Inference Is the New Frontier

Why the Decode Loop, Not the Model, Now Decides Enterprise AI Economics

Binod Kumar

Founder & Chief AI Officer, AgentsArchitects.ai

Mentor at E-Cell, Indian Institute of Technology Bombay and Banaras Hindu University, India

Frontier Research Series

Produced for and with the AuthClaw Foundation and the AGI Learning Hub, India

June 2026

Abstract

The industry spent the last three years learning how to train and pick models. The problem still open in 2026 is how to serve them, because the cost, speed, and margin of an enterprise AI product are now set inside the decode loop rather than inside the model weights. Consider a customer-facing assistant that answers in two seconds at a hundred concurrent users and in nine seconds at ten thousand. Nothing about the model changed between those two states. What changed was the serving regime, and that regime is now where the money is won or lost.

This paper argues that inference has become a first-class systems-engineering discipline, and that recent research on speculative decoding makes the point concrete. A measurement study of speculative decoding on a production engine found that verification by the target model, not drafting, dominates execution time, and that the real speedup falls well short of the theoretical bound. New methods answer that finding from two directions: SpecVocab attacks the cost of the draft step by speculating on which slice of the vocabulary to compute, and DSpark attacks the verification step by scheduling how much to verify against live hardware load. The underlying lesson is an old one in systems work, the difference between a microbenchmark and a production workload, only newly expensive.

For a Chief AI Officer the takeaway is uncomfortable but simple: your unit cost and your latency are properties of how you serve, not of which model you bought, and a competitor on the same model can beat you on both. For a researcher, the open questions are about adaptive, load-aware, lossless inference and how to certify it. The thesis is narrower and more useful than "inference matters now": the decode loop is a tunable system with its own failure modes, and treating it as a fixed cost is the mistake.

Keywords: speculative decoding; inference optimization; LLM serving; verification bottleneck; throughput; latency; draft model; vocabulary speculation; load-aware scheduling; total cost of ownership; enterprise AI economics; semi-autoregressive generation

1. Why the Decode Loop Is the Unit of Risk

Generative AI has moved from pilot to production, and the spending now reflects it. Gartner forecasts that worldwide AI spending will reach roughly 2.59 trillion dollars in 2026, a 47 percent rise year over year, and describes this as the inflection year in which enterprises, rather than hyperscalers, begin to spend at scale. The same analysts note that the cost question has not gone away as models have become cheaper to run. Gartner expects that by 2030 inference on a trillion-parameter model will cost providers over 90 percent less than it did in 2025, and in the same breath warns that enterprise bills will keep rising, because agentic workloads consume between five and thirty times more tokens per task than a single chatbot reply. Unit cost falls while total cost climbs. That is the shape of the problem.

The trouble is that most enterprise AI conversations are still held at the level of the model. Which model is smartest, which has the longest context, which tops a leaderboard this month. Those are real questions, but they are the wrong place to manage cost and speed once a system is in production, because the model is the one part of the stack a buyer cannot change without re-qualifying everything around it. The serving layer, by contrast, is yours to tune.

A useful way to see the shift is to count the questions. The naive framing asks one: which model do I deploy? The production setting forces a second, harder one: under my real traffic, with my real mix of long and short requests and my real concurrency, how fast and how cheaply does that model generate each token, and how does that change as load moves through the day? The first question has a clean answer you can read off a benchmark. The second has an answer that depends on your hardware, your batch sizes, and the scheduling decisions made thousands of times a second inside the decode loop.

So the thing to study is not the model in isolation but the decode loop that turns it into a service.

2. The Decode Loop, Restated as a Serving Problem

The central failure is precise enough to write down. A large language model generates text one token at a time, and each token requires a full forward pass over billions of parameters. The pass is bound not by arithmetic but by memory: the hardware spends most of its time moving model weights from memory to compute units, and the actual multiplication is comparatively cheap. Generating a thousand-token answer means a thousand sequential passes, each waiting on memory. That serial, memory-bound loop is the inference problem in one line. Everything called an inference optimization is an attempt to get more useful tokens out of each expensive trip to memory.

Speculative decoding is the leading such attempt, and it is worth stating formally because the rest of this paper turns on its structure. Let a small, fast draft model q propose a block of k candidate tokens. Let the large target model p verify all k in a single forward pass, accepting the longest prefix that matches what p would itself have produced and correcting the first divergence. Because verification of k tokens costs roughly one target pass rather than k, the method produces the same output distribution as the target model while doing fewer expensive passes. The decomposition has three moving parts: the cost of drafting, the cost of verifying, and the acceptance length, meaning how many proposed tokens survive on average. Speedup rises with acceptance length and falls with the cost of the two model passes.

The wry part is that speculative decoding is a manager who asks a junior analyst to draft the whole memo, then reads it in one sitting and keeps every paragraph up to the first mistake. It is fast precisely when the junior is good and the reading is cheap. When either assumption slips, the manager is back to writing every line.

No adversary is required for this to go wrong, and no exotic model. The failure is structural. If the draft is often wrong, acceptance length collapses and the drafting work is wasted. If verification is expensive relative to the savings, the speedup evaporates even when the draft is good. The method is a bet that drafting is cheap and verification is cheaper still per token saved, and that bet is not always paid.

3. Anatomy of a Decode Step: What Crosses the Boundary

To see where the speedup breaks down, follow what physically crosses the line between draft and target on every step. In the well-built case, the draft model takes the current context, produces k candidate tokens and their probabilities, and hands that block to the target model. The

target runs one parallelized forward pass, compares its own distribution to the draft's at each position, accepts the matching prefix, and emits one corrected token at the point of divergence. The accepted tokens become the new context and the loop repeats. Done right, several tokens are finalized per target pass instead of one.

Here are the ways that clean picture degrades in production.

The vocabulary tax. The draft model's final step is to compute a probability over every token in the vocabulary, and modern vocabularies are large, often well over a hundred thousand entries. Recent work identified that in widely used draft architectures the majority of drafting time is spent on this output projection rather than on the reasoning that precedes it. The draft model is fast in theory and slow in practice because its last act is expensive.

The verification wall. Verifying k tokens is one target pass, which is cheap compared to k separate passes but not free, and it scales with model size. A measurement study of speculative decoding on the vLLM production engine found that verification by the target model, not drafting, is the dominant component of execution time. The part everyone treats as the bargain is in fact the largest bill.

The batch-size cliff. Speculative decoding spends spare compute to avoid memory reads, which is a clear win when one request runs alone and the hardware has idle cycles. As concurrency rises, the hardware moves from memory-bound to compute-bound, the spare cycles vanish, and the extra verification work now competes for compute that is already saturated. The same trick that accelerates a single user can slow a busy server, and the literature is explicit that the gain can trend to zero or below as batch size grows.

Acceptance decay along the block. The further out the draft predicts, the less reliable each successive token, so acceptance does not hold steady across a block. Tokens late in a speculative sequence are likelier to be rejected, which caps how aggressively a system can draft before it is verifying tokens that will be thrown away.

4. Four Failure Modes, and How They Compound

4.1 Drafting that is fast on paper and slow on the clock. When the output projection over a large vocabulary dominates draft time, a draft model that looks lightweight by parameter count is heavy by wall-clock. The fix of simply shrinking the draft's vocabulary to a fixed common subset trades the problem for another: when the next real token falls outside that subset, the draft is forced to miss, and the speculative run ends early.

4.2 Verification that eats the savings. Because verification dominates execution, any method that raises acceptance length without watching verification cost can post a better acceptance number and a worse end-to-end speed. Acceptance length and throughput are not the same metric, and optimizing the first while ignoring the second is a common and costly confusion.

4.3 A serving regime that inverts under load. A configuration tuned at low concurrency, where speculation shines, can become a net loss at the concurrency a production service runs at. A system that does not adapt its speculative behavior to live load is optimized for a demo, not for a Tuesday afternoon at peak.

4.4 Static choices in a dynamic workload. Fixed draft length, fixed vocabulary, fixed verification budget. Each is a single number chosen once, while the workload it serves varies by request, by task, by hour. A static parameter is a bet that the workload is stationary, and production workloads are not.

How they compound. Each missing defense removes the guard the next failure needs. A

heavy draft step (4.1) shrinks the speedup margin, which makes the verification cost (4.2) proportionally more damaging. A system that has not measured its verification cost cannot detect when rising load (4.3) has tipped it past the batch-size cliff. And a system with no adaptive control (4.4) has no lever to pull once it notices, so it rides the wrong configuration straight through peak. The failures do not add. They multiply, because each one disables the instrument that would have caught the next.

5. Why the Model-Centric Stack Cannot Carry This

It is tempting to assume that a better or bigger model solves this. It does not, and the reason is structural rather than a matter of model quality. The model-centric procurement habit, choose the best model and treat serving as a fixed downstream cost, carries a set of baked-in assumptions that the production setting breaks.

It assumes the benchmark regime matches the production regime. Speculative-decoding gains are typically reported at small batch sizes on research prototypes, and the measurement study cited above exists precisely because those conditions do not hold on a loaded production engine. It assumes that a speedup measured for one user generalizes to many, when the batch-size cliff guarantees it does not. It assumes acceptance length is a proxy for speed, when verification cost can break that link. And it assumes the right serving parameters can be set once, when the workload that determines them shifts continuously.

The measurement literature reaches the same conclusion from the data side. The systematic study of speculative decoding on a production engine reports a substantial gap between the speedups observed in practice and the theoretical upper bound, and attributes that gap to factors, verification dominance and wide variance in acceptance across requests and datasets, that a model swap does not touch. These are properties of how generation is scheduled and executed, not of what was generated.

The short version: the model decides what your system can say, and the serving layer decides what it costs you to say it and how fast. You can buy the first. You have to engineer the second.

6. What Inference Optimization Must Mean at Production Scale

A serving layer that an enterprise can stand behind has to satisfy a specific list. Most deployments meet one or two items on it.

1. Losslessness by construction. The optimized path must produce the same output distribution as plain decoding from the target model, so that speed is never bought with a silent quality regression. Speculative decoding has this property by design; any optimization that lacks it is a different and riskier trade.

2. Measured verification cost, not assumed. The system must instrument the verification step as a first-class cost, because it is the dominant one, and report it separately from drafting and from acceptance length.

3. Load awareness. The serving regime must know its current concurrency and where it sits relative to the batch-size cliff, and change behavior as that position changes through the day.

4. Adaptive speculation depth. How many tokens to draft and how many to verify must be decisions the system makes per request and per load condition, not constants set at deploy time.

5. Acceptance calibration. The system must estimate how likely each drafted token is to be accepted, and act on that estimate, rather than drafting a fixed depth and discovering the waste only after verification.

6. Throughput-interactivity accounting. The system must be able to state its position on the trade between per-user latency and aggregate throughput, and move along that frontier deliberately rather than landing wherever a default put it.

7. Workload-representative evaluation. Acceptance and throughput must be measured on the enterprise's own task mix and concurrency, not read off a public benchmark whose regime may not resemble production.

Most production systems satisfy one or two of these. Almost none satisfy all seven. That gap is the inference problem, written out as a requirements list.

7. Candidate Building Blocks, and Where Each Stops

EAGLE-3. A state-of-the-art speculative-decoding method that uses a lightweight draft model and tree-structured drafting to raise acceptance length, and a widely adopted baseline in both research and inference engines. Where it stops: it uses a fixed reduced vocabulary chosen ahead of time, so it pays an acceptance penalty whenever the next token falls outside that static subset.

SpecVocab. A 2026 method from researchers at the University of Sheffield and the Samsung AI Center that reframes the vocabulary problem: rather than fix a common subset, it speculates per step on which small slice of the vocabulary is contextually relevant, computing exact probabilities for only that slice. Reported gains over EAGLE-3 reach up to roughly 8 percent higher average throughput, with the method needing exact computation over as little as one to two percent of the vocabulary per step. Where it stops: it optimizes the draft side, the cost of producing candidates, and leaves the dominant verification cost to other mechanisms.

DSpark. A June 2026 release from DeepSeek that attacks the verification side. It pairs a semi-autoregressive draft design with a confidence head that scores how likely each drafted token is to survive, and a hardware-aware scheduler that sets the verification budget per request against live GPU load, verifying more when the hardware is idle and less when it is busy. In DeepSeek's own production measurements on its V4 models, per-user generation ran 60 to 85 percent faster than their prior baseline. Where it stops: that headline figure is the vendor's own, measured in the vendor's regime against the vendor's baseline, and independent reproduction on other stacks is still pending; the design is also coupled to a specific model family as deployed.

vLLM and SGLang. Production inference engines that supply the execution substrate, paged attention, structured-program execution, and the engineering that makes speculative methods measurable rather than theoretical. The measurement study of speculative decoding was possible because vLLM is a real, optimized engine. Where it stops: an engine gives you the place to implement adaptive, lossless, load-aware serving; it does not by itself decide your policy for doing so.

DeepSpec. An open-source training and evaluation stack released alongside DSpark, supporting multiple speculative-decoding algorithms with open checkpoints. Where it stops: it lowers the cost of building and benchmarking drafters, but the choice of regime, parameters, and load policy for a specific enterprise workload remains an engineering decision the tooling cannot make for you.

No single block closes the list in Section 6. The near-term answer is to compose them, draft-side vocabulary speculation, verification-side scheduling, a measurement-capable engine, and workload-representative evaluation, rather than wait for one method to do everything.

8. A Reference Inference-Governance Architecture

This is where I move from interpretation to a position of my own. The Frontier Research Series has argued in prior work that AI governance should be evaluated at runtime, on every action, against a four-tier identity model: Organization, then User, then Agent, then Capability, with the Agentic Authority Architecture (AAA) placing the decision at the capability level rather than at the perimeter. Inference is the natural next surface for that same discipline, because the decode loop is exactly where an abstract capability becomes a concrete, billable, latency-bearing action. The directives below extend AAA from "who may act" to "how that action is served."

Treat the serving regime as a governed capability, not a fixed cost. In the AAA model a capability is the unit at which policy is evaluated. Token generation is a capability. Its cost, latency, and losslessness are policy properties of the Organization tier, set deliberately, not inherited from a default.

Instrument the decode loop before optimizing it. Measure verification cost, drafting cost, acceptance length, and current concurrency as separate, logged signals. You cannot govern what you do not measure, and verification dominance means the most important number is the one most often left unmeasured.

Make speculation depth and verification budget runtime decisions. Bind them to live load and to per-token acceptance estimates, the way AAA binds an authorization decision to the live context of an action rather than to a setting fixed at provisioning.

Keep every optimization lossless and prove it. An optimization that changes the output distribution is a capability change and must be governed as one, with its own evaluation. Speed bought by silently lowering quality is the inference equivalent of privilege escalation: a capability the system was not authorized to exercise.

Evaluate on your own workload, at your own concurrency. Treat a public benchmark as a vendor's claim, useful, not authoritative, and require workload-representative measurement before a serving change reaches production.

Place the policy at the boundary you own. The model is a procured component; the serving layer is yours. Put the adaptive, measured, lossless control at that boundary, because it is the one place in the stack where you can change behavior without re-qualifying the model.

The shape in plain terms: govern the decode loop the way you govern any other capability that spends money and carries risk on every invocation, by measuring it, binding its behavior to live conditions, and proving it has not quietly traded away the thing you were selling.

9. What Enterprises Should Do in the Next Two Quarters

For a CAIO who wants an action list rather than an architecture diagram, here is where to start.

None of this waits on the research to settle. It waits on you to treat inference as a system you own and govern, rather than a fixed price printed on the model.

1. **Measure your own inference economics.** Establish per-token cost, per-request latency, and throughput on your real workload and concurrency, broken out by drafting and verification, before changing anything.
2. **Separate the model decision from the serving decision.** Stop letting "which model" stand in for "how we serve it," and assign clear ownership of the serving layer as its own engineering surface.

3. **Pilot speculative decoding on a representative workload.** Run a lossless speculative method against your traffic, not a benchmark, and measure end-to-end speed, not acceptance length alone.

4. **Test at production concurrency, not at batch size one.** Reproduce the batch-size cliff deliberately, find where speculation stops paying for your traffic, and size your deployment to the regime you will run in.

5. **Require load-aware serving in your roadmap.** Treat adaptive verification budgeting and load-aware scheduling as standard requirements for any serving stack you build or buy, given how directly they govern cost at scale.

6. **Adopt FinOps discipline for tokens.** Put inference spend under the same budgeting, routing, and review that mature organizations apply to cloud cost, including routing simple tasks to smaller models and reserving frontier models for work that needs them.

7. **Demand workload-representative evidence from vendors.** When a vendor cites a speedup, ask for the baseline, the batch size, the workload, and whether the result has been reproduced outside their own stack, and weight the claim accordingly.

10. Open Problems for Researchers

For the PhD students and fellow researchers reading, the most consequential problems here are unsolved and well-posed.

A standing warning for practitioners: a speedup that has not been measured on your workload at your concurrency is a hypothesis, not a result.

A group that produced a verification primitive as cheap as today's drafting primitives would not be writing a paper. It would be writing the missing layer of the inference stack.

1. **Verification-cost reduction.** Drafting has received intense attention; verification, the dominant cost, has received less. What lowers the per-token cost of verification itself, losslessly, without simply shrinking the target?

2. **Provably optimal adaptive speculation.** Given live load, acceptance statistics, and hardware throughput curves, what is the optimal speculation depth and verification budget per request, and can a scheduler approach it with guarantees rather than heuristics?

3. **Closing the gap to the theoretical bound.** The measurement study shows a wide gap between observed and theoretical speedup. What fraction of that gap is irreducible, and what is recoverable engineering?

4. **Acceptance prediction under distribution shift.** Confidence-based scheduling depends on calibrated acceptance estimates. How well do those estimates hold as workloads drift, and how should they be recalibrated online?

5. **Cross-stack reproducibility.** Vendor speedups are reported in vendor regimes. What is a standard, workload-representative protocol for reporting speculative-decoding gains so that numbers from different stacks can be compared at all?

6. **Collaborative and edge inference.** Splitting drafting on a local device from verification in the cloud raises a new variable, network round-trip latency, into the scheduling decision. What does load-aware, lossless serving look like when the boundary crosses a network?

Conclusion

The failure this paper describes is structural, not accidental. The cost and speed of an enterprise AI product are decided in the decode loop, and the decode loop has its own physics, memory-bound generation, verification-dominated speculation, a batch-size cliff that inverts the economics under load, and static parameters that fit demos and miss workloads. None of that is fixed by buying a better model, because none of it lives in the model.

The fix is not a model. It is an architecture: a serving layer that is measured, load-aware, adaptive, and provably lossless, governed as a capability the organization owns rather than a price it accepts.

Training taught the machine what to say. Inference decides what each word costs you, and whether the next one arrives before your customer leaves.

References

Note: identifiers and years below should be verified against the primary record before formal submission.

References

[1] Williams, M., Kwon, Y. D., Li, R., Kouris, A., and Venieris, S. I. *Speculative Decoding with a Speculative Vocabulary.* Introduces SpecVocab, per-step vocabulary speculation, and reports throughput gains over EAGLE-3. Preprint, arXiv:2602.13836, 2026.

[2] (Sky Computing Lab et al.) *Speculative Decoding: Performance or Illusion?* Systematic measurement of speculative decoding on the vLLM production engine; identifies verification as the dominant cost and a gap to the theoretical bound. Preprint, arXiv:2601.11580, 2026.

[3] DeepSeek-AI. *DSpark: Confidence-Scheduled Speculative Decoding with Semi-Autoregressive Generation.* Verification-side scheduling with a confidence head and hardware-aware budget; production speedup figures on DeepSeek-V4. Technical report, 2026. [identifier: verify]

[4] Li, Y., Wei, F., Zhang, C., and Zhang, H. *EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test.* State-of-the-art speculative-decoding baseline used throughout this paper. Preprint, arXiv:2503.01840, 2025.

[5] Li, Y., Wei, F., Zhang, C., and Zhang, H. *EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees.* Dynamic draft-tree method underpinning later EAGLE work. EMNLP, 2024.

[6] Zhao, W., Pan, T., Han, X., et al. *FR-Spec: Accelerating Large-Vocabulary Language Models via Frequency-Ranked Speculative Sampling.* First identifies the output-distribution bottleneck in drafting; supports the vocabulary-tax discussion. ACL, 2025.

[7] Goel, R., Agrawal, S., Gagrani, M., et al. *VocabTrim: Vocabulary Pruning for Efficient Speculative Decoding in LLMs.* Post-training vocabulary pruning baseline for the draft-side cost discussion. Preprint, arXiv:2506.22694, 2025.

[8] Leviathan, Y., Kalman, M., and Matias, Y. *Fast Inference from Transformers via Speculative Decoding.* Foundational formulation of the draft-then-verify method. ICML, 2023.

[9] Chen, C., Borgeaud, S., Irving, G., et al. *Accelerating Large Language Model Decoding with Speculative Sampling.* Concurrent foundational speculative-sampling work establishing

losslessness. Preprint, arXiv:2302.01318, 2023.

[10] Xia, H., Yang, Z., Dong, Q., et al. *Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding.* Survey grounding the acceptance-length and throughput metrics used here; source of the Spec-Bench evaluation. ACL Findings, 2024.

[11] Kwon, W., Li, Z., Zhuang, S., et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention (vLLM).* The production inference engine on which the measurement study is run. SOSP, 2023.

[12] Zheng, L., Yin, L., Xie, Z., et al. *SGLang: Efficient Execution of Structured Language Model Programs.* Optimized serving framework used for real-world throughput measurement in the SpecVocab work. NeurIPS, 2024.

[13] Cai, T., Li, Y., Geng, Z., et al. *Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads.* Auxiliary-head drafting approach in the building-block lineage. ICML, 2024.

[14] Tang, B., Fu, C. C., Kou, F., et al. *Efficient Speculative Decoding for Llama at Scale: Challenges and Solutions.* Evidence that speculative decoding is deployed at production scale, supporting the demo-to-production framing. Preprint, arXiv:2508.08192, 2025.

[15] Gartner, Inc. *Forecasts Worldwide AI Spending to Grow 47 Percent in 2026.* Source for the 2026 spending figure and the enterprise-inflection framing in Section 1. Press release, May 2026.

[16] Gartner, Inc. *Predicts That by 2030 Inference on a One-Trillion-Parameter LLM Will Cost GenAI Providers Over 90 Percent Less Than in 2025.* Source for the falling-unit-cost and agentic token-multiplier figures in Section 1. Press release, March 2026.