SPECULATIVE SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive decoding is bottlenecked by its *sequential* nature. Speculative decoding has become a standard way to accelerate inference by using a fast *draft model* to predict upcoming tokens from a slower *target model*, and then verifying them *in parallel* with a single target model forward pass. However, speculative decoding itself relies on a *sequential* dependence between speculation and verification. We introduce *speculative speculative decoding* (SSD) to parallelize these operations. While a verification is ongoing, the draft model *predicts* likely verification outcomes and prepares speculations pre-emptively for them. If the actual verification outcome is then in the predicted set, a speculation can be returned immediately, thereby elimination all speculation overhead. We identify three key challenges presented by speculative speculative decoding, and put forth principled methods to solve each after theoretical analysis. The result is SAGUARO, an optimized SSD algorithm which is up to twice as fast as optimized speculative decoding baselines and up to 5x faster than autoregressive decoding with open source inference engines.

1 INTRODUCTION

Modern AI relies on fast language model inference. Applications ranging from chat assistants to coding agents and more depend on low-latency for interactive user experiences. However, the inherently *sequential* nature of autoregressive LLM decoding makes it challenging to attain low latency on modern GPUs, which rely on huge amounts of *parallel* compute for speed.

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) (SD) is a technique introduced to alleviate this problem. It uses a fast "draft model" to predict the next few tokens that would be generated by a slower "target model". These tokens are then "verified" in one *parallel* forward pass of the target model, according to an algorithm that guarantees the resulting tokens are drawn from the target distribution. In each verification, the target model decides both how many speculated tokens to accept, *and* samples an additional *bonus token* that follows all of the accepted tokens. This method exploits the key fact that verifying many tokens *in parallel* is much faster than generating them *sequentially*. Although speculative decoding is an effective technique for accelerating LLM decoding, it is *itself* limited by a *sequential* dependence: verification must complete before the next speculation can begin. In this work, we ask:

Can we eliminate the sequential dependence between drafting and verification?

We introduce *speculative speculative decoding* (SSD), a unifying framework for methods that aim to parallelize drafting and verification. While in SD the draft model *waits* for the verification to complete before beginning to speculate the next round, in SSD the draft model *predicts* what verification outcomes are most likely, and prepares (pre-speculates) for all of them *in parallel* to the verification. If any of these outcomes occurs, the draft model can *immediately* send the pre-speculated tokens to the verifier, thereby introducing no drafting latency.

There are three main challenges. First: the draft model must accurately predict the outcome of a verification, which is hard because the space of possible outcomes is extremely large: the draft must correctly predict not just *how many* speculated tokens will be accepted, but also *which* bonus token will be sampled at the end for each of those cases. Second, we identify a subtle trade-off between the acceptance rate of the speculator and how well it is able to predict verification outcomes, which must be navigated carefully to maximize speedups. Third, the algorithm must have a fallback strategy to

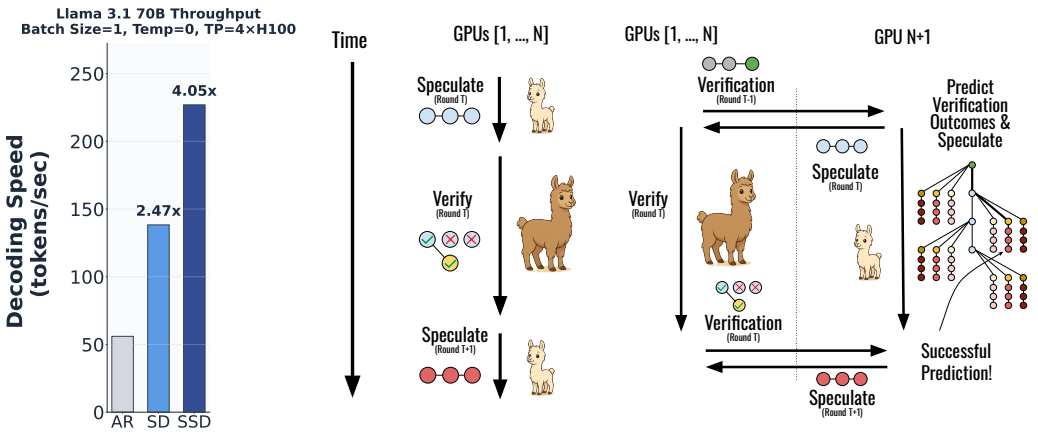

Figure 1: (Left) *End-to-end performance* of speculative speculative decoding (SSD), averaged over four datasets spanning math, code, instruction following, and chat. Comparison with autoregressive (AR) and ordinary speculative decoding (SD) baselines. (Middle) Ordinary speculation requires the verifier to wait idly for the draft to speculate. (Right) In our algorithm, speculation happens on distinct hardware, in parallel, *while verification is taking place*, based on *predicted verification outcomes*. This allows the latency of drafting to be hidden and allows the draft model to speculate more tokens, but risks a "cache miss" if the draft fails to predict the actual verification.

handle cases where the draft model fails to predict the verification outcome correctly, balancing its latency and acceptance rate. Such failures become increasingly common as batch size or sampling temperature increase, and thus necessitate careful treatment.

We present SAGUARO, an optimized SSD algorithm that addresses these three challenges. SAGUARO achieves decoding speedups of up to 2x compared to optimized speculative decoding baselines, and up to 5x that of standard autoregressive generation on a range of datasets and model families (Figure 8). It introduces optimizations to address the three challenges above:

- In Section 4.1, we frame the problem of predicting verification outcomes in terms of constrained optimization, and introduce a technique that uses the most likely draft logits to predict the bonus token, doing so with up to 90% accuracy.
- In Section 4.2, we identify a tension between accurately predicting verification outcomes and generating high-quality speculations, and develop a sampling algorithm that allows balancing the two. This results in 50% end-to-end speedups at high temperatures.
- In Section 4.3, we propose and examine various strategies to handle failed predictions, demonstrating that the optimal fallback strategy varies with batch size. Adopting this, SAGUARO outperforms SD by 60% at low batch sizes and 20% even at larger batch sizes, despite having to do much more compute per batch element by decoding many possible outcomes simultaneously.

2 BACKGROUND

2.1 SPECULATIVE DECODING

Speculative decoding accelerates sampling from a target distribution p_{target} by using a cheaper draft distribution p_{draft} to generate candidate continuations, and then using the target distribution to accept selectively. To verify, the target first computes probabilities for all the speculated tokens in parallel in *a single forward pass*. Drafted tokens are accepted sequentially, each with probability $\min\{1, p_{\text{target}}(x)/p_{\text{draft}}(x)\}$. Upon the first rejection, remaining draft tokens are discarded, and the last-token target logits are used to sample a "bonus token." This is done by sampling the bonus token from a modified distribution that guarantees the resulting sequence is distributed as p_{target}. This modified distribution is called the ***residual distribution***, and takes the form

$r(\cdot) \propto \max\left(p_{\text{target}}(\cdot) - p_{\text{draft}}(\cdot), 0\right)$. In the case where all tokens are accepted, the residual distribution is simply the target distribution.

The expected number of generated tokens per round, and thus the overall efficiency of speculative decoding, is governed by the ***acceptance rate***: the probability of accepting a given token in the speculation, conditioned on accepting all prior tokens. In (Leviathan et al., 2023), it is shown that the acceptance rate can be written in terms of how well the draft distribution approximates the target distribution in the following way.

Theorem 1. *(Leviathan et al., 2023)*

$$\alpha = \sum_x \min\{p_{target}(x),\ p_{draft}(x)\} = 1 - \tfrac{1}{2}\|p_{target} - p_{draft}\|_1$$

Definition 2. *We define a **speculation** at round T, denoted $s^T \coloneqq (s_1^T, \ldots, s_K^T)$, as the sequence of K tokens proposed autoregressively by the draft model. The length K of the speculated sequence is called the **speculative lookahead**.*

Definition 3. *We define a **verification outcome** at round T, denoted $v^T \coloneqq (k, t^*) \in \mathcal{V}^T$, where $(s_1^{T-1}, \ldots, s_k^{T-1})$ are the accepted draft tokens from round $T-1$, and t^* is the **bonus token** sampled from the residual distribution.*

2.2 Related Work

Parallel Speculative Decoding. AMUSD (McDanel, 2025) and PEARL (Liu et al., 2025) propose speculating the next round during ongoing verification, but only prepare for the verification outcome in which all tokens are accepted, which is one special case of many possible outcomes. SwiftSpec (Zhang et al., 2025) and SpecBranch (Shen et al., 2025) prepare a larger cache consisting of a token tree branching off of the speculation being verified (thus enabling larger speedups), but both use fallback strategies that do not work at large batch sizes. SpecBranch (Shen et al., 2025) has similar motivation to SSD: it constitutes (approximately) a special case of the SSD framework where only a single branching point is allowed, where the fallback speculator is equal to the regular speculator, and where the hyperparameters (branching point, number branches, speculation length) are dynamically chosen to maximize speedups. SwiftSpec (Zhang et al., 2025) considers the special case of greedy sampling, and adopts a fallback strategy of just-in-time speculation that struggles at higher temperatures and batch sizes when cache-misses become inevitable. SAGUARO sampling (Section 4.2) and fallback (Section 4.3) allow our techniques to perform better in these important regimes.

Tree-Based Speculative Decoding. Numerous speculative decoding methods have been proposed that increase the expected number of accepted tokens by allowing the draft model to speculate a *tree* of tokens instead of a sequence, thereby giving the verifier several token options at each position (Miao et al., 2024; Li et al., 2024b; Chen et al., 2024; Svirschevski et al., 2024). Our method differs in important ways: First, existing tree-based methods are still sequential: speculate, then verify. Second, tree-based methods introduce a large amount of *verifier compute*—which is quite expensive due to the size of the target model—because the entire tree must be verified in the target model forward pass. Our method, on the other hand, scales up the *speculation compute* by pre-speculating for many verification outcomes in parallel, but does not introduce any additional verifier compute. Lastly, it is important to note that our method can be combined with these tree-based approaches for further gains (see Appendix E for discussion).

Improved Draft Architectures. There have been many advancements in draft model architectures that can improve the acceptance rates and/or speeds of the draft model. For example, EAGLE (Li et al., 2024a;b; 2025b) allows the draft model to take as input the powerful representations from the target model for the current prefix, while GliDe (Du et al., 2024) and LongSpec (Yang et al., 2025) allow the draft model to perform cross-attention on the KV cache of the target model. Alternate model architectures, like diffusion LLMs (Nie et al., 2025; Sahoo et al., 2024; Li et al., 2025a; Christopher et al., 2025; Samragh et al., 2025) or SSMs (Gu et al., 2022; Gu & Dao, 2023; Wang et al., 2024), can also be used to increase the speed with which the draft model can produce the speculated token sequence. We include more technical details on how SSD can fruitfully be combined with these improved draft model architectures (e.g., EAGLE-3; Li et al. (2025b)) in Appendix E.

3 THE SPECULATIVE SPECULATIVE DECODING FRAMEWORK

We introduce *speculative speculative decoding*, a framework to reason about asynchronous variants of speculative decoding (Section 3.1), and present theoretical results comparing the expected speed of SSD to baselines (Section 3.2). **Definitions introduced here will be important in Section 4.**

3.1 ALGORITHM

Algorithm 1: The Speculative Speculative Decoding (SSD) Framework.

```
Function main(prompt, target, draft, backup_spec):
    asynchronously launch speculator(prompt, draft, backup_spec)
    generated_tokens ← verifier(prompt, target)
    return generated_tokens

Function verifier(prompt, target):
    target.prefill(prompt)
    WAIT TO RECEIVE spec_tokens from speculator
    generated_tokens ← []
    while True do
        verify_outcome ← target.verify(spec_tokens)
        generated_tokens.append(verify_outcome.tokens)
        SEND verify_outcome to speculator
        if end_token ∈ verify_outcome then
            return generated_tokens
        end
        WAIT TO RECEIVE spec_tokens from speculator
    end

Function speculator(prompt, primary_spec, backup_spec):
    spec_tokens ← draft.speculate(prompt)
    while True do
        SEND spec_tokens to verifier for verification
        cache_keys ← predict_verification_outcomes(spec_tokens) // Section 4.1
        cache_vals ← speculate_for_outcomes(cache_keys) // Section 4.2
        WAIT TO RECEIVE verify_outcome from verifier
        if verify_outcome ∈ cache then
            spec_tokens ← cache[verify_outcome]
        else
            spec_tokens ← fallback_speculate(
                verify_outcome, primary_spec, backup_spec
            ) // Section 4.3
        end
    end
```

We present the SSD framework in Algorithm 1. The speculator and verifier processes run in parallel on separate hardware. While the verifier is verifying the drafted tokens from round T, the speculator begins speculating round $T + 1$.

It does so by predicting likely verification outcomes, and then preparing speculations for each of these outcomes in parallel, and storing these in a "speculation cache" (defined formally below). Then, when it receives the actual verification outcome, it checks whether this was one of the outcomes in the cache that it had prepared for. If so, it immediately returns it. Otherwise it defers to a fallback speculation strategy.

Definition 4. *We define a* ***speculation cache*** $\mathcal{S}^T$ *as a dictionary mapping a set of verification outcomes* $\mathcal{V}^T$ *to their associated pre-computed speculations* $s^T = (s_1^T, \ldots, s_K^T)$*. We denote a speculated token sequence* s^T *corresponding to an outcome* $v^T \in \mathcal{V}^T$ *by* $\mathcal{S}^T(v^T)$ *as a cache lookup operation.*

Definition 5. *A **cache hit** is when the outcome v^T of verifying s^{T-1} is contained in the speculation cache $\mathcal{S}^T$. A **cache miss** is when $v^T \notin \mathcal{S}^T$.*

Definition 6. *Let $\mathbf{p_{hit,p}}$ denote the probability of a cache hit on an iteration conditional on the previous iteration having been speculated by the primary draft model. Analogously, $\mathbf{p_{hit,b}}$ represents the probability of a cache hit on an iteration conditional on speculating with the backup model. Finally, let $\mathbf{p}_{hit}$ denote the unconditional probability of a cache hit in an iteration.*

3.2 Theoretical Results

We analyze the performance of Saguaro relative to regular autoregressive decoding (Theorem 7) and sequential speculative decoding (Corollary 20).

Theorem 7. *Let the primary and backup speculators take time T_p and T_b relative to the verifier. Let the expected number of generated tokens from the primary speculator be E_{hit}, E_{miss}, respectively. The expected speedup of Algorithm 1 relative to autoregressive decoding is then:*

$$speedup \quad = \quad \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit} \cdot \max(1, T_p) + (1 - p_{hit}) \cdot (1 + T_b)}.$$

The numerator corresponds to the expected number of tokens generated in each iteration of the algorithm, and the denominator corresponds to the expected latency of each iteration relative to autoregressive decoding. This implies two key corollaries.

Corollary 8. ***(Strictly Faster Than SD)**. Suppose we run SD with a given speculator $\mathcal{M}$. Running SSD with primary and backup both set to $\mathcal{M}$ does no worse than SD (strictly better if $p_{hit}, T_{SD} > 0$).*

Corollary 9. ***(Speedup Sandwich)** Suppose we choose a primary speculator for which drafting completes before verification ($T_p < 1$), and a fast backup speculator ($T_b = 0$). Then if T_{SD}, E_{SD} represent the latency and expected number of generated tokens from a draft model in SD, then the SSD speedup over SD can be bounded by:*

$$\left(1 + T_{SD}\right) \cdot \frac{E_{hit}}{E_{SD}} \cdot p_{hit} \leq \frac{speedup_{SSD}}{speedup_{SD}} \leq \left(1 + T_{SD}\right) \cdot \frac{E_{hit}}{E_{SD}}.$$

This equation reveals that the maximum speedup attainable by SSD is proportional to the latency reduction $(1 + T_{\text{SD}})$ from hiding draft latency, and the increase in expected number of generated tokens $(E_{\text{hit}}/E_{\text{SD}})$ from increased drafting time. However, a low cache hit rate p_{hit} reduces the effectiveness of this algorithm, as shown by the lower bound.

4 Saguaro: An Optimized SSD Algorithm

In this section, we present Saguaro, our optimized instantiation of the SSD framework. We present the three core optimizations we introduce to attain them in Sections 4.1, 4.2, 4.3, respectively, then combine and evaluate them end to end.

Setup. Throughout this section, experiments are run on batch size 1 with greedy decoding unless otherwise specified, with the target model on 4×H100 and draft on 1×H100 (in SSD, but colocated with target in SD). We conduct experiments on two model families, Llama-3 and Qwen-3; the former appears the main text, with analogous plots for the latter in Appendix F. We document results across four standard datasets, Alpaca (Dubois et al., 2024), GSM8k (Cobbe et al., 2021), UltraFeedback (Cui et al., 2024), and HumanEval (Chen et al., 2021). Further setup details are in Appendix B.

4.1 Predicting Verification Outcomes: Building the Saguaro Cache

Given a speculation s^T that is in the process of being verified, Saguaro must build a cache $\mathcal{S}^T$ for the most likely verification outcomes. The difficulty is that the space of possible verification outcomes is vast, of size $(K+1)V$; there are $K+1$ possibilities for the number of accepted tokens, and $V-1$ possible bonus tokens (the rejected token in the speculation can never be the bonus, V possibilities if all tokens are accepted). While the draft decodes all anticipated speculations at once, in parallel, using a special attention mask (Appendix B), each branch being decoded adds to the compute-load. This motivates posing verification prediction as constrained optimization.

4.1.1 ALGORITHM

Definition 10. *We define the **fan-out** $F_k^p := \left|\{v^T := (k', t^*) \in \mathcal{S}^T \mid k' = k\}\right|$ at position k to be the number of bonus tokens the draft model anticipates at that position, given the previous iteration was speculated by the primary speculator. F_k^b is defined analogously for the backup speculator.*

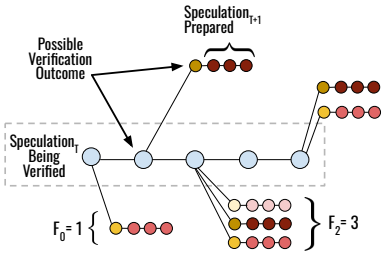

Figure 2: Schematic of speculation cache strategy. We allocate *fan-out* (bonus token guesses) over sequence length $K+1$ according to Theorem 12.

SAGUARO Verification Outcome Prediction Algorithm Given a fan-out strategy $\{F_k^p, F_k^b\}$ (ours based on Theorem 12), SAGUARO takes the top-F_k tokens at each lookahead position k in the draft logits, and adds those to the speculation cache. These are the top-F_k logits *excluding* the sampled token sent for verification at that position, which is *guaranteed* not to be the bonus token.

4.1.2 THEORETICAL RESULTS

We show how to optimize the choice of fan-out values F_k^p and F_k^b to maximize the speedup under a constraint on the size of the speculation cache.

The probability of a verification outcome $(k, \cdot)$ in a given iteration depends on whether the previous speculation was generated by the primary or backup speculator. Since the two have distinct acceptance rates, the probability a given number of tokens being accepted need not be the same between the two. Thus, if the default fan-out strategy when building the cache is to fan out uniformly at each sequence position, the cache hit rate on a given iteration depends on whether the previous iteration was speculated by the primary or backup speculate. These are the definitions of $p_{\text{hit,p}}(F)$ and $p_{\text{hit,b}}(F)$, respectively. We see in Figure 3 that these functions (in fact their complement, the rejection rate) empirically follow a power-law.

Definition 11.[1] *We say that a speculator has a* **r *power-law cache hit rate*** *if the chance of a cache miss with fan-out F is equal to a power-law of F with exponent r, for a sequence drafted by this speculator. More specifically, this means $1 - p_{hit,*}(F) = 1/F^r \;\; \forall F \in \mathbb{N}$, for some $r > 0$.*

We now use this definition to reason about how to optimally select the fan-out values F_k^p and F_k^b under a computational constraint on the size of the cache. We consider the general case, when this assumption doesn't hold, in Appendix A.

Theorem 12. ***(SAGUARO Cache Shape: Geometric Fan-Out)*** *Consider a draft model with a r power-law cache hit rate. Then the optimal choice of F_k^p (F_k^b) values for $k \in [0, K]$ under the constraint $\sum_{k=0}^{K} F_k^p \leq B$, follows a capped geometric series:*

$$\begin{aligned} F_k &= F_0 \cdot a^{k/(1+r)} \;\; \forall k < K, \text{ and} \\ F_K &= F_0 \cdot a^{K/(1+r)} \cdot (1-a)^{-1/(1+r)}, \end{aligned}$$

where F_0 can be chosen such that $\sum_{k=0}^{K} F_k^p = B$ (closed form-equation in Appendix A).

This result cleanly reflects the intuition that the lengths of verified strings follow a capped geometric distribution supported on the speculation lookahead. In other words, if it is unlikely that $j \in [0, K]$ tokens will be accepted by the verifier, we should not waste compute guessing and speculating on the bonus token at that position, and should lower F_j.

4.1.3 EMPIRICAL EVALUATION

We compare the end-to-end performance of using a naive (uniform) fan-out strategy over sequence length against the geometric fan-out strategy advanced by Theorem 12. In Figure 4 we find it improves cache hit rate (right) and end-to-end decoding speed, especially at higher temperatures, where both SD and the uniform fan out begin to flounder.

[1] This definition is closely related to the notion of b power-law acceptance rate in Chen et al. (2024).

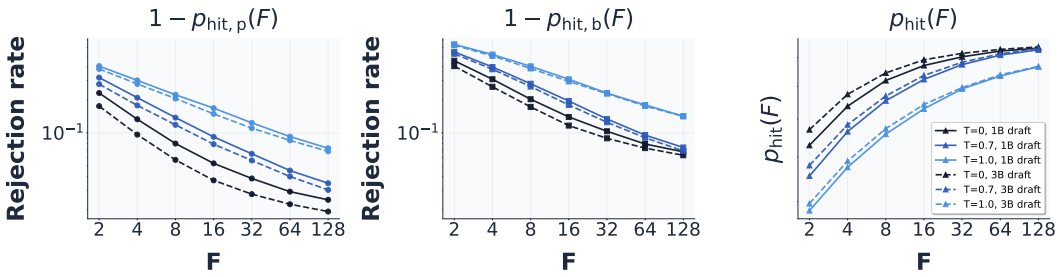

Figure 3: Scaling of cache hit rate with fan out. The overall cache hit rate $p_{\text{hit}}(F)$ (right) computed using Theorem 1. Rejection (cache miss) falls as a power law in draft-fan out, suggesting consistent increases in cache hit rate with larger cache sizes.

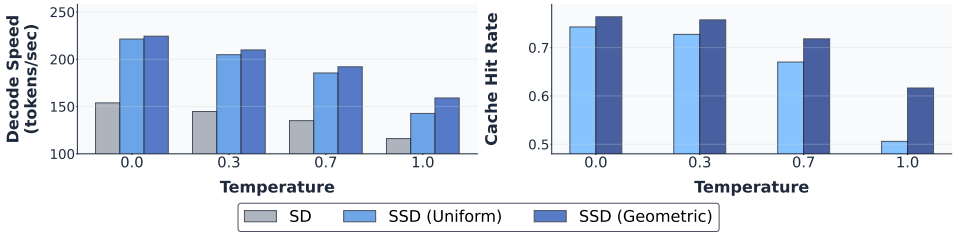

Figure 4: Advantage of geometric fan out strategy increases at higher temperatures, improving both speculation cache hit rate (right) and thus end-to-end speed (left). Results averaged over four datasets. At all temperatures, SSD with either fan out strategy outperforms ordinary speculative decoding.

4.2 Balancing Cache Hit and Acceptance Rate with Saguaro Sampling

The majority of the time, the bonus token is sampled from the *residual distribution* (except when all tokens are accepted, in which case it is sampled from the target directly). The residual distribution can be difficult to predict, especially as sampling temperatures increase. We introduce a novel sampling scheme that makes this residual easier to predict and therefore increases cache hit rates.

This exploits the fact that the residual distribution is a function of the draft distribution. We make the recovery token easier to predict by explicitly increasing the residual probability mass on the most likely draft tokens; and this is done by *decreasing* the corresponding probability mass when sampling from the draft. In biasing the draft distribution, however, we may decrease the acceptance rate by having moved the draft distribution farther from the target (see 1). This induces a tradeoff between the acceptance rate and the cache hit rate, both of which contribute to end-to-end speed (see Figure 5, left).

4.2.1 Algorithm

Definition 13. *We define a **sampling scheme** as a function $\sigma\colon \mathbb{R}^V \to \Delta^{V-1}$ from model logits $z_{draft} \in \mathbb{R}^V$ to a probability distribution $p \in \Delta^{V-1}$.*

To understand how to design a sampling scheme that increases the residual probability mass on the most likely draft tokens, we first note that the probability of token t in the residual distribution is proportional to $\max(p_{\text{target}}(t) - p_{\text{draft}}(t), 0)$. Thus, *decreasing* $p_{\text{draft}}(t)$ *increases* the token the probability of t in the residual. This is exactly what our cache-aware sampling scheme does.

Definition 14. *Given draft logits $z \in \mathbb{R}^V$, we define the* **Saguaro** ***sampling scheme*** *$\sigma_{F,C}(z)$ for fan-out F and downweighting constant $C \in [0, 1]$ as*

$$\sigma_{F,C}(z) \propto \begin{cases} C \cdot \exp\left(z_t\right) & \text{if } t \in \text{top}_F(z) \\ \exp\left(z_t\right) & \text{otherwise,} \end{cases}$$

In practice, C is a hyperparameter found empirically. The optimal choice $C^* \in [0, 1]$ varies with temperature (see Figure 6), and that $C \ll 1$ can be especially advantageous at high temperatures.

4.2.2 THEORETICAL RESULTS

Theorem 15. *For fan-out F and primary speculator logits z, the cache hit rate p_{hit} of the* SAGUARO *sampling scheme increases monotonically as $C \to 0$.*

The new sampling hyperparameter C allows a trade-off between cache hit rate and acceptance rate. We plot this tradeoff in Figure 5 (left). Figure 5 (right) presents an illustration of how SAGUARO sampling allows control of the *residual distribution* by manipulating the draft distribution during speculation. The intuition here is that SAGUARO sampling deliberately suppresses the draft probabilities on the F cached tokens. By downweighting $p_{\text{draft}}(t)$ on this set, the residual $\max\big(p_{\text{target}}(t) - p_{\text{draft}}(t),\, 0\big)$ is pushed to *concentrate* on those same tokens, *increasing* the chance that the bonus token lands inside the cache by construction. The ability to trade-off these two quantities becomes an important way to accelerate inference at high temperatures when cache hit rate falls quickly but acceptance rate only slowly.

4.2.3 EMPIRICAL EVALUATION

We show in Figure 5 how there is a trade-off between acceptance rate and cache hit rate which we can navigate by using the SAGUAROsampling scheme with different values of C. Lower values of C lead to to higher cache hit rates and lower acceptance rates as they bias the draft distribution away from the target to control the residual distribution. In Figure 6, we see that to maximize speedup at larger temperatures it is important to sacrifice some acceptance rate (by choosing $C \ll 1$) to attain higher cache hit rates.

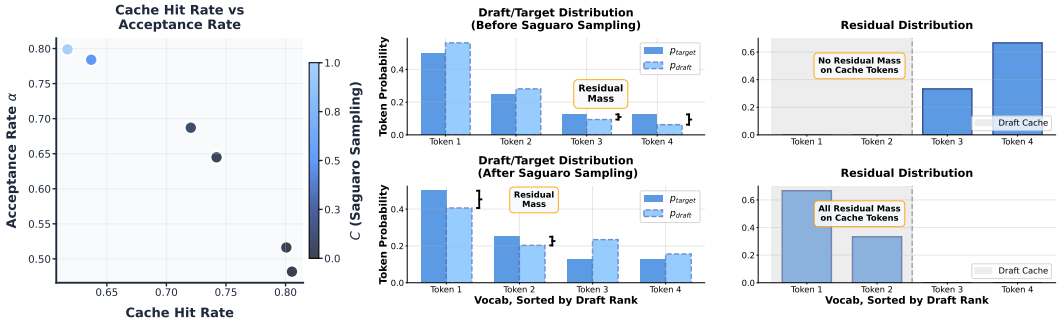

Figure 5: We introduce SAGUARO sampling, a novel sampling scheme designed specifically for SSD. (Left) It interpolates between high cache hit rate and high speculative acceptance rate. (Right) Illustrative schematic for how SAGUARO sampling increases residual probability mass on the top draft tokens, encouraging the sampled bonus token to lie in the speculation cache by construction.

4.3 HANDLING CACHE MISSES WITH SAGUARO FALLBACK

We now discuss how we optimize the handling of cache misses in SAGUARO, and propose an optimal strategy for picking the backup speculator based on the batch size.

4.3.1 ALGORITHM

We design SAGUARO's cache miss strategy based on the observation that cache misses occur *almost certainly* at large batch sizes, and that when this happens, in SSD the *whole batch* must wait for the backup speculator to complete before being verified. We propose the SAGUARO *fallback strategy*: set the backup speculator to be equal to the primary speculator at low batch size, then switch to a low-latency speculator (Oliaro et al., 2024; Liu et al., 2024b; Xu et al., 2025) for larger batch sizes $b > b^*$, where the critical batch size b^* is derived in the following section.

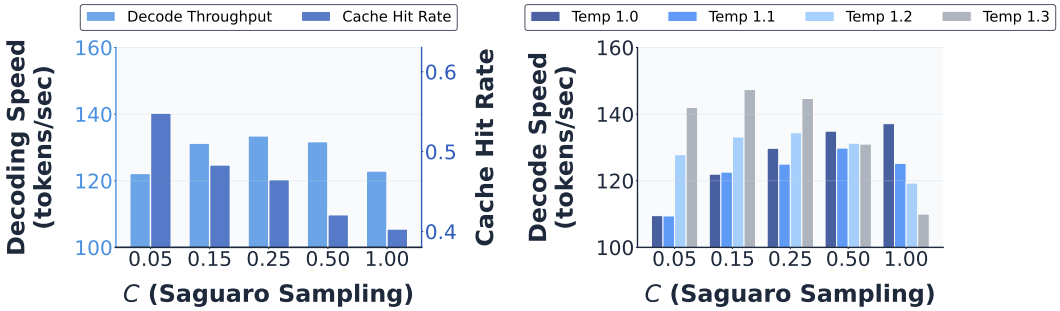

Figure 6: (Left) Tokens/sec and cache hit rate vs C, averaged over four temperatures in $[1, 1.4]$. While cache hit rate always increases in C (Theorem 15), end-to-end speed depends on *both* cache hit rate and acceptance rate. (Right) The optimal C^* depends on temperature: the naive default of $C = 1$ is *suboptimal* at higher temperatures.

4.3.2 Theoretical Results

We prove that Saguaro's backup speculator strategy is optimal, under the conditions that SSD uses a high-quality (slow) speculator (primary), and a lower-quality (fast) speculator (backup). We begin with a corollary to Theorem 7 which accounts for the impact of batch size on the expected speedup attained by SSD. This result assumes that in SSD the whole batch waits for the backup speculator to complete before verifying the batch.

Corollary 16. *At batch size b, the expected speedup from SSD is equal to:*

$$\textit{speedup} = \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit}^{b} \cdot \max(1, T_p) + (1 - p_{hit}^{b}) \cdot (1 + T_b)}, \quad \textit{which approaches}$$
$$\frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{1 + T_b} \quad \textit{as } b \to \infty.$$

In our implementation, the backup strategy is to return random tokens,[2] and the primary strategy is to do just-in-time speculation. As the batch size increases, the entire batch stalls on the latency of the backup speculator as cache misses happen more frequently. This forces the choice of a low latency speculator at larger batch sizes, as the following theorem makes precise.

Theorem 17. *The optimal cache miss strategy given two speculators of varying speed (primary and backup) is to set the backup equal to the primary for batch sizes $b < b^*$, and not otherwise. We solve for b^* in Appendix A.*

4.3.3 Empirical Evaluation

We show in Figure 7 that as the batch size increases, using a fast backup speculator that returns random tokens outperforms using a slow but more accurate neural speculator just-in-time.

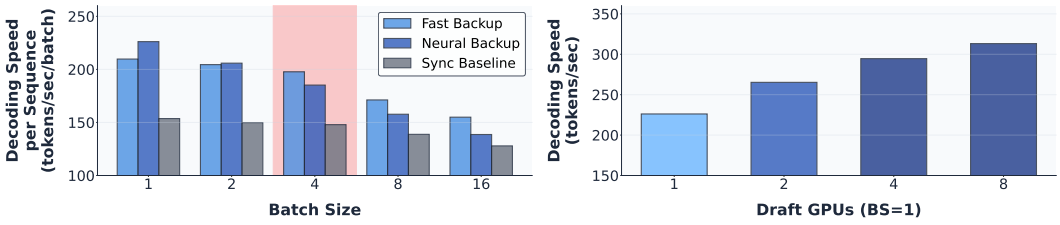

Figure 7: (Left) Optimal backup speculator (fast vs neural) depends on batch size, as Theorem 17 predicts. (Right) We forecast how cache hit rate/decoding speed improves with draft compute.

[2]Note that we can easily improve upon this random token strategy by using extremely fast non-neural speculators, like those based on n-grams (Liu et al., 2024b; Oliaro et al., 2024)

5 End-to-End Evaluation

In Figure 8 we compare the end-to-end performance of SAGUARO to key baselines. Our baselines are very strong: our implementation of ordinary speculative decoding (SD) is faster than that of SGLang with SD, which was the fastest of all the popular inference engines we tried. We attain over a 4x speedup on average, and are almost twice as fast as the optimized speculative decoding baselines, including EAGLE-3 (Li et al., 2025b).

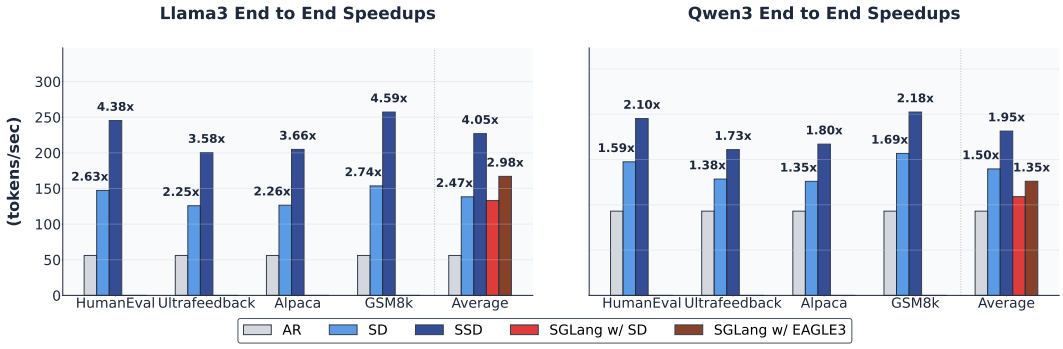

Figure 8: End-to-end decoding speed comparison of SSD compared to SD and standard autoregressive decoding across two model families and four datasets.

5.1 Conclusion and Limitations

Speculative decoding is an important technique for LLM inference acceleration because it allows trading off compute and latency, and since LLM inference is typically memory-bound, there is usually compute to spare. However, it requires drafting and verification to wait synchronously on each other. We take this trade-off to its natural conclusion, parallelizing even this sequential dependence, and reaping commensurate end-to-end speedups. We characterize the expected performance gains as well as performance upper bounds of our method, in addition to studying each major component of the parallel speculative decoding design space in a principled manner.

An important limitation of our method—as with all speculative decoding methods—is that it focuses on improving latency and not throughput. While the former is critical in many modern applications like interactive chat assistants and coding agents, the latter is important in settings like large-scale reinforcement learning or offline synthetic data generation. We note, however, that relative to token-tree verification methods, SSD is able to attain higher throughput by only increasing draft model compute (which is cheap), not verifier compute (which is expensive).

Looking forward, we hope to see our method *combined* with popular variants of speculative decoding like EAGLE (Li et al., 2024a;b; 2025b) or token tree speculation (Miao et al., 2024; Chen et al., 2024) for even larger gains, and for future work to study how to scale parallel speculative decoding techniques to large fleets of many communicating compute nodes, where considerations like prefill-decode disaggregation (Zhong et al., 2024) come to the fore.

REFERENCES

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020. URL https://arxiv.org/abs/2004.05150.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024. URL https://arxiv.org/abs/2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, 2024.

Krzysztof M. Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*.

Jacob K. Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 12042–12059. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.601. URL https://doi.org/10.18653/v1/2025.naacl-long.601.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with scaled ai feedback, 2024. URL https://arxiv.org/abs/2310.01377.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2205.14135.

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. Glide with a cape: A low-hassle method to accelerate speculative decoding, 2024. URL https://arxiv.org/abs/2402.02082.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback, 2024. URL https://arxiv.org/abs/2305.14387.

Yihe Fu, Zhiqiang Zhao, Wenyu Chen, Kai Chen, Zhizhen Zhao, Minghua Chen, and Chuan Wu. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024. URL https://arxiv.org/abs/2402.02057.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.01801.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv*, 2023. doi: 10.48550/arXiv.2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2401.18079.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. *arXiv preprint arXiv:2309.06180*, 2023a. URL https://arxiv.org/abs/2309.06180.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP)*, 2023b. doi: 10.48550/arXiv.2309.06180. URL https://arxiv.org/abs/2309.06180.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. Diffuspec: Unlocking diffusion language models for speculative decoding. *CoRR*, abs/2510.02358, 2025a. doi: 10.48550/ARXIV.2510.02358. URL https://doi.org/10.48550/arXiv.2510.02358.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025b. URL https://arxiv.org/abs/2503.01840.

Anyi Liu, Junrui Xiao, Jiaqi Guo, Wenming Yang, and Jiwen Lu. Minicache: Kv cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/fd0705710bf01b88a60a3d479ea341d9-Abstract-Conference.html.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *CoRR*, abs/2401.17377, 2024b.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length, 2025. URL https://arxiv.org/abs/2408.11850.

Bradley McDanel. AMUSD: asynchronous multi-device speculative decoding for LLM acceleration. In *IEEE International Symposium on Circuits and Systems, ISCAS 2025, London, United Kingdom, May 25-28, 2025*, 2025.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Proceedings of the Thirty-Ninth Conference on Neural Information Processing Systems (NeurIPS 2025)*, 2025. URL https://arxiv.org/abs/2502.09992.

Gabriele Oliaro, Zhihao Jia, Daniel F. Campos, and Aurick Qiao. Suffixdecoding: A model-free approach to speeding up large language model inference. *CoRR*, abs/2411.04975, 2024.

Subham Sekhar Sahoo, Marianne Arriola, Aaron Gokaslan, Edgar Mariano Marroquin, Alexander M Rush, Yair Schiff, Justin T Chiu, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=L4uaAR4ArM.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your LLM knows the future: Uncovering its multi-token prediction potential. *CoRR*, abs/2507.11851, 2025. doi: 10.48550/ARXIV.2507.11851. URL https://doi.org/10.48550/arXiv.2507.11851.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *NeurIPS 2024 (Spotlight Poster)*, 2024. URL https://arxiv.org/abs/2407.08608. Spotlight poster version.

Yuhao Shen, Junyi Shen, Quan Kong, Tianyu Liu, Yao Lu, and Cong Wang. Speculative decoding via hybrid drafting and rollback-aware branch parallelism. *arXiv preprint arXiv:2506.01979*, 2025.

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. Specexec: Massively parallel speculative decoding for interactive LLM inference on consumer devices. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. Bitnet: Scaling 1-bit transformers for large language models, 2023. URL https://arxiv.org/abs/2310.11453.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/723933067ad315269b620bc0d2c05cba-Abstract-Conference.html.

Hao Xu, Jiacheng Liu, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. Infini-gram mini: Exact n-gram search at the internet scale with fm-index. *CoRR*, abs/2506.12229, 2025.

Penghui Yang, Cunxiao Du, Fengzhuo Zhang, Haonan Wang, Tianyu Pang, Chao Du, and Bo An. Longspec: Long-context lossless speculative decoding with efficient drafting and verification, 2025. URL https://arxiv.org/abs/2502.17421.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025. URL https://arxiv.org/abs/2501.01005.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*, pp. 521–538, Carlsbad, CA, jul 2022. USENIX Association. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2007.14062.

Ziyi Zhang, Ziheng Jiang, Chengquan Jiang, Menghan Yu, Size Zheng, Haibin Lin, Henry Hoffmann, and Xin Liu. Swiftspec: Ultra-low latency llm decoding by scaling asynchronous speculative decoding, 2025. URL https://arxiv.org/abs/2506.11309.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*. USENIX Association, 2024.

A THEORETICAL RESULTS

A.1 THEOREM 7 PROOF: MODELING THE SPEEDUP FROM SSD

We copy Theorem 7 for reference:

Theorem 18. *Let the primary and backup speculators take time T_p and T_b relative to the verifier. Let the expected number of generated tokens from the primary speculator be E_{hit} and and backup speculator by E_{miss}. The expected speedup of Algorithm 1 relative to autoregressive decoding is then:*

$$speedup = \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit} \cdot \max(1, T_p) + (1 - p_{hit}) \cdot (1 + T_b)}.$$

The global cache hit rate p_{hit} can be expressed in terms of the cache hit rates $p_{hit,p}$ and $p_{hit,b}$, corresponding to whether the last round's speculator was the primary one or the backup, respectively (assuming $|p_{hit,p} - p_{hit,b}| < 1$):

$$p_{hit} = \frac{p_{hit,b}}{1 + p_{hit,b} - p_{hit,p}}.$$

Proof. We will prove this theorem in two parts:

- **Part 1**: We will first prove this speedup equation, *assuming we know the overall cache hit rate p_{hit}* (conditioned on choice of lookahead, cache topology, sampling algorithm, etc.).
- **Part 2**: We will then prove the functional form of p_{hit}, as a function of the cache hit rates $p_{hit,p}$ and $p_{hit,b}$ of the primary and backup speculators, respectively.

A.1.1 PART 1

In each iteration of SSD, the expected number of generated tokens is E_{hit} if there is a cache hit, and E_{miss} if there is not. Similarly, the latency is $\max(1, T_p)$ if there is a cache hit, and $1 + T_b$ if there is not—this is because backup speculation, which takes time T_b, begins only after verification (which we assume takes 1 unit of time) completes.

Therefore, it is clear that the expected number of generated tokens and the expected latency (relative to autoregressive decoding) in one iteration of SSD are:

$$\begin{aligned} E[\text{\# Generated tokens}] &= p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss} \\ E[\text{Latency}] &= p_{hit} \cdot \max(1, T_p) + (1 - p_{hit}) \cdot (1 + T_b) \end{aligned}$$

Thus, the expected speedup is:

$$\begin{aligned} speedup &= \frac{E[\text{\# Generated tokens}]}{E[\text{Latency}]} \\ &= \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit} \cdot \max(1, T_p) + (1 - p_{hit}) \cdot (1 + T_b)}. \end{aligned}$$

This concludes part 1 of the proof.

A.1.2 PART 2

We will now prove that the overall (unconditional) cache hit rate p_{hit} is equal to:

$$p_{hit} = \frac{p_{hit,b}}{1 + p_{hit,b} - p_{hit,p}}.$$

where $p_{hit,p}$ and $p_{hit,b}$ are the cache hit rates conditioned on the prior iteration being speculated by the primary and backup speculators, respectively.

The challenge in deriving a closed-form solution for the overall cache hit rate p_{hit} is that p_{hit} at iteration T of the SSD algorithm depends on whether there was a cache hit in the previous round (in which case the primary speculator was used) or not (in which case, the backup speculator was used).

In order to deal with this recursive property of p_{hit}, we first write p_{hit} as a recursive equation, which considers the iteration number t of the algorithm. We consider both the base case ($t = 0$), where for now we will assume we always use the non-neural spec, along with all rounds thereafter:

$$\begin{aligned} p_{hit}(0) &= p_{hit,p}, \quad \text{because we use the primary speculator at } T = 0, \\ p_{hit}(T) &= p_{hit}(T-1) \cdot p_{hit,p} + \Big(1 - p_{hit}(T-1)\Big) \cdot p_{hit,b} \end{aligned}$$

We rearrange and unroll this recurrence:

$$\begin{aligned} p_{hit}(T) &= p_{hit}(T-1) \cdot \Big(p_{hit,p} - p_{hit,b}\Big) + p_{hit,b} \\ &= p_{hit}(T-1) \cdot r + p_{hit,b}, \quad \text{letting } r \coloneqq p_{hit,p} - p_{hit,b} \\ &= \Big(p_{hit}(T-2) \cdot r + p_{hit,b}\Big) \cdot r + p_{hit,b} \\ &= (T-2) \cdot r^2 + p_{hit,b} \cdot r + p_{hit,b} \\ &= \Big(p_{hit}(T-3) \cdot r + p_{hit,b}\Big) \cdot r^2 + p_{hit,b} \cdot r + p_{hit,b} \\ &= p_{hit}(T-3) \cdot r^3 + p_{hit,b} \cdot r^2 + p_{hit,b} \cdot p_{hit,b} \\ &= \dots \\ &= p_{hit}(0) \cdot r^T + p_{hit,b} \sum_{t=0}^{T-1} r^t \\ &= p_{hit}(0) \cdot r^T + p_{hit,b} \frac{1 - r^T}{1 - r} \end{aligned}$$

Using the stated assumption that $|r| \coloneqq |p_{hit,p} - p_{hit,b}| < 1$, we can see that the first term above converges to 0, and the second term converges to $\frac{p_{hit,b}}{1-r} = \frac{p_{hit,b}}{1+p_{hit,b}-p_{hit,p}}$, as expected.

This concludes the proof. □

A.1.3 Proving the Two Corollaries

We now prove the two corollaries of Theorem 7, copied below:

Corollary 19. ***(Strictly Faster Than SD)****. Suppose we run SD with a given speculator $\mathcal{M}$. Running SSD with primary and backup both set to $\mathcal{M}$ does no worse than SD (strictly better if p_{hit}, $T_{SD} > 0$).*

Proof. Let $E_{hit} = E_{miss} = E_{SD}$, and $T_p = T_b = T_{SD}$. Then

$$\begin{aligned} speedup &= \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit} \cdot \max(1, T_p) + (1 - p_{hit}) \cdot (1 + T_b)} \\ &= \frac{p_{hit} \cdot E_{SD} + (1 - p_{hit}) \cdot E_{SD}}{p_{hit} \cdot \max(1, T_{SD}) + (1 - p_{hit}) \cdot (1 + T_{SD})} \\ &= \frac{E_{SD}}{p_{hit} \cdot \max(1, T_{SD}) + (1 - p_{hit}) \cdot (1 + T_{SD})} \end{aligned}$$

Recall that the speedup from SD is $E_{SD}/(1 + T_{SD})$. This final term is strictly greater than the SD speedup if $p_{hit} > 0$ and $T_{SD} > 0$, because in this case $\max(1, T_{SD}) < 1 + T_{SD}$. If $p_{hit} = 0$ or $T_{SD} = 0$, then the SSD speed is equal to the SD speed. □

Corollary 20. ***(Speedup sandwich)*** *Suppose we choose a primary speculator for which drafting completes before verification ($T_p < 1$), and a fast backup speculator ($T_b = 0$). Then if T_{SD}, E_{SD}*

represent the latency and expected number of generated tokens from a draft model in SD, then the SSD speedup over SD can be bounded by:

$$\Big(1+T_{SD}\Big)\cdot\frac{E_{hit}}{E_{SD}}\cdot p_{hit}\leq\frac{speedup_{SSD}}{speedup_{SD}}\leq\Big(1+T_{SD}\Big)\cdot\frac{E_{hit}}{E_{SD}}.$$

Proof. By assumption, $T_p<1$ and $T_b=0$. So the SSD speedup is:

$$\begin{aligned}speedup_{SSD} &= \frac{p_{hit}\cdot E_{hit}+(1-p_{hit})\cdot E_{miss}}{p_{hit}\cdot\max(1,T_p)+(1-p_{hit})\cdot(1+T_b)}\\ &= p_{hit}\cdot E_{SD}+(1-p_{hit})\cdot E_{SD}\end{aligned}$$

Recall the SD speedup equation is:

$$speedup_{SD} = \frac{E_{SD}}{1+T_{SD}}.$$

So,

$$\frac{speedup_{SSD}}{speedup_{SD}} = \frac{p_{hit}\cdot E_{hit}+(1-p_{hit})\cdot E_{miss}}{E_{SD}/(1+T_{SD})}.$$

Because $p_{hit}\leq 1$, we get the upper bound (assuming $E_{hit}\geq E_{miss}$):

$$\frac{speedup_{SSD}}{speedup_{SD}} = \Big(1+T_{SD}\Big)\cdot\frac{E_{\text{hit}}}{E_{SD}}.$$

Because $E_{miss}\geq 1$ (bonus token is always generated), we get the lower-bound:

$$\begin{aligned}\frac{speedup_{SSD}}{speedup_{SD}} &= \frac{p_{hit}\cdot E_{hit}+(1-p_{hit})\cdot E_{miss}}{E_{SD}/(1+T_{SD})}\\ &\geq \Big(1+T_{SD}\Big)\cdot\frac{E_{\text{hit}}}{E_{SD}}\cdot p_{\text{hit}}.\end{aligned}$$

□

A.2 Theorem 12 Proof: Optimizing Saguaro cache topology

To prove Theorem 12, we will first prove Theorem 21, which is a more general version of the theorem. Then, Theorem 12 will be an easy corollary of this more general theorem.

Theorem 21. *The choice of F_k^p (and equivalently, F_k^b) values that maximizes the speedup of* Saguaro *under the constraint $\sum_{k=0}^{K}F_k^p\leq B$ (where K is the speculative lookahead), is:*

$$\begin{aligned}\sum_{k=0}^{K}F_k^p &= B,\\ \frac{\partial p_{hit,p}^{k}}{\partial F}\big(F_k\big) &= a^{-k}\cdot\frac{\partial p_{hit,p}^{0}}{\partial F}\big(F_0\big)\quad\forall k<K,\ \textit{and}\\ \frac{\partial p_{hit,p}^{K,all}}{\partial F}\big(F_K\big) &= (1-a_p)\cdot a^{-K}\cdot\frac{\partial p_{hit,p}^{0}}{\partial F}\big(F_0\big).\end{aligned}$$

Proof. To understand how to optimize our cache topology, we first rewrite the speedup equations from Section 3.2, now making explicit how the speedup depends on the F_k^p and F_k^b values:

$$\begin{aligned}speedup\Big(\{F_k^p\}_{k=0}^{K}),\{F_k^b\}_{k=0}^{K})\Big) &= p_{hit}\Big(\{F_k^p\},\{F_k^b\}\Big)\cdot E_{hit}\\ &+ \Big(1-p_{hit}\Big(\{F_k^p\},\{F_k^b\}\Big)\Big)\cdot E_{miss},\quad\text{where}\\ p_{hit}\Big(\{F_k^p\},\{F_k^b\}\Big) &= \frac{p_{hit,b}(\{F_k^b\})}{1+p_{hit,b}(\{F_k^b\})-p_{hit,p}(\{F_k^p\})}.\end{aligned}$$

We can express $p_{hit,p}$ more precisely in terms of:

- The acceptance rate a_p of the primary speculator, and
- The functions $p^k_{hit,p}(F_k)$ and $p^{K,all}_{hit,p}(F_k)$: These functions describe the probability of a cache hit conditioned on (1) the last round's speculator was the primary one, (2) k tokens were accepted, (3) whether all of the tokens were accepted ($p^{K,all}_{hit,p}$) or not ($p^k_{hit,p}$), and (4) F_k verification outcomes were prepared for in case k tokens were accepted. Note that this can be estimated empirically by comparing the draft and target probability distributions for a set of sequences in a calibration set.

We can do the same for $p_{hit,b}$, the cache hit rate when the last round's speculator was the backup.

$$p_{hit,p}(\{F^p_k\}) \quad = \quad a^K_p \cdot p^{K,all}_{hit,p}(F_K) + \sum_{k=0}^{K-1} a^k_p(1-a_p) \cdot p^k_{hit,p}(F_k), \quad and$$

These equation are a direct result of the fact that the chance of accepting exactly k tokens is $a^k_p(1-a_p)$ for $k < K$, and a^K_p for $k = K$, when the acceptance rate is a.

Given these equations, we can now optimize the cache topology (i.e., the choice of F_k values) to maximize speedup. We notice that the speedup is monotonically increasing in p_{hit}, and that p_{hit} is monotonically increasing in both $p_{hit,p}$ and $p_{hit,b}$ (easy to see by taking first derivatives of p_{hit}, and seeing they are always positive). Thus, we must simply maximize $p_{hit,p}(\{F^p_k\})$ and $p_{hit,b}(\{F^b_k\})$ under the constraints that $\sum_{k=0}^K F^p_k \leq B$ and $\sum_{k=0}^K F^b_k \leq B$. We do this below

A.2.1 Maximizing $p_{hit,p}(\{F^p_k\})$ and $p_{hit,b}(\{F^b_k\})$

As discussed, we want to maximize the probability of a cache hit under the budget constraint $\sum_{k=0}^K F_K \leq B$. We do this for $p_{hit,p}$, and the proof is identical for $p_{hit,b}$.

$$\max_{\sum_k F_k \leq B} p_{hit,p}(K,F) = \max_{\sum_k F_k \leq B} a^K_p \cdot p^{K,all}_{hit,p}(F_K) + \sum_{k=0}^{K-1} a^k_p(1-a_p) \cdot p^k_{hit,p}(F_k)$$

We will solve this maximization problem with Lagrange multipliers.

$$\mathcal{L}(F_0, \ldots, F_k, \lambda) = a^K_p \cdot p^{K,all}_{hit,p}(F_K) + \sum_{k=0}^{K-1} a^k_p(1-a_p) \cdot p^k_{hit,p}(F_k) + \lambda \cdot \Big(\sum_{k=0}^{K} F_k - B \Big)$$

Now, we will take the derivative with respect to all the variables, and set it to zero.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial F_k} &= a^k_p(1-a_p) \cdot \frac{\partial}{\partial F_k} p^k_{hit,p}(F_k) + \lambda = 0 \quad \text{for } k < K \\
\frac{\partial \mathcal{L}}{\partial F_K} &= a^K_p \frac{\partial}{\partial F_k} p^{K,all}_{hit,p}(F_K) + \lambda = 0 \\
\frac{\partial \mathcal{L}}{\partial \lambda} &= \sum_{k=0}^{K} F_K - B = 0
\end{aligned}$$

We notice that for all k, $\frac{\partial \mathcal{L}}{\partial F_k}$ are equal to one another (all equal to $-\lambda$). Thus, we see that:

$$\begin{aligned}
(1-a_p)\frac{\partial}{\partial F_k}p^0_{hit,p}(F_0) &= a(1-a_p)\frac{\partial}{\partial F_k}p^1_{hit,p}(F_1) = \dots \\
\dots &= a^{K-1}(1-a_p)\frac{\partial}{\partial F_k}p^{K-1}_{hit,p}(F_{K-1}) = a_p^K\frac{\partial}{\partial F_k}p^{K,all}_{hit,p}(F_K) \\
\Rightarrow \frac{\partial}{\partial F_k}p^k_{hit,p}(F_k) &= a^{-k}\frac{\partial}{\partial F_k}p^0_{hit,p}(F_0) \quad \forall k < K, \text{ and} \\
\frac{\partial}{\partial F_k}p^{K,all}_{hit,p}(F_K) &= (1-a_p)a^{-K}\frac{\partial}{\partial F_k}p^0_{hit,p}(F_0)
\end{aligned}$$

This gives the desired result.

□

We now use this result to prove Theorem 12, which is a special case of the above theorem in the case where the speculators have r power-law cache hit rates.

Theorem 22. ***(SAGUARO Cache Topology)*** *The optimal choice of F^p_k (equivalently, F^b_k) values for $k \in [0, K]$ for* SAGUARO *under the constraint $\sum_{k=0}^K F^p_k \le B$, and under the assumption that the speculator has a r power-law cache hit rate, follows a geometric series (for $k < K$):*

$$\begin{aligned}
F_k &= F_0 \cdot a^{k/(1+r)} \ \ \forall k < K, \textit{ and} \\
F_K &= F_0 \cdot a^{K/(1+r)} \cdot (1-a)^{-1/(1+r)},
\end{aligned}$$

where F_0 can be chosen as follows so that $\sum_{k=0}^K F^p_k = B$.

$$F_0 = \frac{B}{a^{K/(1+r)} \cdot (1-a_p)^{-1/(1+r)} + \left(1 - a^{K/(1+r)}\right)/\left(1 - a^{1/(1+r)}\right)}$$

Proof. Now, substituting $p^k_{hit,p}(F) = 1 - F^{-r}$ (and thus $\frac{\partial}{\partial F_k}p^k_{hit,p}(F) = b \cdot F^{-r-1}$), we get:

$$\begin{aligned}
\Rightarrow r \cdot F_k^{-r-1} &= a_p^{-k} r \cdot F_0^{-r-1} \ \ \forall k < K, \text{ and} \\
r \cdot F_K^{-r-1} &= (1-a_p) \cdot a_p^{-K} \cdot r \cdot F_0^{-r-1}. \\
\Rightarrow F_k &= F_0 \cdot a_p^{k/(1+r)} \ \ \forall k < K, \text{ and} \\
F_K &= F_0^{(1+r)/(1+r)} \cdot a_p^{K/(1+r)} \cdot (1-a_p)^{-1/(1+r)} \cdot (r/r)^{-1/(1+r)}.
\end{aligned}$$

To solve for the exact sequence of F_k fan-out values, we plug the above values into the budget equation:

$$B = F_0 \cdot a_p^{K/(1+r)} \cdot (1-a_p)^{-1/(1+r)} + \sum_{k=0}^{K-1} F_0 \cdot a_p^{k/(1+r)}$$

Solve for F_0 gives:

$$F_0 = \frac{B}{a_p^{K/(1+r)} \cdot (1-a_p)^{-1/(1+r)} + \sum_{k=0}^{K-1} a_p^{k/(1+r)}}$$

We can simplify this further using the equation for the sum of the geometric series:

$$\sum_{k=0}^{K-1} a_p^{k/(1+r)} = \sum_{k=0}^{K-1} c^k = \frac{1-c^K}{1-c}, \quad \text{for } c = a_p^{1/(1+r)}$$

Plugging this in gives the desired result.

$$F_0 = \frac{B}{a_p^{K/(1+r)} \cdot (1-a_p)^{-1/(1+r)} + \left(1 - a_p^{K/(1+r)}\right)/\left(1 - a_p^{1/(1+r)}\right)}$$

□

A.3 THEOREM 15 PROOF: OPTIMIZING SAGUARO SAMPLING ALGORITHM

We now prove Theorem 15, copied below for reference:

Theorem 23. *For fan-out F, and draft logits z, the cache hit rate p_{hit} of the* SAGUARO *sampling algorithm increases as $C \to 0$.*

Proof. Because the overall cache hit rate p_{hit} is monotonically increasing in the cache hit rate $p_{hit,p}$ of the primary speculator, here it is sufficient to show that $p_{hit,p}$ of the SAGUARO sampling algorithm increases as $C \to 0$.

We assume here that the F tokens $(t_1, \ldots, t_F)$ with the highest draft logits are the ones we put in the cache, and we let $p_{draft} \coloneqq \sigma_{F,C}(z)$, the draft distribution constructed with the SAGUARO sampling algorithm. It is clear that by reducing the value of C the amount of residual probability mass on the top F draft tokens increases, while the total residual mass on all other tokens decreases.

We now look at the amount of residual probability mass (before normalization) **in** the top F tokens $\text{in}(C)$, and the amount outside of the top F tokens $\text{out}(C)$, and show that $\text{in}(C)$ is increasing in C, while $\text{out}(C)$ is decreasing in C. I will then use the fact that the probability of a cache hit (for any $k < K$) is equal to:

$$p^k_{hit,p}(F) = \frac{\text{in}(C)}{\text{in}(C) + \text{out}(C)}$$

to prove that the cache hit rate increases as C drops. In particular, I will show that the derivative of $p^k_{hit,p}(F)$ with respect to C is negative, which shows that as C grows, the cache hit rate drops, or conversely that when C shrinks from 1 toward 0, the cache hit rate increases.

$$\begin{aligned}
\frac{\partial p^k_{hit,p}(F)}{dC} &= \frac{\text{in}'(C) \cdot \Big(\text{in}(C) + \text{out}(C)\Big) - \Big(\text{in}'(C) + \text{out}'(C)\Big) \cdot \text{in}(C)}{\Big(\text{in}(C) + \text{out}(C)\Big)^2} \\
&= \frac{\text{in}'(C) \cdot \text{out}(C) - \text{out}'(C) \cdot \text{in}(C)}{\Big(\text{in}(C) + \text{out}(C)\Big)^2}
\end{aligned}$$

It's easy to see that this value is ≤ 0 if $\text{in}'(C) \leq 0$ and $\text{out}'(C) \geq 0$, because $\text{in}(C)$ and $\text{out}(C)$ are both ≥ 0 by definition. We can see $\text{in}'(C) \leq 0$ because increasing C by construction reduces the residual probability mass in the top-F tokens. We can also see that $\text{out}'(C) \geq 0$ because increasing C by construction increases the residual probability mass outside of the top-F tokens.

This concludes the proof.

□

A.4 COROLLARY 16 AND THEOREM 17 PROOFS: OPTIMIZING SAGUARO FALLBACK STRATEGY

We first prove corollary 16, and then Theorem 17, both copied below for reference:

Corollary 24. *At batch size b, the expected speedup from SSD is equal to:*

$$\begin{aligned}
\textit{speedup} &= \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p^b_{hit} \cdot \max(1, T_p) + (1 - p^b_{hit}) \cdot (1 + T_b)}, \quad \textit{which approaches} \\
&\quad \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{1 + T_b} \quad \textit{as } b \to \infty.
\end{aligned}$$

Proof. For each element of the batch, if it gets a cache hit it will generate E_{hit} tokens, and otherwise E_{miss}. The latency of an element of the batch, however, depends on whether any element of the

batch had a cache miss. If so, the latency for the entire batch is $1 + T_b$. Otherwise, if every element of the cache had a hit, the latency is $\max(1, T_p)$. Thus, we can see that:

$$\begin{aligned} E[\text{\# Generated tokens}] &= p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss} \\ E[\text{Latency}] &= p_{hit}^b \cdot \max(1, T_p) + (1 - p_{hit}^b) \cdot (1 + T_b) \end{aligned}$$

Thus, the expected speedup is:

$$\begin{aligned} speedup &= \frac{E[\text{\# Generated tokens}]}{E[\text{Latency}]} \\ &= \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit}^b \cdot \max(1, T_p) + (1 - p_{hit}^b) \cdot (1 + T_b)}. \end{aligned}$$

Noticing that $p_{hit}^b \to 0$ as b grows concludes the proof. This concludes part 1 of the proof. □

Theorem 25. *The optimal cache miss strategy, conditioned on only being able to choose between a high-quality slow speculator (primary), and a lower-quality speculator with negligible latency (backup), is to use the primary speculator for batch sizes $b < b^*$, and the backup speculator otherwise. The value of b^* is given by:*

$$b^* = \frac{1}{\log(p_{hit})} \cdot \log\left(\left(1 + \frac{1}{T_p} - \frac{E_{hit}}{T_p \cdot p_{hit} \cdot E_{hit} + T_p \cdot (1 - p_{hit}) \cdot E_{miss}}\right)\right)$$

Proof. Using the slow primary speculator as the backup speculator gives expected speedup:

$$\begin{aligned} speedup_{slow\_backup} &= \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{hit}}{p_{hit}^b \cdot \max(1, T_p) + (1 - p_{hit}^b) \cdot (1 + T_p)}. \\ &= \frac{E_{hit}}{p_{hit}^b + (1 - p_{hit}^b) \cdot (1 + T_p)}. \\ &= \frac{E_{hit}}{1 + T_p - T_p \cdot p_{hit}^b}. \end{aligned}$$

Using the fast backup speculator (with $T_b = 0$) as the backup speculator gives expected speedup:

$$\begin{aligned} speedup_{fast\_backup} &= \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit}^b \cdot \max(1, T_p) + (1 - p_{hit}^b) \cdot (1 + T_b)}. \\ &= \frac{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}{p_{hit}^b \cdot \max(1, T_p) + (1 - p_{hit}^b)}. \\ &= p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}. \end{aligned}$$

We set these equations equal to each other and solve for b, which gives the equation in the Theorem statement.

$$\begin{aligned} p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss} &= \frac{E_{hit}}{1 + T_p - T_p \cdot p_{hit}^b} \\ 1 + T_p - T_p \cdot p_{hit}^b &= \frac{E_{hit}}{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}} \\ p_{hit}^b &= \frac{1}{T_p}\left(1 + T_p - \frac{E_{hit}}{p_{hit} \cdot E_{hit} + (1 - p_{hit}) \cdot E_{miss}}\right) \\ b^* &= \frac{1}{\log(p_{hit})} \cdot \log\left(\left(1 + \frac{1}{T_p} - \frac{E_{hit}}{T_p \cdot p_{hit} \cdot E_{hit} + T_p \cdot (1 - p_{hit}) \cdot E_{miss}}\right)\right). \end{aligned}$$

Now we must simply show that for batch sizes $b < b^*$, it is better to use the slow backup speculator (a.k.a., the primary speculator), and for $b \geq b^*$ it is better to use the fast backup speculator.

We do this by seeing that the $speedup_{slow\_backup}$ is monotonically decreasing in the batch size b (negative derivative with respect to b), whereas the $speedup_{fast\_backup}$ does not depend on b (obvious from equation). This shows that if there is a value b^* that makes these two speedups equal, then for $b < b^*$ the slow backup option gives a larger speedup, whereas for $b \geq b^*$ the fast backup option gives a larger speedup.

$$\begin{aligned}\frac{\partial speedup_{slow\_backup}}{db} &= \frac{\partial}{db}\left(\frac{E_{hit}}{1+T_p-T_p\cdot p_{hit}^b}\right)\\ &= \frac{E_{hit}\cdot T_p\cdot p^b\cdot \log(p_{hit})}{(1+T_p-T_p\cdot p^b)^2},\end{aligned}$$

which is clearly negative because $\log(p_{hit}) < 0$ and everything else is positive. This concludes the proof. □

B Implementation Details

B.1 Systems Design

Overall Design. We implement SAGUARO as a custom inference engine from scratch, incorporating PagedAttention (Kwon et al., 2023b), continuous batching (Yu et al., 2022), tensor parallelism, BF16 mixed precision, torch compilation, and CUDAGraphs.

The engine orchestrates from a coordinator process on the main GPU. A scheduler paired with a block manager handles prefill/decode scheduling and page-table bookkeeping. A ModelRunner on each target GPU prepares attention metadata and executes forward passes. In async mode, the draft model runs in a separate process on its own GPU. Target and draft communicate once per iteration via NCCL with fused payloads: the target sends cache keys (sequence ID, accepted-prefix length, recovery token), current sequence lengths, draft block tables for KV addressing, and per-row temperatures; the draft returns a cache-hit bitmap, K speculative tokens per sequence, and K-step logits for acceptance.

Crucially, while the target host maintains page-table bookkeeping for both models, the draft KV cache tensor resides on the draft device—no KV data is ever transferred. The scheduler ensures the target has sufficient pages for $K+1$ multi-query decoding steps, preempting sequences when lookahead reservations cannot be satisfied. After verification, both page tables are reconciled: completed pages are finalized (hashed for prefix caching), and any pages allocated beyond the accepted suffix are deallocated. This rollback requirement—undoing allocations for rejected tokens that wrote into pre-allocated pages—necessitates a host-side post-processing step after each verification.

Design Decisions and Performance Engineering. During draft speculation, all $F(K+1)$ branches of each sequence are decoded in parallel using a custom sparse attention mask that allows each branch to attend to the verified trunk and its own forking path. We use FlashAttention kernels (Shah et al., 2024) where possible, falling back to FlashInfer (Ye et al., 2025) for multi-query decoding paths requiring custom masks. An example mask is shown below.

Materializing these masks—which depend on prefix length, B, K, F, and step i—is a substantial overhead. The sparse, non-coalesced memory access patterns in custom-mask attention kernels dominate our critical path, limiting how many steps we can profitably draft. As a result, most end-to-end speedup comes from hiding draft latency rather than increasing lookahead depth. Because accepted branches land in fragmented KV cache locations, we perform a "glue" append of the previous speculation before each round of async decoding, rather than copying fragments into contiguous pages. This corresponds to the "Glue & Recurrence" column in Figure 9, enabling all F forked branches to attend to the same verified prefix.

We originally expanded forked sequences along the batch dimension, which permitted standard causal masks but required reloading the prefix KV cache at every step. This motivated refactoring to multi-query decoding with custom masks, avoiding repeated prefix loads by sharing the trunk KV across all branches in a single forward pass.

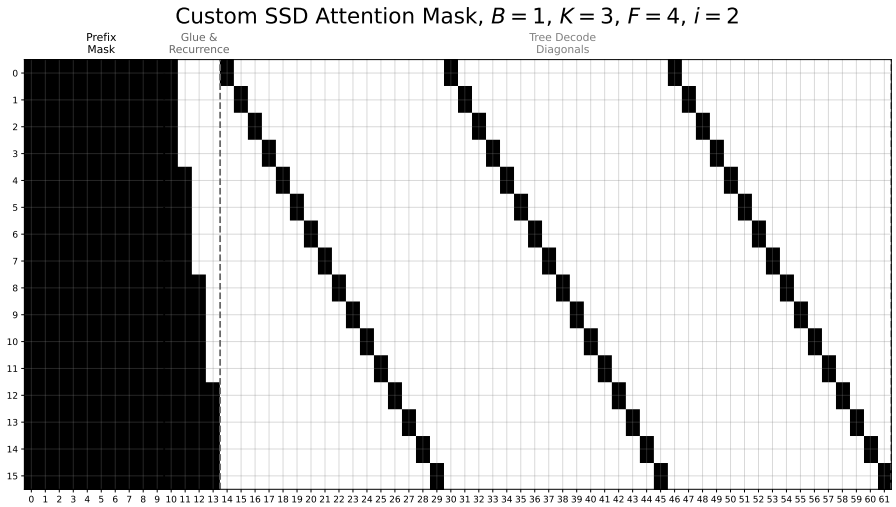

Figure 9: Custom attention mask for multi-query decoding of all $BF(K+1)$ verification branches in parallel. This mask is for $B = 1$, $K = 3$, uniform fan-out $F = 4$, at depth $i = 2$. Black indicates tokens that can be attended to. The left block shows attention to the verified prefix; diagonal bands show each branch attending only within its forking path.

B.2 Experimental Design

Datasets. We take 512 prompts from each dataset (or the maximum number in the dataset, if less than this), using maximum prompt length 128 and sample 512 decoding tokens for every prompt. We use vanilla sampling throughout (not top-p or top-k). We measure decoding throughput, excluding prefill. All experiments are done on a single node of NVIDIA Hopper GPUs.

Baselines. We use an SGLang baseline because we find vLLM has very weak support for speculative decoding (decoding speeds were half of SGLang), and wanted to compare against the strongest possible baselines. For both, we considered vanilla (standalone) speculative decoding, since methods like EAGLE can be combined with speculative speculative decoding, and are thus not mutually exclusive baselines. We always use torch compilation and CUDAgraphs in our baselines, including in our implementation of ordinary speculative decoding.

B.3 Numerical Results

Model	Dataset	AR tok/s	SD tok/s	Speedup	Latency (ms)	SSD tok/s	Speedup	Latency (ms)
Llama-3.3 70B/1B	HumanEval	56	147	2.63×	6.803	245	4.38×	4.082
	Ultrafeedback	56	126	2.25×	7.937	200	3.58×	5.000
	Alpaca	56	127	2.26×	7.874	205	3.66×	4.878
	GSM8k	56	154	2.74×	6.494	257	4.59×	3.891
	Average	56	138	2.47×	7.246	227	4.05×	4.405
Qwen-3 32B/0.6B	HumanEval	93	147	1.59×	6.803	195	2.10×	5.128
	Ultrafeedback	93	128	1.38×	7.812	161	1.73×	6.211
	Alpaca	93	126	1.35×	7.937	167	1.80×	5.988
	GSM8k	93	157	1.69×	6.369	202	2.18×	4.950
	Average	93	140	1.50×	7.143	181	1.95×	5.525

Table 1: Throughput and inter-token latency comparison. AR: Autoregressive baseline, SD: Speculative Decoding, SSD: Staged Speculative Decoding. Latency refers to average inter-token latency in (ms) and is the reciprocal of tokens/sec.

C SSD OVERHEAD

Let B be the target batch size, K the speculation lookahead, and F the (uniform) fan out factor for the draft model guessing verification bonus tokens.

Compute. Let $\hat{c}$ be the compute required for a draft forward pass, normalized by units of target FLOPs. Since the draft decodes $BK(K+1)F$ tokens per round in SSD but BK tokens per round in SD, we incur a factor of $\hat{c}(K+1)F$ more FLOPs on the draft relative to ordinary speculative decoding. Recall this is because each step on (of which there are K) the draft decode $B(K+1)F$ tokens in parallel using a custom attention mask.

In much the same way that SD uses more FLOPs to achieve lower latency (see (Leviathan et al., 2023), Section 3.4), SSD applies the same philosophy to use *even more FLOPs* to achieve *even lower latency* than even SD itself. In the SD setting, tokens speculated by the draft that were rejected by the target constitute wasted compute. In the SSD setting (in addition to the above), we have that entire *chains* decoded pre-emptively in parallel for anticipated verification outcomes constitute wasted compute. Thus, SSD introduces new tradeoffs between compute and latency that were not possible before.

Memory. The draft model must build up a speculation cache as it speculates asynchronously. This has possible verification outcomes as keys and the corresponding tokens/logits as values. Concrete, this means storing a tensor of $BK(K+1)F$ tokens that are decoded, in the form of length-K speculations stored for $B(K+1)F$ possible verification outcomes. For each of these tokens, we also store logits of size V, so the speculation cache stores overall $O\left(BFK(K+1)(V+1)\right)$ bits, ending up around hundreds of megabytes in practice. Given this cache is refreshed every speculation round, this ends up small enough to be a non-issue in practice, as the HBM on modern GPUs is much larger.

Communication. The draft and target model synchronize once per speculation round. The target model sends the outcome of the previous round of speculation (the number of accepted tokens and recovery token for each sequence, $O(B)$ bits of information). The draft sends back the newly speculated tokens ("cache hits") as well as their logits (which the target will need for verification). This is $O(BKV)$ bits of information. All communications are device to device over NCCL via NVLink, which is fast enough that communications are not a bottleneck in practice.

D EXTENDED RELATED WORK

Beyond draft–verify methods like those we study, the LLM inference stack has seen rapid progress along many axes. Our work situates itself within a larger tradition of deep learning scaling, enabled by hardware improvements and an improved science of hardware-aware algorithms. Hardware-aware attention kernels (e.g., FlashAttention) reduce HBM traffic and deliver wall-clock speedups without approximation (Dao et al., 2022). Serving systems co-design scheduling and memory: PagedAttention in vLLM implements virtual-memory–style KV paging and sharing, enabling larger effective batch sizes and higher throughput (Kwon et al., 2023a).

Memory pressure from KV caches has led to compression/eviction and quantization lines: H_2O identifies heavy-hitter tokens to guide KV retention; adaptive schemes discard or compress low-utility states (Ge et al., 2024); quantization approaches (e.g., KVQuant, MiniCache) push extreme compression with minimal quality loss (Hooper et al., 2024; Liu et al., 2024a) – with some recent works even training in binary or ternary precision to specifically alleviate inference-time memory bottlenecks (Wang et al., 2023).

Classic attention sparsification for long contexts (Longformer, BigBird) and kernelized/LSH approximations (Performer, Reformer) trade exactness for favorable scaling while retaining high quality across (Beltagy et al., 2020; Zaheer et al., 2020; Choromanski et al.). Finally, lossless multi-token and feature-level methods (Medusa, EAGLE) and parallel exact decoders (Lookahead) reduce steps or verifier calls via auxiliary heads or tree/parallel verification (Cai et al., 2024; Fu et al., 2024).

E Combining SSD with SD Variants

Advanced draft model architectures (e.g., EAGLE-3). We now discuss how we can combine SSD with advanced draft model architectures like EAGLE-3 (Li et al., 2025b) or GliDE (Du et al., 2024) that allow the draft model to leverage the powerful representations (activations or KV cache, respectively) of the target model to improve acceptance rates. We focus here on EAGLE-3 for illustration purposes, but the ideas transfer to this entire class of methods. We consider the token tree drafting/verification aspect of EAGLE-3 in the section below.

The primary difference between SSD with standalone speculators (e.g., Llama-1B) and SSD with an EAGLE-3 speculator is that the target model must communicate its activations for the latest verified tokens (and the prompt) to the speculator, as these are part of the input to the EAGLE speculator. From an algorithmic perspective, the only reason SSD + EAGLE-3 might have a slightly lower expected number of accepted tokens than EAGLE-3 (assuming the same lookahead), is that SSD does not have access to the target model activations for the token sequence that is currently being verified. Thus, in order to pre-speculate for the next round while verification is taking place, the draft must use its own activations to self-condition for longer than it would ordinarily in EAGLE-3. The EAGLE-3 paper (Figure 7), however, demonstrates that acceptance rates are actually quite stable many tokens ahead of the last target model activations, so this is unlikely to be an issue in practice.

Token-Tree Methods. Token-tree SD methods (Miao et al., 2024; Chen et al., 2024; Li et al., 2024b; 2025b) can be combined quite nicely with SSD. SSD drafts a linear chain for many possible verification outcomes. To combine SSD with these token-tree approaches, one would simply pre-speculate (and verify) a token tree for each verification outcome instead of a token chain. In practice, this requires a more elaborate attention mask than that presented in Figure 9. But fundamentally these methods can combine quite nicely.

F Additional Experiments

Here, we reproduce similar trends from the main text on the Qwen3 model family, showing our results and algorithms are model and dataset agnostic.

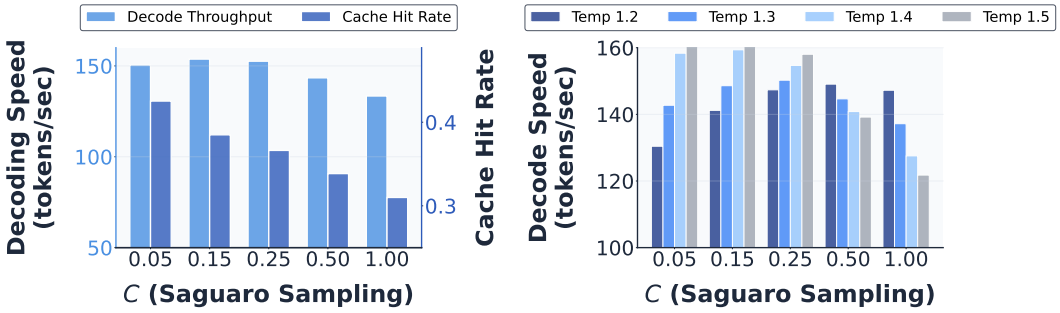

Figure 10: Effect of Saguaro sampling across temperatures for the Qwen3 model family. Qwen3-32B used as the target model, and 0.6B as the draft. We find similar trends as in the main text, where using the default $C = 1$ is suboptimal at higher temperatures.

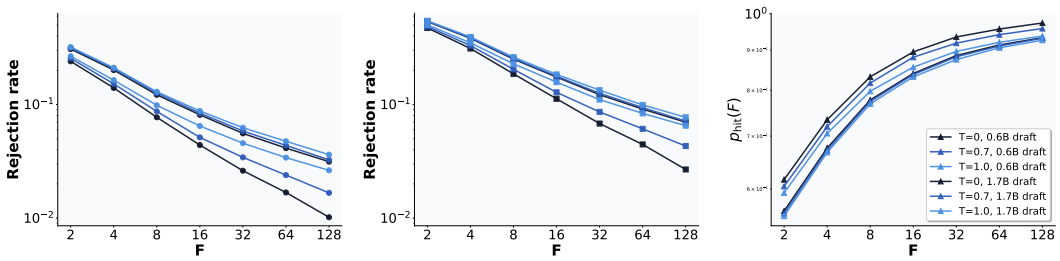

Figure 11: Rejection rate and cache hit rate scaling for Qwen-3 model family. Like the Llama-3 model family, we see it is an approximate power law in the fan-out, so that cache hit rates increase steadily as we grow F, the size of our cache.

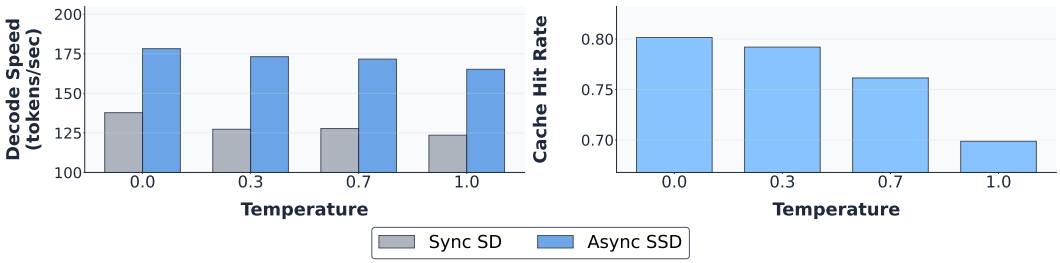

Figure 12: End-to-end speed for Qwen-3 model family compared to synchronous baselines.

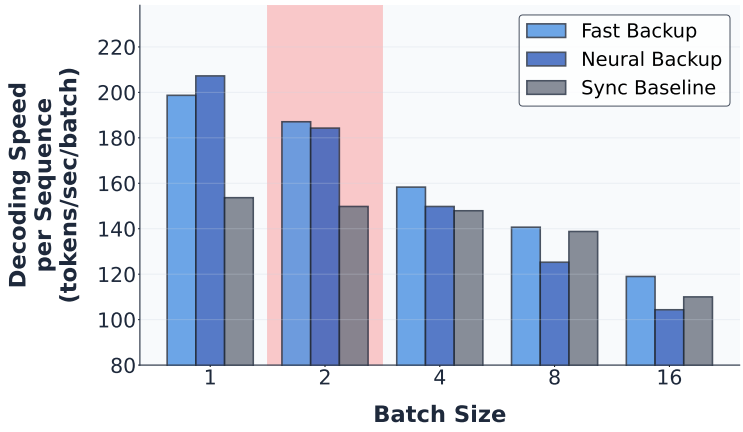

Figure 13: Batch size scaling of Qwen-3 models compared to synchronous baselines.