Efficient and Asymptotically Unbiased Constrained Decoding for Large Language Models

Haotian Ye[1*] **Himanshu Jain**[2] **Chong You**[2] **Ananda Theertha Suresh**[2]
Haowei Lin[3] **James Zou**[1] **Felix Yu**[2]
[1]Stanford University [2]Google [3]Peking University

Abstract

In real-world applications of large language models, outputs are often required to be confined: selecting items from predefined product or document sets, generating phrases that comply with safety standards, or conforming to specialized formatting styles. To control the generation, constrained decoding has been widely adopted. However, existing prefix-tree-based constrained decoding is inefficient under GPU-based model inference paradigms, and it introduces unintended biases into the output distribution. This paper introduces Dynamic Importance Sampling for Constrained Decoding (`DISC`) with GPU-based Parallel Prefix-Verification (`PPV`), a novel algorithm that leverages dynamic importance sampling to achieve theoretically guaranteed asymptotic unbiasedness and overcomes the inefficiency of prefix-tree. Extensive experiments demonstrate the superiority of our method over existing methods in both efficiency and output quality. These results highlight the potential of our methods to improve constrained generation in applications where adherence to specific constraints is essential.

1 INTRODUCTION

Large language models (LLMs) (Lee et al., 2023; Gilbert et al., 2023; Hwang and Chang, 2023; Skjuve et al., 2021) exhibit remarkable capabilities in a variety of tasks and have been widely used in different applications, including question answering (Brown et al., 2020; Black et al., 2022), coding (Achiam et al., 2023), vision understanding (Liu et al., 2023; Zhu et al., 2023; Ye et al., 2023), mathematics (Trinh et al., 2024), and more. Conventional autoregressive decoding does not restrict the tokens generated in each decoding step, which is reasonable in general use cases. However, many real-world scenarios demand particular control over the output of these models. For instance, the user may require the outputs to belong to (or be excluded from) a huge predetermined set (De Cao1 et al., 2021; Rajput et al., 2024), adhere to specific formats (Geng et al., 2023), or conform to particular grammars such as context-free grammars (Beurer-Kellner et al., 2024).

Recent studies have demonstrated *constrained decoding* as a helpful technique for aligning the output of LLM with requirements (Sun et al., 2024; De Cao1 et al., 2021). Given a predefined set of allowed outputs, it dynamically restricts the permissible tokens in each decoding step such that the final output aligns with the predefined constraints, enabling numerous applications where adherence to user-defined requirements is critical. For example, in recommendation systems where the model is asked to generate an identifier for a product code, URL, or legal term from a large database, constrained decoding ensures that the generated text corresponds to exactly one of the valid identifiers (Bevilacqua et al., 2022). Similarly, in content filtering, constrained decoding can prevent the generation of prohibited words or phrases by restricting to a set of valid tokens (Zhang et al., 2023).

While ensuring compliance with user requirements, we demonstrate in this paper that constrained decoding introduces unintended biases into the output distribution, a critical issue that existing works have not addressed (Sun et al., 2024). As illustrated in Figure 1, this bias stems from the unawareness of future constraints during autoregressive decoding. The model may start with generating high-probability tokens that could ultimately lead to sequences that vi-

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.
*Work done during internship at Google Research. Correspondence to: haotianye@stanford.edu.

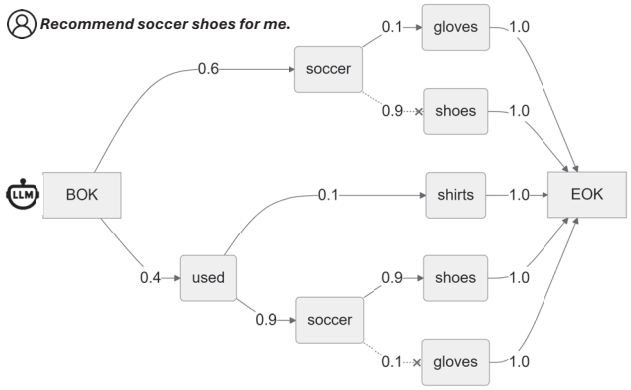

Figure 1: An illustration of how biased generation occurs when constrained decoding is applied. A user requests the model to recommend a product related to "soccer shoes". The probability of selecting each token is represented by the connection lines, where dashed lines with ✕ indicate invalid tokens (e.g., "soccer shoes" and "used soccer gloves" are not available in the recommendation list). Even though the probability of "soccer gloves" (0.06) is much lower than "used soccer shoes" (0.324), the model generates the former with a higher final probability (0.6×1) than the latter ($0.4 \times 0.9 \times 1$), since the model shortsightedly selects "soccer", unaware of the invalidity of "soccer shoes".

olate the constraints. Forced by constrained decoding to adjust, the model has to select subsequent tokens with lower probability, resulting in outputs that do not truly represent the model's learned language patterns. In Theorem 2.1, we quantify the universal impact of bias across diverse models, tasks, and datasets. This significantly degrades the quality and reliability of the generated outputs, posing severe challenges for applications that rely on accurate and unbiased generation.

Moreover, existing constrained decoding methods use prefix trees (a.k.a., *tries*), a data structure used in file systems (Bayer and Unterauer, 1977), to store all valid outputs. However, these tries are typically implemented on CPUs, leading to a heavy data communication overhead when integrating them with language models running on GPUs (Owens et al., 2008) or TPUs (Jouppi et al., 2017). Each time a token is generated, it must be transferred to CPUs to query the trie, incurring substantial communication overhead. While implementing tries on GPUs can reduce the communication cost, their fragmented structures and varying node sizes pose significant challenges for efficient parallel processing. These inefficiencies hinder the practical deployment of constrained decoding in real-world applications where speed and scalability are critical.

To address these challenges, we propose a novel approach that is asymptotically unbiased while enhancing computational efficiency for set-based constrained decoding. Our method comprises two key contributions: a **Dynamic Importance Sampling algorithm for Constrained decoding** (`DISC`), equipped with a **Parallel Prefix-Verification** (`PPV`) subroutine optimized for high-performance computing environments. `DISC` dynamically extends existing importance sampling techniques to LLMs decoding and provides a theoretically guaranteed sampling method that adjusts the token probabilities during decoding to reflect the true distributions under the constraints more accurately. As proved in Theorem 3.1, `DISC` is unbiased with a fixed expected sampling size, ensuring that the generated outputs are both compliant with the constraints and statistically consistent with the language model's learned distribution.

Regarding the efficiency issue, `PPV` significantly accelerates the decoding by alternatively "verifying" rather than "returning" all valid candidate tokens. By leveraging the fact that the language model already computes the scores for all tokens at each decoding step, it verifies only tokens with top scores to determine whether they are valid under the given constraints. This removes the requirements on prefix-tree and can be realized with a ordered candidate array with logarithmic time complexity $\mathcal{O}(\log N)$, where N is the total number of candidates in the constraint set. Importantly, the entire verification process is therefore highly parallelizable across all batches and top tokens. This bypass the inefficient trie loading, frequent data transfers between devices, and fragmented data access, significantly reducing computation time.

We run on 20 datasets across four tasks to evaluate our proposed method compared to existing baselines. This includes existing benchmarks such as entity disambiguation and document retrieval, as well as novel use cases such as generative retrieval under in-context learning. Not surprisingly, we consistently outperform trie-based methods across all datasets while being at most 8.5 times faster. Remarkably, our method can be understood as a meta-algorithm that can be applied to different techniques for certain applications, as it only improves the constrained decoding process.

We summarize our contributions as follows: (1) We uncover and theoretically analyze the bias introduced by constrained decoding in autoregressive models, highlighting its universal impact on output distributions. (2) We propose `DISC` equipped with `PPV`, a novel algorithm that provides asymptotically unbiased sampling for constrained decoding with impressive efficiency. Remarkably, it can be seamlessly integrated with other generation strategies and adapted to various constraints. (3) We provide comprehensive ex-

perimental results across various datasets and tasks, validating the effectiveness and efficiency of our approach. In summary, we aim to enhance the reliability and fairness of LLM-generated content, thereby broadening the applicability of these powerful models. [1]

2 PROBLEM DEFINITION

This section introduces notations for language modeling and background on trie-based constrained decoding. We also explain the two major challenges of existing approaches, namely the biasedness of constrained sampling and the inefficiency of trie implementation.

2.1 Notations and Background

Throughout the paper, we use $\boldsymbol{x} = [x_1, x_2, ..., x_T]$ to represent a sequence where $|\boldsymbol{x}| = T$ represents the sequence length. We use $\boldsymbol{x}_{<t} \triangleq [x_1, \cdots, x_{t-1}]$ to represent the sub-sequence of $\boldsymbol{x}$ containing the first $t-1$ entries. We say that a sequence $\boldsymbol{y}$ is a prefix of the sequence $\boldsymbol{x}$, denoted as $\boldsymbol{y} \preceq \boldsymbol{x}$, if there exists an integer $t \leq T$ such that $\boldsymbol{y} = \boldsymbol{x}_{<t}$. Given two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$, we use $\boldsymbol{x} + \boldsymbol{y}$ to represent their concatenation.

Autoregressive language modeling is the backbone of contemporary LLMs that outputs text in a *token-by-token* manner, closely mimicking the generation process of natural languages. Mathematically, let $\mathcal{V}$ be a pre-defined vocabulary (i.e., token) set[2], a sentence is represented as a sequence $\boldsymbol{x} = [x_1, \cdots, x_T]$ where $x_i \in \mathcal{V}, i = 1, \ldots, T$ are individual tokens. The autoregressive language modeling factorizes the probability of generating $\boldsymbol{x}$ as

$$P(\boldsymbol{x}) = \prod_{t=1}^{T} P(x_t | \boldsymbol{x}_{<t}).$$

Then, each conditional probability $P(x_t|\boldsymbol{x}_{<t})$ may be estimated from the output of a language model L, which we represent as $P_L(x_t|\boldsymbol{x}_{<t})$. Notice that here we use L to denote the language model with input prompts equipped, i.e., $P_L(x)$ is the output distribution conditioned on inputs that are part of L, since our study is regardless of the inputs that users provide. We will omit the subscript L for simplicity if there is no ambiguity.

Constrained decoding is a technique to align the output of a language model with particular constraints. For this paper, we focus on *set constraints*, i.e., the generated sequence is required to be from a pre-defined set $\mathcal{S}$ of sequences, which in practice could be a particular set of products, paragraphs, documents, etc. In autoregressive language generating, this requires that $P(x_t|\boldsymbol{x}_{<t})$ is zero for all $x_t \in \mathcal{V} \setminus \mathcal{A}_{\boldsymbol{x}_{<t},\mathcal{S}}$ where $\mathcal{A}_{\boldsymbol{x},\mathcal{S}}$ for general $\boldsymbol{x}, \mathcal{S}$ is a *valid set* defined as

$$\mathcal{A}_{\boldsymbol{x},\mathcal{S}} \triangleq \{v \in \mathcal{V} : \exists \boldsymbol{s} \in \mathcal{S} \text{ s.t. } \boldsymbol{x} + [v] \preceq \boldsymbol{s}\},$$

or unless the model generates the *end-of-keyword* (`EOK`) token when $\boldsymbol{x}_{<t} \in \mathcal{S}$. A typical way of checking if a token is in the valid set is through encoding $\mathcal{S}$ with a prefix-tree, which we explain next.

Prefix-tree, also known as trie, is a type of search tree for checking if a particular string is from a given set of strings. Each node in the tree represents a token in $\mathcal{V}$ with the root node representing the *begining-of-keyword* (`BOK`) token. Given a path from the root to node $\boldsymbol{x}$, a node $V \in \mathcal{V}$ is the child of the node if and only if $V \in \mathcal{A}_{\boldsymbol{x},\mathcal{S}}$. When generating a sequence using a language model, the algorithm starts from the root node and keeps generating a token that is the child of the last generated token until the entire sequence is returned (reached a leaf node). As an illustration, in Figure 1, only "soccer gloves", "used shirts", and "used soccer shoes" can be generated. We denote the output distribution of constrained decoding as $P_{\mathcal{S}}^{\text{CD}}$.

2.2 Bias Induced by Constrained Decoding

Ideally, given a language model L, we want to sample from the distribution $P_{\mathcal{S}}(\boldsymbol{x}) = P_L(\boldsymbol{x})/P_L(\mathcal{S})$. Naively, one can score all sequences in $\mathcal{S}$ and sample in hindsight, but this is impractical when $\mathcal{S}$ is large, a typical setting in constrained decoding. Although prefix-tree-based constrained decoding offers an alternative to sample, it inevitably induces bias in output distribution, a problem that existing works fail to consider.

Intuitively, the bias can be attributed to the myopic autoregressive decoding when additional constraints are imposed: the model can start generating a prefix $\boldsymbol{x} \preceq \boldsymbol{y}$ when $P_L(\boldsymbol{y})$ is large, but ends up generating a low-probability sequence $\boldsymbol{y}'$ where $\boldsymbol{x} \preceq \boldsymbol{y}'$, because $\boldsymbol{y} \notin \mathcal{S}$ and $\boldsymbol{y}' \in \mathcal{S}$. Remarkably, this is an issue only when constrained decoding is applied, because in conventional model generation, suffices with high probability will not be masked out. Below, we present in theory how biased the output is depends on the "bad" probability p_b that the model assigns on sequences outside $\mathcal{S}$, i.e., $1 - P_L(\mathcal{S})$.

Theorem 2.1. *For arbitrary probability value p_b, there exists an autoregressive model L and a candidate set $\mathcal{S}$ with $p_b = 1 - P_L(\mathcal{S})$, such that the KL*

[1] Code is available at https://github.com/YWolfeee/Large-Scale-Selection-for-LLMs.

[2] Mathematically, we assume that $\mathcal{V}$ is equipped with an order. This holds trivially in modern computers when tokens are represented as integers.

Algorithm 1 **D**ynamic **I**mportance **S**ampling for **C**onstrained Decoding (`DISC`)

Input: Language model L, constraint set $\mathcal{S}$, maximum allowed sampling steps K, temperature T
function SAMPLECANDIDATE():
  Initialize candidate sequence $\boldsymbol{a} = []$, importance score $x = 1$.
  while $\boldsymbol{a} \notin \mathcal{S}$ **do**
    Get next token distribution $P = L(\boldsymbol{a}, T)$.
    Get mask of valid tokens $\texttt{mask} = \texttt{PPV}_{\mathcal{S}}(P, \boldsymbol{a})$. ▷ Call Algorithm 2
    Sample a token t from the distribution $\frac{P \odot \texttt{mask}}{|P \odot \texttt{mask}|_1}$.
    Update $\boldsymbol{a} = \boldsymbol{a} + [t]$, $x = x \times |P \odot \texttt{mask}|_1$.
  end while
  return $\boldsymbol{a}$, x
end function
Initialize sampling step $k = 0$
while $k < K$ **do** ▷ Sample at most K candidates
  $(\boldsymbol{a}, x) = \texttt{SampleCandidate}()$
  if $\boldsymbol{a} \in \mathcal{S}$ **then**
    Update $k = k + 1$.
    if $x > \epsilon \sim \mathcal{U}[0, 1]$ **then** ▷ Accept with probability x
      Return: Sampled candidate sequence $\boldsymbol{a}$.
    end if
  end if
end while
Sample $(\boldsymbol{a}_k, x_k) = \texttt{SampleCandidate}()$ for K times, then sample $\boldsymbol{a}$ from $\{\boldsymbol{a}_k\}_{k=1}^K$ with probability $\frac{x_k}{\sum x_k}$.
Return: Sampled candidate sequence $\boldsymbol{a}$

divergence[3] *between* $P_{\mathcal{S}}$ *and* $P_{\mathcal{S}}^{CD}$ *is lower bounded by*

$$KL(P_{\mathcal{S}} \| P_{\mathcal{S}}^{CD}) \geq \Omega\Big(\ln \frac{1}{1 - p_b}\Big). \tag{1}$$

The proof of Theorem 2.1 is deferred to the appendix. It implies that the bias is universal, regardless of language models and tasks. So long as the probability outside $\mathcal{S}$ is large, the divergence can be significant.

2.3 Inefficiency of Constrained Decoding

In addition to the estimation error, trie-based constrained decoding methods also suffer from efficiency issues. Theoretically, the time complexity for constrained decoding is $\mathcal{O}(|\boldsymbol{x}||\mathcal{V}|)$, and for large $\mathcal{S}$ with large size, $|\boldsymbol{x}|$ is typically in the order of $\log(|\mathcal{S}|)$. In comparison to the LLM inference time, this might seems negligible. However, since LLMs are run on advanced computation devices such as GPUs and TPUs, having a trie run on CPUs means *at each decoding step*, current prefixes have to be sent to CPUs for constrained decoding, and all valid subsequent tokens have to be sent back. Even constrained decoding can be run in parallel in CPUs, data communication itself is highly inefficient, incurring substantial communication overhead in integration with high-throughput GPU-based LLM inference.

A seemingly simple solution is to have a trie ran purely on GPUs for constrained decoding. However, to the best of our knowledge, whether an efficient GPU trie algorithm exists remains to be studied in the first place. Two fundamental challengs have to be addressed. First, as the number of children can vary substantially from node to node, maintaining a tire that consists of a huge number of fragments with different sizes on GPUs, even if not impossible, is difficult and unfriendly to parallel computation. Second, it is unclear how to search multiple candidates on tries in a vectorized manner, such that the resources of GPUs can be fully leveraged. Given that the stopping time and size of children node are intrinsically different from candidates to candidates, organizing them into vectors for parallel computation remains difficult.

Overall, conventional trie-based constrained decoding is unsuitable with LLMs in terms of estimation accuracy and efficiency. Remarkably, this misalignment is irrespective of how the constraints are represented (e.g., whether keywords are sentences, phrases, or ID lists) and how LLMs are capable. It is rather a fundamental issue impeding their downstream applications.

[3] For two discrete distribution P, Q over set $\mathcal{X}$, the KL divergence is defined as $KL(P\|Q) = \sum_{x \in \mathcal{X}} P(x) \ln \frac{P(x)}{Q(x)}$.

3 METHOD

This section presents our method to mitigate the bias and improve computational efficiency.

3.1 Dynamic Importance Sampling for Constrained Decoding (DISC)

DISC incorporates importance sampling into LLM decoding. Unlike traditional fixed-size importance sampling, DISC dynamically adjusts the sampling process, resulting in better efficiency and convergence rate.

The pseudo-code of DISC is in Algorithm 1. It keeps sampling candidate sequences using standard constrained decoding and accepts each sample with a probability computed from how well the unconstrained output aligns with the constraints. This process is continued until a candidate sequence is accepted, upon which the distribution is guaranteed to be unbiased. If all K samples are rejected, we further run an importance sampling on K new sample sequences to minimize the bias. Notice that the algorithm calls a core function $\texttt{PPV}(P, a)$ that returns a mask of valid tokens based on the current partial sequence a and the token probability distribution P. This sub-routine is GPU-optimized and is presented in the following subsection.

Theoretical Analysis. The following theorem states that DISC achieves an upper bound on the Kullback-Leibler (KL) divergence between the distribution induced by DISC and the true constrained distribution $P_{\mathcal{S}}$.

Theorem 3.1. *For any autoregressive model L (equipped with any text prompt), denote P_{IS}^K the distribution induced by Algorithm 1 with a maximum of K sampling steps under constrained decoding. Then, the KL divergence between P_{IS}^K and the true distribution $P_{\mathcal{S}}$, $KL\left(P_{\mathcal{S}} \| P_{IS}^K\right)$ is bounded by*

$$\mathcal{O}\left(\frac{p_b^K}{1-p_b} \cdot \left(\sqrt{\frac{p_b(1-p_b)}{K}} + \frac{p_b}{K}\right)\right), \tag{2}$$

where $p_b = 1 - P_L(\mathcal{S})$ is the total probability mass of sequences outside $\mathcal{S}$, and $\mathcal{O}$ hides universal constants. In addition, the expected number of sampling steps is

$$\mathbb{E}[k] = \frac{1-p_b^K}{1-p_b} + K p_b^K \leq \frac{1+e}{1-p_b}.$$

Furthermore, the algorithm is unbiased, i.e.,

$$KL(P_{\mathcal{S}} \| P_{IS}^K) \to 0,$$

with finite expectation number $\mathbb{E}[k] \to \frac{1+e}{1-p_b}$. Here the expectation is over all LLM sampling, acceptance/rejection procedure, and importance sampling.

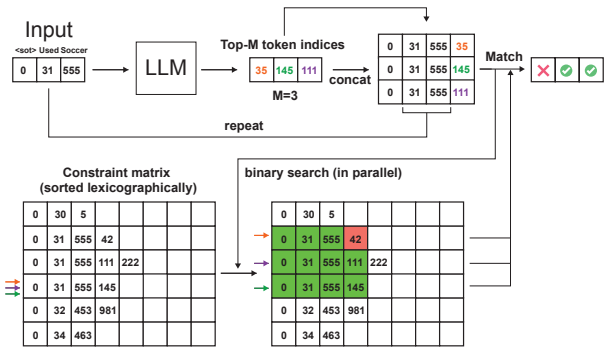

Figure 2: An illustration of PPV. Assume the input is $[0, 31, 555]$, and the top-three token candidates given by the LLM are $\{35, 145, 111\}$. PPV concatenates them to the input and forms a three-row matrix. Then, it compares each row with the prepared array $\mathcal{X}$ where each row is a keyword in $\mathcal{S}$, in parallel. Since $\mathcal{X}$ is ordered alphabetically, the comparison can be performed via a binary search. After the comparison, we can determine whether each of the three partial outputs is the prefix of a row in $\mathcal{X}$. PPV finishes by returning a vector mask indicating the validity of each candidate.

The proof of Theorem 3.1 is provided in the appendix. It shows that the estimator is asymptotically unbiased: when $K \to \infty$, the bound shrinks to zero, i.e., the estimated distribution converges to the ground-truth, and the convergence is *exponentially* fast. Even if we set $K = \infty$ so that the divergence is zero, the expected sample step is still bounded, implying the unbiased constrained decoding is realizable. Otherwise, the expected number of sampling steps is still generally much smaller than K when p_b is not close to 1. Finally, if the invalid probability p_b goes to zero, the unconstrained decoding almost generates valid sequences, and the bound shrinks to zero. This is expected because the constrained decoding does not (need to) take any effect. One may fine-tune the language model on $\mathcal{S}$ to reduce the bound in practice.

3.2 Parallel Prefix-Verification (PPV)

We next turn to the efficiency issue. The core operation in constrained decoding is to mask out invalid tokens given already generated prefixes, as specified in Line 6 of Algorithm 1. We develop the Parallel Prefix-Verification (PPV) subroutine that can benefit from GPU parallel processing by verifying whether the tokens with the highest probabilities from the language model are valid in each decoding step, avoiding complicated data transfers and trie traverse. We provide the pseudo-code in Algorithm 2 with details below, and present an illustration in Figure 2.

Table 1: **Task Statistics**. "#Corpus" refers to the number of valid entities, while "Avg. corpus token length" represents the average number of tokens per entity. "#Query" indicates the number of test queries for each task. "Vocab size" denotes the vocabulary size of the generator LLM, and "Metrics" specifies the evaluation metrics.

Task	#Corpus	Avg. corpus token length	#Query	Vocab size	Metrics
Entity Disambiguation	5,903,530	7.701	24,100	50,264	Micro F1
Document Retrieval	5,903,530	7.701	51,304	50,264	R-Precision
Entity Retrieval	4,635,922	9.485	400	128,256	Relevance score
Fact Checking	5,416,593	9.436	1,535	128,256	Relevance score

Algorithm 2 **P**arallel **P**refix-**V**erification (PPV)

1: **Prepare:** Represent the constraint set $\mathcal{S}$ by a matrix $\mathcal{X} \in \mathcal{V}^{|\mathcal{S}|\times l_{\max}}$ with rows sorted lexicographically based on the vocabulary order on $\mathcal{V}$. Pad shorter sequences to $l_{\max}$, i.e., the maximum length of sequences in $\mathcal{S}$.
2: **Input:** Token distribution P, partial sequence $\boldsymbol{a}$
3: Compute the top M entries of P and record their indices $\boldsymbol{t} = \{t_1, t_2, \ldots, t_M\}$.
4: Construct a candidate matrix $V \in \mathcal{V}^{M\times(|\boldsymbol{a}|+1)}$ where each row $V_i = \boldsymbol{a} + [t_i]$.
5: Simultaneously perform a binary search for each candidate sequence V_i in the rows of $\mathcal{X}$ to determine if V_i is a prefix of any sequence in $\mathcal{S}$. Record the results in a boolean array $\texttt{mask}_V \in \{0,1\}^M$:

$$\texttt{mask}_V[i] = \begin{cases} 1 & \text{if } \exists j \text{ s.t. } V_i \preceq \mathcal{X}_j \\ 0 & \text{otherwise} \end{cases}$$

6: Extend $\texttt{mask}_V$ to $\texttt{mask} \in \{0,1\}^{|\mathcal{V}|}$ with $\texttt{mask}[t_i] = 1$ if $\texttt{mask}_V[i] = 1$ and the remaining to 0.
7: **Return:** Mask of valid tokens mask

Array Preparation. Given the constraint set $\mathcal{S}$, PPV first performs a preprocessing in Line 1 so that it is amenable to binary search during the subsequent verification step. Specifically, the sequences in $\mathcal{S}$ are ordered by first comparing their initial tokens according to the order specified by the vocabulary set $\mathcal{V}$. If the initial tokens are identical, the comparison proceeds to the subsequent tokens in a left-to-right manner. A padding token is appended to the end of shorter keywords as necessary to ensure uniform sequence length. All ordered sequences are aggregated into an array $\mathcal{X}$ on GPUs, which can be efficiently binary searched during decoding. In practice, we also split $\mathcal{S}$ into a few pieces according to the sequence lengths to avoid too much padding that depletes disk storage space.

Parallel Verification. During the generation process, PPV selects the top M candidate tokens $\boldsymbol{t} = \{t_1, \cdots, t_M\}$ with the highest probabilities for each partially generated sequence $\boldsymbol{a}$. Each t_m is concatenated to $\boldsymbol{a}$, producing an array $\mathcal{V}$ for verification (Line 4). Then, simultaneously for each row V_m, PPV performs a binary search on the sorted array $\mathcal{X}$ to identify the index i_m, where the row $\mathcal{X}_{i_m}$ is the smallest row that is lexicographically larger than the V_m. Since A is ordered from left to right, PPV can find the target in $\log(|\mathcal{S}|)$ time. Importantly, we can derive that if $t_m \in \mathcal{A}_{\boldsymbol{a},\mathcal{S}}$ if and only if $V_m \preceq \mathcal{X}_{i_m}$. Therefore, a simple check tells whether t_m is a valid subsequent token. Notice that all computations are performed on GPUs, and additional batching across multiple queries can further extend parallel processing.

Selection of top candidate tokens. In Line 3, PPV performs a sorting and selects only the top M candidate tokens. In theory, we should set $M = |\mathcal{V}|$ to avoid missing valid tokens with low probability. However, we argue that this does not lead to a practical issue. First, most modern LLMs, when doing inference, will limit the choices to the tokens with top 50 probabilities in the first place (Dubey et al., 2024). This implies that verification essentially does not miss any valid candidates. In addition, our efficient parallel verification makes the verification cost almost identical when setting M to 50 or even 500, as verified in the appendix. Additionally, the top 50 probabilities already sum close to 1, so there is no actual difference in setting M to different values.

Advantages over prefix trees. PPV eliminates the need for CPU-based data structures like prefix trees as well as frequent data transfer between devices. In addition, its inference can be further parallelized across samples in a batch, leading to additional acceleration. Lastly, we notice that trie (often stored as JSON (Pezoa et al., 2016)) loading is extremely slower compared with array loading. The evaluation time for a dataset can be 8.5 times different at most.

4 EXPERIMENTS

This section conducts comprehensive experiments to demonstrate the efficacy of our methods. Without further explanation, DISC is always equipped with PPV.

Table 2: The Micro F1 score of unconstrained decoding (Vanilla), trie-based constrained decoding, and DISC on Entity Disambiguation across six datasets and two models. **Avg.** is the average score across datasets. The best score is bold, and Δ (in percentage) is the score improvement from Vanilla to DISC ($K = 4$).

Dataset	ACE2004		AIDA		AQUAINT		CWEB		MSNBC		WIKI		Avg.	
Models	BLINK	AY2	BLINK	AY2	BLINK	AY2	BLINK	AY2	BLINK	AY2	BLINK	AY2	BLINK	AY2
Vanilla	0.794	0.802	0.830	0.770	0.791	0.792	0.630	0.615	0.761	0.742	0.778	0.787	0.764	0.751
Trie	0.809	0.848	0.830	0.796	0.792	0.816	0.629	0.635	0.770	0.761	0.779	0.803	0.768	0.776
DISC ($K = 1$)	0.809	0.841	0.830	0.797	0.792	0.814	0.629	0.635	0.770	0.765	0.779	0.803	0.768	0.776
DISC ($K = 2$)	0.844	0.848	0.852	0.803	0.813	0.821	0.664	0.647	0.803	0.781	0.805	0.811	0.797	0.785
DISC ($K = 3$)	0.844	0.848	0.852	0.808	**0.813**	**0.831**	0.665	**0.651**	0.796	**0.782**	0.808	0.810	0.796	0.788
DISC ($K = 4$)	**0.844**	**0.849**	**0.860**	**0.810**	0.812	0.827	**0.667**	0.651	**0.810**	0.781	**0.810**	**0.816**	**0.800**	**0.789**
Δ (%)	+6.37	+5.89	+3.63	+5.27	+2.61	+4.34	+5.95	+5.80	+6.41	+5.13	+4.13	+3.73	+4.79	+5.00

Table 3: The R-Precision of all methods on Document Retrieval across eleven datasets. Similar to Table 2.

Dataset	AY2	CWEB	ELI5	FEV	HoPo	NQ	zsRE	TREx	TQA	WNED	WoW	Avg.
Vanilla	0.884	0.647	0.0982	0.757	0.237	0.484	0.823	0.675	0.544	0.818	0.506	0.588
Trie-based	0.894	0.670	0.104	0.786	0.265	0.538	0.860	0.754	0.556	0.852	0.502	0.616
DISC ($K = 1$)	0.900	0.671	0.104	0.790	0.266	0.522	0.862	0.749	0.561	0.847	0.508	0.616
DISC ($K = 2$)	0.905	0.683	0.116	0.803	0.284	0.563	0.895	0.773	0.591	**0.863**	0.528	0.637
DISC ($K = 3$)	0.907	0.683	0.113	0.803	0.287	0.562	**0.906**	**0.774**	0.601	0.857	0.531	0.639
DISC ($K = 4$)	**0.912**	**0.687**	**0.114**	**0.804**	**0.288**	**0.566**	0.895	0.773	**0.605**	0.863	**0.532**	**0.640**
Δ (%)	+3.17	+6.30	+15.5	+6.20	+21.6	+16.9	+8.81	+14.7	+11.2	+5.47	+5.11	+8.75

4.1 Experimental Settings

Tasks & datasets. We evaluate on four tasks. The first two tasks are *Entity Disambiguation*, which tries to find the most relevant entities from a list given an article, and *Document Retrieval*, which tries to retrieve the most relevant documents. Each task contains six and eleven datasets, respectively. De Cao1 et al. (2021) fine-tunes BART models to generate entities using trie-based constrained decoding. We follow their settings and generate entities from the set $\mathcal{S}$ that contains approximately six million Wikipedia pages. These datasets help compare our methods with the conventional trie-based approach using models fine-tuned on relevant tasks. Notice that De Cao1 et al. (2021) provides results with additional tricks, such as manually setting $\mathcal{S}$ to a minimal candidate set for each query to achieve the best performance. Our paper seeks to provide a better constrained decoding algorithm, and it suffices to compare algorithms without using these tricks. That said, our method can be seamlessly combined with these add-ons. For instance, we demonstrate in the appendix that our method is still superior when combined with beam search.

Apart from existing benchmarks, we innovatively study the in-context learning setting for generative retrieval tasks, including *Entity Retrieval* from DBPedia (Hasibi et al., 2017) and *Fact Checking* with Wikipedia. We use the newly released LLAMA3.2-3B-INSTRUCT[4] as a generator. For each task, we select three query-answer pairs from the validation set and provide them in-context to the model to clarify the intentions and output formats. All datasets can be found from the information retrieval benchmark BEIR (Thakur et al., 2021).

Methods. We compare vanilla generation (without constraints), trie-based constrained decoding, and DISC equipped with PPV. For our method, the number of maximum sampling step K is set to $\{1, 2, 3, 4\}$, and the top candidate's size M is set to 50. Notice that when $K = 1$, the performance of DISC should be the same as trie-based method (but faster). For sampling, we set model temperature T to 1 and do not include length penalty (rescale probabilities by length). We use A100 GPUs and set batch size to 128.

Evaluation metrics. For Entity Disambiguation and Document Retrieval, we adopt the *InKB* micro-F1 following Le and Titov (2018), and R-precision (Fischer-Hbner and Berthold, 2017) following Petroni et al. (2020) respectively. For in-context learning tasks, we utilize relevance score, which is calculated as the average query-document relevance using TREC evaluation tool (Van Gysel and de Rijke, 2018).

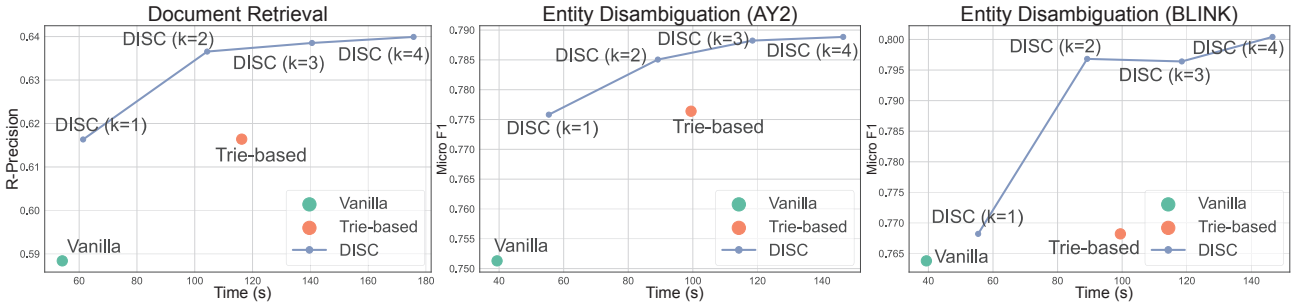

Figure 3: Time vs. Performance Comparison on Document Retrieval and Entity Disambiguation Tasks for all methods. The y-axis presents performance (R-Precision for Document Retrieval and Micro F1 scores for Entity Disambiguation), and the x-axis presents inference time that is average across all tasks belonging to that task. The improvement on time (x-axis) is resulted from PPV, and the improvement on performance (y-axis) is resulted from DISC, respectively.

4.2 Main Results

Entity Disambiguation and Document Retrieval. Tables 2 and 3 present the performance of unconstrained decoding (Vanilla), trie-based constrained decoding, and our method DISC with varying values of maximum sampling steps K. On both tasks, DISC consistently outperforms the trie-based constrained decoding across all datasets and models. On Entity Disambiguation with $K = 2$, the average Micro F1 gain compared to Vanilla is increased from 0.1% (2.5%) to 3.3% (3.4%) for the BLINK (AY2) model. For Document Retrieval, DISC achieves an average R-Precision gain of 5.2%, nearly double the 2.8% improvement achieved by the trie-based method. Remarkably, the effect of improving K quickly plateaus when K increases above 2, while the difference between $K = 1, 2$ is non-trivial. The improvements are particularly pronounced on datasets like HoPo and NQ, where the R-Precision improvements are 21.58% and 16.89% compared to the Vanilla model, respectively.

The inference time for each algorithm and model is presented in Figure 3. Results are averaged across all datasets. The advantages of DISC are evident, with $K = 1$ being significantly faster and $K = 2$ offering much better performance while maintaining higher efficiency. The consistency of this optimal trade-off across all tasks clearly highlights the benefits of PPV.

Entity Retrieval and Fact Checking. Table 4 presents the performance and efficiency comparison among different methods on the Entity Retrieval (DBPedia-Entity) and Fact Checking (Climate-FEVER) tasks. On the first task, DISC with $K = 4$ achieves a score of 0.735, more than doubling the performance of Vanilla (0.340) and 22% higher than the trie-based method. Surprisingly, this is realized with nearly half of the inference time (since $|\mathcal{S}|$ is huge is this dataset). Similarly, on the Climate-FEVER dataset, DISC with $K = 4$ attains a score of 0.295, marking an 84.4% improvement over the Vanilla model and a 2.4% increase over the trie-based method. As this in-context learning pipeline is novel and purely zero-shot, it clearly demonstrates the potential of DISC's capability to enhance performance without task-specific fine-tuning.

Table 4: The relevance score and inference time of all methods on Entity Retrieval and Fack Checking. Models are not tuned and only prompted in context.

	DBPedia-Entity		**Climate-Fever**	
	Score	Time (s)	Score	Time (s)
Vanilla	0.340	20.20	0.160	73.4
Trie-based	0.600	171.73	0.271	234.8
DISC ($K = 1$)	0.640	20.41	0.270	74.0
DISC ($K = 2$)	0.683	37.92	0.294	149.9
DISC ($K = 3$)	0.728	58.04	0.290	225.1
DISC ($K = 4$)	**0.735**	76.88	**0.295**	304.7
Delta (%)	+116.18	-	+84.44	-

5 RELATED WORKS

Constrained decoding. The objective of constrained decoding is to ensure that the generated sequence complies with the specified set of constraints. These constraints could be in the form of lexical constraints (Hokamp and Liu, 2017; Anderson et al., 2017; Post and Vilar, 2018) or regular expressions (REGEX) or grammatical structures defined by context-free grammars (CFGs) (Geng et al., 2023; Beurer-Kellner et al., 2024) or just a list of possible outputs (De Cao1 et al., 2021). Several approaches have been proposed to impose these constraints during the sequence generation process. For example, De Cao1 et al. (2021) uses a trie structure, and Miao et al. (2019)

[4] https://www.llama.com/

utilizes Metropolis-Hastings sampling. While these approaches address the need to generate valid outputs that satisfy all the specified constraints, they often struggle with computational efficiency when the constraint set is large (Stahlberg and Kumar, 2019). Therefore, generating valid outputs on-the-fly without exhaustive enumeration remains a challenge.

Generative retrieval. Recently, new approaches for retrieval have emerged whereby LLMs directly generate relevant documents for a given query without explicitly computing embeddings or doing nearest neighbor search (Rajput, 2024). There are primarily two ways of doing this: (1) the entire set of documents is put in the context of LLM, and then LLM generates relevant documents from this context (Lee et al., 2024), and (2) documents are not put in the context but to make sure that the generated documents are from the retrieval set, the decoding process is constrained (De Cao1 et al., 2021). In applications where the retrieval set is enormous, the first approach would require the LLM to have $\mathcal{O}(N)$ context length where N is the size of documents. As of today, this is too computationally expensive. In this paper, we focus on the second set of approaches.

6 DISCUSSION & CONCLUSION

This paper focuses on comparing trie-based methods and does not consider combining `DISC` with countless existing techniques. It should be pointed out that `DISC` can be seamlessly integrated into existing language model frameworks without requiring any modifications to the underlying architecture. This minimally invasive property means it is compatible with most generation strategies such as temperature scaling or top-k sampling, as well as task-specific techniques such as special tokenizations or retrieval tricks.

In addition, notice that `DISC` and `PPV` can be applied to other types of constraints (beyond set constraints $\mathcal{S}$ we consider). For instance, constraints defined by context-free grammars, specific formatting requirements like generating valid JSON structures, or other complex criteria can also be incorporated, and the theoretical analysis extends. Future work could explore empirical evaluations of `DISC` and `PPV` across a broader spectrum of constrained decoding scenarios.

In conclusion, we address the challenges of bias and inefficiency in constrained decoding by introducing the `DISC` and the `PPV`. We believe that our methods effectively mitigate unintended biases and enhance computational efficiency, broadening the applicability of constrained decoding in real-world scenarios.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Adelani, D. I., Abbott, J., Neubig, G., D'souza, D., Kreutzer, J., Lignos, C., Palen-Michel, C., Buzaaba, H., Rijhwani, S., Ruder, S., et al. (2021). Masakhaner: Named entity recognition for african languages. *Transactions of the Association for Computational Linguistics*, 9:1116–1131.

Anderson, P., Fernando, B., Johnson, M., and Gould, S. (2017). Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Bayer, R. and Unterauer, K. (1977). Prefix b-trees. *ACM Transactions on Database Systems (TODS)*, 2(1):11–26.

Beurer-Kellner, L., Fischer, M., and Vechev, M. (2024). Guiding LLMs the right way: Fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*.

Bevilacqua, M., Ottaviano, G., Lewis, P., Yih, S., Riedel, S., and Petroni, F. (2022). Autoregressive search engines: Generating substrings as document identifiers. *Advances in Neural Information Processing Systems*, 35:31668–31683.

Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., et al. (2022). Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

De Cao1, N., Izacard, G., Riedel, S., and Petroni, F. (2021). Autoregressive entity retrieval. In *Proceedings of the Ninth International Conference on Learning Representations*.

Diggelmann, T., Boyd-Graber, J., Bulian, J., Ciaramita, M., and Leippold, M. (2020). Climate-fever: A dataset for verification of real-world climate claims. *arXiv preprint arXiv:2012.00614*.

Dinan, E., Roller, S., Shuster, K., Fan, A., Auli, M., and Weston, J. (2018). Wizard of wikipedia:

Knowledge-powered conversational agents. *arXiv preprint arXiv:1811.01241*.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Elsahar, H., Vougiouklis, P., Remaci, A., Gravier, C., Hare, J., Laforest, F., and Simperl, E. (2018). T-rex: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Fan, A., Jernite, Y., Perez, E., Grangier, D., Weston, J., and Auli, M. (2019). Eli5: Long form question answering. *arXiv preprint arXiv:1907.09190*.

Fischer-Hbner, S. and Berthold, S. (2017). Privacy-enhancing technologies. In *Computer and information security Handbook*, pages 759–778. Elsevier.

Geng, S., Josifoski, M., Peyrard, M., and West, R. (2023). Grammar-constrained decoding for structured nlp tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Gilbert, S., Harvey, H., Melvin, T., Vollebregt, E., and Wicks, P. (2023). Large language model ai chatbots require approval as medical devices. *Nature Medicine*, pages 1–3.

Guo, Z. and Barbosa, D. (2018). Robust named entity disambiguation with random walks. *Semantic Web*, 9(4):459–479.

Hasibi, F., Nikolaev, F., Xiong, C., Balog, K., Bratsberg, S. E., Kotov, A., and Callan, J. (2017). Dbpedia-entity v2: a test collection for entity search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1265–1268.

Hoffart, J., Yosef, M. A., Bordino, I., Fürstenau, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., and Weikum, G. (2011). Robust disambiguation of named entities in text. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 782–792.

Hokamp, C. and Liu, Q. (2017). Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Hwang, G.-J. and Chang, C.-Y. (2023). A review of opportunities and challenges of chatbots in education. *Interactive Learning Environments*, 31(7):4099–4112.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12.

Le, P. and Titov, I. (2018). Improving entity linking by modeling latent relations between mentions. *arXiv preprint arXiv:1804.10637*.

Lee, J., Chen, A., Dai, Z., Dua, D., Sachan, D. S., Boratko, M., Luan, Y., Arnold, S. M., Perot, V., Dalmia, S., et al. (2024). Can long-context language models subsume retrieval, rag, sql, and more? *arXiv preprint arXiv:2406.13121*.

Lee, P., Bubeck, S., and Petro, J. (2023). Benefits, limits, and risks of gpt-4 as an ai chatbot for medicine. *New England Journal of Medicine*, 388(13):1233–1239.

Levy, O., Seo, M., Choi, E., and Zettlemoyer, L. (2017). Zero-shot relation extraction via reading comprehension. *arXiv preprint arXiv:1706.04115*.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. (2023). Visual instruction tuning. *arXiv preprint arXiv:2304.08485*.

Miao, Y., Zhou, H., and Mou, L. (2019). Cgmh: Constrained sentence generation by metropolis-hastings sampling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C. (2008). Gpu computing. *Proceedings of the IEEE*, 96(5):879–899.

Petroni, F., Piktus, A., Fan, A., Lewis, P., Yazdani, M., De Cao, N., Thorne, J., Jernite, Y., Karpukhin, V., Maillard, J., et al. (2020). Kilt: a benchmark for knowledge intensive language tasks. *arXiv preprint arXiv:2009.02252*.

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., and Vrgoč, D. (2016). Foundations of json schema. In *Proceedings of the 25th international conference on World Wide Web*, pages 263–273.

Post, M. and Vilar, D. (2018). Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*.

Rajput, Shashank, e. a. (2024). Recommender systems with generative retrieval. In *Proceedings of the Thirty-Seventh Annual Conference on Advances in Neural Information Processing Systems*.

Rajput, S., Mehta, N., Singh, A., Hulikal Keshavan, R., Vu, T., Heldt, L., Hong, L., Tay, Y., Tran, V., Samost, J., et al. (2024). Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems*, 36.

Skjuve, M., Følstad, A., Fostervold, K. I., and Brandtzaeg, P. B. (2021). My chatbot companion-a study of human-chatbot relationships. *International Journal of Human-Computer Studies*, 149:102601.

Stahlberg, F. and Kumar, S. (2019). Neural grammatical error correction with finite state transducers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Sun, W., Yan, L., Chen, Z., Wang, S., Zhu, H., Ren, P., Chen, Z., Yin, D., Rijke, M., and Ren, Z. (2024). Learning to tokenize for generative retrieval. *Advances in Neural Information Processing Systems*, 36.

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., and Gurevych, I. (2021). Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*.

Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. (2018). Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. (2024). Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Van Gysel, C. and de Rijke, M. (2018). Pytrec_eval: An extremely fast python interface to trec_eval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 873–876.

Wu, L., Petroni, F., Josifoski, M., Riedel, S., and Zettlemoyer, L. (2019). Scalable zero-shot entity linking with dense entity retrieval. *arXiv preprint arXiv:1911.03814*.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Ye, Q., Xu, H., Xu, G., Ye, J., Yan, M., Zhou, Y., Wang, J., Hu, A., Shi, P., Shi, Y., Li, C., Xu, Y., Chen, H., Tian, J., Qi, Q., Zhang, J., and Huang, F. (2023). mplug-owl: Modularization empowers large language models with multimodality. *arXiv preprint arXiv:2304.14178*.

Zhang, Z., Lei, L., Wu, L., Sun, R., Huang, Y., Long, C., Liu, X., Lei, X., Tang, J., and Huang, M. (2023). Safetybench: Evaluating the safety of large language models with multiple choice questions. *arXiv preprint arXiv:2309.07045*.

Zhu, D., Chen, J., Shen, X., Li, X., and Elhoseiny, M. (2023). Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Not Applicable]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Efficient and Asymptotically Unbiased Constrained Decoding for Large Language Models (Supplementary Materials)

7 Proofs of Theorems

7.1 Proof of Theorem 3.1

The logic of the proof can be decomposed into two parts. First, we compute the probability that upon K rounds of samples, at least one sample is accepted. We demonstrate that the distribution will be unbiased in this case. Second, when K new samples are re-drawn, we demonstrate that the importance sampling can upper bound the remaining KL divergence. Specifically, assume we use Q to denote the sequence distribution of Line 22, i.e., the distribution that K samples $\{\boldsymbol{a}_k\}_{k=1}^K$ are drawn using constrained decoding and further resample based on scores $\{x_k\}_{k=1}^K$. Then we upper bound $KL(P_{\mathcal{S}}\|Q)$ and use it to derive the eventual bound.

We first derive a few quantities useful for the proof. According to Algorithm 1, x is the importance score depending solely on sequence $\boldsymbol{a}$, and we therefore rewrite it as

$$x(\boldsymbol{a}) = \prod_{i\in\{1,\cdots,|\boldsymbol{a}|\}} P_L(\mathcal{A}_{\boldsymbol{a}_{<i},\mathcal{S}}|\boldsymbol{a}_{<i}).$$

Intuitively, $x(\boldsymbol{a})$ denotes the probability of alternatives in $\mathcal{S}$ along the token paths, which is smaller than the probability of all alternatives (which should be 1 at each decoding step). This is because at each decoding step, due to the application of constrained decoding, only tokens in $\mathcal{A}_{\boldsymbol{a}_{<i},\mathcal{S}}$ are considered, leading to a total candidate probability smaller than 1. Readers are encouraged to verify that $\sum_{\boldsymbol{a}\in\mathcal{S}} P_L(\boldsymbol{a})/x(\boldsymbol{a}) = 1$.

With this, the distribution for one sampled sequence using constrained (denote as $\hat{P}$) is

$$\hat{P}(\boldsymbol{a}) = \frac{P_L(\boldsymbol{a})/x(\boldsymbol{a})}{\sum_{\boldsymbol{a}\in\mathcal{S}} P_L(\boldsymbol{a})/x(\boldsymbol{a})} = \frac{P_L(\boldsymbol{a})}{x(\boldsymbol{a})}.$$

Probability of acceptance. Consider one particular draw $\boldsymbol{a}_k$ among the K rounds. The acceptance of $\boldsymbol{a}_k$ is $x(\boldsymbol{a}_k)$, and in expectation, each draw is accepted with probability

$$\sum_{\boldsymbol{a}\in\mathcal{S}} \hat{P}(\boldsymbol{a})x(\boldsymbol{a}) = P_L(\mathcal{S}) = 1 - p_b.$$

Therefore, at least one sample is accepted during K rounds of sample are $1 - p_b^K$.

Unbiased distribution for the accepted sample. To show that when a sample is accepted during K rounds, we only need to show that for each round, the distribution is unbiased. Conditioned on acceptance, the probability that $\boldsymbol{a} \in \mathcal{S}$ is returned is $\hat{P}(\boldsymbol{a})x(\boldsymbol{a})/(1-p_b) = P_L(\boldsymbol{a})/(1-p_b) = P_{\mathcal{S}}(\boldsymbol{a})$.

KL Divergence between $P_{\mathcal{S}}$ and Q. With probability p_b^K, K rounds of samples are all rejected, and `DISC` has to resample K samples and conduct importance sampling. We aim to analyze the KL divergence between the target distribution $P_{\mathcal{S}}$ and the distribution Q resulting from importance sampling.

Since the buffer size is K, the probability $Q(\boldsymbol{a})$ is K times the probability that sequence $\boldsymbol{a}$ is in the first position (which is $\hat{P}(\boldsymbol{a})$) and subsequently wins during the importance sampling step (due to symmetry among the K sequences). The challenge lies in computing the winning probability $W(\boldsymbol{a})$ of sequence $\boldsymbol{a}$.

Specifically, we have

$$W(\boldsymbol{a}) = \sum_{\boldsymbol{a}_2,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=2}^{K} \hat{P}(\boldsymbol{a}_j) \cdot \frac{x(\boldsymbol{a})}{\sum_{j=1}^{K} x(\boldsymbol{a}_j)},$$

where $x(\boldsymbol{a})$ is the importance score defined previously:

$$x(\boldsymbol{a}) = \prod_{i=1}^{|\boldsymbol{a}|} P_L(\mathcal{A}_{\boldsymbol{a}_{<i},\mathcal{S}} | \boldsymbol{a}_{<i}).$$

The KL divergence between $P_{\mathcal{S}}$ and Q can then be expressed as (for simplicity, we set $A = 1 - p_b$)

$$\begin{aligned}
\mathrm{KL}(P_{\mathcal{S}} \| Q) &= \sum_{\boldsymbol{a} \in \mathcal{S}} \frac{P_L(\boldsymbol{a})}{A} \ln \frac{P_L(\boldsymbol{a})/A}{Q(\boldsymbol{a})} \\
&= \sum_{\boldsymbol{a} \in \mathcal{S}} -\frac{P_L(\boldsymbol{a})}{A} \ln \frac{AK P_L(\boldsymbol{a}) W(\boldsymbol{a})}{P_L(\boldsymbol{a}) x(\boldsymbol{a})} \\
&= \sum_{\boldsymbol{a}_1 \in \mathcal{S}} -\frac{P_L(\boldsymbol{a}_1)}{A} \ln \left(\sum_{\boldsymbol{a}_2,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=2}^{K} \hat{P}(\boldsymbol{a}_j) \cdot \frac{AK}{\sum_{j=1}^{K} x(\boldsymbol{a}_j)} \right) \\
&= \sum_{\boldsymbol{a}_1 \in \mathcal{S}} -\frac{\hat{P}(\boldsymbol{a}_1) x(\boldsymbol{a}_1)}{A} \ln \left(\sum_{\boldsymbol{a}_2,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=2}^{K} \hat{P}(\boldsymbol{a}_j) \cdot \frac{AK}{\sum_{j=1}^{K} x(\boldsymbol{a}_j)} \right).
\end{aligned}$$

Applying Jensen's inequality (noting that $\ln \mathbb{E}[f] \geq \mathbb{E}[\ln f]$ for a convex function ln), and considering the negative sign, we obtain

$$\begin{aligned}
\mathrm{KL}(P_{\mathcal{S}} \| Q) &\leq \sum_{\boldsymbol{a}_1 \in \mathcal{S}} -\frac{\hat{P}(\boldsymbol{a}_1) x(\boldsymbol{a}_1)}{A} \sum_{\boldsymbol{a}_2,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=2}^{K} \hat{P}(\boldsymbol{a}_j) \ln \left(\frac{AK}{\sum_{j=1}^{K} x(\boldsymbol{a}_j)} \right) \\
&= \sum_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=1}^{K} \hat{P}(\boldsymbol{a}_j) \left(-\frac{x(\boldsymbol{a}_1)}{A} \right) \ln \left(\frac{AK}{\sum_{j=1}^{K} x(\boldsymbol{a}_j)} \right) \\
&= \sum_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=1}^{K} \hat{P}(\boldsymbol{a}_j) \cdot \frac{x(\boldsymbol{a}_1)}{A} \ln \left(\frac{\sum_{j=1}^{K} x(\boldsymbol{a}_j)}{AK} \right).
\end{aligned}$$

Due to the symmetry among the indices $\{1, \cdots, K\}$ in the summation (each $\boldsymbol{a}_j$ is drawn independently and identically), we can replace $x(\boldsymbol{a}_1)$ with any $x(\boldsymbol{a}_j)$ without affecting the overall value. Therefore, we can generalize the expression by summing over all positions j and averaging:

$$\begin{aligned}
\mathrm{KL}(P_{\mathcal{S}} \| Q) &\leq \frac{1}{K} \sum_{r=1}^{K} \sum_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=1}^{K} \hat{P}(\boldsymbol{a}_j) \cdot \frac{x(\boldsymbol{a}_r)}{A} \ln \left(\frac{\sum_{j=1}^{K} x(\boldsymbol{a}_j)}{AK} \right) \\
&= \sum_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_K \in \mathcal{S}} \prod_{j=1}^{K} \hat{P}(\boldsymbol{a}_j) \cdot \frac{\sum_{j=1}^{K} x(\boldsymbol{a}_j)}{AK} \ln \left(\frac{\sum_{j=1}^{K} x(\boldsymbol{a}_j)}{AK} \right).
\end{aligned}$$

Imagine $\tilde{X} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_K)$ is a random variable equaling to $\frac{\sum_{j=1}^K x(\boldsymbol{a}_j)}{AK}$ with probability $\tilde{P}(\tilde{X}) = \prod_{j=1}^K \hat{P}(\boldsymbol{a}_j)$. Then, we have

$$KL(P_{\mathcal{S}} \| Q) \leq \sum \tilde{P}(\tilde{X}) f(\tilde{X}) = \mathbb{E}_{\tilde{X}} f(\tilde{X}),$$

where $f(x) \triangleq x \ln x \leq |x-1| + \frac{(x-1)^2}{2}$. This implies that

$$KL(P_{\mathcal{S}} \| Q) = \mathbb{E}_{\tilde{X}}[|\tilde{X} - 1|] + \mathbb{E}_{\tilde{X}}[\frac{(\tilde{X}-1)^2}{2}] \tag{3}$$

$$\leq \sqrt{\mathbb{E}_{\tilde{X}}[(\tilde{X}-1)^2]} + \frac{1}{2}\mathbb{E}_{\tilde{X}}[(\tilde{X}-1)^2], \tag{4}$$

where the inequality is by Jensen's inequality. The problem is therefore converted to upper bounding the variance of $\tilde{X}$ (whose expectation is 1 because the expectation of $x(\boldsymbol{a})$ is A). Due to the independence of each $\boldsymbol{a}_j$, we have

$$\mathrm{Var}(\tilde{X}) = \frac{1}{K}\mathrm{Var}(\frac{\boldsymbol{a}}{A}) = \frac{1}{K}\mathrm{Var}(\frac{\boldsymbol{a}}{A}) \leq \frac{A(1-A)}{KA^2},$$

where the inequality is because all r.v. with a value ranging from $[0, 1]$ and expectation ξ has a variance upper bound by $\xi(1-\xi)$. Plugging this into the Equation (4), we have

$$KL(P_{\mathcal{S}} \| Q) \leq \sqrt{\frac{p_b}{K(1-p_b)}} + \frac{p_b}{2K(1-p_b)}.$$

KL Divergence between $P_{\mathcal{S}}$ and P_{IS}^K. The final step is to bound the divergence of `DISC` using all the above results. According to the acceptance probability and the unbiasedness of accepted samples, we have

$$P_{IS}^K = p_b^K Q + (1 - p_b^K) P_{\mathcal{S}}.$$

The final bound is obtained by using Theorem 7.1.

Lemma 7.1. *For any two distribution P, Q, any scalar $t \in [0, 1]$, we have*

$$KL(P \| tP + (1-t)Q) \leq (1-t) KL(P \| Q).$$

7.2 Proof of Theorem 2.1

To rigorously prove the theorem, we only need to construct an LLM output distribution, determine the candidate set $\mathcal{S}$, and derive the divergence. In fact, assume there are only two tokens, i.e., $\mathcal{V} = \{v_1, v_2\}$, and all sequences are with length 2 (so four possible sequences in total, with an ending `EOK` token after each sequence being draw with probability 1. We assume that

$$P_L(v_1v_1) = P_L(v_1v_2) = \frac{1-p_b}{2}, P_L(v_2v_1) = p_b\epsilon, P_L(v_2v_2) = p_b(1-\epsilon).$$

Assume $\mathcal{S} = \{v_1v_1, v_1v_2, v_2v_1\}$. In this case, the ground-truth distribution $P_{\mathcal{S}}$ is

$$P_L(v_1v_1) = P_L(v_1v_2) = \frac{1}{2} \cdot \frac{1-p_b}{1-p_b+p_b\epsilon}, P_L(v_2v_1) = \frac{p_b\epsilon}{1-p_b+p_b\epsilon}.$$

On the contrary, if we apply constrained decoding, then for the first token to be decoded, both v_1, v_2 are valid. Therefore, the probability of choosing v_1, v_2 is $1-p_b$ and p_b, separately. When v_1 is chosen, as v_1v_1 and v_1v_2 belong to $\mathcal{S}$, the probability for each of them conditioned on v_1 being chosen is $\frac{1}{2}$. However, when v_2 is chosen, since v_2v_2 does not satisfy the constraint, the model can only choose v_2v_1. This gives the distribution of $P_{\mathcal{S}}^{CD}$ as

$$P_{\mathcal{S}}^{CD}(v_1v_1) = P_{\mathcal{S}}^{CD}(v_1v_2) = \frac{1-p_b}{2}, P_{\mathcal{S}}^{CD}(v_2v_1) = p_b.$$

Consequently, the KL divergence between the two distributions is (denote $\frac{1-p_b}{1-p_b+p_b\epsilon}$ as κ

$$\begin{aligned} KL(P_{\mathcal{S}} \| P_{\mathcal{S}}^{CD}) &= 2 \times \frac{1}{2} \kappa \ln \frac{\kappa/2}{(1-p_b)/2} + \frac{p_b\epsilon}{1-p_b+p_b\epsilon} \ln(\frac{p_b\epsilon}{p_b(1-p_b+p_b\epsilon)}) \\ &= \kappa \ln \frac{\kappa}{1-p_b} + \frac{p_b\epsilon}{1-p_b+p_b\epsilon} \ln(\frac{\epsilon}{(1-p_b+p_b\epsilon)}). \end{aligned}$$

The final result is obtained by considering $\epsilon \to 0$.

8 Torch code for PPV

```python
def ppv(P, a, X, M, vocab_size):
    """
    Args:
        P (torch.Tensor): Token distribution tensor of shape (vocab_size,).
        a (torch.Tensor): Partial sequence tensor of shape (len_a,).
        X (torch.Tensor): Constraint matrix of shape (N, l_max), sorted
            lexicographically.
        M (int): Number of top tokens to consider.
        vocab_size (int): Size of the vocabulary.
    Returns:
        torch.Tensor: Mask of valid tokens of shape (vocab_size,).
    """
    # Get top M token indices
    _, vt = torch.topk(P, M)  # vt: (M,)

    # Build candidate sequences V by appending vt to a
    V = torch.cat([a.unsqueeze(0).repeat(M, 1), vt.unsqueeze(1)], dim=1).to(X.device)
        # V: (M, len_a + 1)

    # Initialize binary search bounds
    N = X.shape[0]
    low = torch.zeros(M, dtype=torch.long, device=X.device)
    high = torch.full_like(low, N)

    # Perform binary search
    while (low < high).any():
        mid = torch.div(low + high, 2, rounding_mode="floor")
        mid_seqs = X[mid][:, :V.shape[1]]  # Shape: (M, len_a + 1)

        # Lexicographical comparison
        cmp = (V > mid_seqs).float() - (V < mid_seqs).float()
        cmp_result = cmp.cumsum(dim=1)[:, -1]  # Final comparison result per sequence

        # Update bounds based on comparison results
        greater = cmp_result > 0
        low = torch.where(greater, mid + 1, low)
        high = torch.where(~greater, mid, high)

    # Check for matches after binary search
    valid = (low < N)
    V_valid = V[valid]
    matching_seqs = X[low[valid]][:, :V.shape[1]]

    # Verify exact matches
    matches = (V_valid == matching_seqs).all(dim=1)
    valid_indices = vt[valid][matches]

    # Create mask of valid tokens
    mask = torch.zeros(vocab_size, dtype=torch.bool, device=X.device)
    mask[valid_indices] = True
    return mask
```

Listing 1: Parallel Prefix-Verification (PPV) algorithm

9 Additional Experimental Settings

Entity Disambiguation. We use two BART checkpoints provided by De Cao1 et al. (2021) to retrieve entities from ∼6M Wikipedia pages. Two of the BART checkpoints are used to evaluate on Entity Disambiguation: The first BART checkpoint, denoted as `BART-BLINK`, is pre-trained on the BLINK data (Wu et al., 2019), i.e., 9M unique document-entity pairs from Wikipedia. The other BART checkpoint, `BART-AIDA` is further fine-tuned on AIDA-CoNLL dataset (Hoffart et al., 2011). We test Entity Disambiguation with 6 datasets: AIDA-CoNLL, MSNBC, AQUAINT, ACE2004, WNED-CWEB (CWEB) and WNED-WIKI (WIKI) (Guo and Barbosa, 2018).

Document Retrieval. We use the BART checkpoint for Document Retrieval provided by De Cao1 et al. (2021), which is trained on all the BLINK and KILT data (Petroni et al., 2020). Document Retrieval is tested with KILT benchmark (Petroni et al., 2020), consisting fact checking with FEVER (Thorne et al., 2018); open-domain question answering using Natural Questions (Adelani et al., 2021), HotpotQA (Yang et al., 2018), TriviaQA (Joshi et al., 2017), ELI5 (Fan et al., 2019); slot filling with T-REx (Elsahar et al., 2018), Zero Shot RE (Levy et al., 2017); entity disambiguation on AIDA CoNLL-YAGO, WNED-WIKI and WNED-CEWB; dialog with Wizard of Wikipedia (Dinan et al., 2018).

Entity Retrieval & Fact checking Without fine-tuning, the language model is provided with a few demonstrations from the target corpus and leverages its in-context learning ability to generate entities accordingly in these two tasks. For each task, three query-doc pairs from the validation set are provided as in-context examples to demonstrate the task intentions and output formats. We use the datasets from zero-shot information retrieval benchmark BEIR (Thakur et al., 2021), i.e., DBPedia-Entity (Hasibi et al., 2017), Climate-FEVER (Diggelmann et al., 2020) for Entity Retrieval and Fact Checking respectively. The prompts are given as follows:

- ```
  Retrieve entity from DBpedia for my query.
  Query: <what is foreign policy for angola>    Entity: <Foreign relations of Angola>
  Query: <animalia>    Entity: <Animalia (book)>
  Query: <what is asphalt and what is it used for>    Entity: <Asphalt>
  Query: <{$newQuery}>    Entity:
  ```
- ```
  Retrieve entity from Wikipedia for my query.
  Query: <Global warming is driving polar bears toward extinction>
  Entity: <Habitat destruction>
  Query: <The sun has gone into ‘lockdown’ which could cause freezing weather, earthquakes
  and famine, say scientists>
  Entity: <Famine>
  Query: <the Great Barrier Reef is in fine fettle>
  Entity: <Great Barrier Reef>
  Query: <{$newQuery}>
  Entity:
  ```

10 Additional Results

In this section, we present additional experimental results to show that `DISC` doesn't require huge M (the number of top candidate tokens) and is compatible with other advanced decoding strategies such as beam search. In table 5, we evaluate `DISC` on Entity Disambiguation task with M ranges from $\{50, 100, 500, 1000\}$. From the results in Table 5, we observe that reducing the value of M from 1000 to 50 has a minimal impact on the Micro F1 scores across all datasets. For instance, with $K = 1$, decreasing M from 1000 to 50 only results in a slight fluctuation in F1 scores—often within a margin of 0.01. This indicates that even a small set of top candidate tokens ($M = 50$) is sufficient for `DISC` to achieve near-optimal performance. Moreover, a smaller M significantly reduces the evaluation time. For example, with $K = 1$ on the CWEB dataset, the evaluation time drops from 142.65 seconds at $M = 1000$ to 126.86 seconds at $M = 50$. This demonstrates that choosing a smaller M not only maintains performance but also enhances computational efficiency. Increasing the allowed sampling steps K generally improves the F1 scores across datasets, as seen when comparing results from $K = 1$ to $K = 4$. However, this comes at the cost of increased evaluation time. For example, with $M = 1000$, the

Table 5: The Micro F1 score and evaluation time for DISC with different K (allowed sampling steps) and M (top candidate tokens). The task is Entity Disambiguation across six datasets with BART AY2 checkpoint. **Avg.** is the average score across datasets.

	ACE2004		**AIDA**		**AQUAINT**		**CWEB**		**MSNBC**		**WIKI**		**Avg.**	
	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time
$K = 1$														
$M = 1000$	0.8405	3.377	0.8379	55.436	0.7799	9.089	0.6484	142.65	0.7927	8.946	0.8009	89.52	0.7834	51.50
$M = 500$	0.8483	2.961	0.8395	52.747	0.7951	8.636	0.6504	135.50	0.7668	8.350	0.8003	86.47	0.7834	49.11
$M = 100$	0.8483	2.730	0.8459	49.429	0.7937	8.104	0.6518	127.27	0.7774	8.553	0.7948	81.23	0.7853	46.22
$M = 50$	0.8327	2.736	0.8421	48.054	0.7992	7.912	0.6517	126.86	0.7835	8.121	0.7965	80.15	0.7843	45.64
$K = 2$														
$M = 1000$	0.8521	4.751	0.8535	86.152	0.8157	13.944	0.6616	226.15	0.7973	13.358	0.8038	147.77	0.7973	82.02
$M = 500$	0.8483	4.558	0.8531	83.668	0.8088	14.043	0.6641	212.14	0.7851	12.798	0.8047	139.36	0.7940	77.76
$M = 100$	0.8405	4.641	0.8546	80.060	0.8198	12.915	0.6634	204.21	0.7896	12.761	0.8046	133.91	0.7954	74.75
$M = 50$	0.8560	4.224	0.8493	74.577	0.8061	12.575	0.6618	195.74	0.8003	11.783	0.8043	126.26	0.7963	70.86
$K = 3$														
$M = 1000$	0.8249	7.056	0.8544	123.357	0.8143	22.207	0.6652	337.48	0.8049	18.632	0.8102	217.88	0.7956	121.10
$M = 500$	0.8327	6.166	0.8562	109.944	0.8226	18.060	0.6623	281.26	0.7988	16.643	0.8102	187.58	0.7971	103.27
$M = 100$	0.8405	5.447	0.8493	98.833	0.8171	16.148	0.6642	256.41	0.7973	15.214	0.8097	168.73	0.7963	93.46
$M = 50$	0.8405	5.878	0.8542	102.032	0.8157	17.204	0.6646	263.40	0.8018	15.205	0.8059	171.65	0.7971	95.89
$K = 4$														
$M = 1000$	0.8405	7.806	0.8549	137.484	0.8322	22.944	0.6668	357.17	0.8018	20.691	0.8081	241.67	0.8007	131.29
$M = 500$	0.8366	7.246	0.8549	134.291	0.8184	21.457	0.6635	344.06	0.7896	19.285	0.8094	223.37	0.7954	124.95
$M = 100$	0.8444	7.065	0.8555	130.161	0.8171	21.470	0.6655	331.44	0.8018	19.462	0.8080	221.01	0.7987	121.77
$M = 50$	0.8327	6.680	0.8598	121.303	0.8088	20.674	0.6667	318.97	0.8079	18.179	0.8069	211.05	0.7971	116.14

Table 6: The Micro F1 score for DISC with different beam search sizes with different K. The task is Entity Disambiguation across six datasets with BART AY2 checkpoint. **Avg.** is the average score across datasets. We show that DISC can be combined with beam search decoding strategy to improve performance.

Dataset	**ACE2004**		**AIDA**		**AQUAINT**		**CWEB**		**MSNBC**		**WIKI**		**Avg.**	
Beam size	1	2	1	2	1	2	1	2	1	2	1	2	1	2
DISC ($K = 1$)	0.841	0.852	0.797	0.869	0.814	0.824	0.635	0.673	0.765	0.803	0.803	0.818	0.776	0.807
DISC ($K = 2$)	0.848	0.856	0.803	0.871	0.821	0.827	0.647	0.676	0.781	0.806	0.811	0.819	0.785	0.809
DISC ($K = 3$)	0.848	0.860	0.808	0.872	0.831	0.825	0.651	0.677	0.782	0.797	0.810	0.820	0.788	0.809
DISC ($K = 4$)	0.849	0.860	0.810	0.873	0.827	0.825	0.651	0.678	0.781	0.805	0.816	0.820	0.789	0.810

average time increases from 51.50 seconds at $K = 1$ to 131.29 seconds at $K = 4$. Therefore, there's a trade-off between performance gains and computational overhead when adjusting K.

In Table 6, we evaluate the compatibility of DISC with beam search decoding strategies. Incorporating beam search with a beam size of 2 consistently improves the Micro F1 scores across all datasets and values of K. For instance, at $K = 1$, the average F1 score increases from 0.776 with greedy decoding to 0.807 with beam search. This enhancement suggests that beam search helps explore more candidate sequences, leading to better disambiguation outcomes. Combining beam search with higher values of K continues to yield marginal improvements. However, the gains taper off as K increases, indicating that most of the performance benefits are captured with smaller K and beam sizes. This synergy between DISC and beam search showcases the method's flexibility and its ability to integrate with advanced decoding strategies to further boost performance.

In summary, these additional experiments demonstrate that DISC is both efficient and adaptable. It does not require a large number of top candidate tokens to perform effectively, and it can seamlessly incorporate beam search to enhance its disambiguation capabilities. This makes DISC a practical choice for entity disambiguation tasks where computational resources and time are critical considerations.