

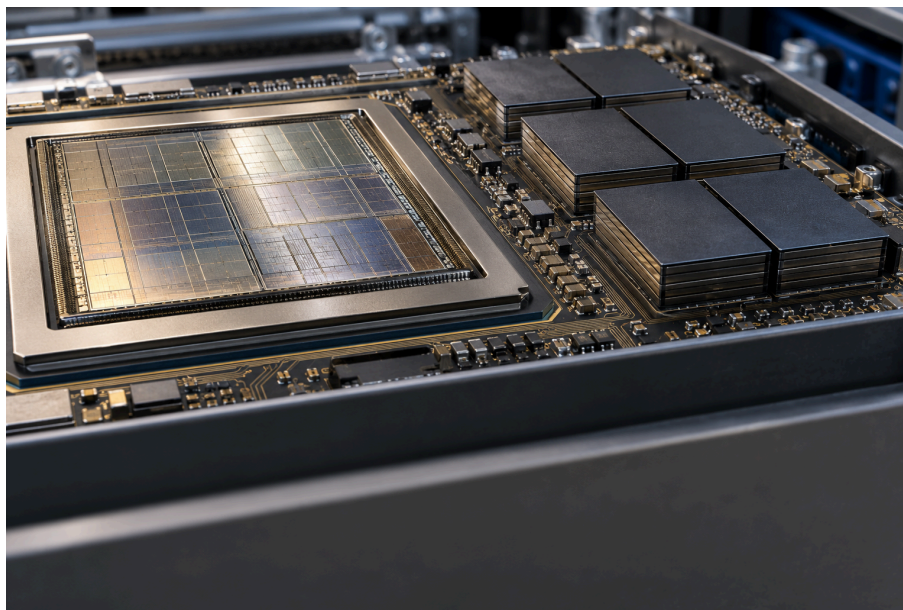


# Speculative Decoding Economics: When Draft Models Cut Cost

Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-10-18 · speculative decoding economics · speculative decoding cost savings · draft model selection · speculative decoding acceptance rate · eagle speculative decoding · vllm speculative decoding · cost per token self-hosted llm · private llm inference optimization · llm inference cost

Original article: <https://gpusmith.com/articles/en/speculative-decoding-economics-draft-model-cost>



## In this article

1. Executive Summary
  2. Introduction and Background
  3. Key Changes: From Tokens per Second to Cost per Accepted Token
  4. Key Changes: Lossless Sampling Versus Heuristic Acceptance
  5. Key Changes: Memory, Compute and Strategy Choice
  6. Key Changes: Concurrency Moves the Break-Even
  7. Implementation Considerations and Process Changes
  8. Negative Results: When Speculation Is Not Worth It
  9. Data Analysis and Evidence
  10. Implications and Future Directions
  11. Frequently Asked Questions (FAQs)
  12. Conclusion
- 

## Executive Summary

Speculative decoding lets a cheap proposer guess several output tokens that the large target model then verifies in a single forward pass. The original paper reported a 2X-3X acceleration of T5-XXL with identical outputs (([arxiv.org](https://arxiv.org))), and Google's retrospective states that the method guarantees an identical output distribution (([research.google](https://research.google))). Those results are real, but they were measured at batch size 1 on specific tasks, so they are observations, not a forecast for a private fleet. The decision a private-inference team faces is narrower: does the technique raise **SLO-qualified accepted output per GPU-hour** after paying for proposer memory, verification compute and lost batching headroom?

Published evidence says the answer depends on concurrency. NVIDIA measured Llama 3.3 70B on one H200 at batch size 1 and reported throughput speedups of 3.55x, 3.16x and 2.63x for 1B, 3B and 8B draft models (([developer.nvidia.com](https://developer.nvidia.com))). At higher load the gain shrinks or reverses: a trained EAGLE-3 draft for Kimi K2.5 gained over 60% at batch size 1 but 26% at batch size 16 (([pytorch.org](https://pytorch.org))), and vLLM's project blog reported a 1.4x slowdown for a 70B draft-model setup at high request rates (([vllm.ai](https://vllm.ai))). vLLM scopes the technique to medium-to-low request rates and memory-bound workloads (([docs.vllm.ai](https://docs.vllm.ai))). The one cost-linked vendor measurement found, from Red Hat, is a **19.4%** lower cost per million output tokens on one workload (([developers.redhat.com](https://developers.redhat.com))), with smaller gains on others.

A closed-form model makes the trade-off explicit. With modeled per-position acceptance probability  $\alpha$ ,  $k$  speculated tokens and a draft-to-target cost ratio  $c$ , the expected speedup is  $(1 - \alpha^{(k+1)}) / ((1 - \alpha)(kc + 1))$  (([arxiv.org](https://arxiv.org))). At  $c = 0.05$  and  $k = 4$ , theoretical break-even  $\alpha$  is about 0.17 and  $\alpha = 0.7$  yields 2.31x. If verification is not free, because the GPU is already near compute saturation, the same configuration at  $v = 0.25$  of one baseline target-step time per proposed token ( $k \cdot v = 1.0$ , or 100% extra verification time per iteration) yields 1.26x and falls below 1.0x at  $\alpha = 0.5$ . The overhead term is this report's modeling assumption, not a published constant.

The recommendation is to pilot speculation where measured qualified throughput and whole-replica cost justify it, including workloads with a binding p95 inter-token latency (ITL) target. Low-to-moderate concurrency is one favorable case. Red Hat also reports throughput gains across its tested concurrency levels for the MoE model gpt-oss-120b ([study](#)); MagicDec demonstrates benefits in long-context, large-batch settings ([research](#)). Prefer lossless rejection-sampling variants over heuristic acceptance, and compare whole replicas at stated p95 time-to-first-token (TTFT) and ITL targets. The implementation section provides a benchmark contract, a calculator and a hypothetical worked example. Acceptance shifts with prompt domain, so one aggregate number is unsafe.

## Introduction and Background

Enterprise teams running a private target model usually reach speculative decoding from a latency complaint: interactive users see slow token streams, and adding GPUs does not shorten a single request. The technique exploits idle arithmetic capacity. NVIDIA describes a small draft model that first predicts several likely next tokens ([developer.nvidia.com](#)), after which the target verifies them in parallel, and IBM summarizes it as one forward pass that can produce two or more tokens for the price of one ([research.ibm.com](#)). Google Cloud frames the payoff as generating 4-5 tokens for the memory cost of one decode step ([cloud.google.com](#)).

The vendor and research literature mostly reports speedup ratios. Those ratios are an input to cost, not a measure of it. A private fleet pays for GPU-hours whether or not tokens are accepted, whether or not requests met their latency targets, and whether or not the extra proposer state forced a larger replica. This report therefore uses one decision metric throughout: cost per million **accepted** output tokens from requests that met their SLO, compared across whole replicas.

The scope is an analyst's worksheet rather than a tutorial. It separates three questions that are often blurred: whether the algorithm preserves the target's output distribution (a property of the verification rule), whether a given engine reproduces that property in practice (numerics and batching), and whether the arrangement lowers fleet cost (hardware, traffic and SLOs). Vendor figures are treated as vendor claims with stated hardware and batch size, and prices are illustrations retrieved on 2026-10-18, not quotes.

GPU Smith is an independent private AI infrastructure engineering consultancy that advises on GPU procurement, workload sizing, inference optimization and air-gapped deployments. It is not a chip manufacturer or cloud provider. No benchmark results are invented here: numerical examples in later sections are either cited, direct outputs of a cited formula, or labeled hypothetical.

## Key Changes: From Tokens per Second to Cost per Accepted Token

### The cost model replaces the headline speedup

Leviathan, Kalman and Matias define the cost coefficient  $c$  as the ratio between the time for a single run of the draft model and a single run of the target ([arxiv.org](#)), with the acceptance rate  $\alpha$  measuring how well the draft approximates the target. Their expected improvement factor in walltime is  $(1 - \alpha^{(k+1)}) / ((1 - \alpha)(kc + 1))$  ([arxiv.org](#)).  $\alpha$  is a property of the model pair and task;  $c$  depends on hardware and software.

The paper's experiments sat at the favorable end of the  $c$  range: the cost coefficient was always less than 0.05 (([arxiv.org](#))). It also states the boundary condition: the method is not helpful for configurations where additional computation resources are not available (([arxiv.org](#))). Check whether the production configuration has spare compute before applying the model. This report therefore adds one explicit assumption, a verification overhead  $v$ , defined as extra target-step time per speculated token. At  $v = 0$  the formula reduces to the published one; at  $v = 1$  verification is fully compute-bound and every speculated token costs a full token of work.

## Four quantities replace one

A deployment decision needs four measured numbers:

- **Observed draft acceptance fraction ( $r$ )**: accepted draft tokens divided by proposed draft tokens, which vLLM defines as the fraction of proposed draft tokens accepted (([docs.vllm.ai](#))), sliced by prompt class and output position. This telemetry fraction counts the accepted prefix against all proposed tokens. Keep it separate from  $\alpha$ , the modeled per-position acceptance probability used under the original paper's independence assumption ([original analysis](#)). Do not insert  $r$  directly into the  $\alpha$ -based formula.
- **Mean acceptance length**: tokens emitted per verification step including the bonus token (([docs.vllm.ai](#))).
- **Net speedup at production concurrency**, not at batch size 1.
- **SLO-qualified goodput**: DistServe defines per-GPU goodput as the request rate that can be served adhering to the SLO (([arxiv.org](#))).

Latency vocabulary should follow a documented standard. NVIDIA's benchmarking documentation defines TTFT as how long a user waits before seeing the model's output (([docs.nvidia.com](#))), ITL as the average time between consecutive tokens (([docs.nvidia.com](#))), and system throughput as total output token throughput across all simultaneous requests (([docs.nvidia.com](#))). Speculation delivers tokens in bursts of one to  $k+1$  per step, so averages hide the tail and percentiles should be measured client-side. Speculation also leaves the prefill stage alone: AWS found that prefill latency is neither improved nor degraded (([aws.amazon.com](#))), and Google's Vertex team notes an increase in TTFT may be observed (([lmsys.org](#))).

## Where the extra work happens

Three costs hide inside a headline gain. First, proposer compute: the draft-to-target single-pass time ratio was about 12.5% for a 70B target and an estimated 37.5% for an 8B target in one vLLM study (([arxiv.org](#))), so small targets leave little room for a separate drafter. Second, verification of rejected tokens: the target still processes every proposed position. Third, scheduling: Google notes some tokens are harder and some easier to generate (([research.google](#))), so wasted verification work is uneven across a batch. Google Cloud describes the net effect as slightly increased compute costs and operational complexity, worthwhile for latency-sensitive workloads (([cloud.google.com](#))).

# Key Changes: Lossless Sampling Versus Heuristic Acceptance

## What the guarantee covers

Chen et al. at DeepMind introduced speculative sampling as a modified rejection sampling scheme which preserves the distribution of the target model (([arxiv.org](#))), and a later survey summarizes that both original papers theoretically proved identical output distributions to the target (([arxiv.org](#))). Engine documentation echoes this: vLLM Speculators states that accepted tokens are guaranteed to come from the same distribution (([docs.vllm.ai](#))), and Hugging Face says the verification forward pass ensures the same outputs are generated as if the large model had run alone (([huggingface.co](#))).

The guarantee has a stated scope. vLLM calls speculative sampling theoretically lossless up to the precision limits of hardware numerics (([docs.vllm.ai](#))) and says its implementation is algorithmically validated to be lossless, yet it does not guarantee stable token log probabilities (([docs.vllm.ai](#))), and changes in batch size may cause variations in logprobs and output probabilities (([docs.vllm.ai](#))). The practical reading: speculative and non-speculative runs are distributionally equivalent under the algorithm, not byte-identical.

## Heuristic variants change the distribution

Several methods trade exactness for acceptance, so an equivalence claim must be checked per method:

- **Lenient acceptance:** the original paper's appendix describes a lenience parameter; on T5-XXL it produced improvement factors of 2.5X, 3.1X, 3.6X and 5X as leniency increased (([arxiv.org](#))), bought by altering the output distribution.
- **Medusa typical acceptance:** the authors write that they do not insist on an exact correspondence between the output and language model distribution (([arxiv.org](#))).
- **Exact variants:** EAGLE states that preservation of the output distribution is theoretically guaranteed (([arxiv.org](#))).
- **Cross-tokenizer drafts:** vLLM's token-level intersection mode allows draft models from a different model family with a different tokenizer (([docs.vllm.ai](#))), but probabilistic draft sampling is not yet supported in that mode (([docs.vllm.ai](#))), so check the sampling configuration before assuming equivalence.

## An equivalence test that survives numerics

Run-to-run differences exist even without speculation. Thinking Machines attributes nondeterminism in nearly all inference endpoints to varying load (([thinkingmachines.ai](#))) and explains that decreasing batch size eventually leads to more cores than batch elements, changing reduction strategies (([thinkingmachines.ai](#))). vLLM does not guarantee reproducibility by default (([docs.vllm.ai](#))) and provides it only on the same hardware (([docs.vllm.ai](#))). A batched speculative decoding study verified equivalence with deterministic greedy decoding to eliminate sampling variance (([arxiv.org](#))) and attributed residual mismatch to floating-point non-determinism.

Keep the repeated non-speculative baseline as a numerical-stability control under matched hardware, engine configuration and load. Use exact-match comparisons separately for deterministic greedy decoding ([greedy evaluation](#)). For stochastic sampling, use a pre-specified distribution test with enough independent samples to detect a declared discrepancy at a declared significance and power; vLLM separately validates rejection-

sampler convergence to the target distribution ([validation documentation](#)). Add a held-out task-quality gate for both lossless and lossy methods. Do not treat two-run exact match or aggregate quality parity as establishing stochastic distributional equivalence.

## Key Changes: Memory, Compute and Strategy Choice

### Memory is paid twice

The target's weights and KV cache are unchanged and the proposer adds its own. vLLM's project blog notes the memory manager now handles the KV cache for both the draft and target models ([vllm.ai](#)), and NVIDIA's NIM documentation lists a separate draft model with its own weights and KV cache ([docs.nvidia.com](#)). IBM's speculator design avoids duplicating the full KV-cache across candidates ([research.ibm.com](#)). In one vLLM study, per-token KV cache rose from 144 KiB to 256 KiB for Qwen3-8B with a 0.6B draft, and from 320 KiB to 352 KiB for Llama3-70B with a 1B draft ([arxiv.org](#)). On smaller accelerators the weights alone matter: an AWS walkthrough with a 70B target and 7B draft found the two models need almost 200 GB of device memory for the weights ([aws.amazon.com](#)). BentoML warns that on a single GPU the draft can quickly squeeze out space for other tasks such as batching ([bentoml.com](#)).

A worked calculation (Hypothetical Example; the arithmetic and the 65 GB assumption are this report's, the per-token figures are the cited ones) shows the effect. Take a 70B model in 8-bit weights on one H200, which NVIDIA lists at 141 GB of HBM3e at 4.8 TB/s ([www.nvidia.com](#)). Assume 65 GB remains for KV cache before a roughly 2.5 GB, 1B-parameter FP16 draft is loaded. At 320 KiB per token the pool holds about 198,000 tokens. With the draft, 62.5 GB at 352 KiB per token holds about 173,000 tokens, a **12.6% reduction** in concurrent-context capacity. Whether that matters depends on how close the fleet runs to its KV limit.

### Strategy matrix

Table 1 below summarizes candidate strategies on the dimensions that drive cost: extra state, output equivalence and evidence. Speedups are the source's own, on the source's hardware, and are not comparable across rows.

| Strategy                       | Extra state                                                                                                                                                                                                                         | Output equivalence                                                                                               | Reported evidence and caveats                                                                                                                                                                                                |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Separate draft model</b>    | A second model's weights and KV cache ( <a href="#">docs.nvidia.com</a> ), plus more serving resources to host and manage ( <a href="#">www.lmsys.org</a> ). Must share the vocabulary by default ( <a href="#">docs.vllm.ai</a> ). | Lossless under rejection sampling; heterogeneous-tokenizer mode needs checking ( <a href="#">docs.vllm.ai</a> ). | NVIDIA: 3.55x with a 1B draft versus 2.63x with an 8B draft on one H200, batch size 1 ( <a href="#">developer.nvidia.com</a> ). PyTorch measured only about a 1.3x boost for one small pair ( <a href="#">pytorch.org</a> ). |
| <b>N-gram or prompt lookup</b> | No model; matches n-grams in the prompt ( <a href="#">docs.vllm.ai</a> ).                                                                                                                                                           | Verified like other proposers, so lossless under the same rule.                                                  | The originator reports 2x-4x on input-grounded tasks ( <a href="#">github.com</a> ); a benchmark survey notes it excels where input and output are highly similar.                                                           |

| Strategy            | Extra state                                                                                                            | Output equivalence                                                                                                                                       | Reported evidence and caveats                                                                                                                                                                                                                   |
|---------------------|------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EAGLE family</b> | A small trained head: 2-5% of the target model per Google's Vertex team ( <a href="https://lmsys.org">lmsys.org</a> ). | Distribution preservation stated as theoretically guaranteed.                                                                                            | 1.38x throughput at batch size 64 in SGLang for EAGLE-3 ( <a href="https://arxiv.org">arxiv.org</a> ); gains fell from over 60% at batch size 1 to 26% at batch size 16 for a Kimi K2.5 head ( <a href="https://pytorch.org">pytorch.org</a> ). |
| <b>Medusa heads</b> | Extra decoding heads; five were sufficient at most.                                                                    | Medusa-1 is reported as not compromising generation quality ( <a href="https://arxiv.org">arxiv.org</a> ); typical acceptance is not distribution-exact. | Over 2.2x for Medusa-1, evaluated at batch size 1.                                                                                                                                                                                              |
| <b>Built-in MTP</b> | Multi-token-prediction modules trained with the model, repurposable for speculation.                                   | Same verification rule applies.                                                                                                                          | DeepSeek reports 85% to 90% second-token acceptance ( <a href="https://arxiv.org">arxiv.org</a> ); vLLM supports it only for model families with MTP ( <a href="https://docs.vllm.ai">docs.vllm.ai</a> ).                                       |

The table shows that "extra state" ranges from nothing (n-gram) to a second full model, and that the strongest evidence for each method sits at batch size 1 or in a vendor's own harness. vLLM's guidance is that EAGLE, MTP, draft models and PARD provide the best latency reduction ([docs.vllm.ai](https://docs.vllm.ai)), but also that real gains depend on model family, traffic pattern, hardware and sampling settings.

## Selecting a draft or speculator

Selection should be an ordered filter, not a leaderboard:

- **Tokenizer and vocabulary:** Standard Hugging Face assisted generation uses a shared tokenizer ([huggingface.co](https://huggingface.co)); Universal Assisted Generation supports different tokenizers ([Hugging Face](https://huggingface.co)). Benchmark the chosen mode, including token translation and its sampling constraints, and TensorRT-LLM warns that a mismatch makes acceptance extremely low and performance regressed ([nvidia.github.io](https://nvidia.github.io)).
- **Size ratio:** Hugging Face reports meaningful speedups are typically seen when the assistant is at least 50-100 times smaller ([huggingface.co](https://huggingface.co)); Baseten's guide says at least ten times smaller by parameter count ([www.baseten.co](https://www.baseten.co)). These rules of thumb differ by 5x, so measure.
- **Smaller is not always better:** in AWS's Trainium test a 0.6B draft was faster but had roughly 60% lower acceptance, cancelling the savings, and the 1.7B draft with 7 speculative tokens gave the best balance ([aws.amazon.com](https://aws.amazon.com)) ([aws.amazon.com](https://aws.amazon.com)). Together AI states the same trade-off: larger speculators yield higher acceptance rates due to their capacity ([www.together.ai](https://www.together.ai)), but draft more slowly.
- **Trained speculators and their cost:** Red Hat reports two to three times the throughput gains for trained speculators ([developers.redhat.com](https://developers.redhat.com)), and Together reports a trained speculator lifting Kimi from about 150 to over 270 tokens per second ([www.together.ai](https://www.together.ai)). Training needs internal hidden states from the target model ([docs.vllm.ai](https://docs.vllm.ai)); a Kimi K2.5 EAGLE-3 draft took 1500 H200 GPU-hours across 600,000 training samples and 6 billion tokens ([pytorch.org](https://pytorch.org)). Snowflake trained its gpt-oss speculators for 3,000 steps (approximately one epoch) on Ultrachat and Magicoder using 16 H200 GPUs: three elapsed hours for gpt-oss-20B and 21 elapsed hours for gpt-oss-120B, equivalent to 48 and 336 H200

GPU-hours, respectively (16 GPUs multiplied by elapsed hours) ([www.snowflake.com](http://www.snowflake.com)). These reports concern different target models and training workloads; do not treat the figures as a controlled training-efficiency comparison.

- **Engine support:** TensorRT-LLM's EAGLE 3 mode does not support tree structures for draft sequences ([nvidia.github.io](https://nvidia.github.io)), so the same method name behaves differently across engines.
- **Context length:** BentoML found out-of-the-box EAGLE 3 heads were not optimized for long contexts ([bentoml.com](https://bentoml.com)).
- **Alignment:** DistillSpec uses knowledge distillation to better align the draft with the target.

## Key Changes: Concurrency Moves the Break-Even

### Why the sign of the benefit depends on load

Speculation spends spare arithmetic to save memory-bandwidth-bound steps. NVIDIA's NIM documentation says it helps most when the GPU has spare compute, for example at low-to-moderate concurrency ([docs.nvidia.com](https://docs.nvidia.com)), and warns that under saturated load it can fail to improve throughput, and can reduce it ([docs.nvidia.com](https://docs.nvidia.com)). NVIDIA's co-design analysis quantifies one side of this: with a draft length of seven, one-eighth of the batch size is needed to become compute bound compared with no speculation ([developer.nvidia.com](https://developer.nvidia.com)). Baseten's guide states the engineering consequence: speculation must be dynamically disabled as compute is too saturated to afford verification ([www.baseten.co](http://www.baseten.co)), and it adds that adopting EAGLE for lower latency requires reduced batch sizes, lowering throughput and increasing cost ([www.baseten.co](http://www.baseten.co)).

### Measured degradation with load

The measurements agree on direction and differ on the threshold. AMD's MI300X study with a 34B target and a 1.1B draft found graph-mode slowdown from batch size 32 ([rocm.blogs.amd.com](https://rocm.blogs.amd.com)). PyTorch and IBM observed throughput reduction beyond a batch size of 64 for 7B to 20B models with speculator heads ([pytorch.org](https://pytorch.org)). vLLM's project blog measured a 1.5x speedup at QPS 1 for Llama3-70B on four H100s ([vllm.ai](https://vllm.ai)) but a 1.4x slowdown at high QPS ([vllm.ai](https://vllm.ai)). BentoML's informal H100 tests on one GPU found total throughput plateaued earlier than the baseline ([bentoml.com](https://bentoml.com)); it labels these results as informal and for reference only ([bentoml.com](https://bentoml.com)). Snowflake saw a CPU-side method become worse at a concurrency of 64 before optimization ([www.snowflake.com](http://www.snowflake.com)).

Research explains the mechanism: as the system becomes compute-bound, the overhead spent proposing and verifying rejected tokens becomes more costly ([arxiv.org](https://arxiv.org)), which is why closed-loop controllers that disable speculation under load have been proposed. TensorRT-LLM documents the static-configuration consequence: speedups are only observable at low batch sizes ([nvidia.github.io](https://nvidia.github.io)). Gains are not uniformly fragile, though. AMD's 405B study found the speedup declined only slightly with increasing request rates ([rocm.blogs.amd.com](https://rocm.blogs.amd.com)), which shows the threshold depends on the model, hardware and engine.

## The long-context exception

MagicDec provides a counterweight: for moderate-to-long sequences, the cost of loading the KV cache dominates as batch size grows, so the high-batch limitation only applies to very short sequences (([arxiv.org](#))). It reports up to 2.51x speedup for LLaMA-3.1-8B at batch sizes 32 to 256 (([arxiv.org](#))). AMD likewise found speculation can boost performance further when the input sequence is longer (([rocm.blogs.amd.com](#))). The region where speculation helps at scale is long-context, KV-bound decode on high-FLOPS hardware, which differs from short-prompt chat at moderate load.

## Implementation Considerations and Process Changes

### Benchmark contract

A pilot is comparable only if both arms obey the same contract. The fields below belong in the results template.

- **Models:** target and draft or speculator checkpoint hashes, tokenizer hash, quantization format.
- **Software:** engine name and version, speculative method, and speculative-token setting (vLLM's `num_speculative_tokens` is the number of tokens proposed per step (([docs.vllm.ai](#)))), sampling parameters.
- **Hardware:** GPU model and count, interconnect, driver and CUDA version, tensor-parallel degree.
- **Load:** closed-loop concurrency levels or open-loop arrival rates, prompt and output length distributions, prompt-class mix.
- **Latency definitions:** TTFT, ITL and end-to-end latency at p50, p95 and p99, with the SLO stated before the run.
- **Throughput definition:** NVIDIA's own throughput figure is inclusive of time to generate the first token (([developer.nvidia.com](#))), so state whether prefill is included.
- **Equivalence:** baseline numerical-stability repeat, deterministic greedy comparison, stochastic distribution-test design and result (including sample size, significance and power), and held-out quality-gate outcome.

The baseline is the same target, engine and hardware with speculation disabled, tuned to its own best batch limit. NVIDIA's documentation advises measuring acceptance and throughput on representative traffic before committing to a method (([docs.nvidia.com](#))).

### Telemetry and slices

Record vLLM's acceptance rate and mean acceptance length, plus rejected-token work, sliced by:

- **Prompt class:** code, extraction, long-form generation, multi-turn chat, retrieval-grounded answers. AWS found open-ended prompts gave no consistent benefit while structured, repetitive prompts did (([aws.amazon.com](#))).
- **Output position:** acceptance usually decays with draft depth, so an average hides which depth to cut.
- **Sampling mode:** Chen et al. observed the acceptance rate depends on the application and the decoding method (([arxiv.org](#))).
- **Concurrency bucket:** net speedup at 1, 4, 16 and 64 concurrent requests, not only the peak.

Domain effects are large. A domain-specific draft study finds the acceptance rate of the generic draft model drops significantly due to domain shift (([arxiv.org](https://arxiv.org))). Fireworks reported a generic draft model increasing latency by 1.5x on a specialized customer workload (([fireworks.ai](https://fireworks.ai))). Snowflake reports acceptance averaging 44%-50% for its gpt-oss speculators (([www.snowflake.com](https://www.snowflake.com))) and notes the best speculator balances acceptance against computational resources for the specific model and workload (([www.snowflake.com](https://www.snowflake.com))).

## Results template and calculator

The idealized speedup grid appears as Table 3 in the Data Analysis section. The fleet-level calculation that matters for budgeting follows this chain: measure SLO-qualified accepted tokens per second per replica (excluding rejected speculative work and SLO-missing requests), divide peak demand by it, round up to **whole replicas**, multiply by GPUs per replica and the GPU-hour price, and divide by tokens produced. A minimal calculator:

```
from math import ceil

def tokens_per_step(alpha_per_position, k):
    # Theoretical worksheet: independent per-position acceptance.
    return sum(alpha_per_position**i for i in range(k + 1))

def net_speedup(alpha_per_position, k, c, v=0.0):
    # c: draft step / baseline target step.
    # v: extra verification time per proposed token / baseline target step.
    return tokens_per_step(alpha_per_position, k) / (1 + k * (c + v))

def measured_decode_speedup(mean_acceptance_length, step_cost_ratio):
    # step_cost_ratio: measured speculative iteration time / baseline decode-step time.
    return mean_acceptance_length / step_cost_ratio

# Fixed-k worksheet algebra, not a fitted production threshold:
# L = 1 + k*r; modeled step-cost ratio = 1 + k*(c+v).
# Break-even: L > 1 + k*(c+v), equivalently r > c+v.
# At k=4, c=0.05, v=0: L > 1.2 and r > 0.05.
# The separate theoretical alpha threshold is approximately 0.17.

def cost_per_million(gpus_per_replica, gpu_hour_usd,
                    qualified_tokens_per_s, demand_tokens_per_s,
                    actual_qualified_fleet_tokens_per_s):
    if min(gpus_per_replica, gpu_hour_usd, qualified_tokens_per_s,
          demand_tokens_per_s, actual_qualified_fleet_tokens_per_s) <= 0:
        raise ValueError("All inputs must be positive")
    replicas = ceil(demand_tokens_per_s / qualified_tokens_per_s)
    capacity = replicas * qualified_tokens_per_s
    if actual_qualified_fleet_tokens_per_s > min(demand_tokens_per_s, capacity):
        raise ValueError("Served qualified throughput exceeds demand or capacity")
    fleet_cost = (replicas * gpus_per_replica * gpu_hour_usd * 1e6
                  / (3600 * actual_qualified_fleet_tokens_per_s))
    return replicas, fleet_cost
```

For the price input, list prices vary widely. As retrieved on 2026-10-18, CoreWeave's undated pricing page listed an 8-GPU H100 node at \$49.24 per hour on demand (([www.coreweave.com](http://www.coreweave.com))) and an 8-GPU H200 node at \$50.44 per hour (([www.coreweave.com](http://www.coreweave.com))), roughly \$6.16 and \$6.31 per GPU-hour. Use the organization's actual fully loaded rate, including power and cooling for owned hardware. Fireworks makes the broader point that a volume-optimized 70B configuration on eight GPUs can be 4x cheaper per token than one tuned for single-request speed (([fireworks.ai](http://fireworks.ai))), the regime where speculation typically helps.

## Worked example (Hypothetical Example)

Table 2 applies the calculator to invented numbers to show the mechanics. Assume 2 GPUs per replica at \$4.00 per GPU-hour (\$8.00 per replica-hour) and a peak demand of 20,000 SLO-qualified tokens per second. None of the throughput figures below is a measurement.

| Scenario (hypothetical)                                                          | Baseline qualified tokens/s per replica | Speculative qualified tokens/s per replica | Whole replicas, baseline / speculative | Cost per million accepted tokens, baseline / speculative |
|----------------------------------------------------------------------------------|-----------------------------------------|--------------------------------------------|----------------------------------------|----------------------------------------------------------|
| <b>A: Loose ITL target, high concurrency, EAGLE-class speculator (1.25x net)</b> | 2,500                                   | 3,125                                      | 8 / 7                                  | \$0.889 / \$0.778                                        |
| <b>B: Same load, separate draft model (0.95x net, extra KV pressure)</b>         | 2,500                                   | 2,375                                      | 8 / 9                                  | \$0.889 / \$1.000                                        |
| <b>C: Tight p95 ITL target, low per-replica batch (1.75x net)</b>                | 800                                     | 1,400                                      | 25 / 15                                | \$2.778 / \$1.667                                        |

Scenario C is where speculation earns its keep: a tight ITL target forces the baseline to run small batches with poor utilization, so spare compute exists. Scenario B is the negative case: a heavier proposer costs more than doing nothing. Rounding to whole replicas matters: in scenario A the 7-replica figure comes from  $20,000 / 3,125 = 6.4$ , so the realized fleet cost per million tokens is \$0.78 rather than \$0.711. The same rounding moves scenario B to \$1.00 and scenario C to \$1.67. Energy should be added where wall-power measurements exist. A published study cautions that lower inference time does not always correlate with proportional energy savings (([arxiv.org](http://arxiv.org))), though its measurements were taken at batch size 1 (([arxiv.org](http://arxiv.org))).

## Production gates, drift monitoring and rollback

- **Entry gate:** net SLO-qualified cost per million accepted tokens lower than baseline by more than the measurement noise, at the production concurrency distribution, on at least three prompt classes.
- **Canary:** route a small share of traffic to a speculative replica and compare acceptance, ITL percentiles and quality-gate scores with a matched baseline replica.
- **Drift monitors:** track rolling mean acceptance length, including the bonus token, for each prompt class ([vLLM metric definition](#)). Compare it with the measured speculative-iteration/baseline-step cost ratio using `measured_decode_speedup`; alert when that ratio no longer shows a decode benefit, and recheck whole-fleet SLO-qualified cost. Keep the theoretical alpha threshold separate from the observed accepted/proposed fraction.

- **Revalidate** after fine-tuning, tokenizer changes, engine upgrades, a draft or speculator swap, or a traffic-mix change. Online adaptation of a draft to the query distribution raised token acceptance by 0.1 to 0.65 in one study, which cuts both ways for drift.
- **Rollback**: keep a speculation-disabled configuration deployable by flag; TensorRT-LLM documents that speculation cannot be disabled dynamically in its mode ([nvidia.github.io](https://nvidia.github.io)), so plan rollback there as a replica-level swap.

## Negative Results: When Speculation Is Not Worth It

The following conditions have published support as cases where speculation yields little or nothing. None is a flaw of a specific product; each follows from the arithmetic above.

- **High concurrency on short contexts**: one study measured EAGLE's speedup falling from 1.73x at batch size 1 to 1.21x at batch size 128 while absolute throughput rose ([arxiv.org](https://arxiv.org)). A 1.21x gain with a 12.6% KV capacity loss can net out to zero.
- **Naive static configuration under load**: a serving study measured a 14% latency increase from even one speculated token at high request rates ([arxiv.org](https://arxiv.org)).
- **Tokenizer mismatch or poorly aligned drafts**: acceptance extremely low and performance regressed ([nvidia.github.io](https://nvidia.github.io)). Fireworks' generic-draft latency penalty of 1.5x is the same failure on a specialized workload ([fireworks.ai](https://fireworks.ai)).
- **Draft cost too high relative to the target**: with  $c = 0.10$  and  $\alpha = 0.5$  the formula's best case is only 1.46x, at  $k = 2$ .
- **Open-ended generation**: AWS saw no consistent benefit on open-ended prompts in its own test.
- **Over-deep speculation**: SGLang's documentation notes that increasing speculation range risks rejection cascades ([docs.sglang.io](https://docs.sglang.io)), and wider candidate trees increase memory and compute consumption ([docs.sglang.io](https://docs.sglang.io)) and GPU memory usage ([docs.sglang.io](https://docs.sglang.io)); reducing them frees headroom for activations and CUDA graph buffers ([docs.sglang.io](https://docs.sglang.io)).
- **Resource contention**: Baseten found draft and target models contending on one GPU ran at half speed until execution was synchronized ([baseten.co](https://baseten.co)).
- **Unsupported serving modes**: Hugging Face assisted generation does not support batched inputs ([huggingface.co](https://huggingface.co)) and launched limited to a batch size of one ([huggingface.co](https://huggingface.co)), so it is unsuited to multi-tenant serving.
- **No whole-fleet saving**: if speculation does not lower SLO-qualified whole-fleet cost after accounting for its memory footprint, reject the configuration. An already-met SLO alone is not a rejection criterion; scenario A illustrates the worksheet's loose-ITL case.
- **Energy-constrained sites**: proportional energy savings are not guaranteed.

A negative result should be published with the same detail as a positive one: engine version, hardware, concurrency, prompt class and measured acceptance. A table of failing cells is the most useful output of a pilot because it marks the edge of the operating region.

## Data Analysis and Evidence

### Idealized speedup grid

Table 3 below is computed directly from the published formula with  $c = 0.05$ , the upper edge of the draft-cost range in the original experiments, and  $v = 0$ . The right-hand columns show the  $k = 4$  configuration under two stress assumptions of this report:  $c = 0.10$ , and  $v = 0.25$  of one baseline target-step time per proposed token. At  $k = 4$ , the latter adds  $k \cdot v = 1.0$  baseline target-step time per iteration: 100% extra verification time, in addition to the 20% draft cost.

| Modeled per-position acceptance probability (alpha) | k = 1 | k = 2 | k = 3 | k = 4 | k = 5 | k = 7 | k = 4, c = 0.10 | k = 4, c = 0.05, v = 0.25 |
|-----------------------------------------------------|-------|-------|-------|-------|-------|-------|-----------------|---------------------------|
| 0.5                                                 | 1.43x | 1.59x | 1.63x | 1.61x | 1.57x | 1.48x | 1.38x           | 0.88x                     |
| 0.6                                                 | 1.52x | 1.78x | 1.89x | 1.92x | 1.91x | 1.82x | 1.65x           | 1.05x                     |
| 0.7                                                 | 1.62x | 1.99x | 2.20x | 2.31x | 2.35x | 2.33x | 1.98x           | 1.26x                     |
| 0.8                                                 | 1.71x | 2.22x | 2.57x | 2.80x | 2.95x | 3.08x | 2.40x           | 1.53x                     |
| 0.9                                                 | 1.81x | 2.46x | 2.99x | 3.41x | 3.75x | 4.22x | 2.93x           | 1.86x                     |

Three readings follow. First, the optimum  $k$  rises with acceptance: at  $\alpha = 0.5$  the grid peaks at  $k = 3$ , while at 0.9 it is still rising at  $k = 7$ , so one global speculative-token setting is a compromise across prompt classes. Second, doubling  $c$  from 0.05 to 0.10 costs 0.2x to 0.5x at  $k = 4$ , so a draft at 10% of the target's step time is materially less valuable than one at 5%. Third,  $v = 0.25$  per proposed token (100% extra verification time per iteration at  $k = 4$ ) removes about 0.7x to 1.6x everywhere and turns  $\alpha = 0.5$  into a net loss. Theoretical break-even per-position  $\alpha$  at  $k = 4$  and  $c = 0.05$  rises from 0.17 at  $v = 0$  to 0.38 at  $v = 0.1$ , 0.58 at  $v = 0.25$  and 0.78 at  $v = 0.5$ , all computed from the same formula. The formula assumes independent per-position acceptances; do not substitute the aggregate accepted/proposed telemetry fraction for  $\alpha$ . Use the calculator's measured acceptance-length path for operational comparisons.

### Published measurements, with context

- **Original task observations:** 2.6X on T5-XXL English-German translation at temperature 1; the 2X-3X range is an observation on those tasks, not a forecast.
- **Batch size 1, vendor harness:** 3.55x for Llama 3.3 70B with a 1B draft on one H200 ([developer.nvidia.com](https://developer.nvidia.com)); NVIDIA's earlier post headlines up to 3.6x ([developer.nvidia.com](https://developer.nvidia.com))).
- **Draft pair dependence:** PyTorch notes runtime performance varies depending on the generated text ([pytorch.org](https://pytorch.org)), and its Llama-7B with a small draft pair obtained about a 1.3x boost ([pytorch.org](https://pytorch.org))).

- **Production claims:** IBM reports cutting the latency of serving its Granite 20B code model in half while quadrupling throughput (([research.ibm.com](https://research.ibm.com))), IBM cites latency lowered by two to three times (([research.ibm.com](https://research.ibm.com))), and PyTorch and IBM report nearly twice as fast for Llama2 13B chat (([pytorch.org](https://pytorch.org))). These are vendor claims on vendor configurations.
- **Cost-linked:** Red Hat's gpt-oss-120b study reports peak output throughput of 2574 tokens/s on ShareGPT (([developers.redhat.com](https://developers.redhat.com))) and a 19.4% reduction in cost per million output tokens on one workload (([developers.redhat.com](https://developers.redhat.com))); the cost figure is workload-specific.
- **Low-load results:** Snowflake's Llama 3.1 70B benchmark used an arrival rate of 0.5 requests per second (([www.snowflake.com](https://www.snowflake.com))), so its speedups are low-load results.
- **Task variation:** a benchmark survey defines six subtasks including translation and summarization (([arxiv.org](https://arxiv.org))), evaluated on Vicuna-7B at a batch size of 1.
- **Production adoption:** Google states speculative decoding enables AI Overviews in Google Search to produce results faster than before (([research.google](https://research.google))), which shows deployment viability, not private-fleet economics.
- **Documentation is not evidence:** vLLM Speculators describes the technique as speeding up inference without changing the output (([docs.vllm.ai](https://docs.vllm.ai))) and shows how to serve a pre-trained speculator (([docs.vllm.ai](https://docs.vllm.ai))), but it documents deployment and does not establish economics for a specific model pair.

The evidence is thick at batch size 1 and thin at production concurrency with SLO-qualified accounting. No public source reviewed provides a fleet break-even for a private target, which is the gap the worksheet in this report is designed to fill.

## Implications and Future Directions

For procurement and capacity planning, compare measured SLO-qualified throughput, proposer footprint and whole-fleet cost at the relevant load. A binding ITL target is one favorable use case, as scenario C illustrates; scenario A also reduces the fleet with a loose ITL target. Include the high-concurrency MoE results discussed above and MagicDec's long-context, large-batch results in the pilot matrix ([MagicDec](#)). GPU Smith, as an independent advisor on inference optimization and workload sizing, would frame a pilot as a sizing question first and an optimization second, on a vendor-neutral measurement contract.

Memory headroom becomes a first-class input. On GPUs with large HBM, such as the H200 at 141 GB (([www.nvidia.com](https://www.nvidia.com))), the draft footprint is easier to absorb; on smaller parts the 12.6% KV capacity loss in the worked example can force a larger tensor-parallel degree and erase the gain.

Three developments deserve monitoring as of October 2026. Trained speculators are becoming packaged artifacts: vLLM publishes EAGLE draft models on the Hugging Face hub (([docs.vllm.ai](https://docs.vllm.ai))), which lowers the cost of trying one. MTP-style heads ship with some models, but vLLM supports them only for families that include them (([docs.vllm.ai](https://docs.vllm.ai))), and its MTP drafter can hold a quantized copy of the output head whose memory must be budgeted (([docs.vllm.ai](https://docs.vllm.ai))). Concurrency-based switching is already documented in vLLM through `num_speculative_tokens_per_batch_size`: its example uses K=3, K=1 and K=0 at increasing concurrency, with K=0 producing no drafts ([dynamic speculation](#)). Pin and validate the installed version: the documented examples set `VLLM_USE_V2_MODEL_RUNNER=0`, the page distinguishes CUDA-graph support by runner, and the

option is incompatible with data parallelism. This configured concurrency policy is separate from automatic feedback-driven tuning. Keep the earlier TensorRT-LLM 1.3.0rc7 rollback limitation scoped to that documented mode and version.

The measurement gap is the largest open item. Energy per accepted token on data-center GPUs at production batch sizes was not found in the sources reviewed, and the only energy study located used workstation-class GPUs at batch size 1. Teams with wall-power meters can contribute the most useful data by publishing it.

## Frequently Asked Questions (FAQs)

### Does speculative decoding reduce inference cost?

It can, but not by default. Cost falls when net speedup at production concurrency exceeds the added GPU footprint and the pilot meets its SLO. Red Hat measured a 19.4% lower cost per million output tokens on one workload (([developers.redhat.com](https://developers.redhat.com))), while vLLM's project blog measured a slowdown at high request rates (([vllm.ai](https://vllm.ai))).

### How much speedup should be expected from a given acceptance rate?

For the modeled per-position alpha, use the published formula: at  $c = 0.05$  and  $k = 4$ ,  $\alpha = 0.6$  gives 1.92x and  $\alpha = 0.8$  gives 2.80x in Table 3, before the report's incremental verification term. For observed telemetry, use the calculator's measured mean acceptance length and step-cost ratio instead. The fixed-k worksheet gives  $r > c+v$ , or  $r > 0.05$  for  $c = 0.05$  and  $v = 0$ ; approximately 0.17 is the separate modeled alpha threshold.

### How should a draft model be selected?

Check tokenizer requirements for the selected engine mode; the standard Hugging Face assisted-generation path uses a shared tokenizer to avoid token re-encoding (([huggingface.co](https://huggingface.co))), prefer the smallest draft that holds acceptance (([aws.amazon.com](https://aws.amazon.com))), and consider trained speculators (([www.together.ai](https://www.together.ai))). Measure acceptance on the actual prompt classes, since acceptance depends on the application and decoding method.

### Where does speculation sit among private LLM inference cost optimizations?

It is a latency lever that can become a cost lever. Baseten notes that adopting EAGLE for lower latency requires reduced batch sizes, lowering throughput and increasing cost (([www.baseten.co](https://www.baseten.co))), while Fireworks observes that a volume-optimized 70B configuration on eight GPUs can be 4x cheaper per token than one tuned for single-request speed (([fireworks.ai](https://fireworks.ai))). Compare speculation against batching, parallelism and capacity changes on the same SLO-qualified metric before committing.

### Is speculative decoding better than increasing batch size?

They compete for the same spare compute. NVIDIA's analysis shows speculation reaches the compute-bound point at a much smaller batch (([developer.nvidia.com](https://developer.nvidia.com))), and vLLM scopes the method to medium-to-low request rates (([docs.vllm.ai](https://docs.vllm.ai))). At saturation, batching generally wins; at tight ITL targets, speculation can win.

## What is the cost per token for a self-hosted model with speculation?

Divide fully loaded GPU-hours by accepted output tokens from SLO-meeting requests, then multiply by the GPU-hour price and one million, on whole replicas. The calculator and Table 2 show the arithmetic.

## When is it not worth it?

At high concurrency on short contexts, with mismatched tokenizers, with a heavy draft, on open-ended prompts, or when draft memory reduces KV capacity enough to force a larger replica (see the negative-results section).

## Does it change the model's outputs?

Under rejection-sampling verification the distribution is the target's, subject to hardware numerics ((docs.vllm.ai)); heuristic acceptance schemes do not preserve it.

## Conclusion

Speculative decoding is a conditional cost reduction, not a free speedup. The published algorithm guarantees the target distribution under rejection sampling, the original experiments show 2X-3X gains at batch size 1, and vendor measurements show similar gains in low-concurrency settings. The same sources and several engineering reports also show the gain shrinking as load rises, reversing for short contexts at high concurrency, and costing memory that the target may need for its KV cache.

The decision rule that follows is to measure SLO-qualified accepted tokens per second per whole replica, with and without speculation, at the production concurrency and prompt mix, and to adopt only where cost per million accepted tokens falls by more than the measurement noise. Pilot a lossless, trained speculator first where a tight p95 ITL binds, keep n-gram methods for input-grounded workloads, slice acceptance by prompt class and output position, and publish negative cells alongside positive ones. Revalidate whenever the model, tokenizer, engine, speculator or traffic mix changes, because acceptance is a property of the workload and not of the technique.

## Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://arxiv.org/abs/2211.17192>
2. <https://research.google/blog/looking-back-at-speculative-decoding>
3. <https://developer.nvidia.com/blog/boost-llama-3-3-70b-inference-throughput-3x-with-nvidia-tensorrt-llm-speculative-decoding>
4. <https://pytorch.org/blog/torchspec-speculative-decoding-training-at-scale>
5. <https://vllm.ai/blog/spec-decode>
6. [https://docs.vllm.ai/en/latest/features/speculative\\_decoding](https://docs.vllm.ai/en/latest/features/speculative_decoding)
7. <https://developers.redhat.com/articles/2026/04/16/performance-improvements-speculative-decoding-vllm-gpt-oss>
8. <https://arxiv.org/pdf/2211.17192>
9. <https://arxiv.org/html/2408.11049>

10. <https://developer.nvidia.com/blog/co-designing-ai-models-using-speculative-decoding-for-faster-llm-inference>
11. <https://research.ibm.com/blog/speculative-decoding>
12. <https://cloud.google.com/blog/topics/developers-practitioners/five-techniques-to-reach-the-efficient-frontier-of-llm-inference>
13. <https://arxiv.org/html/2211.17192>
14. [https://docs.vllm.ai/en/stable/features/speculative\\_decoding/acceptance\\_metrics](https://docs.vllm.ai/en/stable/features/speculative_decoding/acceptance_metrics)
15. <https://arxiv.org/pdf/2401.09670>
16. <https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html>
17. <https://aws.amazon.com/blogs/machine-learning/accelerating-decode-heavy-llm-inference-with-speculative-decoding-on-aws-tranium-and-vllm>
18. <https://lmsys.org/blog/2025-12-01-eagle3-vertex>
19. <https://arxiv.org/html/2601.11580v1>
20. <https://arxiv.org/abs/2302.01318>
21. <https://arxiv.org/html/2401.07851v3>
22. [https://docs.vllm.ai/projects/speculators/en/stable/user\\_guide/getting\\_started](https://docs.vllm.ai/projects/speculators/en/stable/user_guide/getting_started)
23. [https://huggingface.co/docs/transformers/main/en/llm\\_optims](https://huggingface.co/docs/transformers/main/en/llm_optims)
24. <https://arxiv.org/html/2401.10774>
25. <https://arxiv.org/html/2401.15077>
26. [https://docs.vllm.ai/en/latest/features/speculative\\_decoding/draft\\_model](https://docs.vllm.ai/en/latest/features/speculative_decoding/draft_model)
27. <https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference>
28. <https://docs.vllm.ai/en/latest/usage/reproducibility.html>
29. <https://arxiv.org/html/2510.22876v2>
30. <https://docs.nvidia.com/nim/large-language-models/latest/advanced-use-cases/speculative-decoding.html>
31. <https://aws.amazon.com/blogs/machine-learning/faster-llms-with-speculative-decoding-and-aws-inferentia2>
32. <https://bentoml.com/llm/inference-optimization/speculative-decoding>
33. <https://www.nvidia.com/en-us/data-center/h200>
34. <https://pytorch.org/blog/accelerating-generative-ai-2>
35. [https://docs.vllm.ai/en/latest/features/speculative\\_decoding/n\\_gram](https://docs.vllm.ai/en/latest/features/speculative_decoding/n_gram)
36. <https://github.com/apoorvumang/prompt-lookup-decoding>
37. <https://arxiv.org/abs/2503.01840>
38. <https://arxiv.org/abs/2401.10774>
39. <https://arxiv.org/pdf/2412.19437>
40. [https://docs.vllm.ai/en/latest/features/speculative\\_decoding/mtp](https://docs.vllm.ai/en/latest/features/speculative_decoding/mtp)
41. [https://huggingface.co/blog/universal\\_assisted\\_generation](https://huggingface.co/blog/universal_assisted_generation)
42. <https://nvidia.github.io/TensorRT-LLM/1.3.0rc7/features/speculative-decoding.html>
43. <https://www.baseten.co/inference-engineering/book/05-techniques/speculative-decoding>
44. <https://www.together.ai/blog/adaptive-learning-speculator-system-atlas>
45. <https://developers.redhat.com/articles/2026/07/06/smarter-data-generation-faster-speculator-training>

46. <https://www.snowflake.com/en/blog/engineering/faster-gpt-oss-reasoning-arctic-inference>
47. <https://bentoml.com/blog/3x-faster-llm-inference-with-speculative-decoding>
48. [https://rocm.blogs.amd.com/artificial-intelligence/spec\\_decode\\_mi300x/README.html](https://rocm.blogs.amd.com/artificial-intelligence/spec_decode_mi300x/README.html)
49. <https://pytorch.org/blog/hitchhikers-guide-speculative-decoding>
50. <https://www.snowflake.com/en/engineering-blog/suffixdecoding-arctic-inference-vllm>
51. <https://rocm.blogs.amd.com/software-tools-optimization/speculative-decoding---deep-dive/README.html>
52. <https://arxiv.org/pdf/2302.01318>
53. <https://arxiv.org/abs/2503.07807>
54. <https://fireworks.ai/blog/fireoptimizer>
55. <https://www.snowflake.com/en/engineering-blog/fast-speculative-decoding-vllm-arctic>
56. <https://www.coreweave.com/pricing>
57. <https://arxiv.org/pdf/2602.09113>
58. <https://arxiv.org/html/2602.09113>
59. <https://arxiv.org/html/2406.14066>
60. [https://docs.sglang.io/advanced\\_features/speculative\\_decoding.html](https://docs.sglang.io/advanced_features/speculative_decoding.html)
61. <https://baseten.co/blog/how-we-built-production-ready-speculative-decoding-with-tensorrt-llm>
62. <https://huggingface.co/blog/assisted-generation>
63. <https://developer.nvidia.com/blog/tensorrt-llm-speculative-decoding-boosts-inference-throughput-by-up-to-3-6x>
64. [https://docs.vllm.ai/en/latest/features/speculative\\_decoding/eagle](https://docs.vllm.ai/en/latest/features/speculative_decoding/eagle)
65. [https://docs.vllm.ai/en/latest/features/speculative\\_decoding/dynamic\\_speculative\\_decoding](https://docs.vllm.ai/en/latest/features/speculative_decoding/dynamic_speculative_decoding)

## THE PUBLISHER

# About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

## Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

## Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

## Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

## Work with GPU Smith

Explore the [hardware library](#), [GPU references](#), [networking references](#) and [vendor research](#). [Contact GPU Smith](#) with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

**Website:** <https://gpusmith.com>

---

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.