CATS: CASCADED ADAPTIVE TREE SPECULATION FOR MEMORY-LIMITED LLM INFERENCE ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive decoding in large language models (LLMs) is inherently memory-bound: every generation step requires loading the model weights and intermediate results from memory (e.g., high-bandwidth memory (HBM) on GPU servers), making throughput limited by memory bandwidth rather than compute. Speculative decoding addresses this by enabling parallel verification of multiple draft tokens, effectively amortizing the cost of each target-model call. However, existing speculative decoding methods are designed under the assumption that HBM is sufficiently large to hold both the target model and an auxiliary draft model simultaneously—an assumption that breaks down on memory-constrained devices such as edge platforms with limited DRAM. We analyze the inference bottleneck in this memory-limited regime and propose CATS, a self-speculative decoding framework that conducts cascaded verification and correction based on the memory budget and parameter offloading patterns on memory-limited devices. This design maximizes token acceptance rate and end-to-end speedup while keeping the peak memory footprint on the device equal to that of the target model alone. We evaluate CATS on different models across five benchmarks on real edge devices. CATS achieves a wall-clock speedup of up to **5.08**× with no degradation in generation quality, outperforming the SOTA method by up to **1.60**× under edge memory constraints.

1 INTRODUCTION

Deploying Large Language Models (LLMs) for efficient inference remains one of the central challenges in modern machine learning systems. LLM inference must generate tokens one at a time in an autoregressive loop—each step conditioned on all prior outputs and requiring a full model forward pass. This sequential structure is inherently memory-bound: every generation step loads the model weights and intermediate results from memory (e.g., High-Bandwidth Memory (HBM) for GPU servers) to on-chip compute units, and throughput is governed by memory bandwidth rather than arithmetic capacity (Vellaisamy et al., 2026; Alizadeh et al., 2024). Speculative decoding was proposed to break this bottleneck through *parallel decoding* (Leviathan et al., 2023; Chen et al., 2023; Stern et al., 2018): a lightweight draft model speculatively generates a sequence of candidate tokens, which the larger target model then verifies in a single batched forward pass. Because the target model processes multiple tokens simultaneously, the per-token memory bandwidth cost is amortized over several accepted tokens, substantially improving throughput while preserving the exact output distribution.

However, this throughput gain comes at a structural cost: classical speculative decoding introduces a *second* set of weights—the auxiliary draft model—that must coexist with the target model in device memory, inflating both the static memory footprint and the per-step bandwidth traffic. Self-speculative decoding methods (Zhang et al., 2024; Elhoushi et al., 2024; Xia et al., 2025; Liu et al., 2024a) emerged as a practical response to this added memory cost: by generating drafts from a shallow sub-network of the target model itself, they eliminate the need for a separate draft model. However, they still need to introduce additional adapter parameters for drafting, and the performance of these methods often suffers from the sub-network's limited drafting capacity.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

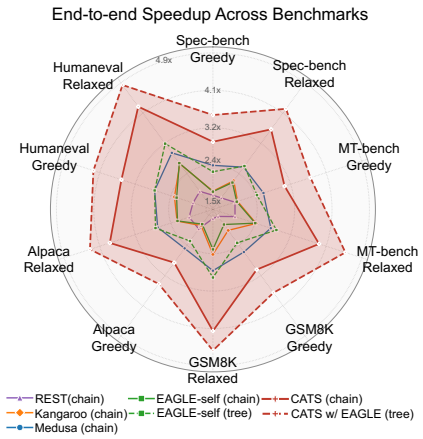


Figure 1: End-to-end speedup across five benchmarks on Vicuna-7B.

A more fundamental limitation runs through *all* existing speculative decoding approaches: they assume model weights remain resident in device memory (e.g., HBM for servers and DRAM for edge devices) for the duration of inference. On large-scale servers, this holds, but on memory-limited platforms such as edge devices, DRAM capacity falls well short of a single 7B-parameter model and must be shared with the operating system—so weights must be streamed from flash memories on every forward pass (Alizadeh et al., 2024), making flash↔DRAM the rate-limiting transfer rather than DRAM↔on-chip memory. Every design choice in existing methods—draft depth, verification strategy, acceptance policy—targets the memory-resident regime and is misaligned with this memory-constrained setting. Auxiliary-model approaches carry a further cost, though only where the resident budget is tight relative to the target: a separate draft model adds both memory capacity demands and extra transfer traffic at every speculative step. A compact drafter alongside an 8B target fits comfortably and is not penalized in this way; but on a large target under a small budget, where the auxiliary draft can occupy 18–33% of the weights the device is able to keep resident, it compounds the bottleneck it was meant to relieve.

We investigate this memory-limited inference bottleneck and propose CATS (**C**ascaded **A**daptive **T**ree **S**peculation), a self-speculative decoding framework that performs cascaded verification to minimize inference time within the memory budget. CATS organizes inference into three stages whose layer boundaries are determined by the available DRAM budget: (1) a **draft** stage using a shallow sub-network to draft candidate tokens by repeating draft iterations; (2) a **shallow verification** stage using intermediate layers under the DRAM budget, streamed from flash once per cycle, to check draft tokens and produce correction candidates in parallel; and (3) a **target verify** stage where the remaining layers are offloaded from the flash chunk by chunk to process a tree-structured input over the draft trace and all correction branches to select the longest accepted prefix. We also propose a **Reduced KL Loss** that focuses distillation supervision on high-probability tokens to maximize the sub-network's drafting and verification capabilities.

Our main contributions are as follows:

- We identify flash↔DRAM data movement as the dominant bottleneck for speculative decoding on edge devices—a constraint entirely overlooked by existing methods—and propose CATS, a cascaded self-speculative decoding framework for this memory-constrained regime.
- CATS is **model-agnostic**: the cascaded framework applies to any transformer-based LLM without architectural modifications.
- We implement our framework on edge devices. Extensive experiments on Vicuna-7B/13B, LLaMA2-7B/13B and LLaMA3-8B across five benchmarks demonstrate that CATS achieves up to **5.08**× speedup, outperforming all compared self-speculative decoding baselines including KANGAROO (Liu et al., 2024a), MEDUSA (Cai et al., 2024), EAGLE (Li et al., 2024b) and EAGLE-3 (Li et al., 2025) under the memory-limited setting.

2 RELATED WORK

Speculative decoding. Speculative decoding verifies multiple draft tokens in one target-model pass (Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2023). Extensions include tree verification (Miao et al., 2024; Chen et al., 2024; Wang et al., 2025; Xiong et al., 2024), draft distillation (Zhou et al., 2024; Liu et al., 2024b; Du et al., 2024), retrieval (He et al., 2024; Li et al., 2024a; Oliaro et al., 2025), and lookahead (Fu et al., 2024). The EAGLE series (Li et al., 2024b;c; 2025) combines feature-level drafting with token trees. Auxiliary drafters add weight storage and transfer costs under tight memory budgets; see Xia et al. (2024) for a broader survey.

Self-speculative decoding and prediction heads. Self-speculative methods reuse target layers for drafting through early exit (Zhang et al., 2024; Elhoushi et al., 2024) or layer selection (Xia et al., 2025; Cha et al., 2026). Kangaroo (Liu et al., 2024a) trains a lightweight adapter to improve shallow-layer predictions, while Medusa and Hydra (Cai et al., 2024; Ankner et al., 2024) use multiple prediction heads. Consistency training (Kou et al., 2024; Guo & Ermon, 2025) and multi-token objectives (Gloeckle et al., 2024; Qin et al., 2025; Monea et al., 2023) provide complementary approaches. Our cascade uses intermediate layers to improve candidate acceptance while keeping autoregressive drafting shallow.

Memory-constrained inference. LLM in a Flash (Alizadeh et al., 2024) reduces flash-to-DRAM traffic through windowed loading and row-column bundling; PowerInfer (Song et al., 2024) exploits activation sparsity. Related offloading systems (Ren et al., 2025; Chen et al., 2025) also address limited memory capacity. CATS complements these approaches by amortizing weight movement across multiple generated tokens.

3 PRELIMINARY AND MOTIVATION

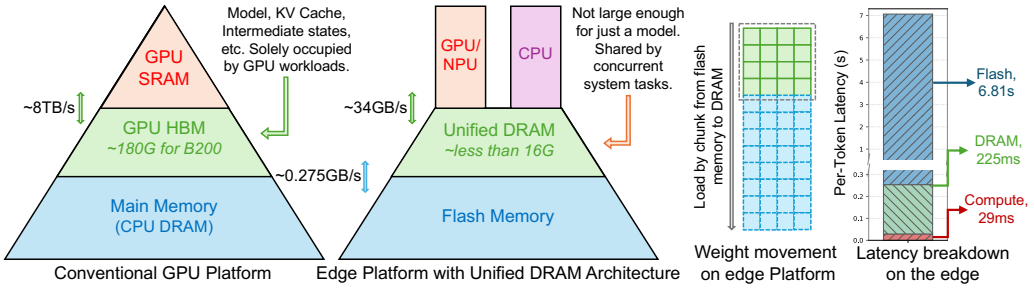


Figure 2: Memory hierarchy on server vs. edge, with measured per-token latency breakdown for autoregressive Vicuna-7B inference. *Server (B200):* model weights reside in HBM; the binding bottleneck is HBM↔SRAM bandwidth, and per-token latency is compute-dominated. *Edge (Jetson AGX Orin):* DRAM cannot hold the full model, so weights must be staged from flash on every forward pass; flash↔DRAM transfer dominates per-token latency.

The structural mismatch identified in Section 1—that all speculative decoding methods assume memory-resident weights, a condition that fails on edge—stems from a fundamental difference in memory hierarchy between the two deployment regimes, which Figure 2 makes explicit. On a conventional GPU platform (e.g., NVIDIA B200), HBM provides ∼180 GB at ∼8 TB/s, comfortably holding the target model, an auxiliary draft model, and all intermediate states simultaneously; the binding cost during decoding is the fast HBM↔SRAM path. On an edge platform with a unified DRAM architecture, the GPU/NPU and CPU share a single DRAM pool of $\lesssim$16 GB—difficult to host even a single 7B-parameter model (∼14 GB), let alone an extra draft model. Transfers between DRAM and flash reach only ∼0.275 GB/s on the AGX Orin. During inference, model weights must be loaded chunk by chunk from flash into DRAM on every forward pass (Alizadeh et al., 2024), and the binding bottleneck shifts to this flash↔DRAM movement, as reflected in the latency breakdown in Figure 2. This bottleneck shift makes existing speculative decoding methods inferior in memory-limited settings. Our profiling on the memory-limited AGX Orin visualizes this effect in Figure 3: the end-to-end speedups of Kangaroo and EAGLE drop by 19.8% and 24.5%, respectively, when moving from server to edge, relative to vanilla inference on the corresponding platforms. This effect is more severe for methods like EAGLE that introduce an additional draft model, since these components must also be staged through the constrained edge memory hierarchy.

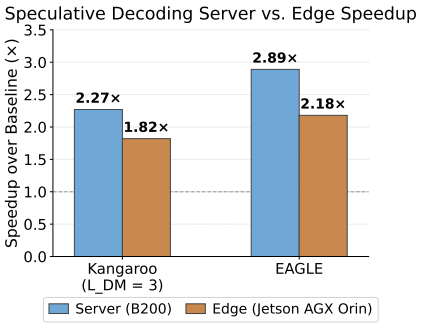


Figure 3: End-to-end speedup on B200 vs. Jetson AGX Orin.

Our measurements further isolate this bottleneck: direct eMMC reads with `O_DIRECT` achieve 0.270–$0.284\,\mathrm{GB/s}$, with flash reads accounting for 98.9% of the measured layer-loading and computation chain. For Vicuna-7B at a context length of 2048, the KV cache occupies $1.074\,\mathrm{GB}$, compared with $12.95\,\mathrm{GB}$ of decoder weights. This motivates our focus on weight movement, although KV growth also reduces the capacity available for resident weights. Appendix D reports the bandwidth, latency decomposition, and KV-cache analysis.

4 METHODOLOGY

In this section, we formalize the memory-adaptive three-stage inference framework that is central to CATS and describe its three-stage inference pipeline. We then detail the verification tree construction and the Reduced KL Loss used to train the draft and shallow-verifier adapters. The detailed algorithm can be found in Appendix A.

4.1 NOTATION

We use *target model* (TM) for the target LLM, *draft model* (DM) for layers 1 to L_{DM} plus a lightweight adapter, which is well-trained to transform the shallow-layer outputs of the TM into calibrated representations suitable for next-token prediction, and *shallow verifier* (SV) for layers $L_{\mathrm{DM}}{+}1$ to L_{SV} plus its adapter. A forward through layers 1 to L_{DM} is a *drafting pass*, through $L_{\mathrm{DM}}{+}1$ to L_{SV} an *SV pass*, and through $L_{\mathrm{SV}}{+}1$ to L_{final} the *final pass*; $\bar{\gamma}$ denotes the number of drafting passes per decoding cycle. Here, L_{DM} denotes the draft boundary, i.e., the last layer included in the draft model; L_{SV} denotes the shallow-verification boundary, i.e., the last layer used by the shallow verifier; and L_{final} denotes the final layer of the target model.

4.2 CASCADED VERIFICATION DESIGN

The left part of Figure 4 illustrates the full CATS pipeline. Each decoding cycle alternates between flash-to-DRAM weight movement and computation across three stages. In this paper, we focus on weight movement because edge inference is typically latency-oriented and operates at small batch sizes, where the KV cache is not the dominant memory consumer as in large-batch serving. When KV-cache pressure becomes relevant, it can be treated as part of the streamed state alongside model weights, so the same memory-adaptive scheduling principle still applies.

Drafting loop. CATS uses a compact DM that is small enough to fit within the DRAM budget of the edge device. The draft boundary L_{DM} is chosen not simply to maximize the number of layers that can be loaded into DRAM, but to balance three constraints: DRAM capacity, limited edge compute throughput, and memory bandwidth. A deeper draft model may improve the drafting quality, but it also increases the cost of each autoregressive drafting step. We therefore keep the draft model intentionally shallow and perform $\bar{\gamma}$ sequential drafting passes to generate $\bar{\gamma}$ candidate tokens with reasonable latency. To preserve draft quality under this small-model constraint, the draft adapter is trained with the reduced KL loss introduced later.

Shallow verification pass. The shallow-verifier boundary is selected according to the memory budget. At the start of each cycle, the intermediate sub-network (layers $L_{\mathrm{DM}}+1$ through L_{SV}) is moved to DRAM with the draft model. The $\bar{\gamma}$ candidates are then forwarded in parallel through these intermediate layers, and the SV adapter decodes a verification token at each position. Each SV token is compared strictly against the corresponding draft token: matching positions are passed through to the target stage; mismatching positions yield a correction candidate. When mismatches happen, the draft tokens and corrections are assembled into a verification tree as shown in Figure 4. The corrections are *per-position siblings* conditioned on the draft trunk rather than an autoregressive chain: each attends to the trunk prefix preceding its own position and to itself, but not to the other corrections, so accepting any one of them terminates the cycle by construction (Appendix B). The correction tokens are then re-forwarded from layer 1 through L_{SV} under tree-masked attention to obtain their hidden states—reusing the already-loaded draft and SV layers—at zero additional flash-transfer cost. This DM-to-SV stage acts as an internal speculative decoding process that uses the limited DRAM capacity for parallel verification.

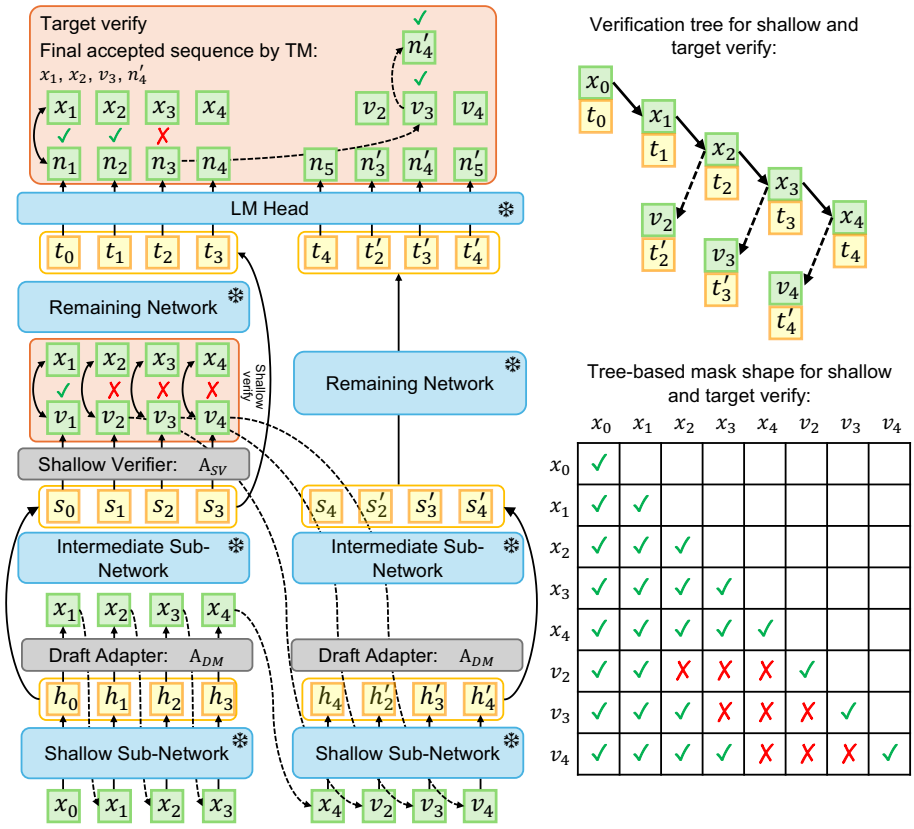


Figure 4: Pipeline of CATS. **Left**: three-stage decoding cycle showing interleaved flash↔DRAM transfers and GPU computation. The bottom blue block (draft sub-network) and intermediate SV layers (middle blue block) are streamed from flash once per cycle for drafting and shallow verification; the remaining target layers (top blue block) are streamed during the final pass. Green blocks: token embeddings; yellow blocks: hidden states; yellow-bordered box: parallel forward computation; orange blocks: verification decision. The draft and SV adapter structures follow Liu et al. (2024a). **Upper right**: verification tree structure. **Lower right**: tree-based attention mask encoding the main draft sequence and SV correction branches for the batched final pass. The symbols x, h, s, t, v and the primed states are defined in Appendix B.

Target verification pass. The remaining layers $L_{\text{SV}}+1$ through L_{total} are then streamed from flash into DRAM chunk by chunk in the rest of the sequential transfer. The assembled verification tree is processed in a single batched forward pass under the corresponding tree attention mask. The target model can verify the main draft branch following greedy decoding or via typical acceptance (Cai et al., 2024); at each rejected position, the SV's correction candidate is evaluated under the same criterion. The longest accepted prefix is committed to the output, and the KV cache is updated accordingly.

4.3 ADAPTER DESIGN AND TRAINING

Draft and verification adapters follow Liu et al. (2024a): two normalization layers, one multi-head attention layer, and a shared LM head. We train them with a **Reduced KL Loss**, concentrating distillation on the target's top-K tokens:

$$\mathcal{L}_{\text{top-}K} = -\frac{1}{|M|} \sum_{t \in M} \sum_{v \in \mathcal{T}_K(t)} \tilde{p}_{\text{target}}(v \mid t) \cdot \log q_{\text{draft}}(v \mid t). \tag{1}$$

Here, M contains unmasked positions, $\mathcal{T}_K(t)$ is the target's top-K support, $\tilde{p}_{\text{target}}$ is renormalized over that support, and q_{draft} is the adapter distribution. Appendix K gives the full-vocabulary objective and derivation.

5 EXPERIMENTS

Models, tasks, baselines, and metrics. We conducted experiments on Vicuna models (7B and 13B) (Chiang et al., 2023), LLaMA2 models (7B and 13B) (Touvron et al., 2023), and Llama-3 (8B) (Grattafiori et al., 2024), which are mainstream LLMs widely tested on different self-speculative decoding algorithms. We evaluated CATS across multiple tasks, including multi-turn dialogue, code generation, mathematical reasoning, and instruction following, using Spec-bench (Xia et al., 2024), MT-bench (Zheng et al., 2023), GSM8K (Cobbe et al., 2021), Alpaca (Taori et al., 2023), and HumanEval (Chen et al., 2021).

Unless otherwise stated, we set the drafting steps $\bar{\gamma} = 5$, choose a compact draft boundary L_{DM}=3, and set the shallow-verifier boundary to L_{SV}=15. Following the design principle in Section 4, L_{DM} is kept shallow to balance DRAM capacity, compute throughput, and repeated drafting cost, while L_{SV}=15 corresponds to representative edge-memory budgets of 8 GB for 7B models and 12 GB for 13B models. We compare against representative speculative decoding algorithms, including REST (He et al., 2024), Lookahead decoding (Fu et al., 2024), Kangaroo (Liu et al., 2024a), Medusa (Cai et al., 2024), and EAGLE (Li et al., 2024b). Since our cascaded verification framework is orthogonal to EAGLE's high-probability candidate branching, CATS w/ EAGLE adds these branches to our verification tree to enlarge the parallel candidate set without additional target-model forward passes. Beyond the self-speculative setting, we also extend CATS to EAGLE-3 (Li et al., 2025), a state-of-the-art speculative decoding method with an auxiliary drafter, on Llama-3.1-8B-Instruct (CATS w/ EAGLE-3).

We report the mean accepted length and end-to-end speedup. Mean accepted length measures the average number of tokens accepted per target-model verification and is platform-independent, while end-to-end speedup captures wall-clock acceleration on the target platform. We used an NVIDIA Jetson AGX Orin to evaluate mean accepted length and end-to-end speedup.

Training. Following Medusa, we fine-tuned our adapters on 68,000 ShareGPT samples (Chiang et al., 2023). We used learning rates of $1e{-}6$ for Vicuna-7B/13B and $1e{-}5$ for LLaMA2-7B/13B and LLaMA3-8B. All adapter-training runs used two B200 GPUs and completed within 13 hours.

Table 1: Mean accepted length (τ) and end-to-end speedup ($S\uparrow$) on Vicuna-7B across five benchmarks under greedy and relaxed (temperature = 0.7) decoding. Kangaroo and CATS use drafting steps = 5; Medusa uses 2 heads; EAGLE is re-implemented in a self-speculative structure with drafting steps = 5. In tree base decoding, we set top-K = 10. Lookahead is greedy-only.

Model	Algorithm	Spec-bench		MT-bench		GSM8K		Alpaca		HumanEval		Mean	
		τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$
Greedy													
Vicuna-7b chain base	REST(He et al., 2024)	1.6243	1.66×	1.8239	1.86×	1.5045	1.54×	1.8406	1.88×	1.7670	1.81×	1.7121	1.75×
	Lookahead(Fu et al., 2024)	2.2785	2.29×	2.2928	2.30×	2.7010	2.72×	2.0623	2.08×	2.5128	2.53×	2.3695	2.38×
	Kangaroo(Liu et al., 2024a)	2.2736	1.75×	2.4863	1.92×	2.4856	1.92×	2.3131	1.79×	2.8979	2.24×	2.4914	1.92×
	Medusa(Cai et al., 2024)	2.3116	2.32×	2.5115	2.52×	2.4966	2.51×	2.3917	2.41×	2.6832	2.70×	2.4789	2.49×
	EAGLE-self(Li et al., 2024b)	2.2995	1.74×	2.5151	1.90×	2.3213	1.76×	2.3104	1.75×	2.9005	2.19×	2.4694	1.87×
	CATS	**2.8816**	**2.99**×	**3.0520**	**3.16**×	**3.0198**	**3.14**×	**2.8280**	**2.94**×	**3.5174**	**3.65**×	**3.0598**	**3.18**×
Vicuna-7b tree base	EAGLE-self, tree	2.8943	2.18×	3.1535	2.37×	3.0059	2.26×	2.9306	2.21×	3.5949	2.70×	3.1158	2.34×
	CATS w/ EAGLE	**3.5097**	**3.51**×	**3.6938**	**3.69**×	**3.6869**	**3.70**×	**3.4339**	**3.45**×	**4.2009**	**4.21**×	**3.7050**	**3.71**×
Relaxed (temperature = 0.7)													
Vicuna-7b chain base	REST	1.6236	1.66×	1.8216	1.86×	1.5192	1.55×	1.8594	1.90×	1.7984	1.84×	1.7244	1.76×
	Kangaroo	2.7777	2.13×	3.0712	2.35×	3.0513	2.34×	2.8348	2.18×	3.3703	2.59×	3.0211	2.32×
	Medusa	2.4884	2.50×	2.6802	2.69×	2.6837	2.70×	2.6179	2.63×	2.8760	2.89×	2.6692	2.68×
	EAGLE-self	2.7487	2.08×	3.0753	2.33×	2.9515	2.23×	2.8644	2.17×	3.4546	2.61×	3.0189	2.28×
	CATS	**3.4791**	**3.60**×	**3.7597**	**3.89**×	**3.9652**	**4.12**×	**3.6742**	**3.82**×	**4.0802**	**4.24**×	**3.7917**	**3.94**×
Vicuna-7b tree base	EAGLE-self	3.3494	2.53×	3.7271	2.82×	3.7629	2.85×	3.5334	2.68×	4.1547	3.14×	3.7055	2.80×
	CATS w/ EAGLE	**4.1802**	**4.18**×	**4.5229**	**4.52**×	**4.5345**	**4.55**×	**4.2745**	**4.29**×	**4.8408**	**4.85**×	**4.4706**	**4.48**×

5.1 MAIN RESULTS

Mean accepted length and end-to-end speedup. Tables 1 and 2 report the main results. Since EAGLE was originally designed around feature-level speculative decoding, we re-implemented it in our self-speculative setting and evaluated both chain and tree decoding. In the edge setting, where each decoding loop is constrained by limited compute and weight-transfer bandwidth, increasing the mean accepted length directly reduces the number of costly inference loops and is reflected in the measured end-to-end speedup.

Table 2: Mean accepted length (τ) and end-to-end speedup ($S\uparrow$) on Vicuna-13B and LLaMA2-7B/13B under greedy and relaxed (temperature = 0.7) decoding. Methods and settings follow Table 1.

Model	Algorithm	Spec-bench		MT-bench		GSM8K		Alpaca		HumanEval		Mean	
		τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$	τ	$S\uparrow$
Greedy													
Vicuna-13b chain base	Kangaroo	2.1937	2.19×	2.4184	2.41×	2.3848	2.38×	2.2226	2.22×	2.9183	2.91×	2.4276	2.42×
	Medusa	2.3920	2.03×	2.5833	2.19×	2.5824	2.20×	2.3959	2.04×	2.8035	2.39×	2.5514	2.17×
	EAGLE-self	2.1952	2.18×	2.3831	2.37×	2.2031	2.19×	2.2183	2.21×	2.9163	2.90×	2.3832	2.37×
	CATS	**2.6588**	**2.64**×	**2.7946**	**2.77**×	**2.7636**	**2.74**×	**2.5493**	**2.53**×	**3.2218**	**3.20**×	**2.7976**	**2.78**×
Vicuna-13b tree base	EAGLE-self, tree	2.8035	2.77×	3.0423	3.01×	2.8760	2.85×	2.8337	2.81×	3.6168	3.58×	3.0345	3.00×
	CATS w/ EAGLE	**3.4708**	**3.42**×	**3.6897**	**3.64**×	**3.5481**	**3.50**×	**3.3568**	**3.32**×	**4.2428**	**4.19**×	**3.6616**	**3.61**×
Llama2-7b chain base	Kangaroo	3.8176	3.29×	4.1930	3.61×	4.4296	3.82×	4.0431	3.49×	4.2946	3.70×	4.1534	3.58×
	Medusa	1.6660	1.67×	1.7817	1.78×	1.7358	1.74×	1.7758	1.78×	1.9657	1.97×	1.7850	1.78×
	EAGLE-self	3.8068	3.28×	4.1813	3.60×	4.7597	4.10×	4.0429	3.49×	4.0588	3.50×	4.1699	3.60×
	CATS	**4.3335**	**4.33**×	**4.6181**	**4.62**×	**5.1055**	**5.10**×	**4.4642**	**4.46**×	**4.7240**	**4.72**×	**4.6491**	**4.65**×
Llama2-7b tree base	EAGLE-self, tree	4.3529	3.75×	4.7199	4.07×	5.0954	4.39×	4.5393	3.91×	4.5866	3.95×	4.6588	4.02×
	CATS w/ EAGLE	**4.8318**	**4.83**×	**5.0634**	**5.06**×	**5.4185**	**5.42**×	**4.9101**	**4.91**×	**5.1741**	**5.17**×	**5.0796**	**5.08**×
Llama2-13b chain base	Kangaroo	3.5409	3.53×	3.9369	3.92×	4.2421	4.23×	3.9754	3.97×	4.1133	4.10×	3.9617	3.95×
	Medusa	1.8250	1.55×	1.9737	1.68×	1.9721	1.68×	1.9425	1.65×	1.9425	1.65×	1.9312	1.64×
	EAGLE-self	3.5478	3.53×	3.9754	3.96×	3.8572	3.84×	3.9805	3.97×	4.1082	4.09×	3.8938	3.88×
	CATS	**4.0570**	**4.02**×	**4.2752**	**4.24**×	**4.3415**	**4.31**×	**4.3696**	**4.34**×	**4.6959**	**4.66**×	**4.3478**	**4.32**×
Llama2-13b tree base	EAGLE-self, tree	4.0829	4.04×	4.4699	4.42×	4.2793	4.24×	4.4566	4.42×	4.6993	4.65×	4.3976	4.35×
	CATS w/ EAGLE	**4.5707**	**4.51**×	**4.8191**	**4.75**×	**4.8534**	**4.79**×	**4.7998**	**4.75**×	**5.0178**	**4.95**×	**4.8122**	**4.75**×
Relaxed (temperature = 0.7)													
Vicuna-13b chain base	Kangaroo	2.4889	2.48×	2.7894	2.78×	2.7695	2.76×	2.5927	2.59×	3.2753	3.27×	2.7832	2.78×
	Medusa	2.5852	2.20×	2.7905	2.37×	2.7920	2.38×	2.7148	2.31×	2.9535	2.51×	2.7672	2.35×
	EAGLE-self	2.5144	2.50×	2.8444	2.83×	2.5893	2.58×	2.6609	2.65×	3.2212	3.20×	2.7660	2.75×
	CATS	**2.9743**	**2.95**×	**3.2421**	**3.22**×	**3.0736**	**3.05**×	**2.9963**	**2.98**×	**3.4792**	**3.45**×	**3.1531**	**3.13**×
Vicuna-13b tree base	EAGLE-self, tree	3.1934	3.16×	3.5232	3.49×	3.3631	3.33×	3.3051	3.28×	3.9784	3.94×	3.4726	3.44×
	CATS w/ EAGLE	**3.9851**	**3.93**×	**4.2360**	**4.18**×	**4.1399**	**4.09**×	**4.0683**	**4.02**×	**4.7269**	**4.67**×	**4.2312**	**4.18**×
Llama2-7b chain base	Kangaroo	4.4849	3.87×	4.9609	4.28×	4.8506	4.18×	4.9796	4.29×	4.0587	3.50×	4.6669	4.02×
	Medusa	1.7482	1.75×	1.8667	1.87×	2.1875	2.19×	1.8955	1.90×	2.2848	2.28×	1.9965	2.00×
	EAGLE-self	4.4910	3.87×	4.9756	4.29×	4.9139	4.24×	4.9867	4.30×	4.2687	3.68×	4.7272	4.08×
	CATS	**4.8831**	**4.88**×	**5.2004**	**5.20**×	**5.1870**	**5.19**×	**5.1976**	**5.20**×	**4.4388**	**4.44**×	**4.9814**	**4.98**×
Llama2-7b tree base	EAGLE-self, tree	4.9713	4.29×	5.2409	4.52×	5.2814	4.55×	5.3228	4.59×	4.7703	4.11×	5.1173	4.41×
	CATS w/ EAGLE	**5.3143**	**5.31**×	**5.5818**	**5.58**×	**5.5096**	**5.51**×	**5.5601**	**5.56**×	**4.9882**	**4.99**×	**5.3908**	**5.39**×
Llama2-13b chain base	Kangaroo	4.1486	4.14×	4.6991	4.68×	4.6031	4.59×	4.7376	4.73×	4.6950	4.68×	4.5767	4.56×
	Medusa	1.9322	1.64×	2.0646	1.75×	2.2499	1.92×	2.0718	1.76×	2.2848	1.95×	2.1207	1.80×
	EAGLE-self	4.1452	4.12×	4.7032	4.68×	4.5883	4.57×	4.7401	4.72×	4.7135	4.69×	4.5781	4.56×
	CATS	**4.6461**	**4.61**×	**4.9133**	**4.87**×	**5.3102**	**5.27**×	**4.9464**	**4.92**×	**5.0678**	**5.03**×	**4.9768**	**4.94**×
Llama2-13b tree base	EAGLE-self, tree	4.7176	4.67×	5.1051	5.05×	5.1449	5.10×	5.0338	4.99×	5.1758	5.12×	5.0354	4.98×
	CATS w/ EAGLE	**5.0813**	**5.01**×	**5.3499**	**5.28**×	**5.5524**	**5.48**×	**5.3241**	**5.27**×	**5.3969**	**5.33**×	**5.3409**	**5.27**×

Tables 1 and 2 show consistent gains across target models and decoding regimes. On Vicuna-7B under greedy decoding, CATS accepts 3.06 tokens per target call, a 23–24% improvement over the self-speculative chain baselines, and achieves 3.18× speedup versus Kangaroo's 1.92×. These gains persist under relaxed decoding and on larger targets. On LLaMA2-7B/13B, CATS achieves greedy speedups of 4.65×/4.32×, rising to 5.08×/4.75× with EAGLE-style tree decoding.

Table 3: End-to-end speedup ($S\uparrow$) over autoregressive decoding on Llama-3-8B-Instruct, on a Jetson AGX Orin with the resident-weight budget fixed at 8 GB. In each benchmark, the left column (G) is greedy decoding and the right column (R) is relaxed acceptance at temperature 0.7; \ marks a method that does not support relaxed decoding.

Model	Algorithm	Spec-bench		MT-bench		GSM8K		Alpaca		HumanEval		Mean	
		G	R	G	R	G	R	G	R	G	R	G	R
Llama-3-8b chain base	REST	1.44×	\	1.53×	\	1.39×	\	1.54×	\	1.64×	\	1.51×	\
	Lookahead	1.95×	\	2.04×	\	2.18×	\	1.87×	\	2.11×	\	2.03×	\
	Kangaroo	2.05×	2.44×	2.21×	2.70×	2.20×	2.50×	2.06×	2.49×	2.82×	3.25×	2.27×	2.68×
	Medusa	1.60×	1.67×	1.60×	1.67×	1.75×	1.80×	1.57×	1.65×	1.73×	1.78×	1.65×	1.71×
	EAGLE-self	2.06×	2.48×	2.19×	2.69×	2.18×	2.41×	2.04×	2.44×	2.74×	3.06×	2.25×	2.61×
	CATS	**2.74**×	**3.31**×	**2.87**×	**3.53**×	**2.99**×	**3.41**×	**2.65**×	**3.24**×	**3.62**×	**4.05**×	**2.98**×	**3.51**×
Llama-3-8b tree base	EAGLE-self, tree	2.67×	3.15×	2.84×	3.44×	2.88×	3.13×	2.65×	3.13×	3.47×	3.79×	2.90×	3.33×
	CATS w/ EAGLE	**3.09**×	**3.68**×	**3.24**×	**3.94**×	**3.37**×	**3.71**×	**3.00**×	**3.59**×	**3.97**×	**4.34**×	**3.33**×	**3.85**×
Llama-3.1-8b EAGLE-3 base	EAGLE-3	5.31×	5.02×	5.52×	5.12×	5.64×	5.34×	5.37×	5.05×	5.90×	5.66×	5.55×	5.24×
	CATS w/ EAGLE-3	**5.71**×	**5.35**×	**6.09**×	**5.65**×	**6.12**×	**5.66**×	**5.78**×	**5.36**×	**6.92**×	**6.58**×	**6.13**×	**5.72**×

Table 3 reports results on Llama-3 models. On Llama-3-8B-Instruct, CATS achieves average greedy speedups of 2.98× with chain decoding and 3.33× with tree decoding, compared with 2.27× for Kangaroo and 2.90× for EAGLE-self tree. We also applied our method to EAGLE-3, which is not a self-speculative decoding method. Building on its stronger draft model, CATS achieves average

speedups of $6.13\times$ under greedy decoding and $5.72\times$ under relaxed sampling on Llama-3.1-8B-Instruct, improving upon EAGLE-3 by 10.5% and 9.2%, respectively. Appendix F further provides matched-structure comparisons that isolate the contribution of our cascade design on Llama-3-8B-Instruct, together with generation-quality results on MT-bench and GSM8K.

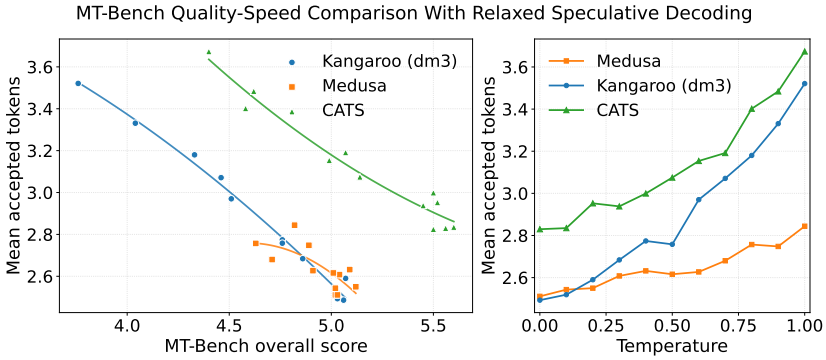


Figure 5: MT-Bench quality–speed comparison under relaxed acceptance. Quality is measured by GPT-4o following the protocol of Zheng et al. (2023); we report the mean score across all MT-Bench categories.

Quality–speed Pareto frontier. We evaluate the quality–speed trade-off under relaxed acceptance on MT-Bench (Figure 5). CATS improves accepted length while preserving generation quality under the evaluated settings. Under matched schedules (τ=0.3, α=0.09, 5 draft steps), CATS lies on the upper-right Pareto frontier on MT-Bench, improving accepted length while maintaining competitive evaluation scores. Sweeping temperature from 0 to 1 preserves the separation, suggesting that cascaded verification increases usable parallelism without a measurable reduction in generation quality under the evaluated settings.

5.2 Ablation Study

We study memory allocation and cross-device performance, shallow-verifier effectiveness, and decoding and training choices. Unless otherwise stated, ablations use Vicuna-7B on MT-Bench.

5.2.1 Memory Budget and Cross-Device Evaluation

Adaptivity across memory budgets. We vary the resident-weight budget over 2, 6, and 8 GB, choosing the deepest L_{SV} that fits each budget. CATS achieves the highest speedup at every budget (Table 4), demonstrating the benefit of adapting verifier depth to available memory.

As the budget grows, the verifier can cover more intermediate layers without changing the repeated draft path. The resulting trend is visible in both acceptance and throughput: at 8 GB, CATS reaches 3.05 accepted tokens and 0.461 tokens/s, compared with 2.80 and 0.329 tokens/s at 2 GB.

Table 4: Acceleration under different memory budgets on Vicuna-7B.

Budget	Method	Comp/tok (s)	Mean acc.	Tok/s↑	Speedup
2 GB	Baseline	0.213	1.00	0.117	1.00×
2 GB	Kangaroo (L_{DM}=3)	0.094	2.27	0.266	2.27×
2 GB	CATS (L_{DM}=3, L_{SV}=5)	**0.067**	**2.80**	**0.329**	**2.82×**
6 GB	Baseline	0.204	1.00	0.126	1.00×
6 GB	Kangaroo (L_{DM}=3)	0.089	2.27	0.267	2.12×
6 GB	CATS (L_{DM}=3, L_{SV}=10)	**0.064**	**2.94**	**0.373**	**2.96×**
8 GB	Baseline	0.253	1.00	0.146	1.00×
8 GB	Kangaroo (L_{DM}=3)	0.098	2.27	0.282	1.93×
8 GB	CATS (L_{DM}=3, L_{SV}=15)	**0.062**	**3.05**	**0.461**	**3.16×**

Deepening the drafter increases both compute and DRAM traffic across $\bar{\gamma}$ sequential passes per cycle. We quantify this cost using *Bytes Per Token* (BPT) (Alizadeh et al., 2024), the weight volume read

from DRAM per generated token. For Kangaroo,

$$\mathrm{BPT}_{\mathrm{Kangaroo}} = \frac{\bar{\gamma} \cdot B_{\mathrm{draft}} + B_{\mathrm{verify}}}{\bar{\beta} + 1}, \tag{2}$$

where B_{draft}, B_{verify} are draft and verification parameter volumes and $\bar{\gamma}$ is drafting passes per cycle. CATS's three-stage pipeline inserts a shallow verifier, splitting B_{verify} into two contiguous segments:

$$\mathrm{BPT}_{\mathrm{CATS}} = \frac{\bar{\gamma} \cdot B_{\mathrm{draft}} + B_{\mathrm{SV}} + B_{\mathrm{target}}}{\bar{\beta} + 1}, \tag{3}$$

with $B_{\mathrm{SV}} + B_{\mathrm{target}} = B_{\mathrm{verify}}$. Because B_{SV} is streamed once per cycle as part of the target forward pass, enlarging L_{SV} increases verifier capacity without inflating the $\bar{\gamma} \cdot B_{\mathrm{draft}}$ term.

Table 5: Same-budget edge ablation on NVIDIA AGX Orin (Vicuna-7B, 8 GB budget).

Method	BPT (GB/tok)↓	Comp/tok (s)	Mean acc.	Tok/s↑	Speedup
Baseline (L=32)	12.95	0.253	1.00	0.146	1.00×
Kangaroo (L_{DM}=3)	8.37	0.098	2.27	0.282	1.93×
Kangaroo (L_{DM}=5)	8.67	0.090	2.41	0.275	1.88×
Kangaroo (L_{DM}=15)	11.10	0.106	2.98	0.224	1.53×
CATS (L_{DM}=3, L_{SV}=15)	**5.84**	**0.062**	**3.05**	**0.461**	**3.16×**

Under the same 8 GB budget, deepening Kangaroo from L_{DM}=3 to 15 raises acceptance but reduces speedup from 1.93× to 1.53× because the deeper drafter runs on every sequential pass. CATS instead reaches comparable acceptance, lower BPT, and 3.16× speedup by evaluating the intermediate verifier once per cycle (Table 5).

Cross-device evaluation. On the natively memory-constrained Jetson Orin Nano 8GB, CATS improves Vicuna-7B throughput on MT-Bench from 0.11 to 0.41 tokens/s (3.70× speedup), demonstrating gains across different memory capacities and storage hardware (Appendix E).

5.2.2 Effectiveness of Shallow Verification

We assess whether the SV identifies useful corrections across draft positions. Increasing L_{SV} from 5 to 15 improves precision/recall from $0.809/0.883$ to $0.911/0.939$ relative to target acceptance (Appendix H); moreover, 52.3–61.2% of accepted corrections occur after draft position 0 (Appendix I), showing that later branches contribute substantially. Because the SV only selects candidates and the target verifies every emitted token, its errors affect efficiency rather than greedy decoding correctness.

5.2.3 Drafting Horizon and Adapter Training

Increasing $\bar{\gamma}$ from 3 to 5 raises mean accepted length from 2.7077 to 3.0520 with a small BPT increase. Longer horizons yield diminishing gains while repeating the autoregressive draft path (Appendix J); we therefore use $\bar{\gamma}$=5 to balance acceptance and drafting cost.

Adapter-training objective. The loss comparison and top-K sweep yield small hit-rate differences of 0.17 and 0.37 percentage points (Appendix K.2).

6 Conclusion and Limitations

CATS amortizes weight movement through a memory-adaptive verification cascade, improving acceptance and end-to-end speed across models and edge devices. The gains hold for chain and tree candidates and extend to EAGLE-3 with a stronger auxiliary drafter, showing that the cascade is not tied to one drafting architecture. Its limitations include evaluation on 7B–13B models, the up-front cost of adapter distillation, and reliance on transformer-style layer representations, which limits direct application to state-space models.

AI USE STATEMENT

The role of generative AI in this study was limited to language-level editing of text originally written by the authors and to minor assistance with LaTeX typesetting and document presentation. Such tools were not involved in generating or processing data, establishing the conceptual or theoretical basis of the work, developing hypotheses, formulating or checking mathematical statements and derivations, designing or assessing the methodology or experiments, implementing the method, producing or revising code, translating material, reviewing the literature, or drawing conclusions from the experimental findings. Qualitative or thematic analysis does not apply to this study.

Any passages revised with AI assistance were subsequently examined by the authors. The authors independently confirmed all reported values and scientific claims and remain fully responsible for the manuscript and for any content or artifacts prepared with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

The complete decoding procedure and Reduced KL Loss derivation are provided in Appendices A and K. Section 5 specifies the models, datasets, benchmarks, adapter-training hyperparameters, default decoding configuration, and evaluation platforms; Section 5.2 reports the memory-budget and drafting-horizon sweeps. An anonymized implementation, including the adapter-training scripts and edge inference harness, is provided as supplementary material.

REFERENCES

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S. Karen Khatamifard, Minsik Cho, Carlo C. Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. LLM in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for Medusa decoding. In *First Conference on Language Modeling*, 2024.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024.

Seongjin Cha, Gyuwan Kim, Dongsu Han, Tao Yang, and Insu Han. KnapSpec: Self-speculative decoding via adaptive layer selection as a knapsack problem. *arXiv preprint arXiv:2602.20217*, 2026.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Hao Chen, Cong Tian, Zixuan He, Bin Yu, Yepang Liu, and Jialun Cao. Inference performance evaluation for LLMs on edge devices with a novel benchmarking framework and metric. *arXiv preprint arXiv:2508.11269*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90% ChatGPT quality. `https://lmsys.org/blog/2023-03-30-vicuna/`, 2023.

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

Cunxiao Du, Jing Jiang, Yuanchen Xu, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. GliDe with a CaPE: A low-hassle method to accelerate speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Gabe Guo and Stefano Ermon. Self-speculative decoding in any-order and any-subset autoregressive models. In *Structured Probabilistic Inference and Generative Modeling Workshop at NeurIPS 2025*, 2025.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL 2024. Association for Computational Linguistics, 2024.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. CLLMs: Consistency large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*. PMLR, 2023.

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Wen-tau Yih, and Xi Victoria Lin. Nearest neighbor speculative decoding for LLM generation and attribution. In *Advances in Neural Information Processing Systems*, volume 37, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432. Association for Computational Linguistics, 2024c.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *Advances in Neural Information Processing Systems*, volume 37, 2024a.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024b.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24. ACM, 2024. doi: 10.1145/3620666.3651335.

Giovanni Monea, Armand Joulin, and Edouard Grave. PaSS: Parallel speculative sampling. In *Efficient Natural Language and Speech Processing Workshop at NeurIPS 2023*, 2023.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Zongyue Qin, Ziniu Hu, Zifan He, Neha Prakriya, Jason Cong, and Yizhou Sun. Optimized multi-token joint decoding with auxiliary model for LLM inference. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zebin Ren, Krijn Doekemeijer, Tiziano De Matteis, Christian Pinto, Radu Stoica, and Animesh Trivedi. An I/O characterizing study of offloading LLM models and KV caches to NVMe SSD. In *Proceedings of the 5th Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*, CHEOPS '25. ACM, 2025. doi: 10.1145/3719330.3721230.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. PowerInfer: Fast large language model serving with a consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. ACM, 2024.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Prabhu Vellaisamy, Shreesh Tripathi, Vignesh Natarajan, Surya Santhan Thenarasu, Shawn Blanton, and John P. Shen. TaxBreak: Unmasking the hidden costs of LLM inference through overhead decomposition. *arXiv preprint arXiv:2603.12465*, 2026.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. OPT-Tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating Seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 2023.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 2024.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yunfan Xiong, Ruoyu Zhang, Yanzeng Li, Tianhao Wu, and Lei Zou. DySpec: Faster speculative decoding with dynamic token tree structure. *arXiv preprint arXiv:2410.11744*, 2024.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*, 2024.

APPENDIX

A CATS ALGORITHM

Algorithm 1 CATS: Cascaded Self-Speculative Decoding with Tree-Masked Final Verification

Require: Prompt $\mathbf{x}$; target model $\mathcal{M}$ with layers $1{:}L_{\text{total}}$ and LM head $\mathcal{H}$; draft cut-point L_{DM}, shallow-verifier cut-point L_{SV} ($1 \le L_{\text{DM}} < L_{\text{SV}} < L_{\text{total}}$); draft adapter $\mathcal{A}_{\text{DM}}$, SV adapter $\mathcal{A}_{\text{SV}}$; drafting horizon $\bar{\gamma}$; max new tokens T; acceptance criterion ACC (greedy or typical (Cai et al., 2024))
Ensure: Generated sequence $\mathbf{y}$

// ===== Initialization =====
Load layers $1{:}L_{\text{SV}}$ (DM $\cup$ SV) into DRAM and pin resident
$(\mathbf{h}^{(1:L_{\text{total}})},\ \text{KV}_{\mathcal{M}}) \leftarrow \mathcal{M}(\mathbf{x})$ ▷ prompt prefill, full target
$y_0 \leftarrow \arg\max \mathcal{H}(\mathbf{h}_{-1}^{(L_{\text{total}})})$; initialize $\text{KV}_{\mathcal{A}_{\text{DM}}}$, $\text{KV}_{\mathcal{A}_{\text{SV}}}$ from $\mathbf{h}^{(L_{\text{DM}})}$, $\mathbf{h}^{(L_{\text{SV}})}$
$s \leftarrow |\mathbf{x}|$ ▷ committed length

while $s < T$ **do**
  // ===== Stage 1: Draft =====
  $\mathbf{d} \leftarrow [\,]$; $\mathbf{H}_{\text{DM}} \leftarrow [\,]$
  for $k = 0, \ldots, \bar{\gamma}-1$ **do**
    $h \leftarrow \mathcal{M}_{1:L_{\text{DM}}}(y_{s+k},\ \text{KV}_{\mathcal{M}})$; append h to $\mathbf{H}_{\text{DM}}$
    $(o,\ \text{KV}_{\mathcal{A}_{\text{DM}}}) \leftarrow \mathcal{A}_{\text{DM}}(h,\ \text{KV}_{\mathcal{A}_{\text{DM}}})$
    $y_{s+k+1} \leftarrow \arg\max \mathcal{H}(o)$; append to $\mathbf{d}$
  end for

  // ===== Stage 2: Shallow Verification =====
  $\mathbf{H}_{\text{SV}} \leftarrow \mathcal{M}_{L_{\text{DM}}+1:L_{\text{SV}}}(\mathbf{H}_{\text{DM}},\ \text{KV}_{\mathcal{M}})$ ▷ $\bar{\gamma}$ positions in parallel
  $(\mathbf{O}_{\text{SV}},\ \text{KV}_{\mathcal{A}_{\text{SV}}}) \leftarrow \mathcal{A}_{\text{SV}}(\mathbf{H}_{\text{SV}},\ \text{KV}_{\mathcal{A}_{\text{SV}}})$; $\hat{\mathbf{c}} \leftarrow \arg\max \mathcal{H}(\mathbf{O}_{\text{SV}})$
  $\mathcal{R} \leftarrow \{\, (i,\ \hat{c}_i)\ :\ \hat{c}_i \neq d_i,\ 0 \le i < \bar{\gamma} \,\}$ ▷ positions where SV proposes a correction

  if $\mathcal{R} = \emptyset$ **then**
    $\mathcal{T} \leftarrow \text{CHAINTREE}(\mathbf{d})$ ▷ straight chain, no correction branches
  else
    $\mathcal{T} \leftarrow \text{BUILDTREE}(\text{main} = \mathbf{d},\ \text{corrections} = \mathcal{R})$
      ▷ main branch is unchanged; each $(i, \hat{c}_i) \in \mathcal{R}$ adds a side branch at position i
    Re-forward correction tokens through layers $1{:}L_{\text{SV}}$ under tree-masked attention
      ▷ reuses DRAM-resident DM and SV layers; **zero** additional flash transfer
  end if

  // ===== Stage 3: Final Verification (stream layers $L_{\text{SV}}{+}1{:}L_{\text{total}}$ from flash) =====
  $\mathbf{H}_{\text{tgt}} \leftarrow \mathcal{M}_{L_{\text{SV}}+1:L_{\text{total}}}(\mathcal{T}.\text{hidden_states},\ \text{mask}{=}\mathcal{T}.\text{tree_mask},\ \text{KV}_{\mathcal{M}})$
  $\mathbf{p}_{\text{tgt}} \leftarrow \text{softmax}(\mathcal{H}(\mathbf{H}_{\text{tgt}}))$

  // —— Walk the tree to find the longest accepted prefix ——
  $a \leftarrow 0$; $\mathbf{y}_{\text{out}} \leftarrow [\,]$
  for $i = 0, \ldots, \bar{\gamma}-1$ **do**
    if $\text{ACC}(d_i,\ \mathbf{p}_{\text{tgt}}^{(i)})$ **then**
      $a \leftarrow a + 1$; append d_i to $\mathbf{y}_{\text{out}}$
    else if $(i,\ \hat{c}_i) \in \mathcal{R}$ **and** $\text{ACC}(\hat{c}_i,\ \mathbf{p}_{\text{tgt}}^{(\text{corr},\,i)})$ **then**
      $a \leftarrow a + 1$; append $\hat{c}_i$ to $\mathbf{y}_{\text{out}}$; **break**
        ▷ correction branch terminates the cycle by construction
    else
      break
    end if
  end for
  Append $\arg\max \mathbf{p}_{\text{tgt}}^{(a)}$ to $\mathbf{y}_{\text{out}}$ ▷ bonus token from target's own next-token prediction

  Commit $\mathbf{y}_{\text{out}}$ to $\mathbf{y}$; $s \leftarrow s + |\mathbf{y}_{\text{out}}|$
  Roll back $\text{KV}_{\mathcal{M}}$, $\text{KV}_{\mathcal{A}_{\text{DM}}}$, $\text{KV}_{\mathcal{A}_{\text{SV}}}$ to length s along the accepted path
end while
return $\mathbf{y}_{0:s}$

B NOTATION USED IN FIGURE 4

Table 6 defines the symbols appearing in Figure 4. All subscripts index position within a single drafting round.

Table 6: Symbols used in Figure 4.

Symbol	Meaning
x_0	the last committed token, which opens the drafting round
$x_1 \dots x_4$	the draft tokens produced by the draft model
h_i	the hidden state at position i at the draft exit layer; h_i yields x_{i+1}
s_i	the same, at the shallow-verifier layer L_{SV}
t_i	the same, at the final layer (target verification)
$v_1 \dots v_4$	the shallow verifier's tokens: v_i is decoded from s_{i-1} and compared with x_i; a mismatch makes v_i a correction branch
$n_1 \dots n_5$	the target's tokens: n_i is decoded from t_{i-1} and decides whether x_i is accepted; the last one is the bonus token appended after the accepted prefix
primed symbols	the same quantities on a correction branch

One round therefore runs $x \to h \to s \to t$ down the layer stack: the shallow verifier reads s, emits the corrections v, and the primed states carry each v through the remaining layers so that it is verified alongside the trunk in the same batched final pass.

In the verification tree, the trunk $x_0 \dots x_4$ is a single chain, and each correction v_i hangs off the trunk at its own position — a child of x_{i-1} and a *sibling* alternative to x_i — rather than forming an autoregressive chain of its own. The tree mask preserves causality: each v_i attends to the trunk prefix $x_0 \dots x_{i-1}$ and to itself, but not to the other corrections. Because the corrections are per-position alternatives to the trunk rather than a continuation of it, accepting any one of them terminates the cycle by construction.

Figure 6 exercises this notation on a single complete decoding cycle.

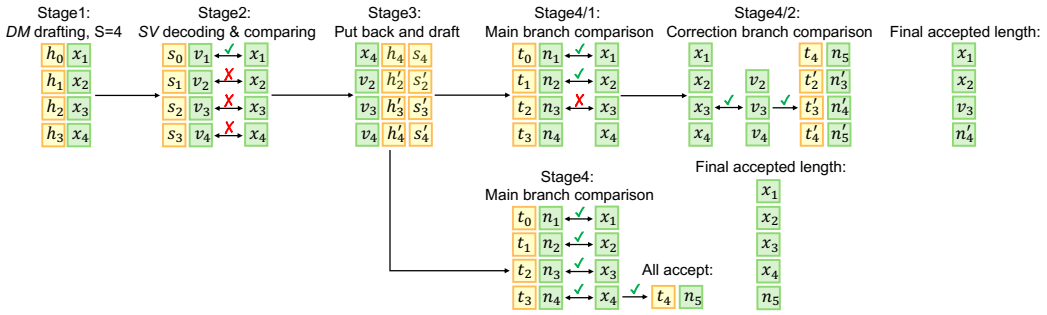


Figure 6: One complete decoding cycle of CATS, worked through end to end in the notation of Table 6. *Stage 1* drafts tokens autoregressively through the draft sub-network (DM): each hidden state h_i yields the draft token x_{i+1}. *Stage 2* forwards those states through the shallow verifier (SV) to the states s_i, decodes an SV token v_{i+1} at each position and compares it against the draft token x_{i+1}. *Stage 3* puts the mismatching v_i — the corrections — together with the last draft token x_4 back through layers 1 to L_{SV} under the tree mask to obtain their primed hidden states. *Stage 4* verifies the main branch and the correction branches against the target model in the same batched pass. Verification walks the main branch to its first mismatch and then checks the correction proposed at that position: if it matches the target, the correction is accepted in place of the mismatched draft token and one further token is decoded from its hidden state, which fixes the accepted length. If every token on the main branch is accepted, verification ends there with the full draft accepted.

C EXPERIMENTAL SETUP AND MEASUREMENT METHODOLOGY

Device. All edge measurements are taken on a Jetson AGX Orin 64GB DevKit running JetPack 5.1.2, with 65.85 GB of LPDDR5 *unified* CPU/GPU memory and 64 GB of eMMC 5.1 storage (no NVMe), under PyTorch 2.0.0+nv23.05 / CUDA 11.4. For Vicuna-7B in fp16 we measure

target weights of 13.544 GB and a KV cache of 512 KiB per token, hard-capped at 1.074 GB by the 2048-token context.

Emulating memory budgets on one platform. We use the AGX Orin as a single edge platform on which to emulate a range of memory budgets. On a memory-constrained device the DRAM pool is shared between CPU and GPU, and the operating system, the runtime, the activations and the KV cache all draw on the same budget as the weights, so only part of the nominal capacity can hold a model. We therefore set a resident-weight budget in code: only that many layers stay in DRAM, and every layer outside the budget is fetched from flash for each target verification. Varying the budget on one device isolates its effect while compute, storage and software stay fixed, and the flash→DRAM→GPU transfers are physically executed, not simulated. We further evaluate CATS on a device whose memory is natively constrained, the Jetson Orin Nano 8GB (Appendix E).

D STORAGE BANDWIDTH, LATENCY, AND KV CACHE ANALYSIS

D.1 STORAGE BANDWIDTH AND LATENCY DECOMPOSITION

Layer weights are read *directly from eMMC with* `O_DIRECT`, bypassing the page cache and reconstructing tensors bit-identically to `torch.load`. Table 7 reports the measured paths.

Table 7: Measured transfer bandwidths on the Jetson AGX Orin.

Path	**Measured**
flash → DRAM (`O_DIRECT`)	0.270 / 0.275 / 0.284 GB/s (three independent runs)
same, buffered (page-cache hit)	3.1 – 4.5 GB/s
DRAM → GPU (pinned H2D)	32.8 – 35.1 GB/s

The buffered row shows why `O_DIRECT` is necessary: a naive read reports ∼4 GB/s of *DRAM page cache* rather than flash. The gap between the first and third rows is roughly 120×, which is why streaming rather than compute sets the pace.

Per 404.8 MB layer the three stages measure **flash** 1.47 **s**, **H2D** 11.5 **ms** and **compute** 5.0 **ms**, so flash accounts for 98.9% of the weight-movement chain. The mechanism is therefore measured rather than inferred from the Bytes-Per-Token analysis: per-token latency is set by how many times the layers must be streamed, and the compute added for drafting and verification is negligible beside it. Because flash dominates,

$$T_{\text{token}} \;\approx\; \frac{(L-R)\,B_{\text{layer}}}{\text{BW}_{\text{flash}}\cdot\tau}, \tag{4}$$

where L is the total layer count, R the number of resident layers and B_{layer} the per-layer volume. The memory budget therefore sets absolute throughput while the accepted length sets the speedup. Lowering the budget raises the streamed volume accordingly — to 7.29 GB at the 6 GB budget and 11.33 GB at the 2 GB budget — and latencies scale with it.

D.2 KV CACHE GROWTH AND MEMORY BUDGET

On edge devices, KV-cache growth is a second-order cost next to weight streaming. Table 8 reports measured KV growth for Vicuna-7B; analytic and measured values agree at all four lengths.

Table 8: KV cache size versus context length (Vicuna-7B, 512 KiB per token).

Context length	128	512	1024	2048
KV cache	0.067 GB	0.268 GB	0.537 GB	**1.074** GB

The KV cache is bounded at 1.074 GB by the 2048-token context — 8.3% of the 12.95 GB of decoder weights. Because it shares the resident budget with the weights, a full context displaces only a few resident layers: at the 8 GB budget the streamed layer count rises from 13 to 15 between empty and full context. Weight streaming therefore remains the dominant per-token cost across the whole context range.

The two offloads are also not symmetric. Weights are fixed-size, read-only, and streamed once per verification, amortised over τ accepted tokens, whereas the KV cache grows with context length and is *written every step* — costly on eMMC in both bandwidth and write endurance, where weight offload is a pure streaming read. At our operating point the KV cache is bounded and small, so it is not the bottleneck; should KV pressure grow, it can be streamed alongside the model weights under the same memory-adaptive scheduling principle.

E A Natively Memory-Constrained Device: Jetson Orin Nano

We also measured on a Jetson Orin Nano 8GB, where the 13.54 GB target exceeds the device's memory outright and weights are read from flash on every forward pass. Its M.2 NVMe is roughly ten times faster than the AGX's eMMC, so the two devices bracket the storage range rather than repeating one operating point.

Table 9: Jetson Orin Nano 8GB, Vicuna-7B, MT-bench.

Metric	**Measured**
Transfer / compute share of per-token latency	95.8% / 4.1%
Baseline throughput	0.11 tok/s
CATS	$\mathbf{0.41}$ tok/s
Speedup	$\mathbf{3.70}\times$

On the Nano every target pass streams the model from flash, so absolute throughput is set by storage bandwidth (Eq. 4) and transfer accounts for 95.8% of per-token latency. CATS multiplies that storage-bound rate by its accepted length: throughput rises from 0.11 to 0.41 tokens/s over autoregressive decoding of the same fp16 target, a $3.70\times$ speedup. With the AGX results, the two devices show that the gain holds across a tenfold range of storage bandwidth and on both an emulated and a native memory constraint.

F Matched-Structure Gains and Generation Quality

We measure the cascade's contribution by comparing methods with the same candidate structure. The shallow verifier operates on both chains and trees, allowing matched-structure comparisons that separate its contribution from the gains of speculative decoding itself. Table 10 reports the gain in end-to-end speedup over the *strongest baseline of the same structure*, computed from the five-benchmark means in Tables 1, 2 and 3 — for chain rows whichever of Kangaroo / Medusa / EAGLE-self is fastest for that target, for tree rows EAGLE-self tree. For the auxiliary-drafter setting, we also report the gain of CATS w/ EAGLE-3 over native EAGLE-3 on Llama-3.1-8B-Instruct.

Table 10: Relative end-to-end speedup gains over matched baselines, computed from five-benchmark means. The upper block compares self-speculative methods of the same candidate structure; the lower block compares against native EAGLE-3 with an auxiliary drafter. Bold marks the largest gain in each column; dashes indicate inapplicable chain comparisons.

Target	**chain, greedy**	**chain,** $T{=}0.7$	**tree, greedy**	**tree,** $T{=}0.7$
Vicuna-7B	$+27.7\%$	$+\mathbf{47.0}\%$	$+\mathbf{58.5}\%$	$+\mathbf{60.0}\%$
Vicuna-13B	$+14.9\%$	$+12.6\%$	$+20.3\%$	$+21.5\%$
LLaMA2-7B	$+29.2\%$	$+22.1\%$	$+26.4\%$	$+22.2\%$
LLaMA2-13B	$+9.4\%$	$+8.3\%$	$+9.2\%$	$+5.8\%$
Llama-3-8B	$+\mathbf{31.3}\%$	$+31.0\%$	$+14.8\%$	$+15.6\%$
End-to-end speedup gain over native EAGLE-3				
Llama-3.1-8B	—	—	$+10.5\%$	$+9.2\%$

All twenty self-speculative comparisons are positive across five targets from three model families, two scales and both decoding regimes: chain gains span 8.3–47.0% and tree gains 5.8–60.0%. The tree comparisons hold branching fixed on both sides, directly measuring the additional speedup from the cascade. With EAGLE-3's auxiliary drafter, the cascade also improves mean end-to-end speedup

by 10.5% under greedy decoding and 9.2% under relaxed acceptance, extending the gains beyond self-speculative decoding.

Generation quality. On Llama-3-8B-Instruct, CATS w/ EAGLE matches the generation quality of the same base model within evaluation noise (Table 11).

Table 11: Generation quality on Llama-3-8B-Instruct. MT-bench uses a GPT-4o judge ($160/160$ judgments per model); GSM8K is exact match over 300 items and involves no judge.

	MT-bench (GPT-4o judge)	**GSM8K (exact match)**
Llama-3-8B-Instruct	7.281	78.33%
CATS w/ EAGLE	7.338	79.33%

G DEPLOYMENT ALTERNATIVES UNDER A MEMORY BUDGET

Under a memory budget, a deployment can either shrink the model until it fits — a smaller model or a quantized one — or keep the target model and accelerate its decoding, which is what CATS does. Table 12 measures the footprint and generation quality of both routes on Llama-3-8B-Instruct.

Table 12: Alternatives under an 8 GB resident budget: replace the target with a model that fits, or keep the fp16 target and accelerate its decoding with CATS. Weights are the stored fp16 model size unless noted; the 8B target streams the layers beyond the budget from flash on each target pass, with or without CATS, while the smaller models fit entirely. MT-bench uses a GPT-4o judge ($160/160$ judgments per model); GSM8K is exact match over 300 items without an LLM judge.

Model	**Weights (stored)**	**MT-bench**	**GSM8K**
Llama-3-8B fp16	16.06 GB	7.281	78.33%
+ CATS w/ EAGLE	16.06 GB (+ adapters)	7.338	79.33%
Llama-3.2-3B fp16	6.42 GB	7.163	75.33%
Llama-3-8B NF4 (4-bit)	5.59 GB	6.856	76.00%
Qwen3-1.7B fp16	3.44 GB	6.781	76.67%

Shrinking the model costs quality. Llama-3.2-3B fits in 6.42 GB but scores 75.33% against 78.33% on GSM8K and 7.163 against 7.281 on MT-bench; Qwen3-1.7B fits in 3.44 GB and scores 6.781 and 76.67%. NF4 quantization brings the 8B model itself to 5.59 GB, including quantization constants and the unquantized embedding and LM head, but lowers its MT-bench score by 0.42 points and its GSM8K accuracy by 2.33 percentage points, and its batch-1 decoding is 36% slower than fp16. A smaller footprint therefore buys neither the target's quality nor, for quantization, faster decoding.

CATS takes the other route. It keeps the fp16 target and its quality (7.338 against 7.281 on MT-bench and 79.33% against 78.33% on GSM8K, matching the same base model within evaluation noise) and instead reduces how often the layers beyond the budget are re-read from flash, decoding $3.33\times$ faster than the autoregressive fp16 target under the same 8 GB budget (Table 3). It is the only option in Table 12 that leaves the target's quality intact.

How the alternatives can be combined. Per-token weight traffic depends on the storage used per weight, the number of weights fetched per forward pass, and the number of target-model passes required per generated token. Quantization reduces storage per weight, while selective loading and activation sparsity (Alizadeh et al., 2024; Song et al., 2024) reduce the weights fetched. CATS reduces target-model calls per generated token by increasing the accepted length. These mechanisms act on different components of the cost and can be combined, although their end-to-end benefits need not multiply exactly. In particular, a quantized model that still exceeds the resident budget must still stream non-resident layers; CATS can reduce how often those layers are reread. We therefore isolate the decoding contribution by comparing each target model with and without CATS under the same model representation.

H SHALLOW-VERIFIER ACCURACY

Treating the shallow verifier as a binary classifier — positive = "SV passes this draft token", ground truth = "the target accepts it" — lets us characterise it independently of end-to-end speedup. Table 13 sweeps the verifier depth on Vicuna-7B with the drafter at layer 3, MT-bench, greedy.

Table 13: Shallow verifier treated as a binary classifier over draft tokens.

L_{SV}	TP	FP	FN	TN	Precision	Recall	F1	Accuracy
5	23,384	5,521	3,087	8,786	0.8090	0.8834	0.8446	0.7889
10	22,135	3,443	2,236	9,946	0.8654	0.9083	0.8863	0.8496
15	21,570	2,108	1,395	10,206	**0.9110**	**0.9393**	**0.9249**	**0.9007**

A false positive is the SV passing a token the target then rejects; a false negative is the SV correcting one the target would have accepted. All four metrics improve monotonically with depth and both error types fall together, so the benefit is structural rather than tuned to one operating point. Recall exceeds precision at every depth, meaning the SV errs toward permissiveness rather than over-correction — the preferable bias, since a false correction consumes tree budget while a miss only forfeits an opportunity.

Crucially, *neither error type can affect correctness*. Every emitted token is still verified by the target's own prefix match, so the SV only decides which candidates are worth proposing: it is a prioritiser, not a correctness gate. Even the shallowest verifier (L_{SV}=5) is exactly as lossless as the deepest one.

I WHERE CORRECTIONS ARE ACCEPTED

The cascade recovers mismatches throughout the drafting round, not only at its first position. The corrections v_i are per-position siblings conditioned on the trunk (Appendix B), so the SV can repair a mismatch at any position in the round. Table 14 logs which correction the target accepts across 102,027 mismatch steps (Vicuna-7B, drafter layer 3, L_{SV}=15, greedy).

Table 14: Distribution of accepted corrections by position within the drafting round.

Benchmark	pos 0	1	2	3	4	beyond pos 0
MT-bench	43.8%	24.7%	15.4%	10.0%	6.1%	**56.2%**
Spec-bench	46.6%	24.5%	14.3%	8.9%	5.6%	**53.4%**
GSM8K	44.0%	25.9%	14.9%	9.2%	6.1%	**56.0%**
HumanEval	38.8%	25.8%	16.1%	11.5%	7.8%	**61.2%**
Alpaca	47.7%	24.8%	13.6%	8.4%	5.4%	**52.3%**

More than half of all accepted corrections occur after draft position 0, and this holds on every benchmark individually (52.3% to 61.2%). The distribution is front-loaded, as the sibling structure implies, but the majority of the benefit comes from mismatches the drafter makes several tokens in, which a correction limited to the first draft position would miss. HumanEval is the flattest, consistent with code drafting diverging later.

J DRAFTING-HORIZON ABLATION

Table 15 reports the full drafting-horizon sweep summarised in Section 5.2.3 (Vicuna-7B, MT-bench, greedy decoding, chain candidates). The gain from the cascade does not depend on the horizon: CATS is 1.61–1.67× faster than Kangaroo and EAGLE-self at every $\bar{\gamma}$, and its BPT at $\bar{\gamma}$=10 (7.16) remains below that of both baselines at $\bar{\gamma}$=3 (11.15 / 11.17).

The horizon itself is set by the cost curve. Moving from $\bar{\gamma}$=3 to 5 raises τ from 2.7077 to 3.0520 and speedup from 2.82× to 3.16× for a 2.7% increase in BPT; moving from 5 to 10 adds a further 8.5% speedup for 22.8% more BPT. BPT counts the DRAM weight traffic of the repeated draft passes; on the Orin, per-token latency is set by flash streaming (Appendix D), so this traffic costs little wall-clock time and speedup keeps rising with the horizon. We set the horizon on the cost curve

rather than on the last increment of speedup: beyond 5, each additional accepted token is bought with disproportionately more drafting traffic. $\bar{\gamma}$=5 is also the horizon at which every chain baseline is run in Tables 1 and 2, so the main-text comparisons are horizon-matched.

Table 15: Drafting-horizon ablation on Vicuna-7B (MT-bench, greedy decoding, chain candidates), reporting mean accepted length, BPT, and end-to-end speedup.

	Kangaroo			EAGLE-self			CATS		
$\bar{\gamma}$	**Mean acc.**	**BPT↓**	**Speedup**	**Mean acc.**	**BPT↓**	**Speedup**	**Mean acc.**	**BPT↓**	**Speedup**
3	2.2501	11.1531	1.75×	2.2475	11.1660	1.73×	**2.7077**	**5.6805**	**2.82×**
5	2.4863	13.3496	1.92×	2.5151	13.1967	1.90×	**3.0520**	**5.8354**	**3.16×**
7	2.6330	15.6804	2.04×	2.6194	15.7618	2.00×	**3.2343**	**6.2574**	**3.34×**
10	2.7142	19.6852	2.09×	2.6984	19.8004	2.05×	**3.3332**	**7.1646**	**3.43×**

K REDUCED KL LOSS: DERIVATION AND ABLATION

K.1 DERIVATION

In this appendix we expand the construction of the **Reduced KL Loss** introduced in Section 4, Equation 1. Recall the setup: we train the draft and shallow-verifier adapters by aligning their output distribution $q_{\text{draft}}(v \mid t)$ with the target-model distribution $p_{\text{target}}(v \mid t)$ across a set M of unmasked token positions, where $v \in \{1, \ldots, V\}$ indexes vocabulary tokens and t indexes positions. The standard full-vocabulary distillation loss,

$$\mathcal{L}_{\text{full}} = -\frac{1}{|M|} \sum_{t \in M} \sum_{v=1}^{V} p_{\text{target}}(v \mid t) \cdot \log q_{\text{draft}}(v \mid t), \tag{5}$$

spreads supervision over all V tokens, but at any position t the vast majority of the V tokens form a long tail of near-zero probabilities that rarely determines acceptance, while almost all of the probability mass sits on a handful of head tokens. To concentrate the limited capacity of our compact adapters on the tokens that actually matter, we restrict the loss to the top-K tokens of $p_{\text{target}}(\cdot \mid t)$ at each position. The construction proceeds in three steps.

Step 1: Select the top-K indices. At each unmasked position $t \in M$, identify the K vocabulary tokens carrying the highest probability mass under the target distribution:

$$\mathcal{T}_K(t) = \underset{v \in \{1,\ldots,V\}}{\arg \text{top-K}} \; p_{\text{target}}(v \mid t). \tag{6}$$

$\mathcal{T}_K(t)$ is a position-dependent support of size $|\mathcal{T}_K(t)| = K$ that captures the tokens with non-negligible probability of being emitted by the target model. In speculative decoding, only tokens within this set can plausibly be accepted, so any supervision spent outside $\mathcal{T}_K(t)$ is effectively wasted.

Step 2: Renormalize over the focused support. The masses $\{p_{\text{target}}(v \mid t)\}_{v \in \mathcal{T}_K(t)}$ in general do not sum to 1. We renormalize them onto $\mathcal{T}_K(t)$ to obtain a valid probability distribution $\tilde{p}_{\text{target}}(v \mid t)$:

$$\tilde{p}_{\text{target}}(v \mid t) = \begin{cases} \dfrac{p_{\text{target}}(v \mid t)}{\displaystyle\sum_{v' \in \mathcal{T}_K(t)} p_{\text{target}}(v' \mid t)}, & v \in \mathcal{T}_K(t), \\ 0, & v \notin \mathcal{T}_K(t). \end{cases} \tag{7}$$

Equivalently, with an indicator function $\mathbf{1}[\cdot]$:

$$\tilde{p}_{\text{target}}(v \mid t) = \frac{p_{\text{target}}(v \mid t) \cdot \mathbf{1}[v \in \mathcal{T}_K(t)]}{\displaystyle\sum_{v' \in \mathcal{T}_K(t)} p_{\text{target}}(v' \mid t)}. \tag{8}$$

By construction $\sum_{v=1}^{V} \tilde{p}_{\text{target}}(v \mid t) = 1$, so $\tilde{p}_{\text{target}}(\cdot \mid t)$ is a proper distribution supported on $\mathcal{T}_K(t)$ and zero elsewhere.

Step 3: Cross-entropy on the focused support. The Reduced KL Loss is the position-averaged cross-entropy between $\tilde{p}_{\text{target}}(\cdot \mid t)$ and the adapter distribution $q_{\text{draft}}(\cdot \mid t)$, evaluated only on $\mathcal{T}_K(t)$:

$$\mathcal{L}_{\text{top-}K} = -\frac{1}{|M|} \sum_{t \in M} \sum_{v \in \mathcal{T}_K(t)} \tilde{p}_{\text{target}}(v \mid t) \cdot \log q_{\text{draft}}(v \mid t). \tag{9}$$

Because $\tilde{p}_{\text{target}}(v \mid t) = 0$ for $v \notin \mathcal{T}_K(t)$, the inner sum is mathematically equivalent to summing over the entire vocabulary; in practice we materialize only the K active terms per position to avoid the $V-K$ vanishing contributions. Up to a constant entropy term $H(\tilde{p}_{\text{target}}(\cdot \mid t))$ that does not depend on the adapter parameters, $\mathcal{L}_{\text{top-}K}$ coincides with the position-averaged KL divergence

$$\frac{1}{|M|} \sum_{t \in M} \mathrm{KL}(\tilde{p}_{\text{target}}(\cdot \mid t) \parallel q_{\text{draft}}(\cdot \mid t)),$$

which is why we call it the *Reduced* KL Loss: it is a KL divergence taken over a reduced support $\mathcal{T}_K(t)$ rather than the full vocabulary V.

Effect on adapter training. Compared with $\mathcal{L}_{\text{full}}$, restricting supervision to $\mathcal{T}_K(t)$ has two effects. First, it concentrates gradient signal on the candidates that drive acceptance, since tokens outside $\mathcal{T}_K(t)$ have p_{target} values too small to be accepted regardless of how q_{draft} allocates mass to them. Second, it spares the capacity of our compact draft and shallow-verifier adapters: the long tail of near-zero target probabilities no longer consumes it. This suits our memory-limited setting, where the draft and SV sub-networks must remain small enough to fit in the DRAM budget while still producing high-acceptance candidates.

K.2 Ablation

The gains of CATS are driven by the verification cascade; the Reduced KL Loss is a complementary training refinement that adds a further improvement on top of it. Table 16 isolates the training objective on Llama-3-8B, holding the learning rate, SV and tree fixed. Hit rate is the fraction of drafted tokens the target accepts, scored on the on-policy prefix.

Table 16: Draft-adapter training objective on Llama-3-8B (MT-bench, greedy decoding, CATS w/ EAGLE with a 40-node tree), with identical learning rate ($1e{-}5$), SV and tree (single seed).

Draft training loss	**Hit rate**	τ
Full-vocabulary KL	57.60%	3.227
Reduced KL, $K{=}20$ (ours)	**57.77%**	**3.237**

Reduced KL improves both hit rate ($+0.17$ percentage points) and τ ($3.227 \rightarrow 3.237$) over full-vocabulary distillation while supervising only K of the target's vocabulary probabilities at each position. On Vicuna-7B with identical SV and tree, raising K from 10 to 20 further lifts hit rate from 65.73% to 66.10% and τ from 3.710 to 3.732. These gains are consistent in direction and modest in size, as expected of a training refinement; the accepted-length gains of CATS are attributable to the cascade (Appendix F).