

Accelerating LLM Inference on **SiMa.ai Modalix** with Speculative Decoding

Prasad Naik

Software Engineer, Platform
Architecture, [SiMa.ai](#)

Florian Voss

Senior Machine Learning
Engineer, [SiMa.ai](#)

David Olsen

Senior Manager, Product
Marketing, [SiMa.ai](#)

Executive Summary

On-device AI assistants need to generate responses quickly while operating within limited power and memory budgets. Traditional LLM decoding produces one token per model step, which can make generation slow even when compute capacity is available. We have implemented three speculative decoding approaches in the LLiMa framework - EAGLE3, DFlash, and Gemma 4 MTP - that propose multiple tokens for the target model to verify together. Each of these approaches lets the target model yield multiple tokens per step, decreasing decoding latency.

Across several prompts suitable for Physical AI applications, we evaluated the three speculative decoding

approaches on the [SiMa.ai](#) Modalix MLSoC platform. Tested configurations increased decode throughput by **2.21× to 3.49×** over traditional decoding of the same target models. In a separate automotive assistant test, all **900 measured requests** produced the expected structured tool call, while complete tool-call time improved by **1.35× to 2.06×** and estimated SoC compute energy per request fell by **25.9% to 51.4%**. These results demonstrate faster generation and faster actionable responses for the tested workloads, at broadly similar mean SoC power, while energy per token decreases significantly.

The Autoregressive Decoding Bottleneck

AI assistants deployed on Physical AI applications need speed to succeed. If an LLM feels slow, users notice immediately. Yet, achieving fast, fluid responses on edge hardware is challenging. It brings constraints that datacenter service does not: limited memory and bandwidth, a bounded power envelope, and little room to compensate for memory-bound decoding by throwing more compute at the problem.

In standard autoregressive decoding, exactly one output token is produced per forward pass through a model. That per-token cost sets a ceiling on a model's throughput as measured in tokens-per-second (TPS). Each pass requires accessing the model weights and then generating tokens one-at-a-time, which means the process spends most of its time waiting on data movement rather than computing. This highlights the well-known distinction between compute-bound and memory-bound regimes, with decoding sitting in the latter category. As a result, TPS is limited by how fast weights can be delivered no matter how much processing power is available [1].

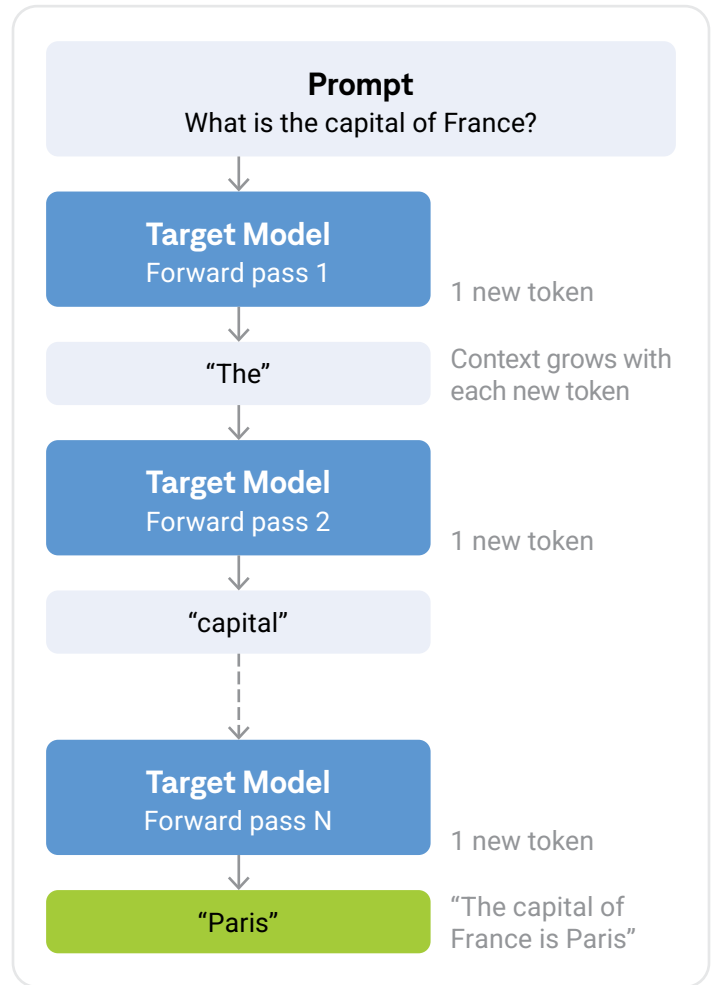


FIGURE 1. Traditional decoding generates one token per target-model pass.

What SiMa.ai Is Doing About It

SiMa.ai has brought speculative decoding to LLaMa, its framework for deploying LLMs and VLMs on Modalix MLSoCs, to accelerate LLM operations. Three approaches were evaluated - EAGLE3, DFlash, and Gemma 4's multi-token prediction (MTP) - across several

models (Llama 3.1 8B, Qwen3 8B, Qwen3.5 9B, and Gemma 4 E4B). The results show how these approaches can accelerate on-device generation and, in the case of an automotive assistant, reduce the time to produce a complete tool call.

How Speculative Decoding Works

LLM inference contains two separate stages: **(1)** prefilling/prompt processing - which processes an input like “What is the capital of France?” and fills the Key-Value (KV) cache. **(2)** token generation/decoding, which generates one token (i.e. word/partial word) at a time, like “capital / of / France / ...” Consider an AI assistant that writes out a response one word at a time: before it can write the next word, it must decide what follows everything it has already written. As a result, each successive token requires another complete step through the model.

Speculative decoding is an inference technique that lets a fast, lightweight “draft model” (drafter) look ahead and, instead, propose a sequence of candidate tokens that might come next. The full LLM (called the “target model”) checks those candidates together. It accepts a valid sequence of tokens and corrects the response at the first rejected candidate. After several candidates are accepted, the response moves forward with several

tokens in one target-model step. This can increase tokens per second (TPS) significantly [2][3]. The drafter is central to speculative decoding: it must propose tokens quickly enough and accurately enough to save time overall.

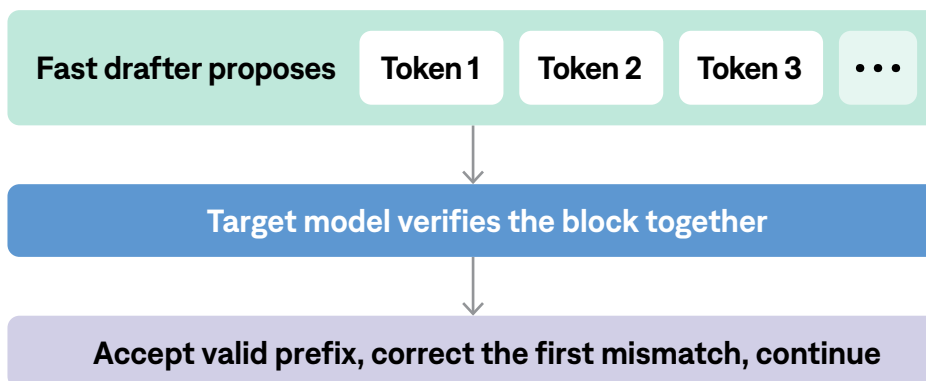
Speculative decoding fits well in a memory-bound regime because it does not ask for more bandwidth; it recovers the compute that memory-bound decoding leaves idle [1]. Another important property of speculative decoding is that it is lossless. Because acceptance is governed by rejection sampling against the target model’s own distribution, the sequence of tokens ultimately emitted has the same distribution as if the target model had generated every token itself one at a time [2][3]. This distinguishes speculative decoding from techniques like pruning or quantization, which can trade accuracy for speed.

A useful mental model is shown below:

Traditional decoding



Speculative decoding



Speculative decoding primarily accelerates the decode phase after the first token. It does not inherently reduce prompt processing or the time-to-first token (TTFT) in the prefilling stage. That is visible in the results observed, as some speculative decoding configurations had similar or slightly higher TTFT while still finishing the full response much sooner.

Speculative Decoding Methods

Three methods were implemented in LLiMa and evaluated on the Modalix MLSoC: EAGLE3, DFlash, and Gemma 4 MTP.

EAGLE3

Speculative decoding has evolved through several designs. Early systems used a separate, smaller language model, often from the same family as the target, to draft tokens for the larger model to verify. The EAGLE family took a more closely connected approach, with its drafter using information from inside the target model rather than working as an independent language model. This gives the drafter access to information the target has already computed when processing the prompt or verifying earlier candidates.

The original EAGLE predicts the target model's next top-layer features, then passes those predictions through the target's output layer to produce candidate tokens. It also uses previously sampled tokens to account for uncertainty in the continuation [4]. EAGLE-2 retained this feature-level approach but replaced the fixed draft-tree structure with a dynamic tree that allocates branches according to the drafter's confidence [5]. The draft tree is built with a fixed token budget. At each expansion step, only the highest-confidence branches are kept and extended, so the tree grows within budget rather than expanding without limit.

The target checks the resulting tree in a single forward pass using a structured attention mask, which allows each candidate to attend to the existing context and earlier tokens on its own branch, while excluding competing branches. This lets the target verify multiple possible continuations in parallel.

EAGLE-3 retains dynamic tree drafting but removes the requirement to predict the target's future features, training the drafter to predict tokens directly. It also combines features from lower, middle, and upper layers of the target model to provide richer information for drafting. Its "training-time test" procedure simulates successive drafting steps during training, so the drafter learns to continue from its own intermediate outputs under conditions closer to inference. These changes

allow it to benefit more from additional training data and improve candidate acceptance [6]. Further, the lightweight drafter reuses information from the target, providing an architecture suited to exploring faster inference within the memory and power constraints of edge devices.

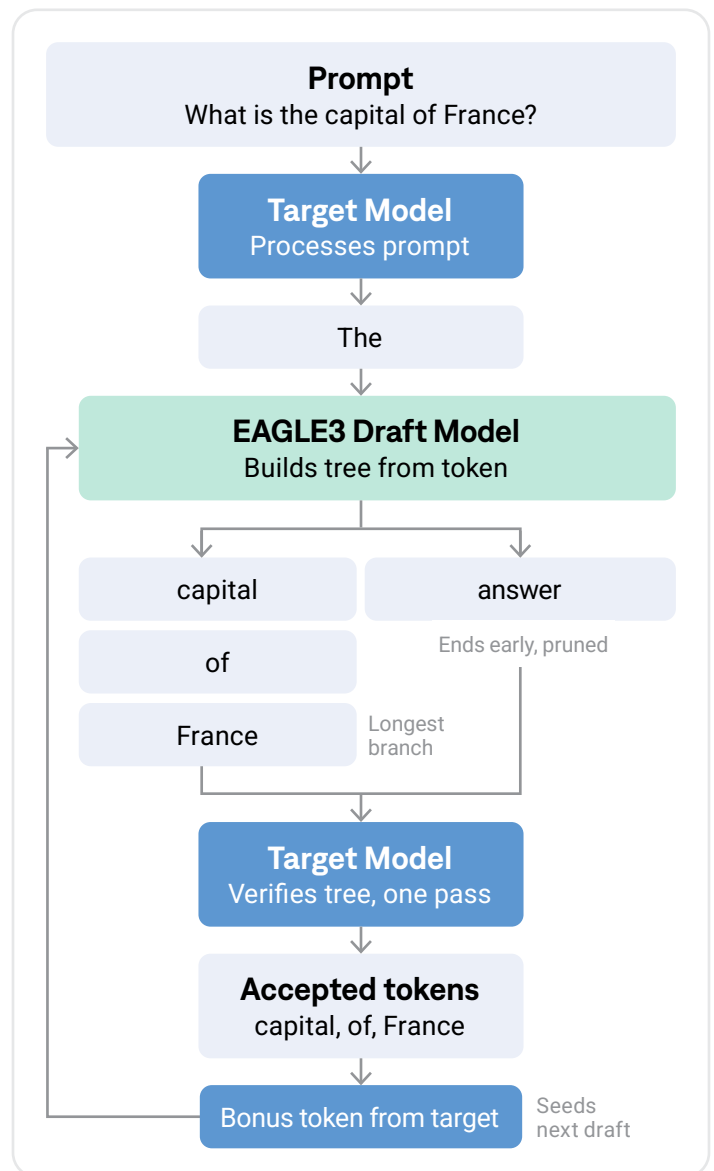


FIGURE 2. Block Diagram – EAGLE3 speculative decoding

Speculative Decoding Methods

DFlash

DFlash changes the drafting problem. A conventional autoregressive drafter creates its proposed tokens one after another, so drafting a longer block adds more serial steps. DFlash uses a lightweight block-diffusion mode instead: it starts with one clean anchor token from the target and masked positions for the rest of the block, then predicts those masked positions together in one draft-model pass. Draft cost therefore does not grow one-for-one with the number of proposed tokens [7].

The drafter is closely conditioned on the target model. During target execution, hidden states are captured at several depths, fused into a compact representation, and supplied to every DFlash layer through its key/value path. This repeated KV injection gives the small drafter access to the target's semantic features throughout its network instead of providing that information only at its input.

The target model remains frozen during DFlash training; its embedding and output head are shared with the drafter, while only the lightweight draft transformer is trained. Training places extra weight on early block positions because a mismatch near the start prevents the later proposals from being committed.

At inference, the full target evaluates the proposed block in one verification pass. LLMa compares target and draft tokens in order, commits the longest matching prefix, then appends the target's token at the first mismatch. Rejected draft state is discarded, so the target remains the authority for generated output. The tested packages used draft blocks of 8 token positions (N8) or 16 token positions (N16). Larger blocks offer more potential progress per verification pass, but later candidates are more likely to diverge from the target model's output. This structure is a good fit for an accelerator because it replaces many small serial draft steps with wider parallel operations, while still protecting target-model behavior during verification.

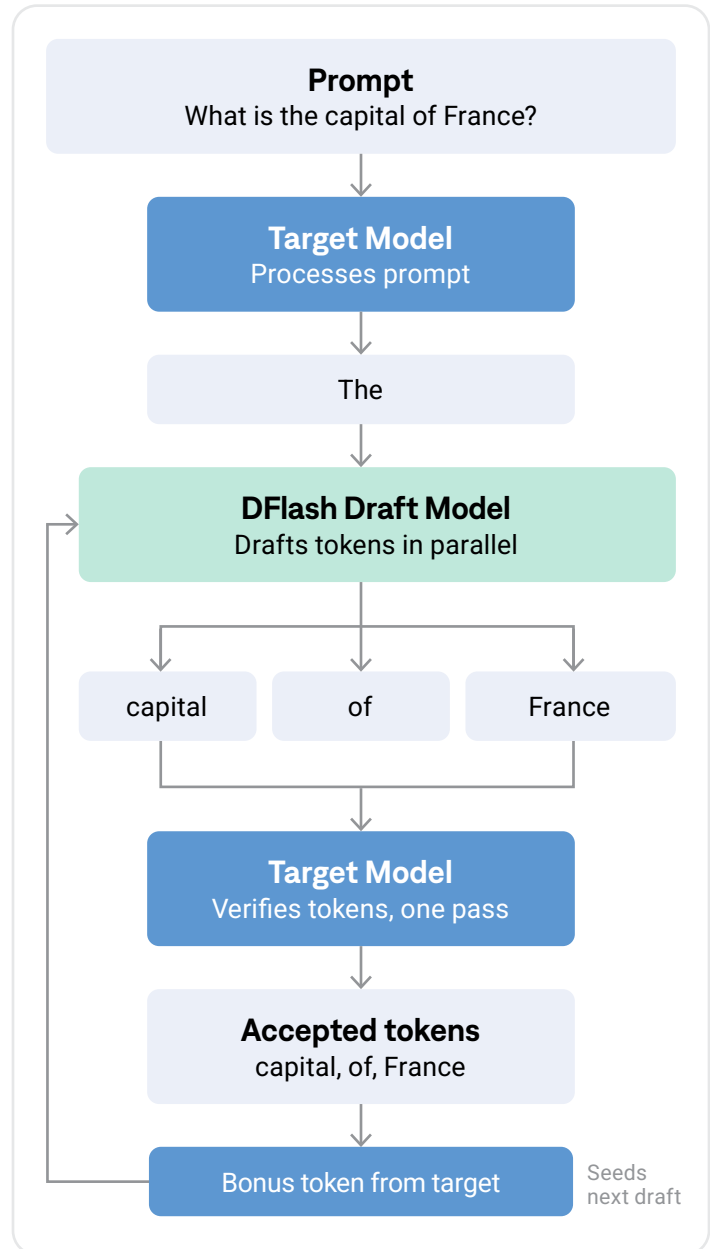


FIGURE 3. Block Diagram – DFlash speculative decoding

Speculative Decoding Methods

Gemma 4 MTP

Gemma 4 multi-token prediction is designed into the model family as a speculative-decoding companion. Rather than pairing the target with an unrelated small language model, the MTP drafter consumes information already produced by Gemma 4. It shares the target's input embedding table and combines the current token embedding with the target's last-layer activation. A projection maps that combined representation into the smaller drafter, which proposes several future tokens before the target verifies them together. See Google's Gemma 4 MTP overview [8].

The output stage is particularly relevant for Gemma 4 E4B. E4B has a 262K-token vocabulary, according to the Gemma 4 model card, so applying a full output projection for every draft step would be expensive. The MTP path first scores token clusters, selects the most likely clusters, and evaluates candidate tokens within them. This narrows the output work while retaining the target's full-vocabulary verification. The target accepts the matching prefix and supplies the correct token at the first mismatch, just as in the common speculative-decoding contract.

The SiMa.ai runtime keeps this exchange on-device. It passes the target hidden state and shared embedding data into the assistant, reuses compatible target KV-cache state for configured shared layers, and executes target verification at the compiled speculative width. This reduces data movement and lets Modalix spend most of its time on dense accelerator work rather than repeatedly rebuilding model context on the CPU.

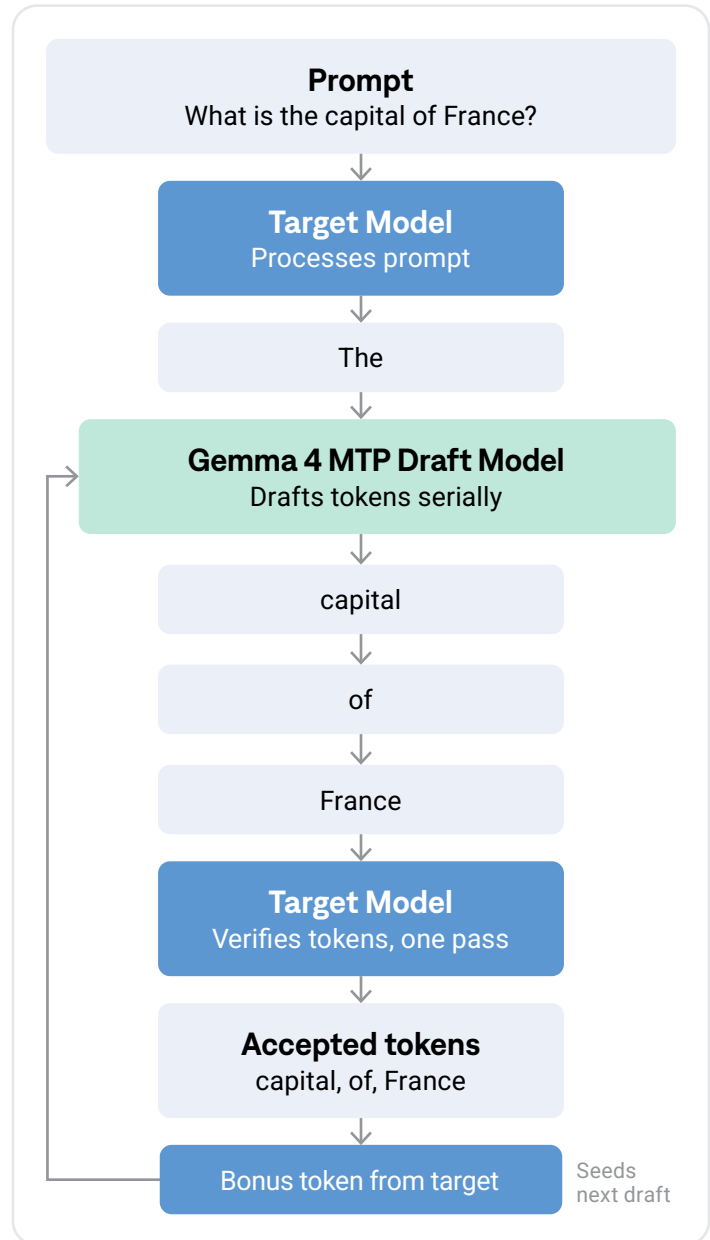


FIGURE 4. Block Diagram – Gemma 4 MTP speculative decoding

Comparison of Methods

All three methods keep the target model in control, but they reduce serial decoding in different ways:

Method	How it drafts	Target information used by the drafter	Main source of acceleration
EAGLE3	Lightweight feature-conditioned token/tree drafting	Fused low-, middle-, and high-level target features	High-quality candidates and multiple candidate paths
DFlash	One parallel block-diffusion pass over anchor plus masked positions	Multi-depth target features injected through every draft layer's KV path	Removes serial dependence among positions within the draft block
Gemma 4 MTP	Gemma-integrated smaller assistant proposes several future tokens	Shared embeddings and the target's last-layer activation	Cheap drafting, clustered vocabulary output, and batched target verification

Benchmarking

EAGLE3, DFlash, and Gemma 4 MTP were each evaluated in two separate tests. The first test measured how quickly the models generate tokens on a variety of prompts. The second measured how long an automotive assistant takes to produce a complete, actionable tool call. We compared each speculative decoding configuration with traditional decoding of the same target model.

Benchmarking

Benchmark 1: Sustained Generation Behavior on Mixed Prompts

Description

Benchmark 1 isolates sustained generation behavior on five fixed prompts, involving: a calculation, Python code, a multiplication table, a truth table, and a short mathematical proof. Each model/prompt combination received one unmeasured warm-up followed by three measured repetitions. Acceptance came from LLaMa's internal token counters; HTTP stream chunks were not treated as tokens.

Goal

Determine how much speculative decoding can improve sustained token generation under favorable conditions.

Prompts

1. A man drives 60 mph for 6 hours. How fast would he have to drive over the next 3 hours to get an average speed of 80 mph?
2. Write a python script to find fibonacci series.
3. Generate an ASCII multiplication table up to 12 x 12.
4. Show a truth table for (A AND B) OR (NOT C).
5. Prove that $\sqrt{2}$ is irrational in under 5 steps.

Example Interactions

Response to Prompt 1 (using Gemma 4 E4B MTP):

Total time is 9 hours, so an 80 mph average requires 720 miles. The first 6 hours at 60 mph cover 360 miles. The remaining 360 miles must be covered in 3 hours, requiring 120 mph.

Response to Prompt 2 (using Qwen3 8B DFlash):

```
def fibonacci_series(n_terms):
    if n_terms <= 0:
        print("Please enter a positive integer.")
        return
    a, b = 0, 1
    fibonacci_numbers = []
    for _ in range(n_terms):
        fibonacci_numbers.append(a)
        a, b = b, a + b
    return fibonacci_numbers
```

Benchmarking

Benchmark 1: Sustained Generation Behavior on Mixed Prompts

Performance Data

Target model	Method	Decode speed: traditional → speculative	Speedup	Mean SoC power: traditional → speculative	Energy/token: traditional → speculative
Llama 3.1 8B	EAGLE3	12.68 → 28.04 TPS	2.21×	7.00 → 6.62 W	0.556 → 0.238 J
Llama 3.1 8B	DFlash N8	12.68 → 29.60 TPS	2.33×	7.00 → 6.96 W	0.556 → 0.238 J
Qwen3 8B	DFlash N16	13.42 → 41.98 TPS	3.13×	7.52 → 7.63 W	0.565 → 0.184 J
Qwen3.5 9B	DFlash N16	11.98 → 41.75 TPS	3.49×	7.47 → 7.46 W	0.627 → 0.182 J
Gemma 4 E4B	MTP	19.77 → 47.65 TPS	2.41×	7.19 → 7.17 W	0.366 → 0.156 J

Observations

Across the five prompts evaluated in Benchmark 1, speculative decoding increased decode throughput by 2.21x to 3.49x, while mean SoC power remained within roughly the same operating range. In other words, more useful tokens are produced in less time, which reduces compute energy per token significantly. For Physical AI systems with constrained energy budgets, that efficiency gain is meaningful alongside performance boost.

Benchmarking

Benchmark 2: Automotive Assistant Tool Calling

Description

Benchmark 2 measures an application outcome rather than a long-form token rate. An in-car assistant receives a natural-language command and returns one structured OpenAI-style function call, which is graded against a frozen expected tool name and exact arguments.

Twenty fixed prompts were examined, covering climate and seat controls, media, navigation, nearby search, and vehicle status. They include direct commands and indirect requests such as “I’m cold.” Each scenario specifies one expected tool and its arguments. Timing begins with the assistant request and ends when the complete structured tool call is returned. It does not include executing the vehicle function or generating a spoken confirmation.

Each configuration received an unmeasured warm-up, then ran all 20 scenarios five times: 100 measured requests per configuration and 900 overall. A response

counted as correct only if it produced the expected structured call and arguments, with no extra call or assistant prose. Specifically:

- Syntactically valid structured output
- Exactly one call
- The expected tool name
- The expected argument names and values
- No additional assistant prose

For example, under the test rules, “I’m cold. Increase the temperature” must produce a call to `adjust_cabin_temperature` for the driver with a two-degree Celsius increase.

Goal

Determine whether faster decoding helps an application produce a usable result sooner.

Example Interactions | Using Gemma 4 E4B MTP

Prompt 1

I’m cold. Increase the temperature.

Response to Prompt 1

```
{
  "name": "adjust_cabin_temperature",
  "arguments": {
    "delta_celsius": 2,
    "zone": "driver"
  }
}
```

Prompt 2

Navigate to 1 Market Street, San Francisco.

Response to Prompt 2

```
navigate_to(destination="1 Market Street,
San Francisco")
```

Benchmarking

Benchmark 2: Automotive Assistant Tool Calling

Performance Data

Target model	Method	Tool-call time: traditional → speculative	Speedup	Mean SoC power: traditional → speculative	Energy/request: traditional → speculative
Llama 3.1 8B	EAGLE3	2.177 → 1.618 s	1.35×	7.17 → 7.16 W	15.62 → 11.58 J
Llama 3.1 8B	DFlash N8	2.177 → 1.204 s	1.81×	7.17 → 7.64 W	15.62 → 9.20 J
Qwen3 8B	DFlash N16	2.543 → 1.500 s	1.69×	8.15 → 8.47 W	20.73 → 12.71 J
Qwen3.5 9B	DFlash N16	3.277 → 2.104 s	1.56×	7.69 → 8.07 W	25.22 → 16.99 J
Gemma 4 E4B	MTP	1.170 → 0.569 s	2.06×	7.23 → 7.22 W	8.46 → 4.12 J

Observations

All models called the correct tools in all 900 requests in Benchmark 2. The application result is stronger than a raw throughput claim: every measured interaction remained correct while the complete structured tool call arrived sooner. The lower energy per request follows from the shorter execution time, even where instantaneous mean power increased slightly. These were 20 fixed, fully specified scenarios, each requiring exactly one tool call. The measured improvement is the time to return the structured call, not the time to carry out the vehicle action. The 900 correct responses establish correctness on this test set, and do not establish a reliability rate for arbitrary requests.

Synopsis of Benchmarks 1 and 2

Across the five selected prompts, speculative decoding increased decode throughput by **2.21× to 3.49×** relative to traditional decoding of the same target models. In the automotive assistant test, all **900 measured requests** produced the expected structured tool call. Speculative decoding configurations completed those calls **1.35× to 2.06× faster** and reduced estimated SoC compute energy per request by **25.9% to 51.4%**. Mean SoC power remained broadly similar, so the energy savings came primarily from completing generation sooner.

The methods exhibit different strengths: EAGLE3, paired with **Llama 3.1 8B**, increased curated decode throughput by **2.21×** and completed automotive tool calls **1.35× faster**. DFlash N16 achieved the largest curated speedup: **3.49×** on Qwen3.5 9B, reaching **41.75 tokens/s**. DFlash's

automotive tool-call speedups ranged from **1.56× to 1.81×** across the three tested target models. Gemma 4 E4B MTP achieved the highest curated throughput at **47.65 tokens/s**, or **2.41×** its baseline. It also delivered the largest automotive tool-call speedup, **2.06×**, and the greatest estimated energy reduction per request, **51.4%**.

These results illustrate why decode throughput and complete-response time should be measured separately. Gemma MTP increased automotive decode throughput by **3.13×**, but the complete tool-call speedup was **2.06×**, with a slightly higher time to first token. Faster decoding improved application response time, while prompt processing and other overhead limited the overall gain. Performance will depend on the model, method, and workload.

Conclusion

On-device AI assistants must deliver responsive interactions within limited power and memory budgets. Speculative decoding addresses a significant source of delay by allowing the target model to verify several proposed tokens together instead of generating every token in a separate step. On [SiMa.ai](#) Modalix MLSoCs, EAGLE3, DFlash, and Gemma 4 MTP all improved decode throughput in the token generation tests. Further, in the defined automotive assistant workload, they reduced the time to produce a complete, actionable tool call while lowering estimated SoC compute energy per request.

Faster decode throughput translates directly into product outcomes: lower time-to-response for interactive on-device assistants and better responsiveness for LLM-driven use cases. These benefits align with why customers choose local inference in the first place - data privacy, offline availability, and avoiding recurring per-query costs. Developers can use these results as a starting point for evaluating speculative decoding with their own models, prompts, and response requirements on Modalix.

Explore EAGLE3 on [Hugging Face](#) today!
(DFlash and Gemma 4 MTP available in October 2026.)

References

- [1] R. Pope et al. "Efficiently Scaling Transformer Inference." MLSys 2023. arXiv:2211.05102.
- [2] Y. Leviathan, M. Kalman, Y. Matias. "Fast Inference from Transformers via Speculative Decoding." ICML 2023. arXiv:2211.17192.
- [3] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, J. Jumper. "Accelerating Large Language Model Decoding with Speculative Sampling." arXiv:2302.01318, 2023.
- [4] Y. Li, F. Wei, C. Zhang, H. Zhang. "EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty." ICML 2024. arXiv:2401.15077.
- [5] Y. Li, F. Wei, C. Zhang, H. Zhang. "EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees." EMNLP 2024. arXiv:2406.16858.
- [6] Y. Li, F. Wei, C. Zhang, H. Zhang. "EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test." arXiv:2503.01840, 2025.
- [7] J. Chen, Y. Liang, and Z. Liu, "DFlash: Block Diffusion for Flash Speculative Decoding," arXiv:2602.06036, 2026.
- [8] Google, "Speed-up Gemma 4 with Multi-Token Prediction," Google AI for Developers <https://ai.google.dev/gemma/docs/mtp/overview>
- [9] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, T. Dao. "Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads." arXiv:2401.10774, 2024.



SiMa.ai is a leader in Physical AI, delivering a purpose-built, software-centric platform that brings best-in-class performance, power efficiency, and ease of use to Physical AI applications. The company is focused on scaling Physical AI across robotics, automotive, drones, industrial automation, aerospace and defense, smart vision, and healthcare. Learn more at www.sima.ai.