SHAKUDO

# Speculative Decoding: A Reference Architecture for LLM Serving

Speculative decoding is a proven, mathematically lossless way to cut LLM latency and raise throughput 2-3x. This whitepaper covers how it works, variant comparison, framework support, and a production deployment reference architecture.

---

WHAT IS SPECULATIVE DECODING · THE LATENCY PROBLEM IN AUTOREGRESSIVE LLM INFERENCE · HOW SPECULATIVE DECODING WORKS · DRAFT MODEL STRATEGIES AND SELECTION · ACCEPTANCE, VERIFICATION, AND LOSSLESSNESS · SPECULATIVE DECODING VARIANTS COMPARED · THROUGHPUT, LATENCY, AND COST TRADEOFFS · PRODUCTION DEPLOYMENT REFERENCE ARCHITECTURE

WRITTEN FOR TECHNICAL AND SECURITY REVIEW
VENDOR-NEUTRAL · FORWARD FREELY

EDITION 1.0
2026

SECTION 01

## What Is Speculative Decoding

Speculative decoding is an inference acceleration technique that lets a large language model produce text faster without changing its output distribution. A small, fast draft model proposes several tokens in sequence, and the large target model verifies all of them in a single forward pass. Accepted tokens are kept; rejected tokens are replaced with samples from the correct distribution. The result is mathematically lossless: the output is provably identical to what the target model would have produced on its own.

The technique was introduced independently by two teams in late 2022 and early 2023. Leviathan et al. at Google Research published "Fast Inference from Transformers via Speculative Decoding" (arXiv:2211.17192, ICML 2023), and Chen et al. at DeepMind published "Accelerating Large Language Model Decoding with Speculative Sampling" (arXiv:2302.01318). Both papers describe the same core idea: trade a small amount of extra compute for a large reduction in wall-clock latency.

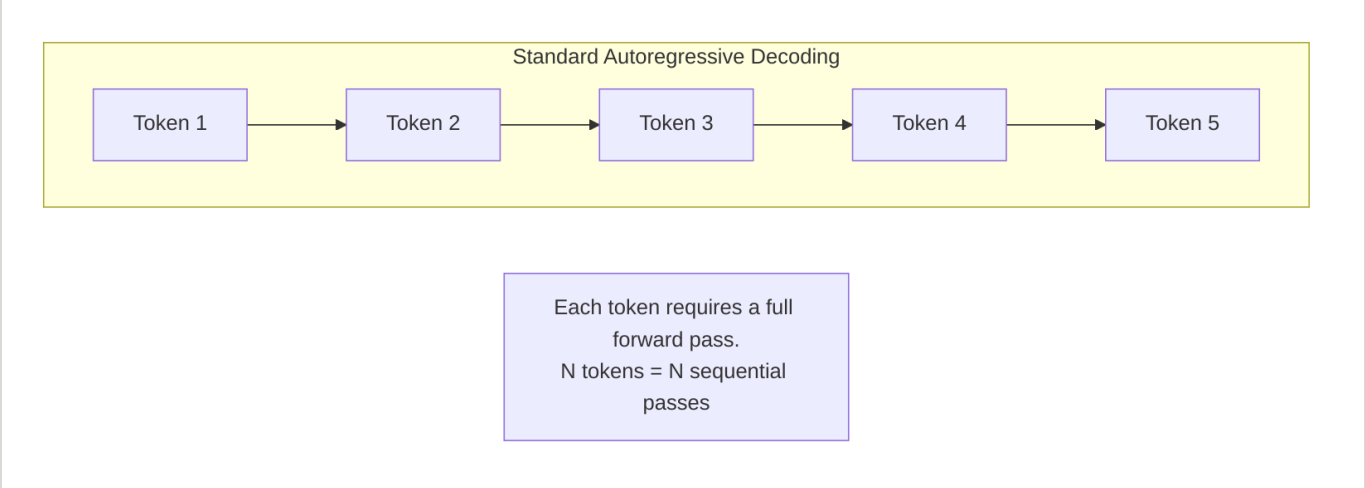
> The key insight is that standard autoregressive decoding is memory-bandwidth-bound, not compute-bound. The GPU sits mostly idle waiting for data to move, which means verifying several tokens at once costs roughly the same as generating one.

For platform teams, speculative decoding has become one of the most practical levers for cutting inference cost. Production inference frameworks including vLLM, TensorRT-LLM, Text Generation Inference (TGI), SGLang, and LMDeploy now ship native support, and leading model families such as DeepSeek V3/R1 and Gemma 4 integrate multi-token prediction directly into their architectures. This whitepaper explains how speculative decoding works, compares the major variants, and provides a reference architecture for deploying it in production.

SECTION 02

## The Latency Problem in Autoregressive LLM Inference

Large language models generate text autoregressively: each token is produced one at a time, and every new token depends on all tokens that came before it. This sequential dependency is the fundamental bottleneck in LLM serving.




The cost of this serialization is dramatic. Consider a real benchmark from our own infrastructure: an A100 80GB GPU running Qwen3-32B-FP8 achieves roughly 270,000 tokens per second during prefill (the parallel processing of the input prompt) but only about 40 tokens per second during decode (the sequential generation of the output). That is a 6,750x gap between the two phases. For a request that generates 8,192 output tokens, decoding takes approximately 206 seconds while prefilling the prompt takes 0.04 to 0.12 seconds.

The reason for this gap is that decoding processes a single token at a time. Each step requires loading the full model weights from memory, performing a matrix multiplication that uses only one row of the weight matrix, and writing out a single token.

The GPU's compute units are barely engaged; the bottleneck is memory bandwidth. This is why scaling up to larger, more expensive GPUs yields diminishing returns for decode throughput: you are paying for compute capacity that goes unused.

Several approaches attempt to address this bottleneck:

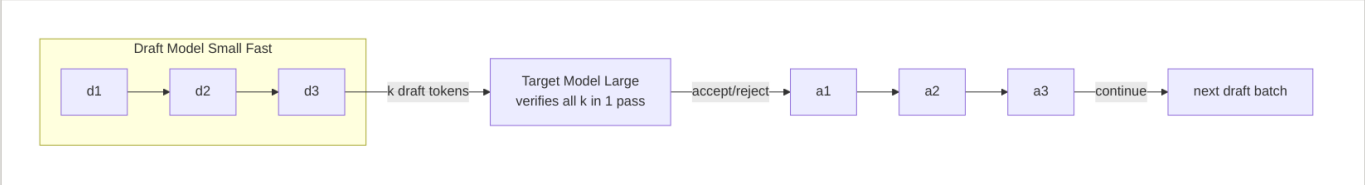
- **Batching** combines multiple requests so each forward pass produces tokens for many users, improving GPU utilization. But it increases per-request latency because each request waits for the batch to fill.
- **Quantization** reduces the memory footprint of weights, which can speed up memory-bound decoding. But it does not change the fundamental one-token-at-a-time structure.
- **Speculative decoding** attacks the problem directly by verifying multiple tokens per forward pass, converting idle compute capacity into higher throughput without increasing latency.

Speculative decoding is complementary to batching and quantization, and all three can be combined in a well-tuned serving stack.

SECTION 03

## How Speculative Decoding Works

Speculative decoding runs two models in concert: a draft model and the target model. The draft model is small and fast; the target model is the large model whose output quality you want to preserve.




The process works in three steps:

1. **Draft.** The draft model autoregressively generates K candidate tokens (the "draft length"). Because the draft model is small, this is fast.
2. **Verify.** The target model processes the K draft tokens in a single forward pass. Because the GPU is underutilized during standard decoding, verifying K tokens costs roughly the same as generating one token normally.
3. **Accept or reject.** Each draft token is compared against the target model's distribution. Accepted tokens are appended to the output. The first rejected token is discarded and replaced with a sample from the corrected distribution, and the process repeats.

The number K (draft length) is a tunable parameter. A larger K means more tokens verified per pass, but also more wasted compute if the draft model's predictions are poor. Typical values range from 2 to 8, with 4 being a common starting point.

The effective speedup depends on the acceptance rate: the fraction of draft tokens the target model accepts. If the draft model and target model are well matched, acceptance rates of 50 to 80 percent are achievable, yielding 2 to 3x speedups. The relationship is approximately:

```
speedup ~ 1 / (1 - acceptance_rate * (1 - 1/K))
```

For example, with K=4 and an acceptance rate of 0.7, the theoretical speedup is roughly 2.3x. Real-world results from NVIDIA, Hugging Face, Google, and vLLM consistently report 2-3x speedups, with code generation tasks sometimes exceeding 3x because code is highly predictable.

SECTION 04

## Draft Model Strategies and Selection

The choice of draft model is, in the words of practitioners at General Compute, "the most consequential decision" when deploying speculative decoding. The draft model must be fast enough to generate tokens quickly, yet accurate enough that the target model accepts a high fraction of them. There are three main strategies:

- **Same-family smaller siblings.** Use a smaller model from the same model family as the target. For example, pair Llama-3.3-70B with Llama-3.2-1B. This is the simplest approach and often works well because the models share training data and tokenization.
- **Distilled student models.** Train a small model specifically to mimic the target model's output distribution. DistillSpec (ICLR 2024) shows that knowledge distillation can produce draft models with higher acceptance rates than generic smaller models.
- **Self-speculative decoding.** Use the target model itself as the draft model by skipping layers or exiting early. This avoids maintaining a separate model but requires architectural support and careful tuning.

A traditional constraint is that the draft and target models must share the same vocabulary and tokenizer. If they do not, the target model cannot verify draft tokens directly. Newer methods relax this constraint:

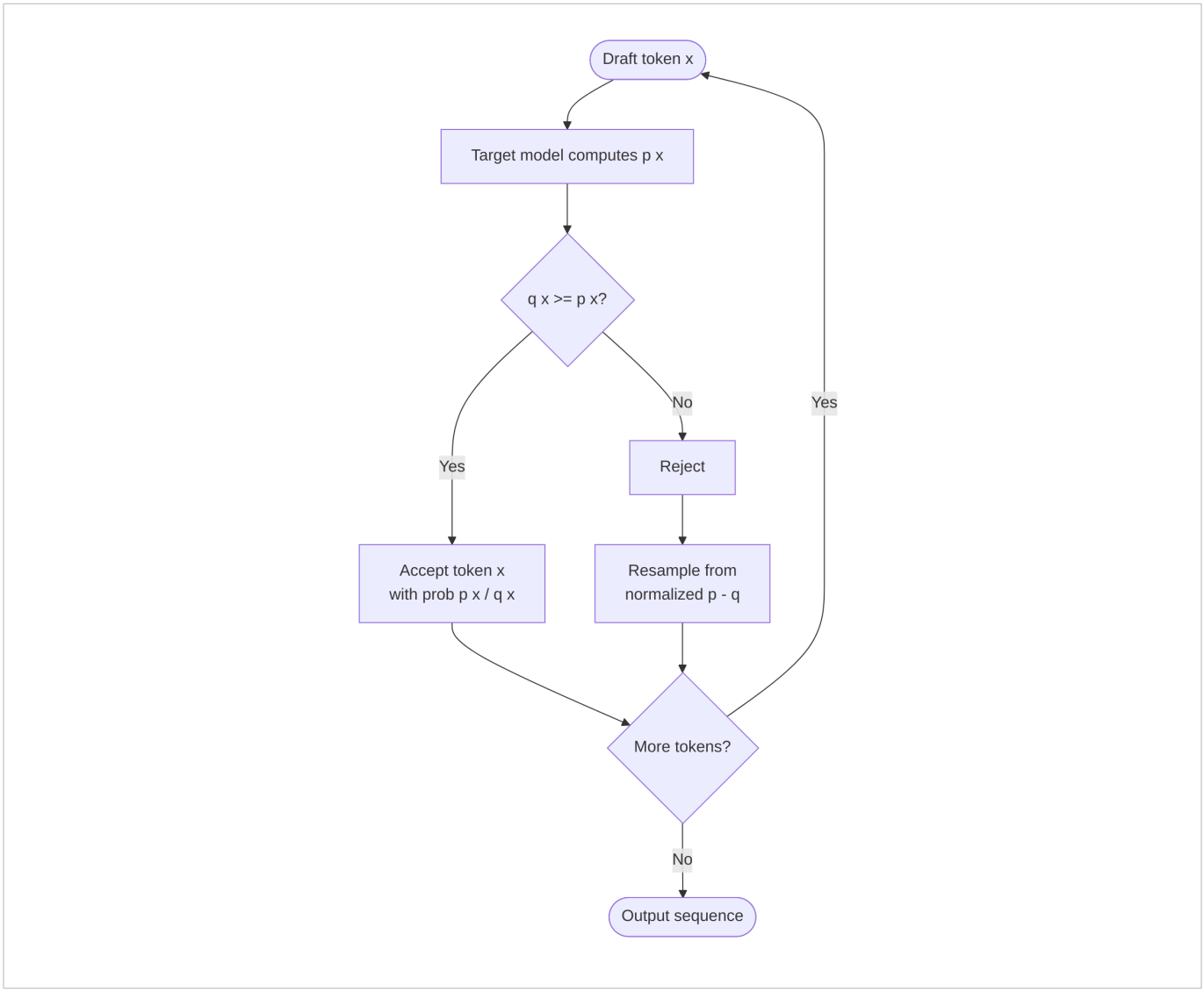
- **OmniDraft** (Qualcomm, arXiv:2507.02659) introduces a learned mapping that enables cross-vocabulary speculative decoding.
- **Heterogeneous vocabulary algorithms** (arXiv:2502.05202) allow pairing models with different tokenizers.
- **TokenTiming** (ACL 2026) provides timing-aware cross-vocab verification.

For teams just starting, the recommended path is to use a same-family smaller sibling, verify vocabulary compatibility, and benchmark acceptance rates before production deployment.

SECTION 05

## Acceptance, Verification, and Losslessness

The mathematical guarantee that makes speculative decoding safe for production is losslessness. The output distribution is provably identical to what the target model would produce on its own. No quality regression occurs, regardless of the draft model's accuracy.




The acceptance mechanism uses modified rejection sampling. For each draft token x sampled from the draft model's distribution q:

1. The target model computes its distribution p over the same position.
2. If p(x) >= q(x), the token is accepted.
3. If p(x) < q(x), the token is accepted with probability p(x)/q(x). If rejected, a replacement token is sampled from the residual distribution max(0, p - q), normalized.

This procedure ensures that every token in the final output is distributed exactly according to p, the target model's distribution. A poor draft model does not degrade quality; it only reduces the speedup because more tokens are rejected.

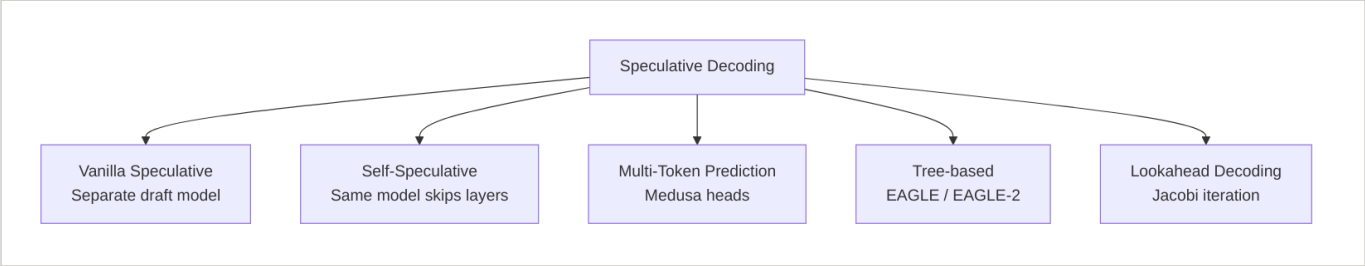
> Losslessness means you can deploy speculative decoding without re-evaluating model quality. The risk is purely a performance risk: if acceptance rates are low, you gain little or nothing. There is no accuracy risk.

The break-even point depends on the cost ratio between the draft model and the target model. If the draft model is very small relative to the target, even modest acceptance rates yield net speedups. If the draft model is large, higher acceptance rates are needed to overcome the drafting overhead.

SECTION 06

## Speculative Decoding Variants Compared

Since the original papers, the research community has developed many variants that improve on vanilla speculative decoding. The diagram below shows the taxonomy:




The major families are:

- **Vanilla speculative decoding** uses a separate, smaller draft model. This is the original formulation and remains the most widely deployed.
- **Multi-token prediction (Medusa)** adds multiple prediction heads to the target model itself, eliminating the need for a separate draft model. Medusa (arXiv:2401.10774) trains lightweight heads that predict future tokens from intermediate hidden states.
- **Feature-level autoregression (EAGLE)** improves draft accuracy by predicting at the feature level rather than the token level. EAGLE-1 (ICML 2024) introduced this approach. EAGLE-2 (EMNLP 2024, arXiv:2406.16858) adds dynamic draft trees that adapt the speculation structure. EAGLE-3 (NeurIPS 2025, arXiv:2503.01840) is the current state of the art on Spec-Bench, abandoning feature prediction for a simpler but more effective formulation.
- **Lookahead decoding** (arXiv:2402.02057) uses Jacobi iteration to generate draft tokens without any draft model at all. It reuses the target model's own computations from previous steps.
- **Self-speculative decoding** uses the target model with early exit or layer skipping as its own draft model.
- **N-gram and prompt lookup decoding** uses simple n-gram matching against the prompt or a suffix cache. It requires no model and has zero drafting overhead, making it ideal for retrieval-augmented generation (RAG) and document QA workloads where the output often copies from the input.
- **Model-native MTP** integrates multi-token prediction into the model architecture itself. DeepSeek V3/R1 and Gemma 4 ship with MTP heads baked in, enabling speculative decoding without any external draft model.

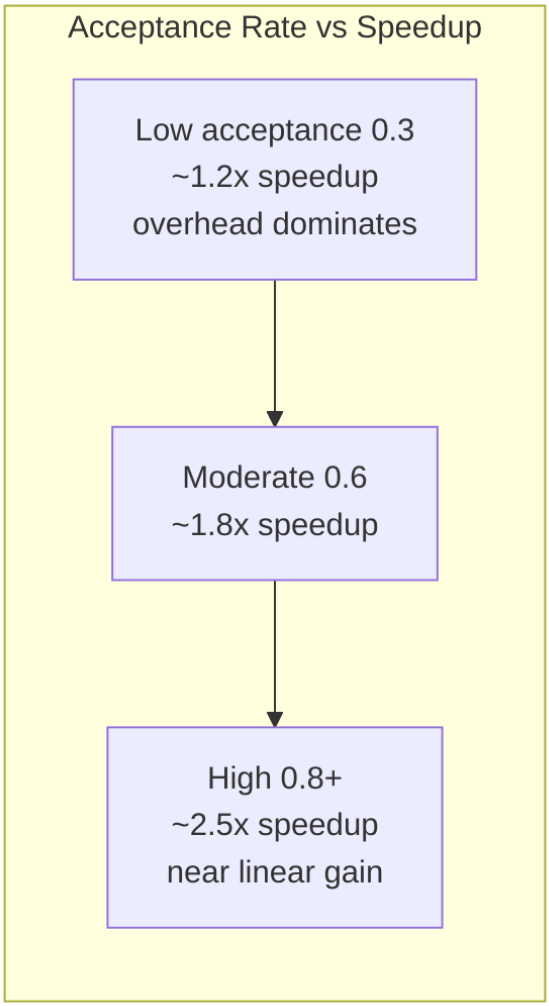
| VARIANT | DRAFT SOURCE | EXTRA MODEL | BEST FOR | KEY PAPER |
|---|---|---|---|---|
| Vanilla SD | Separate small model | Yes | General purpose, flexible pairing | Leviathan 2023 |
| Medusa | Target model heads | No | Moderate speedup, no draft model | Cai 2024 |
| EAGLE-2/3 | Feature-level drafter | Yes (lightweight) | Highest speedup, SOTA on Spec-Bench | Li 2024/2025 |
| Lookahead | Jacobi iteration | No | No draft model, moderate gain | Fu 2024 |
| N-gram / Prompt lookup | Suffix cache | No | RAG, document QA, code | Community |
| Model-native MTP | Built-in heads | No | DeepSeek V3/R1, Gemma 4 | Model-specific |

Table 6.1 · Comparison of speculative decoding variants by draft source, model overhead, ideal workload, and reference paper.

SECTION 07

## Throughput, Latency, and Cost Tradeoffs

Speculative decoding trades extra compute for lower latency. Whether this tradeoff is favorable depends on the workload, the batch size, and the acceptance rate.




The speedup curve has three regimes:

- **Low acceptance (below 0.3).** The overhead of running the draft model exceeds the benefit. Speedup is marginal, around 1.2x or less. This happens when the draft model is poorly matched or the task is highly creative.
- **Moderate acceptance (0.4 to 0.6).** Speedups of 1.5x to 2x are typical. This is the regime where most general-purpose text generation falls.
- **High acceptance (0.7 and above).** Speedups of 2x to 3x or more. Code generation, structured output, and RAG workloads often reach this regime because the text is highly predictable.

The interaction with batch size is critical. Speculative decoding helps most when the batch size is small and the GPU is memory-bound. As batch size increases, the GPU becomes compute-bound: every forward pass is already doing useful work for many requests. At high batch sizes, speculative decoding can actually reduce throughput because the extra draft verification adds compute load to an already saturated GPU.

vLLM's documentation explicitly warns that model-based speculative methods increase workload during peak traffic. Cohere's dynamic speculative decoding (DSD) blog addresses this by dynamically adjusting the draft length based on current load: using longer drafts when the system is idle and shorter drafts (or none) under heavy load.

The cost implications are:

- **Latency-sensitive workloads** (interactive chat, code completion) benefit the most. A 2-3x latency reduction directly improves user experience.
- **Throughput-oriented workloads** (batch processing, offline inference) benefit less and may be harmed at high concurrency.

- **Mixed workloads** should use dynamic adjustment to enable speculative decoding during low-traffic periods and disable it during peaks.

SECTION 08

## Production Deployment Reference Architecture

Deploying speculative decoding in production requires choosing a serving framework, selecting a method, and configuring it correctly. The table below summarizes framework support:

| FRAMEWORK | SUPPORTED METHODS | CONFIG API | NOTES |
|---|---|---|---|
| vLLM | EAGLE, Medusa, MTP, Draft, N-gram, Lookahead, Suffix, PARD, MLP | `--speculative-config` JSON | Broadest support, most methods |
| TensorRT-LLM | EAGLE, Medusa, MedusaTree, ReDrafter, Draft, NGram, Lookahead | Speculative Sampling config | NVIDIA-optimized, best on H100/A100 |
| TGI | Medusa, N-gram, assisted generation | Server flags | "2-3x faster, much more for code" |
| SGLang | EAGLE, draft model | Server config | High throughput, good for multi-tenant |
| LMDeploy | EAGLE, Medusa, draft, n-gram | TurboMind engine config | Optimized for quantized models |

Table 8.1 · Speculative decoding support across major LLM serving frameworks, including supported methods and configuration APIs.

A production reference architecture for speculative decoding includes the following components:

- **Target model server.** The primary inference engine (vLLM, TensorRT-LLM, TGI, or SGLang) running the large target model with KV cache management and continuous batching.
- **Draft model.** Either a separate small model loaded alongside the target, or an integrated method (Medusa heads, EAGLE drafter, or model-native MTP).
- **Acceptance monitor.** A metrics collector that tracks the acceptance rate in real time. If the rate drops below a threshold, the system should reduce draft length or disable speculation.
- **Dynamic controller.** Adjusts draft length K based on current load. During low traffic, increase K for lower latency. During peak traffic, reduce K or disable speculation to maximize throughput.
- **Benchmark harness.** A pre-deployment validation step that measures acceptance rates and speedups on representative workloads using the team's actual hardware.

The deployment process follows this sequence:

1. Identify the workload type (interactive, batch, mixed).
2. Select a draft model or method based on the target model family.
3. Verify vocabulary and tokenizer compatibility between draft and target.
4. Choose the speculative decoding method supported by your serving framework.
5. Configure the framework with the appropriate API (for example, `--speculative-config` in vLLM).
6. Tune the draft length K. Start with K=4 and adjust based on measured acceptance rates.
7. Benchmark on production hardware with representative prompts.
8. Set up monitoring for acceptance rate, latency, and throughput.

> Monitor acceptance rates continuously. A sudden drop signals a distribution shift in your traffic, which means the draft model is no longer well matched to the current workload. Build alerting around this metric.

SECTION 09

## When Speculative Decoding Helps (and When It Does Not)

Speculative decoding is not a universal speedup. Understanding when it helps and when it hurts is essential for making the right deployment decision.

Speculative decoding helps when:

- **Batch sizes are small.** The GPU is memory-bound, so verifying extra tokens is nearly free.
- **The workload is interactive.** Low latency directly improves user experience for chat, code completion, and copilot applications.
- **The task is predictable.** Code generation, structured output (JSON, SQL), RAG, and document summarization have high acceptance rates.
- **The draft model is well matched.** Same-family models or distilled draft models achieve high acceptance rates.
- **Traffic is low to moderate.** The GPU has spare capacity for draft verification.

Speculative decoding hurts when:

- **Batch sizes are large.** The GPU is compute-bound, and draft verification adds load to an already saturated system.
- **Traffic is at peak.** vLLM warns that model-based methods increase workload during peak traffic, reducing overall throughput.
- **The task is creative or diverse.** High-temperature generation, open-ended creative writing, and broad-domain QA have low acceptance rates.
- **Sequences are short.** The overhead of warming up the draft model may exceed the benefit for short generations.
- **The draft model is poorly matched.** If acceptance rates fall below 30 percent, the overhead exceeds the gain.

A practical decision framework:

- [ ] Is the workload latency-sensitive (interactive, real-time)?
- [ ] Are batch sizes typically small (under 8-16 concurrent requests)?
- [ ] Is the output text predictable (code, structured, RAG-grounded)?
- [ ] Can you find a same-family or distilled draft model?
- [ ] Do you have a benchmark harness to measure acceptance rates?
- [ ] Can you implement dynamic adjustment for peak traffic periods?

If you answer yes to four or more of these, speculative decoding is likely to deliver meaningful speedups. If you answer yes to fewer than three, the overhead may not be worth the complexity.

SECTION 10

## Conclusion and Next Steps

Speculative decoding is a proven, mathematically lossless technique for accelerating LLM inference by 2-3x. It works by converting idle GPU compute capacity into higher throughput, exploiting the fundamental memory-bandwidth bottleneck of autoregressive decoding. The technique is mature: it is supported by every major serving framework, has been validated in production by teams at Google, NVIDIA, Hugging Face, and others, and has spawned a rich family of variants that improve on the original formulation.

For platform teams evaluating inference acceleration, the path forward is:

- [ ] Identify your workload profile (interactive vs batch, predictable vs creative)
- [ ] Select a draft model or method matched to your target model
- [ ] Verify vocabulary and tokenizer compatibility
- [ ] Configure speculative decoding in your serving framework
- [ ] Benchmark acceptance rates and speedups on production hardware
- [ ] Deploy with dynamic adjustment enabled for peak traffic resilience

- [ ] Monitor acceptance rates and set alerts for distribution shift

The field continues to evolve rapidly. Model-native multi-token prediction, as seen in DeepSeek V3/R1 and Gemma 4, represents a paradigm shift: speculative decoding capabilities are increasingly built into the model itself rather than added at the serving layer. Cross-vocabulary methods like OmniDraft are removing the constraint that draft and target models must share tokenizers, opening up new pairing possibilities. And dynamic speculation techniques are making it safe to run speculative decoding under variable load without throughput regression.

If your team is running LLM inference workloads and wants to reduce latency and cost without sacrificing output quality, speculative decoding is one of the highest-impact optimizations available today. Start with a same-family draft model, benchmark on your real traffic, and let the acceptance rate data guide your configuration.