we truncate the ratio to two significant digits after the decimal. Ratios for other experiments are not affected by this.



(a) AprAD Probability Distribution (from Figure 4). (b) Speculative Decoding Probability Distribution.

Figure 6: Comparison of probability distribution between AprAD and Speculative Decoding

B Extended Example of Probability Amplification

We further illustrate the cause of probability amplification, despite using the speculative decoding algorithm, through an extended example.

Consider the running example in Figure 6. In the first round, there is a $\frac{1}{4}$ probability each of selecting sequence AA, AB, BA, or BB.

In the case where AA is selected, AprAD has identical behavior to speculative decoding where P is the speculative model and $\hat{P}^{\{\text{AA}\}}$ is the main model; SpecSample is invoked in both cases. We note that in cases where $|\mathcal{B}| > 1$, after the first round, AprAD has still only observed one element x of $\mathcal{B}$ and thus must invoke SpecSample with $\hat{P}^{\{x\}}$ rather than with $\hat{P}^{\mathcal{B}}$.

In the case where AB is selected, the behavior is completely different. With speculative decoding, it is still the case that $\hat{P}^{\mathcal{B}}(A) < P(A)$, leading to a potential backtrack. With AprAD however, since a non-violating sequence is found, the process terminates—the process has no mechanism to discover $\hat{P}^{\mathcal{B}}$ other than through encounters with elements of $\mathcal{B}$.

With both BA and BB, the speculative decoding process accepts the SSM's output because $\hat{P}^{\mathcal{B}}(B) \geq P(B)$, $\hat{P}^{\mathcal{B}}(A|B) \geq P(A|B)$, and , $\hat{P}^{\mathcal{B}}(B|B) \geq P(B|B)$. AprAD also accepts these sequences immediately, as both sequences are non-violating.

Summary. The probability amplification of AprAD occurs due to the cases where SpecSample is *not* invoked.

C Bounding the Probability Amplification Factor

While the following sketch is not a rigorous proof, we argue that the probability amplification due to AprAD is at most 2 per iteration of the algorithm.

For sequences $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_m$, let $SRS(y|x, S, P)$ (SpeculativeReSample) be the probability of eventually selecting sequence y with speculative decoding, conditional on having drawn x from speculative model S.

Speculative Decoding Identity. The original concept of speculative decoding relies on the identity $P(y) = \sum_{x \in \Sigma^*} S(x) SRS(y|x, S, P)$—the probability of selecting y with the main model P should be equal to the probability of selecting y using the speculative decoding process.

SRS Subset Inequality. If $B \subseteq \mathcal{B}$, then $SRS(y|x, S, \hat{P}^B) \leq SRS(y|x, S, \hat{P}^{\mathcal{B}})$ for $y \notin \mathcal{B}$. Intuitively, $\mathcal{B}$ excludes more invalid sequences compared to $\hat{P}^B$, so the probability mass of these sequences during the re-sampling process should be distributed among all non-error sequences, including y.

Generation Subset Inequality. For similar reasons, $\hat{P}^B(x) \leq \hat{P}^{\mathcal{B}}(x)$ for $B \subseteq \mathcal{B}, x \notin \mathcal{B}$, as excluding additional errors $\mathcal{B} \setminus B$ and re-normalizing the probability means that all other sequences become more likely.

The probability of sequence $y \notin \mathcal{B}$ being generated after the first iteration of AprAD is equal to the probability that y is generated directly, plus the probability that y is re-sampled after some invalid sequence is generated:

$$AprAD(y|P, \mathcal{B}) = P(y) + \sum_{x \in \mathcal{B}} P(x) SRS(y|x, P, \hat{P}^{\{x\}})$$

We apply both of the subset inequalities described above:

$$\leq \hat{P}^{\mathcal{B}}(y) + \sum_{x \in \mathcal{B}} P(x) SRS(y|x, P, \hat{P}^{\mathcal{B}})$$

As $P(x)$ and $SRS(y|x, P, \hat{P}^{\mathcal{B}})$ are always nonnegative, we can add additional elements to the sum—expanding it to include all sequences, rather than just error sequences—while maintaining the inequality:

$$\leq \hat{P}^{\mathcal{B}}(y) + \sum_{x \in \Sigma^*} P(x) SRS(y|x, P, \hat{P}^{\mathcal{B}})$$

And finally use the speculative decoding identity:

$$\leq 2\hat{P}^{\mathcal{B}}(y)$$

Note that this amplification is per-iteration, where an iteration is defined as encountering an error sequence, potentially backtracking, and resampling. In practice, the per-iteration amplification is likely much less, as several of the inequalities involved are very loose. However, there may be a cumulative effect as more iterations are required in dense error sets.

In contrast, the probability amplification after encountering even a single error sequence with constrained decoding is unbounded, as the best non-error token may have an arbitrarily low probability.

D Additional BigCodeBench Results

Size	Method	Pass@1	Pass@5	!NameErr@1	!NameErr@5	Gen. Ratio
15b	Unconstrained	0.31 ± 0.01	0.58 ± 0.01	0.95 ± 0.00	0.99 ± 0.00	$\mathbf{1.00 \pm 0.00}$
	AprAD (Ours)	$\mathbf{0.32 \pm 0.01}$	$\mathbf{0.59 \pm 0.01}$	$\mathbf{0.98 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	1.02 ± 0.00
	ASAp	$\mathbf{0.32 \pm 0.01}$	$\mathbf{0.59 \pm 0.01}$	$\mathbf{0.98 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	1.11 ± 0.02
	Constrained	0.31 ± 0.01	0.58 ± 0.01	0.97 ± 0.00	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
7b	Unconstrained	0.20 ± 0.01	0.47 ± 0.01	0.93 ± 0.00	$\mathbf{0.99 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
	AprAD (Ours)	$\mathbf{0.21 \pm 0.01}$	0.47 ± 0.01	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.99 \pm 0.00}$	1.02 ± 0.00
	ASAp	$\mathbf{0.21 \pm 0.01}$	$\mathbf{0.48 \pm 0.01}$	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.99 \pm 0.00}$	1.13 ± 0.02
	Constrained	0.20 ± 0.01	0.47 ± 0.01	0.95 ± 0.00	$\mathbf{0.99 \pm 0.00}$	1.00 ± 0.00

Table 4: Results for each method on entirety of BigCodeBench. Note that these results are identical to those in Table 3, except that they are consistently offset and scaled to include values for tasks in which all tasks return the same result.

Table 4 includes results for the entirety of BigCodeBench; not just the tasks for which the methods diverged in their output.

E Generalization of Error-Free Decoding

Constrained generation, ASAp, and AprAD may all be generalized by their backtracking behavior after an error is discovered. Algorithm 4 shows this generalization.

Algorithm 4 Multiple error-free decoding methods differ only in their backtracking selection.

procedure ErrFreeDec($P, \mathcal{B}, x_{1\ldots n}$, Strategy)
  $\hat{P}^B \leftarrow P$
  $m \leftarrow n$ ▷ *Current token index*
  while Stopping condition not met **do**
    Sample one token $x_{m+1} \sim \hat{P}^B(\cdot|x_{1\ldots m})$
    Increment m
    if $x_{1\ldots m} \in \mathcal{B}$ **then**
      ▷ *Algorithm 5* ◁
      $\hat{P}^{B\cup\{x\}} \leftarrow$ AddBadSample($\hat{P}^B, x_{1\ldots m}$)
      ▷ *m may decrease* ◁
      $x_{1\ldots m} \leftarrow$ Strategy($\hat{P}^B, \hat{P}^{B\cup\{x\}}, x_{1\ldots m}$)
      $\hat{P}^B \leftarrow \hat{P}^{B\cup\{x\}}$
  return $x_{1\ldots m}$
procedure AprADStrategy($\hat{P}^B, \hat{P}^{B\cup\{x\}}, x_{1\ldots m}$)
  ▷ *Algorithm 2* ◁
  return SpecSample($\hat{P}^B, \hat{P}^{B\cup\{x\}}, 0, x_{1\ldots m}$)
procedure ASApStrategy($\hat{P}^B, \hat{P}^{B\cup\{x\}}, x_{1\ldots m}$)
  return [] ▷ *Backtrack to beginning*
procedure ConstrainedDecodingStrategy($\hat{P}^B, \hat{P}^{B\cup\{x\}}, x_{1\ldots m}$)
  ▷ *Delete error token but don't backtrack further* ◁
  return $x_{1\ldots m-1}$

Method	Overhead ↓	Dist. Conformance ↑	Constraint Class	Posterior Estimate
AprAD (Ours)	Medium	Medium	Black Box[a]	Not Required
ASAp [25]	High	High	Black Box[a]	Not Required
Constrained (Multiple)	Low	Low	Black Box[a]	Not Required
FUDGE [34]	Low[b]	High[c]	Prefix-Dependent[d]	Learned Discriminator
SMC Steering [14]	High	High	Black Box[a]	Sample Many Rollouts
Ctrl-G [35]	Low[b]	High[c]	Represent as DFA	Exact in distilled HMM

[a] Oracle classifies whether a given output violates constraint.
[b] Requires additional one-time training step per task.
[c] Conditional on learned or distilled model perfectly capturing LLM behavior.
[d] Requires that it is possible to determine probability of constraint violation from incomplete prefix.

Table 5: Overview of several methods for controllable generation with a LLM, with subjective estimate of inference overhead and conformance to the LLM's original output distribution, and a brief description of constraint expressivity and method of posterior estimation.

F Method Comparisons

An overview comparison of several methods is presented in Table 5.

G Compute Resources

Computation time was not precisely tracked during preliminary or final experiments; several experiments ran in parallel, resulting in a high variation of inference speed. Inference time was dominated by the BigCodeBench trials (1140 tasks, 4 methods, 2 models, 5 trials per model = 45600 generations. Order of magnitude of 10^3 output tokens per trial, resulting in about $10^7 - 10^8$ output tokens). This took about a day or two on an AWS p4d.24xlarge instance. The lipogram task was comparatively much less resource intensive, with 25 prompts, 4 models, and 200 output tokens per prompt, with a higher generation ratio than BigCodeBench, resulting in about $10^4 - 10^5$ model invocations. The simulated testbench experiments run in seconds to minutes on a consumer laptop.

H Implementation Details

We provide notes related to our implementation of the methods discussed in this paper. In particular, floating point error accumulation was a major source of issues due to invariants breaking late in the generation process; we note where recalculations may be required to avoid this.

H.1 ASAp

An implementation of ASAp is provided in Algorithm 5.

Algorithm 5 ASAp [25]

procedure ASAp($P, \mathcal{B}, x_{1\ldots n}$) ▷ $x_{1\ldots n}$ *is prompt*
  $\hat{P}^B \leftarrow P$
  while Limit not reached **do**
    Sample sequence $x_{n+1\ldots m} \sim \hat{P}^B(\cdot|x_{1\ldots n})$ until error or stopping condition
    if $x_{1\ldots m} \notin \mathcal{B}$ **then break**
    ▷ *Remove $x_{1\ldots m}$ as a possible sequence* ◁
    $\hat{P}^B \leftarrow$ AddBadSample$\hat{P}^B, x_{1\ldots m}$
  return X

H.2 Trie-Structured Probability Cache, AddBadSample, and Cached Probabilities

After each token probability distribution is generated from the language model, we add it to a trie structure to represent $\hat{P}^B$ efficiently.

The node representing prefix $x_{1\ldots m}$ contains the following:

- A single token x_m, and a pointer to a parent node representing $x_{1\ldots m-1}$
- The original probabilities generated by the LLM $P(\cdot|x_{1\ldots m})$.
- The modified conditional probabilities $\hat{P}^B(\cdot|x_{1\ldots m})$.
  - Due to floating point implementation issues, and for efficiency purposes, we store these modified probabilities un-normalized; i.e. we store a table $\hat{P}^{B*}(\cdot|x_{1\ldots m})$ where $\sum_{x_{m+1}\in\Sigma} \hat{P}^{B*}(x_{m+1}|x_{1\ldots m}) \leq 1$.
  - We track this sum in a variable, f, and divide the un-normalized probabilities by f as necessary to obtain normalized probabilities when queried. When f is small, and likely to suffer from accumulated floating point errors, we periodically recalculate it by summing the $\hat{P}^{B*}$ table.
  - Additionally, when an entry of $\hat{P}^{B*}$ is sufficiently small, or becomes negative, we assume that its value is zero, but has excessively suffered from accumulated floating point errors; we therefore set it to zero.

This structure allows for an efficient implementation of AddBadSample, as given in Algorithm 6. The same trie structure is also used to track the adjusted probabilities for the comparison methods.

H.3 Backtracking Strategies

Our implementation uses essentially the same structure as detailed in Appendix E, where it is parameterized by a backtracking strategy. However, it would be complex to maintain full probability tries representing both $\hat{P}^B$ and $\hat{P}^{B\cup\{x\}}$ every time a violating sample is found. We observe that it is unnecessary to do so; rather, we only need both probabilities along the "path" of the violating sample; i.e. $\hat{P}^B(x_1), \hat{P}^B(x_2|x_1), \ldots \hat{P}^B(x_m|x_{1\ldots m-1})$. Therefore, we invoke CollectProbabilities (Algorithm 7) before calling AddBadSample.

The SpecSample algorithm is used nearly unmodified, except that P is given by the adjusted conditional probabilities in the trie, and S is given by the cached list collected by CollectProbabilities.

Algorithm 6 An implementation-oriented description of AddBadSample

procedure AddBadSample(Node n)
  ▷ *n represents a violating sample* ◁
  $t \leftarrow n.token$
  $c \leftarrow n.parent$ ▷ *Current node*
  $r \leftarrow 1$
  while c is not null **do**
    ▷ *Calculate r as probability of violating sample in c, in original distribution* ◁
    $r \leftarrow r \times c.P(t)$
    $c.\hat{P}^{B*}(t) \leftarrow c.\hat{P}^{B*}(t) - r$
    $c.f \leftarrow c.f - r$ ▷ *Recalculate $c.f$ as sum of $c.\hat{P}^{B*}$ if needed due to FP errors*
    $t \leftarrow c.token$
    $c \leftarrow c.parent$

Algorithm 7 Collect all probabilities along a path

procedure CollectProbabilities(Node n)
  $t \leftarrow n.token$
  $c \leftarrow n.parent$ ▷ *Current node*
  $o \leftarrow []$
  while c is not null **do**
    ▷ *Calculate r as probability of violating sample in c, in original distribution* ◁
    Append$(o, c.f \times c.\hat{P}^{B*}(t))$
    $t \leftarrow c.token$
    $c \leftarrow c.parent$
  return Reverse(o)

We make one modification to constrained decoding, necessary due to our use of top-p sampling such that there may not always be an available token that satisfies the constraint in the next step. If the current leaf node of the probability trie has no non-zero entries in $\hat{P}^{B*}$, then our implementation of constrained decoding will backtrack until there is at least one available continuation.

I Additional Lipogram Examples

The following pages contain generations from 10 random prompts of the lipogram task. The examples are not cherry-picked. If the banned letter appears—this occurs in all unconstrained examples, and no other method—it is **bolded**. Non-ASCII characters are underlined and highlighted in red. We include </s> where the language model chooses to stop generation; otherwise, it was cut off after 200 tokens or 2000 model invocations, whichever occurred first. Generations from all prompts are available in the supplemental material.

Provide instructions to tie a tie without using the letter "I". **AprAD (Ours)**

To create an elegant neckwear look, follow these steps to adorn your collar:
1. Gather your tools: Choose a well- Press your collar down and adjust the top button.
2. Select the perfect knot: There are many knots to choose from; select a preferred one.
3. Dress your end: Take the end nearest to you and form a loop, called the "small" or "narrow" end.
4. Create an overlap: Cross the longer end, called the "large" end, over the loop.
5. Form a second loop: Take the small end and make another loop.
6. Create a "V" shape: Place the large end over the two loops. Adjust for symmetry.
7. Repeat: Cross the small end over the large end, then the large end over the small end to create a "V" shape once more.
8. Tuck and adjust: T

Provide instructions to tie a tie without using the letter "I". Unconstrained

To create an elegant necktie knot, follow these steps:
1. Lay the tie flat on the table with the wide end (the end you'll be tying first) facing you. The thin end (the side you'll be attaching to theshirt) should be hanging on the left.
2. Cross the wide end over the thin end and bring it behind the looped thin end.
3. Now, pass the wide end over the looped thin end from left to right, going around the back of the necktie.
4. Keeping the wide end behind the looped thin end, cross it over to the left side and bring it in front of the looped end.
5. Now, the wide end should be on the left and the looped end on the right. Cross the wide end over the looped end again to the right side.
6. Pass the wide end under

Provide instructions to tie a tie without using the letter "I". ASAp

To create

Provide instructions to tie a tie without using the letter "I". Constrained

To create an elegant necktube knot for your formal att attar, follow these steps:
1. Lay the necktube flat on the table, end towards you.
2. Fold one end about four to four and a half инches (10-12 cm) over the other end.
3. Create a loop, called the "small loop," by fold the part of the necktube above the folded end, about two to three инches (5-7.5 cm) down.
4. Create a second loop, called the "large loop," by fold the part of the necktube around your neck, and rest the large loop over the small loop.
5. Tuck the end of the necktube under the large loop, then the small loop.
6. Pass the end of the necktube under the small loop and through the hole created by the knot.
7. Tug on

Provide instructions to tie a tie without using the letter "O". **AprAD (Ours)**

1. Begin with the tie unwinded and the widest end (the "bracelet end") tucked under the neckband.
2. Bring the wider end (the "tie end") behind the neckband, then tuck it under the bracelet end and pin it in place with a thin fabric "shelf" called the "keeper."
3. Create a "Wing Tip" by lifting the neckband and slipping the tie end under it, then pull it up evenly and adjust the width until it's level with the bracelet end.
4. With the tie end still held in place with the neckband, create a "Shelf" by lifting the bracelet end and passing the tie end beneath it, then adjust the tie width again by pulling the tie end evenly.
5. Create a "Three-inch rule" by sliding the bracelet end up three inches (7.6 cm

Provide instructions to tie a tie without using the letter "O". Unconstrained

1. Begin by laying **o**ut y**o**ur tie **o**n a flat surface with the wide end, kn**o**wn as the "tail," facing y**o**u. The narr**o**wer end is called the "Tip."
2. Cr**o**ss the Tip **o**ver the Tail fr**o**m left t**o** right.
3. Bring the Tail ar**o**und the back **o**f the neck and cr**o**ss it **o**ver the Tip.
4. N**o**w, take the Tip and thread it under the l**oo**p created by the Tail.
5. Pull the Tip up thr**o**ugh the l**oo**p.
6. Adjust the p**o**siti**o**n **o**f the Tip and Tail t**o** achieve a symmetrical kn**o**t.
7. Repeat the pr**o**cess **o**n the **o**ther side, cr**o**ssing the Tip **o**ver the Tail, bringing the Tail ar**o**und the back, threading the Tip under the l**oo**p, and pulling it thr**o**ugh.
8. With b**o**th sides adjusted, adjust the kn**o**t **o**ne final time t**o** ensure a neat and symmetrical l**oo**k.</s>

Provide instructions to tie a tie without using the letter "O". ASAp

1

Provide instructions to tie a tie without using the letter "O". Constrained

1. Begin by laying the necktie flat in a figure eight shape. The wide end (called the "butt end") will be anchered at the left side, and the thin end (called the "tip end") will be adjusted at the right side.
2. Bring the right side (the side with the tip end) up and behind the left side, creating a "U" shape, then pull it straight up and tuck the tip end under the wide part at the nape (the back part) near the center.
3. Next, bring the left side up and pass it (the wide end) behind the neck, then pull it straight up and rest it against the right side.
4. With the left side still in hand, take the right side (tip end) and bring it up and pass it behind the left side.
5. Tuck the tip end behind and under the wide part (where the

Write a story without using the letter "E". **AprAD (Ours)**

In a quaint, old-world town, a mystical story unfolds. A solitary knight, Sir Gavr, hails from a distant land. His mission was vital, as his king had sought his aid. A monstrous dragon, with a long, sinuous form, had laid claim to a bountiful land.
Sir Gavr, galloping through twisting hills and past lilac valms, his stallion, Swiftpaw, carrying him swiftly. Through woodlands dark and mossy, a path narrow and winding. His armor glinting gold, a radiant sight, as knight and mount, in rhythmic flight, cut through air, crisp and light.
A labyrinth of hills and hollows, Sir Gavr and Swiftpaw, onward bound. A looming mountain, a daunting sight, as dragon's lair, a

Write a story without using the letter "E". Unconstrained

In a quaint, old-world villag**e** liv**e**d a wis**e** artisan nam**e**d Gobo. His humbl**e** abod**e** was n**e**stl**e**d 'twixt a grov**e** of tow**e**ring, v**e**rdant tr**ee**s. His days w**e**r**e** pass**e**d in crafting intricat**e**, **e**xquisit**e** pott**e**ry pi**e**c**e**s, **e**ach mor**e** b**e**autiful than th**e** last.
Gobo sourc**e**d clay from a n**e**arby str**e**am, wh**e**r**e** its cool, crystallin**e** wat**e**rs danc**e**d ov**e**r smooth, mossy p**e**bbl**e**s. Th**e** sun, a gold**e**n orb, bath**e**d th**e** clay in warm hu**e**s b**e**for**e** Gobo shap**e**d it with skill**e**d, nimbl**e** fing**e**rs. A wh**ee**l, spun by muscular arms, transform**e**d th**e** mall**e**abl**e** substanc**e** into symm**e**trical rounds.
Onc**e** shap**e**d, pi**e**c**e**s w**e**r**e** s**e**t to dry und**e**r th**e** cl**e**ar, azur**e** sky. With pati**e**nc**e**, th**e**y transform**e**d into d**e**licat**e**, hard**e**n**e**d cr**e**ations. With d**e**ft strok**e**s, Gobo adorn**e**d pots and bowls with intricat**e**,

Write a story without using the letter "E". ASAp

I.

Write a story without using the letter "E". Constrained

In a quaint, old-world town, N, a kindly cobblep, dwlt with his donk mulk. His daily task was to mak sho'd that N's villagrs' shojs staid sound. With nimbl fingrts and skillful hand, N smotd, nwd, and polisht prgrssvly.
Bys N's humbl abod, a gldng fountain splshd light thru wndw shds. It was a gif frm hm t Ols N., N's fav or old fnd, who hd lvd n ths tow n yrs ag.
Today, though, N fnd dffrlt to wk. Hs thmb, hnd, awoy aggrv, paining hm wth sharp, stng p

Describe elephants without using the letter "O". **AprAD (Ours)**

Elephants are large, gray, thick-skinned pachyderms, characteristically having extended, curved tusks, and a fan-shaped, large ear structure. They are the largest living terrestrial animals, with a distinct, massive build. These gentle giants have a wrinkled visage, and large, dangling, gray, prehensile ears, which are effective at dissipating heat. The trunk, a lengthy, muscular, and versatile appendage, is an integral feature, serving multiples uses, including breathing, grasping, and trumpeting. Their legs are pillar-like, ending in large, flat feet, which help them maneuver in diverse terrains. Their size and strength make them apex terrestrial animals, and they are crucial elements in their respective habitats.</s>

Describe elephants without using the letter "O". Unconstrained

Elephants are large, gray, thick-skinned pachyderms, kn**o**wn f**o**r their l**o**ng, curved tusks. They are the largest land animals, reaching up t**o** 13 feet in length and 10 feet in height at the sh**o**ulder. With str**o**ng, sturdy legs, they supp**o**rt up t**o** 13,000 p**o**unds. Their el**o**ngated trunk, a flexible extensi**o**n **o**f their upper lip, is used f**o**r tasks including grasping f**oo**d and water, trumpeting, and bathing. Their ears, which functi**o**n in b**o**th temperature regulati**o**n and c**o**mmunicati**o**n, can reach up t**o** 5 feet in width. These gentle giants live in herds, displaying c**o**mplex s**o**cial behavi**o**rs and intricate hierarchies.</s>

Describe elephants without using the letter "O". ASAp

Large, gray, thick-sk

Describe elephants without using the letter "O". Constrained

Elephants are large, gray, thick-skinned pachyderms, inhabiting African and Asian savannas, rain-drenched jungles, and diverse habitats in between. Kneeling, they rest their massive, curved tusks, used primarily in feeding and defense, against the jungle's underbrush. Their elastic, dexterity-laden trunks, extended appendages, twist and twirl, grabbing leaves, branches, and fruits. Their bulbus, enlarged, sensitive tips, caress and manipulate their quenching water and tender infants. Males and females, banded, marked, and tusked, live in matriarchal, familial, and herd-like structures, their distinct, rumbling rumbles, a testament, that despite their gargantuan frames, these gentle giants are an integral part, in the intr

Critique the Mona Lisa without using the letter "E". **AprAD (Ours)**

In this artwork, a captivating woman, known as La Joconda or La Hora, holds a puzzling grin. A portrait, it is, of this cryptic woman. With a backdrop of lush, rolling hills, this painting is a symphony of color and form.
This woman, clad in a rich, royal gown, sits in a chair in an unusual position, twisting slightly, inviting an intimacy that is both alluring and intriguing. A slight smirk, or is it a sly grin? This ambiguity is what draws us in, as if to ask, "Am I amusing or mocking?"
Boldly, shadows and light play in this work, as if dancing a waltz, with chiaroscuro as its music. This contrast, striking and dramatic, is what adds an almost magical quality to this artwork.
A mosaic of colors, this painting is

Critique the Mona Lisa without using the letter "E". Unconstrained

The Mona Lisa, a singular 16th-century painting, is Leoni da Vinci's epitome. This masterpiece, penned by the virtuoso artist, evokes a sense of enigma and captivation. The woman portrayed, with her enigmatic smile, gazes directly at the observer, challenging one's interpretation.
The painting's composition masterfully balances the human form against a lush background, showcasing Leonardo's virtuosic skill. The woman, dressed in a subtle, intricate robe, is contrasted by the stark, plain backdrop. Her expression, subtly shifting between coy and contemplative, invites endless analysis.
The intricate play of light and shadow on her face adds depth and dimension, as if the painting itself holds a secret. The artist's adept use of sfumato technique, where colors and tones blend

Critique the Mona Lisa without using the letter "E". ASAp

In this

Critique the Mona Lisa without using the letter "E". Constrained

I'll do my utmost to critic this Mona Lisa without utilizing "é" or any of its variants, including "é" as in "étudiant" or "è" as in "mèasuring." I'll focus on "critic" as in "judging with an analytical and authoritarian approach."
This painting, La Gioconda, by da Vinci, is an intriguing, cryptic work of art. Its woman's countanza, or counténanc, or visagio, or visagio da vita, or simply, its facia, is a captivating, almost hypnotizing, sight.
With a sfumato background that dissipatas, or dissipas, into an indistinct, unbound horizon, our ojo is drawn to its sujeto principal. Its sfumato, or "smoky," background is in striking contrast

Critique the Mona Lisa without using the letter "A". **AprAD (Ours)**

Leonymus Léonifor de Vinci's iconic piece, the 'Lizone Jocundity' or 'Smiling Lioness,' epitomizes the refined intrigue of the 16th century. This work, completed in the mid-1500s, is renowned for its subject's elusive expression, veiled in mystery.
The 'Lizone' is executed with the finest of techniques. Leonifor's virtuosity is evident in the meticulous, lifelike rendering of the figure's physiognomy, dressed in the subdued splendor of the epoch. The sitter, shrouded in the mists of time, seems to defy definition.
The composition is striking; the figure is posed, enshrouded in the folds of her veil, the folds of her robe - the folds of her being.

Critique the Mona Lisa without using the letter "A". Unconstrained

The Mona Lisa, a iconic masterpiece by Leonardo da Vinci, is a 16th century portrait rendered in oil on poplar wood. This artwork, renowned for its enigmatic smile, is noted for its exceptional fine detail and the subtle interplay of light and shadow.
The subject, believed to be Lisa Gherardini, is depicted in three-quarter view against a landscape backdrop. Her expression, often described as enigmatic or melancholic, is conveyed through her slightly parted lips, a subtle curve of her mouth, and her enigmatic gaze.
The painting showcases da Vinci's mastery of sfumato, a technique that creates a soft transition between colors and tones. This is evident in the delicate shading on the subject's face, hands, and the background landscape.
The background is a harmonious blend of mountains, water, and trees, which adds

Critique the Mona Lisa without using the letter "A". ASAp

The

Critique the Mona Lisa without using the letter "A". Constrained

The Monáá Licá, renowned work of Leonido de Vinci, is renowned for its enigmistic smile, subtle yet intriguing. This 16th century oil on poplár wood portritr é, housed in the Louvre, Fránce, boggles the mind with its complexití. The composition, with the subject's eyes meeting the observer's, invites introspection. The intricè intricàtions of the sitter's dress, mirroring the folds of the sitter's neckline, exhibit the virtuosití of the míster. The sfumátto technique, used in the rendering of the shádows, highlights the subtlety of the lighting on the subject. The delicácy of the brushwork, seen in the subtle blending of colors, reflects the meticulousness of the míster.
The over

Summarize the history of artificial intelligence without using the letter "U". AprAD (Ours)

Artificial Intelligence (AI) traces its roots to the concept of "thinking machines" in ancient Greece. Early theories on creating sentient beings were proposed by philosophers like René Descartes and Blaise Pascal. The term "Artificial Intelligence" emerged in the mid-1950s when scientists and engineers began working on creating programs capable of solving complex problems, learning from experience, and interacting with the environment.
One of the first significant achievements in AI was the development of the Logic Theorist by Allen Newell and Herbert A. Simon in 1956. This program was designed to prove mathematical theorems, and it was considered a major milestone in the field.
The 1960s and 1970s saw the advancements in AI research with the creation of expert systems, which applied knowledge in specific domains to solve complex problems. These systems, like MYCIN and EMYC

Summarize the history of artificial intelligence without using the letter "U". Unconstrained

Artificial Intelligence (AI) is a field of comp**u**ter science that foc**u**ses on creating intelligent machines capable of performing tasks that typically req**u**ire h**u**man intelligence. Its roots can be traced back to the mid-20th cent**u**ry.
In the 1940s and 1950s, early pioneers like Alan T**u**ring and Marvin Minsky explored the possibility of creating machines that co**u**ld think and learn. T**u**ring proposed the T**u**ring Test, which meas**u**res a machine's ability to imitate h**u**man conversation. Minsky and John McCarthy coined the term "Artificial Intelligence" at a workshop at Dartmo**u**th College in 1956.
Initially, AI research foc**u**sed on symbolic AI, **u**sing r**u**les and symbols to represent knowledge and reason. However, this approach faced limitations, and researchers t**u**rned to other methods.
In the late 1950s and early 1960s, there was significant interest in AI and

Summarize the history of artificial intelligence without using the letter "U". ASAp

Artificial Intelligence (AI) traces

Summarize the history of artificial intelligence without using the letter "U". Constrained

Artificial Intelligence (AI) is a field of компьютерная наука, dealing with creating machines or programs that mimic intelligent behavior. Its roots can be traced back to the mid-20th венгарда, when Alan Тюринг, an English mathematician, proposed the concept of a "logical machine" capable of performing any task that a person can.
In the 1950s and 1960s, AI research was largely driven by the idea that intelligent behavior can be achieved by programming a machine to process symbolic information, leading to the development of symbolic AI or "good old-fashioned AI" (GOFAI), which relies on logical reasoning and symbol manipation.
However, the limitations of this approach became apparent in the late 1960s, with the realization that symbolic reasoning alone was insignificant in tackling real-world problems

Describe elephants without using the letter "U". **AprAD (Ours)**

Massive land animals, elephants are the largest mammals on Earth. They have long, gray, wrinkled skin, and large, fan-like ears. Their tails are thick at the base and taper towards the tip. They possess two long front legs and a pair of smaller hind legs. They are herbivores, eating a wide variety of plants, leaves, bark, and branches. Their diet provides them with an incredible capacity to store food in their digestive tracts, which can take days to process. These gentle giants have a keen memory and form strong family bonds, living in matriarchal herds. They help one another, displaying behaviors of cooperation and protection, making them an integral part of their ecosystems.</s>

Describe elephants without using the letter "U". Unconstrained

Massive, gray creat**u**res, they are the largest living land animals. Their long, c**u**rved t**u**sks are elongated ivory appendages. They possess large, fan-like ears that help dissipate body heat. Intelligent and social, they form intricate matriarchal herds. These pachyderms have st**u**rdy legs s**u**pporting their massive bodies, ending in broad, flat feet. Their elongated tr**u**nks are prehensile, **u**sed for gathering food, drinking water, bathing, and comm**u**nicating. With strong memories, they display complex behaviors and emotions. They are essential components of vario**u**s ecosystems, partic**u**larly in Africa and Asia. These magnificent beings are called elephants.</s>

Describe elephants without using the letter "U". ASAp

Large, gray, thick-skinned pachyderms. These massive land animals are characterized by

Describe elephants without using the letter "U". Constrained

Massive, gray animals. Large, fan-like ears. Long, flexible, prehensile, and hairless necks. Long, ivory tipped, elongated front teeth for grasping and feeding. Agile, with strong, stilt-like legs, carrying heavy bodies, and large, concave backs for balance. They have large, wrinkled skin, long, flexible, and dexteritally, sensitive, gray-black, elongated, flexible and agile nose or proboscis. Intelligent, social, herd animals. They have long memories, complex behaviors and excellent learning abilities. They have long lifespan, large brain, and play a significant role in their ecosystem. Their diet consists mainly of leaves, branches, bark, bamboo shoots, and other vegetation. They are the largest land animals on Earth.</s>

Critique the Mona Lisa without using the letter "U". **AprAD (Ours)**

The Mona Lisa, a iconic masterpiece by Leonardo da Vinci, is a captivating image of a woman whose enigmatic smile and mesmerizing gaze have left viewers in awe since its inception. The portrait, renowned for its exceptional detail and realistic representation, evokes a sense of mystery and refinement.
The lady, believed to be Lisa Gherardini, is depicted while seated before a rocky, landscaped backdrop. Her visage is characterized by the delicate interplay of light and shadow, with the folds of her clothing creating an intricate, almost three-dimensional effect. Her eyes, with their faint hint of a smirk, invite the viewer into a world of enigma, daring one to decipher her emotions.
The painting's composition is noteworthy for its asymmetrical balance, which keeps the viewer constantly engaged. The delicate folds of her veil, the

Critique the Mona Lisa without using the letter "U". Unconstrained

The Mona Lisa, a iconic masterpiece by Leonardo da Vinci, is a captivating image of a woman whose enigmatic smile and inscr**u**table expression have baffled and intrig**u**ed viewers for cent**u**ries. Her soft, delicate feat**u**res and lifelike q**u**ality are achieved thro**u**gh the painter's virt**u**osic **u**se of sf**u**mato, a techniq**u**e involving the s**u**btle blending of colors to create a hazy, smoky effect.
The backgro**u**nd landscape, rendered with metic**u**lo**u**s detail, provides an idyllic setting for the enigmatic fig**u**re. The artist's exceptional skill in capt**u**ring the play of light and shadow, as well as his mastery of perspective, are evident in the way the backgro**u**nd elements recede into depth. Mona Lisa's gaze, seemingly directed at the viewer, invites one to ponder the secret tho**u**ghts hidden behind her enigmatic expression. Her hands, delicately inter

Critique the Mona Lisa without using the letter "U". ASAp

The Mon

Critique the Mona Lisa without using the letter "U". Constrained

The Mona Lisa, a iconic masterpiece by Leonardo da Vinci, is a captivating image of a woman whose enigmatic smile and inscrütableness have intrigighed viewers for over fivecenthies. The painting is renowned for its exceptional detail and lifelike portrayal of the sitter.
The composition is masterly balanced, with the sitter positioned slightly off center. Her gaze, following no discernible focal point, invites prolonged contemplation. The delicate rendering of her facial expression, with its slight hint of a smile and the faintest of eyebrows, adds to the overall alliteration.
The intricately patterned veil she wears adds a sense of mystery and intrigness. The delicate play of light and shadow enhances the sense of depth and dimensionality. The sfoglio, or smoky sfondato, in the backdrop contibtes to

Provide instructions to tie a tie without using the letter "E". **AprAD (Ours)**

1. Starting with your collar straight and flat:
2. Position your collar: Fold it up at about an inch (2.5 cm) and adjust it to sit comfortably.
3. Form a 'Width' and'Slip' or 'Four in Hand' knot:
a. Width: Pick up a small strip of fabric (about an inch or two, 2.5 cm to 5 cm) with your non-dominant hand. This is your 'width' or 'wingtip' for this knot.
b. Slip: With your dominant hand (usually your right hand), grab a loop of fabric that's around a foot (30 cm) long. This is your 'slip'.
4. Cross and Tuck:
a. Bring your slip hand up and across your width, crossing your width hand's strip.
b. Tuck

Provide instructions to tie a tie without using the letter "E". Unconstrained

1. Place the tie around your neck, with one end hanging lower than the other. This is called the "working end."
2. Cross the working end over the undone side (the side without the Windsor knot).
3. Bring the working end behind the neck, crossing it over the center of the knot.
4. Pull the working end straight down, making a loop on the side with the Windsor knot.
5. Tuck the end of the loop under the Windsor knot, towards the narrower side of the tie.
6. Pull the working end to tighten the knot, and then repeat steps 3-5 to create a second loop on the other side.
7. Cross the second loop over the first loop, just like in step 2.
8. Bring the working end behind the neck and under the first loop, crossing it over the center of the

Provide instructions to tie a tie without using the letter "E". ASAp

1

Provide instructions to tie a tie without using the letter "E". Constrained

1. Position your shirt: Worn collar flat and button up, slip your arms through armh Hols of your suit or blaz Jacket. Tuck in your shirt and fasten buttons.
2. Taut knot: Grip your narrow finish, also known as "point," with your right hand, holding it at about a 45-diagonal from your body. Wrap your "broad" or "wing tip" around your collar and cross it ov-r your finish, making a "V" Shap at your collar.
3. Tightly wrap: Grip your broad tip with your right hand, and pull it across and o'r your finish, going toward your l-ft (non-dominant) sid- and passing it thru'g your "working" n-d (right) knot loop.
4. Wrap again: Cross your broad tip to your "back," going