SpecHub: Provable Acceleration to Multi-Draft Speculative Decoding

Anonymous ACL submission

Abstract

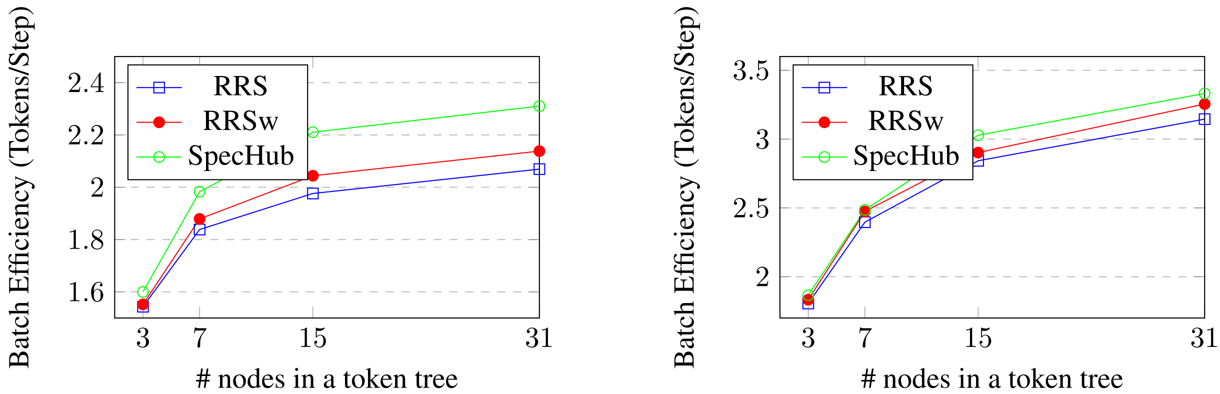
As large language models (LLMs) become integral to advancing NLP tasks, their sequential decoding becomes a bottleneck to achieving more efficient inference. Multi-Draft Speculative Decoding (MDSD) emerges as a promising solution, where a small draft model produces a tree of tokens with each path as a draft predicting the target LLM's outputs, which is then verified by the target LLM in parallel. However, current methods rely on Recursive Rejection Sampling (RRS) and its variants, which suffer from low acceptance rates in proceeding drafts, diminishing the merits of multiple drafts. In this work, we investigate this critical inefficiency and sub-optimality through an optimal transport (OT) formulation that aims to maximize the acceptance rate by optimizing the joint distribution $\pi(x_{1:k}, y)$ of k-draft tokens $x_{1:k}$ and an accepted token y. We show that the OT can be greatly simplified to a much smaller Linear Programming (LP) focusing on a few probabilities in $\pi(x_{1:k}, y)$. Moreover, our analysis of different choices for the marginal distribution $Q(x_{1:k})$ reveals its importance to the sampling effectiveness and efficiency. Motivated by the new insight, we introduce SpecHub, which adopts a special design of $Q(x_{1:k})$ that significantly accelerates the LP and provably achieves a higher acceptance rate than existing strategies. SpecHub can be seamlessly integrated into existing MDSD frameworks, improving their acceptance rate while only incurring linear computational overhead. In extensive experiments, Spechub consistently generates 0.05-0.27 and 0.02-0.16 more tokens per step than RRS with and without replacement and achieves equivalent batch efficiency with half as much concurrency. We attach our code at anonymous.4open.science/r/SpecHub.

1 Introduction

With the growing adoption of Large Language Models (LLMs) in diverse applications, there is a significant demand for faster inference and lower latency in both local computing and online API services. However, the sequential generation process of autoregressive language models complicates parallel computation. This challenge is exacerbated by the memory limitations of current hardware architectures, where RAM and cache communication latencies often constrain performance, resulting in underutilized computing capacity.

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023a) accelerates LLM inference while preserving the model's output distribution. By generating a sequence of draft tokens in advance using a smaller model, it leverages GPUs to verify tokens simultaneously through rejection sampling. Recent advancements (Chen et al., 2024; Jeon et al., 2024; Sun et al., 2024; Miao et al., 2023) have further enhanced this approach by introducing tree-structured multi-drafts, where each path represents a draft. These tokens are verified in parallel during a single forward pass of the LLM. Using a token tree increases the number of accepted tokens by providing multiple options for each token position, thus increasing the overall acceptance rate of the algorithm and generation efficiency.

Despite having various tree constructions, draft model designs, and hardware optimizations, existing multi-draft methods depend on recursive rejection sampling (RRS) for acceptance, which is far from optimal. While RRS greedily accepts the token from the first draft, it does not consider the subsequent drafts and misses the opportunity to dynamically adjust the current token's acceptance strategy to improve the acceptance rates of the later drafts. Consequently, later iterations in RRS accept tokens according to a residual distribution modified by previous acceptances, which may no longer align with the draft distribution these tokens are drawn from, resulting in low acceptance rates (Chen et al., 2023b). Meanwhile, Sun et al. (2024) shows the design of an acceptance rule can be optimized by solving an Optimal Transport



(a) Llama2-7B with JF68m draft model on CNN dataset.

(b) Vicuna-7B with EAGLE draft model on MT-Bench.

Figure 1: Decoding efficiency of SpecHub, RRS, and RRSw with different # nodes in a token tree on a binary tree using temperature $T = 1.0$.

problem with Membership Cost (OTM). However, OTM requires tremendous computation overhead and is not practically feasible.

In this paper, we solve the dilemma of the computational efficiency and sampling optimality in Multi-Draft Speculative Decoding (MDSD). We first reduce the OTM formulation to a much smaller linear programming (LP) by focusing only on the transport plan of scenarios where at least one draft gets accepted. We then investigate the overlooked design choice of draft sampling. While all previous methods used either sampling with or without replacement, which makes finding the optimal solution notoriously hard, we show that an optimal acceptance rule can be trivially obtained if we instead choose only certain drafts of tokens. As a result, we can develop practical algorithms that balance acceptance rate with computation overhead.

Building on the new LP formulation and insights, we introduce SpecHub, a faster sampling and verification paradigm with only linear computational overhead. Instead of constructing a dense distribution of k-draft and the accepted token, SpecHub strategically selects drafts containing the highest probability token sampled from the draft model. The top draft token serves as a transport hub for an oversampled token [1] to transfer its excessive probability mass to an undersampled token. This sparse structure simplifies and accelerates the underlying linear programming. SpecHub performs particularly well on LLMs since their output distributions concentrate on the top token, leading to a higher acceptance rate than RRS. It even provably outperforms OTM under certain situations. The algorithm is widely applicable and can seamlessly integrate into various MDSD algorithms, enhancing their efficiency and overall decoding speed.

[1]Draft model probability exceeds that of the target model.

We empirically test SpecHub by implementing it to various MDSD frameworks (Li et al., 2024; Chen et al., 2024; Miao et al., 2023). We observe a $1-5\%$ increase in the second draft acceptance rate, which yields a consistent $0.02-0.16$ improvement in batch efficiency over current methods. More impressively, SpecHub uses a tree with only half the nodes of other methods to reach the same level of batch efficiency. In our ablation study, SpecHub brings consistent acceleration to LLM decoding under different temperatures. Our toy experiments further show that SpecHub sometimes outperforms OTM in high-entropy regions.

2 Background and Related Work

Here, we review the sampling and verification schema of speculative decoding. We discuss the theory behind rejection sampling and explain why naively extending it to Multi-Draft Speculative Decoding (MDSD) becomes inefficient.

Speculative Sampling Language model decoding is intrinsically serial. Let $\mathcal{V}$ denote the vocabulary, a discrete set of tokens that the language model may generate. Let $x^{1:t} = (x^1, \ldots, x^t) \in \mathcal{V}^{\otimes t}$ denote a sequence of tokens. Then, the target language model produces a conditional probability $p(\cdot|x^{1:t})$, from which we sample the next token $x^{t+1} \sim p(\cdot|x^{1:t})$. However, this process is slow for its serial execution.

Speculative decoding (Chen et al., 2023a; Leviathan et al., 2023) addresses the issue by parallelizing the decoding process with a draft and

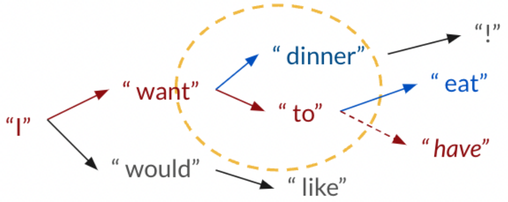


Figure 2: An example of a token tree of depth $d = 4$. The tree is generated sequentially with the draft model and evaluated concurrently with the target model. Each path in the tree corresponds to a potential sequence of tokens, with accepted tokens and rejected tokens highlighted. The black arrows indicate tokens that were not visited. The dashed line represents a sample drawn from the residual distribution after all drafts are rejected. Our paper focuses on the evaluation of one step, how we choose to sample the $k = 2$ tokens " dinner" and " to" from the draft distribution $q(\cdot|\text{"I want"})$ and decide which of them to get accepted based on the target probabilities $p(\text{" dinner"}|\text{"I want"})$ and $p(\text{" to"}|\text{"I want"})$.

verify phase. It first uses a smaller draft model $q(\cdot|x^{1:t})$ to generate a draft $(x^{t+1}, \dots, x^{t+d})$ sequentially. The depth of the draft, d, is usually around 5. This draft allows us to compute the target distributions $p(x^{t+\tau}|x^{1:t+\tau-1})$ in parallel for $\tau \leq d$. Then, we iteratively accept each draft token using rejection sampling with acceptance probability $\min\left(1, \frac{p(x^{t+\tau}|x^{1:t+\tau-1})}{q(x^{t+\tau}|x^{1:t+\tau-1})}\right)$. In this single draft setting, speculative decoding equates to sampling directly from the target distribution. After rejection, we sample from the residual distribution $\mathrm{norm}(\max(0, p(\cdot|x^{1:t+\tau-1}) - q(\cdot|x^{1:t+\tau-1})))$.

With only a single draft, the expected number of tokens generated at each iteration is upper-bounded. Assume the average acceptance rate for each token is α, the maximum acceleration is $1/(1-\alpha)$ (Chen et al., 2024). Multi-Draft Speculative Decoding solves this issue (Miao et al., 2023; Sun et al., 2024). Instead of verifying one sequence per time, MDSD generates a tree of tokens and calculates their target probability in parallel. Thus, when the first draft gets rejected, the other drafts can be picked up, and their offspring get verified in the current step. By doing so, we trade more parallel inference for more tokens generated in each step.

In the rest of the paper, we ignore any temporal relationship and only focus on a single temporal step in the decoding process. In particular, given $q(\cdot|x^{1:t-1})$ and $p(\cdot|x^{1:t-1})$, we discuss the sampling and verification algorithm for generating the offspring drafts and accepting one. We simplify the notation and use $p = p(\cdot|x^{1:t-1}) \in \Delta^{|\mathcal{V}|-1}$ to denote the target model's probability distribution and $q = q(\cdot|x^{1:t-1}) \in \Delta^{|\mathcal{V}|-1}$ to denote the draft model's distribution. Here, $\Delta^{|\mathcal{V}|-1} = \left\{ p \in \mathbb{R}^{|\mathcal{V}|} \,\middle|\, \sum_{x \in \mathcal{V}} p(x) = 1,\ p(x) \geq 0\ \forall x \in \mathcal{V} \right\}$ is the probability simplex of dimension $|\mathcal{V}|$. We also notate the probability simplex of joint distributions over a group of drafts $x_{1:k} = (x_1, \dots, x_k)$ as:

$$\Delta^{|\mathcal{V}|^k-1} = \{ P \in \mathbb{R}^{|\mathcal{V}|^k} \,\Big|\, \sum_{X \in \mathcal{V}^{\otimes k}} P(x_{1:k}) = 1,$$
$$P(x_{1:k}) \geq 0\ \forall x_{1:k} \in \mathcal{V}^{\otimes k} \}$$

Rejection Sampling in Speculative Decoding We here provide a geometric intuition behind rejection sampling. Given a target distribution p and a sample token from the draft distribution $x \sim q$, we seek to accept x as much as possible while ensuring the outputted token from the process follows p. We can visualize the process as sampling a point under the probability mass function (PMF) of p. The draft sample lies under the PMF of q. If the token x is undersampled ($q(x) < p(x)$), we always accept it. If it is oversampled ($q(x) > p(x)$), the data point may or may not fall under p, in which case we accept it with probability $p(x)/q(x)$, the height ratio between the two curves at this token. Such methods fully utilize the overlap between the two distributions and give the highest theoretical acceptance rate. See Figure 3.

The residual distribution $\mathrm{norm}(\max(0, p-q))$ captures the remaining probability mass that was not covered by q. Sampling from this residual distribution ensures that any rejections are accounted for by exploring the regions where p exceeds q. This approach aligns the accepted samples closely with p, effectively achieving maximal coupling and ensuring the samples represent the target distribution p.

Recursive Rejection Sampling To facilitate MDSD, previous methods use Recursive Rejection Sampling, which naively applies rejection sampling on the residual distributions. First, Recursive Rejection Sampling (RRS) samples k candidates independently from the draft distribution. Then, it accepts each candidate with rejection sampling. If the token is rejected, the target distribution is updated to the residual distribution $\mathrm{norm}(\max(p-q, 0))$. While the acceptance of the first candidate is high,

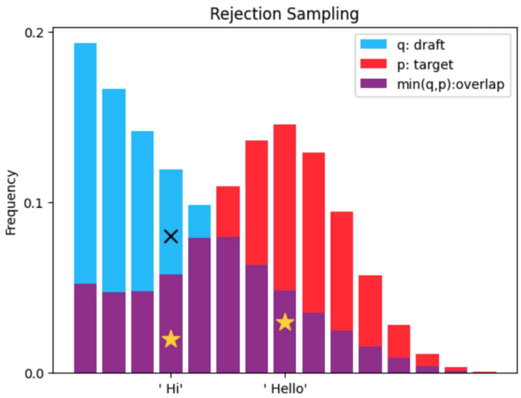


Figure 3: An illustration of rejection sampling. Sampling from the draft distribution gives a point under the blue distribution q. If the sample is also under the overlap with the target distributions p, we accept it. If not, we reject the token and sample from the residual distribution, the remaining unexplored area $\max(p - q, 0)$ normalized. The misalignment of the residual distribution and draft distribution makes Recursive Rejection Sampling (RRS) inefficient in proceeding runs.

subsequent candidates suffer from the potential mismatch between the residual distributions and draft distribution q. Essentially, our residual distribution deducts draft distribution, so we expect it to diverge from the draft distribution q we used to generate our samples, leading to small overlapping areas and inefficiencies.

Algorithm 1 Token-level RRS

Input: Target model distribution p, draft model distribution q, number of candidates k
Output: A token x selected using RRS without replacement.
Generate k samples $x_1, \ldots, x_k$ independently or without replacement from q
for $i = 1 \to k$ **do**
  sample $r_i \sim \mathrm{Uniform}(0, 1)$
  if $r_i < \frac{p(x_i)}{q(x_i)}$ **then**
    Return x_i
  else
    $p \leftarrow \mathrm{norm}(\max(p - q, 0))$
    if without replacement **then**
      $q(x_i) \leftarrow 0$
      $q \leftarrow \mathrm{norm}(q)$
    end if
  end if
end for
Return $x \sim p$

Recursive Rejection Sampling without Replacement In low-temperature settings, RRS may repeatedly sample the same token and fail to diversify the tree. Furthermore, a rejected token will continuously get rejected since the corresponding entry of the residual probability is 0. Following this intuition, several works(Chen et al., 2024; Jeon et al., 2024; Li et al., 2024; Yang et al., 2024) proposed Recursive Rejection Sampling without Replacement (RRSw). Instead of independently sampling, it samples tokens without replacement. It also modifies the draft distribution after each rejection to maintain a correct marginal distribution. The differences are highlighted in Algorithm 1 in red. While the method speeds up the decoding process by avoiding repetition, it still falls short of a theoretically optimal verification method as the misalignment between residual distribution and the draft distribution remains.

3 Mathematical Formulation of Multi-Draft Speculative Decoding

In this section, we lay out the mathematical formulation of the sampling and verification paradigm of MDSD. We start by reviewing the Optimal Transport with Membership Cost framework by Sun et al. (2024) in Section 3.1. We show that it can simplified and propose an equivalent LP formulation that greatly reduces computation complexity in Section 3.2. Lastly, we point out that changing the design of sampling can make the LP feasible for real-world calculation in Section 3.3 while preserving the acceleration. We also discuss some considerations for a real-world algorithm.

3.1 Optimal Transport with Membership Cost

We show how we finding the optimal sampling and verification algorithm of MDSD that maximizes the acceptance rate as solving an Optimal Transport problem with Membership Cost(Sun et al., 2024). Let the target distribution be p and the joint draft distribution $Q = q^{\otimes k} \in \Delta^{|\mathcal{V}|^k - 1}$ be the Cartesian product of the draft distributions that gives the probability of sampling any particular series of draft tokens $x_{1:k}$, so $Q(x_{1:k}) = \prod_{i=1}^{k} q(x_i)$. Let y denote the accepted token. We define the coupling between p and Q or equivalently a transport plan from Q to p be a joint distribution $\pi(x_{1:k}, y) \in \Delta^{|\mathcal{V}|^{k+1}-1}$ whose marginal distributions satisfies $\sum_{y \in \mathcal{V}} \pi(x_{1:k}, y) = Q(x_{1:k})$ and $\sum_{x_{1:k} \in \mathcal{V}^k} \pi(x_{1:k}, y) = p(y)$. We use the term coupling and transport plan interchangebly. The Membership Cost is $c(x_{1:k}, y) = \prod_{i=1}^{k} \mathbb{1}_{y \neq x_i}$, an indicator function of whether the accepted token y equals any of the draft tokens x_i. The transport cost then calculates the expected rejection rate:

$$C(\pi) = \mathbb{E}_{x_{1:k}, y \sim \pi}\left[\prod_{i=1}^{k} \mathbb{1}_{y \neq x_i}\right].$$

It is well-known that Optimal Transport on discrete space can be solved as a linear programming problem as

$$\min_{\pi \in \Pi(p,q)} \sum_{x_{1:k}} \sum_{y \in \mathcal{V}} \pi(x_{1:k}, y) \prod_{i=1}^{k} \mathbb{1}_{y \neq x_i} \quad (1)$$

where $\Pi(p, q)$ is the set of all valid couplings between p and $q^{\otimes k}$. However, such a program contains $O(|\mathcal{V}|^{k+1})$ variables, so even the fastest linear programming algorithm struggles to calculate in real-time.

3.2 A Simplified Linear Programming Formulation

While the Optimal Transport formulation provides a theoretical framework for understanding Multi-Draft Speculative Decoding, its computational complexity renders it impractical for real-time applications. To address this, we introduce a simplified Linear Programming (LP) formulation that significantly reduces the number of variables while preserving the essence of the problem.

The key insight behind this simplification is that the acceptance rate is primarily determined by how the sampled draft tokens are handled. Once a token is rejected, the subsequent actions, which involve recalculating the residual distribution and resampling, can be performed efficiently without explicitly considering the full coupling.

Instead of representing the entire coupling π, which has $O(|\mathcal{V}|^{k+1})$ variables, our simplified LP formulation focuses on $\pi(x_{1:k}, y = x_i)$, $i = 1, \ldots, k$, a smaller subset of transport plan which denotes the probability of sampling the series of drafts and accepting the i-th token x_i. This effectively reduces the number of variables to $O(|\mathcal{V}|^k)$, making the problem more tractable. The remaining probabilities in the coupling, which correspond to cases where the target token does not match any of the draft tokens, are implicitly handled by the residual distribution.

The simplified LP formulation is then:

$$\text{minimize}_{\pi}\ 1 - \sum_{x_{1:k} \in \mathcal{V}^k} \sum_{i=1}^{k} \pi(x_{1:k}, x_i)$$

subject to

$$\begin{aligned}
&\pi(x_{1:k}, x_i) \geq 0 && \forall x_{1:k} \in \mathcal{V}^k, i \\
&\sum_{i=1}^{k} \pi(x_{1:k}, x_i) \leq Q(x_{1:k}) && \forall x_{1:k} \in \mathcal{V}^k \\
&\sum_{i=1}^{k} \sum_{x_{1:k} \in \mathcal{V}^k, x_i = y} \pi(x_{1:k}, y) \leq p(y) && \forall y \in \mathcal{V}
\end{aligned}$$

Given a solution to this simplified LP formulation, we can reconstruct the complete transport plan $\pi(x_{1:k}, y)$. For any series of drafts $x_{1:k}$ and target token y, if y does not equal one of the draft tokens in $x_{1:k}$, the entry is calculated as:

$$\begin{aligned}
&\pi(x_{1:k}, y) \quad \text{\# where } y \neq x_i\ \forall i = 1, \ldots, k \\
&= \frac{p(y) - \sum_{i=1}^{k} \sum_{x_{1:k} \in \mathcal{V}^k, x_i = y} \pi(x_{1:k}, y)}{\sum_{y \in \mathcal{V}} p(y) - \sum_{i=1}^{k} \sum_{x_{1:k} \in \mathcal{V}^k, x_i = y} \pi(x_{1:k}, y)} \\
&\quad \cdot \left(Q(x_{1:k}) - \sum_{i=1}^{k} \pi(x_{1:k}, x_i)\right)
\end{aligned}$$

The first term is the unallocated target probability mass or the residual probability of y normalized . The second term is the remaining probability mass of the series of drafts $x_{1:k}$ after allocating probabilities to cases where the target token matches a draft token. This reconstruction process ensures that the validity of the coupling. This simplified LP formulation, while ignoring the explicit representation of the full coupling, retains the essential information needed to optimize the acceptance rate. It provides a practical and computationally feasible approach to solving the MDSD problem.

Theorem 1 (Equivalence of LP to OTM)**.** *For a given joint draft distribution Q and target distribution p, the optimal solution of the simplified LP formulation achieves the same transport cost as the maximal coupling in the Optimal Transport with Membership Cost (OTM) problem, i.e., $1 - \sum_{x_{1:k} \in \mathcal{V}^k} \sum_{i=1}^{k} \pi(x_{1:k}, x_i) = C(\pi^*)$, where π^* is the optimal coupling for the OTM problem as defined in Equation 1.*

Proof. See Appendix B. □

		Second Draft					
		a 0.5		b 0.3		c 0.2	
First Draft	a 0.5			0.3		0.2	
				a	b	a	c
				0	0.3	0.1	0.1
	b 0.3	0.214				0.086	
		b	a			b	c
		0.214	0			0.086	0
	c 0.2	0.125		0.076			
		c	a	c	b		
		0.125	0	0.075	0		
	accepted	0.1		0.6		0.3	
	cost	0		0		0	

(a) Optimal solution to LP

		Second Draft					
		a 0.5		b 0.3		c 0.2	
First Draft	a 0.5			0.3		0.2	
				a	b	a	c
				0.06	0.24	0.04	0.1
	b 0.3	0.214				0.086	
		b	a			b	c
		0.214	0			0.086	0
	c 0.2	0.125		0.076			
		c	a	c	b		
		0.125	0	0.075	0		
	accepted	0.1		0.54		0.3	
	cost	0		0.06		0	

(b) RRSw solution to LP

Figure 4: A comparison of an optimal solution to an RRSw solution under the LP formulation. Here, the draft distribution $q = [0.5, 0.3, 0.2]$ and the target distribution $p = [0.1, 0.6, 0.3]$. Each number on the top of the cell is $Q(x_1, x_2)$, and the numbers at the bottom of the cell show $\pi(x_1, x_2, x_1)$ and $\pi(x_1, x_2, x_2)$, i.e. how much of those draft probabilities are transferred to the target probability. RRSw has a transport cost of 0.06 for not generating enough token 'b'.

Examining Recursive Rejection Sampling (RRS) How does an optimal solution to the Linear Programming (LP) formulation differ from RRS? Consider the simple case of $k = 2$. When a series of drafts x_1, x_2 is sampled according to $Q(x_{1:2})$, we must decide whether to accept x_1 or x_2 based on the target distribution p. If x_1 is significantly oversampled, meaning $p(x_1) < q(x_1)$. RRS makes this decision independently for each draft token, while the OTM solution considers the entire series. Specifically, the OTM solution will tend to allocate less probability mass to accepting x_1 if x_2 is undersampled ($p(x_2) > q(x_2)$) and more probability mass if x_2 is also oversampled. This flexible adaptation ensures a more targeted distribution in subsequent drafts, leading to more efficient sampling and verification.

Unbalanced Tree and Asymmetric Verification When considering a single temporal step in the sampling and verification process, the order in which a pair of samples $x_{1:k}$ is selected appears inconsequential, as the branches are executed concurrently. However, as suggested by Sequoia (Chen et al., 2024), the most efficient tree structure is often unbalanced. If the acceptance rate of the early draft is higher than that of the second, designing a tree that extends deeper along the first few branches while keeping other branches shallower can enhance efficiency. Optimal algorithms may decrease the first few drafts' acceptance rate slightly to achieve a higher overall acceptance rate, which we need to carefully balance to leveraging the advantages of unbalanced tree structures and significantly improving decoding speed and performance.

3.3 Design of Sampling

While the simplified LP formulation significantly reduces the computational burden compared to the OTM, it remains computationally expensive for large vocabularies. Directly solving the LP problem is impractical, and previous research has predominantly focused on developing heuristics to approximate the optimal solution. These heuristics, such as Recursive Rejection Sampling (RRS) or SpecTr(Sun et al., 2024), operate under a fixed joint draft distribution, typically assuming independent sampling with ($Q = q^{\otimes k}$) or without replacement ($Q(x_{1:k}) = \frac{\prod_{i=1}^{k} q(x_i)}{\prod_{i=1}^{k-1}(1-\sum_{j=1}^{i} q(x_j))}$).

However, a crucial and often overlooked aspect is the ability to **modify the joint draft distribution** Q, which unlocks a new dimension for optimization that has not been fully explored. The key to designing a practical and efficient sampling strategy is recognizing that Q does not need to be a dense distribution over all possible drafts. Instead, we can strategically construct a sparse Q that simplifies the LP formulation while capturing the essential features of the target distribution. This sparsity reduces the number of variables and constraints in the LP, making it significantly easier to solve or approximate.

Ideally, the design of Q should satisfy two key criteria: 1) **Sparsity**; Q should be sparse, concentrating on a small subset of highly probable draft series to reduce computational complexity; and 2) **Efficiency**; Q should effectively capture the essential features of target distribution p, ensuring that the sampled drafts are likely to contain the target token. By carefully designing Q, we can balance computational efficiency and acceptance rate, paving the way for practical and high-performance MDSD algorithms.

4 SpecHub

Building on the aforementioned insights, we introduce SpecHub, a faster sampling-and-verifying paradigm with only linear computational overhead. It effectively captures the transport features of OTM solutions to enhance the acceptance rate and can be applied to various multi-draft speculative sampling algorithms. Since using more than two drafts offers little gains in efficiency, SpecHub uses two drafts (i.e., $k = 2$) to reduce complexity. We thoroughly discuss expanding the algorithm to more drafts in Appendix D.

First, we identify the token with the highest draft probability, denoted as a, and sample it alongside other tokens. We only populate the first column and the first row in the joint draft distribution Q. In particular, we define the joint draft distribution $Q(x_1, x_2)$ as follows:

$$Q(x_1, x_2) = \begin{cases} q(x_1) & \text{if } x_2 = a, \\ \frac{q(a)q(x_2)}{1-q(a)} & \text{if } x_1 = a, \\ 0 & \text{otherwise.} \end{cases}$$

This specific design of Q makes the solution to the simplified LP formulation straightforward. $\forall x \in \mathcal{V}, x \neq a$, we have

$$\pi(x, a, x) = \min(p(x), q(x))$$
$$\pi(a, x, x) = \min(p(x) - \pi(x, a, x), Q(a, x))$$

After transporting draft probabilities to target probabilities of non-top tokens, the remaining draft accepts the top token a evenly out of $p(a)$ The remaining entries in π can be reconstructed as described in the previous section. This solution effectively allocates as much probability mass as possible to the non-hub draft tokens while ensuring that the hub token a is never undersampled. This strategy maximizes the utilization of the draft distribution and leads to a higher acceptance rate compared to traditional methods like RRS.

Analysis SpecHub offers several theoretical advantages. First, since all drafts contain the top token a, it is accepted with a probability of $p(a)$ and is never undersampled (see Corollary 1). Additionally, let α be the first draft acceptance rate of rejection sampling, defined as $\alpha = \sum_x \max(p(x), q(x))$. SpecHub achieves a higher acceptance rate than Recursive Rejection Sampling (RRS) if the top token a satisfies the condition $\frac{q(a)}{1-q(a)} > 1 - \alpha$. We even guarantee acceleration over OTM sampled with replacement $Q = q^{\otimes}2$ if $\frac{1}{1-q(a)} > 2$ or $q(a) > \frac{1}{2}$. Detailed proofs of these results are in Appendix C.3.

While SpecHub might theoretically decrease the first draft acceptance rate for the top token a in rare cases, our empirical results, detailed in Appendix C.4, show that this effect is negligible.

<table>
<tr><td></td><td></td><td colspan="6">Second Draft</td></tr>
<tr><td></td><td></td><td colspan="2">a 0.5</td><td colspan="2">b 0.3</td><td colspan="2">c 0.2</td></tr>
<tr><td rowspan="9">First Draft</td><td rowspan="3">a 0.5</td><td colspan="2" rowspan="3"></td><td colspan="2">0.3</td><td colspan="2">0.2</td></tr>
<tr><td>a</td><td>b</td><td>a</td><td>c</td></tr>
<tr><td>0</td><td>0.3</td><td>0.1</td><td>0.1</td></tr>
<tr><td rowspan="3">b 0.3</td><td colspan="2">0.3</td><td colspan="2" rowspan="3"></td><td colspan="2" rowspan="3"></td></tr>
<tr><td>b</td><td>a</td></tr>
<tr><td>0.3</td><td>0</td></tr>
<tr><td rowspan="3">c 0.2</td><td colspan="2">0.2</td><td colspan="2" rowspan="3"></td><td colspan="2" rowspan="3"></td></tr>
<tr><td>c</td><td>a</td></tr>
<tr><td>0.3</td><td>0</td></tr>
<tr><td></td><td>accepted</td><td colspan="2">0.1</td><td colspan="2">0.6</td><td colspan="2">0.3</td></tr>
<tr><td></td><td>cost</td><td colspan="2">0</td><td colspan="2">0</td><td colspan="2">0</td></tr>
</table>

Figure 5: SpecHub under the LP formulation. Here, the draft distribution $q = [0.5, 0.3, 0.2]$ and the target distribution $p = [0.1, 0.6, 0.3]$. SpecHub focuses on the top token "a", sampling pairs (x, a) and (a, x) with probabilities $q(x)$ and $\frac{q(a)q(x)}{1-q(a)}$, respectively. This method ensures efficient allocation of acceptance probabilities.

5 Experiments

In this section, we empirically show can improve batch efficiency in speculative multi-draft decoding. We first show that SpecHub gives a significantly higher acceptance rate for its better coupling properties in the second draft acceptance rate. We then illustrate how the improvement transfers to higher batch efficiency.

5.1 Experiment Setup

Our experimental setup is based on the Llama and Vicuna models. To mimic the setup of Chen et al. (2024), we utilize the JackFram/Llama-68m and JackFram/Llama-160m (JF68m, JF160m) (Miao et al., 2023) models as our draft models and the Llama2-7B (Touvron et al., 2023) models as our target models. We evaluate our results on the OpenWebText (Gokaslan and Cohen, 2019) and CNN DailyMail (See et al., 2017) datasets. For each run, we use 200 examples to measure the acceptance rate vector and sample another 200 examples for evaluation. The prompt length and generation length are both set to 128 tokens. We evaluate our

T	RRS	RRSw	**SpecHub**
0.3	0.0426	0.1114	**0.1184**
0.6	0.074	0.1089	**0.1379**
1.0	0.1021	0.114	**0.166**

Table 1: Acceptance Rate for JF68m Model

T	RRS	RRSw	**SpecHub**
0.3	0.0399	0.1129	**0.1221**
0.6	0.073	0.1212	**0.1351**
1.0	0.091	0.1176	**0.166**

Table 2: Acceptance Rate for the JF160m Model

Figure 6: The change in batch efficiency at different temperatures.

system on a single RTX A5000 GPU.

We also implement our algorithm on EAGLE (Li et al., 2024). In short, EAGLE trains an autoregressive decoding head that takes both the embedding in the last layer of the target model and the draft tokens to predict a draft. We test its performance on Vicuna-7b (Zheng et al., 2024), a fine-tuned LLaMA chatbot using ChatGPT (OpenAI et al., 2024) to generate responses. We use the MT-Bench dataset and temperatures $T = 0.6,\ 1.0$ with binary trees and binary Sequoia trees.

5.2 Main Experiments

Second Draft Acceptance Rate We evaluate SpecHub at different temperatures $T = 0.3, 0.6, 1.0$ using JF68m and JF160m as draft models. We observe that SpecHub consistently outperforms RRS and RRSw. In particular, at higher temperatures, SpecHub achieves up to 5% improvements in the second draft acceptance rate from $0.114 - 0.117$ to 0.166. At a lower temperature, the improvement over RRSw becomes smaller since the whole process assimilates greedy decoding. In fact, SpecHub is equivalent to RRS without replacement at zero temperature since both algorithms become top-2 greedy decoding. Results are shown in Table 1 and 2.

Batch Efficiency We examine how the increased second-draft acceptance rate translates to better batch efficiency in different tree configurations. We empirically test SpecHub and RRS without replacement on binary trees of depth d with $2^d - 1$ nodes and report the batch efficiency in 1. We see that with JF68M as the draft model, SpecHub consistently outperforms RRS and RSSw by $0.02 - 0.10$ and $0.04 - 0.20$ in batch efficiency at temperatures $T = 0.6,\ 1.0$. Meanwhile, using the EAGLE decoding head as the draft model, SpecHub generates up to 3.53 and 3.33 tokens per iteration in the binary tree setting at $T = 0.6,\ 1.0$, an additional 0.08 tokens than RRS without replacement. We also tested the batch efficiency on optimal binary Sequoia trees(Chen et al., 2024). The full experiment results are in Appendix G.

5.3 Ablations

We analyze the performance of SpecHub across different temperatures (T) and compare it with Recursive Rejection Sampling (RRS) and RRS without replacement (RRSw). We use a binary token tree of depth $d = 5$ with JF68m as the draft model for Llama-2-7b. As shown in Figure 6, SpecHub consistently outperforms both RRS and RRSw regarding batch efficiency across all temperature settings. At lower temperatures $(T < 0.4)$, SpecHub assimilates RRSw in performance. At medium $(0.4 \leq T \leq 0.6)$ and higher temperatures $(T > 0.6)$, SpecHub maintains superior performance, demonstrating its robustness and adaptability across varying entropy levels.

6 Conclusion

We presented SpecHub, a versatile and provably faster verification method for Multi-Draft Speculative Decoding. By improving the coupling of the draft and target distributions, SpecHub can increase the acceptance rate of the second draft by $1 - 5\%$, which increases the batch efficiency of autoregressive LLM inference by up to 0.27 tokens per iteration. In addition to providing practical speedups, we believe SpecHub also provides insight into the underlying mathematical structure in MDSD. We hope this insight promotes future research in this area.

Limitations

Our algorithm, SpecHub, is currently designed to support only two drafts due to the computational complexities associated with using more drafts. This limitation may affect users who rely heavily on large-scale parallel computations, particularly when the number of nodes in the token tree exceeds 32. However, such extensive parallelism is rarely utilized in practical applications, and most users will not encounter this limitation.

Ethical Statement

This work focuses on accelerating LLM inferencing. There are no potential risks or negative effects that the authors are aware of. Additionally, we ensured that all datasets and benchmarks used in the article comply with their intended purposes and standards.

Use of AI

During our research, we used LLMs to help write code, parse experiment results, and revise languages in paper writing.

References

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 preprint arXiv:2401.10774*.

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

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

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Jie Huang, and Kevin Chen-Chuan Chang. 2023b. Cascade speculative drafting for even faster llm inference. *arXiv preprint arXiv:2312.11462*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. 2023. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*.

Wonseok Jeon, Mukul Gagrani, Raghavv Goel, Junyoung Park, Mingu Lee, and Christopher Lott. 2024. Recursive speculative decoding: Accelerating llm inference via sampling without replacement. *arXiv preprint arXiv:2402.14160*.

Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. 2023. Online speculative decoding. *arXiv preprint arXiv:2310.07177*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Benjamin Spector and Chris Re. 2023. Accelerating llm inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024. Multi-candidate speculative decoding. *arXiv preprint arXiv:2401.06706*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *Preprint*, arXiv:2309.08168.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*.

A Related Work

Speculative Decoding Speculative decoding aims to execute multiple decoding steps in parallel. Early work (Stern et al., 2018) predicts future tokens to accelerate greedy decoding. Speculative Sampling (Chen et al., 2023a; Leviathan et al., 2023) extends to non-greedy decoding and uses rejection sampling to recover target distribution optimally. Recent works focus on reducing the running time of the draft model and increasing the acceptance rate. OSD (Liu et al., 2023) and DistillSpec (Zhou et al., 2023) train draft models on text generated by the target model. REST (He et al., 2023) constructs drafts through retrieval. Lookahead Decoding (Fu et al., 2024) breaks the sequential dependency with Jacobi Iterations. Self-Speculative Decoding (Zhang et al., 2023; Elhoushi et al., 2024) avoids additional models and generates draft tokens by skipping intermediate layers. Several works, such as MEDUSA (Cai et al., 2024) and EAGLE (Li et al., 2024), reuse the feature embedding of LLMs' last attention layer to predict multiple future tokens in a non-causal or autoregressive manner.

Multi-Draft Speculative Decoding Recent research explores using tree attention to generate multiple drafts for speculative decoding (Miao et al., 2023; Spector and Re, 2023; Li et al., 2024). Sun et al. (Sun et al., 2024) formulate the acceptance of multiple drafts as a maximal coupling problem between the drafts and the target distributions and propose SpecTr with $1-\frac{1}{e}$ optimality guarantee. CS Drafting (Chen et al., 2023b) swaps in a lower-quality model to generate drafts for less relevant branches. Medusa (Cai et al., 2024) establishes candidates according to the Cartesian product of the multi-head predictions. Independently, Jeon et al.(Jeon et al., 2024) and Yang et al. (Yang et al., 2024) notice that a rejected token has zero probability in the residual distribution and use sampling-without-replacement in the draft generation round with the stochastic beam search technique (Kool et al., 2019). Sequoia (Chen et al., 2024) designed a dynamic programming algorithm to search for the optimal tree topology.

B Correctness of the LP formulations

We prove Theorem 1 to show that the simplified LP formulation is equivalent to the Optimal Transport with Membership Cost (OTM) problem.

Proof. We first show that we can construct a valid coupling from a valid solution to the simplified LP formulation. Given a solution represented by $\pi(x_{1:k}, x_i)$, we can derive a complete coupling $\pi(x_{1:k}, y)$, which represents the joint probability distribution of the k draft tokens $x_{1:k}$ and the target token y.

The construction process involves allocating probabilities based on the LP solution. For each possible combination of draft tokens and target token $(x_{1:k}, y)$, if y matches any of the draft tokens, meaning $y = x_i$ for some i, then the corresponding entry in the transport plan is given by the solution to the LP:

$$\pi(x_{1:k}, y) = \pi(x_{1:k}, x_i)$$

If the target token y is different from all draft tokens, the probability is calculated as the product of two terms:

$$\begin{aligned}&\pi(x_{1:k}, y)\\ &= \frac{p(y) - \sum_{i=1}^{k}\sum_{x_{1:k}\in\mathcal{V}^k, x_i=y}\pi(x_{1:k}, y)}{\sum_{y\in\mathcal{V}} p(y) - \sum_{i=1}^{k}\sum_{x_{1:k}\in\mathcal{V}^k, x_i=y}\pi(x_{1:k}, y)}\\ &\quad\cdot\left(Q(x_{1:k}) - \sum_{i=1}^{k}\pi(x_{1:k}, x_i)\right)\end{aligned}$$

The first term is the unallocated target probability mass or the residual probability of y normalized. The second term is the remaining probability mass of the series of drafts $x_{1:k}$ after allocating probabilities to cases where the target token matches a draft token.

We now verify that the constructed π is indeed a valid coupling. First, we need to show that the marginal distribution on the target token y is indeed

$p(y)$:

$$\sum_{x_{1:k}} \pi(x_{1:k}, y)$$
$$= \sum_{i=1}^{k} \sum_{x_{1:k}, x_i = y} \pi(x_{1:k}, y)$$
$$+ (p(y) - \sum_{i=1}^{k} \sum_{x_{1:k}, x_i = y} \pi(x_{1:k}, y))$$
$$= p(y).$$

Then, we verify that the marginal distribution on the series of drafts is the joint draft distribution:

$$\sum_{y} \pi(x_{1:k}, y)$$
$$= \sum_{i=1}^{k} \pi(x_{1:k}, x_i)$$
$$+ \sum_{y \neq x_i \forall i} \left(\frac{p(y) - \sum_{i=1}^{k} \sum_{x_{1:k}, x_i = y} \pi(x_{1:k}, y)}{\sum_{y \in \mathcal{V}} p(y) - \sum_{i=1}^{k} \sum_{x_{1:k}, x_i = y} \pi(x_{1:k}, y)} \right.$$
$$\left. \cdot (Q(x_{1:k}) - \sum_{i=1}^{k} \pi(x_{1:k}, x_i)) \right)$$
$$= Q(x_{1:k})$$

Now, we show that an optimal solution to the simplified LP formulation is also optimal for the OTM problem.

We prove this by contradiction. Assume there exists a coupling π' that achieves a lower transport cost than the optimal solution to the simplified LP formulation. We can construct a solution $\pi''(x_{1:k}, x_i)$ to the LP from π' by setting $\pi''(x_{1:k}, x_i) = \pi'(x_{1:k}, x_i)$. This π'' will have the same objective value as the transport cost of π', contradicting the optimality of the LP solution. Therefore, an optimal solution to the simplified LP formulation is also an optimal solution to the OTM problem. □

C Properties of SpecHub

C.1 Pseudocode Implementation of SpecHub

The transport plan of top token a is:

$$\pi(x, a, x) = \min(p(x), q(x))$$
$$\pi(a, x, x) = \min(p(x) - \pi(x, a, x), Q(a, x))$$
$$\pi(a, x, a) = \min(p(a), \sum_{x \in \mathcal{V}} (Q(a, x) - \pi(a, x, x))$$
$$\cdot \frac{Q(a, x) - \pi(a, x, x)}{\sum_{x \in \mathcal{V}} (Q(a, x) - \pi(a, x, x))}$$
$$\pi(x, a, a) = \min(p(a) - \sum_{x} \pi(a, x, a),$$
$$\sum_{x \in \mathcal{V}} q(x) - \pi(x, a, x))$$
$$\cdot \frac{q(x) - \pi(x, a, x)}{\sum_{x \in \mathcal{V}} q(x) - \pi(x, a, x)}$$

Here we provide the pseudocode for using SpecHub in real life. We follow a sequential procedure and avoid explicitly writing out the underlying coupling π.

Algorithm 2 GetResidual

1: **Inputs:** target distribution p, draft distribution q, highest probability token a
2: **for all** x in $\mathcal{V}$, $x \neq a$ **do**
3: $\quad p'(x) = \max(p(x) - q(x), 0)$
4: $\quad q'(x) = \max(q(x) - p(x), 0)$
5: **end for**
6: $p'(a) = p(a)$
7: $q'(a) = 0$
8: **return** p', q'

C.2 Correctness

Here, we proof that SpecHub does not sacrifice the quality of generation.

Theorem 2. *Given a target distribution p and a draft distribution q, SpecHub generates tokens such that for any token $x \in \mathcal{V}$, the probability of generating x under SpecHub, denoted as $\mathbb{P}(X = x)$, is equal to $p(x)$.*

Proof. Given a target distribution p and a draft distribution q, we need to show that SpecHub generates tokens such that for any token $x \in \mathcal{V}$, the probability of generating x under SpecHub, denoted as $P_{\text{SpecHub}}(x)$, is equal to $p(x)$.

First, all draft pairs sampled by SpecHub involve the top token $a = \arg\max_{x \in \mathcal{V}} q(x)$. For all $x \neq a$, pairs (x, a) and (a, x) are sampled with probabilities $Q(x, a) = q(x)$ and $Q(a, x) = \frac{q(a)q(x)}{1 - q(a)}$, respectively.

Algorithm 3 Sampling and Verification with SpecHub

Inputs: target distribution p, draft distribution q, vocabulary $\mathcal{V}$
Let $a = \arg\max_x q(x)$ be the token with the highest draft probability.
for all $i \in \mathcal{V}$, $x \neq a$ **do**
  $Q(x,a) = q(x)$, $Q(a,x) = \frac{q(a)q(x)}{1-q(a)}$
end for
Sample draft tokens $x^{(1)}, x^{(2)} \sim Q$
if $x^{(2)} = a$ **then**
  Return $x^{(1)}$ with probability $\min\left(\frac{p(x^{(1)})}{Q(x^{(1)},a)}, 1\right)$
end if
$p', Q'(*,a) =$ `GetResidual`$(p, Q(*,a), a)$
if $x^{(1)} = a$ **then**
  Return $x^{(2)}$ with probability $\min\left(\frac{p'(x^{(2)})}{Q(a,x^{(2)})}, 1\right)$
end if
$p'', Q'(a,*) =$ `GetResidual`$(p', Q(a,*), a)$
if $x^{(1)} = a$ **then**
  Return a with probability $\min\left(\frac{p(a)}{\sum_x Q'(a,x)}, 1\right)$
  $p'(a) = \max(p(a) - \sum_x Q'(a,x), 0)$
end if
if $x^{(2)} = a$ **then**
  Return a with probability $\min\left(\frac{p'(a)}{\sum_x Q'(x,a)}, 1\right)$
  $p''(a) = \max(p'(a) - \sum_x Q'(x,a), 0)$
end if
Return a token sampled from the residual distribution norm(p'')

For a token $x \neq a$, in the first draft, SpecHub generates x with probability

$$\begin{aligned}
&\mathbb{P}(x = x^{(1)} \text{ and } X = x) \\
&= Q(x,a)\min\left(\frac{p(x)}{Q(x,a)}, 1\right) \\
&= \min\left(p(x), q(x)\right).
\end{aligned}$$

In the second draft, given that $x \neq a$, the residual probability for token x after the first draft, denoted as $p'(x)$, is:

$$\begin{aligned}
p'(x) &= \max(p(x) - q(x), 0) \\
&= p(x) - \min(p(x), q(x))
\end{aligned}$$

SpecHub generates x in the second draft with probability

$$\begin{aligned}
&\mathbb{P}(x = x^{(2)} \text{ and } X = x) \\
&= Q(a,x)\min\left(\frac{p'(x)}{Q(a,x)}, 1\right) \\
&= \min\left(p(x) - \min(p(x), q(x)), Q(a,x)\right) \\
&= \min\left(p(x) - \min(p(x), q(x)), \frac{q(a)q(x)}{1-q(a)}\right).
\end{aligned}$$

Now, let's calculate the residual distribution after both drafts for tokens $x \neq a$. The residual probability $p''(x)$ for token x is calculated as follows:

$$\begin{aligned}
&p''(x) \\
&= \max(p'(x) - Q(a,x), 0) \\
&= \max\left(p(x) - q(x) - \frac{q(a)q(x)}{1-q(a)}, 0\right)
\end{aligned}$$

Since $p''(x)$ represents the remaining probability after both drafts, it ensures that:

$$\begin{aligned}
&\mathbb{P}(X = x) \\
&= \mathbb{P}(x = x^{(1)} \text{ and } X = x) \\
&\quad + \mathbb{P}(x = x^{(2)} \text{ and } X = x) \\
&\quad + p''(x) \\
&= \min(p(x), q(x)) \\
&\quad + \min\left(p(x) - \min(p(x), q(x)), \frac{q(a)q(x)}{1-q(a)}\right) \\
&\quad + \max\left(p(x) - q(x) - \frac{q(a)q(x)}{1-q(a)}, 0\right) \\
&= p(x)
\end{aligned}$$

Now for $x = a$:
In the first draft, SpecHub generates a with probability

$$\begin{aligned}
&\mathbb{P}(a = x^{(1)} \text{ and } X = a) \\
&= \sum_x Q'(a,x)\min\left(\frac{p(a)}{\sum_x Q'(a,x)}, 1\right) \\
&= \min\left(p(a), \sum_x Q'(a,x)\right).
\end{aligned}$$

In the second draft, given that $a = x$, the residual probability for token a after the first draft, denoted as $p'(a)$, is:

$$p'(a) = \max(p(a) - \sum_x Q'(a,x), 0).$$

SpecHub generates a with probability

$$\begin{aligned}
&\mathbb{P}(a = x^{(2)} \text{ and } X = a) \\
&= \sum_x Q'(x,a) \min\left(\frac{p'(a)}{\sum_x Q'(x,a)}, 1\right) \\
&= \min\left(\max(p(a) - \sum_x Q'(a,x), 0), \sum_x Q'(x,a)\right)
\end{aligned}$$

The total probability for generating a is:

$$\begin{aligned}
&\mathbb{P}(X = a) \\
&= \mathbb{P}(a = x^{(1)} \text{ and } X = a) \\
&\quad + \mathbb{P}(a = x^{(2)} \text{ and } X = a) \\
&= \min\left(p(a), \frac{p(a)}{\sum_x Q'(a,x)}\right) \\
&\quad + \min\left(\max(p(a) - \sum_x Q'(a,x), 0), \frac{p(a)}{\sum_x Q'(x,a)}\right) \\
&= \min\left(p(a), \sum_x Q'(a,x) + Q'(x,a)\right)
\end{aligned}$$

It can be shown that $p(a) < \sum_x Q'(a,x) + Q'(x,a)$. First, since $Q(a,a) = 0)$, we have

$$\begin{aligned}
&\sum_x Q(a,x) + Q(x,a) \\
&= \sum_{x \in \mathcal{V} \setminus \{a\}} q(x) + \frac{q(a)q(x)}{1 - q(a)} \\
&= 1
\end{aligned}$$

Also, we have $p(a) = 1 - \sum_{x \in \mathcal{V} \setminus \{a\}} p(x)$. Thus,

$$\begin{aligned}
&\sum_{x \in \mathcal{V} \setminus \{a\}} Q'(a,x) + Q'(x,a) \\
&= \sum_{x \in \mathcal{V} \setminus \{a\}} (\max(Q(a,x) - p(x), 0) \\
&\quad + \max(Q(x,a) - p'(x), 0)) \\
&= \sum_{x \in \mathcal{V} \setminus \{a\}} \max(Q(a,x) + Q(x,a) - p(x), 0) \\
&\geq \sum_{x \in \mathcal{V} \setminus \{a\}} Q(a,x) + Q(x,a) - p(x) \\
&= \sum_{x \in \mathcal{V} \setminus \{a\}} Q(a,x) + Q(x,a) - \sum_{x \in \mathcal{V} \setminus \{a\}} p(x) \\
&= 1 - (1 - p(a)) = p(a)
\end{aligned}$$

Thus, for any token $x \in \mathcal{V}$, the probability of generating x under SpecHub is equal to $p(x)$, ensuring that the output distribution matches the target distribution p. □

As a corrolary of the last part of the proof, SpecHub accepts as much top token a as $p(a)$.

Corollary 1 (Top Token Acceptance)**.** *Given a draft distribution q and a target distribution p, let $a = \arg\max_{x \in \mathcal{V}} q(x)$ denote the token with the highest draft probability. Then, SpecHub generates token a with probability $p(a)$.*

C.3 Acceptance Rate

We here prove a sufficient condition for SpecHub to run faster than RRS.

Theorem 3 (Superiority over RRS)**.** *Let $\alpha = \sum_{x \in \mathcal{V}} \min(q(x), p(x))$ be the acceptance rate of the first draft. SpecHub has a higher acceptance rate in the second draft if $\frac{q(a)}{1-q(a)} > 1 - \alpha$.*

Proof. First, by Lemma 1, SpecHub generates the top token a with probability $p(a)$. This maximizes the acceptance rate for a. Next, we calculate the second draft acceptance rate for every other token $x \in \mathcal{V} \setminus \{a\}$.

For RRS, the acceptance rate for token x in the first draft is $\min(p(x), q(x))$. The residual probability for token x after the first draft, denoted as $r(x)$, is:

$$p'(x) = \frac{p(x) - \min(p(x), q(x))}{1 - \alpha}$$

where $\alpha = \sum_{x \in \mathcal{V}} \min(p(x), q(x))$ is the overall acceptance rate in the first draft. The second draft acceptance rate for token x under RRS is then:

$$(1 - \alpha) \min\left(\frac{p(x) - \min(p(x), q(x))}{1 - \alpha}, q(x)\right)$$

which simplifies to:

$$\min\left(p(x) - \min(p(x), q(x)), (1 - \alpha) q(x)\right)$$

For SpecHub, the second draft acceptance rate for token x is:

$$\min\left(p(x) - \min(p(x), q(x)), \frac{q(a)}{1 - q(a)} q(x)\right)$$

Comparing these rates shows that SpecHub has a higher acceptance rate if $\frac{q(a)}{1-q(a)} > 1 - \alpha$. □

In practice, this condition is usually satisfied. For example, if $\alpha = 0.5$, then as long as the top token has probability $q(a) > \frac{1}{3} = 0.333$, we guarantee acceleration. Meanwhile, since SpecHub accepts top tokens up to $p(a)$, the above sufficient conditions become necessary only in unusual cases when $p(a) = 0$.

Using a similar proof strategy, we can show it guarantees to outperform OTM with independent sampling in rare cases.

Theorem 4 (Superiority over OTM)**.** *SpecHub guarantees a higher total acceptance rate compared to OTM with independent sampling if* $q(a) > 1/2$.

Proof. Let $Q = q^{\otimes 2}$. Then, for a token x, it is contained in any draft pair with probability $1 - (1 - q(x))^2 < 2q(x)$. Meanwhile, for the first and second drafts, we can accept up to $\frac{q(a)}{1-q(a)} q(x) + q(x) = \frac{q(x)}{1-q(a)}$. Thus, we can accept more of token x if $\frac{q(x)}{1-q(a)} > 2q(x)$, or $q(a) > 1/2$ $\square$

Compared to the previous theorem, this bound is nowhere near as tight since we are using a loose lower bound on OTM's performance. In reality we expect OTM to perform worse.

C.4 First Draft Acceptance Rate

SpecHub is designed to optimize the acceptance rate across multiple drafts, but in rare cases, it might slightly decrease the acceptance rate of the top token in the first draft. This occurs when the probability of the top token in the target distribution, $p(a) > q(a)$, while another token x takes some of the probability mass $Q(a, x)$. However, our empirical evaluations demonstrate that this effect is not noticeable in practice, as the acceptance rates of the first draft remain high.

Table 3: First Draft Acceptance Rates for SpecHub and RRSw across different models and temperatures.

T	Draft	SpecHub	RRSw
0.3	JF68m	0.4921	0.4498
	JF160m	0.5578	0.5465
0.6	JF68m	0.4842	0.4821
	JF160m	0.5632	0.5587
1	JF68m	0.4248	0.4418
	JF160m	0.5130	0.5257

D A discussion on more drafts

D.1 Diminishing Returns of Increasing Drafts

While theoretically appealing, using more drafts in practice offers diminishing returns. As we increase the number of drafts, the probability mass of the residual distribution decreases, leading to lower acceptance rates for subsequent drafts. This phenomenon is illustrated in Figure 7, where we present the acceptance rates for up to 10 drafts using both RRSw and RRS with temperature $T = 1.0$. As evident from the plots, the acceptance rate drastically decreases after the first few drafts, suggesting that the benefit of using more than 5 drafts is negligible.

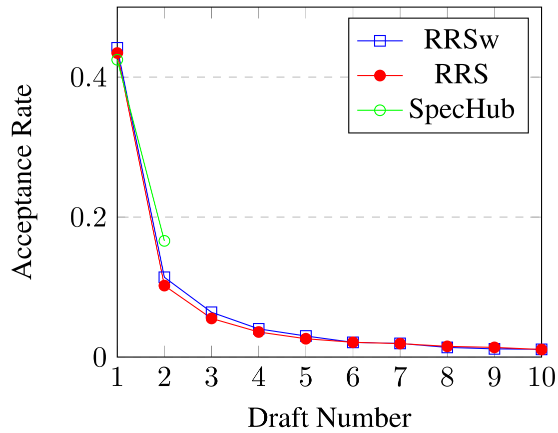


Figure 7: Acceptance rate decay for different drafts with temperature $T = 1.0$.

D.2 Curse of Dimensionality

The computational complexity of finding the optimal coupling in Multi-Draft Speculative Decoding grows exponentially with the number of drafts. This is often referred to as the curse of dimensionality. Specifically, the number of variables in the LP formulation is on the order of $O(|\mathcal{V}|^{k+1})$, where $|\mathcal{V}|$ is the vocabulary size and k is the number of drafts. As k increases, solving the LP becomes computationally intractable for even moderately sized vocabularies.

D.3 Potential for Sparse Algorithms on more drafts

The diminishing returns of additional drafts and the curse of dimensionality suggest that a practical approach should focus on a small number of drafts while ensuring an efficient probability of mass transport. One promising direction is to explore sparse algorithms that leverage the structure of the draft and target distributions. For instance, instead of considering all possible combinations of drafts, we can prioritize those with higher sampling probabilities or those that exhibit significant overlap between the draft and target distributions. One potential approach is to extend the "hub" concept of SpecHub to multiple drafts. Instead of designating a single token as the hub, we can identify

Table 4: **Acceptance Rates for Toy Experiments** The acceptance rates for SpecHub, Recursive Rejection Sampling (RRS), and Optimal Transport (OTM) algorithms using toy example drafts and target distributions. T represents the temperature, and λ controls the similarity between the draft and target distributions. We highlight the **best**, second best, and *third best* entries.

T	λ	RRS	RRSw	OTM	OTMw	SpecHub
0.1	0.7	0.6273	*0.7120*	0.6380	0.7345	**0.7402**
0.1	0.5	0.3323	*0.4057*	0.3346	**0.4125**	0.4123
0.25	0.7	0.7354	0.7653	*0.7846*	**0.8321**	0.8113
0.25	0.5	0.4564	0.4978	0.4743	**0.5245**	*0.4968*
0.5	0.7	0.8090	0.8122	0.9037	**0.9150**	*0.8500*
0.5	0.5	0.6456	*0.6593*	0.7052	**0.7206**	0.6403

a small set of high-probability tokens and create a sparse flow network where probability mass is primarily transported through these hubs. This approach could potentially maintain high acceptance rates while significantly reducing the computational complexity compared to solving the full LP. Further research in this direction could lead to more efficient and scalable algorithms for MDSD.

E Comparing SpecHub to OTM in Toy Settings

We demonstrate the acceptance rate for SpecHub, RRS, and OTM algorithms using a few toy example drafts and target distributions with a small vocab size $|\mathcal{V}| = 50$ in Table 4. Given temperature T and a hyperparameter λ that controls the similarity between the two distributions, we generate two logits using uniform distributions such that $u_p \sim \text{Unif}(0,1)^{\otimes|\mathcal{V}|}$ and $u_q \sim \text{Unif}(0,1)^{\otimes|\mathcal{V}|}$. The corresponding target and draft distributions are $p = \text{softmax}(\frac{u_p}{T})$ and $q = \text{softmax}(\lambda\frac{u_p}{T} + (1-\lambda)\frac{u_q}{T})$. We calculate the acceptance rate for all methods theoretically except for RRS without replacement, where we perform a Monte-Carlo Simulation with a thousand repetitions. We conduct the experiment on a hundred pairs of toy distributions and report the average. The results in Table 4 quantitatively illustrate the performance differences among SpecHub, Recursive Rejection Sampling (RRS), RRS without replacement, and Optimal Transport (OTM) methodologies under varying conditions of temperature T and similarity parameter λ. In high similarity scenarios ($\lambda = 0.7$), SpecHub outperforms other methods significantly at lower temperatures ($T = 0.1$), achieving the best acceptance rate of **0.7402**, closely followed by OTM without replacement at 0.7345. At higher temperatures ($T = 0.5$), OTM methods, particularly OTM without replacement, dominate, marking the best performance with **0.9150** at $T = 0.5$ and $\lambda = 0.7$. This suggests that SpecHub is particularly effective in tightly controlled environments with high similarity between distributions and low entropy, whereas OTM shines with increased distribution complexity. SpecHub's consistent performance across different conditions emphasizes its robustness, particularly when distribution similarity is moderate ($\lambda = 0.5$), where it maintains competitive acceptance rates, closely trailing the best results.

F Maximum Flow Problem Formulation

At $k = 2$, our Linear Programming (LP) formulation describes an equivalent Maximum Flow Problem formulation. This formulation effectively models the Multi-Draft Speculative Decoding process as the transportation of probability mass through a network of pipes.

Given an LP formulation with vocabulary set $\mathcal{V}$, pair sampling distribution $Q \in \Delta^{|\mathcal{V}|^2-1}$, and target distribution $p\Delta^{|\mathcal{V}|-1}$, we construct a graph $G = (V, E)$ where the vertex set V consists of the vocabulary $\mathcal{V}$, a source vertex s, and a sink vertex t. The capacity function $g : (u, v) \in E \to [0, 1]$ is defined for each edge as follows:

$$g(u,v) = \begin{cases} \sum_{x^{(2)}} Q_{vx^{(2)}}, & \text{if } u = s \text{ and } v \in \mathcal{V}, \\ p(v), & \text{if } u \in \mathcal{V} \text{ and } v = t, \\ Q_{uv}, & \text{if } u, v \in \mathcal{V} \text{ and } u \neq v, \\ 0, & \text{otherwise.} \end{cases}$$

In this formulation, the source vertex s distributes the total probability mass to the vertices in the vocabulary set $\mathcal{V}$, while the sink vertex t collects the transported probability mass from the vocabulary vertices. The edges between the vocabulary vertices represent the possible transitions dictated by the pair sampling distribution Q. This network flow model not only provides an intuitive visualization of the probability mass transport process but also allows us to leverage well-established algorithms in network flow theory to solve the MDSD problem efficiently.

G More Experiment Details

JF68m on Full Binary Trees and Binary Sequoia Unbalanced Trees We conducted experiments to measure the batch efficiency of the JF68m model on both full binary trees and binary Sequoia unbalanced trees. For the full binary trees, we tested tree depths ranging from $d = 2$ to $d = 5$, and for the binary Sequoia trees, we used an unbalanced tree structure with varying depths. The results demonstrate that SpecHub consistently outperforms both RRS and RRSw across all tree depths. In the full binary tree configuration, SpecHub achieves a batch efficiency improvement of $0.02 - 0.10$ over RRS and $0.04 - 0.20$ over RRSw at temperatures $T = 0.6$ and 1.0. For the binary Sequoia unbalanced trees, SpecHub maintains a higher batch efficiency, confirming its robustness across different tree structures.

JF160m on Binary and Ternary Trees We also evaluated the batch efficiency of the JF160m model on both binary and ternary trees. For binary trees, we tested tree depths from $d = 2$ to $d = 6$, and for ternary trees, we considered depths up to $d = 4$. The JF160m model shows significant improvements in batch efficiency when using SpecHub. At temperatures $T = 0.6$ and 1.0, SpecHub outperforms RRS by $0.03 - 0.12$ and RRSw by $0.05 - 0.15$ in binary tree configurations. In the ternary tree settings, SpecHub's batch efficiency gain is even more pronounced, highlighting its effectiveness in handling more complex tree structures.

EAGLE Decoding Head To further explore the efficiency of our proposed method, we implemented the SpecHub algorithm using the EAGLE decoding head. The batch efficiency was evaluated on binary trees of depths $d = 2$ to $d = 5$. SpecHub with the EAGLE decoding head shows a substantial increase in efficiency, generating up to 3.53 and 3.33 tokens per iteration at temperatures $T = 0.6$ and 1.0, respectively. This represents an additional 0.08 tokens per iteration compared to RRS without replacement. The experimental results reinforce the benefits of integrating SpecHub with advanced decoding heads like EAGLE, particularly in enhancing batch efficiency.

Table 5: **Batch Efficiency Results for JF68m Data** Average accepted tokens and batch efficiency for different configurations of target model and draft model pairs across various temperatures. SpecHub consistently outperforms RRS and RRSw in both acceptance rate and batch efficiency. We also include binary Sequoia trees and show that SpecHub performs well on unbalanced trees.

T	Data	Tree	RRS	RRSw	SpecHub	Tree	RRS	RRSw	SpecHub
0.6	CNN	2^2	1.5540	1.5997	**1.6157**	biSeq4	1.7938	1.8304	**1.8498**
0.6	OWT	2^2	1.5485	1.5895	**1.6080**	biSeq4	1.7971	1.8225	**1.8424**
0.6	CNN	2^3	1.8482	1.9685	**1.9863**	biSeq8	2.0361	2.1540	**2.1542**
0.6	OWT	2^3	1.8576	1.9241	**1.9632**	biSeq8	2.0247	2.1005	**2.1285**
0.6	CNN	2^4	2.0510	2.1694	**2.2456**	biSeq16	2.1354	2.2667	**2.2839**
0.6	OWT	2^4	2.0256	2.1299	**2.2103**	biSeq16	2.1378	**2.2153**	2.2064
0.6	CNN	2^5	2.1385	2.3149	**2.4031**	biSeq32	2.2452	2.4198	**2.4353**
0.6	OWT	2^5	2.0867	2.2295	**2.3416**	biSeq32	2.2007	2.3556	**2.3868**
1.0	CNN	2^2	1.5432	1.5521	**1.5997**	biSeq4	1.7401	1.7469	**1.8057**
1.0	OWT	2^2	1.5488	1.5509	**1.5905**	biSeq4	1.7355	1.7437	**1.7879**
1.0	CNN	2^3	1.8384	1.8790	**1.9832**	biSeq8	1.9522	2.0063	**2.0667**
1.0	OWT	2^3	1.8232	1.8585	**1.9661**	biSeq8	1.9304	2.0008	**2.0720**
1.0	CNN	2^4	1.9762	2.0441	**2.2106**	biSeq16	2.0529	2.1662	**2.2843**
1.0	OWT	2^4	1.9954	2.0493	**2.1957**	biSeq16	2.0330	2.1030	**2.2619**
1.0	CNN	2^5	2.0694	2.1383	**2.3104**	biSeq32	2.1197	2.1604	**2.3445**
1.0	OWT	2^5	2.0890	2.1574	**2.3149**	biSeq32	2.1008	2.1950	**2.3571**

Table 6: **Batch Efficiency Results for JF160m Data** Average accepted tokens and batch efficiency for different configurations of target model and draft model pairs at $T = 0.6$ and $T = 1.0$. The results are presented for CNN and OpenWebText datasets, comparing RRS, RRS without replacement, and TransportHub. We also contained ternary trees to showcase that using $k > 2$ gives marginal gain.

T	Data	Tree	RRS	RRS w/o	SpecHub
0.6	CNN	2^2	1.633994691	1.667634674	**1.6861**
0.6	OpenWebText	2^2	1.641550493	1.672971282	**1.677**
0.6	CNN	2^3	2.016376307	2.142804292	**2.1758**
0.6	OpenWebText	2^3	2.052868003	2.113952048	**2.115**
0.6	CNN	3^2	1.66262118	1.734944266	
0.6	OpenWebText	3^2	1.669826224	1.70473377	
0.6	CNN	2^4	2.282944241	2.369522017	**2.4841**
0.6	OpenWebText	2^4	2.28490566	2.411659014	**2.4492**
0.6	CNN	3^3	2.113219655	2.279599835	
0.6	OpenWebText	3^3	2.111602497	2.212962963	
0.6	CNN	2^5	2.378323523	2.604486152	**2.7238**
0.6	OpenWebText	2^5	2.449243411	2.642651616	**2.6901**
0.6	CNN	3^4	2.39760652	2.681949084	
0.6	OpenWebText	3^4	2.433582166	2.667044296	
1.0	CNN	2^2	1.608515798	1.633187465	**1.6748**
1.0	OpenWebText	2^2	1.633351663	1.635781207	**1.6834**
1.0	CNN	2^3	1.959878368	2.053886546	**2.1362**
1.0	OpenWebText	2^3	2.028797337	2.077786547	**2.1584**
1.0	CNN	3^2	1.663016602	1.689861121	
1.0	OpenWebText	3^2	1.677094972	1.701585742	
1.0	CNN	2^4	2.20357984	2.286009649	**2.4204**
1.0	OpenWebText	2^4	2.295532975	2.379759419	**2.4922**
1.0	CNN	3^3	2.105012354	2.165854573	
1.0	OpenWebText	3^3	2.166307084	2.233691623	
1.0	CNN	2^5	2.315296164	2.41812897	**2.6624**
1.0	OpenWebText	2^5	2.429887821	2.532017591	**2.7334**
1.0	CNN	3^4	2.382244389	2.474047719	
1.0	OpenWebText	3^4	2.467950678	2.550284031	

Table 7: **Batch Efficiency Results for SpecHub and RRS using EAGLE** The batch efficiency of SpecHub and Recursive Rejection Sampling (RRS) methods when applied with EAGLE. The table reports average accepted tokens per step across different temperatures and datasets, demonstrating that SpecHub consistently outperforms RRS.

T	Tree	RRS	RRS-wo	SpecHub
0.6	2^2	1.8211	1.8687	**1.8825**
0.6	2^3	2.4325	2.5585	**2.5939**
0.6	2^4	2.9125	3.0899	**3.1192**
0.6	2^5	3.2501	3.4838	**3.5380**
1.0	2^2	1.8054	1.8327	**1.8655**
1.0	2^3	2.3961	2.4737	**2.4850**
1.0	2^4	2.8425	2.9019	**3.0281**
1.0	2^5	3.1451	3.2548	**3.3318**