Fast Inference via Hierarchical Speculative Decoding

Anonymous Authors[1]

Abstract

Transformer language models generate text autoregressively, making inference latency proportional to the number of tokens generated. Speculative decoding reduces this latency without sacrificing output quality, by leveraging a small draft model to propose tokens that the larger target model verifies in parallel. In practice, however, there may exist a set of potential draft models—ranging from faster but less inaccurate, to slower yet more reliable. We introduce Hierarchical Speculative Decoding (HSD), an algorithm that stacks these draft models into a hierarchy, where each model proposes tokens, and the next larger model verifies them in a single forward pass, until finally the target model verifies tokens. We derive an expression for the expected latency of any such hierarchy and show that selecting the latency-optimal hierarchy can be done in polynomial time. Empirically, HSD gives up to 1.2× speed-up over the best single-draft baseline, demonstrating the practicality of our algorithm in reducing generation latency beyond previous techniques.

Most language models are trained with teacher-forcing to predict the next token in an autoregressive fashion. While this allows for a highly parallelizable training process, inference remains a sequential process: a model must finish a full forward pass and predict a token before it can start processing the new context to predict the following token. This sequential process typically does not fully utilize the compute capabilities of modern accelerators, making text generation slow and costly.

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023a) addresses this limitation by leveraging a smaller drafter model that autoregressively generates multiple tokens ahead. Then, these tokens are verified, and possibly discarded, by the larger target model in parallel with a single forward pass. By following the speculative sampling rejection rule, the output distribution of verified tokens is identical to that of the large model. Every round of drafting and verification yields at least one verified token in the worst case, and one more token than the number of drafted tokens in the best case.

Notably, there is a natural tradeoff in selecting the drafter for speculative decoding—a larger drafter will improve token acceptance rate but increase drafting latency. Many recent studies have investigated approaches for pushing the Pareto frontier of drafters (Liu et al., 2023; Xiao et al., 2024; Zhang et al., 2024; Miao et al., 2024; Hooper et al., 2023; Zhou et al., 2023). However, ultimately the practitioner may select the single drafter that provides the best accuracy-cost ratio. For example, when early exits from the target model are considered as drafters (Elhoushi et al., 2024; Kim et al., 2023; Zhang et al., 2024), the layer that gives the best accuracy vs. depth tradeoff will be used.

In this paper, we question the paradigm of using only a single drafter. We study the prospect of leveraging multiple drafters in a cost-effective way. To this end, we introduce the Hierarchical Speculative Decoding (HSD) algorithm. In HSD, each drafter validates sequences generated by lower drafters in the hierarchy, and only the smallest drafter (i.e., lowest in hierarchy) generates autoregressively. We prove that using multiple drafters can further reduce latency while preserving the quality of the output.

Next, we turn to the question of finding the hierarchy which results in the optimal latency. A key challenge is that the number of possible hierarchies grows exponentially with the number of drafters available, and therefore naive enumeration would be costly. Furthermore, our algorithm has tunable parameters which should also be optimized. To address this, we derive an expression for the expected latency incurred by a given hierarchy and its parameters, and show that this expression can in fact be optimized in polynomial time. This is done via a reduction to the Generalized Shortest Path problem (Oldham, 2001), which admits a polynomial-time solution.

We validate the effectiveness of HSD empirically by implementing it on top of public open-source Large Language Models. In particular, we validate our findings across three

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

different classes of models: Gemma2 (Team et al., 2024), T5 (Raffel et al., 2020), and LayerSkip (Elhoushi et al., 2024), as well as four different datasets. Compared to both vanilla autoregressive decoding and to a single-drafter speculative decoding baseline, our method shows significant speedup gains. Hence, our main contributions are as follows:

1. **Theoretical**: We introduce the Hierarchical Speculative Decoding algorithm for accelerating inference in LLMs and analyze its latency in Section 1. In Section 2, we formulate its corresponding optimization problem, and provide an efficient solution for optimal hierarchy construction.

2. **Empirical**: In Section 3, we evaluate our method on open-source language models, and demonstrate speed up over speculative decoding with a single draft model.

Related Work

Speculative decoding. We build on the speculative decoding method (Chen et al., 2023a; Leviathan et al., 2023) for accelerating transformers. In this framework, an efficient draft model generates tokens autoregressively, which are then verified in parallel by a target model using a sampling method that guarantees an identical output distribution to the target model. While some follow up work suggests new verification algorithms (Liu et al., 2024; Narasimhan et al., 2025; Sun et al., 2025), the vast majority of studies focus on improving the performance of drafters via techniques such as distillation (Zhou et al., 2023), enhanced attention to past verifier predictions (Aishwarya et al., 2024), multi-token prediction heads (Cai et al., 2024; Gloeckle et al., 2024b; Li et al., 2024), and other self-speculation solutions (Zhang et al., 2024) that further leverage signals from the target model. Elhoushi et al. (2024) train a target model with an auxiliary early-exit loss (Elbayad et al., 2020; Schuster et al., 2022) in order to obtain draft tokens from a post-hoc selected earlier layer in the target model.

Our work is complementary to most previous advancements, and presents a departure from the single drafter paradigm by replacing it with a hierarchy of drafters with increasing cost and accuracy. Perhaps most relevant are the work of Sun et al. (2024) and Chen et al. (2023b). Sun et al. (2024) proposes a tailored two-stage hierarchy drafting method that leverages memory bottlenecks in certain deployment setups. In contrast, we introduce a general hierarchy framework with any set of appropriate drafter candidates, and develop an optimization solution for constructing the optimal hierarchy. Chen et al. (2023b) propose a similar algorithm with both horizontal and vertical cascades for speculative decoding. However, their work requires extensive hyperparameter search whereas ours provides a direct optimization algorithm. We show that we are able to quickly recover the performance of vertical and horizontal cascades leveraging our optimization algorithm. We describe further related work in Appendix B.

1. Hierarchical Speculative Decoding

We introduce the Hierarchical Speculative Decoding algorithm. The algorithm leverages several draft models in order to generate samples from a target model, and can improve upon the latency of using a single draft model. A key idea in our framework is that models in the hierarchy serve as both drafters and verifiers. We introduce the algorithm in Section 1.3 and analyze its latency in Section 1.4.

1.1. Background

We begin with a brief overview of speculative decoding. Given two language models $\mathcal{M}_q, \mathcal{M}_p$, the goal is to perform speculative decoding where $\mathcal{M}_q$ is a small draft model and $\mathcal{M}_p$ is a large target model. These may be arbitrary models, provided they share the same vocabulary $\mathcal{V}$.

We refer the reader to Leviathan et al. (2023) for further details and proofs. We also make use of the expected acceptance rate over the input distribution $\mathcal{D}$, $\alpha = \mathbb{E}_{c\sim\mathcal{D}}[\alpha_c]$. As with previous literature (Leviathan et al., 2023) we assume for our theoretical purposes that acceptances occur in an independent and identically distributed fashion. Although this is not necessarily the case in practice, we empirically validate that this assumption is not too strong in Section 3 as well as in Appendix C.

1.2. Preliminaries

We are given language models $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_K$, where $\mathcal{M}_K$ is the target model. All models share the same vocabulary $\mathcal{V}$, but are otherwise arbitrary. For example, models could be early-exits at different stages from the same model (Schuster et al., 2022). Each model $\mathcal{M}_i$ has an inference cost $c_i > 0$ which, for our purpose, is the time to complete a forward pass. The verification cost is similar to the token generation cost. We also assume that the cost of verifying a batch of tokens is the same as one token generation. The acceptance rate between $\mathcal{M}_i$ and $\mathcal{M}_j$ is $\alpha_{i,j} \in [0, 1]$, as defined in Section 1.1. $\mathcal{M}_i$ takes as input a context $c \in \{\mathcal{V}\}^L$, where $L > 0$ is the context length. It outputs a tuple (t, p) where $p \in [0, 1]^{|V|}$ is the distribution over the next token and $t \sim p$.

1.3. Main algorithm

We introduce HSD in Algorithm 2, a recursive procedure in which each model in the hierarchy requests draft tokens from the model below it. Upon receiving these draft tokens,

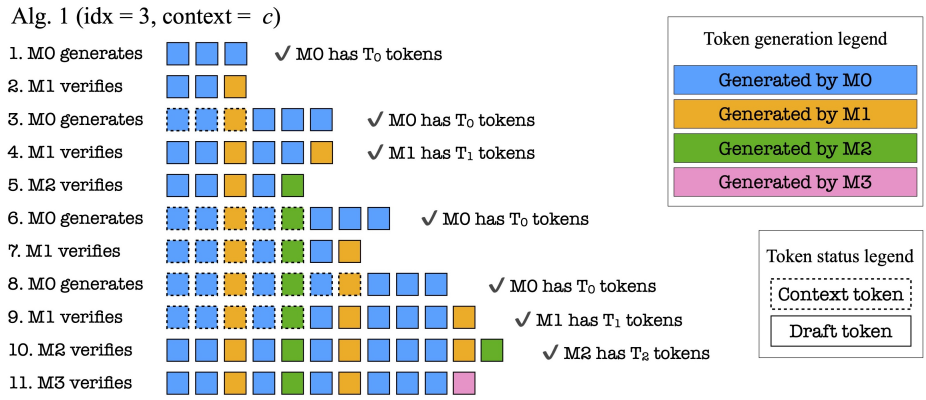


Figure 1. An example stack trace of HSD for $T_0 = 3, T_1 = 6, T_2 = 12$. The color of a token represents which model generated that token. A token can be generated either auto-regressively by the base model $\mathcal{M}_0$, or by the verification rule which can either replace a token or generates an additional token when all draft tokens are accepted. A token is considered to be part of the context for a certain model if a model above it accepted this token.

verification is performed. Every model, except the final target model, maintains a buffer of verified tokens that must reach a specified size before returning tokens upstream. Figure 1 illustrates an example stack trace.

To generate tokens from the target model $\mathcal{M}_K$, the process begins with the a call to HSD with the initial context and the model index set to K. The recursion reaches the base case when the smallest model, $\mathcal{M}_0$, generates tokens autoregressively. $\mathcal{M}_0$ generates T_0 tokens sequentially, which it passes to model $\mathcal{M}_1$ for verification. $\mathcal{M}_1$ verifies these tokens, and if fewer than T_1 tokens have been accepted, $\mathcal{M}_0$ continues generating batches of T_0 tokens for verification by $\mathcal{M}_1$. The verification procedure is detailed in Appendix C.

Pseudo-code for the verification function is provided in Appendix C, and is the same as that of Leviathan et al. (2023). Throughout, we use '+' to denote the concatenation of token sequences. The output of HSD follows the distribution of the target model. The proof is given in Appendix B.

1.4. Latency analysis

In this section, we derive an expression for the latency when all models are included. In order to analyze the latency theoretically, we assume that acceptances occur in an IID fashion, as already stated. This also implies that the number of tokens generated per recursive round is IID. Empirical results in Section 3 show that it is not an unreasonable assumption due to the alignment of the derived expected latency and the true latency.

We define the function $\gamma : [0,1] \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$, which counts the expected number of rounds of drafting and verification between a given pair of models. In particular, if $\mathcal{M}_j$ requires T_j tokens but receives T_i tokens at a time from $\mathcal{M}_i$, then $\gamma(\alpha_{i,j}, T_i, T_j)$ is the expected number of draft and verification rounds. Each one of these rounds results in a recursive call querying $\mathcal{M}_i$ for more tokens. In practice, we estimate the value of γ empirically. [1]

Theorem 1.1. *For a set of models $\{\mathcal{M}_i\}_{i\in[K]}$ where the pairwise acceptance rates are $\alpha_{i,j}$ for all $i, j \in [K]$, and parameters $T = \{T_0, \ldots, T_{K-1}\}$, the expected latency per token of HSD is:*

$$\sum_{i=0}^{K} c_i \prod_{j=i}^{K} R(\alpha_{j-1,j}, j),$$

where $R : [0,1] \times [K] \to \mathbb{R}$ is defined as:

$$R(\alpha, n) = \begin{cases} (1-\alpha)/\left(1-\alpha^{T_{K-1}+1}\right) & \text{if } n = K \\ \gamma(\alpha, T_{n-1}, T_n) & \text{if } 1 \le n < K \\ T_0 & \text{if } n = 0. \end{cases}$$

We defer the proof to Appendix B. A similar result, Corollary B.3, holds for subsets of models and their T parameters, which we defer to Appendix B as well.

1.5. Motivating example

We return to the question: *does there exist a configuration of models such that increasing the number of models included in the hierarchy decreases the latency from the target model?* We answer this question in the affirmative with an example configuration in Table 1, and provide details of the configuration used in the Appendix C.

2. Efficient Optimization of Hierarchies

The HSD algorithm is specified by a set of models and parameters. Thus, given a set of K potential draft models from

[1] Since we receive the tokens in multiples of T_j, we may collect more than T_i token. This makes it difficult to give an exact formula for γ.

Num. Models	Speedup	Latency
1	1.0000×	33.00
2	2.2971×	14.37
3	3.0211×	10.89
4	3.0620×	10.64
5	3.0829×	10.63
6	3.0839×	10.61

Table 1. An example of the expected speedup as the number of models provided to HSD increases.

which to choose, there are $O(2^K)$ possible sets of models. A question which arises is, *how do we find the hierarchy with the best latency?* Including all models might not necessarily be the optimal solution: perhaps there is a model which suffers a poor acceptance rate to the subsequent model, or perhaps two models are somewhat redundant. The problem becomes even more challenging when the objective is also to identify the optimal T parameters.

Finding the optimal hierarchy naturally requires having an estimate of the latency corresponding to each hierarchy. While this could be obtained via simulation, it would be costly and inefficient. In Section B.3, we provide the latency analysis for a subset of models. In Section 2.2, we show that, after selecting a maximum value for any parameter in T, the optimization can in fact be solved in polynomial time.

2.1. Preliminaries

Definition 2.1 (HSD problem)**.** Given a set of models $\{\mathcal{M}_0\}_{i=0}^K$ where $\mathcal{M}_K$ is the target model, and their pairwise acceptance rates, find the subset and parameters which attain the minimum latency L^*: $L^* = \min_{\sigma,T} L(\sigma, T)$.

Assuming a maximum T parameter value, we next show that the HSD problem can be solved via a reduction to the Generalized Shortest Path problem (Oldham, 2001), which is defined below.

Definition 2.2 (Generalized Shortest Path (GSP) Problem)**.** Given a directed graph $G = (V, E)$, an edge multiplier $\mu : E \to \mathbb{R} > 0$, an edge cost function $c : E \to \mathbb{R}$, and a source vertex $v \in V$, find the flow function $f : E \to \mathbb{R} \geq 0$ which satisfies:

$$\begin{aligned} \min \quad & \sum_{e \in E} f(e)c(e) \\ \text{s.t.} \quad & \sum_{(v,w)\in E} f(v,w) - \sum_{(u,v)\in E} \mu(u,v)f(u,v) \\ & \qquad = \mathbb{I}[v = s], && \forall v \in V \\ & f(e) \geq 0, && \forall e \in E. \end{aligned}$$

GSP describes a problem in which one unit of flow is sent from a designated source vertex. The objective is to find a path which minimizes the cost of sending out this unit of flow, subject to the constraint that the path must be flow-conserving. A key challenge in GSP is that, in addition to edges having a cost c, they also have flow multipliers: when flow traverses edge e, the flow is multiplied by $\mu(e)$. Given a graph with n vertices and m edges, GSP can be solved in $O(mn^2 \log n)$ time (Oldham, 2001). We give two definitions to be used in Lemma 2.5, which motivates our reduction.

Definition 2.3. A *lossy cycle* is a cycle whose product of flow multipliers is strictly less than 1.

Definition 2.4. An *augmented path* $s \rightsquigarrow v \rightsquigarrow w \to v$[2] is a nonempty path $s \rightsquigarrow v \rightsquigarrow w$ with an extra edge $w \to v$ forming a lossy cycle $v \rightsquigarrow w \to v$. It is a feasible solution to the GSP because the path transports the source's unit supply to a lossy cycle which "consumes" the flow reaching it.

Lemma 2.5 ((Oldham, 2001))**.** *Solutions to GSP must be augmented paths, or convex combinations of augmented paths with the same cost.*

Without loss of generality, we assume there is a unique augmented path which is optimal because an augmented path with equivalent cost can be obtained from a convex combination of such paths. The path which is the solution is determined by all the edges e for which $f(e) > 0$.

2.2. Reduction from HSD to GSP

Consider a set of models $\{\mathcal{M}_i\}_{i=0}^K$, where $\mathcal{M}_K$ is the target model. Each model $\mathcal{M}_i$ has cost c_i, and the acceptance rates are $\alpha_{i,j}$, $i, j \in [K]$. We set the maximum value for any of the T parameters to be $\overline{T} \in \mathbb{N}$. We create a graph G with the following vertices: **(1)** $(\mathcal{M}_K)$, **(2)** $(\mathcal{M}_i, j) \in \{\mathcal{M}_i\}_{i=0}^{K-1} \times \{1, \ldots, \overline{T}\}$, **(3)** $(\mathcal{M}_i, L) \in \{\mathcal{M}_i\}_{i=0}^{K-1} \times \{L\}$. The first vertex, $(\mathcal{M}_K)$, is the source vertex and corresponds to the target model. The second set of vertices correspond to the choices of models and parameter to use in the hierarchy. The third category of vertices are self-looping vertices representing the smallest model in the hierarchy. The graph G has directed edges:

(1) $(\mathcal{M}_K) \to (\mathcal{M}_i, j)$ $\forall i, j$, **(2)** $(\mathcal{M}_i, j) \to (\mathcal{M}_k, \ell)$ $\forall i > k, j \geq \ell$, **(3)** $(\mathcal{M}_i, j) \to (\mathcal{M}_i, L)$ $\forall i, j$, **(4)** $(\mathcal{M}_i, L) \to (\mathcal{M}_i, L)$ $\forall i$. Having defined the edges, we define μ and c over these edges as shown in Table 9. We provide an example visualization of the graph reduction in Appendix D.

Theorem 2.6. *In the above reduction, a path is a solution to the GSP instance defined above if and only if the corresponding hierarchy is an optimal solution to original HSD problem.*

[2] $s \rightsquigarrow v$ denotes some path starting at s and ending at v.

Model	Method	σ	T	Speedup(↑)	Seconds per Token(↓)
LayerSkip2-7B (CNN-DM)	HSD	$[7, 9, 32]$	$[2, 5]$	**1.76×**	0.0102
	Baseline	$[8, 32]$	$[12]$	1.62×	0.0113
	Target Model	-	-	1.00×	0.0182
LayerSkip2-13B (CNN-DM)	HSD	$[7, 18, 40]$	$[2, 6]$	**1.41×**	0.0162
	Single Draft	$[15, 40]$	$[12]$	1.20×	0.0190
	Target Model	-	-	1.00×	0.0228
LayerSkip2-70B (CNN-DM)	HSD	$[7, 23, 80]$	$[2, 6]$	**1.77×**	0.0410
	Single Draft	$[19, 80]$	$[5]$	1.58×	0.0456
	Target Model	-	-	1.00×	0.0723
Gemma2-9B (CNN-DM)	HSD	$[0, 2, 42]$	$[1, 1]$	**1.06×**	0.0407
	Single Draft	$[1, 42]$	$[2]$	1.03×	0.0418
	Target Model	-	-	1.00×	0.0430
Gemma2-9B (XSUM)	HSD	$[0, 1, 42]$	$[1, 2]$	**1.15×**	0.0373
	Single Draft	$[1, 42]$	$[2]$	1.08×	0.0395
	Target Model	-	-	1.00×	0.0429

Table 2. Results for the LayerSkip models and Gemma2 models. We compare HSD against the single draft baseline and the autoregressive baseline.

The proof relies on showing a bijection between augmented paths in the GSP instance and hierarchies with their parameters in the HSD instance. The bijection shows that the cost of a path in the GSP instance is equal to the latency of the corresponding hierarchy with those parameters in the HSD instance. We defer the proof to Appendix B.

Theorem 2.7. *HSD can be solved in time* $O(\overline{T}^4 K^4 \log(\overline{T}K))$.

The proof is provided in Appendix B.

3. Self-speculative validation

In this section, we empirically validate HSD where all of the candidate draft models are part of the target model.

3.1. Datasets and Models

We evaluate our method on datasets commonly used for evaluating speculative decoding: CNN-DM (Hermann et al., 2015) and XSUM (Narayan et al., 2018). We use two classes of models for evaluation.

LayerSkip: The LayerSkip (Elhoushi et al., 2024) class of models have been trained with an early-exit objective. Thus, each of their layers can be used as a draft model. This is done by applying the LM head to any of the layers. We consider the 7B, 13B and 70B versions of these models, which have $32, 40, 80$ layers respectively. We use the published checkpoints for each of these models.

Gemma2-9B: The Gemma2-9B model (Team et al., 2024) has 42 layers. However, Gemma2-9B was not trained to perform early-exiting like the LayerSkip models, so we undertake additional training; for every layer, we attach a language model head (a linear layer mapping from the embedding dimension to the vocabulary) and train it to match the output distribution of the final layer (Hinton et al., 2015). We use a learning rate of 2^{-4} with a 5% linear warmup followed by cosine decay for two epochs. Only the LM head is trained and the backbone remains frozen. We train a variant of this model using the respective training sets for CNN-DM and XSUM, and further finetune the smallest model in the hierarchy to match the intermediate model.

In both classes of models, we have a candidate draft model $\mathcal{M}_i$ for each layer i. In the Gemma setting, memory overhead grows linearly with the number of models included in the hierarchy. This is because one LM head is required to be stored in memory per model. In our experiments, the model and additional LM heads fit comfortably onto one GPU. Because the LayerSkip model uses the same LM head for each layer, the same overhead does not apply in the LayerSkip setting.

In contrast to standard autoregressive decoding, speculative decoding introduces an additional memory overhead. This is because the output distributions of all draft tokens must be stored. As with speculative decoding, HSD also incurs this overhead.

3.2. Optimization and Results

In order to find the optimal hierarchy, we first require knowledge of the acceptance rates $\alpha_{i,j}$ for each pair of candidate models. We approximate the rates in an efficient manner by doing a pass over the dataset for a subset of prompts, recording the output distributions from all layers during each forward pass, and computing the empirical acceptance

rate using the total variation distance of distributions. This takes about one hour with four GPUs. Simultaneously, we record the cost associated with each layer. We use these values to create our GSP instance, and run a GSP solver.

The GSP solver identifies the optimal hierarchy given the acceptance rates and costs. We consider a sufficiently large maximum value for T to be 15. The GSP solver runs on a CPU and takes about two hours to run to completion. We consider the following baselines.

Single Draft: Among all candidate models we take use the one that results in the minimal latency when used as a single draft model as in standard speculative decoding. We use the acceptance rates to identify the optimal two-layer setting as suggested in Leviathan et al. (2023). This baseline checks if using a hierarchy is advantageous over a single draft.

Target Model: We sample autoregressively from the target model, without any speculative decoding.

There are of course many other potential baselines. However, since other methods such as Ankner et al. (2024); Li et al. (2024) work on top of the standard two layer speculative decoding setting, we believe that the hierarchical setting should be extended to such methods in order to provide a valid baseline. Hence, we leave these methods for future work.

We present our results in Table 2. We report the average time per token as well as the speedup over autoregressive decoding, with batch size one. In all cases studied, HSD improves latency over both the single draft baseline as well as autoregressive generation. The speedup with respect to the single draft baseline is as much as 1.17× faster, showing the benefits of using HSD over standard speculative decoding. The results hold across various model sizes, as we experiment with models going from 7B parameters to 70B parameters. The greatest improvement can be seen on the LayerSkip class of models, showing that pretraining with an early-exit loss yields better draft models.

4. External Candidate Draft Pool

We present experimental results where the candidate draft models are external to the target model. Building upon (Chen et al., 2023b), we evaluate our approach using the T5 model family (Raffel et al., 2020), specifically using T5-XXL (11B parameters) as the target. For candidate models, we employ the set of smaller T5 variants and a bigram model. As in (Chen et al., 2023b), we report Standardized Walltime Improvement (SWI), and generally follow their provided setup.

While (Chen et al., 2023b) relies on extensive hyperparameter sweeping to identify the optimal vertical and horizontal setup, we instead apply our optimization algorithm to determine the vertical hierarchy, extending it with the simple heuristic suggested by the authors and described in Appendix E. Notably, we find that **without any hyperparameter sweeping**, our combined approach achieves performance within 2% of the exhaustive sweep over both vertical and horizontal cascade parameters. We evaluate these results on GSM8K (Cobbe et al., 2021) and MMLU (Hendrycks et al., 2020).

Temp	Method	GSM8K (SWI)	MMLU (SWI)
1.0	Baseline	3.44	3.61
	HSD	3.68	4.06
	HSD+HH	3.70	4.16
0.6	Baseline	3.44	3.67
	HSD	3.67	4.10
	HSD+HH	3.69	4.17
0.0	Baseline	3.38	3.68
	HSD	3.76	4.10
	HSD+HH	3.79	4.13

Table 3. We evaluate over different temperatures on GSM8K and MMLU where the target model is the T5-XXL model. HSD represents the optimal vertical hierarchy, and HSD+Horizontal Heuristic (HSD + HH) represents using the heuristic suggested by (Chen et al., 2023b). In the case of MMLU and temperature 1.0, using HSD + HH accelerates inference by 15% over standard speculative decoding.

We further extend this analysis to varying target model temperatures. Leveraging the flexibility of our algorithm, we perform rate collection across all models at different temperatures. Interestingly, we observe that for a target temperature of 0.6, a draft model temperature of 0.5 yields optimal results. These findings are summarized in Table 3, with full hyperparameter details provided in the Appendix E.

5. Conclusion

We introduce an algorithm which extends speculative decoding to a more general setting, involving multiple models of varying cost and accuracy. We show that, given the acceptance rates between models, the optimal hierarchy in Hierarchical Speculative Decoding (HSD) can be found efficiently via a polynomial-time algorithm. Empirically, we confirm that our theoretical insights hold in practice and yield an improvement upon standard speculative decoding.

Future work might explore how to integrate HSD with other speculative decoding techniques for the single-draft setting, such as (Gloeckle et al., 2024a; Cai et al., 2024; Miao et al., 2024). Furthermore, our method opens the door to consider different temperatures for different draft models in a hierarchy.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Aishwarya, P. S., Nair, P. A., Samaga, Y., Boyd, T., Kumar, S., Jain, P., and Netrapalli, P. Tandem transformers for inference efficient llms. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Ankner, Z., Parthasarathy, R., Nrusimha, A., Rinard, C., Ragan-Kelley, J., and Brandon, W. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

Bae, S., Fisch, A., Harutyunyan, H., Ji, Z., Kim, S., and Schuster, T. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=WwpYSOkkCt.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Charnes, A. and Raike, W. M. One-pass algorithms for some generalized network problems. *Operations Research*, 14 (5):914–924, 1966.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023a.

Chen, Z., Yang, X., Lin, J., Sun, C., Chen, Y., Chang, K. C.-C., and Huang, J. Cascade speculative drafting for even faster llm inference. *arXiv preprint arXiv:2312.11462*, 2023b.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Deng, Y. and Rush, A. Cascaded text generation with markov transformers. *Advances in Neural Information Processing Systems*, 33:170–181, 2020.

Dohan, D., Xu, W., Lewkowycz, A., Austin, J., Bieber, D., Lopes, R. G., Wu, Y., Michalewski, H., Saurous, R. A., Sohl-dickstein, J., Murphy, K., and Sutton, C. Language model cascades, 2022.

Elbayad, M., Gu, J., Grave, E., and Auli, M. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJg7KhVKPH.

Elhoushi, M., Shrivastava, A., Liskovich, D., Hosmer, B., Wasti, B., Lai, L., Mahmoud, A., Acun, B., Agarwal, S., Roman, A., Aly, A., Chen, B., and Wu, C.-J. LayerSkip: Enabling early exit inference and self-speculative decoding. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024a.

Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024b.

Gupta, N., Narasimhan, H., Jitkrittum, W., Rawat, A. S., Menon, A. K., and Kumar, S. Language model cascades: Token-level uncertainty and beyond, 2024.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hochbaum, D. S. and Naor, J. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, 1994.

Hooper, C., Kim, S., Mohammadzadeh, H., Genc, H., Keutzer, K., Gholami, A., and Shao, S. Speed: Speculative pipelined execution for efficient decoding. *arXiv preprint arXiv:2310.12072*, 2023.

Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., and Keutzer, K. Speculative decoding with big little decoder. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 39236–39256. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/7b97adeafa1c51cf65263459ca9d0d7c-Paper-Conference.pdf.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Liu, T., Li, Y., Lv, Q., Liu, K., Zhu, J., and Hu, W. Parallel speculative decoding with adaptive draft length. *arXiv preprint arXiv:2408.11850*, 2024.

Liu, X., Hu, L., Bailis, P., Cheung, A., Deng, Z., Stoica, I., and Zhang, H. Online speculative decoding. *arXiv preprint arXiv:2310.07177*, 2023.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

Narasimhan, H., Jitkrittum, W., Rawat, A. S., Kim, S., Gupta, N., Menon, A. K., and Kumar, S. Faster cascades via speculative decoding, 2024.

Narasimhan, H., Jitkrittum, W., Rawat, A. S., Kim, S., Gupta, N., Menon, A. K., and Kumar, S. Faster cascades via speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vo9t20wsmd.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

Oldham, J. D. Combinatorial approximation algorithms for generalized flow problems. *Journal of Algorithms*, 38(1): 135–169, 2001.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V. Q., Tay, Y., and Metzler, D. Confident adaptive language modeling. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.

Sun, H., Chen, Z., Yang, X., Tian, Y., and Chen, B. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

Sun, Z., Mendlovic, U., Leviathan, Y., Aharoni, A., Ro, J. H., Beirami, A., and Suresh, A. T. Block verification accelerates speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=frsg32u0rO.

Team, G., Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Wayne, K. D. *Generalized maximum flow algorithms*. Cornell University, 1999.

Wayne, K. D. and Fleischer, L. Faster approximation algorithms for generalized flow. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 981–982, 1999.

Xiao, Z., Zhang, H., Ge, T., Ouyang, S., Ordonez, V., and Yu, D. Parallelspec: Parallel drafter for efficient speculative decoding. *arXiv preprint arXiv:2410.05589*, 2024.

Zhang, J., Wang, J., Li, H., Shou, L., Chen, K., Chen, G., and Mehrotra, S. Draft & verify: Lossless large language model acceleration via self-speculative decoding, 2024. URL https://arxiv.org/abs/2309.08168.

Zhou, Y., Lyu, K., Rawat, A. S., Menon, A. K., Rostamizadeh, A., Kumar, S., Kagy, J.-F., and Agarwal, R. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*, 2023.

Contents

A. Related work

Hierarchical models. Other uses of model hierarchies, ordered from weakest and cheapest to most capable and expensive, have demonstrated promising potential. One related domain is cascade models (Deng & Rush, 2020; Dohan et al., 2022; Gupta et al., 2024; Narasimhan et al., 2024) where typically the decision whether to use a larger model is based on a confidence measure over the prediction of the smaller model. Early exits in language models (Bae et al., 2025; Elbayad et al., 2020; Schuster et al., 2022) can be viewed as a form of cascades that are nested within a single model. (Narasimhan et al., 2024) use a speculative decoding technique to perform deferral in a two-model cascade. While these methods can provide promising speedups with quality guarantees in expectation, in contrast to speculative decoding, they do not give a per-example guarantee on the output distribution.

B. Proofs

B.1. Proof of Theorem 1.1

Proof. We give a proof by induction. We show that for all $i \in [K]$, Algorithm 2 returns the output distribution of $\mathcal{M}_i$.

The base case in the context of this proof is when $idx = 1$, and $\mathcal{M}_1$ verifies the draft tokens obtained from $\mathcal{M}_0$. Due to the verification rule, all verified tokens in *out* follow the distribution of $\mathcal{M}_1$. Furthermore, by outputting the token probabilities obtained directly $\mathcal{M}_1$, we also have the correct probabilities over next-tokens.

Suppose the inductive step holds for all values of $idx \leq k$. Next, we consider the case in which $idx = k + 1$. Then, the function call to Algorithm 2 with $idx = k$ correctly returns output tokens and their distributions according to the true distribution of $\mathcal{M}_k$. Hence, when $\mathcal{M}_{k+1}$ performs verification of tokens and replaces the token probabilities with its own, we obtain an output token distribution according to that of $\mathcal{M}_{k+1}$.

Hence, it follows that for $idx = K$, the output distribution over tokens is guaranteed to follow the distribution of $\mathcal{M}_K$. □

B.2. Proof of Theorem Correctness of HSD

Proof. We give a proof by induction over the value of idx given to Algorithm 2. In the base case, $idx = 0$. The cost of the algorithm in this case is simply $T_0 c_0$. The inductive hypothesis states that for all $k < K - 1$, the cost of Algorithm 2 with $idx = k$ is $\sum_{i=0}^{k} c_i \prod_{j=1}^{k} R(\alpha_{j-1,j}, j)$. Consider now the case where $idx = k + 1$. According to the function description, while T_{k+1} tokens have not been accepted, further tokens will be requested from $\mathcal{M}_k$ via function calls of the algorithm with $idx = k$. The expected number of such rounds is $\gamma(\alpha_{k,k+1}, T_k, T_{k+1})$. By the inductive hypothesis, in expectation, each of these rounds takes time $\sum_{i=0}^{k} c_i \prod_{j=i}^{k} R(\alpha_{j-1,j}, j) + c_{k+1}$. The additional cost of c_{k+1} is incurred due to verification. The expected cost at $idx = k + 1$ is thus:

$$
\begin{aligned}
&R(\alpha_{k,k+1}, k+1)\left(\sum_{i=0}^{k} c_i \prod_{j=i}^{k} R(\alpha_{j-1,j}, j) + c_{k+1}\right) \\
&= \sum_{i=0}^{k+1} c_i \prod_{j=i}^{k+1} R(\alpha_{j-1,j}, j).
\end{aligned}
$$

Hence, the inductive hypothesis holds for all $k < K$. If $idx = K$, we must instead divide by the expected number of generated tokens in order to obtain the latency. This is because $\mathcal{M}_K$ verifies all tokens from $\mathcal{M}_{K-1}$ and outputs those which it accepts. As shown in Leviathan et al. (2023), the expected number of tokens generated from $\mathcal{M}_K$ is $(1 - \alpha_{K-1,K}^{T_{K-1}+1})/(1 - \alpha_{K-1,K})$. □

B.3. Subsets of models

We present Corollary B.1, a natural extension of Theorem 1.1 that is useful for discussing the latency of a subset of models, rather than the entire set of models. The proof follows from that of Theorem 1.1.

Corollary B.1. *Given models $\{\mathcal{M}_i\}_{i=0}^{K}$, an ordered subset $\sigma \subseteq [K]$ of model indices with $|\sigma| \geq 2$ and final element K, and parameters $T = \{T_0, \ldots, T_{|\sigma|-1}\}$, the expected latency of HSD using $\{\mathcal{M}_i\}_{i \in \sigma}$ is:*

$$
L(\sigma, T) = \sum_{i=0}^{|\sigma|} c_{\sigma[i]} \prod_{j=i}^{|\sigma|} R_{\sigma,T}(\alpha_{\sigma[j-1],\sigma[j]}, j),
$$

where $R_{\sigma,T} : [0,1] \times [|\sigma|] \to \mathbb{R}$ is defined as:

$$
R_{\sigma,T}(\alpha, n) = \begin{cases} (1-\alpha)/\left(1-\alpha^{T_{|\sigma|-1}+1}\right) & \text{if } n = |\sigma| \\ \gamma(\alpha, T_{n-1}, T_n) & \text{if } 1 \leq n < |\sigma| \\ T_0 & \text{if } n = 0. \end{cases}
$$

B.4. Proof of Theorem 2.7

Proof. In the reduction from HSD to GSP, the number of vertices is $O(\overline{T}K)$ and the number of edges is $O(\overline{T}^2K^2)$. The time to create the GSP instance is thus $O(\overline{T}^2K^2)$. While GSP can be solved using a linear program, significant work (Wayne, 1999; Charnes & Raike, 1966; Wayne & Fleischer, 1999; Hochbaum & Naor, 1994) has been undertaken to reduce the running time. In particular, Oldham (2001) gives a strongly polynomial time algorithm: a GSP instance with n vertices and m edges can be solved in $O(mn^2 \log n)$. Consequently, HSD can be solved in $O(\overline{T}^4K^4 \log(\overline{T}K))$. □

B.5. Proof of Theorem 2.6

Proof. The proof relies on showing a bijection between augmented paths in the GSP instance and hierarchies with parameters in the HSD instance. Recall that all optimal solutions to GSP are augmented paths. Hence, upon solving GSP and decoding the solution into a path, we obtain a set of vertices along a simple path terminating at a lossy cycle. Define P as the set of all paths in G which start at s and terminate at a lossy cycle. By construction, all lossy cycles have zero cost and the objective can be re-written as a minimization over paths that terminate at a loop vertex:

$$\begin{aligned}\min \sum_{e\in E} f(e)c(e) &= \min_{p=(e_1,\ldots,e_{|p|})\in P} \sum_{i=1}^{|p|} f(e_i)c(e_i) \\ &= \min_{p=(e_1,\ldots,e_{|p|})\in P} \sum_{i=1}^{|p|} \left(\prod_{j=1}^{i-1} \mu(e_j)\right) c(e_i).\end{aligned}$$

Any fixed augmented path p in the graph is of the following form ($\ell \geq 0$):

$$p = (\mathcal{M}_K) \to (\mathcal{M}_{p_1}, t_{p_1}) \to \cdots \to (\mathcal{M}_{p_\ell}, t_{p_\ell}) \to (\mathcal{M}_{p_{\ell+1}}, t_{p_{\ell+1}}) \to (\mathcal{M}_{p_{\ell+1}}, L) \circlearrowleft .$$

The corresponding set of models in HSD is $\mathcal{M}_{p_{\ell+1}}, \mathcal{M}_{p_\ell}, \ldots, \mathcal{M}_{p_1}, \mathcal{M}_K$ and the corresponding T parameters are $j_{p_{\ell+1}}, j_{p_\ell}, \ldots, j_{p_1}$. Denote $\sigma = \{p_{\ell+1}, p_\ell, \ldots, p_1, K\}$. Substituting in the values from μ and c, the cost of this path in the GSP instance is:

$$\begin{aligned}&\frac{(1-\alpha_{p_1,K})}{(1-\alpha_{p_1,K}^{t_{p_1}})}c_K + \frac{(1-\alpha_{p_1,K})}{(1-\alpha_{p_1,K}^{t_{p_1}})}\gamma(\alpha_{p_2,p_1}, t_{p_2}, t_{p_1})c_{p_1} + \frac{(1-\alpha_{p_1,K})}{(1-\alpha_{p_1,K}^{t_{p_1}})}\gamma(\alpha_{p_2,p_1}, t_{p_2}, t_{p_1})\gamma(\alpha_{p_3,p_2}, t_{p_3}, t_{p_2})c_{p_2} + \cdots \\ &= \sum_{i=0}^{|\sigma|} c_{\sigma[i]} \prod_{j=i}^{|\sigma|} R_{\sigma,T}(\alpha_{\sigma[j-1],\sigma[j]}, j) = L(\sigma, T).\end{aligned}$$

Hence, the cost of a path in the GSP reduction is equal to the latency of the hierarchy which it specifies. By construction, every possible hierarchy is encoded as an augmented path in the GSP reduction; this is because any model can terminate the augmented path due to its designated lossy cycle vertex. Furthermore, every augmented path in the graph corresponds to exactly one subset σ and T parameters. Thus, there exists a bijection between augmented paths in the GSP instance and hierarchies in the HSD instance. Because the cost of a path in GSP exactly corresponds to the latency of that hierarchy, a path is a solution to the GSP instance if and only if the corresponding hierarchy is a solution to HSD. □

C. Sections 2 and 3 Details

C.1. Algorithms

First, we provide the algorithm description for verification.

Algorithm 1 Token Verification and Correction

Input: Index idx, Draft tokens $\mathbf{x} = (x_1, \ldots, x_t)$, Draft probs $\mathbf{q} = (q_1, \ldots, q_t)$, Context C
Output: output_tokens, output_probs
$t \leftarrow \text{length}(\mathbf{x})$
▷ *Run verifier $\mathcal{M}_{idx}$ in parallel on all prefixes*
$p_1, \ldots, p_{t+1} \leftarrow \mathcal{M}_{\text{idx}}(C), \ldots, \mathcal{M}_{\text{idx}}(C + x_1 \ldots x_t)$
$n \leftarrow t$ ▷ *Initialize accepted count to max*
for $i = 1$ **to** t **do**
  Sample $r \sim U(0, 1)$
  if $r > p_i(x_i)/q_i(x_i)$ **then**
▷ *Rejection sampling*
    $n \leftarrow i - 1$
    break
  end if
end for
accepted_tokens $\leftarrow (x_1, \ldots, x_n)$
final_dist $\leftarrow p_{n+1}$
if $n < t$ **then**
▷ *If token rejected, modify distribution*
  final_dist$(x) \leftarrow \text{NORMALIZE}(\max\{0, p_{n+1}(x) - q_{n+1}(x)\}) \quad \forall x$
end if
Sample $m \sim$ final_dist ▷ *Sample corrected token*
output_tokens $\leftarrow$ accepted_tokens $+ [m]$
output_probs $\leftarrow (p_1, \ldots, p_n, p_{n+1})$
return output_tokens, output_probs

This verification algorithm is exactly the same as that proposed in (Leviathan et al., 2023).

We also provide the exact details of the main algorithm.

C.2. Examining the assumptions of HSD

We conduct an ablation study to evaluate the impact of two simplifying assumptions made in our theoretical analysis: (1) that acceptance rates are IID, and (2) that generation and verification costs remain constant throughout inference. We use a four-layer hierarchy with Gemma2 9B to introduce more variability than the settings in our main results.

In order to assess the the validity of the first assumption, we simulate IID acceptance rates. To that end, we replace the verification rule in Algorithm 1 with a biased coin toss for each token. The probability of acceptance is set to the empirical average rate. In order to assess the validity of the second assumption, we simulate a constant cost. We substitute the measured wall-clock time at each step with a fixed, artificial cost, and the total latency is the sum of these costs. Thus, we measure latency across four settings and report the results in Table 4.

Real Acceptance / Real Cost The standard setting, which uses the true acceptances from Algorithm 1 and measures actual wall-clock time.

IID Acceptance / Real Cost We use simulated acceptances but measure actual wall-clock time.

Real Acceptance / Artificial Cost We use the true acceptances but measure a fixed, artificial cost per step.

IID Acceptance / Artificial Cost This represents the fully simplified model, using both simulated acceptances and fixed costs.

Algorithm 2 Hierarchical Speculative Decoding (HSD)

Require: Model index $k \in [0, K]$, Context C
Require: Models $\{\mathcal{M}_i\}$, Budgets $\{T_i\}$
if $k = 0$ **then**
  // Base case: Standard autoregressive generation
  Return $\mathcal{M}_0$.Generate(C, length $= T_0$)
end if
tokens $\leftarrow [\,]$, probs $\leftarrow [\,]$
while len(tokens) $< T_k$ **do**
  // Recursively get drafts from smaller model
  drafts, d_probs $\leftarrow$ HSD($k-1, C +$ tokens)
  // Verify drafts with current model
  valid, v_probs $\leftarrow$ VERIFY($\mathcal{M}_k$, drafts, $C +$ tokens)
  Append valid, v_probs to tokens, probs
  if $k = K$ **then**
    break
  end if
end while
Return tokens, probs

Acceptance Rate	Cost Type	Latency
Real	Real	0.0438226
IID	Real	0.0437504
Real	Artificial	0.0439264
IID	Artificial	0.0438760

Table 4. Comparison of latency under different conditions.

As shown, there is very little variability between all of these settings. We ran many experiments of this nature in order to both validate our assumptions and also verify that our algorithm was indeed running correctly. Hence, we conclude that the assumptions made by our theoretical work are not too strong to capture the empirical aspects.

C.3. HSD Example

We expand further on the example provided in Table 1. This example was constructed manually. We constructed this example in order to convey a setting in which adding more models improves the latency of HSD. In our example, we add one more model at a time by adding a new smallest model. Then, we solve for the optimal hierarchy. In our example, every time a new model is added as an option, it is optimal to use it in HSD.

We note the acceptance rate matrix must follow a certain structure. This is because acceptance rates are obtained via the TV distance of distributions, a distance metric that respects the triangle inequality. This implies that for any $i \neq j \neq k$, the following must hold:

$$\alpha_{i,j} + \alpha_{j,k} \leq \alpha_{i,k} + 1.$$

We create an acceptance rate matrix as shown in Table 5, where the acceptance rate from $\mathcal{M}_i$ to model $\mathcal{M}_j$ is in the i'th row and j'th column.

Table 5. First example acceptance rate matrix.

	2	**3**	**4**	**5**	**6**
1	0.750	0.500	0.250	0.000	0.000
2	–	0.750	0.500	0.250	0.050
3	–	–	0.750	0.500	0.300
4	–	–	–	0.750	0.550
5	–	–	–	–	0.800

We use the following costs: $c_1 = 0.00001, c_2 = 0.003, c_3 = 0.01, c_4 = 0.25, c_5 = 4, c_6 = 33$. While increasing the number of available models, we run the GSP solver to identify the optimal hierarchy to provide to HSD, and compute the expected latency.

We can instantiate many other such examples simply by changing the costs and acceptance rates. Suppose we let the costs be $c_1 = 0.00005, c_2 = 0.0002, c_3 = 0.05, c_4 = 2.0, c_5 = 8.0, c_6 = 33.0$ and let the acceptance rate matrix be as in Table 6. Then, adding more models yields speedup as shown in Table 7.

Table 6. Second example acceptance rate matrix.

	2	**3**	**4**	**5**	**6**
1	0.525	0.125	0.000	0.000	0.000
2	–	0.600	0.275	0.025	0.000
3	–	–	0.675	0.425	0.225
4	–	–	–	0.750	0.550
5	–	–	–	–	0.800

C.4. Analysis of optimal HSD configurations

Due to the introduction of a new optimization problem for each hierarchy, it is difficult to straightforwardly quantify when introducing a new model would lower the latency. However, we conduct an experiment to explore this. In the experiment, we fix a target model A with cost 1024, and a draft model B with cost 256. We fix the acceptance rate between the two to be 50%. Then, we introduce a third model, C, where we vary both its cost and its acceptance rate to B to identify settings in which it is optimal to use the hierarchy A-B-C. We use a lower bound to determine the acceptance rate from C to A. For each configuration, we run our optimization algorithm to identify the optimal hierarchy to minimize latency.

Number of Models	Expected Speedup	Expected Latency
1	1.0000×	33.00
2	1.7090×	19.31
3	2.1366×	15.45
4	2.2587×	14.61
5	2.2817×	14.46
6	2.2910×	14.40

Table 7. A second example of the expected speedup as the number of models provided to HSD increases.

We present our findings in Table 8. For each choice of cost for model C and acceptance rate from model C to B, we solve for the optimal latency. We color-code the cell based on which hierarchy achieves this latency.

As we can see, as the cost of C increases, it is less appealing to use it unless it also has a strong acceptance rate to B. When C has both a low cost and high acceptance rate to A, it eventually becomes optimal only to use model C. In between these scenarios, we see numerous instances in which the complete hierarchy A-B-C is the optimal one to use.

Acceptance Rate	Cost of Model C							
(C to B)	1.0	2.0	4.0	8.0	16.0	32.0	64.0	128.0
0.0	1.20	1.20	1.20	1.20	1.20	1.20	1.20	1.20
0.1	1.20	1.20	1.20	1.20	1.20	1.20	1.20	1.20
0.2	1.21	1.21	1.20	1.20	1.20	1.20	1.20	1.20
0.3	1.23	1.23	1.22	1.22	1.21	1.20	1.20	1.20
0.4	1.25	1.25	1.24	1.24	1.23	1.21	1.20	1.20
0.5	1.27	1.27	1.27	1.26	1.25	1.23	1.20	1.20
0.6	1.30	1.29	1.29	1.28	1.27	1.25	1.22	1.20
0.7	1.34	1.33	1.33	1.32	1.30	1.28	1.24	1.20
0.8	1.42	1.41	1.40	1.38	1.35	1.31	1.27	1.21
0.9	1.65	1.64	1.63	1.60	1.55	1.48	1.39	1.25

Hierarchy A-B-C is optimal. Hierarchy A-B is optimal. Hierarchy A-C is optimal.

Table 8. Speedup from optimal hierarchy across various parameters.

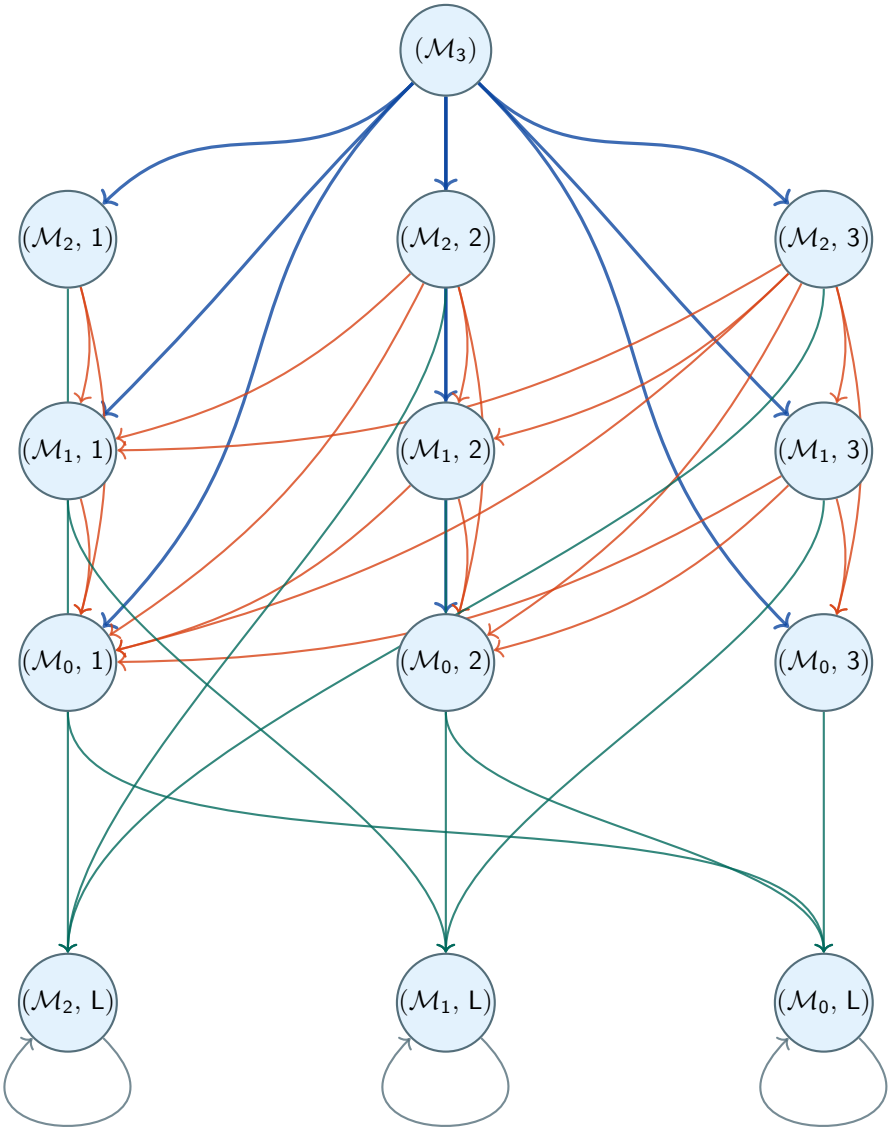
Edge (u, v)	**Multiplier** $\mu(u, v)$	**Cost** $c(u, v)$
$(\mathcal{M}_K), (\mathcal{M}_i, j)$	$\frac{1-\alpha_{i,K}}{1-\alpha_{i,K}^{j+1}}$	$\frac{1-\alpha_{i,K}}{1-\alpha_{i,K}^{j+1}} c_K$
$(\mathcal{M}_i, j), (\mathcal{M}_k, \ell)$	$\gamma(\alpha_{k,i}, \ell, j)$	$\gamma(\alpha_{k,i}, \ell, j)\, c_i$
$(\mathcal{M}_i, j), (\mathcal{M}_i, L)$	1	$j\, c_i$
$(\mathcal{M}_i, L), (\mathcal{M}_i, L)$	$\frac{1}{2}$	0

Table 9. Costs and multipliers for different graph edges.

D. Reduction from HSD to GSP

We provide the table of costs and multipliers for different graph edges.

In the following example, we draw the graph of the reduction for when $K = 3$ and $\bar{T} = 3$. The source vertex is $(\mathcal{M}_3)$ and the functions μ and c are as defined in Section 4. By finding the cheapest flow-conserving path which takes one unit of flow out of the source vertex, we also find the optimal speculative decoding hierarchy.


$(\mathcal{M}_3)$
$(\mathcal{M}_2, 1)$
$(\mathcal{M}_2, 2)$
$(\mathcal{M}_2, 3)$
$(\mathcal{M}_1, 1)$
$(\mathcal{M}_1, 2)$
$(\mathcal{M}_1, 3)$
$(\mathcal{M}_0, 1)$
$(\mathcal{M}_0, 2)$
$(\mathcal{M}_0, 3)$
$(\mathcal{M}_2, \mathrm{L})$
$(\mathcal{M}_1, \mathrm{L})$
$(\mathcal{M}_0, \mathrm{L})$

Dataset	Target Temp	Method	SWI	Draft Models	Draft Model Temp	T
GSM8K	1.0	Baseline	3.44	Base	1.0	10
		HSD	3.68	Base, MAG	1.0	11, 13
		HSD+HH	3.70	Base, MAG	1.0	11, 10
	0.6	Baseline	3.44	Base	0.6	10
		HSD	3.67	Base, MAG	0.5	12, 9
		HSD+HH	3.69	Base, MAG	0.5	12, 10
	0.0	Baseline	3.38	Base	0.0	10
		HSD	3.76	Base, Small, MAG	0.0, 0.0	12, 3, 11
		HSD+HH	3.79	Base, Small, MAG	0.0, 0.0	12, 3, 10
MMLU	1.0	Baseline	3.61	Base	1.0	11
		HSD	4.06	Base, MAG	1.0	13, 11
		HSD+HH	4.16	Base, MAG	1.0	13, 10
	0.6	Baseline	3.67	Small	0.6	14
		HSD	4.10	Base, MAG	0.5	10, 15
		HSD+HH	4.17	Base, MAG	0.5	10, 10
	0.0	Baseline	3.68	Small	1.0	19
		HSD	4.10	Base, MAG	0.0, 0.0	15, 22
		HSD+HH	4.13	Base, MAG	0.0, 0.0	15, 10

Table 10. Hyperparameters for T5 experiments.

E. Experimental Details

For full reproducibility, we report hyperparameters used to produce the results from the table.

Horizontal heuristic The horizontal heuristic involves appending 10 tokens from the smallest model at every vertical level. This heuristic was suggested by (Chen et al., 2023b), and we do not sweep over it.